A category-driven approach to deriving domain specific subsets of Wikipedia

Anton V. Korshunov, Denis Yu. Turdakov
{korshunov, turdakov} @ispras.ru

Jinguk Jeong, Minho Lee, Changsung Moon
Convergence Solution Team,
DMC R&D Center,
Samsung Electronics Co., Ltd.

{jinguk.jeong, minho03.lee, albert.moon}
@samsung.com

Abstract. While many researchers attempt to build up different kinds of ontologies by means of Wikipedia, the possibility of deriving high-quality domain specific subset of Wikipedia using its own category structure still remains undervalued. We prove the necessity of such processing in this paper and also propose an appropriate technique. As a result, the size of knowledge base for our text processing framework has been reduced by more than order, while the precision of disambiguating musical metadata (ID3 tags) has decreased from 98% to 64%.

Keywords: Wikipedia; ontology; automated ontology building; category; taxonomy; semantic relatedness; natural language processing; Texterra**.**

1. Introduction

There's no need to introduce Wikipedia as a world's largest and most rapidly expanding source of information on many domains in almost every language. At the time of drafting this paper, there are 279 Wikipedias in different languages, with more than 17,870,000 articles, 1,920,000 uploaded images, and 27,540,000 registered users [1]. 39 Wikipedias contain more than 100,000 articles each. The English edition remains the largest Wikipedia, over three times as large as the second largest edition, the German Wikipedia. That's why it is reasonable to start investigating new possibilities with English Wikipedia as a most comprehensive source.

By now, Wikipedia has already got a lot of colorful detailed descriptions. Conventional features of Wikipedia are well-known and discussed widely in [2, 25]. They include concept identification by ID or URL, multiple and dense link structure, and category system which is edited and maintained by Wikipedia users as well as articles.

Researching community has proved more than once that structure and content of Wikipedia are very peculiar and valuable domains to study. One of the most promising directions is *automated ontology building* which may be accomplished by extracting well-defined concepts and relations among them from Wikipedia. The latest efforts in this field are DBpedia [3], the work of Ponzetto et al. [4], YAGO [5], etc. All these approaches mainly exploit the data derived from infoboxes and category structure. Due to continuous enriching these techniques with better algorithms and data sources, the quality of resulting ontologies becomes remarkable.

Despite these successes in automated upper-level ontology building, most of domain specific ontologies (for instance, UMLS [6]) are still assembled manually. Needless to say, they are often costly to build and to keep them up to date.

In this paper, we consider the possibility of loosely supervised extraction of domain specific ontologies from upper-level ones. This is becoming feasible in recent years as high-quality upper-level ontologies grow more versatile and involve the knowledge of many new domains. Moreover, many field experts from all around the world often tend to pay their efforts to expand existing widespread ontologies, rather than to create their own or refine domain specific ones.

Our interest in this research is caused by necessity of reducing the knowledge base for our text processing framework Texterra[1]. This base comprises a number of textual indices produced by our Wikipedia parser. These indices are loaded into RAM during the initialization of Texterra server and take roughly 4.5 Gb of disk space and 2 Gb of RAM. Such a consumption is acceptable for workstations, but not for mobile devices with restricted amount of memory. Thus, if one would attempt to build a standalone mobile application intended for text processing, then its data structures simply would not fit in the RAM. Obviously, such applications - if developed - would be highly demanded to date.

The disambiguation[2] of musical metadata (i.e., ID3 tags of MP3 files) is an important part of Texterra functionality. These algorithms are well-tuned by the

1 http://modis.ispras.ru/texterra

2 **Word sense disambiguation** is an open problem of natural language processing, which governs the process of identifying which sense of a word (i.e. meaning) is used in a sentence, when the word has multiple meanings (polysemy).

moment and show the precision of 98% on our test set. But it's obvious that only a narrow range of Wikipedia dictionary is utilized during processing ID3 tags of MP3 files. There are mostly named entities, such as musical compositions, song writers, singers, etc. Generally speaking, the amount of knowledge required to perform such a disambiguation is likely much less than that available from the entire knowledge base.

Making the knowledge base more specialized by removing concepts which are unimportant for this task is what comes to mind first in this case. So, we've implemented a system intended to produce Wikipedia subsets covering the knowledge of the given domain. We decided not to narrow the scope of our research to Wikipedia derivates (such as YAGO). But, although the algorithms have been evaluated with Wikipedia only, they are applicable to any ontology with well-defined polyhierarchical taxonomy.

Of many interesting aspects of Wikipedia, here we take into account only categorization and linkage within its content. The category system in Wikipedia plays the role of a taxonomy and provides the function to search articles by narrowing down categories. Given this, the task of deriving domain specific concepts from Wikipedia dictionary seems to be solvable by mining the category system for the list of concepts tightly connected with the given *base categories*. However, it is impossible to simply determine all concepts belonging to a particular category. The reason is that the category system of Wikipedia is organized into network structure, not a perfect tree structure. Furthermore, the lists of parent categories for many articles are redundant and contradictory. Thus, they don't allow detecting the most relevant categories for a given page without analyzing the neighbour pages.

Therefore, we utilized *concept vectorization method* specialized for the category system in Wikipedia, with several additional expansion methods, as proposed in [2]. The authors express affiliation relations among concepts as *category-based concept vectors*. Each element (dimension) of a concept vector represents not only binary affiliation information (whether the concept belongs to a certain category or not), but also the degree of affiliation (called *belonging degree* in the rest of this paper). After applying different cutting techniques to the obtained concept vectors, the list of domain specific concept IDs is outputted. This list is further applied to Texterra knowledge base resulting in the reduced domain specific version of the latter. The explanation of our approach is given in Section 4.

Section 2 contains the review of related work. The key features of Wikipedia category structure are discussed in Section 3.

Having a fast access to any point of Wikipedia category structure was one of the crucial tasks for our research. After a number of unsuccessful tries to reuse the existing graph libraries, we've coded our own implementation, named **WikiGraph**. See Section 5 for details.

Given a set of specialized versions of the knowledge base, we conducted a series of experiments in order to learn how the reducing of Texterra knowledge base affects the accuracy of results. Refer to Section 6 for evaluation methodology. Section 7 contains experimental results and discussion.

We conclude in Section 8 with our research findings and discussion about possible improvements.

2. Related work

2.1 Automated ontology building

The field of automated ontology learning usually acts by taking textual input and transforming it into a taxonomy or a proper ontology. The texts are usually obtained from printed sources (books, magazines, newspapers) and Internet (online media, blogposts, results of querying search engines). However, the learned ontologies are small and hard-to-update; in addition, evaluations have revealed a rather poor performance [7].

As mentioned before, the most accurate and, thus, valuable non-human assembled ontologies are now built by automatically deriving explicit facts from Wikipedia. One of the early attempts was the work of Gregorowicz and Kramer [8]. They focused on deriving a term-concept map which consists of terms, concepts, and relationships between them. Only articles, redirects, and disambiguation pages are considered. Ponzetto and Strube [4] were deriving a taxonomy from the entire Wikipedia category structure. Despite their successes, the resulted taxonomy is rather simple (supports only *is-a* and *not-is-a* relations) and domain independent. The work described in [9] enhances this taxonomy with *instance* and *class* information for each node. Cui et al. [10] introduce even more sophisticated approach to building the ontology of concepts by making use of *infobox* structures, definition sentences, and category labels. Finally, YAGO2 [11] is probably the most complete and accurate semantic knowledge base derived automatically from Wikipedia and other sources. The information extraction technique for YAGO2 assumes varied utilization of infoboxes, category structure, redirects, and other data within Wikipedia. Furthermore, the quality check is performed to find possible

mistakes. As a result, the quality of extracted ontology is sufficient for the majority of IR- and NLP-related tasks.

Notwithstanding the foregoing, newly emerging research fields often require well-structured and comprehensive *domain specific* ontologies. Researchers don't need a huge knowledge base, but an extensible corpus of specific concepts with good coverage of domain knowledge. If such ontologies were built in a completely automated way, then this would avoid the necessity of assembling them from scratch manually. We believe that Wikipedia contains enough information for this task to be completed. Below is the description of approaches we've applied to narrowing Wikipedia dictionary to domain specific subset.

2.2 Computing semantic relatedness

We consider the key problem of our study as the computation of *semantic relatedness* between base top-level categories and underlying articles. According to Budanitsky and Hirst [12], semantic relatedness is defined to cover any kind of lexical or functional association that may exist between two words. This definition suits us more than *semantic similarity*, which is typically defined via the lexical relations of synonymy and hypernymy.

There were many approaches proposed for estimation of semantic relatedness between concepts in Wordnet (Rada et al., Leacock and Chodorow, Wu and Palmer, Resnik, Jiang and Conrath, Lin) and between Wikipedia articles [13] (Dice, Jaccard, SimRank). Among them are purely graph-based measures and those involving information content. In this paper, we consider only graph-based approaches.

Since a significant part of Wikipedia knowledge is encoded in its graph-like link structure, it seems reasonable to apply existing graph-based methods or introduce new ones. Zesch and Gurevych [14] proved that many Wordnet-based semantic similarity measures are applicable to Wikipedia with minor changes.

Obviously, it’s also attractive to estimate the strength of ties between different levels of taxonomy (between base categories and its articles, in our case). In this context, the links among articles appear not so important. Therefore, new category-based semantic relatedness measures are emerging.

Chernov et al. [15] measured semantic relatedness *between categories*, not between concepts and categories, as it's required for our task. Nonetheless, they proposed several very useful and applicable techniques.

Strube and Ponzetto [16] employed Wikipedia, Wordnet, and Google for computing semantic relatedness *between concepts*. For two Wikipedia articles being compared, they extracted two categories lists. Given the category lists, for each category pair they performed a depth-limited search of maximum depth of 4 for a least common ancestor. As they noticed, limiting the search improves the results. But this is obviously inappropriate for computing relatedness between top-level category and articles from all its subcategories.

2.3 Identifying domain specific concepts

Syed et al. [17] tried to predict the topic of textual documents by matching them against Wikipedia articles based on cosine similarity[3]. Then, they extracted categories of found articles and scored them based on different scoring schemes with or without spreading activation[4]. The proposed approach implies scoring the links with many categories for each given Wikipedia article using bottom-up traversing of category structure. This is acceptable for a small set of articles, but not for our task.

To the best of our knowledge, Cui et al. [18] were the first who introduced an approach to deriving domain specific corpus from Wikipedia. The main idea is to generate a domain hierarchy from the hyperlinked pages of Wikipedia. Then, only articles strongly linked to this hierarchy are selected. They build a so-called *Classification Tree* by traversing down the directed graph of Wikipedia category structure starting from the *root node*. This tree includes both categories and articles and in fact is merely a connected branch of Wikipedia classification graph with a specified root node. Then, the Classification Tree is traversed with a simple adaption of breadth-first search algorithm. During the traversal, each node is given a score on the relevance to the specific domain. Once the traversal is completed, the terminal nodes (article pages) are ranked according to the domain relevance scores. Pages over a certain threshold are considered domain relevant. The node score can consider either ingoing or outgoing edges. Despite the proposed technique is quite simple, the results are remarkable.

3 **Cosine similarity** is a measure of similarity between two vectors by measuring the cosine of the angle between them. Calculating the cosine of the angle between two vectors thus determines whether two vectors are pointing in roughly the same direction. This is often used to compare documents in text mining.

4 **Spreading activation** is a method for searching associative networks, neural networks, or semantic networks. The search process is initiated by labelling a set of source nodes with weights or "activation" and then iteratively propagating or "spreading" that activation out to other nodes linked to the source nodes.

A more sophisticated algorithm has been proposed later by Shirakawa et al. [2]. The *concept vectorization method* is introduced for finding concepts which are highly correlated with the base category (refer to Section 1 for brief explanation). The main assumption there is that the relatedness between categories gets lower as the number of traversed pages (i.e., hopcount) increases. In addition to the number of links for each node, they also take into account the *number of paths* between the concept and the base category, as well as *hopcounts* of these paths. As it seems to us, this understanding reflects the nature of Wikipedia classification approach much more precisely than ever before. Several heuristics are suggested for estimation of semantic relatedness by counting paths properties in the subgraph of desired base category. However, authors didn't compute these scores for thousands of articles with hundreds of paths for each of them at a time, as it's required in this research. Moreover, the efficiency of both approaches described in [2] and [18] has not been evaluated in real tasks. In both cases, the evaluation was performed by comparing the results of algorithms with answers of experts knowledgeable in certain fields. This looks persuasively when proving the theoretical applicability, but is not enough for unconditional embedding into the real system.

In this work, we mainly exploited the ideas formulated in [2] and [18]. Our main goal was to estimate the scalability and practical applicability of these approaches for real tasks which imply processing of large amount of data.

3. Features of Wikipedia category structure

The advantages of Wikipedia category structure were studied by authors of [14] and many others. Here we summarize only those features needed for better understanding of our approach.

Categories of Wikipedia can be organized in a graph, where the nodes are categories and the edges are hyperlinks. In this work we also add articles to this graph. However, we still name it the *Wikipedia category graph* (*WCG* in the rest of the paper).

The links expressing which concept belongs to what categories are called *category links*. We call them *belonging links* or *belonged links* according to their direction. In this paper, we only consider the belonging links, i.e. links from articles or subcategories to upper-level categories. The English version of Wikipedia, as of September 2010, contains ~13 million category links.


```
<page>
  <title>My Heart Will Go On</title>
  <id>923235</id>
  <revision>
    <id>390604275</id>
    <timestamp>2010-10-14T00:14:19Z</timestamp>
    ...
    <text xml:space="preserve">
    {{Infobox single
      | Name           = My Heart Will Go On
      ...
    }}

    "'''My Heart Will Go On'''" is the
    [[theme song]] of the 1997 blockbuster
    [[film]] ''[[Titanic (1997 film)|Titanic]]''.

    ...

    [[Category:Celine Dion songs]]
    [[Category:1997 singles]]
    [[Category:1998 singles]]
    [[Category:1990s ballads]]
    [[Category:Love themes]]
    [[Category:Pop ballads]]
    ...
    </text>
  </revision>
</page>
```

Fig. 1. Sample XML structure of categorized Wikipedia article page

The typical code of categorized article page is shown at Fig. 1. It combines XML and Wiki markup. The list of belonging categories is situated at the bottom. Categories have their own pages similar to articles. Category links at these pages also express which category belongs to what categories.

Categorization is a useful tool to group articles for ease of navigation, and correlating similar information. However, not every verifiable fact (or the intersection of two or more such facts) in an article requires an associated category. For lengthy articles, this could potentially result in hundreds of categories, most of which aren't particularly relevant. This may also make it more difficult to find any particular category for a specific article. Such *overcategorization* is also known as "*category clutter*" [19].

For these reasons, the WCG has an extremely complex nature. It is directed and has not a strong hierarchical structure as some may expect. Any category may branch into subcategories, and it is possible for a category to be a subcategory of more than one parent [20]. Upon closer inspection, the WCG is rather a *polyhierarchy*, or even a *net* (Fig. 2).

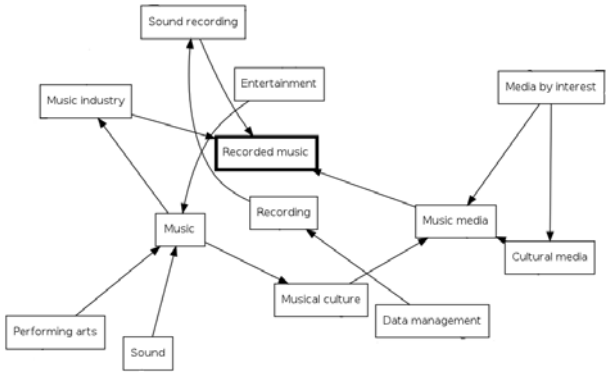

Fig. 2. A fragment of Wikipedia category structure

The figure has been produced by *CatGraph* [21]. This tool draws a cloud of links for the desired category. Each rectangle represents a category. Each arrow connecting two rectangles denotes a "*belongs-to*" relation, that is, the destination category is a subcategory of the initial one (an example of *belonging link*). The cloud shown in the figure is for "Recorded music" category (bolded).

It's worth noting here that not every Wikipedia page is categorized. According to statistics [22, 23], there are thousands of uncategorized articles and categories. Moreover, certain categories are assigned incorrectly [24]. We suggest considering these facts as a possible drawbacks for any category-based algorithm. In addition, automated categorizing (i.e., determining a topic of an uncategorized page) seems to be a challenging task. This can be done with certain accuracy by processing page title and text with specific NLP techniques and finding appropriate categories in WCG.

4. Deriving a domain specific subsets

The developed system consists of three main parts:

1. **Link Filter** produces a ready-for-load textual representation of WCG;
2. **Topic Deriver** performs the main processing;
3. **Reducer** produces a domain specific version of the Texterra knowledge base[5].

All algorithms evaluated in this paper were implemented in the Java programming language.

4.1. Link Filter

The input for **Link Filter** is *Wikipedia links file* containing information about all links between Wikipedia pages, along with their type and direction. The result is *category links file* that contains only links forming the WCG. Every line of this file denotes the affiliation of *belonging* between two pages and sets the *type of the belonging page*. For example,

12 780754 0

means that page with ID 12 ("Anarchism") is *belonging* to the category with ID 780754 ("Category:Anarchism"). Moreover, *the belonging page* ("Anarchism") is *an article* because the last field is "0" ("1" would mean that the belonging page is *a category*).

Thus, category links file is a complete textual representation of the WCG and contains no unwanted data such as page titles and link types. Furthermore, unlike the authors of [18], we've also removed pages of certain types: lists, classifications, portals, redirects, disambiguation pages, and user pages. This helped us to make the WCG more lightweight without loss of any meaningful concepts. As a result, category links file contains 13,001,687 links between 593,796 categories and 3,156,822 articles.

4.2. Topic Deriver

Topic Deriver loads category links file on start and fills in the internal structures of WikiGraph (refer to Section 5 for details). The workflow for this stage is shown at the Fig. 3. Here we touch on only the main steps.

5 The Texterra knowledge base for this research has been obtained by parsing the dump of English Wikipedia, as of September 2010.

For our analysis, we denote W as a set of concepts, V as a set of categories, and E as a set of belonging links. Then, the category system in Wikipedia is expressed as a directed graph G={W, V,E}. A path is a sequence of edges that connects one node with another. The path length (hopcount) is the number of edges along that path.

Musical compositions
Recorded music
Music-related lists

Texterra knowledge base

ID_1
940477
ID_2
2722900
ID_3
1520543

Wikipedia category graph

"Musical compositions"
SUBGRAPH
"Recorded music"
SUBGRAPH
"Music-related lists"
SUBGRAPH
"Musical compositions"
VECTOR
"Recorded music"
VECTOR
"Music-related lists"
VECTOR

16256
723678
90381
2763203
392
84678
280482
1.56
1.12
0.92
0.56
0.35
0.24
0.03

1234
491358
76036
386421
6032
16937
1.25
1.042
0.89
0.33
0.19
0.01
67891
2297721
6788201
404313
51578
30557
18358
1.02
0.90
0.78
0.44
0.36
0.20
0.09

LIST OF EXTRACTED CONCEPTS

Knowledge base (complete)

Knowledge base (domain specific)

Step 1. Get the **list of categories** to be processed.

Step 2. Get connected to Texterra server and obtain an **ID for each category**.

Step 3. Explore the pre-loaded Wikipedia category graph and build a **separate subgraph for each category**.

Step 5. Apply **cutting technique** to each vector. Collect the remaining IDs into common **list of extracted concepts**.

Step 4. Traverse the subgraphs and build **concept vectors**. Each vector consists of "**concept ID → belonging degree**" pairs.

Step 6. Process Texterra knowledge base in a way to save **only records associated with obtained list**.

Fig. 3. Overall architecture of Topic Deriver

The key task of Topic Deriver is to obtain the list of concepts connected semantically with certain domain. Herewith, this connection should be the tightest one, that is, these concepts should be more relevant to the desired domain than to others. As this task is computationally complex and, thus, supposed to be run rarely, it's allowably to choose the base categories manually for experiments. We've selected 3 base categories that likely cover the majority of concepts required for disambiguating musical metadata:

- *Category:Musical compositions*
- *Category:Recorded music*
- *Category:Music-related lists*

For each of selected base categories, a separate *subgraph* is built. This subgraph is almost the same as *Classification Tree* in [18]. The base category serves as a root node, and a tree-like structure of underlying pages is obtained from WCG. The only difference from approach proposed in [18] is that we use depth-first search (DFS), not breadth-first search (BFS). The reason for this is that the resulting subgraphs are often large enough, thus, it's inappropriate to waste the memory for storing the FIFO queue required for BFS traversal [26]. Moreover, as depth-first tree is expected to contain back edges and cross edges, the list of visited nodes has been added to avoid repetitive visiting and loops.

A *concept vector* in our research specifies the degree of affiliation between *the base category* and *each of articles* reachable by traversing down the subgraph of the base category starting from its root node. As mentioned, the heuristics for building concept vectors have been borrowed from [2] with some modifications. We describe them briefly below, for more detailed information refer to the source paper.

BVG (Basic Vector Generation method) generates concept vectors by tracking back parent categories in the category system and calculating the belonging degree to each concept.

The belonging degree I(wi,vj) from concept wi to category vj is defined by the following equation:

$$Iwi,vj = p \in Pij 1d(tl) \qquad (1)$$

Here, Pij denotes a set of paths from wi to vj, tl denotes the hopcount of path pl, d denotes a monotonically increasing function on the hopcount of path pl (given as 2tl).

It's noteworthy that in the original method [2] paths with a hopcount of more than 4 were ignored. We asked the authors for the reasons of this. The response was "*Because long paths scarcely affect values in concept vectors in most cases. Of course, sometimes long paths affect the values*". We've decided to remove this constraint in our experiments, that is, we consider *all paths* between two nodes.

As a result, processing time may become too large for base categories from high levels of the WCG hierarchy. The reason for such behaviour is an exclusive computational complexity of finding all paths between two arbitrary nodes in the graph. It's well-known that this task is NP-complete in general case.

Since WCG contains millions of edges, the maximal path length may reach hundreds of edges, leading to impetuous increase of processing time when trying to process top-level categories. This is exactly why we've picked up a "safe" set of categories, which are processed relatively fast and cover the knowledge of field we've chosen for experiments.

Notwithstanding, we believe that taking all existing paths between two nodes into account allows to estimate the belonging degree more precisely. However, we didn't confirm this assumption experimentally.

To reduce the complexity of finding all paths between two arbitrary nodes, we tried to re-use one of existing techniques [27-29]. Finally, the APAC algorithm [29] has been chosen. This algorithm does not need to keep track of all visited vertices and only stores the feasible paths.

For domain specific areas where categories are excessively segmentalized, the BVG method cannot extract accurately concept vectors due to the increase in hopcount. To solve this problem, the **Single Parent Integration (SPI) method** is proposed. The authors confirmed from their experiences that a part in the category system which corresponds to (excessively segmentalized) categories for a domain specific area forms almost a tree structure. Based on this fact, when a concept or a category has only one (onehop/multihop) belonging link, the SPI method shortens the belonging link. This is based on the idea that the characteristic is not dispersed even when parent categories are tracked back if the concept or category has only one (onehop or multihop) belonging link.

In the SPI method, if there is only one belonging link ek from node vi (or wi) to v∈V, the path length of ek is accounted as 0, which results in reformation of E to E', and then the BVG method is applied to G'={W, V,E'}.

VVG (Variance-based Vector Generation method) considers the weight of each category link. This method is based on the idea that the belonging degree from a certain category (concept) to parent categories depends on the number of parent categories, thus the weight of each category link is inversely proportional to the number of parent categories.

Thus, the weight of a category link becomes 1 if the category has only one parent category. That's why the authors argue that the VVG method contains the same feature as the SPI method. Therefore, they didn't combined VVG with SPI. We, on the other hand, tried both BVG + SPI and VVG + SPI combinations and confirmed that VVG + SPI performs slightly better than VVG itself (see Section 7 for details).

In the VVG method, weights are set to all belonging links, and the belonging degree from concept wi to category vj is calculated according to the weights. When the number of belonging links from node vi (or wi) to category v∈V is n, weight bek of each of the belonging links ek is defined as follows:

$$bek=1n \quad (2)$$

Then, given all paths P={p1,p2,...,pn} from wi to vj, belonging degree I(wi,vj) from concept wi to category vj is defined as follows:

$$Iwi,vj=p\in Pc(pl) \quad (3)$$

c(pl) is the weight of path pl, calculated by the following equation:

$$cpl=eh\in Elbeh \quad (4)$$

Here, El={e1,e2,...,em} denotes a set of all belonging links forming path pl and eh denotes a belonging link.

After the vector is built and sorted in descending order of belonging degree, it's time to apply cutting technique to it and get the list of IDs *most relevant* to the base category. We've tried out two approaches:

1. *belonging degree threshold* – concepts with belonging degrees less than the mean value of belonging degree for each vector are filtered;
2. *percent threshold* - 25% of concepts with the lowest belonging degrees are filtered.

Finally, Topic Deriver produces *domain concepts file* that contains IDs of derived concepts.

4.3. Reducer

Reducer is the final part of the system. It takes domain concepts file as input, applies the concepts' list to complete Texterra knowledge base, and produces the reduced domain specific version of the latter. It contains not only concept IDs, but also full information about each of them, including the part of category structure that covers selected concepts. Therefore, the domain specific version is consistent and ready for loading into Texterra.

5. An approach to storing Wikipedia category graph

As showed above, fast access to any point of Wikipedia category structure is necessary for efficiency of all described computations. In particular, VVG method requires both entire WCG and subgraph of current base category to be available simultaneously.

Chernov et al. [15] studied semantic relationships between Wikipedia categories. They exported the dataset of about 670 thousands pages into a MySQL database. The data size was ~1.2 Gb. But, like many other researchers, they picked just a small sample of pages for processing (few thousands). For such small-scale approaches, even a usual on-disk relational DB is fast enough.

But our goal was to create a technique for fast iterative traversing through even a top-level categories with millions pages. Thus, we resorted to in-memory storage of WCG.

5.1. Evaluation results for known graph libraries

We've tried out two third-party libraries for storing the WCG in the JVM's memory. First of them, JUNG [30], showed satisfying performance results on small-scale subgraphs. But the entire WCG was impetuously expanding while loading and didn't fit in the RAM of the test machine (8 Gb). Second one, JGraphT [31], demonstrated almost the same behaviour: the WCG consumed a bit less amount of memory, but still too much. These observations hinder to utilize these libraries as a solution for WCG storing.

But there are a number of other libraries for graph storage which provide handy interface to stored data. We've found **neo4j** [32] and **WebGraph** [33] libraries. They may appear useful during the further research.

5.2. WikiGraph

The common shortcoming of all Java graph implementations we've tested seemed to be the redundancy of data stored in the RAM. Thus, the right way is to change the data storage manner. We put this into practice in WikiGraph.

All the prominent features of WikiGraph are due to the fact that it's *intended* to store WCG:

1. It is *directed* (as category links have a direction);
2. It introduces the notions of *category* and *article* and provides a powerful tooling to store and maintain the data on affiliations between them;
3. Only *IDs and types of pages* are stored. Each vertex is presented as a map consisting of [*ID, isCategory*] entries. This allows to store page type as a Boolean variable (*TRUE* is for category, *FALSE* is for article). All page data are saved as primitive variables, not an objects;
4. *Incidence list* has been chosen as a main data structure (along with vertices and edges lists). This is particularly important as the WCG is quite *dense*: $\frac{\text{number of edges}}{\text{number of vertices}} \approx 41$. The incidence list is organized into a set of [*vertex*, [*list of incident edges*]] entries. Each list is sorted in ascending order of edges IDs just after the loading. This avoids the need to look over the entire incidence list to get all the edges incident to an arbitrary vertex. Moreover, due to sorting of the lists, it's allowed to interrupt the search over them after the edge with greatest expected ID is found;
5. All *kinks* (self-to-self links) are removed;
6. *Initial capacity* of the incidence list is beforehand set to approximate amount of vertices in WCG (3,500,000 for this case). This saves some memory allocation costs while loading;

7. A set of *helper methods* is developed also (for instance, a method for deriving a subgraph of a given base category). This set provides usable and fast interface to the WCG data.

After the described features were implemented, they allowed us to fit WCG entirely in the RAM (~4 Gb needed) and lead to significant speed-up of loading and processing.

6. Evaluation methodology

Obviously, a domain specific subset of Wikipedia should have a good coverage of domain knowledge. But there is no easy direct way to evaluate quality of such a subset. The reason is that we must evaluate the completeness of knowledge available from Wikipedia's articles in resulting subset compared to that of specific domain. It's clear that this is rather difficult. In addition, the quality of link structure in the resulting subset should be also evaluated.

Therefore, we applied so called *in vivo* approach for evaluation. To estimate the quality of proposed methods, we studied how applying of the extracted subsets affects the performance of Texterra as a whole.

As mentioned before, one of Texterra parts is the system that enriches ID3 tags for musical recordings with links to corresponding articles of Wikipedia. This system utilizes graph structure to compute semantic relatedness between Wikipedia pages [13]. Then, semantic relatedness is exploited by word sense disambiguation algorithm. The latter is intended to choose the most relevant Wikipedia page from several homonymic variants.

We assume that each page of Wikipedia describes one possible meaning. WSD algorithm selects the most consistent combination of meanings that correspond to input ID3 tags. For sequence of input tags, it computes similarity between all pairs of meanings. The weight of a sequence is a sum of weights of all its pairs. Then, the algorithm detects a sequence with greatest weight.

To show good results, the derived subset should include as much as possible Wikipedia articles associated with a specific domain. In additional, link structure should be good enough for relatedness computation. Therefore, this approach allows evaluating both the quality of dictionary content and the quality of link structure.

For testing purpose, we derived several music-related subsets of Wikipedia by running different combinations of heuristics and used these subsets for described system. Then, we consequentially loaded these domain specific versions of

knowledge base into Texterra and ran the tests. We used a small corpus of 20 random musical compositions and 49 different tags. Then, we estimated the precision of automated disambiguation by comparing the results of algorithm with manually disambiguated tags.

7. Experimental results

The configuration of test workstation was as follows: Intel Core 2 Duo CPU (3.16 GHz), 8 Gb RAM, Windows 7 Enterprise 64 bit, Java SE 6 Development Kit 1.6.0.20.

Sample vectors for different combinations of heuristics are provided in Tables 1-4. Each sample vector comprises three concepts with highest belonging degree and three concepts with lowest values. The base category is *Category:Musical compositions*. It's noteworthy that BVG vector differs significantly from VVG one. Furthermore, enabling SPI affects both vectors.

Table 1. Sample vector produced by the BVG method

ID	Title	Belonging degree
1784928	Candle in the Wind 1997	0.54
1523941	Axel F	0.50
1728643	Jeremy (song)	0.49
3720518	Ludwig Streicher	$2.44 * 10^{-4}$
2175948	Ian Bousfield	$2.44 * 10^{-4}$
875344	Willi Boskovsky	$2.44 * 10^{-4}$

Table 2. Sample vector produced by the VVG method

ID	Title	Belonging degree
27684606	Niagara Falls Suite	0.50
20053503	Kumikyoku Nico Nico Douga	0.50
2501716	Megamix	0.50
14054430	Oh, by the Way	$6.02 * 10^{-7}$
454136	A Collection of Great Dance Songs	$6.02 * 10^{-7}$
361654	Echoes: The Best of Pink Floyd	$6.02 * 10^{-7}$

Table 3. Sample vector produced by the BVG method with SPI enabled

ID	Title	Belonging degree
1523941	Axel F	5.53
1815726	I Will Always Love You	3.88
923235	My Heart Will Go On	3.50
9010	Dance Dance Revolution	0.00
4527	Béla Bartók	0.00
1370	Ambrose	0.00

Table 4. Sample vector produced by the VVG method with SPI enabled

ID	Title	Belonging degree
27684606	Niagara Falls Suite	0.50
20053503	Kumikyoku Nico Nico Douga	0.50
2501716	Megamix	0.50
9010	Dance Dance Revolution	0.00
4527	Béla Bartók	0.00
1370	Ambrose	0.00

The results of the experiments with different combinations of vector generation methods and cutting techniques are presented in Table 5. Contents of all 3 base categories listed in Section 4.2 are included.

Table 5.

Experimental results

	SPI enabled	Threshold	Number of IDs	Size, Mb	Precision, %
Basic Vector Generation	yes	belonging degree	334,575	112,3	45,10
		percent	574,810	251,2	**62,75**
	no	belonging degree	434,229	159,2	45,10
		percent	574,940	251,8	**64,71**
Variance-based Vector Generation	yes	belonging degree	420,302	154,8	49,02
		percent	549,791	244,5	60,78
	no	belonging degree	418,998	150,1	41,18
		percent	549,639	245,5	56,86
No threshold	—	—	675,228	311,5	72,55
Ground truth	—	—	8,476,942	4528,3	98,04

Ground truth row corresponds to original Texterra knowledge base. As can be seen, it is huge, but ensures the best accuracy of disambiguation.

No threshold is for case when no cutting technique is applied to the concept vectors. In other words, this set of IDs exactly matches the set of all articles from subgraphs of all base categories. This version is much smaller, but the precision gets lower also. This precision drop (when no threshold is applied yet) is only due to imperfect choice of base categories. They merely don't cover all concepts required for precise disambiguation. It's also obvious that all comparisons of heuristics results should be done with no threshold results, not with ground truth.

As one can see, BVG performs a bit better than VVG. The most accurate combinations of heuristics are *BVG + percent threshold* and *BVG + SPI + percent threshold*. Enabling SPI for VVG slightly increases the precision of disambiguation. Percent threshold is definitely better than belonging degree threshold.

What's important here is that the size of Texterra knowledge base (both on disk and in RAM) depends linearly on the number of concepts. Thus, the challenge is to find a compromise between the precision of disambiguation and the size (and contents) of the knowledge base.

The conducted experiment was just our first effort of this kind. Implemented algorithms allowed us to reduce the size of Texterra knowledge base by more than order, while the precision of disambiguating musical metadata has decreased from 98% to 64%. We believe that these results prove the applicability of proposed approach for deriving domain specific subset of Wikipedia. Certainly, there're still many things to improve.

8. Conclusion and future work

According to the results of this study, we outline the following:

- Wikipedia categories network may be utilized for domain specific subset of Wikipedia;
- Using concept vectors seems to be appropriate way to represent the affiliations of belonging between Wikipedia pages;
- BVG performs a bit better than VVG;
- SPI often improves the results;
- Percent threshold showed the best results as a cutting technique.

Possible directions of future work include:

1. As noted by the authors of [18], the selection of the root node is vital to the quality of the domain specific corpus. Thus, it’s reasonable to introduce some heuristics for automated identifying of the most appropriate base category given just a set of specific keywords. Moreover, there can be several base categories with either manually or automatically set *relevance levels*. For example, to perform the search for "*Musicians of World War II*" a user should provide 2 base categories as an input: *Category:Musicians* with relevance level of 0.9 and *Category:World_War_II* with relevance level of 0.5;
2. Try other cutting techniques for concept vectors (i.e., attempt to detect the *distribution* of belonging degrees and utilize it);

3. Detect and remove meaningless pages from the WCG (i.e., pages from administrative section of Wikipedia [4]);

4. Distinguishing between classes and instances among categories [9] may help to prune and/or reorganize the WCG;

5. Add a facility for storing the results of semantic relatedness computation to boost the further processing;

6. Develop the approximation algorithm for finding all paths between two arbitrary nodes in the WCG.

In this work, we've demonstrated the possible benefits of automated building of the domain specific ontologies. Also, we've tested different heuristics while implementing the system for such processing. An original approach to storing WCG in the RAM has been proposed, along with specific evaluation methodology.

The described approach can be applied to any ontology with well-defined polyhierarchical taxonomy (for instance, YAGO2). As it seems to us, weighting the existing semantic connections is always a challenging task while building any more or less large ontology. This may be helpful for any domain dependent Wikipedia-related research [34, 35].

9. Acknowledgement

This research was collaborated with and supported by the Samsung Electronics Co., Ltd. DMC R&D Center Convergence Solution Team.

References

[1] *List of Wikipedias - Meta*. http://meta.wikimedia.org/wiki/List_of_Wikipedias
[2] M. Shirakawa, K. Nakayama, T. Hara, S. Nishio. *Concept Vector Extraction from Wikipedia Category Network*. In Proceedings of 3rd International Conference on Ubiquitous Information Management and Communication (ICUIMC 2009), pp. 71-79, 2009.
[3] S. Auer, C. Bizer, G. Kobilarov, J. Lehmann, R. Cyganiak, Z. G. Ives. *Dbpedia: A nucleus for a web of open data*. In ISWC, volume 4825 of LNCS, pages 722–735. Springer, 2007.

[4] Simone P. Ponzetto, Michael Strube. *Deriving a large scale taxonomy from Wikipedia*. In AAAI'07: Proceedings of the 22nd national conference on Artificial intelligence, pp. 1440-1445, 2007.
[5] Fabian M. Suchanek, Gjergji Kasneci, Gerhard Weikum. *YAGO: A Large Ontology from Wikipedia and WordNet*. In Elsevier Journal of Web Semantics, Vol. 6, No. 3, pp. 203-217, 2008.
[6] *Unified Medical Language System (UMLS) - Home*. http://www.nlm.nih.gov/research/umls/
[7] P. Buitelaar, P. Cimiano, B. Magnini (Eds.). *Ontology Learning from Text: Methods, Evaluation and Applications*. In Frontiers in Artificial Intelligence and Applications Series, Vol. 123, IOS Press, July 2005.
[8] A. Gregorowicz, M. A. Kramer. *Mining a Large-Scale Term-Concept Network from Wikipedia*. Technical Report #06-1028, The MITRE Corp., Oct. 2006.
[9] Cäcilia Zirn, Vivi Nastase, Michael Strube. *Distinguishing between instances and classes in the Wikipedia taxonomy*. In Proc. of ESWC-08, pages 376-387, 2008.
[10] Gaoying Cui, Qin Lu, Wenjie Li, Yi-Rong Chen. *Mining Concepts from Wikipedia for Ontology Construction*. In Proceedings of Web Intelligence/IAT Workshops, pp.287-290, 2009.
[11] J. Hoffart, F. Suchanek, K. Berberich, G. Weikum. *YAGO2: A Spatially and Temporally Enhanced Knowledge Base from Wikipedia*. Research Report MPI-I-2010-5-007, Max-Planck-Institut für Informatik, November 2010.
[12] A. Budanitsky, G. Hirst. *Evaluating WordNet-based measures of semantic distance*. In Computational Linguistics, 32(1), pp. 13-47, March 2006.
[13] D. Turdakov, P. Velikhov. *Semantic Relatedness Metric for Wikipedia Concepts Based on Link Analysis and its Application to Word Sense Disambiguation*. In Proc. of SYRCoDIS, 2008.
[14] T. Zesch, I. Gurevych. *Analysis of the Wikipedia Category Graph for NLP Applications*. In Proceedings of the TextGraphs-2 Workshop (NAACL-HLT), 2007.
[15] S. Chernov, T. Iofciu, W. Nejdl, X. Zhou. *Extracting Semantic Relationships between Wikipedia Categories*. In Proceedings of the First International Workshop on Semantic Wikis - From Wiki To Semantics, June 2006.
[16] M. Strube, S. P. Ponzetto. *WikiRelate! Computing semantic relatedness using Wikipedia*. In Proceedings of the 21st national conference on Artificial intelligence (AAAI'06), pp. 1419-1424, 2006.
[17] Z. Syed, T. Finin, and A. Joshi. *Wikipedia as an Ontology for Describing Documents*. In Proceedings of the Second International Conference on Weblogs and Social Media, 2008.
[18] G. Y. Cui, Q. Lu, W. J. Li, Y. R. Chen. *Corpus Exploitation from Wikipedia for Ontology Construction*. In LREC 2008, Marrakech, pp. 2125-2132, 2008.
[19] *Wikipedia:Overcategorization - Wikipedia, the free encyclopedia*. http://en.wikipedia.org/wiki/Wikipedia:Overcategorization
[20] *Wikipedia:Categorization - Wikipedia, the free encyclopedia*. http://en.wikipedia.org/wiki/Wikipedia:Categorization
[21] *Catgraph*. http://toolserver.org/~dapete/catgraph/
[22] *Wikipedia:WikiProject Categories/uncategorized - Wikipedia, the free encyclopedia*. http://en.wikipedia.org/wiki/Wikipedia:WikiProject_Categories/uncategorized

[23] *Wikipedia:Database reports/Uncategorized categories - Wikipedia, the free encyclopedia*. http://en.wikipedia.org/wiki/Wikipedia:Database_reports/Uncategorized_categories
[24] *Category:Better category needed - Wikipedia, the free encyclopedia*. http://en.wikipedia.org/wiki/Category:Better_category_needed
[25] J. Soto. *Wikipedia: A Quantitative Analysis*. PhD thesis, 2009.
[26] T. H. Cormen, C. E. Leiserson, R. L. Rivest, C. Stein. *Introduction to Algorithms, Second Edition*. MIT Press and McGraw-Hill, 2001. Section 22.3: Depth-first search, pp. 540–549.
[27] L.-E. *Thorelli*. *An algorithm for computing all paths in a graph*. In BIT 6, 347—349, 1966.
[28] M. Migliore , V. Martorana , F. Sciortino. *An algorithm to find all paths between two nodes in a graph*. In Journal of Computational Physics, v.87 n.1, pp.231-236, March 1990.
[29] R. Simoes. *APAC: An exact algorithm for retrieving cycles and paths in all kinds of graphs*. In Tékhne, no.12, p.39-55, 2009.
[30] *JUNG - Java Universal Network/Graph Framework*. http://jung.sourceforge.net/
[31] *JGraphT - a free Java graph library*. http://www.jgrapht.org/
[32] *neo4j open source nosql graph database*. http://neo4j.org/
[33] *WebGraph*. http://webgraph.dsi.unimi.it/
[34] *Wikipedia:Academic studies of Wikipedia - Wikipedia, the free encyclopedia*. http://en.wikipedia.org/wiki/Wikipedia:Academic_studies_of_Wikipedia
[35] *Academic studies about Wikipedia - Wikipedia, the free encyclopedia*. http://en.wikipedia.org/wiki/Academic_studies_about_Wikipedia#Natural_language_processing