COBE: A CONJUNCTIVE ONTOLOGY BROWSER AND EXPLORER FOR
VISUALIZING SNOMED CT FRAGMENTS

by

MENGMENG SUN

Submitted in partial fulfillment of the requirements
For the degree of Master of Science

Thesis Advisor: Dr. Guo-Qiang Zhang

Department of Electrical Engineering and Computer Science

CASE WESTERN RESERVE UNIVERSITY

August, 2015
CASE WESTERN RESERVE UNIVERSITY
SCHOOL OF GRADUATE STUDIES

We hereby approve the thesis of

MENGMENG SUN

candidate for the Master of Science degree*.

(signed) Dr. Guo-Qiang Zhang, Ph.D.

(chair of the committee)

Dr. Licong Cui, Ph.D.

Dr. Rong Xu, Ph.D.

(date) July 1, 2015

*We also certify that written approval has been obtained for any proprietary material contained therein.
# TABLE OF CONTENTS

<table>
<thead>
<tr>
<th>LIST OF TABLES</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>v</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>LIST OF FIGURES</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>vi</td>
</tr>
</tbody>
</table>

1 Introduction ............................................. 1

2 Background ................................................ 4

2.1 SNOMED CT ............................................. 4
2.2 SNOMED CT Quality Assurance ......................... 5
2.3 SNOMED CT Browsers .................................... 6
2.4 Conjunctive Navigational Exploration ................ 7

3 Overview of COBE Search Interface ....................... 9

3.1 COBE Functional Architecture .......................... 9
3.2 Navigational Exploration .............................. 10
3.3 Direct Lookup ........................................ 11

4 Data Processing ............................................. 13

4.1 Hadoop Map-Reduce Framework .......................... 13
4.2 Active Concepts, Descriptions and Relationships Extraction .. 14
4.3 Computing SNOMED CT Non-lattice Fragments ............. 16
4.4 Navigational Search Stems Computation ................. 19

5 COBE Interface Implementation ............................. 22

5.1 Development Environment .............................. 22
5.2 Navigational Search Stems Menu Implementation ......... 23
5.2.1 Model Implementation ................................ 23
5.2.2 Controller Implementation ........................... 24
5.2.3 View Implementation .................................. 25
5.3 Concept Search Implementation ........................ 26
5.3.1 Model Implementation ................................ 26
5.3.2 Controller Implementation ........................................ 27
5.3.3 View Implementation ............................................ 27
5.4 Non-lattice Fragment Visualization Implementation ............. 30
  5.4.1 Model Implementation ........................................... 30
  5.4.2 Controller Implementation ....................................... 31
  5.4.3 View Implementation ............................................ 32
6  Result ................................................................. 33
  6.1 Basic Statistics ...................................................... 33
  6.2 Conjunctive Ontology Browser to Explore SNOMED CT Fragments (COBE) 34
  6.3 Visualizing Non-lattice Fragments ................................. 35
7  Evaluation ............................................................. 39
  7.1 Evaluation Method .................................................. 39
  7.2 Navigational Exploration Evaluation Result ......................... 41
  7.3 Direct Lookup Evaluation Result ................................... 41
8  Discussions ............................................................ 44
9  Conclusion ............................................................. 46
APPENDIX  Stop Words List .............................................. 47
APPENDIX  Ruby on Rails Code .......................................... 49
LIST OF REFERENCES .................................................... 54
LIST OF TABLES

<table>
<thead>
<tr>
<th>Table</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>5.1</td>
<td>The MySQL table and fields corresponding to model Stem.</td>
<td>24</td>
</tr>
<tr>
<td>5.2</td>
<td>The MySQL table and fields corresponding to model Concept.</td>
<td>26</td>
</tr>
<tr>
<td>5.3</td>
<td>The MySQL table and fields corresponding to model Fragment.</td>
<td>31</td>
</tr>
<tr>
<td>6.1</td>
<td>Summary of 10 sub-hierarchies containing the most fragments.</td>
<td>34</td>
</tr>
<tr>
<td>6.2</td>
<td>Summary of the most frequent concepts in upper bounds of non-lattice fragments of SNOMED CT.</td>
<td>35</td>
</tr>
<tr>
<td>6.3</td>
<td>Top-10 most frequent navigation search stems and the frequency.</td>
<td>36</td>
</tr>
<tr>
<td>7.1</td>
<td>Comparison of searching results of 3 SNOMED CT Browser.</td>
<td>42</td>
</tr>
<tr>
<td>7.2</td>
<td>The example-based precision, recall, and $F_1$ measure for the results of concept search using COBE, IHTSDO and NLM browser.</td>
<td>43</td>
</tr>
</tbody>
</table>

Appendix

Table
# LIST OF FIGURES

<table>
<thead>
<tr>
<th>Figure</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>2.1</td>
<td>A non-lattice fragment in SNOMED CT [6].</td>
</tr>
<tr>
<td>3.1</td>
<td>Overview of the Conjunctive Ontology Browser and Explorer COBE. NE: Navigational Exploration, DL: Direct lookup.</td>
</tr>
<tr>
<td>4.1</td>
<td>MaPLE MapReduce Job 1: collecting ancestors for each concept.</td>
</tr>
<tr>
<td>4.2</td>
<td>MaPLE MapReduce Job 2: detecting non-lattice fragments.</td>
</tr>
<tr>
<td>5.1</td>
<td>A screenshot of the menu of search stems.</td>
</tr>
<tr>
<td>5.2</td>
<td>A screenshot of the concept search and displaying area.</td>
</tr>
<tr>
<td>5.3</td>
<td>A screenshot of hovering box displaying synonyms of “Myocardial infarction (disorder)”</td>
</tr>
<tr>
<td>5.4</td>
<td>A screenshot of the fragment index page of concept “Myocardial infarction (disorder)”</td>
</tr>
<tr>
<td>5.5</td>
<td>A screenshot of the visualization of a fragment.</td>
</tr>
<tr>
<td>6.1</td>
<td>A screenshot of the conjunctive ontology browser explorer interface COBE.</td>
</tr>
<tr>
<td>6.2</td>
<td>Searching “neck head region” in both NE and DL modes to render related non-lattice fragments.</td>
</tr>
</tbody>
</table>

Appendix

Figure
ACKNOWLEDGMENTS

First of all, I would like to express my deepest gratitude to my advisor, Dr. Guo-Qiang Zhang, for his excellent guidance, profound insight and constant support throughout my Master's program. I have been extremely fortunate to have an advisor who let me experience several interesting research projects and learn how to write scientific papers. He has provided an excellent atmosphere for doing research, which making the study experience at Case Western Reserve University unique.

Next, I would like to thank Dr. Licong Cui. She is a great mentor, from whom I have learned many clinical data retrieving techniques and programming skills. She has been like a second advisor to me, ready with honest advice and encouraging words whenever I needed them. This thesis would not have been possible without her.

Furthermore, I appreciate my committee member, Dr. Rong Xu for her time, insights, and valuable feedback that have helped improve my thesis and enrich my work.

I would also like to thank Wei Zhu and Shiqiang Tao. They are great sources of knowledge regarding web application developing and Ruby on Rails programming.
Also I am grateful to all the members of Center for Clinical Investigation, both past and present, for their support and friendship. I would like to extend my thanks to “Falafel lunch group.” Friday lunch time would not have been as colorful without them.

Last but not least, I want to thank my family, for their unconditional love and support for my education.
COBE: A Conjunctive Ontology Browser and Explorer for Visualizing SNOMED CT Fragments

Abstract

by

MENGMENG SUN

Many clinical ontology search interfaces have been developed to provide access to ontological information. However, existing search interfaces are insufficient for ontology quality assurance work. My master thesis focuses on developing a Conjunctive Ontology Browser and Explorer (COBE) for searching and exploring SNOMED CT concepts and visualizing SNOMED CT non-lattice fragments.

COBE combines navigational exploration (NE) with direct lookup (DL) as two complementary search modes. The NE mode provides a conjunctive mechanism for users to quickly find the most frequent concepts related to SNOMED CT non-lattice fragments. Users can interactively and incrementally narrow down (hence conjunctive) the search space by adding precomputed word stems provided by COBE, one at a time. Such word stems serve as attribute constraints, or “attributes” in Formal Concept Analysis, which allows users to navigate to specific SNOMED CT concept clusters. The DL mode represents the common search mechanism by using a collection of key words given by users, as well
as concept identifiers. The other defined feature of COBE is the visualization of fragments, which facilitating the further investigation and curation of such structures.

With respect to the NE mode, COBE leverages 28,371 of total 302,902 concepts occurring in the fragments of interest to construct the stem cloud. With merely 9.37% of the total SNOMED CT concepts, the exploratory navigation mode reaches 98.97% coverage of the entire concept collection. With respect to the DL mode, evaluation against manually created reference standard shows that COBE attains an example-based precision of 0.958, recall of 0.917, and F1-measure of 0.875.
Chapter 1

Introduction

Search and browsing interfaces for ontologies are an integral part of terminology system dissemination. Although ontology search interfaces are distinct from general search interfaces because of the semantic and structural information already contained in the ontologies, they can still benefit from the latest developments in information retrieval. For example, conjunctive exploratory navigation interfaces CENI and SCENI [1–3] have been developed for exploring consumer health questions with health topics as dynamically searchable tags complementing keyword-based lookup. The conjunctive exploration mechanism allows users to quickly narrow down to the most relevant results in the most effective way.

In this thesis, a conjunctive ontology browser and explorer (COBE) [4] is introduced for searching SNOMED CT concepts and visualizing SNOMED CT fragments. The COBE search interface has three prominent features. First, it supports both direct lookup and navigational exploration to retrieve concepts. In the direct lookup mode, a user comes with specific terms or SNOMED CT identifier, enters into the search interface, and retrieves a list of relevant concepts. In the exploratory mode, a user may not have a targeted term,
or cannot easily or effectively formulate descriptive lookup terms, and may rely on navigational features to browse and explore the concepts. COBE provides conjunctive search combining direct lookup with navigational exploration. Second, COBE meets the need of a concept retrieval interface to reach and visualize erroneous SNOMED CT fragments for quality assurance. COBE makes it feasible to systematically learn the structure of non-lattice fragments, discover the internal mistaken relationships accounting for the cause of it. Third, in the navigation exploration mode, COBE utilizes a cloud of core stems mined from non-lattice fragments to serve as the navigation menu.

In previous study [5], SNOMED CT mining non-lattice fragments have been shown to be an effective approach for detecting abnormal structures that are inconsistent with the principle that the subsumption relation (is-a) in an ontological system should conform to the lattice property. This principle is further enforced in more recent work [6], where non-lattice fragments have shown up to 40 times the change rate against background changes in the evolution of SNOMED CT.

However, such obscure structure is hard to interpret since the barrier of visualizing multiple layer of concepts and is-a relations commonly in one graph. Although several SNOMED CT browser is available online, such as IHTSDO SNOMED CT Browser [7] and NLM SNOMED CT Browser [8], they are not suitable for ontology quality assurance tasks.
It’s necessary to visualize the fragments in a more comprehensive way since our result is indicated by SNOMED Identifier. It is not necessarily easy for user to access such fragments information for ontology assurance. In this thesis, a conjunctive ontology browser is introduced to explore SNOMED CT fragments for quality assurance, which enabling multi-ways search method to locate concept and display associated non-lattice fragments.

The effectiveness of COBE for retrieving SNOMED CT concepts is evaluated in two ways. For the direct lookup mode, it is compared with the well-known IHTSDO SNOMED CT Browser and NLM SNOMED CT Browser. Evaluation against manually created reference standard showed that COBE attains an example-based precision of 0.958, recall of 0.917, and F1-measure of 0.875. For the navigational exploration, we conduct an evaluation experiment on the search stems. The result shows that only using stems generated from 28,371 concepts which occurring in the upper bounds (uppermost level) of fragments, which is 9.37% of total, as exploration navigation menus covers 98.97% of concepts.
Chapter 2

Background

2.1 SNOMED CT

SNOMED CT (Systematized Nomenclature of Medicine - Clinical Terms) is the world’s largest and most comprehensive clinical healthcare terminology, developed and maintained by the International Health Terminology Standard Development Organization (IHTSDO). SNOMED CT content is represented using three components: concepts, descriptions, and relationships. Concepts represent clinical thoughts in the form of unique numeric concept identifiers. Descriptions link appropriate human readable terms to concepts. A concept can have several associated descriptions, each representing a synonym that describes the same clinical concept, discriminated by description identifiers. SNOMED CT concepts are linked to other concepts whose meaning is related in some way. These relationships provide formal definitions and other properties of the concept. One type of relationship is the is-a relationship which relates a concept to more general concepts. These is-a relationships define the hierarchy of SNOMED CT concepts.

SNOMED CT categorizes terminology in 19 different sub-hierarchies. Within each sub-hierarchy, concepts are organized from the general to the more detailed ones. Each
sub-hierarchy has its own sub-hierarchy root, which shares one particular root: SNOMED CT Concept. No concept is shared across sub-hierarchies except for the root. We use the sub-hierarchy root concept to represent its sub-hierarchy. The root node for 19 sub-hierarchies are: Procedure, Physical force, Event, Staging and scales, Substance, Environment or geographical location, Situation with explicit context, Body structure, Observable entity, Pharmaceutical/biologic product, Physical object, Qualifier value, Special concept, Specimen, Social context, Clinical finding, Organism, Linkage concept, and Record artifact. From a structural perspective, SNOMED CT can be seen as a series of concepts linked to each other, which form acyclic graphs [9].

2.2 SNOMED CT Quality Assurance

It has been noted that the biomedical domain are developing tremendously. For example, SNOMED CT releases a new version almost every 6 months. The quality of an ontology is a key issue that determines its usability. Thus ontology quality assurance has become an indispensable part of the ontological development. Many auditing methods exist to ensure the quality of ontologies. They include but are not limited to lexical [10], structural [11], semantical [12, 13] and statistical-based [14]. A lattice structure is often one of the criterion of the well-formed of ontologies [5]. Lattice is a structure that every two concepts in the ontology have no more than one minimal common ancestor or maximal common descendant. In SNOMED CT, the subsumption relationship (is-a hierarchy) should form a lattice [15]. Non-lattice fragments often indicates the structural anomalies
in ontological systems and represents the possible areas of focus for the subsequent quality assurance work [5].

The previous work MaPLE [6] has found a large number of such non-lattice structure in SNOMED CT. Fig. 2.1 is a non-lattice fragment from SNOMED CT. The double-circled concepts “Tissue specimen from breast” and “Tissue specimen from heart” share two minimal ancestor: “Tissue specimen” and “Specimen from trunk,” highlighted in pink, which makes them a non-lattice pair. To make it lattice-conforming, one could add the concept “Tissue specimen from trunk” (dashed in Fig. 2.1) [5, 6, 11].

![Figure 2.1 A non-lattice fragment in SNOMED CT [6].](image)

### 2.3 SNOMED CT Browsers

SNOMED CT Browsers are applications and tools used for viewing terminology content or hierarchy structure of SNOMED CT. Typically, a SNOMED CT browser can display descriptions and relationships of a concept by providing searching terms or identifier. The official *IHTSDO SNOMED CT Browser* is an online, multilingual, multi-Edition ontology
browsing application. It enables browsing of International Edition of SNOMED CT, as well as various National Editions. User can enter terms or navigate in the taxonomy hierarchies to find target concept, with filters in semantic, refset, or language. The application can provide almost everything related to the concept, including descriptions, parents and children, attributes and diagram of structure. Another online browser, provided by National Library of Medicine (NLM), is *NLM SNOMED CT Browser*. Different from the IHTSDO SNOMED CT Browser, it provides a SNOMED CT-centric view of the UMLS Metathesaurus and retrieves SNOMED CT content in UMLS concepts, expanding searches to include synonymous terms from over 100 Metathesaurus vocabulary sources. CSIRO provides a graph-based interactive interface for navigating SNOMED CT concepts, which is called *Shrimp* [16]. One can expand the hierarchy graph by clicking the concept, which provides an intuitive visualization of its relationships.

### 2.4 Conjunctive Navigational Exploration

Two basic search mechanisms exist in information retrieval. One is direct lookup (DL), and the other is navigational exploration (NE) [4]. Direct lookup refers to an intuitive search mode where a user knows precisely what to look for and comes up with input strings leading uniquely to the search target. However, direct lookup search is insufficient for consumers to take full advantage of rich public information resources. Exploratory navigation is considered the other search mechanism where the user may not be able to easily and
effectively formulate a descriptive search string, and must rely on navigational tools such as menus to browse and explore the content in order to inform the user “what is there.”

Conjunctive Exploratory Navigation Interface (CENI [1]) is a recently introduced technique for effective retrieval and navigation of online consumer health information using the NE mechanism. CENI is achieved by assigning multiple topics for information items using semantic tagging [2, 3]. Crowd-sourced comparative evaluation revealed that anonymous users from Amazon Mechanical Turk preferred 2 to 1 for CENI against other search mechanisms [1].

A defining feature of CENI is an interface which provides the user multiple paths to quickly narrow down to relevant contents. By selecting one topic at a time, in an incremental fashion, a user arrives at narrower and narrower content areas that are relevant to all the topics selected so far, conjunctively.

A novel feature of COBE is the navigational exploration search mode, adapted from CENI. Instead of consumer health topics as menus in CENI, COBE uses word stems from an important sub-collection of SNOMED concepts – those appearing in the uppermost level of non-lattice fragments. Such word stems serve as attribute constrains, or “attributes” in Formal Concept Analysis, which allows the user to navigate to specific SNOMED CT concept clusters in multiple ways.
Chapter 3

Overview of COBE Search Interface

3.1 COBE Functional Architecture

Figure 3.1 depicts the overall architecture of the proposed COBE to search SNOMED CT concepts and facilitate the retrieval of non-lattice fragments in SNOMED CT.

As can be seen in Fig. 3.1, two basic search pipelines exist in COBE, which denoted by the double border rectangular box. One is NE-based navigational exploration search, and the other is DL-based direct lookup search. The NE mode is a novel feature of COBE interface adapted from CENI.

In the Navigational Exploration mode, a user may want to explore “what is there;” or may not be able to easily or effectively formulate a descriptive lookup term to target non-lattice fragments. In either case, the user may rely on navigational menus or facets to browse and explore. To address such need, we construct a cloud of informative terms serving as the navigational menus for the user to effectively explore SNOMED CT concepts and non-lattice fragments.

In the direct lookup (DL) mode, a user knows what to look for, comes with specific term or exact SNOMED Identifier, and tries to retrieve a list of concepts. After a user types
a specific term or SNOMED CT identifier into the search box, COBE performs terming splitting, and DL-based search is conducted to find a list of relevant concepts as well as numbers of related non-lattice fragments. The user need to specify an input term in the direct search box instead of formulating a term by the available navigation stems.

3.2 Navigational Exploration

For the navigational exploration mode, SNOMED CT concepts are preprocessed (dotted rectangular box) to obtain a collection of core stems. SNOMED CT concepts are filtered
by non-lattice fragments, and performed word tokenizing and frequency ranking. As a result, a collection of core stems are mined and used for tagging concepts. In the navigational exploration (NE) mode of the COBE search interface, a user’s input (NE-input in Fig. 3.1) is the selection of stem tags, based on which COBE performs NE-based conjunctive search and returns to the user a list of relevant concepts as well as numbers of related non-lattice fragments. To achieve this, data processing is performed to formulate the search stems. This data processing procedure will be talked in section 4.4.

Given a selection of input stems by a user, COBE performs conjunctive search to retrieve matching concepts, that is, only concepts matching all the selected stems are returned, ranked by the number of related non-lattice fragments in the descending order. COBE allows concepts being investigated and narrowed down by several times of navigation sequentially. For example, clicking navigation stem “heart” followed by clicking “attack” will narrow down the results to “Myocardial infarction (disorder).” COBE enables the combination of navigational exploration and direct lookup. For instance, clicking “attack” followed by typing “heart” in the direct search box will narrow down the results to “Myocardial infarction (disorder).”

### 3.3 Direct Lookup

Direct lookup refers to an intuitive search mode where a user knows precisely what to look for and comes up with input strings leading uniquely to the search target. A user’s input (DL-input in Fig. 3.1) is a specified term formulated by the user. After a user types
a specific term into the direct search box, COBE splits the term into stems and performs conjunctive search to retrieve concepts matching all these stems, and returns them by the number of related non-lattice fragments in the descending order.
Chapter 4

Data Processing

This chapter describes the processing of the input data for COBE interface. Since Hadoop Map-Reduce framework is used in several steps of the raw data processing, this chapter starts with the introduction of the distributed computing framework and then explaining each execution step of the data processing. First step is extracting the active concepts, concept descriptions and is-a relationships from SNOMED CT US Edition raw data files. The next step is computing the transitive closure and non-lattice fragments using Map-Reduce. The last is processing the concept descriptions to form the search stems served as the navigation menu of the browser.

4.1 Hadoop Map-Reduce Framework

MapReduce is a programming model originally synthesized by Google to process and generate large data sets in a parallel distributed way. A MapReduce job consists of a map and a reduce function, usually defined by the user to adapt the form of key-value pairs of the data. Instead processing data on one single powerful machine, MapReduce is designed
to distribute the computational load to a cluster of computing machines to work at the same
time, which can shorten the overall computational time.

One famous MapReduce computing framework is Apache Hadoop. Hadoop has a
master-slave architecture [17]. Master refers to the name node and slave indicates the
data node. Systematically, name node breaks the MapReduce job into several tasks exe-
cuted in parallel across a cluster of machines of data nodes. The JobTracker runs on the
master node and TaskTracker runs on the slave node, where JobTracker is responsible for
assigning tasks to various slave nodes and TaskTracker is responsible for running map and
reduce processes on the data stored in HDFS - a distributed file system that stores data on
commodity machines. Once each data node gets its share of data to be processed, it will
map the required data to unique keys in map phase. In reduce phase, sorting and filtering
of the mapped data takes place and finally outputs the data either into HDFS or into any
other data store.

4.2 Active Concepts, Descriptions and Relationships Extraction

Data processing phase starts with the extraction of SNOMED CT core components:
active concepts, descriptions and relationships. The SNOMED CT raw data files can be
found though the National Library of Medicine website: http://www.nlm.nih.gov/
research/umls/Snomed/us_edition.html. The SNOMED CT version I use for the
The data source of active concepts is the file /SnomedCT_Release_US1000124_20140901/RF2Release/Snapshot/Terminology/sct2_Concept_Snapshot_US1000124_20140901.txt. The file contains four columns, which are *id*, *effectiveTime*, *activemoduleId* and *definitionStatusId*. However only the column *id*, which containing the concept identifiers, is demanded. In column *activemoduleId*, code 1 indicates active and 0 for inactive. Here I write a simple java code to extract the active concepts Identifiers from *id* column where the value of *activemoduleId* equal to 1. After the processing, the concepts list is generated in the form of SNOMED CT Identifier. 302,902 active concepts were found in this version.

The concept descriptions is in the file /SnomedCT_$Release_US1000124_20140901/RF2Release/Snapshot/Terminology/sct2_Description-en_Snapshot_US1000124_20140901.txt. The data desired from this file is concept identifiers and descriptions. Four columns are used to extract the description: *id*, *active*, *typeId* and *term*. Column *id* has the concept identifiers and column *term* contains the concept descriptions. In *typeId*, different typeId values indicated different kinds of specific descriptions. With the *typeId* value of 900000000000013001, the string in *term* is Fully Specified Name and 900000000000013009 is Preferred Name and Synonym. To enable the function of searching concept synonyms, the fully specified name, preferred name and synonyms are all extracted, using another java program, and then stored into one table called “concepts” in MySQL database.

The usage of the relationships is essential while computing or visualizing fragments. The is-a relationships can be found in the file /SnomedCT_Release_US1000124_20140901/RF2Release/Snapshot/Terminology/sct2_Relationship_Snapshot_US1000124_20140901.
txt. Another java code is executed to fetch the data of column sourceId and column destinationId where the value of column typeID equal to 116680003 and active equal to 1, which indicates “sourceId” is-a “destinationId.”

4.3 Computing SNOMED CT Non-lattice Fragments

Computing fragments consists of two sequential steps. First non-lattice pairs of concepts $C = c1, c2$ are computing according to the MaPLE algorithm [6] using MapReduce big data approach. Given the result of non-lattice pairs and the set $L$ of their maximal lower bounds, the corresponding non-lattice fragments are then computed.

MaPLE algorithm consists of two MapReduce jobs, as shown in Fig. 4.1 and Fig. 4.2 [6]. As described in Fig. 4.1, the first MapReduce job simply collects descendants for each concept, given a set of transitively closed pairs of SNOMED CT concepts. In the mapping phase (lines 3-5), each mapper reads in a set of transitively closed concept pairs and emits key-value pairs $(c, d)$ where $c$ is a concept and $d$ is a descendant of $c$. In the reduce stage
(lines 6-8), each reducer collects all the descendants \( c = \{d_1, d_2, \ldots \} \) of a concept \( c \) and emits the concept-ancestors pairs \((c, c')\).

```plaintext
1: Input: Concept-descendants pairs
2: Output: Non-lattice pairs and their maximal lower bounds
3: class Mapper
4: method Initialize
5: Initialize a HashMap CD to load concept-descendants pairs
6: method Setup
7: Load Distributed Cache files and update CA
8: method Map\((c_1, \{d_{11}, d_{12}, \ldots \})\)
9: for each concept \( c_2 \) in CD do
10: if \( c_1.\text{id} < c_2.\text{id} \) then \( \triangleright c_1 \)'s ID is less than \( c_2 \)'s ID
11: Emit((\( c_1, c_2 \)), \((\{d_{11}, d_{12}, \ldots \}, \{d_{21}, d_{22}, \ldots \})\))
12: end if
13: end for

14: class Reducer
15: method Initialize
16: Initialize a HashMap CD to load concept-descendants pairs
17: method Setup
18: Load Distributed Cache file and update CD
19: method Reduce((\( c_1, c_2 \)), \((\{d_{11}, d_{12}, \ldots \}, \{d_{21}, d_{22}, \ldots \})\))
20: \( N \leftarrow \{d_{11}, d_{12}, \ldots \} \cup \{d_{21}, d_{22}, \ldots \} \) \( \triangleright \) Union descendants
21: if \( c_1 \notin N \) and \( c_2 \notin N \) then \( \triangleright c_1, c_2 \) are incomparable
22: \( D \leftarrow \{d_{11}, d_{12}, \ldots \} \cap \{d_{21}, d_{22}, \ldots \} \) \( \triangleright \) Intersect descendants
23: \( L \leftarrow \emptyset \)
24: for each concept \( d \) in \( D \) do
25: \( L \leftarrow L \cup CD.Get(d) \) \( \triangleright \) Union \( d \)'s descendants
26: end for
27: \( B \leftarrow D - L \) \( \triangleright \) Calculate maximal lower bounds
28: if |\( B \)| > 1 then
29: Emit((\( c_1, c_2 \)), \( B \))
30: end if
31: end if
```

Figure 4.2 MaPLE MapReduce Job 2: detecting non-lattice fragments.

Fig. 4.2 shows the second MapReduce job to identify non-lattice pairs and their maximal lower bounds. The input is a set of concept-descendants pairs \((c, c')\) resulting from
MapReduce Job 1. In the map stage, each mapper first loads all concept-descendants pairs $(c, \varphi)$ into a hash map $CD$ to facilitate the generating of candidate pairs (lines 4-7). Then each mapper generates candidate pairs for each concept $c_1$ (lines 8-13) by iterating through each concept $c_2$ in $CD$, checking if $c_1$’s concept ID is less than $c_2$’s concept ID to ensure uniqueness of the pair. A concept pair $(c_1, c_2)$ is then emitted as a key and their descendants $(\varphi_1, \varphi_2)$ where $\varphi_1 = \{d_{11}, d_{12}, \ldots\}$, $\varphi_2 = \{d_{21}, d_{22}, \ldots\}$, respectively, as a value. In the reduce stage, each reducer first loads all concept-ancestors pairs $(c, \varphi)$ into a hash map $CD$ (lines 15-18). Then each reducer checks if $c_1$ and $c_2$ in a concept pair are incomparable (lines 20-21), calculates their maximal lower bounds (lines 22-27), and emits pairs with more than one maximal lower bound as keys and their maximal lower bounds as values (lines 28-30). After implementing MaPLE in Cloudera Hadoop 4.3 on a 30 node cluster, non-lattice pairs and their maximal lower bounds are systematically extracted.

After the above steps, non-lattice fragments is generated using non-lattice pairs and their maximal lower bounds according to the formula [9]:

$$C \cup L \cup \bigcup_{c \in C, l \in L} \{c \cap l\}$$

(4.1)

Following are the steps of generating non-lattice fragments in parallel way using MapReduce.

- First two hash maps are built to store concepts and their upper-closures and down-closures, and distribute them to every computing nodes.
• Then, in the map phase, each mapper reads in a non-lattice pair of concepts $C$ and their maximal lower bounds $L$, finds down-closures for the concepts in $C$ and upper-closures for the concepts in $L$ from the hash maps, and performs set operations to get the non-lattice fragment.

• In the reduce phase, each reducer emits the non-lattice pairs and their corresponding non-lattice fragments.

4.4 Navigational Search Stems Computation

Navigational search stems offer a list of search stems to users whom may not come up with specific concepts. Since the search stems serves as the core role of navigation function, formulating a list of comprehensive search stems is essential for the COBE browser. The COBE will end up with the concept if no fragment found and no lattice structure to visualize. Thus the strategy for formulating stems is to limit the source of stems to those concepts occurring in the upper bounds of non-lattice fragments.

The strategy used to formulate the word stems is essential since the performance of COBE can be influenced by the size of stems. First thought is using fully specified name of concepts to generate the search stems. The second is formulating the word stems from all the concept descriptions including synonyms. The third one is using all descriptions of those concepts occurring in the upper bounds of non-lattice fragments.

I discard the first strategy in advance, since searching synonyms is an indispensable feature of SNOMED CT browsers, not limited to COBE. Many important word stems in
synonyms will be filtered out if using this strategy. To find out which one is best, the other two strategies are both experimented and then compared.

The winner is the thought of limiting the source of stems to those concepts occurring in the upper bounds of non-lattice fragments. Since upper bounds are keys to navigate to non-lattice fragments in COBE, they are deemed a bridge between concepts and fragments. Using this strategy, whatever stem tags the user clicks, the circumstance that all returning concepts having no related fragments to show will not happen in COBE. This guarantees that retrieved concepts from COBE at least have one related non-lattice fragment. Additionally, the exploratory navigation mode reaches 98.97% coverage of the entire concept collection with merely 9.37% of the total SNOMED CT concepts. This will be explained in detail in the section 7.1. Therefore, the thesis only concentrates on how stem process using the winner strategy. Following are the steps of processing search stems data.

- First, the active concepts are filtered by precomputed non-lattice fragments, and only keep those concepts appearing in the uppermost level of non-lattice fragments. If a concept appears in the uppermost level of a non-lattice fragment, then the non-lattice fragment is called related to the concept.
- Second, after all punctuation marks are removed, each of the remaining concepts are processed into lower case and then tokenized into individual words of stems.
- Third, all stop-words such as “of” and “out” are removed. All stop words are listed in the appendix.
• Fourth, for each stem, the frequency it appears in the the remaining concepts is computed.

After the above steps, the stems ranked by the precomputed frequencies serve as the navigational menus. Each stem is guaranteed to hit at least one fragment, which prevents the circumstance that retrieved concepts from COBE have no related non-lattice fragment.
Chapter 5

COBE Interface Implementation

To better understand this thesis, this chapter describes how COBE is implemented. This chapter starts with the COBE development environment, and then depicts the implementation of COBE interface. The implementation of COBE interface is divided into three core components: navigational search stems menu, concept search and display, and fragments visualization. Thus the following sections explain the development of each component in three dimensions - model, view, and controller, which facilitate the understanding of MVC (Model-View-Controller) architecture.

5.1 Development Environment

COBE is implemented using agile web development with Ruby on Rails. We used Ruby 2.1.2, Rails 4.0.0, along with MySQL 5.6.19 database to implement COBE search interface.

Ruby on Rails is an open source web development framework written in ruby programming language. We favor Ruby on Rails since its model-view-controller (MVC) framework
benefits web application developers from increasing development flexibility and productivity. The MVC framework is a software architectural pattern for web application. MVC framework consists of three interconnected parts: controller, model, and view. The controller send commands to the model to update current state, and send commands to the view to change the view’s presentation of the model. The model provide the logic for the application’s data domain, and stores the data directly retrieved from the database and displayed in the view. The view displays the output representation of the state of the model to the user.

5.2 Navigational Search Stems Menu Implementation

5.2.1 Model Implementation

In Ruby on Rails, Active Record is the M in MVC - the model - which responsible for representing data and logic of the application. Model Stem is built to manage the data related to the pre-computed stems. Following the mechanism of Ruby on Rails, model Stem has a corresponding MySQL database table called “stems.” This model can read stem data from database or write stem data to the database according to the actions. Table 5.1 describes the model, corresponding database table, and fields in “stems” table. The last column is the description of each field of the table.

Inside Stem class, we define a method called search. This method facilitates the user to find complicated stems when only knowing the linguistic root. For example, when user typing “laparo,” COBE will return “Laparoscopy,” “Laparotomy,” “Laparotrachelotomy”
and “Laparorrhaphy.” This method inserts the parameter into an underlying SQL where query and pass the statement to the database to find all stems containing this parameter.

<table>
<thead>
<tr>
<th>Model</th>
<th>Table</th>
<th>Field</th>
<th>Field Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Stem</td>
<td>stems</td>
<td>id</td>
<td>The id for the stem in stems table</td>
</tr>
<tr>
<td></td>
<td></td>
<td>word</td>
<td>The stem word</td>
</tr>
<tr>
<td></td>
<td></td>
<td>count</td>
<td>The number of concepts where the stem occurring in the descriptions</td>
</tr>
<tr>
<td></td>
<td></td>
<td>created_at</td>
<td>Timestamps of created</td>
</tr>
<tr>
<td></td>
<td></td>
<td>updated_at</td>
<td>Timestamps of updated</td>
</tr>
</tbody>
</table>

Table 5.1 The MySQL table and fields corresponding to model Stem.

5.2.2 Controller Implementation

A controller can be thought of as a bridge linked models and views together. In one direction, it requests data from models and passes the data to the view so that view can display that data to the user. In the other direction, it transfers data from the user input to the corresponding model to save or update.

For model Stem the corresponding controller is StemsController. Within StemsController, search action is defined to interact with model and view. After user types in a word into the search box, the StemsController passes model Stem the word as a parameter through this search action. After the model gets the search result from the database, the result will be saved to model Stem and wait StemsController to pass to the corresponding view to display to the user. Another main action within TopicsController is get_concepts. This action
is created to interact with model and view to accomplish the task of mining in the concepts. In each of the three concept search modes, when the user clicking stem tags or typing term into the direct search box, action `get_concepts` will be called and then determine what underlying data will be passed as parameters and which method of another model `Concept`, which discussed in the following section, will be used to search. Then the view displays the results from the controller action to the user.

5.2.3 View Implementation

By default, the user is presented with a list of search stems ranked by the frequent of stems occurring in the concepts related to non-lattice fragments as shown in Figure 5.1. The view is implemented using Twitter Bootstrap framework. Bootstrap is the most popular HTML, CSS, and JS framework for web application.

![Search Stem](image)

Figure 5.1 A screenshot of the menu of search stems.
5.3 Concept Search Implementation

5.3.1 Model Implementation

To search concepts, model Concept is built to retrieve and update information of all the concepts. Similar to Stem, for model Concept, we have a MySQL database table called “concepts” corresponding to the model. Table 5.2 describes the model, corresponding database table, and fields in each table for Concept. The last column is the description of each field.

<table>
<thead>
<tr>
<th>Model</th>
<th>Table</th>
<th>Field</th>
<th>Field Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Concept</td>
<td>concepts</td>
<td>id</td>
<td>The id for the concept in concepts table</td>
</tr>
<tr>
<td></td>
<td></td>
<td>cid</td>
<td>The SNOMED CT identifier for the concept</td>
</tr>
<tr>
<td></td>
<td></td>
<td>label</td>
<td>The fully specified name for the concept</td>
</tr>
<tr>
<td></td>
<td></td>
<td>description</td>
<td>All descriptions for the concept</td>
</tr>
<tr>
<td></td>
<td></td>
<td>syn</td>
<td>The synonyms for the concept</td>
</tr>
<tr>
<td></td>
<td></td>
<td>count</td>
<td>The number of fragments where the concept occurring in the upper bounds</td>
</tr>
<tr>
<td></td>
<td></td>
<td>created_at</td>
<td>Timestamps of created</td>
</tr>
<tr>
<td></td>
<td></td>
<td>updated_at</td>
<td>Timestamps of updated</td>
</tr>
</tbody>
</table>

Table 5.2 The MySQL table and fields corresponding to model Concept.

Within model Concept, three methods are defined to take different concept search tasks. Method search_by_stem is implemented for navigational exploration (NE) mode. When the user clicks search tags, COBE calls search_by_stem method. Similarly, search_by_concept is implemented for direct lookup (NL) mode; search_by_stem_concept is implemented for
navigational exploration (NE) mode. Which method being called depends on the controller actions. Each method can construct SQL query using parameters passed in, run the query in the database, and then retrieve the results.

5.3.2 Controller Implementation

Controller ConceptsController is implemented to receive the request to search concepts, fetch or save data from Concept model. Action get_concepts in StemsController is implemented to construct a bridge between Stem model and Concept model, which in charge of displaying all concepts resulted from a concept search tasks. The main action within ConceptsController is show. Action show is the bridge between Concept model and Fragment model, which responsible for fetching and displaying all the fragments associated with one concept. When the user clicking the leftmost button or text area of the concepts displaying area, action show will be called and all data of related fragments willed be returned. Then the view displays another page of index menu of fragments to the user.

5.3.3 View Implementation

As shown in Figure 5.2, the upper part is the direct lookup search box and the lower part is the concepts displaying area. By default, concepts displaying area is presenting a list of SNOMED CT concepts ranked by the frequent of concepts occurring in the upper bounds of non-lattice fragments, if neither search term nor search keyword is specified. The leftmost button shows the number of found non-lattice fragments related to that concept.
The button is green when there exists fragments related to that concept, while the button is orange if no fragments found related. The text shows the fully specified name for that concept and the rightmost blue button is the SNOMED CT Identifier. When hovering over the name, COBE pops up a floating text box showing all the synonyms of the concept.

Figure 5.2 A screenshot of the concept search and displaying area.

Figure 5.3 is an example for concept “Myocardial infarction (disorder).” When hovering over, COBE shows all synonyms of that concept, which are “Myocardial infarction,” “Heart attack, Cardiac infarction,” “Infarction of heart,” “MI - Myocardial infarction” and “Myocardial infarct.”
Figure 5.3 A screenshot of hovering box displaying synonyms of “Myocardial infarction (disorder).”

Figure 5.4 is the screenshot the fragment index page after clicking the concept “Myocardial infarction (disorder).” The left column of the index page contains all fragments denoted by upper bounds concept Identifier, and right column displays the corresponding concept labels. The leftmost button shows the number of concepts contained in that fragment. When clicking the fragments index, the browser directs to the page for the visualization of that fragment.

Figure 5.4 A screenshot of the fragment index page of concept “Myocardial infarction (disorder).”
5.4 Non-lattice Fragment Visualization Implementation

The method to visualize a graph is used for reference due to the similar structure of graph and fragments. Thus we use SVG (scalable vector graphics) empowered by the open-source D3.js drawing library to accomplish the visualization. This method is commonly used to view graphs.

5.4.1 Model Implementation

*Fragments* model is built to interact with the data used to visualize the fragments. To facilitate the technique of visualizing graphs, we need to store all the fragment data corresponding to the nodes and the edges of the graph. Here we use nodes to denote the concepts within the fragments and edges to present the is-a relationship that link two concepts together. Column “name,” “lower_bound” and “nodes” are used to save all the information of nodes and edges of the graph. Since we use topological sort as the rendering algorithm, each concept node can be assigned a level in the resulting graph. However the uppermost and the lowest level need to be determined in advance. As in Table 5.3, “name” denotes the concepts in uppermost level of the fragment; “lower_bound” represents concepts in the lowest level; “nodes” saves entire nodes in the fragment. The edges that link concept nodes together are stored in the column “edges.” Other information such like hierarchy, which indicates in which sub-hierarchy the fragments found, are also saved. These additional information could be used for the further development.
<table>
<thead>
<tr>
<th>Model</th>
<th>Table</th>
<th>Field</th>
<th>Field Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Fragment</td>
<td>fragments</td>
<td>id</td>
<td>The id for the fragments in fragments table</td>
</tr>
<tr>
<td></td>
<td></td>
<td>name</td>
<td>The fragment upperbounds</td>
</tr>
<tr>
<td></td>
<td></td>
<td>version</td>
<td>The SNOMED CT version of the fragment</td>
</tr>
<tr>
<td></td>
<td></td>
<td>lower_bound</td>
<td>The lowerbounds of the fragment</td>
</tr>
<tr>
<td></td>
<td></td>
<td>hierarchy</td>
<td>The hierarchy for the fragment</td>
</tr>
<tr>
<td></td>
<td></td>
<td>nodes</td>
<td>The nodes contained in the fragment</td>
</tr>
<tr>
<td></td>
<td></td>
<td>edges</td>
<td>The is-a edges contained in the fragment</td>
</tr>
<tr>
<td></td>
<td></td>
<td>node_count</td>
<td>The number of nodes in the fragment</td>
</tr>
<tr>
<td></td>
<td></td>
<td>created_at</td>
<td>Timestamps of created</td>
</tr>
<tr>
<td></td>
<td></td>
<td>updated_at</td>
<td>Timestamps of updated</td>
</tr>
</tbody>
</table>

Table 5.3  The MySQL table and fields corresponding to model Fragment.

5.4.2 Controller Implementation

Controller FragmentsController is implemented to receive the request to visualize the fragments. In this controller, the main action to manage the interaction is index. The data from model Fragment are passed to FragmentsController and processed after index being called. A function called multiple_top_sort is specified to assign depth to each node in index. This function topological sorts all the nodes using breadth first search. Then views use the data of nodes, along with the node depth and linked edges passed from the controller to display the visualization.
5.4.3 View Implementation

Figure 5.5 is the screenshot of a fragment with upper bounds “301365009,82423001.” The green nodes represent the upper bounds of the fragments, the yellow ones represent lower bounds and the grey ones represent intermediate nodes in between. The solid gray edges represent is-a relationship of two concepts. For example, “Chronic pain (finding)” and “Pain of head and neck region (finding)” are upper bounds of the fragments, while “Chronic neck pain (finding)” and “Chronic pain in face (finding)” are lower bounds of the fragment. The leftmost grey edge means “Chronic neck pain (finding)” is-a “Chronic pain (finding).”

![Figure 5.5 A screenshot of the visualization of a fragment.](image-url)
Chapter 6

Result

6.1 Basic Statistics

302,902 active concepts and 60,322 non-lattice fragments are found in September 2014, U.S. edition of SNOMED CT. Table 6.1 shows 10 largest sub-hierarchies that containing most non-lattice fragments. N means the total number of concepts in the sub-hierarchy. NL represents the total number of fragments found in the sub-hierarchy.

Table 6.2 shows 10 most frequent concepts occurring in the upper bounds of fragments. NN means the total number of non-lattice fragments upper-bounds related to the concept. As can be seen from Table 6.2, the top 10 frequent concepts are from the 3 largest sub-hierarchies - 5 are form “Clinical finding,” 4 from “Procedure” and 1 from “Body structure.”

Among all the 302,902 active concepts, only 28,371 are in uppermost level of non-lattice fragments. 12,623 search stems are generated from the descriptions of 28,371 concepts. Table 6.3 displays the 10 most frequent stems appearing in the concepts related to non-lattice fragments.
Table 6.1  Summary of 10 sub-hierarchies containing the most fragments.

<table>
<thead>
<tr>
<th>ROOT ID</th>
<th>Sub-Hierarchy</th>
<th>N</th>
<th>NL</th>
</tr>
</thead>
<tbody>
<tr>
<td>404684003</td>
<td>Clinical Finding</td>
<td>101,777</td>
<td>66,513</td>
</tr>
<tr>
<td>71388002</td>
<td>Procedure</td>
<td>54,407</td>
<td>58,498</td>
</tr>
<tr>
<td>123037004</td>
<td>Body Structure</td>
<td>30,719</td>
<td>23,018</td>
</tr>
<tr>
<td>105590001</td>
<td>Substance</td>
<td>24,257</td>
<td>6,415</td>
</tr>
<tr>
<td>373873005</td>
<td>Pharmaceutical/Biologic Product</td>
<td>17,003</td>
<td>3,525</td>
</tr>
<tr>
<td>410607006</td>
<td>Organism</td>
<td>33,294</td>
<td>666</td>
</tr>
<tr>
<td>123038009</td>
<td>Specimen</td>
<td>1,483</td>
<td>565</td>
</tr>
<tr>
<td>48176007</td>
<td>Social Context</td>
<td>4,726</td>
<td>314</td>
</tr>
<tr>
<td>362981000</td>
<td>Qualifier Value</td>
<td>9,137</td>
<td>256</td>
</tr>
<tr>
<td>363787002</td>
<td>Observable Entity</td>
<td>8,342</td>
<td>226</td>
</tr>
</tbody>
</table>

6.2  Conjunctive Ontology Browser to Explore SNOMED CT Fragments (COBE)

Figure 6.1 shows a sample screenshot after clicking “neck” and “head” from search stems for navigational exploration and typing “region” for direct search. The left column displays the list of navigation terms, while clicking the navigation menu bars of the left column, the chosen words are displayed inside the horizontal bar on the top of the right column, where the “Reset” button is used to start a new exploration by clearing the search words. The leftmost button of the right column shows the number of non-lattice fragments where the upper bounds contains this concept, where green indicates existing non-lattice fragments and orange for no fragments found. The center area indicates the fully specified
Table 6.2  Summary of the most frequent concepts in upper bounds of non-lattice fragments of SNOMED CT.

name and blue button for the SNOMED CT concept identifier, when hover the name COBE shows all the preferred terms and synonyms. Clicking both the green button and the concept name links to the page of the visualizing the fragments.

### 6.3 Visualizing Non-lattice Fragments

As shown in the right column in Figure 6.1, COBE enables user to target some specified concepts in the ontology display area. Once the concepts are located, a user can click the leftmost green or orange button to link to the index page of associated fragments and finally direct users to the visualization of such specific non-lattice fragments.
Table 6.3  Top-10 most frequent navigation search stems and the frequency.

<table>
<thead>
<tr>
<th>Search Stem</th>
<th>Frequency</th>
</tr>
</thead>
<tbody>
<tr>
<td>structure</td>
<td>2,640</td>
</tr>
<tr>
<td>finding</td>
<td>1,292</td>
</tr>
<tr>
<td>disorder</td>
<td>1,070</td>
</tr>
<tr>
<td>neoplasm</td>
<td>1,041</td>
</tr>
<tr>
<td>procedure</td>
<td>971</td>
</tr>
<tr>
<td>joint</td>
<td>755</td>
</tr>
<tr>
<td>artery</td>
<td>654</td>
</tr>
<tr>
<td>bone</td>
<td>641</td>
</tr>
<tr>
<td>system</td>
<td>629</td>
</tr>
<tr>
<td>entire</td>
<td>615</td>
</tr>
</tbody>
</table>

Figure 6.2 demonstrates searching “neck head region” in both NE and DL modes to render a non-lattice fragment. The upper left part of Figure 6.2 demonstrates the navigational exploration mode of COBE, and upper right part for the direct lookup. Using each mode or combining using these two modes leads to the visualization part. The lower left part of Figure 6.2 is the fragment index page for the concept “Pain of head and neck (disorder).” The left column of the index page contains all fragments denoted by upper bounds concept Identifier, and right column displays the corresponding concept labels. After clicking the fragments index, the browser directs to the page for the visualization of that fragments which is shown in the lower right part. The green nodes represent the upper bounds of the
Figure 6.1  A screenshot of the conjunctive ontology browser explorer interface COBE.

fragments, the yellow ones represent lower bounds and the grey ones represent intermediate nodes in between. The solid gray edges represent is-a relationship of two concepts. For example, “Chronic pain (finding)” and “Pain of head and neck region (finding)” are upper bounds of the fragments, while “Chronic neck pain (finding)” and “Chronic pain in face (finding)” are lower bounds of the fragment. The leftmost grey edge means “Chronic neck pain (finding)” is-a “Chronic pain (finding).”
Figure 6.2  Searching “neck head region” in both NE and DL modes to render related non-lattice fragments.
Chapter 7

Evaluation

7.1 Evaluation Method

The COBE interface is evaluated in two ways. For the navigational exploration mode, it's evaluated if the search stems obtained from the preprocessing step can cover a wider range of concepts. To calculate the percentage of the coverage, we use the total number of concepts as the denominator, and the number of concepts that can be reached by at least one search stem as the numerator:

\[
\text{coverage} = \frac{\text{number of concepts that can be reached by at least one search stem}}{\text{total number of concepts}}
\]

For the direct lookup mode, the search performance of the COBE interface is evaluated by being compared with other two SNOMED CT browsers: IHTSDO SNOMED CT Browser and NLM SNOMED CT Browser. 24 concepts are randomly selected as search tasks (one lookup term per task) from the CORE Problem List Subset of SNOMED CT to compare the three browsers. Since most SNOMED CT concepts have a short length of descriptions, the terms chosen for search tasks are 1-word, 2-words, 3-words or 4-words to make sure a certain number of concepts be returned. For each search task, two evaluators use three browsers to retrieve a list of relevant concepts. A gold standard is created for
each search task: the common concepts found by all three browsers are considered correct results and included in the gold standard; for the other concepts found but not shared by all three browsers, two evaluators manually review them and include the relevant concepts into the gold standard.

For each search task, an evaluator uses three browsers to retrieve a list of relevant concepts, respectively.

A gold standard is created afterwards for each search task: the common concepts found by all three browsers are considered correct results and included in the gold standard. For those concepts found but not shared by all three browsers, two evaluators manually independently review them and decide whether including into the gold standard or not. Those results are included in gold standard only if both evaluators consider them correct according to the relevancy of the results and the search task.

Example-based precision, recall and $F_1$ measure [2, 3, 18, 19] is used to evaluate the performances of the three browsers. Let $R$ be the reference standard consisting of $m = 24$ search tasks $\{(s_i, Y_i) \mid i = 1, \ldots, m\}$, where $Y_i$ is the set of all concepts included in the gold standard for the search task $s_i$. Let $Z_i$ be the set of concepts retrieved from a search interface for $s_i$. The example-based precision ($P$), recall ($R$) and $F_1$ measure ($F_1$) are calculated as follows:

\[
P = \frac{1}{m} \sum_{i=1}^{m} \frac{|Y_i \cap Z_i|}{|Z_i|},
\]

\[
R = \frac{1}{m} \sum_{i=1}^{m} \frac{|Y_i \cap Z_i|}{|Y_i|}, \text{ and}
\]

\[
F_1 = \frac{2 \cdot P \cdot R}{P + R}.
\]
\[ F_1 = \frac{1}{m} \sum_{i=1}^{m} \frac{2|Y_i \cap Z_i|}{|Z_i| + |Y_i|}. \]

### 7.2 Navigational Exploration Evaluation Result

Instead of the descriptions of all 302,902 concepts, a subset of 28,371 concepts is used to generate the search stems for navigational exploration. Each of the 28,371 concepts must occur in the upper bounds of a fragment at least one time. In the processing described in section 4.4, 12,623 search stems are generated after punctuation marks being removed, tokenized, and stop words being removed.

To evaluate the search stems, the evaluator comes up with a piece of test code to query all search stems for each of the 302,902 concepts, and finds that a subset of 299,789 concepts can be reached at least once. This indicates that using navigation exploration, at most 3,113 concepts might be missed. These 3,113 concepts must have no occurrence in the upper bounds of fragments, which implies targets of no interest. The coverage of the search stems to reach overall concepts is 98.97% (299,789/302,902). This indicates that the search stems formed by a small subset of concepts 9.37% (28371/302902) can cover most words of SNOMED CT concepts.

### 7.3 Direct Lookup Evaluation Result

To evaluate the search interface, 24 search tasks are designed to compare the three SNOMED CT browsers’ performances: COBE interface, IHTSDO SNOMED CT Browser and NLM SNOMED CT Browser. Table 7.1 shows the search tasks, the number of concepts
<table>
<thead>
<tr>
<th>Search Term</th>
<th>Y</th>
<th>$\mathbb{Z}_C(\mathbb{Z}_C \cap Y)$</th>
<th>$\mathbb{Z}_I(\mathbb{Z}_I \cap Y)$</th>
<th>$\mathbb{Z}_N(\mathbb{Z}_N \cap Y)$</th>
</tr>
</thead>
<tbody>
<tr>
<td>Panniculitis</td>
<td>39</td>
<td>39 (39)</td>
<td>39 (39)</td>
<td>39 (39)</td>
</tr>
<tr>
<td>Hepatoblastoma</td>
<td>2</td>
<td>2 (2)</td>
<td>2 (2)</td>
<td>2 (2)</td>
</tr>
<tr>
<td>Neurilemmoma</td>
<td>13</td>
<td>13 (13)</td>
<td>13 (13)</td>
<td>13 (13)</td>
</tr>
<tr>
<td>Oligomenorrhea</td>
<td>3</td>
<td>3 (3)</td>
<td>3 (3)</td>
<td>3 (3)</td>
</tr>
<tr>
<td>Microcephalus</td>
<td>5 (5)</td>
<td>5 (5)</td>
<td>5 (5)</td>
<td>5 (5)</td>
</tr>
<tr>
<td>Thalassemia</td>
<td>64</td>
<td>64 (64)</td>
<td>57 (57)</td>
<td>63 (63)</td>
</tr>
<tr>
<td>Hand pain</td>
<td>2</td>
<td>2 (2)</td>
<td>3 (2)</td>
<td>2 (2)</td>
</tr>
<tr>
<td>Mononucleosis syndrome</td>
<td>5</td>
<td>5 (5)</td>
<td>5 (5)</td>
<td>5 (5)</td>
</tr>
<tr>
<td>Deficiency anemias</td>
<td>2</td>
<td>2 (2)</td>
<td>2 (2)</td>
<td>2 (2)</td>
</tr>
<tr>
<td>Viral screening</td>
<td>4</td>
<td>4 (4)</td>
<td>4 (4)</td>
<td>4 (4)</td>
</tr>
<tr>
<td>Angina decubitus</td>
<td>1</td>
<td>1 (1)</td>
<td>1 (1)</td>
<td>1 (1)</td>
</tr>
<tr>
<td>Renal hypertension</td>
<td>17</td>
<td>17 (17)</td>
<td>16 (16)</td>
<td>16 (16)</td>
</tr>
<tr>
<td>Sleep terror disorder</td>
<td>1</td>
<td>1 (1)</td>
<td>1 (1)</td>
<td>1 (1)</td>
</tr>
<tr>
<td>Mantle cell lymphoma</td>
<td>4</td>
<td>4 (4)</td>
<td>4 (4)</td>
<td>4 (4)</td>
</tr>
<tr>
<td>Tricuspid incompetence, non-rheumatic</td>
<td>1</td>
<td>1 (1)</td>
<td>1 (1)</td>
<td>1 (1)</td>
</tr>
<tr>
<td>Acute frontal sinusitis</td>
<td>2</td>
<td>2 (2)</td>
<td>2 (2)</td>
<td>2 (2)</td>
</tr>
<tr>
<td>Renal tubular acidosis</td>
<td>11</td>
<td>11 (11)</td>
<td>11 (11)</td>
<td>11 (11)</td>
</tr>
<tr>
<td>Iatrogenic Cushing’s disease</td>
<td>1</td>
<td>1 (1)</td>
<td>1 (1)</td>
<td>1 (1)</td>
</tr>
<tr>
<td>Drusen of optic disc</td>
<td>4</td>
<td>1 (1)</td>
<td>1 (1)</td>
<td>4 (4)</td>
</tr>
<tr>
<td>Therapeutic drug monitoring assay</td>
<td>1</td>
<td>1 (1)</td>
<td>1 (1)</td>
<td>1 (1)</td>
</tr>
<tr>
<td>Malignant neoplasm of brain</td>
<td>14</td>
<td>14 (14)</td>
<td>13 (13)</td>
<td>12 (12)</td>
</tr>
<tr>
<td>Bronchopulmonary dysplasia of newborn</td>
<td>1</td>
<td>1 (1)</td>
<td>1 (1)</td>
<td>1 (1)</td>
</tr>
<tr>
<td>Antenatal ultrasound scan abnormal</td>
<td>1</td>
<td>2 (1)</td>
<td>2 (1)</td>
<td>1 (1)</td>
</tr>
<tr>
<td>Derangement of temporomandibular joint</td>
<td>2</td>
<td>1 (1)</td>
<td>1 (1)</td>
<td>2 (2)</td>
</tr>
</tbody>
</table>

Table 7.1 Comparison of searching results of 3 SNOMED CT Browser.
in gold standard, the number of concepts found by each browser and the number of concepts included in gold standard found by each search interface. \( Y \) is the number of concepts in gold standard \( Y \). \( Z_c \) means the number of concepts found by COBE. \( Z_c \cap Y \) indicates the number of concepts found by COBE that included in gold standard. \( Z_i \) is the number of concepts found by IHTSDO browser. \( Z_i \cap Y \) represents the number of concepts found by IHTSDO browser that included in gold standard. \( Z_N \) denotes the number of concepts found by NLM browser. \( Z_N \cap Y \) shows the number of concepts found by NLM browser that included in gold standard.

Three interfaces yield the same results for 17 of the 24 search tasks. The example-based precision (\( P \)), recall (\( R \)) and \( F_1 \) measure (\( F_1 \)) are calculated based on the results and the gold standard using formula in section 7.1. Table 7.2 shows the overall example-based precision, recall, and \( F_1 \) measures for the results of 24 search tasks using NLM, IHTSDO and COBE. The result shows COBE carries out the best recall of 0.917, NLM achieves the best precision of 1.0 and both COBE and NLM have the best \( F_1 \) of 0.875.

<table>
<thead>
<tr>
<th>SNOMED CT Browser</th>
<th>Precision</th>
<th>Recall</th>
<th>( F_1 )</th>
</tr>
</thead>
<tbody>
<tr>
<td>COBE</td>
<td>0.958</td>
<td>0.917</td>
<td>0.875</td>
</tr>
<tr>
<td>IHTSDO</td>
<td>0.958</td>
<td>0.792</td>
<td>0.75</td>
</tr>
<tr>
<td>NLM</td>
<td>1.0</td>
<td>0.875</td>
<td>0.875</td>
</tr>
</tbody>
</table>

Table 7.2 The example-based precision, recall, and \( F_1 \) measure for the results of concept search using COBE, IHTSDO and NLM browser.
Chapter 8

Discussions

I also experiment using the entire SNOMED CT concepts to generate search stems, which enabling us to compare the performance with the search stems from concepts tagged by non-lattice pair. Using the same processing method, 83,139 search stems are generated from the total of 302,902 concepts. The number of stems generated by non-lattice pairs is 12,623, which is 15.18% of the entire stems. However, using 15.18% of the entire stems can navigate to 98.97% of SNOMED CT concepts. This result demonstrates that non-lattice pairs hold significant information of SNOMED CT, which also consistent with the observation that the average change rates on the non-lattice pairs are up to 38.6 times higher than the change rates of the background structure (concept nodes) in our previous study [6].

In the evaluation of direct search, we manually review the search results to decide the gold standard for the example-based precision, recall and $F_1$ measure to evaluate the direct search. The common results found by all three browsers we always consider them correct ones. For those additional results, 2 evaluators independently check them according to the semantic relevance of the concepts. The results are classified as mismatches if even one evaluator consider those shouldn’t being returned. For example, both COBE and IHTSDO
browsers find an extra concept “Antenatal ultrasound scan for possible abnormality (procedure)” for search task “Antenatal ultrasound scan abnormal.” One evaluator considers it correct, but the other disagrees. So “Antenatal ultrasound scan for possible abnormality (procedure)” is excluded from the gold standard and lowers the precision of both COBE and IHTSDO from 1.0 to 0.958.
Chapter 9

Conclusion

In this paper, a conjunctive ontology browser COBE is presented to search and explore SNOMED CT concepts and non-lattice fragments for ontology quality assurance. The direct lookup and navigational exploration of COBE allows multiple entry points for users to explore information of interest. Enhanced by the search mechanism and feature of non-lattice fragments visualization, COBE provides a novel way for structural auditing of SNOMED CT. COBE combines navigational exploration (NE) with direct lookup (DL) as two complementary modes for finding specific SNOMED CT concepts. With respect to the DL mode, evaluation against manually created reference standard showed that COBE attains an example-based precision of 0.958, recall of 0.917, and F1-measure of 0.875. With respect to the NE mode, COBE leverages 28,371 concepts in non-lattice pairs to construct the stem cloud. With merely 9.37% of the total SNOMED CT stem cloud, our exploration navigation mode covers 98.97% of the entire concept collection. COBE has been deployed as the resident search interface for linking, exploring, and rendering of SNOMED CT non-lattice fragments, which represent the most active areas of ontology curation work.
# APPENDIX

## Stop Words List

<table>
<thead>
<tr>
<th>a</th>
<th>b</th>
<th>c</th>
<th>d</th>
<th>e</th>
<th>f</th>
<th>g</th>
</tr>
</thead>
<tbody>
<tr>
<td>h</td>
<td>i</td>
<td>j</td>
<td>k</td>
<td>l</td>
<td>m</td>
<td>n</td>
</tr>
<tr>
<td>o</td>
<td>p</td>
<td>q</td>
<td>r</td>
<td>s</td>
<td>t</td>
<td>u</td>
</tr>
<tr>
<td>v</td>
<td>w</td>
<td>x</td>
<td>y</td>
<td>z</td>
<td>about</td>
<td>above</td>
</tr>
<tr>
<td>after</td>
<td>again</td>
<td>against</td>
<td>am</td>
<td>an</td>
<td>and</td>
<td>any</td>
</tr>
<tr>
<td>are</td>
<td>as</td>
<td>at</td>
<td>be</td>
<td>because</td>
<td>been</td>
<td>before</td>
</tr>
<tr>
<td>being</td>
<td>below</td>
<td>between</td>
<td>both</td>
<td>but</td>
<td>by</td>
<td>cannot</td>
</tr>
<tr>
<td>could</td>
<td>did</td>
<td>do</td>
<td>does</td>
<td>doing</td>
<td>down</td>
<td>during</td>
</tr>
<tr>
<td>each</td>
<td>few</td>
<td>for</td>
<td>from</td>
<td>further</td>
<td>had</td>
<td>has</td>
</tr>
<tr>
<td>have</td>
<td>haven’t</td>
<td>having</td>
<td>he</td>
<td>her</td>
<td>here</td>
<td>hers</td>
</tr>
<tr>
<td>herself</td>
<td>him</td>
<td>himself</td>
<td>his</td>
<td>how</td>
<td>if</td>
<td>in</td>
</tr>
<tr>
<td>into</td>
<td>is</td>
<td>it</td>
<td>itself</td>
<td>me</td>
<td>more</td>
<td>most</td>
</tr>
<tr>
<td>my</td>
<td>myself</td>
<td>no</td>
<td>nor</td>
<td>not</td>
<td>of</td>
<td>off</td>
</tr>
<tr>
<td>on</td>
<td>once</td>
<td>only</td>
<td>or</td>
<td>other</td>
<td>ought</td>
<td>our</td>
</tr>
<tr>
<td>ours</td>
<td>ourselves</td>
<td>out</td>
<td>over</td>
<td>own</td>
<td>same</td>
<td>should</td>
</tr>
<tr>
<td>so</td>
<td>some</td>
<td>such</td>
<td>than</td>
<td>that</td>
<td>the</td>
<td>their</td>
</tr>
</tbody>
</table>
theirs  them  themselves  then  there  these  they
this  those  through  to  too  under  until
up  very  was  we  were  what  when
where  which  while  who  whom  why  with
would  you  your  yours  yourself  yourselves
APPENDIX
Ruby on Rails Code

B.1 Sample Controller

class StemsController < ApplicationController
  before_action :set_stem, only: [:show, :edit, :update, :destroy]

  def search
    unless params[:stem_search].blank?
      @stems = Stem.search(params[:stem_search]).order("count DESC")
    else
      @stems = Stem.all.order("count DESC").paginate(:per_page => 200, :page => params[:page])
    end
    respond_to do |format|
      format.js
    end
  end

  def add_stem_tags
    unless params[:stem_ids].blank?
      @stem_ids = params[:stem_ids]
      respond_to do |format|
        format.js
      end
    else
      render :nothing => true
    end
  end

  def get_concepts
    unless params[:stem_ids].blank?
      @stem_ids = params[:stem_ids]
      if params[:concept_search].blank?
        @concepts = Concept.search_by_stem(params[:stem_ids]).order("count DESC")
      else
        @concepts = Concept.search_by_stem(params[:stem_ids], :concept => params[:concept_search]).order("count DESC")
      end
      respond_to do |format|
        format.js
      end
    else
      render :nothing => true
    end
  end

end
B.2 Sample Model

class Stem < ActiveRecord::Base
  has_many :concepts
  belongs_to :concepts

  scope :sorted, lambda { order("stems.count DESC") }
  scope :search, lambda {|query|
    where(["word LIKE ?", "%#{query}%"])
  }

  def self.search(search)
    if search
      where("word LIKE ?", "%#{search}%")
    end
  end
end
B.3 Sample View

```html
<div class="row">
  <div class="span">
    <% form_tag search_stems_path, :remote => true, :method => :get, :id => "stem-search-form", :class => "navbar-form", :style => "margin-top:20px;margin-left:10px" do %>
      <div class="span">
        <div class="navbar">
          <ul id="stem-tab-list" class="nav nav-tabs" style="width: 200px;margin-left:20px;height: 460px;border-radius: 6px;">
            <% @stems = Stem.all.paginate(:per_page => 100, :page => params[:page]).order("count DESC")%>
            <% render :partial => 'stem_tab_list' %>
          </ul>
        </div>
      </div>  
    <% end %>
  </div>  
</div>

<div class="span">
  <%= form_tag get_concepts_stems_path, :remote => true, :method => :get, :id => "stem-concepts-search", :class => "navbar-form pull-right" do %>
    <div class="navbar-inner", style="width:700px;background:white;margin-top:25px;">  
      <div id="stem-tag-item-container", style="margin-left:10px;margin-top:5px;margin-bottom:10px;"%>
      </div>  
    </div>  
</div>  

<div id="search-item-container", style="border: none;margin-top:5px;margin-bottom:10px;">
  <%= text_field_tag :concept_search, params[:concept_search], :style => "width:270px;height:35px;padding-left:20px;padding-right:20px;margin-right:20px;border-radius:6px;"%>
  <%= submit_tag "Search Inside concepts", :name => nil, :id => "concept-search-button", :class => "btn btn-primary", :style => "margin-right:20px;"%>
  <%= button_tag "Reset", :type => 'reset', :class => "btn btn-primary start-over-button", :style => "float-right"%>
</div>  
</div>  
</div>

<% if false %> <!-- move search inside concepts to bottom -->
  <div class="navbar">
    <div class="navbar-inner">  
</div>  
</div>  
</div>
</div>
```
<% text_field_tag :concept_search, params: {
  concept_search } %>
<% submit_tag "Search Inside concepts", :name => nil, :id => "concept-search-button", 'data-disable-with' => 'Searching ...", :class => "btn btn-primary" %>
<% button_tag "Reset", :type => 'reset', :class => "btn btn-primary start-over-button" %>
$(
    '#stem-concepts-search').find('span').remove();
// each (function () {
    (this).remove();
})

$(
    '#stem-concepts-search').reset();

$(
    '#stem-search-form').reset();

// get stems
$.ajax({
    url: "/stems/search",
    beforeSend: function () { },
    data: $(
        '#stem-search-form').serialize(),
    success: function () { }
});

// get concepts
$.ajax({
    url: "/stems/get_concepts",
    beforeSend: function () { },
    data: $(
        '#stem-search-form, #stem-concepts-search').serialize(),
    success: function () { }
});
});
</script>
LIST OF REFERENCES


