CHAIN DOCUMENTATION

TABLE OF CONTENTS

OBJECTIVES OF CHAIN

The CHAIn (Combining Heterogeneous Agencies' Information) system dynamically re-writes queries to databases when mismatches led to query failure.

In general, queries to databases only succeed if they are correctly writing according to the schema of that database, which is only possible if the schema of the database is known in advance. During an emergency response (and, indeed, many other situations), this is plausible in some situations, but emergencies are characterised by their unpredictability and such foresight is not always possible. If one wishes to broadcast a query to all relevant agencies, or to a new agency, or to an previous collaborator who has updated their data sources, for example, a query may fail even if there is relevant data in the data source.

CHAIn sits locally within an agency. When a query to the data sources of that agency fails, CHAIn will use various matching techniques to determine on-the-fly whether there is mismatch between the schema of the query and the schema of the database and, if so, whether there is anything in the database that matches, or approximately matches the query. If so, CHAIn rewrites the query according to the schema of the database and sends back an appropriate response (or responses), together with a score indicating how good the match is and information about the assumptions and approximations made in the match.

RUNNING CHAIN

1. After downloading the CHAIn project, run the following four files that can be found in *spsm/s-match-source* called *0cloneSMatch, 2buildSMatch, 3createBinary, 4extractBinary.*

   Doing this should create the following folders in that directory: *s-match-parent, s-match-core, s-match-io, s-match-logic, s-match-nlp, s-match-nlp-annotation, s-match-nlp-opennlp, s-match-spsm, s-match-wordnet, s-match-examples, s-match-utils, s-match-spsm-prolog.*

2. Open Eclipse and select *File > New > Project > Java Project* and enter the name for the project. Uncheck the *Use Default Location* radio button and set the location to point to where this project was downloaded to and click *Next*.

3. Select the *Libraries* tab and then click on *Add External JARS*, navigate to where this project was downloaded and into *spsm/s-match/lib* and select all of the files in this directory. Then click on the *Add Library* button and make sure that the *JRE System Library* & *Junit* libraries have been added to the project.

4. Finally, click on *Finish*. You should now have access to all the source and testing files for this project.

FILE STRUCTURE

TBC

TESTING RESULT DOCUMENTS

All tests will write their results to a text file that can be found in the *outputs/testing* folder. These files have been structured to give details about each individual test including what schemas CHAIn is being run with, what the expected result is and what the actual result is that is returned.

Please note: There are some tests that will return empty results either because there have been no matches found between the source and target schema. Some tests will return the message, *Null Results,* due to SPSM not returning anything to the application after being called.

BREAKDOWN OF TESTING DOCUMENTS

Task 1 Testing Documents

- *SPSM_Test_Cases.java* (which produces *Call_SPSM_Tests.txt)*
- *Match_Results_Test_Cases.java* (which produces *Filter_Limit_Tests.txt)*
- *SPSM_Filter_Results_Test_Cases.java* (which produces *Task_1_Tests.txt)*

These files can be run at once by running *Task1_Test_Suite.java*

Task 2 Testing Documents

- *Repair_Schema_Test_Cases.java* (which produces *Schema_Repair_Tests.txt)*
- *Create_Query_Test_Cases.java* (which produces *Create_Queries_Tests.txt)*

DESCRIPTION OF IMPLEMENTATION FILES

MATCH_STRUC.JAVA

BASIC FUNCTIONALITY

Data structure that has been created to store the results that are returned after calling SPSM. The structure allows for easy interaction to find out information about the match details with a certain target schema.

PRIVATE VARIABLES/FIELDS

similarity

This is the variable that stores the similarity value between the source and this current target schema. (double)

dataset

This variable holds the name of the target schema that this structure is representing the results for. (String)

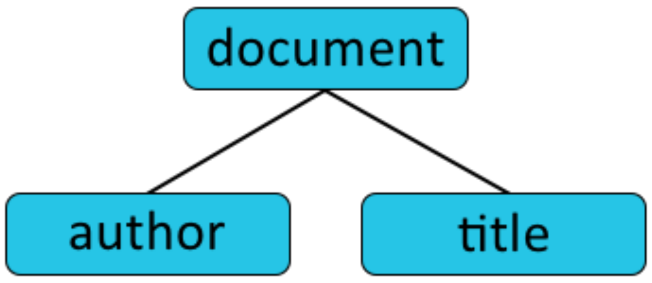
matches

List of arrays containing the match concepts between the source and this current target schema. (ArrayList of String arrays)

repairedSchemaTree

Tree structure that represents the repaired schema in a tree form. (Node)

For example: If the schema is *document(title,author)* then the tree will look like the following where *document* is the root node with two child nodes with the values *author* and *title*,



repairedSchema

This variable holds the repaired schema in a format that can easily be read. (String)

numMatches

Stores the total number matches which is calculated by the number of elements in the matches ArrayList. (int)

query

Stores the query that has been produced using the repaired schema (*String*)

METHODS

public Match_Struc (double sim, String targetSchema)

Constructor that is responsible for initialising the Match_Struc variable with the similarity value and the name of the target schema.

public void setSimilarity (double sim)

Sets the similarity of this match to be the value passed in as the parameter, *sim.*

public void setDatasetSchema (String targetSchema)

Sets the target schema to be the String that is passed in as the parameter, *targetSchema*.

public void addMatch (String[] match)

Responsible for adding another match concept to the list of matches stored in the ArrayList<String[]> called *matches.*

public int getNumMatches ()

Returns the total number of matches based on the number of elements inside *matches*.

public double getSimValue ()

Returns the similarity value for that target schema.

public String getDatasetSchema ()

Returns the name of the target schema.

public ArrayList<String[]> getMatches ()

Returns the ArrayList of arrays containing the match concept details.

public String[] getMatchAtIndex (int index)

Returns the specific match concept detail at the specified index passed as a parameter, *index*.

public void setRepairedSchemaTree (Node schemaTree)

Sets the repaired schema tree from the parameter, *schemaTree*.

public void setRepairedSchema (String stringSchema)

Sets the String version of the repaired schema from the parameter, *stringSchema*.

public String getRepairedSchema ()

Returns the String version of the repaired schema.

public Node getRepairedSchemaTree ()

Returns the repaired schema as a tree structure.

public void setQuery(String matchQuery)

Sets the query for this match structure.

public String getQuery()

Returns the query for this structure.

ASSOCIATED TESTING DOCUMENTS

SPSM_Test_Cases.java (which produces *Call_SPSM_Tests.txt*)

*Match_Results_Test_Cases.java (*which produces *Filter_Limit_Test.txt)*

SPSM_Filter_Results_Test_Cases.java (which produces *Task_1_Tests.txt)*

Task1_Test_Suite.java (which runs the testing files for task 1)

RUNNING THIS FILE

-

CALL_SPSM.JAVA

BASIC FUNCTIONALITY

This class is responsible for initially taking in both the target and source schemas before calling SPSM with these schemas. It will then store the results as an ArrayList of Match_Struc objects.

PRIVATE VARIABLES/FIELDS

targetList

Used temporarily for splitting the target schema String if there are more than one target schema within the String. This allows the application to then call SPSM for each individual target schema. (*String[]*)

METHODS

public ArrayList<Match_Struc> getSchemas (ArrayList<Match_Struc> results, String sourceSchema, String targetSchema)

Retrieve the schemas from the user, either through the console or as a parameter which gets written to their respective files. From there we can start to make a call to SPSM with the current target.

public ArrayList<Match_Struc> makeCallToSPSM (ArrayList<Match_Struc results, String currTarget)

Makes the call to SPSM using a bash file called *call-spsm.sh.* This should allow for SPSM to run and then write the results out as a serialised object to *serialised-results.ser*. It will then call the appropriate method to read this file and store the results in a Match_Struc object.

public ArrayList<Match_Struc> recordSerialisedResults (ArrayList<Match_Struc> results, String targetSchema)

Responsible for reading the serialised object from *serialised-results.ser* and calls the appropriate method that will then go through the different pieces of information stored within this object and stores what is required for CHAIn.

public ArrayList<Match_Struc> readObject (ArrayList<Match_Struc> results, String targetSchema, IContextMapping<INode> mapping)

Reads the information stored in the object read in and picks out the data that is required for use by CHAIn.

public String getNodePathString (INode node)

Responsible for taking in a *INode* object and turns it into something readable so that we can store the match concepts in a convenient way.

ASSOCIATED TESTING DOCUMENTS

SPSM_Test_Cases.java (which produces *Call_SPSM_Tests.txt*)

SPSM_Filter_Results_Test_Cases.java (which produces *Task_1_Tests.txt)*

Task1_Test_Suite.java (which runs the testing files for task 1)

RUNNING THIS FILE

Running this file will call SPSM with the source and target schemas that have been declared in the *main* method and will then print out the results that have been returned from SPSM by printing out the target schema, it's similarity to the source schema and the match concepts.

BEST_MATCH_RESULTS.JAVA

BASIC FUNCTIONALITY

Responsible for taking results after calling SPSM and then filtering to ensure each result is over the determined threshold, cut the number of results to the desired *n* and sort them so that the match with the highest similarity is at the top of the list.

PRIVATE VARIABLES/FIELDS

-

METHODS

public ArrayList<Match_Struc> getThresholdAndFilter (ArrayList<Match_Struc> results, Double threshVal, int limNum)

Start off by taking in the results, threshold value and the number of results wanted but if these values haven't been passed in through parameters, then ask the user to enter them through the console. It will then strip off any matches that are lower than the threshold value that has been entered.

public ArrayList<Match_Struc> sortResultingMatches (ArrayList<Match_Struc> filteredRes, int limitNo)

This method will sort the resulting matches after they have been filtered to ensure that only those greater than or equal to the threshold value are left. These matches are sorted so that the matches with the highest similarity value is at the top of the list.

public void displayResults (ArrayList<Match_Struc> res)

This method is used to print out the results after they have been filtered and sorted.

ASSOCIATED TESTING DOCUMENTS

Match_Results_Test_Cases.java (which produces *Filter_Limit_Tests.txt)*

SPSM_Filter_Results_Test_Cases.java (which produces *Task_1_Tests.txt)*

Task1_Test_Suite.java (which runs the testing files for task 1)

RUNNING THIS FILE

Running this file will use the Match_Struc variables in the list of results and filter the list to ensure that anything in the list if over a specified threshold and that there are n number of entries if specified by the user. The *main* method will then print out each element in the list after filtering and sorting the list.

REPAIR_SCHEMA.JAVA

BASIC FUNCTIONALITY

Responsible for taking in an ArrayList of Match_Struc objects and going through each one and creating a repaired schema based on the relation between the original source and target schemas.

PRIVATE VARIABLES/FIELDS

-

METHODS

public ArrayList<Match_Struc> prepare (ArrayList<Match_Struc> matchRes)

Takes in the ArrayList of objects and starts processing them in turn to create a repaired schema.

public Match_Struc repairSchema(Match_Struc matchStructure)

Creates a tree structure of the schema and adds that as a field to the Match_Struc object based on the relation between the source and target schemas. It also adds a String version of the repaired schema as a field to the Match_Struc object.

public Node modifyRepairedTree(Node root, String concept)

Using the relation concept data, the tree is modified to ensure that there are only links to parts of the schema that we think there is a match with. This tree is then returned.

ASSOCIATED TESTING DOCUMENTS

Repair_Schema_Test_Cases.java (which produces *Schema_Repair_Tests.txt*)

Create_Query_Test_Cases.java (which produces *Create_Queries_Tests.txt*)

RUNNING THIS FILE

Running this file will go take in the source and target schemas, call SPSM and then create the repaired schema based on the match details between these two original schemas. The results will be printed to the console.

NODE.JAVA

BASIC FUNCTIONALITY

Structure that is being used to store the repaired schema tree that consists of a node that has a value and also a list of children nodes.

PRIVATE VARIABLES/FIELDS

value

Holds the value of the node such as the predicate or parameter name (*String*)

children

The list of nodes that are the children of the current node (*List<Node>*)

METHODS

public Node(String val)

The constructor that creates the node structure and sets the value of that node.

public void addChild(Node child)

Adds the node, *child*, to the list of children nodes.

public String getValue()

Returns the value of the current node as a String.

public List<Node> getChildren()

Returns the list of children nodes.

public ArrayList<String> getChildrenValues()

Returns a list containing the values of all the children nodes.

public Boolean hasChildren()

Returns whether or not the node has any children nodes.

public int getNumChildren()

Returns the total number of children nodes.

public String printTree()

Prints out the tree as a repaired schema.

public String printChildren(List<Node> childrenList)

Prints out the children in the tree to create the repaired schema as a *String*.

public void addToTree(String[] paramParts)

Modifies the tree to add a list of parameters correctly into the current tree.

ASSOCIATED TESTING DOCUMENTS

Repair_Schema_Test_Cases.java (which produces *Schema_Repair_Tests.txt)*

RUNNING THIS FILE

-

ONTOLOGY_STRUC.JAVA

BASIC FUNCTIONALITY

Structure that is used to store the ontology structure being used to create the queries.

PRIVATE VARIABLES/FIELDS

name

The name of the ontology structure (*String*)

link

The web link of the ontology structure (*String*)

properties

The list of key words or properties associated with the structure (*String[]*)

METHODS

public Ontology_Struc(String ontName, String ontLink, String[] ontProperties)

Constructor for creating an ontology structure.

public void setName(String ontName)

Sets the name of the structure.

public String getName()

Returns the name of the structure.

public void setLink(String ontLink)

Sets the link for the structure.

public String getLink()

Returns the link of the structure.

public void setProperties(String[] ontProperties)

Sets the properties or key words associated with the structure.

public String[] getProperties()

Returns the list of key words or properties associated with the structure.

ASSOCIATED TESTING DOCUMENTS

Create_Query_Test_Cases.java (which produces *Create_Queries_Tests.txt*)

RUNNING THIS FILE

-

CREATE_QUERY.JAVA

BASIC FUNCTIONALITY

Responsible for using the repaired schema to create either a sepa or dbpedia query, this query is then added as a field onto the Match_Struc object structure.

PRIVATE VARIABLES/FIELDS

-

METHODS

public ArrayList<Match_Struc> createQueryPrep(ArrayList<Match_Struc> matchRes, String queryType, String additionalInfo)

Sets up so that the methods can create a query for each of the repaired schemas in the list of Match_Struc objects.

public String createSepaQuery(Match_Struc matchDetails, String datafileDir)

Creates structure for a local sepa query.

public String createDbpediaQuery(Match_Struc matchDetails)

Creates structure for a remote dbpedia query.

public ArrayList<Ontology_Struc> setupPrefixes(JSONArray jsonArr, ArrayList<Ontology_Struc> ontologies)

Reads ontology file and processes it into Ontology_Struc objects.

public String writePrefix(ArrayList<Ontology_Struc> ontologies)

Writes prefixes for query through reading the Ontology_Struc objects.

public String dataMatching(List<Node> children, ArrayList<Ontology_Struc> ontologies)

Finds what ontology file a specific key word or property is from or relates to.

public void runSepaQuery(String query, String datasetToUseDir, Match_Struc currMatchStruc)

Runs a local sepa query.

public void runDbpediaQuery(String query, Match_Struc currMatchStruc)

Runs a remote dbpedia query.

ASSOCIATED TESTING DOCUMENTS

Create_Query_Test_Cases.txt (which produces *Create_Queries_Tests.txt*)

RUNNING THIS FILE

Running this file will either setup and run a sepa or dbpedia query based on the repaired schema produced from calling SPSM on the target and source schemas. These details are specified in the *main* method. It also prints out the repaired schema, the query string created and the results from calling this query.