Completeness-aware Rule Learning from Knowledge Graphs

Vinay Pyati[1]

RWTH Aachen University, Germany
vinay.pyati@rwth-aachen.de

Abstract

With the advent of Knowledge graphs (KGs) the huge stored data became smarter, easily manageable and understandable. They are now being extensively used in semantic search, Question Answering and many other important applications. Rule Mining is an important aspect to discover patterns in KGs. In theory KGs consist of high degree of incompleteness. Mining over a incomplete data might lead to spurious results.The mined rules which are considered to be of good quality are actually wrongly estimated. In this paper[1] the in-completeness is exposed by using in-completeness meta-information and completeness aware scoring functions for relational association rules. Evaluation is been done and proved that this approach provides better rule ranking than the state-of-the-art techniques used to expose the missing facts.

1 Introduction

The availability and demand of structured data and the rise of the semantic web bring new challenges for machine learning and data mining. Knowledge Graphs (KGs) contain billions of positive facts and are generally incomplete. KGs follow the Open World Assumption (OWA) which says missing facts are unknown rather than false. KGs have to undergo rule mining to extract info from it. As the rules are learned from incomplete data, they might be spurious and predict incorrectly on missing facts. On knowing the downside of using KGs it was important to put efforts into detecting the firm numbers of facts of certain types that hold in the real world (e.g., Einstein has 3 children).This could be done by web extraction.Acquring such meta-data can help us to derive clues on the topology of KGs. These clues could help in rule learning methods.

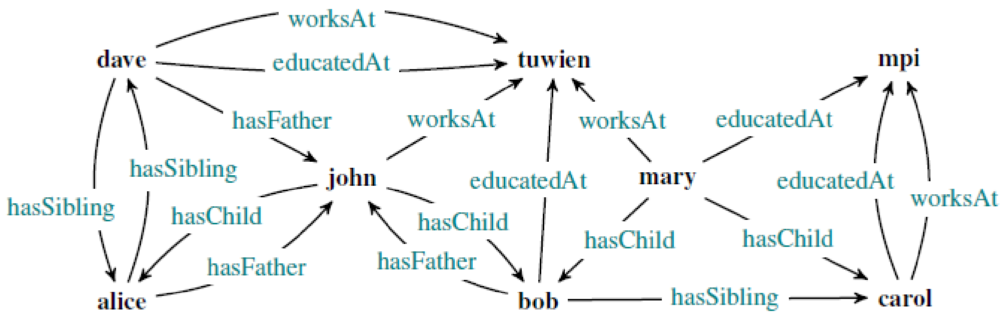


Figure 1: Example of KG

Consider two rules *r1: hasChild(X,Y)* $\leftarrow$ *worksAt(X,Z), educatedAt(Y,Z)* and *r2: hasSibling(X, Z)* $\leftarrow$ *hasFather(X,Y),hasChild(Y,Z).* From figure number 1 one can infer that r1 should be ranked lower than r2 as workers of some institutions may have children among the people educated there. Even though knowing it's importance there are no methods as of now that concretely use these clues in rule learning.

In this work, meta-data is exploited to find the expected number of edges in KGs to assess the quality of learned rules. Mining rules could be a huge and time consuming task. Hence the idea of implementing completeness aware rule learning algorithm using spark and scala i.e. distributed data sets using the Apache Spark engine could prove to be very promising. Since KGs contain billions of facts and running mining algorithms on the facts is often not scalable. This is a big data problem which could be solved with the Apache Spark engine.

2 Approach

Important step in gathering meta-information about in-completeness is finding out number of edges that should exist for a given subject-predicate pair in a KG. Approximation of upper and lower bounds on the number of edges that should exist in KGs and proposing rule ranking measures are the main blocks in this approach. Unlike the rule mining system AMIE[2] that predicts which parts of knowledge graph are complete or incomplete, the work in this paper[1] focuses on missing edges and estimate the exact number of edges to be present in a KG to make it complete. Completeness information is put forward in the rule acquisition phase so that it prunes away problematic rules and wrong predictions. The ideas [1, 4] from here helped to design such approach. Before the approach is discussed I would like to set up some basic terms and definitions.

Triples: The content of KGs is a set of $\langle$ subject predicate object $\rangle$ triples, e.g., $\langle$ john hasChild alice $\langle$. These KGs have Open World Assumption and contain only a subset of true information. Triples could be written as binary predicates, like hasChild(john, alice). An ideal Graph is the one which contains all true facts and is an imaginary construct which contains to which extent the available graph lacks information wrt. the ideal graph, e.g., Einstein is missing 2 children and Feynman none). This information is called as Cardinality statement.

Cardinality statements: The in-completeness meta-data can be constructed from cardinality statements by reporting(the numerical restriction on) the absolute number of facts over a certain relation in the ideal graph. For example:

$$john\, hasChild\, mary \tag{1}$$

one can infer 6 cardinality templates from 1
(1) children of john;
(2) edges from john to mary;
(3) incoming edges to mary;
(4) facts with john as a subject;
(5) facts over hasChild relation;
(6) facts with mary as an object

Interesting templates could be (1) and (3) as they can be extracted from the web extraction techniques. (1) could obtain inverse relations For instance,
⟨#s : hasChild(s, john) = #o : hasParent(john; o)⟩

for the predicates hasChild and hasParent, which are inverses of one another.A predicate p and a constant s outputs a natural number corresponding to the number of facts in ideal graph over p with s as the first argument:

$$num(p, s) := \#o : p(s, o) \tag{2}$$

One can further construct missing facts from the 2

Algorithm 1: CARL Algorithm

```
Data: a set of PSO triple IDs $pso$, a set of expected cardinality IDs $ecpv$ and a set of
      expected properties $prop$
Result: a set of Scored Rules R
Comment: $pso$ and $ecpv$ are a map of IDs obtained after triple read. $prop$ are the set
 of predicates

- set standard metrics $m$
- create property instances $pv$
- for each $i \in pso$
- add $i$ to $pv$

- create expected triples per relation $et$
- for each $i \in prop$
- get expected cardinality $ec$ from $ecpv$ and derive actual cardinality $ac$
- add $i$,$ec$-$ac$ to $et$

- for each $(p) \in prop$
- for each $(q) \in prop$
- for each $(r) \in prop$
- find subjects with cardinality $swc$
- set $metric$ using $pv$ and $et$
- if $metric < m$
- continue
- add $m$ to R
- sort R with completeness confidence
```

This algorithm can be directly applied to small and medium size datasets. However, big datasets are commonly distributed on different nodes as a *Resilient Distributed Dataset* (RDD) and should be processed in parallel by using frameworks like Spark.

The triples are read and an ID Store is prepared which basically has unique ID for each subject predicate and object. This triple pair is a nested LinkedHashMap. Another nested map is prepared and assigned IDs. It is named as *ecpv* which is expected cardinality by property value.

ecpv is prepared and read from a separate file which is given as input to the application which contains the cardinality statements.

This algorithm concerns the association rule learning to discover the frequent patterns in a data set and the subsequent transformation of these patterns into rules. Usual scoring of association rules is based on rule support, body support and confidence. Support and confidence were originally set up for scoring rules over complete data. If data is missing, their interpretation could be tricky and they can be misleading. Confidence under the Partial Completeness Assumption (PCA) has been proposed as measure, which guesses negative facts by assuming that data is usually added to KGs in batches, i.e., if at least one child of John is known then most probably all Johns children are present in the KG.

Completeness-aware rule scoring aims to score rules not only by their predictive power on the known KG, but also wrt. the number of wrongly predicted facts in complete areas and the number of newly predicted facts in known incomplete areas. Three important metrics to achieve this are
1. The completeness confidence
2. Completeness precision and recall
3. Directional metric.

.

3 Implementation

I implement the triple and cardinality file reader with the help of RDDs. The paper[1] does not contain any explicit algorithm for rule learning to impement but one resemblance of such algorithm could be AMIE[2] in which the methodology is different but both use the basic principle of rule mining with metric as focus. The application requires 2 files one is input triples and other is cardinality information which has to be acquired from the offcial site of wikidata which is around 21.5GB of collection of facts. A python program which takes this huge file and outputs out these 2 input files for further processing to the algorithm. Following are the details of the source.

Source Location:
https://dumps.wikimedia.org/wikidatawiki/entities/
Files arranged as wikidata-YYYYMMDD-truthy-BETA.nt.gz

Output of this file:
input_triples.tsv and input_cardinalities.tsv

3.1 Classes and Inputs

1. Triples

tripleRDD: *TripleRDD*	Loaded *TripleRDD* object
subject predicate object: *String*	mapped and returned

2. Cardinalities

tripleRDD: *TripleRDD*	Loaded *CardinaliitiesRDD* object
subject predicate cardinality: *String*	mapped and returned

3. ScoredRule

p: *Int*	*rule p* ID
q: *Int*	*rule q* ID
r: *Int*	*rule r* ID

3.2 Other Functions

Some other main and utility functions.

Function	Input	Output
parseTriples	RDDString	mapped Triple:RDD map
parseCardinlalities	RDDString	mapped cardinality:RDD map
CARLAlgorithm	|–pso: LinkedHashMap |– nodes: ArrayBuffer |– ecpv: LinkedHashMap |– properties: TreeSet |– numRules: Int	ScoredRule:ListBuffer

Utils.scala: This file contains many utility functions which are required and called by the algorithm .
Preprocessor.scala: This file contains the main function where the execution begins. The CARL Algorithm is called from here.
Algorithm.scala: This contains the main algorithm which implements the rule learning and outputs the rules ranked by completeness confidence.

The Algorithm is optimized to handle small and medium size datasets. The initial preprocessing is done fully on RDDs. To allow the changes made to the local variables declared I made use of the accumulators. RDDs do not preserve change in a variable declared before the rdds come into action accumulators are gateway to preserve data and transforming as required. All the accumulators store the collections and are retrieved back as normal collections after the RDD operations end. Introducing breakable block to have a behaviour similar to break/continue. Scala by default does not have the keyword 'continue' but we can emulate the continue functionality with placing breakable just after the loop starts and add break wherever we want. This breaks and finds an exception handling, if it does not find one it continues the loop for next iteration. There are sections where the Algorithm that uses the normal scala utilities. This is being improved by a beta implementation and I have added this to the improvements/ future section to make the whole application completely free of normal scala functions. This can be found in the project repository.

4 Evaluation

Top k mined by CARL are compared with that of stadard rule learing algorithms such AMIE[2] and WarmeR[3].

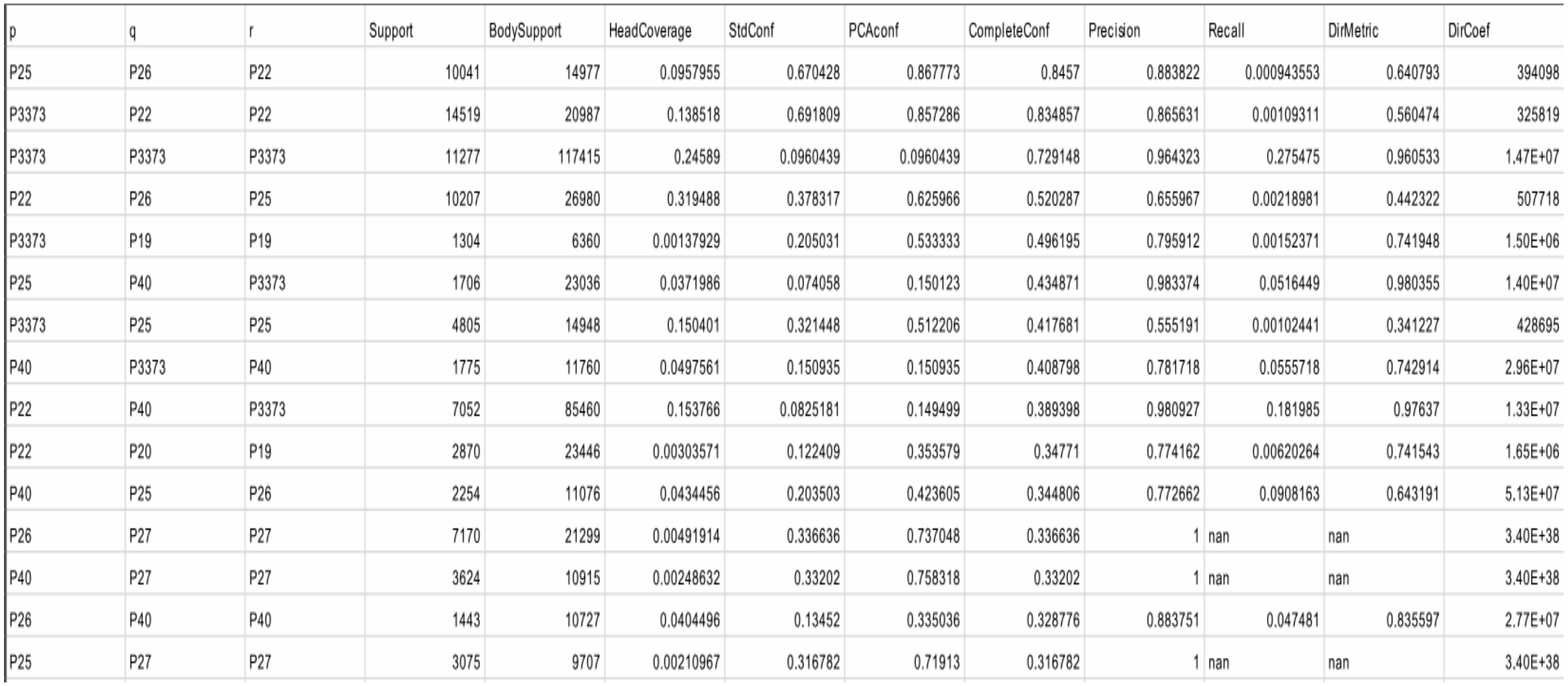
Dataset used is *WikidataPeople* from which the 2 input files *triples* and *cardinalities* are extracted. The size of triples is 6M facts and 12M cardinallities. For the WikidataPeople dataset, the approximation of the ideal KG ($\mathcal{G}^i$) is obtained by exploiting available information about inverse relations (e.g., *hasParent* is the inverse of *hasChild*), functional relations (e.g., *hasFather, hasMother*) as well as manually hand-crafted solid rules.

Cardinality statements are acquired by exploiting properties of functional relations, e.g., *hasBirthPlace, hasFather, hasMother* must be uniquely defined, and everybody with a *hasDeathDate* has a *hasDeathPlace*. For the other relations, the PCA is used. Firstly the available KG ($\mathcal{G}^a$) is constructed by removing parts of the data from $\mathcal{G}^i$ and introducing a synthetic bias in the data (i.e., leaving many facts in $\mathcal{G}^a$ for some relations and few for others). This would be good test case as the most of the actual information is unreported. For example only for 3% of non-living people sibling information is recorded and the actual information way more than that.

Second the *global ratio*, which determines a uniform percentage of data retained in the available graph. To further refine this,a factor *predicate ratio* individually for each predicate is applied. For the WikidataPeople KG, this ratio is chosen as (*i*) 0.8 for *hasFather* and *hasMother*; (*ii*) 0.5 for *hasSpouse, hasStepParent, hasBirthPlace, hasDeathPlace* and *hasNationality*; (*iii*) 0.2 for *hasChild*; and (*iv*) 0.1 for *hasSibling*. For a given predicate, the final ratio of facts in $\mathcal{G}^a$ retained from those in $\mathcal{G}^i$ is then computed as $\min(1, 2 * k * n)$, where k is the predicate ratio and n is the global ratio. I could evaluate the global factor case of 0.5 and top 15 rules are are displayed in the figure 2. For the WikidataPeople KG, directional metric, weighted directional metric and completeness confidence show the best results, followed by completeness precision.

4.1 Runtime

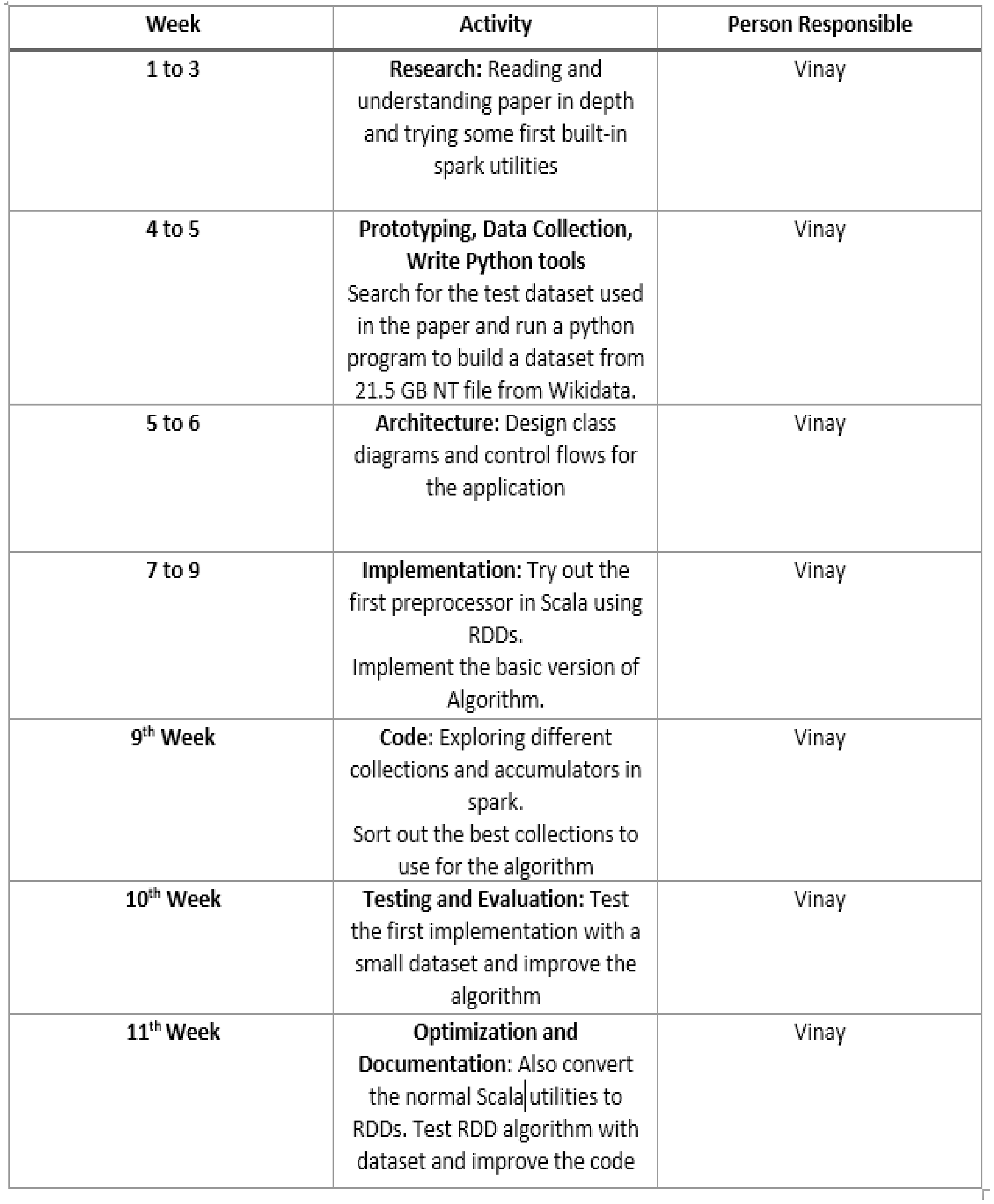
The runtime of the algorithm performs very well in case of small and medium size datsets. Typically for less than 0.5M million facts the algorithm can be run in a short time.To output the result for a global ratio of 0.5 took a very long time as the algorithm does not fully use the spark features.The downside of having lesser facts and cardinalities is that the metric do not come out well. This is a concern as of now that the algorithm run time is high for the fact sizes of 1M and above even though the it has been optimized to a great extent. I have an improvement and future work section in the repository where the code is been converted to RDDs and shown signs of significant improvement of over the normal one. This code has to be tested as this a very beta implementation.



p	q	r	Support	BodySupport	HeadCoverage	StdConf	PCAconf	CompleteConf	Precision	Recall	DirMetric	DirCoef
P25	P26	P22	10041	14977	0.0957955	0.670428	0.867773	0.8457	0.883822	0.000943553	0.640793	394098
P3373	P22	P22	14519	20987	0.138518	0.691809	0.857286	0.834857	0.865631	0.00109311	0.560474	325819
P3373	P3373	P3373	11277	117415	0.24589	0.0960439	0.0960439	0.729148	0.964323	0.275475	0.960533	1.47E+07
P22	P26	P25	10207	26980	0.319488	0.378317	0.625966	0.520287	0.655967	0.00218981	0.442322	507718
P3373	P19	P19	1304	6360	0.00137929	0.205031	0.533333	0.496195	0.795912	0.00152371	0.741948	1.50E+06
P25	P40	P3373	1706	23036	0.0371986	0.074058	0.150123	0.434871	0.983374	0.0516449	0.980355	1.40E+07
P3373	P25	P25	4805	14948	0.150401	0.321448	0.512206	0.417681	0.555191	0.00102441	0.341227	428695
P40	P3373	P40	1775	11760	0.0497561	0.150935	0.150935	0.408798	0.781718	0.0555718	0.742914	2.96E+07
P22	P40	P3373	7052	85460	0.153766	0.0825181	0.149499	0.389398	0.980927	0.181985	0.97637	1.33E+07
P22	P20	P19	2870	23446	0.00303571	0.122409	0.353579	0.34771	0.774162	0.00620264	0.741543	1.65E+06
P40	P25	P26	2254	11076	0.0434456	0.203503	0.423605	0.344806	0.772662	0.0908163	0.643191	5.13E+07
P26	P27	P27	7170	21299	0.00491914	0.336636	0.737048	0.336636	1	nan	nan	3.40E+38
P40	P27	P27	3624	10915	0.00248632	0.33202	0.758318	0.33202	1	nan	nan	3.40E+38
P26	P40	P40	1443	10727	0.0404496	0.13452	0.335036	0.328776	0.883751	0.047481	0.835597	2.77E+07
P25	P27	P27	3075	9707	0.00210967	0.316782	0.71913	0.316782	1	nan	nan	3.40E+38

Figure 2: 0.5 Global Ratio

5 Project timeline



Week	Activity	Person Responsible
1 to 3	**Research:** Reading and understanding paper in depth and trying some first built-in spark utilities	Vinay
4 to 5	**Prototyping, Data Collection, Write Python tools** Search for the test dataset used in the paper and run a python program to build a dataset from 21.5 GB NT file from Wikidata.	Vinay
5 to 6	**Architecture:** Design class diagrams and control flows for the application	Vinay
7 to 9	**Implementation:** Try out the first preprocessor in Scala using RDDs. Implement the basic version of Algorithm.	Vinay
9th Week	**Code:** Exploring different collections and accumulators in spark. Sort out the best collections to use for the algorithm	Vinay
10th Week	**Testing and Evaluation:** Test the first implementation with a small dataset and improve the algorithm	Vinay
11th Week	**Optimization and Documentation**: Also convert the normal Scala utilities to RDDs. Test RDD algorithm with dataset and improve the code	Vinay

Further ideas:
As mentioned up until now the algorithm have only been tested locally with medium size data files with the geberal algorithm. The next step would be to bring out the RDD version of the algorithm to tests with bigger data on a real cluster with multiple nodes to see how the different algorithms perform under those conditions. This would then help to churn out all the global ratios from 0.2 to 0.9 and would be apt for comparsions.

References

[1] Tonan, Stepanova, Razniewski, Mirza, Weikum. *Completeness aware Rule Learning from Knowledge Graphs*,

[2] Galarraga, L., Teflioudi, C., Hose, K., Suchanek, F.M. Fast Rule Mining in Ontological knowledge bases with AMIE+

[3] Goethals, B., den Bussche, J.V Relational association rules: Getting WARMeR. In: Pattern Detection and Discovery, Springer Berlin Heidelberg, Berlin, Heidelberg, 2011.

[4] Thomas Tanon. CARL-github `https://github.com/Tpt/CARL`.

[5] Apache Software Foundation Apache Spark `https://spark.apache.org/`