Java Libraries for Accessing the Princeton Wordnet: Comparison and Evaluation

Mark Alan Finlayson
Computer Science and Artificial Intelligence Laboratory
Massachusetts Institute of Technology
32 Vassar Street, Room 32-258, Cambridge, Massachusetts, 02139
markaf@mit.edu

Abstract

Java is a popular programming language for natural language processing. I compare and evaluate 12 Java libraries designed to access the information in the original Princeton Wordnet databases. From this comparison emerges a set of decision criteria that will enable a user to pick the library most suited to their purposes. I identify five deciding features: (1) availability of similarity metrics; (2) support for editing; (3) availability via Maven; (4) compatibility with retired Java versions; and (5) support for Enterprise Java. I also provide a comparison of other features of each library, the information exposed by each API, and the versions of Wordnet each library supports, and I evaluate each library for the speed of various retrieval operations. In the case that the user's application does not require one of the deciding features, I show that my library, JWI, the MIT Java Wordnet Interface, is the highest-performance, widest-coverage, easiest-to-use library available.

A Java developer seeking to access the Princeton Wordnet is faced with a bewildering array of choices: there are no fewer than 12 Java libraries that provide off-the-shelf access to Wordnet data, each with various combinations of features and performance. In addition to these 12 libraries, there are also at least 12 additional libraries[1] that, while not providing direct access to Wordnet data themselves, provide functions such as similarity metrics and deployment of Wordnet data to database servers. In this paper I compare, contrast, and evaluate each of the 12 libraries[2] that provide direct access to the Princeton Wordnet data, so as to help Java developers find the library that is right for their application. To my knowledge this is the first paper to attempt a thorough comparison of any of these libraries.

I proceed as follows. First I present the bottom line, which is a set of five deciding features most commonly encountered when using Wordnet in a Java. I then discuss other features that distinguish some libraries from the others. I present an assessment of what Wordnet data is accessible via which library, and which libraries are compatible with which Princeton Wordnet versions. I also evaluate the performance of each library on nine different retrieval metrics, as well as the time to initialize in-memory Wordnet dictionaries for those libraries that suport that function.

The code for reproducing the evaluation (including all required source code, copies of all the described libraries, and the various versions of Wordnet) is available online.[3]

While the software evaluated in this paper is exclusively for Java, and is limited to libraries available at the time of writing that are designed for accessing the original Princeton Wordnet, this work should be helpful to those who seek to evaluate other application programming interfaces (APIs) for interacting with Wordnet data. In particular the set of features identified here and the set of retrieval metrics should be of some use.

1 Deciding on a Library

Before discussing the feature and performance evaluation in detail I will lay out the bottom line: which library a developer should choose if your application falls into one of the common situations described below. First, I will outline which library a developer should choose if there are no particular constraints. Next, I list five deciding features that, if an application needs that feature, will de-

[1] See Table 6 for a list of all libraries and their URLs.

[2] I have made my best effort to be as complete as possible in identifying libraries that support access to Wordnet. It is possible, however, that I have missed some more obscure libraries, especially libraries whose primary purpose is not Wordnet access but some other function.

[3] Via the MIT DSpace repository as an MIT CSAIL Work Product: http://hdl.handle.net/1721.1/81949

termine which library the developer should choose (or which libraries there are to choose from).

Note that an application may have additional or alternate special requirements that are not explicitely discussed here. If this is the case the developer should examine the tables and figures in this paper, as well as the project websites (Table 6), to determine what library provides the right combination of features and performance.

1.1 No Special Requirements

If there have no special requirements, then the library a developer should choose is my own: JWI, the MIT Java Wordnet Interface. JWI is a mature library, nearly five years old, and has demonstrated its stability and utility, having been downloaded over 15,000 times in the past five years. It has the following nine advantages: (1) JWI supports access to the widest array of information in the widest selection of Princeton Wordnet versions (see Tables 2 and 3), plus has been tested on a number of Wordnet variants; (2) JWI uses the Wordnet files as they are distributed with no modifications; (3) JWI provides both file-based and in-memory dictionary implementations, allowing you to trade off speed and memory consumption; (4) JWI sets no limit on the number of dictionaries that may be instantiated in each JVM; (5) JWI is high-performance, with top-ranked speeds on various retrieval metrics and in-memory dictionary load time (see Tables 4 and 5 and Figure 1); (6) JWI has a small on-disk footprint and requires no additional Java libraries, no native dynamically-loaded libraries (dlls), and no configuration files; (7) JWI has extensive documentation, including Javadoc and a User's guide with code examples; (8) JWI is open-source and distributed under a license which allows it to be used for any purpose; and (9) JWI is being actively supported and developed by myself.

There are, however, at least five deciding features that, if an application requires them, will potentially lead to another library. These features are listed below (and are included in Table 1).

1.2 Similarity Metrics

The availability of similarity metrics is the most common deciding feature, as many developers want to use Wordnet not *per se*, but so as to measure the semantic similarity between words. JWNL has the most similarity metrics to choose from, with at least three different compatible libraries providing this function: RitaWN, WNSim, and WordnetSim.

Choosing JWNL, however, entails a few penalties: First, JWNL requires a notoriously confusing and error-prone external configuration file; second, JWNL depends on an external library, Apache Commons Logging; third, JWNL follows the singleton dictionary model, in that it only allows one dictionary to be open at a time; finally, JWNL has rather poor performance relative to other libraries. If these factors outweigh the positives of having the widest array of similarity metrics, then there are four other libraries that have some measure of similarity metric support: Javatools, Jawbone, JawJaw, and JWI.

1.3 Editing

If your application depends on being able to edit the Wordnet data, there is only option: extJWNL. This library is a re-implementation of JWNL for Java 1.5, copying much of the same source code, and so it suffers from the same problems as JWNL as described above, with the additional caveat that has an additional dependency: a custom Map implementation.

1.4 Maven

If an application's build process uses Maven, and the project absolutely requires that dependent libraries be available in the Maven repository, then extJWNL is the only choice.[4] As noted above, extJWNL suffers from a number of problems.

1.5 Retired Java Versions

Java is backward-compatible, meaning libraries compiled on older Java versions will still run under newer versions, but it is not forward-compatible: libraries compiled with newer compilance levels will not run in older JVMs. If an application requires libraries that will run under Java 1.4, then the developer should choose JWNL[5]. If an application requires Java 1.5, then the developer should choose JWI[6].

[4] Some versions of JWNL and JWI are available in the Maven repository. However, publishing artifacts to the repository is not currently a part of the JWI build process, and therefore there is no guarantee that future versions will be available there.

[5] JAWS will also run under 1.4, but lacks significantly in features and performance.

[6] JawJaw also will run under 1.5, but is sorely lacking in features, performance, and compatibility.

Feature	CICWN	extJWNL	Javatools	Jawbone	JawJaw	JAWS	JWI	JWNL	URCS	WNJN	WNPojo	WordnetEJB
Version	1.0	1.6.10	10-1-2012	2009-07-04	1.0.2	1.3	2.3.0	1.4.1rc2	1.0	1.0	1.0.1	1.0.0-beta
License	GPL	BSD	CC-BY	MIT	Apache	Custom[1]	CC-BY	BSD	GPL	GPL	GPL	GPL
Minimum Java	1.6	1.6	1.6[2]	1.6	1.5	1.4	1.5	1.4	1.6	1.5	1.6	1.6
Binary Size	1.25mb	235kb	398kb	30kb	40.9mb	58kb	148kb	202kb	188kb	11kb	119kb	11.45mb
Standalone	Yes[3]	-[4]	Yes	Yes	Yes	Yes	Yes	-[5]	Yes	-[6]	-[7]	-[8]
Last Release	2011	2013	2012	2009	2013	2009	2013	2008	2010	2006	2010	2010
Active	-	Yes	-	-	Yes	-	Yes	-	-	-	-	-
Maven	-	Yes[9]	-	-	-	-	-[10]	Yes	-	-	-	-
Editing	-	Yes	-[11]	-	-	-	-	-	-	-	-	-
EJBs	-	-	-	-	-	-	-	-	-	-	-	Yes
Multiple Dicts	-	-	Yes[12]	-	-[13]	-	Yes	-	Yes	-	Yes	Yes
Normal Files	Yes[3]	Yes[14]	-[15]	Yes	-[16]	Yes	Yes	-[17]	Yes	Yes	-[18]	-
GUI	-	-	-	-	-	-	-	-	Yes	Yes	-	-
Similarity Metrics	-	-	Yes	Yes[19]	Yes[20]	-	Yes[21]	Yes[22]	-	-	-	-
File-Based Dict	-	Yes	-	Yes	Yes	Yes	Yes	Yes	Yes	Yes	-	-
Database Dict	-	Yes	-	-	-	-	-	Yes	-	-	Yes	Yes
In-Memory Dict	Yes	Yes	Yes	-	Yes	-	Yes	Yes	-	-	-	-

Table 1: Information on and supported features of each library.

License

[1] JAWS license is similar to the MIT License.

Minimum Java

[2] Javatools requires a 64-bit JVM to load all supported pointers into memory.

Standalone

[3] CICWN requires Wordnet files to be placed in a particular sub-directory, plus a file containing a list of prepositions to use the plain Wordnet functionality; it requires additional libraries and data files to use the full stemming functionality.
[4] extJWNL requires an external properties file, Apache Commons Logging, and a custom Map implementation.
[5] JWNL requires Apache Commons Logging.
[6] WNJN requires a native library that depends on the wordnet version in use. The native library is available in for Windows and Linux 32-bit, but would have to be re-compiled using C++ for other platforms.
[7] WNPojo requires approximately 14 supporting libraries.
[8] WordnetEJB requires a Database server and a Java Application server deployed with the WordnetEJB implementation.

Maven

[9] extJWNL versions 1.5.0 to 1.5.3 and 1.6.0 to 1.6.10 are available in the Maven repository.
[10] JWI Versions 2.2.1, 2.2.2, and 2.2.3 are available in the Maven repository.

Editing

[11] Javatools allows you to remove synsets from the in-memory dictionary only.

Multiple Dictionaries

[12] Javatools allows multiple dictionaries to be instantiated, but each dictionary only captures one relation.
[13] JawJaw only allows single dictionary to be opened for the life of each JVM.

Normal Files

[14] extJWNL in-memory dictionary uses special files that must be compiled from the normal Wordnet files.
[15] Javatools uses the Prolog-formatted Wordnet files.
[16] JawJaw uses an sqlite3 file, generated from the Japanese Wordnet files.
[17] JWNL's in-memory dictionary implementation requires special files that must be compiled separately from the Wordnet files.
[18] WNPojo requires the normal Wordnet files to be processed and loaded into a relational database.

Similarity Metrics

[19] Jawbone has similarity metrics via the RitaWN library.
[20] JawJaw similarity metrics are provided by the WS4J library.
[21] JWI similarity metrics are available via the Java Wordnet::Similarity library (JWS).
[22] JWNL similarity metrics are available via the RitaWN, WNSim, and WordnetSim libraries.

1.6 Enterprise Java

Finally, if an application absolutely requires that Wordnet data be accessible via an Enterprise Java Bean (EJB), the only out-of-the-box choice is WordnetEJB, which provides all the tools to deploy an EJB that provides access to Wordnet onto a Java application server. Unfortunately, given WordnetEJB's dismal performance and difficulty of use, one is probably better off implementing one's own EJB by wrapping another library.

2 Features and Information

I expand now on other features of the libraries which, while not necessarily decisive, are worthy of consideration when other factors do not compel your choice.

2.1 Features

As noted, Table 1 shows the basic list of features, which was constructed by taking the union of all features[7] for all libraries. I describe in this section those not yet discussed. A dash in a particular cell means that I determined, either by reading the documentation or the code, that the library did not support that feature. It is important to understand that I consider here only out-of-the-box features and compatibility: because the source code for each library is available, an enterprising developer could certainly modify any of these libraries to provide any of the lacking features. Most developers, however, will not be willing or able to invest the time required for this, and thus are restricted to the features provided.

Binary Size This feature indicates the size of the binary jar file on disk. This number does not include the size of any required dependencies or external files, and does not include the size of the Wordnet data files. The size of the libraries ranges dramatically: from a mere 11kb for WNJN to 40.9mb for JawJaw. JWI clocks in at a quite modest 202kb, which is approximately the median of the range.

Standalone Whether or not the library requires additional Java libraries or external resources to run (other than the Wordnet files themselves). In certain cases, such as WNPojo, these external libraries are extensive: at least 14, comprising over 10mb of jar files.

Perhaps the most pernicious requirements are those for the JWNL/extJWNL pair and WNJN. Both JWNL and extJWNL require an external configuration file (in XML format) that sets various properites of the singleton dictionary. These parameters cannot be set programmatically, and the file is not well documented, which leads to quite a bit of consternation in the use of these libraries.

WNJN, on the other hand, is a JNI interface to a native dll. Using WNJN thus means that one looses the platform-independence so prized in Java (unfortunately for not much gain: WNJN is impoverished both in features and performance compared to other libraries).

JWI is especially easy to use: it requires no external libraries or files to run (other than the Wordnet files themselves), its out-of-the-box defaults are suitable to most applications, and any configuration required can be done programmatically.

Last Release The year when the most current version was released. JWI is one of only three libraries that saw an update in 2013, the year this paper was written.

Active Whether or not the project appears to be under active development. The last release year, along with indications of activity on the project's webpage or correspondence with the developer, were used to determine this feature.

Multiple Dictionaries Here *dictionary* refers to a Java object which manages access to the Wordnet data. This feature indicates whether or not multiple dictionaries can be open at the same time. This, for example, would be useful in a context where you want simultaneous access to different Wordnet versions. Many of the Wordnet libraries have, unfortunately, adopted the singleton design pattern, where only one Wordnet dictionary may be instantiated at a time. Fortunately, most of these libraries do allow the dictionary to be closed and a new dictionary to be opened.[8] JWI allows any number of dictionaries to be open simultaneously.

Normal Files Whether or not the library uses the normal Wordnet files as distributed. Some libraries require an unusual format (e.g., the Prolog versions of the files), or require the files to be processed in some way before the library can be used to access the data. JWI uses the Wordnet files as provided.

[7] Note that due to space limitations I do not discuss in detail the ease of use of the various APIs.

[8] The exception to this is JawJaw, which does not allow the dictionary to be disposed and thus only allows a single dictionary to open for the life of the JVM.

GUI Whether or not the library provides a graphical user interface (GUI) to interact with Wordnet data. Only two libraries, URCS and WordnetEJB, provide a GUI.

File-based Dictionary Whether or not the library provides a dictionary implementation that reads Wordnet information directly from the files when requested. Four libraries do not provide such an implementation: CICWN and Javatools, which provide in-memory implementations only; and WNPojo and WordnetEJB, which use a database-backed implementation.

Database-backed Dictionary Whether or not the library provides a dictionary implementation that retrieves Wordnet data from a database server. JWI does not provide database-backed access, but four libraries do: JWNL, extJWNL, WNPojo, and WordnetEJB.

In-Memory Dictionary Whether or not the library provides a dictionary implementation that loads Wordnet information completely into memory. These implementations allow for extremely fast data access speeds, at the price of initialization time (see Figure 1). JWI provides an in-memory dictionary implementation.

2.2 Accessible Data

Each library provides access to a different subset of the information contained in Wordnet. Information in Wordnet is stored across four different types of files: *index* files, *data* files, *exception* files, and the *sense.index* file. Each Wordnet library provides access to various subsets of the information contained in Wordnet, and this is captured in Table 2. The only library that provides complete access to all the Wordnet data is JWI, although JWNL, extJWNL, WNPojo, and WordnetEJB all come close.

2.3 Supported Wordnet Versions

Table 3 shows which libraries are compatible with which Wordnet versions. Most libraries support Princeton Wordnet versions 1.6 and above. No library supports Wordnet 1.5, and no library supports access to the Wordnet 1.6 cousin files or 3.1 stand-off annotations.

The final row in Table 3 indicates known compatibility with other Princeton Wordnet variants. JWI is the only library I know for sure that supports Wordnet variants, namely, the Stanford Augmented Wordnets (Snow et al. 2006). Other libraries can probably support Princeton Wordnet variants that conform to the Wordnet file specifications, and so the question mark only indicates that, to my knowledge, compatibility has not been demonstrated or documented.

3 Performance Evaluation

In addition to the features listed above, I also evaluated the performance of each library under nine different retrieval metrics (as applicable). I wrote a standard test harness that ran each library through its paces in exactly the same environment.[9] For those libraries that provide an in-memory dictionary implementation, I also measured how long it took for that implementation to load Wordnet into memory.

3.1 Retrieval Times

I measured three different types of retrieval metrics. First, I measured the speed of iteration over the four main object types (corresponding to the four file types). For index files, for example, I measured the average time for the dictionary to iterate over all index words in Wordnet. Second, I measured the speed of retrieval for individual objects of the four different types, given the minimally necessary identifying information. For index files, for example, I measured the average time to retrieve an index word given a lemma and part of speech. Third, I measured the time to iterate across all index words and retrieve the synsets listed in those index words.

Not every library supports all nine different types of retrieval: Tables 4 and 5 show which libraries support which retrieval type. The only libraries that support every type of retrieval are JWI and WNPojo. For retrieval of individual objects, JWI outperforms WnPojo by a factor of 10. For iteration over object types, JWI and WNPojo are approximately equivalent, except for iteration over synsets by index words, where JWI outperforms WNPojo by a factor of 25.

A note on CICWN: I include CICWN's retrieval times even though the library does not provide

[9] The testing machine was a Windows 7 Enterprise 64-bit server-class machine, with 2 Intel Xeon X5570 CPUs (4 cores each, running at 2.9 GHz), 24 GB of RAM, and two 15krpm high-performance Sata 3 drives in a RAID 0 configuration (The machine was state-of-the-art in approximately 2010). Tests were performed within Eclipse 3.8.0, using Sun Java 1.6 64-bit, revision 22. MySQL version 5.6 was used for the database server, and JBoss 5.1.0 was used for the Java Application Server. During testing the machine was unburdened with other tasks.

File type	Data	CICWN	extJWNL	Javatools	Jawbone	JawJaw	JAWS	JWI	JWNL	URCS	WNJN	WNPojo	WordnetEJB
Index	Synsets	Yes	Yes	-	Yes	Yes	Yes	**Yes**	Yes	Yes	Yes	Yes	Yes[1]
	Synset Counts	Yes	Yes	-	Yes	Yes	Yes	**Yes**	Yes	-	Yes	Yes	Yes[1]
	Pointer Counts	-	-	-	Yes	-	-	**Yes**	-	-	Yes	-	-
	Pointer List	-	-	-	Yes	-	-	**Yes**	-	-	-	-	-
Data	Tag Sense Count	-	-	-	Yes	-	Yes	**Yes**	Yes	-	Yes	-	-
	Synonyms	Yes	Yes	Yes	Yes	Yes	Yes	**Yes**	Yes	Yes	Yes	Yes	Yes[1]
	Lexical Filenum	-	Yes	-	Yes	-	-	**Yes**	Yes	Yes	Yes	Yes	Yes[1]
	WordCount	Yes	Yes	Yes	Yes	Yes	Yes	**Yes**	Yes	Yes	Yes	Yes	Yes[1]
	LexicalID	-	Yes	-	Yes	Yes	-	**Yes**	-	Yes	Yes	Yes	Yes[1]
	Semantic Pointers	Yes	Yes	Yes[2]	Yes	Yes	Yes	**Yes**	Yes	Yes	Yes	Yes	Yes[1]
	Lexical Pointers	Yes	Yes	-	Yes	-	Yes	**Yes**	Yes	Yes	Yes	Yes	Yes[1]
	Verb Frames	-	Yes	-	Yes	-	Yes	**Yes**	Yes	Yes	Yes	Yes	Yes[1]
	Adjective Marker	-	Yes	-	Yes	-	Yes	**Yes**	Yes	-	Yes	Yes	Yes[1]
	Gloss	Yes	Yes	-	Yes	Yes	Yes	**Yes**	Yes	Yes	Yes	Yes	Yes[1]
Exception	Inflected Form	Yes	Yes	-	-	-	-	**Yes**	Yes	-	-	Yes	Yes[1]
	Base Forms	Yes	Yes	-	-	-	Yes	**Yes**	Yes	-	-	Yes	Yes[1]
Sense	Sense Key	-	Yes	-	-	-	-	**Yes**	Yes	Yes	-	Yes	Yes[1]
	Tag Counts	-	Yes	-	-	-	-	**Yes**	Yes	-	-	Yes	Yes[1]

Table 2: Wordnet data accessible from each library.

[1] WordnetEJB returns all data as XML documents: it provides no Java API for accessing data within an index word, word, synset, sense entry, or exception entry record.
[2] Javatools only supports some semantic pointer types.

Version	CICWN	extJWNL	Javatools	Jawbone	JawJaw	JAWS	JWI	JWNL	URCS	WNJN	WNPojo[1]	WordnetEJB[1]
1.6	Yes	Yes	-	Yes	-	Yes	**Yes**	Yes	Yes	Yes	Yes	Yes
1.7	Yes	Yes	-	Yes	-	Yes	**Yes**	Yes	Yes	Yes	Yes	Yes
1.7.1	Yes	Yes	Yes	Yes	-	Yes	**Yes**	Yes	Yes	Yes	Yes	Yes
2.0	Yes	Yes	Yes	Yes	-	Yes	**Yes**	Yes	Yes	Yes	Yes	Yes
2.1	Yes	Yes	Yes	Yes	-	Yes	**Yes**	Yes	Yes	Yes	Yes	Yes
3.0	Yes	Yes	-[2]	Yes	Yes	Yes	**Yes**	Yes	Yes	Yes	Yes	Yes
3.1	Yes	Yes	-	Yes	-	Yes	**Yes**	Yes	Yes	Yes	Yes	Yes
Other	?	?	?	?	-	?	**Yes**	?	?	?	?	?

Table 3: Versions of the Princeton Wordnet supported by each library. No library supports version 1.5, version 1.6 cousin files, or the 3.1 stand-off files.

[1] WNPojo/WordnetEJB do not provide pre-compiled Wordnet database images other than for Wordnet 3.1 for MySQL; other Wordnet versions require the user to compile the Wordnet files into the database image (and load it into the appropriate database server) using the WNSQLBuilder project.
[2] Javatools throws an exception when loading Wordnet 3.0 prolog files.

a file-based dictionary implementation. This is not a completely direct comparison, however, as CICWN requires all of WordNet be loaded into memory (with associated memory footprint and initialization time penalties). It is interesting to note, however, that CICWN's in-memory performance is comparable to JWI's file-based performance, with retrieval times around the neighborhood of 10 microseconds. JWI's in-memory retrieval significantly outperforms CICWN (I do not show those results here for lack of space).

3.2 In-Memory Dictionaries

Six libraries support in-memory dictionary implementations. Of them, JawJaw supports only Wordnet 3.0. JWNL, extJWNL and JawJaw all have average load times (the time to load the Wordnet data fully into memory) in the 15-20 second range. Of the remaining three, Javatools and CICWN do not support access to the full range of Wordnet data. Only JWI has a load time of a few seconds and supplies complete access to all Wordnet data.

4 Conclusion

For an application without special constraints, most Java developers should use JWI to access Wordnet, for three reasons. First, it is among the easiest to use: it has extensive documention, a small disk footprint, requires no special configuration or supporting libraries, and is completely configurable programmatically. Second, it supports the most Wordnet versions and variants, and its API exposes all available Wordnet data. Third, it has top-tier performance, often outperforming other Java libraries by factors of 5 to 100.

Acknowledgments

The preparation of this article was supported by DARPA under grant D12AP00210.

References

Rion Snow, Daniel Jurafsky, and Andrew Y. Ng. 2006. Semantic taxonomy induction from heterogenous evidence. In *Proceedings of the 44th Annual Meeting of the Association for Computational Linguistics*, Sydney, Australia, pages 801–808.

Retrieval of... (μs)	**WordnetEJB**	**JAWS**	**URCSWordnet**	**extJWNL**	**JWNL**	**Jawbone**	**WNJN**	**WNPojo**	**JawJaw**	**JWI**	**CICWN**
Index Word	506ms	4.1ms	2662.5	1.5ms	1.5ms	-	253.3	184.5	67.4	12.3	22.9
Synset	-	-	-	3.3ms	479.6	768	228.6	226.6	61.9	7.1	4.1
Word-by-Sense-Key	-	-	-	11.1ms	-	-	-	176.1	-	17.2	-
Exception Entry	-	2.1	-	545.3	537.9	-	-	138.5	-	16.1	1.7

Table 4: Average time to retrieve an object of the named type (from Wordnet 3.0) using a file-backed dictionary, for libraries that support this functionality. Times are in microseconds (μs), unless otherwise noted (ms = milliseconds).

Iteration Over... (ms)	**extJWNL**	**JWNL**	**Jawbone**	**WNPojo**[1]	**JWI**	**CICWN**[2]
Index Words	16.4s	16.4s	192	393	296	-
Synsets	6.4m	56.1s	-	273	798	1
Words via Sense Keys	-	-	-	635	141	-
Exception Entries	271	274	-	10	4	1
Synsets by Index Words	15.7m	2.1m	5.6m	51.0s	1.8s	-

Table 5: Average time to iterate over all objects of the named type (from Wordnet 3.0) using a file-backed dictionary, for libraries that support this functionality. Times are in milliseconds, unless otherwise noted (s = seconds, m = minutes).

[1] WNPojo uses a database-based dictionary implementation.
[2] CICWN only provides an in-memory dictionary implementation.

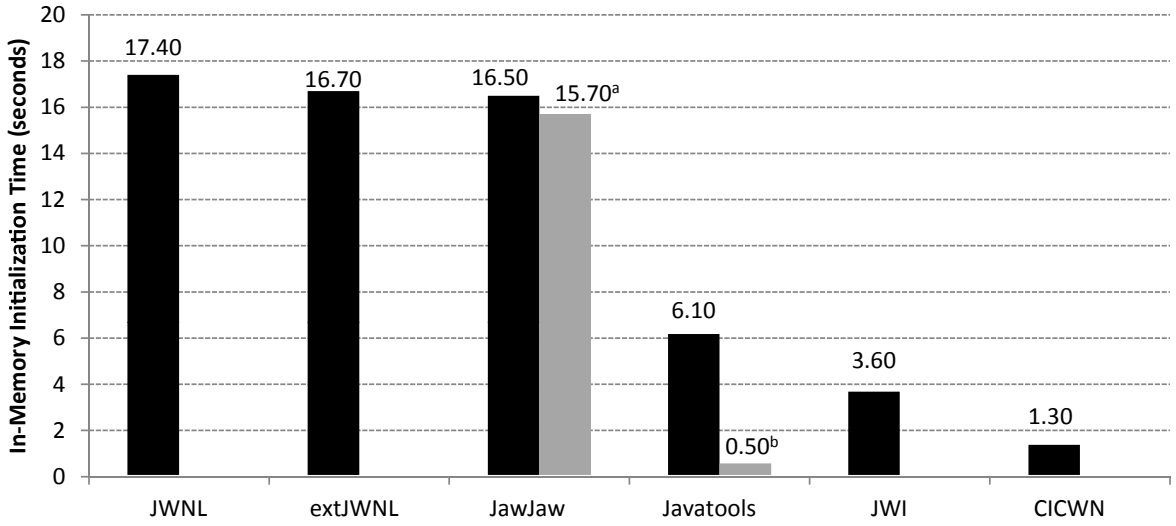

Figure 1: Times to load Wordnet into memory for the libraries that support in-memory dictionaries.

[a] JawJaw has a slightly lower load time when the data file is already present in the temporary directory.
[b] Javatools has a lower load time when loading only synsets, with no pointers.

	Library	URL
Wordnet Libraries	CICWN	http://fviveros.gelbukh.com/wordnet.html
	extJWNL	http://extjwnl.sourceforge.net/
	Javatools	http://www.mpi-inf.mpg.de/yago-naga/javatools/
	Jawbone	http://sites.google.com/site/mfwallace/jawbone/
	JawJaw	http://www.cs.cmu.edu/~hideki/software/jawjaw/
	JAWS	http://lyle.smu.edu/~tspell/jaws/
	JWI	http://projects.csail.mit.edu/jwi/
	JWNL	http://sourceforge.net/apps/mediawiki/jwordnet/
	URCS	http://www.cs.rochester.edu/research/cisd/wordnet/
	WNJN	http://wnjn.sourceforge.net/
	WNPojo	http://wnpojo.sourceforge.net/
	WordnetEJB	http://wnejb.sourceforge.net/
Similarity	JWS	http://www.sussex.ac.uk/Users/drh21/
	JWordnetSim	http://nlp.shef.ac.uk/result/software.html
	Rita.WordNet	http://rednoise.org/rita/wordnet/documentation/index.htm
	WNSim	http://cogcomp.cs.illinois.edu/page/software_view/36
	WordnetSim	http://nlp.shef.ac.uk/result/software.html
	ws4j	http://code.google.com/p/ws4j/
Other	Lucene Wordnet	http://mvnrepository.com/artifact/org.apache.lucene/lucene-wordnet/
	WNSQL	http://wnsql.sourceforge.net/
	WNSQLBuilder	http://wnsqlbuilder.sourceforge.net/
	WNTrans	http://wntrans.sourceforge.net/
	WNWA	http://wnwa.sourceforge.net/
	XSSM	http://code.google.com/p/xssm/

Table 6: URLs for each library. The libraries listed in the first section are evaluated in this paper. The similarity libraries provide similarity metrics which use the wordnet libraries. The libraries listed in the "Other" section are mentioned because they do not provide direct access to Wordnet data, but may be confused for libraries that do.