Table of Contents

Syntethic Dataset

```
CREATE TABLE `dataset` (
  `dataset_id` int(11) NOT NULL AUTO_INCREMENT,
  `num_of_sources` int(11) NOT NULL,
  `num_of_data_item` int(11) NOT NULL,
  `coverage` double NOT NULL DEFAULT '0',
  `num_of_different_values` int(11) NOT NULL,
  `dependency_id` int(11) NOT NULL,
  `similarity_id` int(11) NOT NULL,
  `control_num_of_src_per_value_method_id` int(11) NOT NULL,
  `control_src_true_value_method_id` int(11) NOT NULL,
  `di_different_values_method_id` int(11) NOT NULL,
  `dataset_file_name` char(255) NOT NULL,
  `num_of_sources_per_value` int(11) NOT NULL,
  `percentage_true_val_per_src` float NOT NULL,
  `num_of_independent_sources` int(11) NOT NULL DEFAULT '-1',
  `percentage_of_copied_values` float NOT NULL DEFAULT '0',
  `data_items_coverage_method_id` int(11) NOT NULL DEFAULT '1',
  PRIMARY KEY (`dataset_id`),
  KEY `dependency_id` (`dependency_id`),
  KEY `similarity_id` (`similarity_id`),
  KEY `control_num_of_src_per_value_method_id` (`control_num_of_src_per_value_method_id`),
  KEY `control_src_true_value_method_id` (`control_src_true_value_method_id`),
  KEY `di_different_values_method_id` (`di_different_values_method_id`),
  CONSTRAINT `dataset_ibfk_1` FOREIGN KEY (`dependency_id`) REFERENCES `dependency` (`id`),
  CONSTRAINT `dataset_ibfk_2` FOREIGN KEY (`similarity_id`) REFERENCES `similarity` (`id`),
  CONSTRAINT `dataset_ibfk_3` FOREIGN KEY (`control_num_of_src_per_value_method_id`) REFERENCES
`control_num_of_src_per_value_method` (`id`),
  CONSTRAINT `dataset_ibfk_4` FOREIGN KEY (`control_src_true_value_method_id`) REFERENCES
`control_src_true_value_method` (`id`),
  CONSTRAINT `dataset_ibfk_5` FOREIGN KEY (`di_different_values_method_id`) REFERENCES
`di_different_values_method` (`id`)
) ENGINE=InnoDB AUTO_INCREMENT=49477 DEFAULT CHARSET=latin1 COMMENT='latin1_swedish_ci'
```

Dataset Database Table Content

- **dataset_id:** Unique ID for each dataset.
- **num_of_sources:** The number of sources in this dataset.
- **num_of_data_item:** The number of data items in this dataset.
- **Coverage:** The uniform coverage value. If the **data_items_coverage_method_id** is not Uniform (= 1), *this value is not used*.
- **num_of_different_values:** The number of distinct (different) values provided for each data item in the dataset. When the **di_different_values_method_id** is Uniform (=1), this value is the amount of distinct value for all data item. If Exponential or 80-20, this value is the maximum amount of distinct values over all dta items.
- **dependency_id:** Whether to control dependency (=1) or not (=- 1) between sources. If dependency is considered: **num_of_different_values, di_different_values_method_id** are not taken into consideration.
- **similarity_id:** whether the distinct value for each data items are highly similar (=-1) or highly dissSimilar (=1).
- **control_num_of_src_per_value_method_id:** Not implemented yet.
- **control_src_true_value_method_id:** Usually considered.
- **di_different_values_method_id:** The distribution of the distinct values over the data items.

- **dataset_file_name:** The folder name of the dataset on the hard disk, under the DAFNAData/formatted/synthetic path.
- **num_of_sources_per_value:** Not used as not implemented yet.
- **percentage_true_val_per_src:** Only considered when Uniform (=1) **control_src_true_value_method_id.**
- **num_of_independent_sources:** The number of independent sources among the set of sources in the dataset. Only considered when sources dependencies are considered.
- **percentage_of_copied_values:** The percentage of copied claims a copier source will identically copy. Only considered when sources dependencies are considered.
- **data_items_coverage_method_id:** The way the number of data items is distributed over the sources.

Create a Synthetic Dataset:

go to **DataSetCollector** Project, to
qcri.dafna.syntheticDataset package, to
CreateSyntheticDataset Class

CreateSyntheticDataset is the class responsible for All datasets creation.

The class attributes:

```
int numOfSources = 50;
int numOfObjects = 200;
int numOfProperties = 5;
int numOfSourcesPerValue = -1;
int dependencyID =Globals.controlSourcesDependency_Nocontrol;
int similarityID = Globals.valueSimilarity_dissSimilar;
```

These attributes determine the dataset characteristics that can hold only one value (i.e. all datasets will have this same numOfSources …).
The remaining dataset charcteristincs can be set in the internal objects in the method "**startCreateDatasets**()" :

- **List<Integer> controlDICoverageMethodID_list**

Contains the different ways for the source-coverage control:

- 1: Uniform Coverage.
- 2: Exponential Coverage.
- 3: Linear Coverage.

- **List<Double> coverage_List**

Contains the different desired values for Uniform Coverage *(from zero to one)*.
Note: *in case of uniform or linear coverage this list MUST contain only one entry value (prefered to be -1). If it contains multiple entries, the datasets will be generated multiple times with exponential coverage, but the database field for coverage will contains different meaningless values.*
If it contains zero-entry, no datasets will be generated.

- ***List<Double> percentageOfCopiedValues_List***

Contains the percentage (from zero to one) of the copied values by a copier-sources from a seed-source, all other copier-source claims are going to be Distinct And Wrong.
Note: *in case of Source Dependency NO-Control this list MUST contain only one entry value (prefered to be -1). If it contains multiple entries, the datasets will be generated multiple times with Source Dependency NO-Control, but the database field for percentage of copied values will contains different meaningless values.*
If it contains zero-entry, no datasets will be generated.

- **List<Integer> controlDistinctValueGenerationMethodID_List**

Contains the way to control the number of Distinct value:

- 1: Uniform-Constant
- 2: 80-20

- 3: Exponential

- **List<Integer> numOfDistinctValues_List**

Contains the number of Distinct value. In the Uniform-Constant model, this value is used uniformly over all data items. If Exponential of 80-20 models, this value is used as the maximum.

- **List<Integer> controlSrcTrueValMethod_List**

Contains the way to control the number of true values per sources:

- -1: No control
- 1: Uniform
- 2: Fully Pessimistic
- 3: Fully Optimistic
- 4: 80-Pessimistic
- 5: Exponential
- 6: 80-Optimistic

- **List<Double> percentageOfTrueValuePerSource_List**

Contains the percentage of true value per source used in the Uniform Model.
Note: If the uniform model is not choosed, the values in this list are neglicted. If the Uniform mpdel is used and this list is empty: No datasets will be generated for the Uniform model.

- **List<Double> independentSrcPercentage_List**

Contains the percentage (from zero to one) of the number of independent sources amoung the whole number of sources in the dataset.
Note: *in case of Source Dependency NO-Control this list MUST contain only one entry value (prefered to be -1). If it contains multiple entries, the datasets will be generated multiple times with Source Dependency NO-Control, but the database field for percentage of independent sources will contains different meaningless values.*
If it contains zero-entry, no datasets will be generated.

- **int maxNumOfDiffdataSet = 10;**

this value limit the maximum number of dataset created for EACH same characterization.

Storage

Each created datasets is stored under the “synthetic” folder under the “formatted” folder, under the “DAFNAData” folder.
This path is customizable in the Globals class from the QCRITruthDiscovery project.
Note: The database only contain the dataset folder name, not the full path.

- The dataset folder name is chosen as the current time in millisecond generated by Java at the time of creating the dataset.

All Schema

Datasets Table

```
CREATE TABLE `dataset` (
  `dataset_id` int(11) NOT NULL AUTO_INCREMENT,
  `num_of_sources` int(11) NOT NULL,
  `num_of_data_item` int(11) NOT NULL,
  `coverage` double NOT NULL DEFAULT '0',
  `num_of_different_values` int(11) NOT NULL,
  `dependency_id` int(11) NOT NULL,
  `similarity_id` int(11) NOT NULL,
  `control_num_of_src_per_value_method_id` int(11) NOT NULL,
  `control_src_true_value_method_id` int(11) NOT NULL,
  `di_different_values_method_id` int(11) NOT NULL,
  `dataset_file_name` char(255) NOT NULL,
  `num_of_sources_per_value` int(11) NOT NULL,
  `percentage_true_val_per_src` float NOT NULL,
  `num_of_independent_sources` int(11) NOT NULL DEFAULT '-1',
  `percentage_of_copied_values` float NOT NULL DEFAULT '0',
  `data_items_coverage_method_id` int(11) NOT NULL DEFAULT '1',
  PRIMARY KEY (`dataset_id`),
  KEY `dependency_id` (`dependency_id`),
  KEY `similarity_id` (`similarity_id`),
  KEY `control_num_of_src_per_value_method_id` (`control_num_of_src_per_value_method_id`),
  KEY `control_src_true_value_method_id` (`control_src_true_value_method_id`),
  KEY `di_different_values_method_id` (`di_different_values_method_id`),
  CONSTRAINT `dataset_ibfk_1` FOREIGN KEY (`dependency_id`) REFERENCES `dependency` (`id`),
  CONSTRAINT `dataset_ibfk_2` FOREIGN KEY (`similarity_id`) REFERENCES `similarity` (`id`),
  CONSTRAINT `dataset_ibfk_3` FOREIGN KEY (`control_num_of_src_per_value_method_id`) REFERENCES
`control_num_of_src_per_value_method` (`id`),
  CONSTRAINT `dataset_ibfk_4` FOREIGN KEY (`control_src_true_value_method_id`) REFERENCES
`control_src_true_value_method` (`id`),
  CONSTRAINT `dataset_ibfk_5` FOREIGN KEY (`di_different_values_method_id`) REFERENCES
`di_different_values_method` (`id`)
) ENGINE=InnoDB AUTO_INCREMENT=49477 DEFAULT CHARSET=latin1 COMMENT='latin1_swedish_ci'
```

Experiment_results Table

```
CREATE TABLE `experiment_results` (
  `experiment_results_id` int(11) NOT NULL AUTO_INCREMENT,
  `dataset_id` int(11) NOT NULL,
  `voter_name` char(35) NOT NULL,
  `precision` double NOT NULL,
  `accuracy` double NOT NULL,
  `recall` double NOT NULL,
  `specificity` double NOT NULL,
  `number_of_iteration` int(11) NOT NULL,
  `voter_duration_ms` int(11) NOT NULL,
  PRIMARY KEY (`experiment_results_id`)
) ENGINE=InnoDB AUTO_INCREMENT=419090 DEFAULT CHARSET=latin1 COMMENT='latin1_swedish_ci'
```

contengency Table

```
CREATE TABLE `contengency` (
  `contengency_id` int(11) NOT NULL AUTO_INCREMENT,
  `dataset_id` int(11) NOT NULL,
  `dataItem_key` char(100) NOT NULL, (objectId+PropertyId)
  `truthFinder_value` char(100) NOT NULL DEFAULT '',
  `cosine_value` char(100) NOT NULL DEFAULT '',
  `twoEstimates_value` char(100) NOT NULL DEFAULT '',
  `threeEstimates_value` char(100) NOT NULL DEFAULT '',
  `simpleLCA_value` char(100) NOT NULL DEFAULT '',
  `guessLCA_value` char(100) NOT NULL DEFAULT '',
  `copySimAccu_value` char(100) NOT NULL DEFAULT '',
  `copyNoSimNoAccu_value` char(100) NOT NULL DEFAULT '',
  `copyNoSimAccu_value` char(100) NOT NULL DEFAULT '',
  `copySimNoAccu_value` char(100) NOT NULL DEFAULT '',
  `voting_value` char(100) NOT NULL DEFAULT '',
  `LTM_value` char(100) NOT NULL DEFAULT '',
  `MLE_value` char(100) NOT NULL DEFAULT '',
  PRIMARY KEY (`contengency_id`)
) ENGINE=InnoDB AUTO_INCREMENT=16102729 DEFAULT CHARSET=latin1 COMMENT='latin1_swedish_ci'
```

All the next look-up tables are not used in the implementation. Only for reference.

control_src_true_value_method lookup-Table

```
CREATE TABLE `control_src_true_value_method` (
  `id` int(11) NOT NULL,
  `name` varchar(255) NOT NULL,
  `info` varchar(255) NOT NULL,
  PRIMARY KEY (`id`)
) ENGINE=InnoDB DEFAULT CHARSET=latin1
```

control_num_of_src_per_value_method lookup-Table

```
CREATE TABLE `control_num_of_src_per_value_method` (
  `id` int(11) NOT NULL,
  `name` varchar(255) NOT NULL,
  `info` varchar(255) NOT NULL,
  PRIMARY KEY (`id`)
) ENGINE=InnoDB DEFAULT CHARSET=latin1
```

dependency lookup-Table

```
CREATE TABLE `dependency` (
  `id` int(11) NOT NULL,
  `name` varchar(255) NOT NULL,
  `info` varchar(255) NOT NULL,
  PRIMARY KEY (`id`)
) ENGINE=InnoDB DEFAULT CHARSET=latin1
```

di_different_values_method lookup-Table

```
CREATE TABLE `di_different_values_method` (
  `id` int(11) NOT NULL,
  `name` varchar(255) NOT NULL,
  `info` varchar(255) NOT NULL,
  PRIMARY KEY (`id`)
) ENGINE=InnoDB DEFAULT CHARSET=latin1
```

similarity lookup-Table

```
CREATE TABLE `similarity` (
  `id` int(11) NOT NULL,
  `name` varchar(255) NOT NULL,
  `info` varchar(255) NOT NULL,
  PRIMARY KEY (`id`)
) ENGINE=InnoDB DEFAULT CHARSET=latin1
```

Go to ***QCRITruthDiscovery*** Project

in the **qcri.dafna.experiment** package, there is the main class for launching the synthetic datasets experiment: **SyntheticExperiment**.

The main steps for the synthetic datasets experiment are:

1. we generate an SQL query, requesting the datasets file name (and Id) for a specific set of synthetic datasets from the database.
2. Then apply the Experiments, get the results for each dataset.
3. Finally, write these results to the database.

Get the Required Datasets

The **getSyntheticExperimentSelectQuery**() return the select query that is used to query the database asking for the datasets filename and id.
The SQL query should specify all required dataset characteristic as:

num_of_sources
num_of_data_item
data_items_coverage_method_id
coverage
di_different_values_method_id
num_of_different_values
control_num_of_src_per_value_method_id
num_of_sources_per_value
control_src_true_value_method_id
percentage_true_val_per_src
similarity_id
dependency_id
num_of_independent_sources
percentage_of_copied_values

Note: If a wider set of datasets are to be involved in an experiment, any line can be commented in the SQL query.

One final important part is added to the SQL query:
If the experiment_results table already contains a specific dataset's results, the experiment will not be re-run on this dataset:
"dataset_id NOT IN (SELECT dataset_id from experiment_results)"

Apply the experiments to the datasets

The second step is simply applying the set of experiments to the selected datasets.

Go tp ***QCRITruthDiscovery*** Project
in the **qcri.dafna.experiment** package, there is the main class for launching any dataset experiment:

Experiment.java : Include the set of models to be run on a given dataset.

1. Each model parameter should be set here, in the **runExperiment()** method.
2. Two boolean has to be set for the experiment:
    1. **static boolean** *logExperimentName***:** Whether to log the experiment results to the standard output or not.
    2. **static boolean** *profileMEmory: Whether to profile the memory usage or not.*

Note: When memory usage is profiled, the experiment takes longer time, but the memory consumption logged value is accurate. When the memory usage is not profiled, the experiment processing time is accurate but the logged memory consumption is not a valid value.

3. The **runExperiment()** method returns a hashMap, with the Key=VoterName, and Value=VoterQualityMeasures-Object.
4. The **runExperiment()** method creates 3-Files in the resultFolderName-Folder (method parameter)
    1. dataSetInfo.txt: The dataset characteristics
    2. VoterQuality.csv: Each voter quality measures, in the next format:

"**VoterName** & **Precision** & **Accuracy** & **Recall** & **Specificity** & **N. of Iterations** & **Voter duration** & **Memory Consumtion(MB)** & \n";
The "&" delimiter was chosen to be ready for use in a TEX file.
Th

3. ParametersPerIteration.csv: This file has exactly the same format as the voterQuality.csv, but it contains an entry for each voter for every iteration.

Note: These files are generated using the class : **DataQualityLogger** in the package: **qcri.dafna.dataModel.quality.dataQuality.logger**

the **experiment_results** table is the table that contains results for all voter-experiments for all datasets.

Note: When experiment is launched on a dataset, ALL voters are applied to the dataset then ALL results are written all together to the database.

The experiment_results schema:

```
CREATE TABLE `experiment_results` (
  `experiment_results_id` int(11) NOT NULL AUTO_INCREMENT,
  `dataset_id` int(11) NOT NULL,
  `voter_name` char(35) NOT NULL,
  `precision` double NOT NULL,
  `accuracy` double NOT NULL,
  `recall` double NOT NULL,
  `specificity` double NOT NULL,
  `number_of_iteration` int(11) NOT NULL,
  `voter_duration_ms` int(11) NOT NULL,
  PRIMARY KEY (`experiment_results_id`)
) ENGINE=InnoDB AUTO_INCREMENT=419090 DEFAULT CHARSET=latin1
COMMENT='latin1_swedish_ci'
```

A tuple in the experiment_results table contains:
`**experiment_results_id**`: unique ID for this row.
`**dataset_id**`: The dataset on which this experiment took place.
`**voter_name**`: The voter applied for the (dataset_id) conducting this results
`**precision**`:The voter precision over this dataset.
`**accuracy**`:The voter accuracy over this dataset.
`**recall**`:The voter recall over this dataset.
`**specificity**`:The voter specificity over this dataset.
`**number_of_iteration**`: Number of iterations accomplished by this voter.
`**voter_duration_ms**`: The voter duration in milliseconds.

Charts

Draw Synthetic datasets Charts using GNUPLOT

Go to **datasetCollector** Project
in the **qcri.dafna.database.chart.GNUPLOT** package:
The **ExperimentAverageChartDrawer** class is responsible to launch a chart drawer.

4 options of graphs:

1. Plot the sources-dependency control datasets: The X-Axe contains the number of Independent sources.
2. Plot the datasets (with no dependency control): The X-Axe contains the number of distinct values
3. Plot the models precision: Te X-Axe contains the models name(Majority, TuthFinder...)
4. The scalability experiment: This class is customized to plot the needed final plot.

***Note**:*
the method
plotModel(List<List<List<Double>>> seriesList, List<String> keys, String voterName, String xLabe, String yLabe, List<HashMap<String, String>> params)
is the one responsible for plotting the given series.
It returns the file name (full path) of the .png file, to be then added to a list of files to be concatenated and plotted in one file.

Sources Dependency Plots

The ExperimentAverageGNUPLOT_dependencyPlots class is the one responsible for these plots.

- In the start() method, all ***datasets characteristics parameters have to be set*** in order to choose which datasets to be plotted:

nSources: number of sources.
NDI: number of Data items.
DiffValMethod: Controllin number of distict values method.
dependency_id = Globals.*controlSourcesDependency_control*;
similarity_id = Globals.*valueSimilarity_dissSimilar*;
data_items_coverage_method_id: The coverage method.
List<Double> covList: The uniform coverage values.
Note: If not unifrom coverage, the covList must contains ONLY ONE value , if empty, nothing will be plot.
List<Integer> controlSrcTruthMethodList: The set of truth control to be plot.
List<Double> percentage_true_val_per_srcList: Onlu used for unifrom truth control model.
List<Integer> numIndepSrc_List
percentageOfCopiedValues: value from zero to one representing the percentage of copied values.

Distinct values Plots

The ExperimentsAverageChartsGNUPLOT_diffValuePlots class is the one responsible for these plots:

- In the start() method, all ***datasets characteristics parameters have to be set*** in order to choose which datasets to be plotted, like in the last part.
- The plotModelPerPlot() plots all the datasets (for different characteristics) but only for one truth discovery model in each graph.
- The plotParameterPerPlot plots all truth discovery models results for a specific dataset characteristics in each graph.
- These paths set where the plots will be saved:

private String saveToEPS: Where each graph will be saved as eps file.

private String saveToPNG: where each graph will be saved as png file.

private String perModelFinalPlot: where the graphs matrix file for the plots per model will be saved.

private String perParameterFinalPlot: where the graphs matrix file for the plots per parameter (dataset characterization) will be saved.

- The method plotModel(List<List<List<Double>>> seriesList, List<String> keys, String voterName, String xLabe, String yLabe, List<HashMap<String, String>> params)

is the one responsible to use GNUPLOT.
It polts the given series, using the given keys (legend), the voter name is the graph title, xLabe and yLabe are the X and Y axes labels, finally, the params the set of GNUPLOT parameters for each series. Removing the title, the key(legend) changing the key orientation … should be done in this method.

Models Precision Plots

The ExperimentsAverageChartsGNUPLOT_Models class is the one responsible for these plots.

- In the start() method, all ***datasets characteristics parameters have to be set*** in order to choose which datasets to be plotted
- The real world datasets can be added to the plot.

The part responsible for the adding each real world dataset and its plot colors are at the end of the method start().

- Note: Both the start() methods parameters and the getSelectStatement() method are responsible for the set of datasets to be selected. Both methods have to be revised carefully

Scalability Plots

The scalability plots Class has all values hard codded.
No database is queried to get any data.
The class *ExperimentsAverageChartsGNUPLOT_Scalability* is the one responsible for the scalability plots.

- The method: set1000And10000() add the execution time for both datasets with 1,000 sources and 10,000 sources to the series lists.
- The method *plotCopyModelsAndLTM*() only plot the sources dependency aware models and LTM.
- The method *plotOthers*() only plot the remaining models.

- The class variables:
- **private** String saveToEPS: The path to save the eps files.
  private String saveToPNG: The path to save the png files.
  private String saveToALL: The path to save the file collecting all graphs.

Plotting average, standard deviation …

Computing new aggregation function

Go To **DataSetCollector** project, **qcri.dafna.database.chart.GNUPLOT** package, to the **ExperimentResults** class

- Each ExperimentResults object contains results for a specific dataset characterization. precision, accuracy, recall, specificity, numOfIteration and voterDuration contain the addition for all datasets values, and counter contains the number of dataset with this characteristics (*most of the time 10 datasets*).
- The addition of new dataset metrics is done in the method addValue(...).
- Every getter return the value (e.g. precision) divided by the counter in order to return the average.
- The ExperimentResults object also contains the minPrecision and maxPrecision values.
- Any new aggregation function can be computed in this class.

Plotting new aggregation function instead of average precision

Go To **DataSetCollector** project, **qcri.dafna.database.chart.GNUPLOT** package, to the **AbstractExperimentAverageChart** class.

The method
getXY(List<ExperimentResults> voterResults, String xAxe)
Assign the needed values to the X-Axe, the Y-Axe and the Min-Max-Average used for candle-stick plot.

If another value rather than the precision is needed to be plotted, it should be set here.

New Real World Dataset

Format the dataset

Each new real world dataset comes with its own format.

A new reader-parser should be implemented.

To write the formatted dataset, the next method (writeClaim) can be used:

In the package **qcri.dafna.dataModel.dataSet**
in the class **ClaimWriter**
the method
public static boolean writeClaim(BufferedWriter writer, **int** claimId , String objectIdentifier, String propertyName, String propertyValueString, String timeStamp, String sourceId, String dlim)
The dlim should be ","

Note:
All new datasets are written in a comma separator format.
Only old Datasets are written in a " |" (Tab pipe)separation format. Meanwhile the reading of the datasets are not impacted.

Every property value type must be declared.
Thus for every new dataset, if it contains new properties (e.g. "expected-arrival-time") this property value type must be added to:
qcri.dafna.dataModel.dataFormatter package, to the **DataTypeMatcher** class: in the **getPropertyDataType** method a line for the new property should be added, the **propertyName** must be exactly the property name appearing in the dataset.

The default property value type is **String.**

Adding a new ValueType

If a new value type is needed,

- The new Value Type must be added to the **ValueType** enumeration on the same class.
- Then a ***cleaning function*** must be added to the class **DataCleaner** on the same package.
- As well as an update to ***ALL methods*** in the class **DataComparator** must be implemented.
- A new method "**insertclaimInBucketValueTypeName**" similar to "**insertclaimInBucketNumerical**" or "**insertclaimInBucketBoolean**". This method decides in which bucket a value must be inserted, and if there is not any, a new bucket is created, set its max and min value, and add the value to it.

Start the experiment

An experiment must be launched in a class that **extends** qcri.dafna.experiment.Experiment.

2 main steps must be done

1. create the dataset
2. launch the experiment

Prepare the Dataset folder

To start an experiment on a dataset, a folder for the dataset must have the next format:

- Main Folder
  - "claims" (folder with the claims)
  - "truth" (folder with the ground truth)
  - "experimentResult" (empty folder for the results)

There are 2 ways to construct the dataset
the first using the ExperimentDataSetConstructor_Development.*readDataSet(...)* Class.
the first using the ExperimentDataSetConstructor.*readDataSet(...)* Class.

The ExperimentDataSetConstructor supports only comma separated files for both claims files and ground truth files.

While the ExperimentDataSetConstructor_Development class detect the format between (tab pipe) and (comma separated) files for the Claims files. And and experiment name must be added to the argument in order to choose the correct format to read the ground truth (i.e. there is no a fixed format for the ground truth, while a default one can be implemented if needed).

Launch the experiment

the method runExperiment is used run the experiment on the dataset. This method returns as hashmap with a key = the voter name, and a value = voterQualityMeasure object which contains all needed quality measures. These measures are automatically logged into a VoterQuality file in the experimentResult folder.

Adding new Model

Adding new Model

Implement the new model

- New models implementation should be added to the **QCRITruthDiscovery** Project, to the **qcri.dafna.voter** package.
- The new model must implement the super class: **Voter.**

The **Voter** super-class contains 2 main methods to be overridden:

1. **initParameters**()

2 main parameters are to be initialized:

- singlePropertyValue: **false** if the model accept the list values claimed by a source to appear in one claim as atomic value.

True if the model needs the list values claimed by a source to appear each on a separate claim.

- OnlyMaxValueIsTrue**: True** if only one value should be true for every data item: the value with maximum confidence; **False** if all values with confidence greater than 0.5 are considered true.

2. **runVoter**(**boolean** convergence100)

The **runVoter** method contains the logic of the model.

boolean convergence100: if true the computation should last until the maximum number of iterations (= Globals.*maxIterationCount)*. False if the computation should stop at convergence.

- The voter name should be added to the Globals class in order to be further used

e.g. Globals.voterNewName = "New Name";

Add the new model to the set of experiments

The model must be added to the set of models to be run on each dataset-experiments.

Go to the **QCRITruthDiscovery** Project, to the **qcri.dafna.experiment** package, the **Experiment** class on the **runExperiment** method:

- The model object should be instantiated.
- launchVoter(convergence100, *profileMemory)* is called
  - convergence100: boolean: whether to stop at convergence or continue.
  - ProfileMemory: whether to profile the memory consumption or not.

The **launchVoter** method return the **VoterQualityMeasure** object used to log all quality metrics.

- The **log(DataSet dataSet, BufferedWriter writer, BufferedWriter precisionWriter, DataQualityLogger logger, String VoterName, VoterQualityMeasures voterQualityMeasure, boolean header** /* *whether this line to be logged in the file header or a data line, this is always false, only true for logging the header of the file at the beginning of the launchVoter Method* */)

method is called in order to write this model results in the output files.

Add the new model to the plotted models

In order to add a new model results to a plot, we need to:

I-Initialize the data-structure to hold the results values for this model

II- Add these results to the plots

Initialize the data-structure to hold the results values for this model

Go To **DataSetCollector** project, **qcri.dafna.database.chart.GNUPLOT** package, **AbstractExperimentAverageChart** class

1. add

```
protected static List<ExperimentResults> newModelResuts = new ArrayList<ExperimentResults>();
```

2. add

```
protected static List<List<List<Double>>> newModelSeriesList = new
ArrayList<List<List<Double>>>();
```

3. go to **computeAverage**() method
   1. add

```
ExperimentResults newModel = new ExperimentResults(nSources,nDI, numDiffVal, diffValMethod,
cov, dependency_id, similarity_id, controlSrcTruthMethod, percentage_true_val_per_src,
Globals.voterNewName, numOfIndependentSources, percentageOfCopiedValues,
data_items_coverage_method_id);
```

All parameters are the same as the others ONLY the voter name must be changed.

2. Add

```
else if (voterName.equals(Globals.voterNewName)) { newModel.addValues(precision, recall,
accuracy,specificity, duration, iterationCount);}
```

to the while-loop

3. add

```
newModelResuts.add(newModel);
```

to the end of the method.

4. Go to **addSeries(String xAxeName, boolean computeMLE)** method
   1. add

```
newModelSeriesList.add(getXY(newModelResults, xAxeName));
```

2. then add

```
newModelResults = new ArrayList<ExperimentResults>();
```

at the end of the method

5. go to **getGnuplotParametersForPlotPerConfigurationColored**()

add

```
params = new HashMap<String, String>();
params.put("ps", "2");// specify the point size
params.put("pointtype", "11");// specify the point type
paramMap.put(Globals.voterNewName, params);
```

6. go to **getGnuplotParametersForPlotPerConfiguration**() and add

```
params = new HashMap<String, String>();
        params.put("ps", "2");// point size
        params.put("lw", "3");// line width
        params.put("lt", "1");// line type
```

```
params.put("lc", "-1");// line color
params.put("pointtype", "30");
paramMap.put(Globals.voterNewName, params);
```

Add these results to the plots

you will notice that the same steps will be redone to the 3 kinds of plots (dependency, distinct values and models precision.)

- Go To **DataSetCollector** project, **qcri.dafna.database.chart.GNUPLOT** package, **ExperimentAverageGNUPLOT_dependencyPlots** class

1. Go to **plotParameterPerPlot**() method

add

```
series.add(newModelSeriesList.get(i));
keys.add(Globals.voterNewName);
params.add(paramMap.get(Globals.voterNewName));
```

2. Go to **plotModelPerPlot**() method

add

```
chartsFiles.add(plotModel(newModelSeriesList, seriesKeyList, Globals.voterNewName, xLabel, yLabel, null));
```

Note: in this case, one graph is added to the Big A3 charts matrix file, so the last line in the method should be adjusted:
ChartsMatrix.*creatChartsMatrix*(6, 2, chartsFiles,...
depending on the number of charts

- Go To **DataSetCollector** project, **qcri.dafna.database.chart.GNUPLOT** package, **ExperimentsAverageChartsGNUPLOT_diffValuePlots** class

1. Go to **plotParameterPerPlot**() method

add

```
series.add(newModelSeriesList.get(i));
        keys.add(Globals.voterNewName);
        param.add(gnuplotParams.get(Globals.voterNewName));
```

2. Go to **plotModelPerPlot**() method

add

```
chartsFiles.add(plotModel(newModelSeriesList, seriesKeyList, Globals.voterNewName, Xlabel, Ylabel, null));
```

Note: in this case, one graph is added to the Big A3 charts matrix file, so the last line in the method should be adjusted:
ChartsMatrix.*creatChartsMatrix*(6, 2, chartsFiles,...
depending on the number of charts

- Go To **DataSetCollector** project, **qcri.dafna.database.chart.GNUPLOT** package, **ExperimentsAverageChartsGNUPLOT_Models** class

1. Go to **plotPrecisionPerModel**() method

add

```
param.add(gnuplotParams.get(Globals.voterNewName));
x.add(12.0); // increment by 1 for each new model
y.add(newModelSeriesList.get(i).get(1).get(0));
lineMin.add(newModelSeriesList.get(i).get(2).get(0));
lineHigh.add(newModelSeriesList.get(i).get(3).get(0));
boxHigh.add(newModelSeriesList.get(i).get(4).get(0));
```

2. In each of the methods: **addBook(), addFlight(), addWhether(), addPopulation(), addBiography()**

add

```
x.add(12.0); // increment by 1 for each new model
y.add(0.000);// this new model precision
```

3. Go to **plotModel()** method, go to the line:

```
plotter.getAxis("x").set("xtics", "(\"Voting\" 1.0, \"TruthFinder\" 2.0, \"Cosine\" 3.0, \"3-Estimates\" 4.0,  \"2-Estimates\" 5.0, \"AccuSim\" 6.0, \"Depen\" 7.0, \"Accu\" 8.0, \"AccuNoDep\" 9.0, \"SimpleLCA\" 10.0, \"GuessLCA\" 11.0 , \"NewName\" 12.0)  nomirror rotate by " + "-45");
```

This line associate the model name on the X-Axe with the number associated with each model in the 2 past steps. So the new model must be added with its exact index.

- Go To **DataSetCollector** project, **qcri.dafna.database.chart.GNUPLOT** package, **ExperimentsAverageChartsGNUPLOT_Scalability** class

1. At the beginning of the class add

```
protected static List<List<List<Double>>> newModelSeriesList_2 = new ArrayList<List<List<Double>>>();
```

2. Go to **set1000And10000**() method

add the next to the 1,000 sources part

```
newModelSeriesList = new ArrayList<List<List<Double>>>();
        temp = new ArrayList<List<Double>>();
        x = new ArrayList<Double>();
        x.add((double) 100);
        x.add((double) 1000);
        x.add((double) 2000);
        y = new ArrayList<Double>();
        y.add(0.0);// add time in millisecond for 1,000 sources and 100    data items
        y.add(0.0);// add time in millisecond for 1,000 sources and          1,000 data items
        y.add(0.0);// add time in millisecond for 1,000 sources and         10,000 data items
        temp.add(x); temp.add(y);
```

```
newModelSeriesList.add(temp);
```

3. in the same **set1000And10000**() method add the next to the 10,000 sources part

```
newModelSeriesList_2 = new ArrayList<List<List<Double>>>();
temp = new ArrayList<List<Double>>();
x = new ArrayList<Double>();
x.add((double) 100);
x.add((double) 1000);
y = new ArrayList<Double>();
y.add(0.0);// add time in millisecond for 10,000 sources and 100 data items
y.add(0.0);// add time in millisecond for 10,000 sources and 1,000 data items
temp.add(x); temp.add(y);
newModelSeriesList_2.add(temp);
```

4. in the **plotOthers**() method

add

```
series.add(getSeconds(newModelSeriesList.get(i)));
        keys.add(Globals.voterNewName+"(s = 1,000)");
        param.add(gnuplotParams.get(Globals.voterNewName));
```

a the end of the 1000 sources part.
And add

```
        series.add(getSeconds(newModelSeriesList_2.get(i)));
        keys.add(Globals.voterNewName + "(s = 10,000)");
        p2 = new        HashMap<String,String>(gnuplotParams.get(Globals.voterNewName));
        p2.put("lt", "2");
        param.add(p2);
```

at the end of the 10,000 sources part.

Read World Dataset Experiment

Run/Rerun a real World Dataset experiment

In the QCRITruthExperiment Project. In the qcri.dafna.experiment package.

There exist an experiment class for every real World dataset. This class is simply responsible to read the required dataset, then launch the experiment.