WikiDragon: A Java Framework For Diachronic Content And Network Analysis Of MediaWikis

Rüdiger Gleim[1], **Alexander Mehler**[1], **Sung Y. Song**[2]
Text-Technology Lab, Goethe University[1]
Robert-Mayer-Straße 10, 60325 Frankfurt
{gleim, mehler}@em.uni-frankfurt.de, gnosygnu@gmail.com[2]

Abstract

We introduce WikiDragon, a Java Framework designed to give developers in computational linguistics an intuitive API to build, parse and analyze instances of MediaWikis such as Wikipedia, Wiktionary or WikiSource on their computers. It covers current versions of pages as well as the complete revision history, gives diachronic access to both page source code as well as accurately parsed HTML and supports the diachronic exploration of the page network. WikiDragon is self-enclosed and only requires an XML dump of the official Wikimedia Foundation website for import into an embedded database. No additional setup is required. We describe WikiDragon's architecture and evaluate the framework based on the simple English Wikipedia with respect to the accuracy of link extraction, diachronic network analysis and the impact of using different Wikipedia frameworks to text analysis.

Keywords: Wikipedia, Java Framework, Diachronic Text Analysis, Diachronic Network Analysis, Link Extraction

1. Introduction

Wikis such as Wikipedia, Wiktionary, WikiSource or WikiNews constitute a popular form of collaborative writing and thus a valuable resource for computational linguistics. This relates to research exploiting wiki data as a resource, e.g. for word sense disambiguation (Uslu et al., 2018; Dandala et al., 2013), explicit semantic analysis (Gabrilovich and Markovitch, 2007) or training data for named entity recognition (Nothman et al., 2008). Furthermore, wikis are also considered as a research object on their own, for example with respect to measuring the impact of authors and edits (Priedhorsky et al., 2007), studies on discourse structure (Mehler et al., 2018) as well as on edit categories (Daxenberger and Gurevych, 2012) or dynamics of conflicts in Wikipedias (Yasseri et al., 2012). The Wiki principle is also used to publish research results like digital editions of texts as Wikiditions (Mehler et al., 2016).

Generally speaking, researchers need efficient as well as manageable means to access wiki data. However, one has to face the challenge that this data is not limited to articles but also includes discussions, pages in other namespaces such as portals, rich metadata, revision histories, information about (anonymous or registered) writers as well as about the link-based networking of pages. For small queries, using the Web API of wikis to collect data may be feasable. But in order to analyse large amounts of data (e.g. all articles of the main namespace) the approach to work online is inadequate because of the intense use of bandwith and server load. The alternative is to work offline based on local reconstructions of wikis. Notice that this scenario induces a big data problem: take the example of the German Wikipedia: as of 2016-02-03 its pages sum up to 23.84 GB of uncompressed data (current versions only). Its history pages sum-up to 2.92 TB of uncompressed data. Building an offline instance of a wiki based on the MediaWiki software requires extensive work to setup the environment, ensuring that all plugins are installed and importing the official XML dumps into a local MySQL database. Alternative approaches to work with wikis offline lack proper interfaces to explore the data or cover only certain aspects: for example by modeling only a subset of all namespaces (e.g. the main namespace and categories) or by developing interfaces to special instances such as Wiktionary.

Wikimedia's XML dumps, which are the only and most complete data sets available contain revision texts in a proprietary markup which needs to be processed to get HTML. This process is a challenging task, since MediaWiki allows for recursively including templates, uses parser functions and even a scripting language (Lua). Ignoring these particularities entails a high risk of information loss. This also holds, for example, for text network analysis, since page links are only provided as part of SQL dumps which, in contrast to the latter XML dumps, only map the latest state of text linkage. Thus, reconstructing past states of a wiki's article network cannot be done by referring to SQL dumps.

WikiDragon aims to solve the challenge of providing (nearly) complete representations of wikis offline *and* deliver a Java API which meets all requirements to import, represent, extract and analyze MediaWikis in a diachronic manner with respect to text, authorship, network structure and metadata. It is self-contained, runs on Windows as well as Linux and does not require any additional setup. Java developers can use WikiDragon to focus on their research question and code extensions fitting their needs.

Section 1.1. gives an overview of related work in this field. Section 2. points out special features of WikiDragon's architecture and sketches the essential qualities of the Core API. Section 3. provides an evaluation by measuring computation time and required space to build a database, by measuring the accuracy of link extraction against a gold standard, by performing an exemplary diachronic network analysis and by examining the impact of variants of markup parsing to subsequent text analysis. Section 4. gives a summary and a prospect of future work.

1.1. Related Work

There is a vast number of alternative parsers to MediaWiki markup [1] and related tools to extract specific information. Simple approaches attempt to convert MediaWiki Markup into plain text or HTML by applying replacement rules such as regular expressions. More sophisticated approaches also try to implement functions and templates. Nonetheless such advanced frameworks still lack supporting the MediaWiki extension Scribuntu which allows the use of the scripting language Lua. This becomes an issue especially in Wiktionary where whole conjugation and declination tables are written in Lua. The *Sweble Wikitext Parser*[2] (Dohrn and Riehle, 2011) is a Java framework to transform MediaWiki markup into a full fledged object model which can then be transformed into plaintext or HTML. Sweble supports template expansion and a selection of parser functions. The *Java Wikipedia API (Bliki engine)*[3] takes it a step further by providing support for Lua. XOWA[4] is an offline reader for Wikimedia wikis such as Wikipedia and Wiktionary, dedicated to accurate rendering of MediaWiki markup. It provides template expansion, parser functions and processing of Lua code. The project consists of the *XOWA App* and the underlying *XOWA Embeddable Parser*. The offline reader is implemented in Java and based on SWT and Mozilla XULRunner for visualization and HTML rendering. SQLite is used as database backend and Apache Lucene for indexing. The Embeddable Parser is based on Java, Lua and JTidy. It provides an API which allows it to be used for MediaWiki source parsing by other applications bringing their own data model and store. WikiDragon uses XOWA Embeddable Parser to provide (nearly) accurate HTML rendering of Wiki pages and to extract network structures induced by page links.

The *Java Wikipedia Library (JWPL)* (Zesch et al., 2008; Ferschke et al., 2011) is a framework which, based on a MySQL database, provides local access to Wikipedia instances. It includes Sweble to provide plain text versions of articles as well as access to the object model. The standard integration of Sweble into JWPL does not include template expansion. Current versions of a Wikipedia can be build based on dumps using the *JWPL DataMachine* and imported into MySQL. The *JWPL Core* is supplemented by the *JWPL RevisionMachine* to get access to the edit history whereas *JWPL TimeMachine* provides means to create automatized snapshots of a Wikipedia in a specific time interval. Both data model and API of JWPL Core focus by design on the Wikipedia and on the Article- and Category Namespace (including discussion). JWPL supports the import of SQL dumps representing article as well as category links and provide access to this network structure by means of the Java API. Past states of contents can be generated via RevisionMachine or TimeMachine. However it is not possible to reconstruct past states of the network structure as SQL dumps only represent the current state of pages.

Java Wiktionary Library (JWKTL) (Zesch et al., 2008) is a related but stand-alone project which uses an embedded BerkeleyDB to create a parsed lexicon based on a Wiktionary XML dump for English, German and Russian.

WikiWho (Flöck and Acosta, 2014) is an algorithm to compute token-level provenance and changes for Wiki revisioned content. The approach thus focuses on the aspect of the authorship of text components at token level. The system can be used via a web service[5]. WikiWho does not expand templates nor does it track links of pages.

DBpedia (Lehmann et al., 2015) is a community project to extract structured information from Wikipedia in different languages on large scale. Parsed information is represented in RDF which can either be downloaded or queried online via SPARQL. Thus the focus is on explicating information which is hidden in uni- or semi-structured text. Even if extraction- and data projects use their own pipelines for the acquisition and processing of raw data, there could be added value in quickly making any past state available offline using WikiDragon. That way, the information growth of a Wikipedia instance could be investigated over time. In this context, the integration of XOWA for the translation of wiki markup into HTML could is also interesting to related projects.

WikiDragon goes beyond the objectives of offline readers for an accurate representation of the current state by including all content and meta data available to current and past states. Furthermore it is unique in the way it combines the design goal of complete coverage with access through a consistently object-oriented Java API, which can be adapted to any database system via interface implementation.

2. WikiDragon Architecture

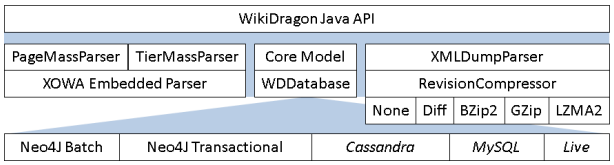

Figure 1: Architecture of WikiDragon depicting the layer model of the system. Italic components are subject of future work.

WikiDragon is designed on the principle to focus on accessing a collection of MediaWiki instances by means of an intuitive Java API while minimizing efforts for setting up the system environment. Figure 1 depicts the general architecture of the framework. The WikiDragon Java API marks the top layer of the architecture. This is the primary interface for users to work on Wikis. Importing, compressing and representing Wiki instances is an integral part of the system: A new Wiki is imported by providing a path to a compressed XML dump file (either current revisions only, or the full history version) to the XMLDumpParser. The import applies multi-threading to speed up the compression of revisions. The most effective compression is achieved

[1] MediaWiki maintains a list of alternative parser on https://www.mediawiki.org/wiki/Alternative_parsers

[2] https://osr.cs.fau.de/software/sweble-wikitext/

[3] https://bitbucket.org/axelclk/info.bliki.wiki/

[4] http://xowa.org/

[5] https://api.wikiwho.net/en/api/

by only storing differences between subsequent page revisions. However, other compression techniques like BZip2, LZMA2 or none at all are also available and may be used when disc space is less an issue. Another important component of WikiDragon is the integration of XOWA for ad hoc or multi-threaded mass parsing of page-revisions or diachronic snapshots called *PageTiers*.

WikiDragon supports multiple database systems by design to be able to provide the best fitting paradigm for a given user environment or task: An embedded database requires no setup, is ideal for a user working on a workstation and yet suitable to host a collection of small or medium-size Wikis or one large instance. Server-based DBMS require additional setup but have the opportunity to detach the experimentation environment from the storage layer. Large scale distributed DBMS like Cassandra are perfect to host multiple large Wikis and still access them by using the same Java API as for the embedded databases. Currently there exist two implementations which use the embeddable graph database management system Neo4J as backend: One variant is based on the batch-inserter interface of Neo4J which is fast, but at the cost of reduced features such as lack of transactions and deletion of elements. The second variant is based on the transactional interface of Neo4J and provides concurrent access which makes it suitable to build a webservice on top of WikiDragon. Note that the integration of Cassandra, MySQL and the Live-mode (based on Web APIs) are subject of future work.

2.1. Process Chain

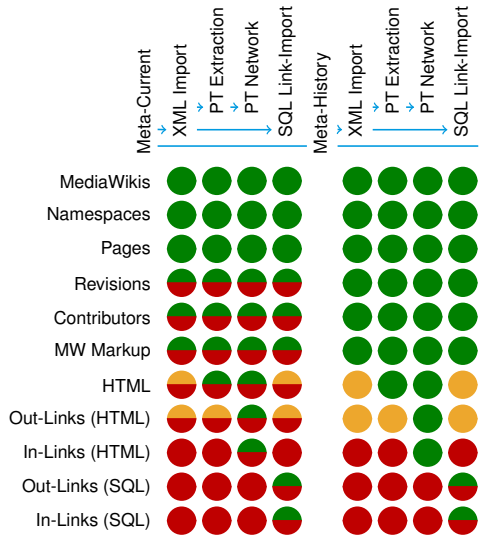

Figure 2: Resource-Feature matrix of WikiDragon for two XML dump variants *meta-current* and *meta-history*. Green denotes direct availability, red unavailability and orange indicates that the information can be parsed ad hoc (not necessarily efficient). Split circles distinguish availability with regard to current (top) or history (bottom) version.

Figure 2 depicts the availability of API features with respect to the XML dump used and the processing steps executed. Using the Meta-Current XML dump is the fastest way to build up a database as it contains only the most recent version of each page. In contrast the Meta-History dump takes considerably more time for import but allows diachronic access. Most features are available directly after the XML import. HTML can be parsed ad hoc (marked yellow in the matrix), which is suitable for small amounts of access. For mass text processing, performing a PageTier (PT) extraction is required which represents a snapshot of a specific point in time (which could also be the most recent state). Outgoing page links are available by ad hoc parsing. For incoming links a PageTier Network needs to be extracted. Finally it is possible to import SQL link dumps from the official Wikimedia sources which represent the current state of linking.[6]

2.2. Core API

The WikiDragon Core Model API (see figure 3) provides classes and methods to browse through collections of MediaWikis, pages, revisions, namespaces and contributors. For example a developer can fetch a page by a specific namespace and title, ask for the latest revision, get the contributor and then ask for all revisions he or she has written. Also accessing HTML and outgoing and incoming links (depending of the kind of processing, c.f section 2.1.) is possible.

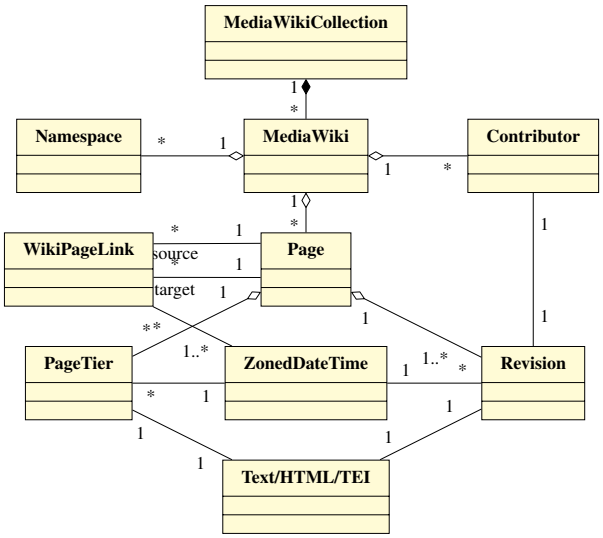

Figure 3: UML diagram depicting the (simplified) Core Model API.

WikiDragon provides two means to access past states of a MediaWiki instance. The first and most straight-forward solution is to fetch the revision of a page which has been active at a specific point in time. This gives access to HTML as well as outgoing links because they are parsed ad hoc. It is possible to perform a complete parsing of all revisions in HTML. Because of the extensive amount of time and space this takes on large MediaWiki instances it is also possible to restrict the parsing to specific pages or revisions by implementing a filter. The second method to access past states

[6]Import links from SQL is performed into whatever database system used- it does not depend on a relational DBMS.

is to create a *PageTier* representing a snapshot at a specific point in time. A snapshot is solely identified by its timestamp and contains all contents parsed in HTML. Optionally this HTML can be used to explicitly compute and store the page network at that time. The advantage is that with this performed it is possible to ask for all all incoming links of a given page at a specific time. In practice this second approach is the best choice when a sequence of snapshots should be examined since only those revisions are parsed and processed which are relevant.

3. Evaluation

In this section we evaluate how WikiDragon performs on typical use cases when working on Wikis. We start by importing the history dump of the simple English Wikipedia into a Neo4J database as backend using non-transactional mode[7]. For our experiments we use uncompressed versions of XML und SQL dumps which are available online[8]. The import is done using up to 8 threads for a combined Diff-BZip2 compression of revisions. This is the basis for all subsequent experiments. As a reference, we also perform an import without any compression. Then we create annual snapshots within the system to get parsed HTML of all pages (including all namespaces) and extract the page networks. Finally we create and evaluate the page network of the latest state of each page against an import of the official SQL dumps representing the current state of page- and category-links as well as redirects.

Task	Time	DB Size	$\|V\|$	$\|E\|$	$\frac{\|GCC\|}{\|V\|}$
Import ref	*00:29:01*	*32.35* GB	-	-	-
Import Diff	01:29:05	5.18 GB	-	-	-
2004-01-01	00:02:12	5.39 GB	1,025	3,250	0.870,244
2005-01-01	00:01:10	5.40 GB	3,938	25,248	0.886,237
2006-01-01	00:02:08	5.44 GB	12,158	109,030	0.898,339
2007-01-01	00:04:54	5.53 GB	32,821	301,002	0.925,932
2008-01-01	00:12:17	5.70 GB	61,773	619,343	0.940,039
2009-01-01	00:28:00	6.00 GB	106,729	1,099,710	0.947,371
2010-01-01	00:35:39	6.63 GB	152,914	2,784,841	0.953,771
2011-01-01	00:42:35	7.17 GB	193,434	3,463,370	0.950,913
2012-01-01	00:53:36	7.81 GB	225,884	4,169,790	0.951,347
2013-01-01	01:12:14	8.75 GB	268,830	5,157,335	0.961,254
2014-01-01	03:09:10	9.40 GB	309,732	3,727,148	0.961,431
2015-01-01	01:11:51	10.23 GB	342,115	6,669,247	0.961,530
2016-01-01	01:07:12	11.10 GB	382,936	7,119,001	0.930,064
2017-01-01	01:11:26	12.03 GB	410,131	7,545,065	0.931,100
2017-10-01	01:08:56	13.00 GB	433,189	7,895,909	0.932,076
Link-Dump	06:20:37	13.31 GB	433,214	8,022,464	0.938,425

Table 1: Statistics of the creation of PageTiers of the simple English Wikipedia representing annual snapshots of HTML content and page network.

Table 1 summarizes our evaluation on import and PageTier creation. The latter consists of two steps: parsing content in HTML and extracting a network based on the HTML code. Extracted information is stored in the database. If the network structure is not needed, this step could be skipped to save disc space and computing time. The combined Diff-Zip2 compression for revisions results in a database of 5.18GB, saving about 84% of disc space compared the uncompressed version while taking about 3 times longer for import. While the number of pages $|V|$ constantly grows, the number of $|E|$ edges drops intermediately when extracted on 2014-01-01. Nonetheless the ratio of the size of the greatest connected component $|GCC|$ to the number of vertices $|V|$ is generally stable, even around this time.
We use the network structure of the PageTier on midnight 2017-10-01 to evaluate the extraction against an import of the official SQL link dumps[9]. Table 2 shows that the results for page-links and redirects are above 98%. Recall and FScore for categorization is about 4% less, indicating that the extraction still misses an overproportional amount of category links. This will be adressed in future work.

Link-Type	Precision	Recall	FScore
Page-Link	0.990,380	0.987,971	0.985,423
Categorization	0.988,782	0.944,177	0.944,971
Redirect	0.999,885	0.999,901	0.999,785

Table 2: Evaluation of link extraction based on HTML against SQL link dumps of the simple English Wikipedia.

These experiments were conducted using Neo4J as backend. However WikiDragon is by design not limited to a specific backend but can be adapted to other database systems by implementing specific interfaces. Since Neo4J does not provide horizontal scaling for write operations, incorporating other database systems like Cassandra[10] or BlazeGraph[11] are likely to significantly improve performance for import- especially when processing large Wikipedias like German or English. Neo4J has been chosen because it provides sufficient performance in many use cases and does not require any additional standalone installation.

3.1. Comparing WikiDragon and JWPL

Whenever conclusions are drawn on any given resource we need to ensure that the representation of the resource is as accurate as possible. In this subsection we examine the impact of the different approaches of WikiDragon (XOWA) and JWPL (Sweble) to HTML rendering on exemplary use cases of language studies. We assume that different technologies lead to different results which should be reflected in terms of distribution characteristics such as the type-token ratio. We use the actual Web API of the simple English Wikipedia to build a reference as gold standard. We chose this instance because it allows us to perform a complete analysis over all articles. A complete download based on the English Wikipedia based on the Web API would not have been feasable.
For WikiDragon and JWPL we use the meta-current dumps of the simple English Wikipedia. The import for JWPL was created using DataMachine with standard parameters. Since JWPL only considers pages in the namespaces of *articles*, *articles talk* and *categories*, we focus our analysis on articles. Furthermore JWPL appears to automatically resolve redirects by default so that only non-redirect articles are considered. For our experiment we use the intersection

[7] Transactional mode can be activated in Neo4J lateron without the need to rebuild the database. All experiments published in this work are conducted without transactions.

[8] https://dumps.wikimedia.org/simplewiki/20171001/

[9] Note that there is a delta of 25 pages which have been created on 2017-10-01 after midnight.

[10] http://cassandra.apache.org/

[11] https://www.blazegraph.com/

extracted by WikiDragon, JWPL and Web API of the simple English Wikipedia (extracted on 2017-09-29, using the latest revision IDs of the dump used for WikiDragon and JWPL) resulting in 127,634.00 articles. We extract plain text from each article by the default methods provided by the frameworks. As normalization towards JWPL we strip the title header as well as the categories section from the WikiDragon output. From the reference Web API output we strip the section index which is automatically created for larger pages as well as the "edit"-labels which are generated for section titles. For tokenization we use *PTBTokenizer* of *Stanford CoreNLP*(Manning et al., 2014).

Figure 4 shows a comparison of the type-token ratio distributions based on WikiDragon/XOWA (blue) and JWPL/Sweble (red) against the Web API as reference. The distributions are sorted in ascending order based on the Web API variant. As the plots indicate, both distributions deviate from the gold standard to some degree. In case of Sweble the distribution is scattered much more, indicating a significant difference to the original data. Accordingly, the correlation coefficients[12] between XOWA and Web API are considerably higher (0.955, T=15,679, $P < 2.2e^{-16}$) than between Sweble and Web API (0.663, T=3,462, $P < 2.2e^{-16}$).

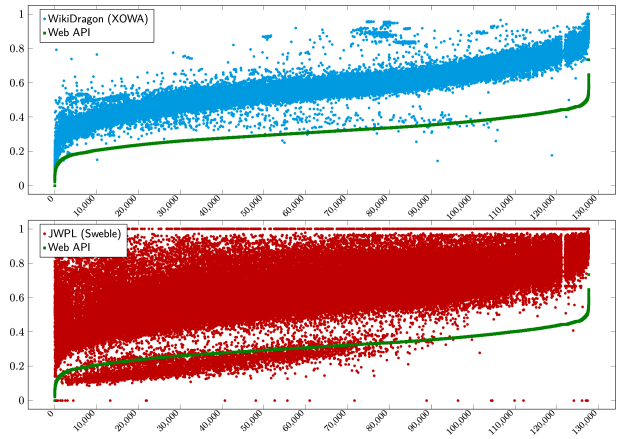

Figure 4: Comparison of the type-token distributions based on WikiDragon/XOWA (blue) and JWPL/Sweble (red) against the Web API as reference.

4. Conclusion

We introduced WikiDragon as a Java framework to build, process and analyse Wikis offline. It is unique in terms of combining the aim of complete coverage of wikis with an efficient API access for developers. We showed that the system enables researchers to conduct diachronic studies on content and network structure of wikis. An evaluation against the official Wikimedia dumps showed high accuracies for link extraction based on HTML. To our knowledge the diachronic network analysis on the simple English Wikipedia is the first ever performed. We examined the impact of using different HTML rendering engines for MediaWiki markup. Results show that XOWA performs more accurate than Sweble. Future work will address the integration of additional DBMS, the development of API extensions for Wikimedia projects such as Wiktionary and WikiNews and increasing accuracy of link detection. WikiDragon is available on GitHub[13].

[12]The distance correlations haven been computed and averaged over random samples based on 10 cycles using 10,000 items each.

5. Bibliographical References

Dandala, B., Mihalcea, R., and Bunescu, R. (2013). Word sense disambiguation using wikipedia. In Gurevych I. et al., editors, *The People's Web Meets NLP. Theory and Applications of Natural Language Processing.*, pages 241–262. Springer, Berlin, Heidelberg.

Daxenberger, J. and Gurevych, I. (2012). A corpus-based study of edit categories in featured and non-featured wikipedia articles. In *Proceedings of the 24th International Conference on Computational Linguistics (COLING 2012)*, pages 711–726, December.

Dohrn, H. and Riehle, D. (2011). Design and implementation of the sweble wikitext parser: Unlocking the structured data of wikipedia. In *Proceedings of the 7th International Symposium on Wikis and Open Collaboration*, WikiSym '11, pages 72–81, New York, NY, USA. ACM.

Ferschke, O., Zesch, T., and Gurevych, I. (2011). Wikipedia revision toolkit: Efficiently accessing wikipedia's edit history. In *Proceedings of the ACL-HLT 2011 System Demonstrations*, pages 97–102, Portland, Oregon, June. Association for Computational Linguistics.

Flöck, F. and Acosta, M. (2014). Wikiwho: Precise and efficient attribution of authorship of revisioned content. In *Proceedings of the 23rd International Conference on World Wide Web*, WWW '14, pages 843–854, New York, NY, USA. ACM.

Gabrilovich, E. and Markovitch, S. (2007). Computing semantic relatedness using wikipedia-based explicit semantic analysis. In *In Proceedings of the 20th International Joint Conference on Artificial Intelligence*, pages 1606–1611.

Lehmann, J., Isele, R., Jakob, M., Jentzsch, A., Kontokostas, D., Mendes, P. N., Hellmann, S., Morsey, M., van Kleef, P., Auer, S., and Bizer, C. (2015). DBpedia - a large-scale, multilingual knowledge base extracted from wikipedia. *Semantic Web Journal*, 6(2):167–195.

Manning, C. D., Surdeanu, M., Bauer, J., Finkel, J., Bethard, S. J., and McClosky, D. (2014). The Stanford CoreNLP natural language processing toolkit. In *Association for Computational Linguistics (ACL) System Demonstrations*, pages 55–60.

Mehler, A., Gleim, R., vor der Brück, T., Hemati, W., Uslu, T., and Eger, S. (2016). Wikidition: Automatic lexiconization and linkification of text corpora. *Information Technology*, pages 70–79.

Mehler, A., Gleim, R., Lücking, A., Uslu, T., and Stegbauer, C. (2018). On the self-similarity of Wikipedia talks: a combined discourse-analytical and quantitative approach. *Glottometrics*, 40:1–44.

Nothman, J., Curran, J. R., and Murphy, T. (2008). Transforming wikipedia into named entity training data. In *In*

[13]https://github.com/texttechnologylab/WikiDragon

Proceedings of the Australasian Language Technology Association Workshop 2008, pages 124–132.

Priedhorsky, R., Chen, J., Lam, S. T. K., Panciera, K., Terveen, L., and Riedl, J. (2007). Creating, destroying, and restoring value in wikipedia. In *Proceedings of the 2007 International ACM Conference on Supporting Group Work*, GROUP '07, pages 259–268, New York, NY, USA. ACM.

Uslu, T., Mehler, A., Baumartz, D., Henlein, A., and Hemati, W. (2018). fastsense: An efficient word sense disambiguation classifier. In *Proceedings of the 11th edition of the Language Resources and Evaluation Conference, May 7 - 12*, LREC 2018, Miyazaki, Japan.

Yasseri, T., Sumi, R., Rung, A., Kornai, A., and Kertész, J. (2012). Dynamics of conflicts in wikipedia. 7:e38869, 06.

Zesch, T., Müller, C., and Gurevych, I. (2008). Extracting lexical semantic knowledge from wikipedia and wiktionary. In *in Proceedings of LREC*.