Wikipedia Revision Toolkit: Efficiently Accessing Wikipedia's Edit History

Oliver Ferschke, Torsten Zesch, and Iryna Gurevych
Ubiquitous Knowledge Processing Lab
Computer Science Department, Technische Universität Darmstadt
Hochschulstrasse 10, D-64289 Darmstadt, Germany
http://www.ukp.tu-darmstadt.de

Abstract

We present an open-source toolkit which allows (i) to reconstruct past states of Wikipedia, and (ii) to efficiently access the edit history of Wikipedia articles. Reconstructing past states of Wikipedia is a prerequisite for reproducing previous experimental work based on Wikipedia. Beyond that, the edit history of Wikipedia articles has been shown to be a valuable knowledge source for NLP, but access is severely impeded by the lack of efficient tools for managing the huge amount of provided data. By using a dedicated storage format, our toolkit massively decreases the data volume to less than 2% of the original size, and at the same time provides an easy-to-use interface to access the revision data. The language-independent design allows to process any language represented in Wikipedia. We expect this work to consolidate NLP research using Wikipedia in general, and to foster research making use of the knowledge encoded in Wikipedia's edit history.

1 Introduction

In the last decade, the free encyclopedia Wikipedia has become one of the most valuable and comprehensive knowledge sources in Natural Language Processing. It has been used for numerous NLP tasks, e.g. word sense disambiguation, semantic relatedness measures, or text categorization. A detailed survey on usages of Wikipedia in NLP can be found in (Medelyan et al., 2009).

The majority of Wikipedia-based NLP algorithms works on single snapshots of Wikipedia, which are published by the Wikimedia Foundation as XML dumps at irregular intervals.[1] Such a snapshot only represents the state of Wikipedia at a certain fixed point in time, while Wikipedia actually is a dynamic resource that is constantly changed by its millions of editors. This rapid change is bound to have an influence on the performance of NLP algorithms using Wikipedia data. However, the exact consequences are largely unknown, as only very few papers have systematically analyzed this influence (Zesch and Gurevych, 2010). This is mainly due to older snapshots becoming unavailable, as there is no official backup server. As a consequence, older experimental results cannot be reproduced anymore.

In this paper, we present a toolkit that solves both issues by reconstructing a certain past state of Wikipedia from its edit history, which is offered by the Wikimedia Foundation in form of a database dump. Section 3 gives a more detailed overview of the reconstruction process.

Besides reconstructing past states of Wikipedia, the revision history data also constitutes a novel knowledge source for NLP algorithms. The sequence of article edits can be used as training data for data-driven NLP algorithms, such as vandalism detection (Chin et al., 2010), text summarization (Nelken and Yamangil, 2008), sentence compression (Yamangil and Nelken, 2008), unsupervised extraction of lexical simplifications (Yatskar et al., 2010), the expansion of textual entailment corpora (Zanzotto and Pennacchiotti, 2010), or assesing the trustworthiness of Wikipedia articles (Zeng et al., 2006).

[1] http://download.wikimedia.org/

Proceedings of the ACL-HLT 2011 System Demonstrations, pages 97–102,
Portland, Oregon, USA, 21 June 2011.

However, efficient access to this new resource has been limited by the immense size of the data. The revisions for all articles in the current English Wikipedia sum up to over 5 terabytes of text. Consequently, most of the above mentioned previous work only regarded small samples of the available data. However, using more data usually leads to better results, or how Church and Mercer (1993) put it "more data are better data". Thus, in Section 4, we present a tool to efficiently access Wikipedia's edit history. It provides an easy-to-use API for programmatically accessing the revision data and reduces the required storage space to less than 2% of its original size. Both tools are publicly available on Google Code (`http://jwpl.googlecode.com`) as open source software under the *LGPL v3*.

2 Related Work

To our knowledge, there are currently only two alternatives to programmatically access Wikipedia's revision history.

One possibility is to manually parse the original XML revision dump. However, due to the huge size of these dumps, efficient, random access is infeasible with this approach.

Another possibility is using the *MediaWiki API*[2], a web service which directly accesses live data from the Wikipedia website. However, using a web service entails that the desired revision for every single article has to be requested from the service, transferred over the Internet and then stored locally in an appropriate format. Access to all revisions of all Wikipedia articles for a large-scale analysis is infeasible with this method because it is strongly constricted by the data transfer speed over the Internet. Even though it is possible to bypass this bottleneck by setting up a local Wikipedia mirror, the MediaWiki API can only provide full text revisions, which results in very large amounts of data to be transferred.

Better suited for tasks of this kind are APIs that utilize databases for storing and accessing the Wikipedia data. However, current database-driven Wikipedia APIs do not support access to article revisions. That is why we decided to extend an established API with the ability to efficiently access Wikipedia's edit history. Two established Wikipedia APIs have been considered for this purpose.

Wikipedia Miner[3] (Milne and Witten, 2009) is an open source toolkit which provides access to Wikipedia with the help of a preprocessed database. It represents articles, categories and redirects as Java classes and provides access to the article content either as MediaWiki markup or as plain text. The toolkit mainly focuses on Wikipedia's structure, the contained concepts, and semantic relations, but it makes little use of the textual content within the articles. Even though it was developed to work language independently, it focuses mainly on the English Wikipedia.

Another open source API for accessing Wikipedia data from a preprocessed database is *JWPL*[4] (Zesch et al., 2008). Like Wikipedia Miner, it also represents the content and structure of Wikipedia as Java objects. In addition to that, JWPL contains a MediaWiki markup parser to further analyze the article contents to make available fine-grained information like e.g. article sections, info-boxes, or first paragraphs. Furthermore, it was explicitly designed to work with all language versions of Wikipedia.

We have chosen to extend JWPL with our revision toolkit, as it has better support for accessing article contents, natively supports multiple languages, and seems to have a larger and more active developer community. In the following section, we present the parts of the toolkit which reconstruct past states of Wikipedia, while in section 4, we describe tools allowing to efficiently access Wikipedia's edit history.

3 Reconstructing Past States of Wikipedia

Access to arbitrary past states of Wikipedia is required to (i) evaluate the performance of Wikipedia-based NLP algorithms over time, and (ii) to reproduce Wikipedia-based research results. For this reason, we have developed a tool called *TimeMachine*, which addresses both of these issues by making use of the revision dump provided by the Wikimedia Foundation. By iterating over all articles in the revision dump and extracting the desired revision of each article, it is possible to recover the state of Wikipedia at an earlier point in time.

[2] `http://www.mediawiki.org/wiki/API`

[3] `http://wikipedia-miner.sourceforge.net`

[4] `http://jwpl.googlecode.com`

Property	Description	Example Value
language	The Wikipedia language version	`english`
mainCategory	Title of the main category of the Wikipedia language version used	`Categories`
disambiguationCategory	Title of the disambiguation category of the Wikipedia language version used	`Disambiguation`
fromTimestamp	Timestamp of the first snapshot to be extracted	`20090101130000`
toTimestamp	Timestamp of the last snapshot to be extracted	`20091231130000`
each	Interval between snapshots in days	`30`
removeInputFilesAfterProcessing	Remove source files [true/false]	`false`
metaHistoryFile	Path to the revision dump	`PATH/pages-meta-history.xml.bz2`
pageLinksFile	Path to the page-to-page link records	`PATH/pagelinks.sql.gz`
categoryLinksFile	Path to the category membership records	`PATH/categorylinks.sql.gz`
outputDirectory	Output directory	`PATH/outdir/`

Table 1: Configuration of the TimeMachine

The TimeMachine is controlled by a single configuration file, which allows (i) to restore individual Wikipedia snapshots or (ii) to generate whole snapshot series. *Table 1* gives an overview of the configuration parameters. The first three properties set the environment for the specific language version of Wikipedia. The two timestamps define the start and end time of the snapshot series, while the interval between the snapshots in the series is set by the parameter *each*. In the example, the TimeMachine recovers 13 snapshots between Jan 01, 2009 at 01.00 p.m and and Dec 31, 2009 at 01.00 p.m at an interval of 30 days. In order to recover a single snapshot, the two timestamps have simply to be set to the same value, while the parameter '*each*' has no effect. The option *removeInputFilesAfterProcessing* specifies whether to delete the source files after processing has finished. The final four properties define the paths to the source files and the output directory.

The output of the TimeMachine is a set of eleven text files for each snapshot, which can directly be imported into an empty JWPL database. It can be accessed with the JWPL API in the same way as snapshots created using JWPL itself.

Issue of Deleted Articles The past snapshot of Wikipedia created by our toolkit is identical to the state of Wikipedia at that time with the exception of articles that have been deleted meanwhile. Articles might be deleted only by Wikipedia administrators if they are subject to copyright violations, vandalism, spam or other conditions that violate Wikipedia policies. As a consequence, they are removed from the public view along with all their revision information, which makes it impossible to recover them from any future publicly available dump.[5] Even though about five thousand pages are deleted every day, only a small percentage of those pages actually corresponds to meaningful articles. Most of the affected pages are newly created duplicates of already existing articles or spam articles.

4 Efficient Access to Revisions

Even though article revisions are available from the official Wikipedia revision dumps, accessing this information on a large scale is still a difficult task. This is due to two main problems. First, the revision dump contains all revisions as full text. This results in a massive amount of data and makes structured access very hard. Second, there is no efficient API available so far for accessing article revisions on a large scale.

Thus, we have developed a tool called *RevisionMachine*, which solves these issues. First, we describe our solution to the storage problem. Second, we present several use cases of the RevisionMachine, and show how the API simplifies experimental setups.

[5] `http://en.wikipedia.org/wiki/Wikipedia:DEL`

4.1 Revision Storage

As each revision of a Wikipedia article stores the full article text, the revision history obviously contains a lot of redundant data. The RevisionMachine makes use of this fact and utilizes a dedicated storage format which stores a revision only by means of the changes that have been made to the previous revision. For this purpose, we have tested existing diff libraries, like Javaxdelta[6] or java-diff[7], which calculate the differences between two texts. However, both their runtime and the size of the resulting output was not feasible for the given size of the data. Therefore, we have developed our own diff algorithm, which is based on a *longest common substring search* and constitutes the foundation for our revision storage format.

The processing of two subsequent revisions can be divided into four steps:

- First, the RevisionMachine searches for all common substrings with a user-defined minimal length.
- Then, the revisions are divided into blocks of equal length. Corresponding blocks of both revisions are then compared. If a block is contained in one of the common substrings, it can be marked as *unchanged*. Otherwise, we have to categorize the kind of change that occurred in this block. We differentiate between five possible actions: `Insert`, `Delete`, `Replace`, `Cut` and `Paste`[8]. This information is stored in each block and is later on used to encode the revision.
- In the next step, the current revision is represented by means of a sequence of actions performed on the previous revision.
  For example, in the adjacent revision pair
  r_1 : This is the very first sentence!
  r_2 : This is the second sentence
  r_2 can be encoded as
  `REPLACE 12 10 'second'`
  `DELETE 31 1`
- Finally, the string representation of this action sequence is compressed and stored in the database.

With this approach, we achieve to reduce the demand for disk space for a recent English Wikipedia dump containing all article revisions from 5470 GB to only 96 GB, i.e. by 98%. The compressed data is stored in a MySQL database, which provides sophisticated indexing mechanisms for high-performance access to the data.

Obviously, storing only the changes instead of the full text of each revision trades in speed for space. Accessing a certain revision now requires reconstructing the text of the revision from a list of changes. As articles often have several thousand revisions, this might take too long. Thus, in order to speed up the recovery of the revision text, every n-th revision is stored as a full revision. A low value of n decreases the time needed to access a certain revision, but increases the demand for storage space. We have found $n = 1000$ to yield a good trade-off[9]. This parameter, among a few other possibilities to fine-tune the process, can be set in a graphical user interface provided with the RevisionMachine.

4.2 Revision Access

After the converted revisions have been stored in the revision database, it can either be used standalone or combined with the JWPL data and accessed via the standard JWPL API. The latter option makes it possible to combine the possibilities of the RevisionMachine with other components like the JWPL parser for the MediaWiki syntax.

In order to set up the RevisionMachine, it is only necessary to provide the configuration details for the database connection (see *Listing 1*). Upon first access, the database user has to have write permission on the database, as indexes have to be created. For later use, read permission is sufficient. Access to the RevisionMachine is achieved via two API objects. The *RevisionIterator* allows to iterate over all revisions in Wikipedia. The *RevisionAPI* grants access to the revisions of individual articles. In addition to

[6] `http://javaxdelta.sourceforge.net/`

[7] `http://www.incava.org/projects/java/java-diff`

[8] `Cut` and `Paste` operations always occur pairwise. In addition to the other operations, they can make use of an additional temporary storage register to save the text that is being moved.

[9] If hard disk space is no limiting factor, the parameter can be set to 1 to avoid the compression of the revisions and maximize the performance.

```
//Set up database connection
DatabaseConfiguration db = new DatabaseConfiguration();
db.setDatabase("dbname");
db.setHost("hostname");
db.setUser("username");
db.setPassword("pwd");
db.setLanguage(Language.english);
//Create API objects
Wikipedia wiki = WikiConnectionUtils.getWikipediaConnection(db);
RevisionIterator revIt = new RevisionIterator(db);
RevisionApi revApi = new RevisionApi(db);
```

Listing 1: Setting up the RevisionMachine

that, the *Wikipedia* object provides access to JWPL functionalities.

In the following, we describe three use cases of the RevisionMachine API, which demonstrate how it is easily integrated into experimental setups.

Processing all article revisions in Wikipedia The first use case focuses on the utilization of the complete set of article revisions in a Wikipedia snapshot. *Listing 2* shows how to iterate over all revisions. Thereby, the iterator ensures that successive revisions always correspond to adjacent revisions of a single article in chronological order. The start of a new article can easily be detected by checking the timestamp and the article id. This approach is especially useful for applications in statistical natural language processing, where large amounts of training data are a vital asset.

Processing revisions of individual articles The second use case shows how the RevisionMachine can be used to access the edit history of a specific article. The example in *Listing 3* illustrates how all revisions for the article *Automobile* can be retrieved by first performing a page query with the JWPL API and then retrieving all revision timestamps for this page, which can finally be used to access the revision objects.

Accessing the meta data of a revision The third use case illustrates the access to the meta data of individual revisions. The meta data includes the name or IP of the contributor, the additional user comment for the revision and a flag that identifies a revision as minor or major. *Listing 4* shows how the number of edits and unique contributors can be used to indicate the level of edit activity for an article.

5 Conclusions

In this paper, we presented an open-source toolkit which extends *JWPL*, an API for accessing Wikipedia, with the ability to reconstruct past states of Wikipedia, and to efficiently access the edit history of Wikipedia articles.

Reconstructing past states of Wikipedia is a prerequisite for reproducing previous experimental work based on Wikipedia, and is also a requirement for the creation of time-based series of Wikipedia snapshots and for assessing the influence of Wikipedia growth on NLP algorithms. Furthermore, Wikipedia's edit history has been shown to be a valuable knowledge source for NLP, which is hard to access because of the lack of efficient tools for managing the huge amount of revision data. By utilizing a dedicated storage format for the revisions, our toolkit massively decreases the amount of data to be stored. At the same time, it provides an easy-to-use interface to access the revision data.

We expect this work to consolidate NLP research using Wikipedia in general, and to foster research making use of the knowledge encoded in Wikipedia's edit history. The toolkit will be made available as part of *JWPL*, and can be obtained from the project's website at Google Code. (`http://jwpl.googlecode.com`)

Acknowledgments

This work has been supported by the Volkswagen Foundation as part of the Lichtenberg-Professorship Program under grant No. I/82806, and by the Hessian research excellence program "Landes-Offensive zur Entwicklung Wissenschaftlich-ökonomischer Exzellenz" (*LOEWE*) as part of the research center "Digital Humanities". We would also like to thank Simon Kulessa for designing and implementing the foundations of the RevisionMachine.

```
//Iterate over all revisions of all articles
while (revIt.hasNext()) {
    Revision rev = revIt.next()
    rev.getTimestamp();
    rev.getArticleID();
    //process revision ...
}
```

Listing 2: Iteration over all revisions of all articles

```
//Get article with title "Automobile"
Page article = wiki.getPage("Automobile");
int id = article.getPageId();
//Get all revisions for the article
Collection<Timestamp> revisionTimeStamps = revApi.getRevisionTimestamps(id);
for (Timestamp t:revisionTimeStamps) {
    Revision rev = revApi.getRevision(id, t);
    //process revision ...
}
```

Listing 3: Accessing the revisions of a specific article

```
//Meta data provided by the RevisionAPI
StringBuffer s = new StringBuffer();
s.append("The article has "+revApi.getNumberOfRevisions(pageId)+" revisions.\n");
s.append("It has "+revApi.getNumberOfUniqueContributors(pageId)+" unique contributors.\n");
s.append(revApi.getNumberOfUniqueContributors(pageId,true)+ " are registered users.\n");
//Meta data provided by the Revision object
s.append((rev.isMinor()?"Minor":"Major")+" revision by: "+rev.getContributorID());
s.append("\nComment: "+rev.getComment());
```

Listing 4: Accessing the meta data of a revision

References

Si-Chi Chin, W. Nick Street, Padmini Srinivasan, and David Eichmann. 2010. Detecting wikipedia vandalism with active learning and statistical language models. In *Proceedings of the 4th workshop on Information credibility*, WICOW '10, pages 3–10.

Kenneth W. Church and Robert L. Mercer. 1993. Introduction to the special issue on computational linguistics using large corpora. *Computational Linguistics*, 19(1):1–24.

Olena Medelyan, David Milne, Catherine Legg, and Ian H. Witten. 2009. Mining meaning from wikipedia. *Int. J. Hum.-Comput. Stud.*, 67:716–754, September.

D. Milne and I. H. Witten. 2009. An open-source toolkit for mining Wikipedia. In *Proc. New Zealand Computer Science Research Student Conf.*, volume 9.

Rani Nelken and Elif Yamangil. 2008. Mining wikipedia's article revision history for training computational linguistics algorithms. In *Proceedings of the AAAI Workshop on Wikipedia and Artificial Intelligence: An Evolving Synergy (WikiAI)*, WikiAI08.

Elif Yamangil and Rani Nelken. 2008. Mining wikipedia revision histories for improving sentence compression. In *Proceedings of ACL-08: HLT, Short Papers*, pages 137–140, Columbus, Ohio, June. Association for Computational Linguistics.

Mark Yatskar, Bo Pang, Cristian Danescu-Niculescu-Mizil, and Lillian Lee. 2010. For the sake of simplicity: unsupervised extraction of lexical simplifications from wikipedia. In *Human Language Technologies: The 2010 Annual Conference of the North American Chapter of the Association for Computational Linguistics*, HLT '10, pages 365–368.

Fabio Massimo Zanzotto and Marco Pennacchiotti. 2010. Expanding textual entailment corpora from wikipedia using co-training. In *Proceedings of the COLING-Workshop on The People's Web Meets NLP: Collaboratively Constructed Semantic Resources*.

Honglei Zeng, Maher Alhossaini, Li Ding, Richard Fikes, and Deborah L. McGuinness. 2006. Computing trust from revision history. In *Proceedings of the 2006 International Conference on Privacy, Security and Trust*.

Torsten Zesch and Iryna Gurevych. 2010. The more the better? Assessing the influence of wikipedia's growth on semantic relatedness measures. In *Proceedings of the Conference on Language Resources and Evaluation (LREC)*, Valletta, Malta.

Torsten Zesch, Christof Mueller, and Iryna Gurevych. 2008. Extracting Lexical Semantic Knowledge from Wikipedia and Wiktionary. In *Proceedings of the Conference on Language Resources and Evaluation (LREC)*.