gAnswer User Guide

Catalogue

1 Preface 1

2 Getting Started 2

2.1 Quick Guide 2

2.1.1 Start Running 2

2.2 System requirement 3

2.3 Introduction 4

2.3.1 What is gAnswer 4

2.3.2 Open source and licence 4

2.4 Install 5

2.4.1 Package dependency 5

2.4.2 External interface dependency 5

2.4.3 File dependency 6

2.5 How to use 7

2.5.1 Data format 7

2.5.2 gAnswerHttp 7

3 Advanced 7

3.1 HTTP API Introduction 7

3.1.1 Example 7

3.1.2 API structure 9

3.2 Publication 11

3.3 FAQ 11

4 Others 11

4.1 Contributors 11

4.1.1 Leader 11

4.1.2 Student 12

4.2 Update Log 12

4.3 Legal Issues 12

Appendix A: Export jar in Eclipse 13

Appendix B: How to preprocess rdf data 14

1 Preface

Knowledge base question answering, KB-QA, means given a natural language question, a system parse the question semantically to understand it, and further leverage some knowledge bases to get its answer. In detail, varying from different application fields, KB-QA consists of open-domain question answering, like QA over encyclopedic knowledge, and specific-domain question answering, involving economy, healthcare, religion and so forth. KB-QA usually serve us in the form of robots or search engines. Besides, RDF is an important way to represent knowledge bases.

RDF, Resource Description Framework, is created by W3C. It is a sort of markup language intended to represent various kinds of information and help develop Semantic Web. In RDF model, every object on the Internet is labeled with an unique resource name, an URI, in other words. RDF also use URI to name the relation between resources and their properties, as well as the both ends of the relation, which we often call a triple. Therefore, a RDF data set can be represented by a directed and labeled graph. In this graph, each node stands for a single resource and each vertex means relations or properties.

Based on large RDF data set, we design and implement a natural language QA system for large knowledge base, gAnswer. This is a research program developed and maintained by Data Management Lab of Institute of Computer Science & Technology, Peking University. To learn more about how gAnswer works, please refer to the paper in *Chapter 3.2 Publication*.

The rest of this document include the installation, usage, API, use case and FAQ. gAnswer is now open source on github, following BSD protocol. You are welcome to use gAnswer, report problems and make suggestions. We are devoted to making gAnswer better. You can also leverage gAnawer for all types application as long as our work is acknowledged.

Please ensure you have read *Chapter 4.3 Legal Problems* before you use gAnswer.

2 Getting Started

2.1 Quick Guide

gAnswer is a QA system based on large knowledge base. Given a user question, it will output corresponding SPARQL queries and answers to the question. The program is written in Java, so it is able to run on different platform.

For source code, please check our github page: https://github.com/pkumod/gAnswer

2.1.1 Start Running

Deploy via jar

We strongly recommend you to deploy gAnswer by jar we supply. You should:

1. Download Ganswer.jar and data.rar. We recommend you to download the up-to-date version of these two files from the releases page on our github
2. Decompress Ganswer.jar, you can unzip it under any path, but please ensure that Ganswer.jar itself is under the same path with the unzipped files.
3. Decompress data.rar, you should place the unzipped files under the same path with Ganswer.jar
   Now, the file structure looks like this:
   ./
   ++ addition
   ++ application
   ++ data
   ++ fgmt
   ++ jgsc
   ++ lcn
   ++ lib
   ++ log
   ++ META-INF
   ++ nlp
   ++ paradict
   ++ qa

++ rdf

++ utils

++ Ganswer.jar

4. Run Ganswer.jar and wait for the initialization to end. When you see “Sever Ready!”, you have successfully deploy gAnswer and you can access gAnswer via Http requests!

Run in Eclipse

When you use eclipse to run gAnswer, just clone or download the source code, import the project, add the jar in lib to the build path and put the data in the program.

Utilize

Now we only supply HTTP API. It accept user question in json and return answers and sparqls in the same format. For detailed information please check *Chapter 2.5.2* and *Chapter 3.1*

2.2 System requirement

Data set. gAnswer requires data set in RDF format. Now we use dbpedia 2016. We have make some preprocessing on dbpedia 2016.

External graph database system. gAnswer system needs a graph database system supporting SPARQL to get answers. In current version, we choose gStore. We build a database on gStore with preprocess dbpedia 2016 data set, and interact with it via HTTP. About gStore, please check its github: https://github.com/pkumod/gStore

External toolkit. In the question understanding phrase, gAnswer relies on some NLP toolkit, including maltparser and StanfordNLP. When generating SPARQL, it needs Lucene to search supplementary information.

Others. As follows:

Item	Requirement
Operating System	Linux，Windows
System Architecture	x86_64

Disk	>8GB
Main Memory	>20GB
Java	version >= 1.6

2.3 Introduction

2.3.1 What is gAnswer

As increasing structural data is available on the Web, RDF has become an important form of data. To fully utilize such structural knowledge has been a significant topic. Although SPARQL can handle queries over RDF data well, untrained ordinary users often find it hard to freely use it due to the complexity of SPARQL grammar and RDF schema. Thus, in terms of NLP and database, natural language QA systems have drawn great attention. Therefore, we developed gAnswer. It is able to transfer natural language questions into query graphs containing semantic information, and then turn the query graphs to standard SPARQL queries, execute in graph databases, and get the final answers. What's worth mentioning is that in the procedure of query graph generation, we keep the ambiguity in natural language, which will be dealt with later in answer generation procedure. In all, gAnswer has the following 3 features.

1. gAnswer combines semantic disambiguation with query parsing.
2. gAnswer is based on subgraph matching to get answers.
3. gAnswer comes up with a graph-based data mining approach and generate a predicate dictionary as support information for the system.

2.3.2 Open source and licence

gAnswer is an open source project under BSD license. You can freely download gAnswer, make suggestions and report bugs, or join us to make gAnswer better. On the premise of respect for our original work, you can develop all sorts of application using gAnswer.

2.4 Install

2.4.1 Package dependency

In the current version, gAnswer needs the following jar packages:

commons-codec-1.3.jar
commons-httpclient.jar
commons-logging.jar
GstoreJavaAPI.jar
jetty-all-9.0.4.v20130625.jar
json.jar
liblinear-1.8.jar
libsvm.jar
log4j.jar
lucene-core-2.0.0.jar
maltparser-1.9.1.jar
mysql-connector-java-5.1.7-bin.jar
Stanford-corenlp-1.3.4-models.jar
Stanford-corenlp-1.3.4.jar
xom.jar

2.4.2 External interface dependency

In the current version, gAnswer requires some external interface. In the open source project, you don't have to install these external systems on your computers thanks to our API. Of course, you can install required systems of your own

System name	Requirement	Location in project
gStore	version >= v0.7.0	qa.GAnswer.getAnswerFromGStore2()
DBpediaLookup		qa.mapping.DBpediaLookup

For better performance, we strongly recommend you to use your local-installed gStore and DBpediaLookup service.

If you need gStore, please check its github page here https://github.com/pkumod/gStore

If you need DBpediaLookup, please check here https://wiki.dbpedia.org/lookup/

To use your own gStore and DBpediaLookup, you should download our source code and modify the gStore and DBpediaLookup service IP and port number (the above table shows the exact location). In this case, you can merely run gAnswer in IDEs like Eclipse. If you insist the jar package deployment, you should export a new jar on your own. About how to create a jar for your project, please see Appendix A.

2.4.3 File dependency

In gAnswer, some support information is required. They should be stored on disk and loaded to main memory after system initialization. You can download these files freely

文件/文件夹	描述
16predicate_id	Mapping from predicate to id in RDF data
16entity_id	Mapping from entities to id in RDF data
16basic_types_id	Mapping from rdf types to id in RDF data
16yago_types_list	List of yago type in RDF data
16type_id_all	Mapping from both rdf and yago type to id in RDF data
16type_fragment	Collection of related entities for all basic types
16entitiy_fragment	Collection of related entities and predicate for all entities
16predicate_fragment	Collection of acceptable type for all predicates
16dbo_predicates	List of dbo predicates in RDF data
stopEntDict	Hand-made list of useless entity names
dbpedia-relation-paraphrases-withScorebaseform-merge-sortedrerank-slct	A dictionary mapping standard predicates in RDF data to natural language patterns

2.5 How to use

2.5.1 Data format

Currently, the system only support gAnswerHttp API, which allows you to interact with gAnswer via http request and json-formatted data. For more specific data format, please check *Chapter 3.1.2*.

2.5.2 gAnswerHttp

gAnswerHttp is developed based on jetty. It is a light, embedded http server. Having started gAnswerHttp in your console or terminal, you can fetch system-generated sparql and answer to given question through http requests.

This is how you activate gAnswerHttp: Switch to the location of the project and input *java -jar Ganswer.jar* (There is a detailed example on README of our github page). The default port is 9999. If you want to use another port, like 8888, please input *java -jar Ganswer.jar port=8888*.

3 Advanced

3.1 HTTP API Introduction

3.1.1 Example

In this part, a simple example of using gAnswerHttp API will be given.

Having successfully activate gAnswerHttp, you should construct a piece of json data looks like the following instance:

```
{
    “maxAnswerNum”: “3”
    “maxSparqlNum”: “2”
    “question”: “Who is the wife of Ming Yao?”
}
```

Above json data indicates that, we are asking gAnswer to answer " Who is the wife of Ming Yao?", and we need at most 3 different answers and only 1 SPARQL.

Next, transfer the json data to a string, conduct url encoding and access gAnswer via uri like this: *ip:port/gSolve/?data=%json string%.* In this example, we are actually accessing the following uri:

http://ip:port/gSolve/?data={maxAnswerNum:3, maxSparqlNum:1,question:Who is the wife of Ming Yao?}

If you neglect maxAnswerNum and maxSparqlNum, the system will use a default value (100 for maxAnswerNum and 5 for maxSparqlNum). If the system returns a successful result, it will also be in json. For example:

```
{
    "status": "200"
    "query": "Who is the wife of Ming Yao?"
    "vars" : [ "?wife" ]
    "sparql": [
    "select DISTINCT ?wife where { ?wife <spouse> <Yao_Ming>. } LIMIT 3"
    ]
    "results":{
     "bindings" : [
       {
            "?wife" :
            {
             "type" :  "uri"
             "value":  "<Ye_Li>"
            }
       },
       ]
    },
}
```

What's worth mentioning is that in the above example data, "vars" means the identified variable word in the question, "result" contains the actual answers, and "status" indicates that this is a successful return.

If any errors occur, the system still returns json data instead of crush or throw exceptions:

{
“query”: “”,
“message”: “UnvalidQuestionException: the question you input is invalid, please check”,
“status”: “500”
}

This is the json you will get for an invalid question (too short), in which “message” tells you the information about this exception, and “status” is the error code.

3.1.2 API structure

gSolve

uri

ip:port/gSolve/

function

Ask gAnswer for SPARQL and answers to a question. The number of SPARQL and answer is configurable.

Input

{
“maxAnswerNum”: // An integer indicating the maximum of return answers
“maxSparqlNum”: //An integer indicating the maximum of return SPARQL
“question”: //User question
}

Output

{
“status”: // status code of user request

“question”: // user natural language question

“vars”: // Identified variable in the user question

“sparql”: [… , … , …] // Generated SPARQLs

“results”:{ // Collection of answers

“bindings”:[

{

“?x”:{ // The key will be the same as identified variable name.

“type”: // Type of the answer, may be literal or uri

“value”: // The exact value of the answer

}

}

]

}

}

getInfo

uri

ip:port/getInfo/

Function

To get status and metadata of currently running gAnswer system.

Input

Not needed

Output

{

“version”:

“dataset”:

“GDB system”: //Currently used gStore version

}

3.2 Publication

[1]. Sen Hu, Lei Zou, Haixun Wang, Jeffrey Xu Yu, Wenqiang He: Answering Natural Language Questions by Subgraph Matching over Knowledge Graphs. IEEE TKDE 2017 [pdf]

[2]. Shuo Han, Lei Zou, Jeffrey Xu Yu, Dongyan Zhao: Keyword Search on RDF Graphs - A Query Graph Assembly Approach. CIKM 2017: 227-236 [pdf]

[3]. Lei Zou, Ruizhe Huang, Haixun Wang, Jeffrey Xu Yu, Wenqiang He, Dongyan Zhao: Natural language question answering over RDF: a graph data driven approach. SIGMOD Conference 2014: 313-324 [pdf]

[4]. Ruizhe Huang, Lei Zou: Natural language question answering over RDF data. SIGMOD Conference 2013: 1289-1290 (undergraudate student's poster) [pdf]

[5]. Weiguo Zheng, Lei Zou, Xiang Lian, Jeffrey Xu Yu, Shaoxu Song, Dongyan Zhao. How to Build Templates for RDF Question/Answering: An Uncertain Graph Similarity Join Approach SIGMOD Conference, 2015 (to appear). [pdf]

3.3 FAQ

4 Others

4.1 Contributors

4.1.1 Leader

Lei Zou（Peking University） Leader of the project

4.1.2 Student

Sen Hu（Peking University） Doctor

Yinnian Lin（Peking University） Master

4.2 Update Log

v 0.1.2
Simplify the parameters in the HTTP request.
Support PORT designation when running Ganswer.jar.
Fix the bugs in Ganswer Handler.
Support DBpedia 2016 subset.
[KEY] Fix the bugs of wrong port number

v 0.1.0
The first version

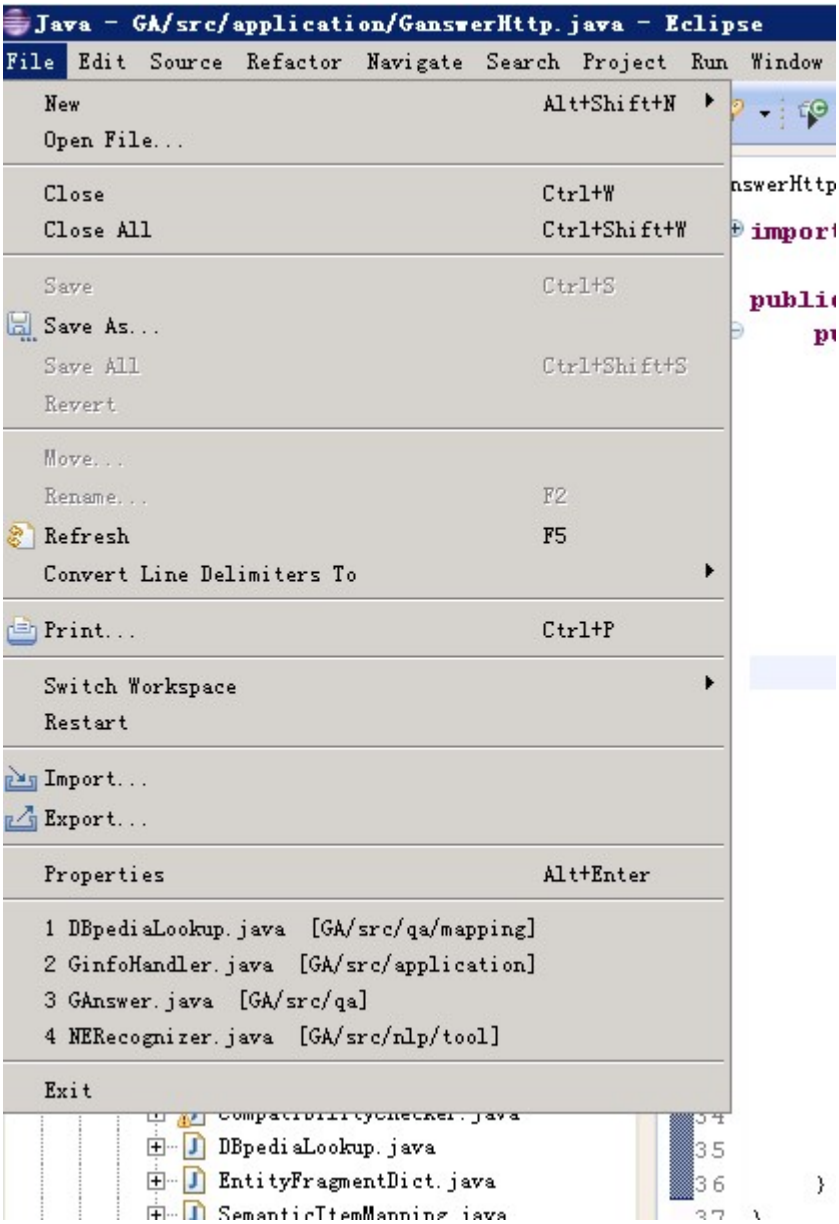
4.3 Legal Issues

CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.
What's more, you need to include the label "powered by gAnswer", as well as the logo of gAnswer, in your software product which is using gAnswer.
We would be very grateful if you are willing to tell us about your name, institution, purpose and email. Such information can be sent to us by emailing to linyinnian@pku.edu.cn, and we promise not to reveal privacy.

Appendix A：Export jar in Eclipse

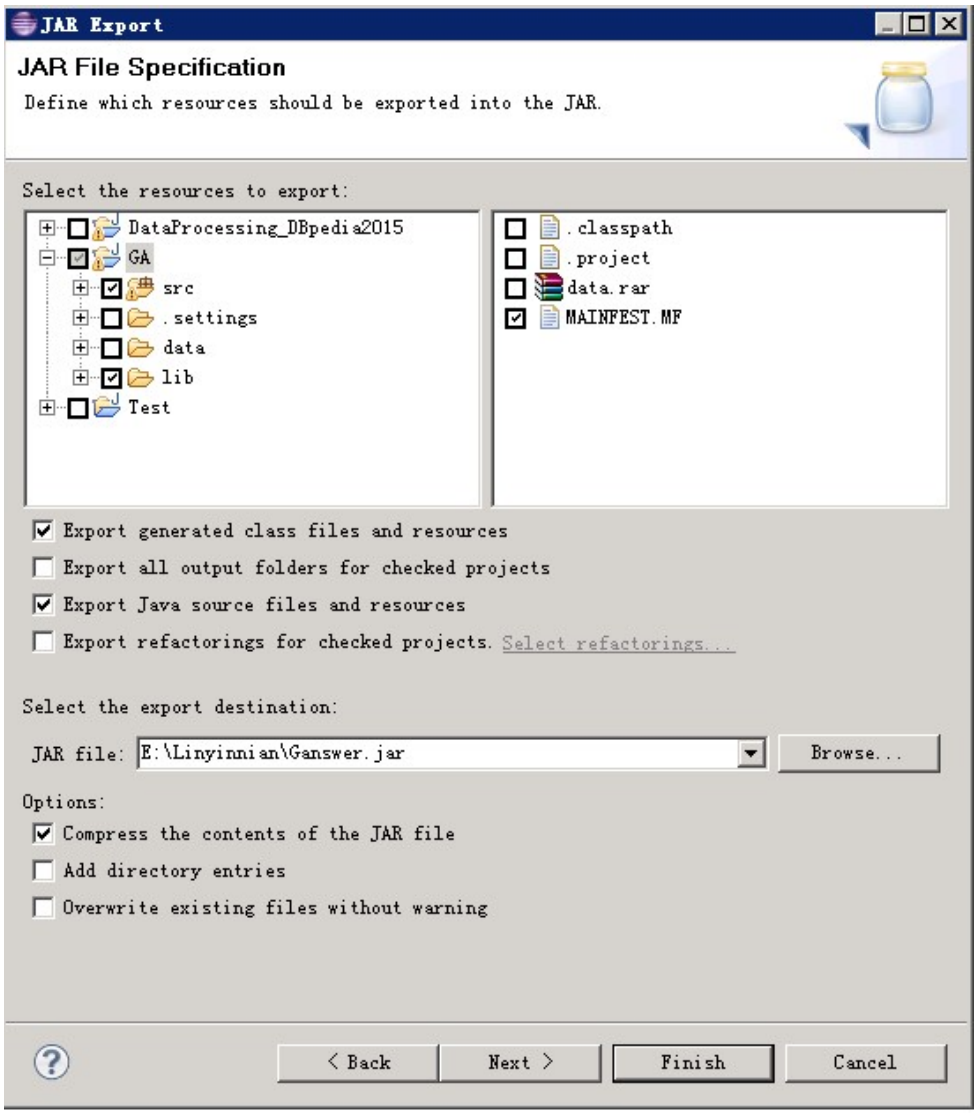
If you need to export jar in eclipse, the following instructions will help：

1. Under the project, click File->Export in the navigator menu.



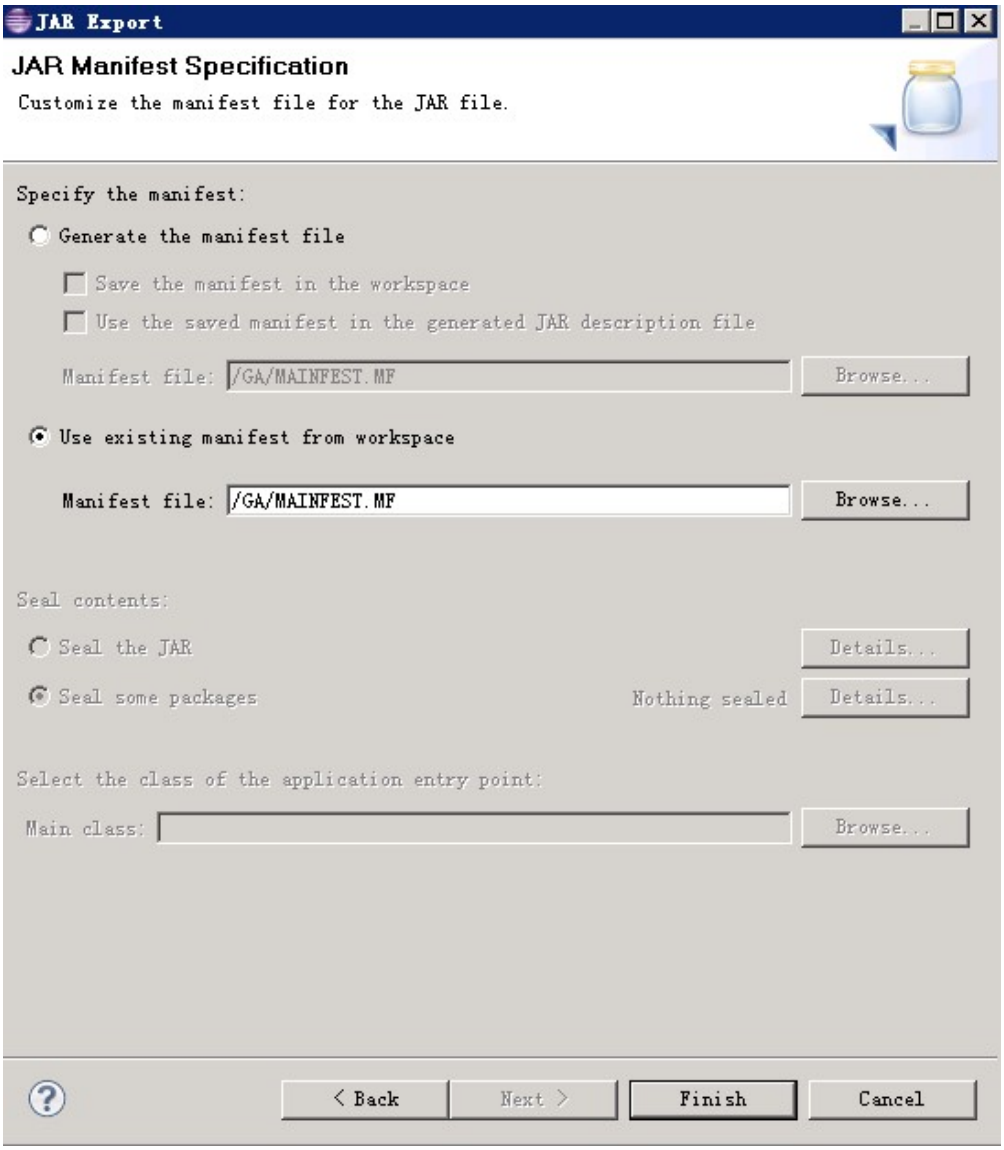
Attention, at this moment all the external jar should be put in “lib” folder.

2. Choose “jar file” to export. Then you will see the beneath dialog box. Select the content that you would like to export. Please ensure that you select “src” and “lib” folders. You can name your jar package here.



You may notice that “data” folder is not selected. It is because it’s too large, which may lead to failure in generating jar package. Meanwhile, MAINFEST.MF is not necessary here, either.

3. Click “Next” twice and you can see the JAR Manifest Specification box. Here you should tick “Use existing manifest from workspace” and select MAINFEST.MF as Manifest file. This file indicates the main class and classpath. If you have modified the main class or external jar, you must modify this file accordingly. The format of this file is with strict requirement. For more details, please check the official Java Help doc.



4. Click “Finish” and wait for a few minute, the generation will end.

Appendix B: how to preprocess rdf data

gAnswer depends on a set of supporting information to perform well. In this tutorial we will try to illustrate how to generate these supporting information using raw RDF data.

In the current version of gAnswer, we use subset of dbpedia 2016 as primitive data, which contains over 1.9 billion triples. The following files are extracted from dbpedia 2016 data:

ID files:

1. 16predicate_id.txt
2. 16entity_id.txt
3. 16basic_types_id.txt
4. 16yago_types_list.txt
5. 16type_id.txt

Fragment files:

1. 16entity_fragment.txt
2. 16type_fragment.txt
3. 16predicate_fragment.txt

Other files:

1. 16dbo_predicates.txt
2. stopEntDict.txt
3. dbpedia-relation-paraphrases-withScorebasefrom-merge-sortedrerank-slct.txt
4. predicate dictionary

These files are actually contained in Dbpedia16.rar, which is free for you to download, but the predicate dictionary is special. The version we are using now is old and we are still figuring out a better way to generate predicate dictionary. The usage of predicate dictionary can be found in the gAnswer paper. This tutorial does not include how to create your own predicate dictionary.

In the following sections of this tutorial, we will show you how we extract ID files, Fragment files and Other files respectively. Besides, building lucene index for these files is also included.

ID files

The generation of ID files is very easy. They are actually mapping predicate's, entity's and type's name to integers and Fragment files are based on ID files. Therefore all you need to do is extracting all the predicates, entities and types and giving them an unique id. The format of id files looks like:

[entity/predicate/type name string] \t [integer]

Notice that “type” means entities that once serve as the object of predicate showing type information like <type> and <a> (usually there will be some prefix in the name of predicate like *http://www.w3.org/1999/02/22-rdf-syntax-ns* for convenience we ignore prefix here). For example, if there is a triple “<Barack_Obama> <type> <Person>”, then <Person> should be a type. In dbpedia 2016 data, we only use yago types and rdf types. In other words, we only accept predicate that ends with “type” and has prefix “yago:” or “http://www.w3.org/1999/02/22-rdf-syntax-ns”. In the id files mentioned above, basic type stands for rdf type.

Fragment files

We will give you a Java example on how to generate fragment files.

entity fragment

Entity fragment of a given entity A contains information of the predicates, entities and types related to A. A piece of entity fragment consists of 6 sections.

The first is the ID of the central entity. We use <Donald_Trump> as an example, supposing its id is 9999.

The second is the in edge and entity list. This means when the central entity is the object of a triple or say the edge points at the central entity, we records the predicate id and subject entity id to form a in edge and entity pair. For example, there is 2 triple “<Donald_Trump> <leader> <the_United_States>” and “<Donald_Trump> <birthPlace> <the_United_States>”, and the id of <leader> is 47, <birthPlace> is 99 and <the_United_States> is 8532. Then the first 2 section of entity fragment of <Donald_Trump> should be “9999 8532:47;99……”(There is a \t between the two section).

The third section is the out edge and entity list. This is contrast to the second section. Edges that originates from the central entity is recorded.

The forth and fifth sections are in edge list and out edge list. Here we only record the predicate id of in edge and out edge. So some of the predicate id in the second and the third section is also in these sections, but those predicates that relates to literal value is also included.

The last section is the type list, which records all the type id of the central entity.

We will give you a comprehensive example. If in the RDF data, triples involving <Donald_Trump> is as follows:
<Donald_Trump> <leader> <the_United_States>
<Donald_Trump> <birthPlace> <the_United_States>
<Donald_Trump> <father_of> <Ivanka_Trump>
<Jefferson_Trump> <father of> <Donald_Trump>
<Walton_college> <alumni> <Donald_Trump>
<Donald_Trump> <host> “Celebrity Apprentice”
<Donald_Trump> <type> <Person>
<Donald_Trump> <type> <US_citizen>
<Jefferson_Trump> <type> <Person>

<Ivanka_Trump> <type> <Person>
And the id is:
<Donald_Trump> 9999
<the_United_States> 8532
<Ivanka_Trump> 7891
<Jefferson_Trump>19723
<Walton_college> 5512
<Person> 4
<US_citizen> 56
<leader> 47
<birthplace> 99
<type> 135
<father_of> 78
<alumni> 326
<host> 17

Then the entity fragment of <Donald_Trump> should be:

9999 8532:47;99,7891:78 | 19723:78,5512:326 | 47,99,78,17 | 78,326 | 4,56

As you can see, sections are divided by ‘ | ’, and different edge or entity id is divided by ‘ , ‘. When there is more than 1 in/out edges related to the same entity, please make sure predicate number is divided by ‘ ; ’.

type fragment

Type fragment records the predicate and entity related to a given type. It has 4 sections.

The first section, of course, is the id of the given type.

The second is a list of in edge predicate id. If an edge **points to** an entity of the given type, then we consider it an in edge for this type.

The third is a list of out edge predicate id. If an edge **originate from** an entity of the given type, then we consider it an out edge for this type.

The last one is a list of entity that fall into the given type.

In the Trump example, type fragment of <Person> will be:

4 47,99,78,17 | 78,326 | 9999,19753,1891

Notice that the first 2 sections are divided by a \t.

predicate fragment

Predicate fragment records the types that a given predicate may accept as object or subject. When the object is not an entity, we consider it a “literal”.

In the Trump example, the predicate fragment of <father_of> should be:

[19723,9999] 78 [7891]

Other files

16dbo_predicates.txt is a list of predicates that has a dbo prefix. It is useful when you use dbpedia data.

stopEntDict.txt is a hand-made list of incorrect entity name. Entity in this list will not be recognize as an entity. We use this file to get rid of some troublesome false entity names.

dbpedia-relation-paraphrases-withScorebasefrom-merge-sortedrerank-slct.txt is a little bit tricky. When we test gAnswer we collect information of failure and non-ideal results. Sometimes they are due to unsuccessful predicate recognition. So we basically use hand-write pattern to enhance our system's performance when it comes to predicate recognition, which is very important.

Building Lucene index

When you have fragment files as well as id files, you need to build a Lucene index. Related code is in our project. You can find them under src/lcn. Basically, if you generate fragment files and id files correctly, just running the code for Lucene index building is all you need to do.