Universität des Saarlandes
Max-Planck-Institut für Informatik
AG5

Learning Rules With Categorical Attributes from Liked Data Sources

Masterarbeit im Fach Informatik
Master's Thesis in Computer Science
von / by

André de Oliveira Melo

angefertigt unter der Leitung von / supervised by

Prof. Dr. Gerhard Weikum

betreut von / advised by

Dr. Martin Theobald

begutachtet von / reviewers

Dr. Max Mustermann
Prof. Dr. Gerhard Weikum

Dezember / December 2012

Hilfsmittelerklärung

Hiermit versichere ich, die vorliegende Arbeit selbständig verfasst und keine anderen als die angegebenen Quellen und Hilfsmittel benutzt zu haben.

Non-plagiarism Statement

Hereby I confirm that this thesis is my own work and that I have documented all sources used.

Saarbrücken, den 11. Dezember 2012,

(André de Oliveira Melo)

Einverständniserklärung

Ich bin damit einverstanden, dass meine (bestandene) Arbeit in beiden Versionen in die Bibliothek der Informatik aufgenommen und damit veröffentlichtwird.

Declaration of Consent

Herewith I agree that my thesis will be made available through the library of the Computer Science Department, Saarland University.

Saarbrücken, den 11. Dezember 2012,

(André de Oliveira Melo)

To my father Cicero, my mother Marlene and my sister Carolina

- Andre

Abstract

With millions of articles in multiple languages, Wikipedia has become the de-facto source of reference on the Internet today. Each article on Wikipedia contains encyclopedic information about various topics (people, events, inventions, etc.) and implicitly represents an entity. Extracting the most important facts about such entity will help users to find desired information more quickly and effectively. However, this task is challenging due to the incomplete and noisy nature of Wikipedia articles. This calls for a mechanism to detect and summarize the most important information about an entity on Wikipedia.

This thesis proposes and implements CATE (**C**ontext-**A**ware **T**imeline for **E**ntity Exploration), a framework that utilizes Wikipedia to summarize and visualize the important aspects of entities in a timeline fashion. Such a system will help users to draw quickly an informative picture of an entity (e.g. life of a person, or evolution of a research topic, etc.). The novelty of CATE lies in seeing the entity in different contexts, synchronous with contemporaneous events. In addition, CATE puts the entity in a relationship with other entities, and thus offers a broader portrait about it. In order to efficiently query and visualize the events related to the entity, a number of techniques have been developed, combining information extraction and information retrieval with a novel ranking model. The thesis also discusses several experiments and evaluation results to show the effectiveness of the methods proposed.

Acknowledgements

Firstly, I would like to thank my advisor Dr. Martin Theobald, for his invaluable guidance. I feel deeply grateful for his technical assistance and motivational encouragement.

A special note of thanks to Prof. Gerhard Weikum for giving me the opportunity to pursue this thesis at Infomation and Database Systems department under his supervision. It was a very enriching and pleasant experience to write my Master Thesis here.

Contents

Chapter 1

Introduction

In the last years, the volume of semantic data available, in particular RDF, has dramatically increased. Initiatives like the W3C Semantic Web, which provides a common standard that allows data to be shared and reused across different applications, and the Linked Open Data, which provides linkages between different datasets that were not originally interconnected, have great contribution in such development. Moreover, advances in information extraction have also made strong contribution, by crawling multiple non-structured resources in the Web and extracting RDF facts.

Nevertheless, information extraction still has its limitations and many of sources might contain contradictory or uncertain information. Therefore, many of the extracted datasets suffer from incompleteness, noise and uncertainty.

In order to reduce such problems, one can apply to the knowledge base a set of inference rules that describes its domain. With that, it's possible to resolve contradictions or strengthen or weaken their confidence values. It's also possible to derive new facts that are originally not existent due to incompleteness. Such inference rules can be of two types:

1. *Hard Rules*: Consistency constraints which represent which might represent functional dependencies, functional or inverse-functional properties of predicates or Mutual exclusion. For example:

    - $marriedTo(x, y) \leftarrow marriedTo(y, x), (x \neq y)$
    - $grandChildOf(x, y) \leftarrow childOf(x, z), childOf(z, y)$
    - $parentOf(x, y) \leftarrow childOf(y, x)$
    - $(z = y) \leftarrow wasBornIn(x, z), wasBornIn(x, y)$

2. *Soft Rules*: Weighted Datalog rules that frequently, but not always hold in the real world. As they might also produce incorrect information, each rule itself must have a confidence value which should be applied to derived facts, for example married people live in the same place as their partner has confidence 0.8:

$$livesIn(x,y) \leftarrow marriedTo(x,z) livesIn(z,y)\ [0.8]$$

So, if we have an incomplete knowledge base, which lacks information about where *Michelle Obama*l lives, but we know that she's married to *Barack Obama* and he lives in *Washington, D.C.*, both with confidence 1, we could then apply this soft rule to derive the fact *livesIn(MichelleObama, WashingtonDC)* with confidence 0.9.

Such rules are rarely known beforehand, or are too expensive to be manually extracted. Nevertheless, the data itself can be used to mine these rules using *Inductive Logic Programming (ILP).*

ILP is a well-established framework for inductively learning relational descriptions (in the form of logic programs) from examples and background knowledge. Given a logical database of facts, an ILP system will generate hypothesis in a pre-determined order and test them against the examples. However in a large knowledge base, ILP becomes too expensive as the search space grows combinatorially with the knowledge base size and the larger the number of examples, the more expensive it is to test each of the hypothesis.

rules with constants might be really interesting.

Moreover, testing hypothesis with constants increases the search space dramatically, making it unfeasible to test all possible hypothesis with constants in a large knowledge base. In such case, it's necessary to arbitrarily reduce the search by restricting the set of constants to be included in ...

data mining, rule mining, datalog rules

talk a bit about ilp

1.1 Motivation

Given the huge size of search space and the great interestingness of rules with constants, we need to smartly prune constants or combinations of constants that , learning datalog rules can be already extremely costly.

Numerical constants are a special case, and they need to be treated differently. Depending on the numerical attribute domain, in case of a continuous real number domain for example, setting a numerical constant as an individual value will very likely have a very low support and also that would result and extremely large number of possible constants. Therefore, we split the attribute's domain into k buckets and then check if any of the buckets present any gain in comparison with its correspondent numerical constant-free hypothesis.

For example if we test the hypothesis and we find support=100 and confidence=0.4:

$$isMarriedTo(x, y) \leftarrow hasAge(x, z)$$

and then we split z into three buckets:

- $k = 1 : z \in [0, 20]$
- $k = 2 : z \in (20, 40]$
- $k = 3 : z \in (40, \infty]$

we then test the hypothesis for each of the three buckets and obtain

- $isMarriedTo(x, y) \leftarrow hasAge(x, z), z \in [0, 20]$
  support=40, confidence=0.1
- $isMarriedTo(x, y) \leftarrow hasAge(x, z), z \in (20, 40]$
  support=40, confidence=0.5
- $isMarriedTo(x, y) \leftarrow hasAge(x, z), z \in (40, \infty]$
  support=20, confidence=0.8

for k=2 and k=3, the hypothesis has significant gain by specifying numerical constants. Adding a relation to the body might produce totally different support and confidence distributions along the buckets. For example, if we add the relation hasChild(x,a), we could obtain other interesting rules:

- $isMarriedTo(x, y) \leftarrow hasAge(x, z)hasChild(x, a)$
  support=50, confidence=0.625
- $isMarriedTo(x, y) \leftarrow hasAge(x, z)hasChild(x, a), z \in [0, 20]$
  support=2, confidence=0.5
- $isMarriedTo(x, y) \leftarrow hasAge(x, z)hasChild(x, a), z \in (20, 40]$
  support=30, confidence=0.7

- $isMarriedTo(x, y) \leftarrow hasAge(x, z)hasChild(x, a), z \in (40, \infty]$
  support=18, confidence=0.9

adding some relations might not bring any gain or even loss in confidence, but when bucketing per age, present a different distribution,

Nevertheless, adding some relations with no correlation to the rule might not generate any interesting rules if

1.2 Contributions

In this Thesis, we propose a pre-processing step to build a graph we call Correlation Latticefor each numerical property. In each graph, that has a numerical property as root, we first query the frequency distribution on the numerical attribute, then split them in k buckets. Then we pick a set of c categorical properties that can be joined with the root, and analyze how the distribution of sub-population created by joining them with the root is affected. Afterwards we try to combine each of the categories and see if they still produce interesting sub-populations, like in frequent set mining.

We also evaluate different heuristics and interestingness measures.

In a hypothesis containing a numerical attribute in the body, we can obtain a support and confidence value for each of the buckets, and

With that, during the ILP algorithm, once we add one of the root properties, we can then search for the most interesting categorical properties that could result in different accuracy distributions. For every categorical property we can also suggest the most interesting constants and other categorical properties to be combined in a subcategory of both.

1.3 Outline

Chapter 2

Related Work

2.1 Logic Programming

2.2 Inductive Logic Programming

2.3 Mining Opimized Rules for Numberic Attributes

[?]

2.4 Minimum Description Length

[?]

2.5 Semamtic Web

2.6 Linked Open Data

Chapter 3

Correlation Lattice

The idea is to build during preprocessing a graph inspired in the Itemset Lattice that describes the influence of different categorical relations on a given numerical attribute's distribution. We call such graph a Correlation Lattice. To illustrate the idea, let's analyze a simple real-world example with the *hasIncome* relation. If we have two categorical relations, one strongly correlated to income, e.g. *hasEducation*, and one uncorrelated (or very weakly correlated), e.g. *wasBornInMonth*.

Let's assume that for the relation *wasBornInMonth(x,y)* we have the 12 months from the Gregorian Calendar as constants and for *hasEducation(x,y)* we can have 10 different categorical constants for y: "Preschool, "Kindergarten, "ElementarySchool", "MiddleSchool", "Highschool", "Professional School", "Associate's degree", "Bachelor's degree", "Master's degree" and "Doctorate degree".

It's expected that the income distribution will be roughly the same for people born in any of the months, whereas for different education levels, e.g. Elementary School and Doctoral Degree, their income distribution are expected to be different between them and different from the overall income distribution.

In a further step, we try to join every possible pair of categorical relations and including the constants. For the given example with the relations *hasEducation* and *wasBornInMonth* we would then create the nodes:

hasIncome(x,y)wasBornInMonth(x,"January"),hasEducation(x,"Preschool")
hasIncome(x,y)wasBornInMonth(x,"January"),hasEducation(x,"Kindergarten")
...
hasIncome(x,y)wasBornInMonth(x,"January"),hasEducation(x,"Doctorate Degree")

hasIncome(x,y)wasBornInMonth(x,"February"),hasEducation(x,"Preschool")
hasIncome(x,y)wasBornInMonth(x,"February"),hasEducation(x,"Kindergarten")
...
hasIncome(x,y)wasBornInMonth(x,"February"),hasEducation(x,"Doctorate Degree")

...

hasIncome(x,y)wasBornInMonth(x,"December"),hasEducation(x,"Preschool")
hasIncome(x,y)wasBornInMonth(x,"December"),hasEducation(x,"Kindergarten")
...
hasIncome(x,y)wasBornInMonth(x,"December"),hasEducation(x,"Doctorate Degree")

Based on this idea, we basically check how different categorical relations affect a numerical distribution. Such information, together with other measures like support, provides valuable cues on what categorical attributes and what categorical constants might be the most interesting to be added to the hypothesis in the core ILP algorithm.

3.1 Categorical Relation Definition

In this section, we formally define a categorical relation as used in the Correlation Lattice.

First of all, a candidate relation must be joined with root relation's 1st argument (assuming that the numerical attribute is in the 2nd argument).

A candidate categorical relation $r(x, y)$, should be equivalent a non-injective function:

$r(x, y) \equiv f : X \to Y,\ s.t. |Y| < |X|$ and $|Y| = n,\ n > 1$
$\nexists g : Y \to X,\ s.t. f(g(x)) = x,\ \forall x \in X$

We can define subsets of $X_i \in X$, with which of them belonging to one category $y_i \in Y$:

$X_i \subset X,\ s.t. X_i = \{x \in X \mid f(x) = y_i,\ y_i \in Y\}$
$X = \bigcup_{i=1}^{n} X_i$ and $X_i \cap X_j = \emptyset,\ \forall i, j \in [1, n],\ i \neq j$

We can also broaden this definition by composing functional relations to a categorical or multiple categorical relations:

If we have:

$r_1(x, y) \equiv f_1 : X \to Y$ (categorical or not)
$r_2(y, z) \equiv f_2 : Y \to Z$ (categorical relation)

Then, $r'(x, z) \equiv f : X \to Z$, where $r'(x, z) = r_2(f_1(x), z)$ is also categorical

Numerical relations can also be turned into a categorical, by simply applying a bucketing function that maps a numerical domain into a finite set of k buckets:

$b : \mathbb{N} \to B$, where $B = \{b_1, b_2, \ldots, b_k\}$

So a numerical relation:

$r(x, y) \equiv f : X \to \mathbb{N}$

combined with a bucketing function b, $r'(x, b(y))$ would be categorical

(Then talk about non-categorical relations as categorical by considering its presence/absence)

3.2 Support

As described in ([**?**]), in top-down ILP every refinement causes the support to decrease, therefore we know that for every node in the Correlation Lattice, its support will be greater or equal than any of its children, so support is a monotonically decreasing measure so we can safely prune a node that doesn't reach the minimum support threshold.

3.2.1 Independece between Nodes

By simplicity, we assume that every possible pair of categorical relations are independent and we search for evidence to prove the contrary.

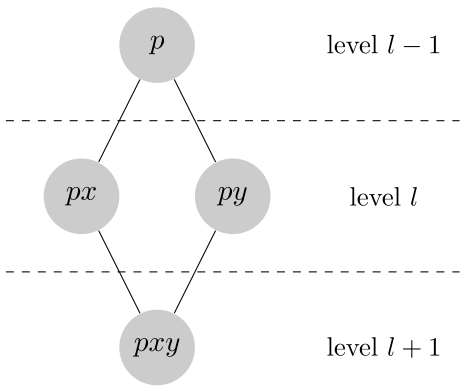
For 2 nodes to be joined, they must have a common parent, i.e. two nodes at level l (with $l + 1$ literals) are joinable if they share l literals. Therefore, it's straightforward to calculate the conditional probabilities of each of the joining nodes given the common parent, and estimate the frequency distribution for the conditional independence case.

If we are joining hasEducation()...

Let's say we have the following join case:

For the example shown in **??**, we can calculate the conditional probability $p_i(x|p)$ and $p_i(y|p)$ assuming conditional independence given common parent p in order to estimate $\hat{h_i}(pxy)$:

Figure 3.1: Node join example for independence test



$$\begin{aligned}
p_i(x|py) &= p_i(x|p) \\
&= \frac{h_i(x)}{h_i(p)} \\
p_i(y|px) &= p_i(y|p) \\
&= \frac{h_i(y)}{h_i(p)}
\end{aligned} \tag{3.1}$$

$$\begin{aligned}
\hat{h_i}(pxy) &= p_i(x|py)p_i(y|p) * h_i(p) \\
&= p_i(x|p)h_i(y) \\
\hat{h_i}(pxy) &= p_i(y|px)p_i(x|p) * h_i(p) \\
&= p_i(y|p)h_i(x)
\end{aligned}$$

After that, we query the actual frequency distribution on the Knowledge Base and do an Pearson's chi-squared independence test. As null hypothesis and alternative hypothesis we have:

- $H_0 =$ *x and y are conditionally independent given their common parent p*
- $H_1 =$ *x and y are conditionally dependent given their common parent p*

Number of degrees of freedom is the number of buckets minus one:

$$df = k - 1$$

We calculate the χ^2 value:

$$\chi^2 = \sum_{i=1}^{k} \frac{(h_i - \hat{h_i})^2}{\hat{h_i}} \tag{3.2}$$

[?]

Then it's possible to obtain the p-value and check whether there's enough confidence to reject the null hypothesis H_0. In level 1 from Correlation Lattice, nodes can be directly pruned, on the other hand, for further levels, for a node to be pruned by independence, all the possible joins resulting the node must be independent. In level 2, for example, in order to prune the node $ra_1b_1c_1$, given that in level 1 the nodes ra_1b_1, ra_1c_1 and rb_1c_1 were not pruned. All the three possible join combinations should fail the independence test, i.e.:

$$\begin{aligned} freq(ra_1b_1c_1) &\approx freq(ra_1)p(rb_1|ra_1)p(rc_1|ra_1) \\ &\approx freq(rb_1)p(ra_1|rb_1)p(rc_1|rb_1) \\ &\approx freq(rc_1)p(ra_1|rc_1)p(rb_1|rc_1) \end{aligned} \tag{3.3}$$

This applies to nodes at any level l, with $p \leq l$ parents and C_2^p possible join pairs.

If any of the join pairs has enough evidence of being dependent, then 2 edges are created connecting each of the joined nodes to the result of their join

3.3 Heuristics

As seen in the previous sections, the number of nodes in Correlation Latticegrow exponentially with the number of categorical relations and its constants. For n categorical relations, each with m constants, the total number of nodes is 2^{nm}. Frequently, pruning by support and independence is not sufficient to make it feasible and it's necessary to apply heuristics to prune more aggressively.

As we are interested in categories that produces a subsets with a distribution different from the root and its super-categories (parent nodes, for the case of joining multiple categories), distribution divergence measures are a good option. Some of the state-of-the-art measures we can use are the following:

- Kullback-Leibler [**?**]:

$$D_{KL}(P||Q) = \sum_i \ln\left(\frac{P(i)}{Q(i)}\right) * P(i) \tag{3.4}$$

- Chi-squared (χ^2):

$$D_{\chi^2}(P||Q) = \sum_i \frac{(P(i) - Q(i))^2}{P(i)} \tag{3.5}$$

- Jensen-Shannon [**?**]:

$$D_{JS}(P||Q) = \frac{1}{2}D_{KL}(P||M) + \frac{1}{2}D_{KL}(Q||M), \tag{3.6}$$

Where P and Q are the distributions to be compared and $M = \frac{1}{2}(P + Q)$

As discussed in [**?**], although Jensen-Shannon is computationally more expensivee, it has the advantage of being a symmetric and smoothed version of the Kullback-Leibler measure.

Pruning by divergence is clearly not safe. Let's suppose we have a root numerical relation $r(x, y)$, and two categorical relations $a(x, z)$ with constants A_1 and A_2 and $b(x, w)$ with constants B_1 and B_2. For simplicity, let's assume y is divided in two buckets and root r has an uniform distribution [0.5 0.5] from frequencies [2 2]. It's possible to have ra_1 and ra_2 as well as rb_1 and rb_2 with the same uniform distribution with frequencies [1 1]. Nevertheless when combined we can have the following:

ra_1b_1 : [1.0 0.0]
ra_1b_2 : [0.0 1.0]
ra_2b_1 : [0.0 1.0]
ra_2b_2 : [1.0 0.0]

As shown above, divergence is not a monotonically decreasing measure, thus it can only be used as heuristics.

Moreover, using a divergence measure alone might also be problematic. Nodes with low support are more likely to present high divergence and then nodes with high support, supposing that they were drawn under the same distribution. Therefore, it's important to use divergence combined with support as pruning heuristics.

Moreover, we are interested not only in rules with high confidence, but also rules with high support so more facts can be derived.

Talk about how to use such measure? (Threshold, Top-k, Cost-Benefit)

Chapter 4

Algorithmic Framework

4.1 Knowledge Base Backend

For storing and retrieving RDF data we use RDF3X [**?**]. It has the advantage of being specialized and optimized for RDF data, using a strong indexing approach with compressed B$^+$-Tree indices for each of six permutations of *subject (S)*, *predicate (P)* and *object (O)*: *SPO*,*SOP*,*OSP*,*OPS*,*PSO* and *POS*.

4.2 Preprocessing

In this section, we will present the preprocessing steps required by our proposed algorithm. It basically consists of building a joinable relations map for each of the four join patterns, according to relations domain and range types as well as support threshold. Afterwards, we search the available categorical properties for each numerical relation that will be used in the Correlation Lattice. At last we describe the Correlation Latticestructure and the algorithm to build it.

4.2.1 Relation Preprocessing

In this step, we focus on creating for each of the four join patterns between two relations:

- Argument 1 on Argument 1: e.g. *hasIncome(**x**,y)hasAge(**x**,z)*
- Argument 1 on Argument 2: e.g. *hasIncome(**x**,y)isMarriedTo(z,**x**)*
- Argument 2 on Argument 1: e.g. *livesIn(y,**x**)isLocatedIn(**x**,z)*

- Argument 2 on Argument 2: e.g. *livesIn(y,**x**)wasBornIn(z,**x**)*

4.2.1.1 Exploiting Relation Range and Domain Types

A knowledge base is expected to have an ontology defining the structure of the stored data (the types of entities and their relationships). Additionally, every relation's range (type of 1st argument) and domain (type of 2nd argument) should be defined. These information can help us identify the allowed joining relations for each join pattern.

For every possible pair of relations,

The algorithm is shown in the pseudo-code bellow:

4.2.1.2 Exploiting Support Monotonicity

As seen in (???), support is the only monotonically decreasing measure in top-down ILP. So we know that by adding any literals to the hypothesis, we can only get a smaller or equal support. Therefore, for each pair of joinable relations in each of the join patterns, we can query the knowledge base and check whether they reach the minimum support threshold.

Thus, if any pair of relations doesn't reach the minimum support for a given join pattern, we know that any hypothesis containing such join will therefore fail the support test as well, so we don't need to test such hypothesis in the core ILP algorithm.

The relation preprocessing will result in 4 maps, one for each join pattern. Each map has relations as keys and a set of joinable relations as value. The refinement step at the ILP algorithm, will then access this map when choosing a new literal to be added.

4.2.2 Correlation Lattice

4.2.2.1 Graph Node

Every node essentially contains the following attributes:

- Set of pointers to parent nodes
- Set of pointers to child nodes
- Set of pointers to constant nodes
- Histogram with facts distribution over root numerical property

4.2.2.2 Building the Correlation Lattice

For building the Correlation Lattice, we start with the root node, which has a numerical property as literal and no constants assigned, e.g. *hasIncome(x,y)*. We then query the distribution of positive examples over the property in the whole Knowledge Base.

SELECT COUNT ?y WHERE { ?x <hasIncome> ?y } GROUP BY (?y)

It's also necessary to specify the bucketing technique and the number of buckets in order to extract the histogram from the obtained query results. These buckets are used to build the histograms of all nodes in the graph.

Afterwards, we select the the categorical properties that will be used in the lattice. For each of the selected properties, we join them with the root numerical property (for simplicity we'll assume all the categorical properties are joined with both 1st arguments) and we query the distribution again. In the first level, it's necessary to extract a histogram for each of the categorical constants in the selected properties. Therefore, it's a good strategy to group the results also by these categorical constants so If we select *hasEducation* for example, we would then fire the following SPARQL query:

SELECT COUNT ?z ?y WHERE { ?x <hasIncome> ?y . ?x <hasEducation> ?z }
GROUP BY (?z,?y)

With such query, it's possible to extract a histogram for the node *hasIncome(x,y)hasEducation(x,z)* and its correspondent constants

4.2.2.3 Searching Rules in Correlation Lattice

In the Correlation Latticeitself, it's possible to extract valuable rules. Given its characteristic of all the relations being joined on the same root argument, those rules represent how different categories are related along root's numberical constants.

For every non-root node in the Correlation Lattice, any of its parents can be seen as rule's body and the remaining literal as head, e.g.:

Let's denote a_i as a relation $a(x, y)$ with constants A_i for y, so for example $a_1 \equiv a(x, A_1)$:

For the node $ra_1b_1c_1$ with parents ra_1b_1, ra_1c_1 and rb_1c_1, we can extract and easily evaluate three rules:

$$Rule_1 : \quad a_1 \leftarrow b_1 c_1 r$$
$$supp_i(Rule_1) = h_i(r a_1 b_1 c_1)$$
$$conf_i(Rule_1) = \frac{h_i(r a_1 b_1 c_1)}{h_i(r b_1 c_1)}$$

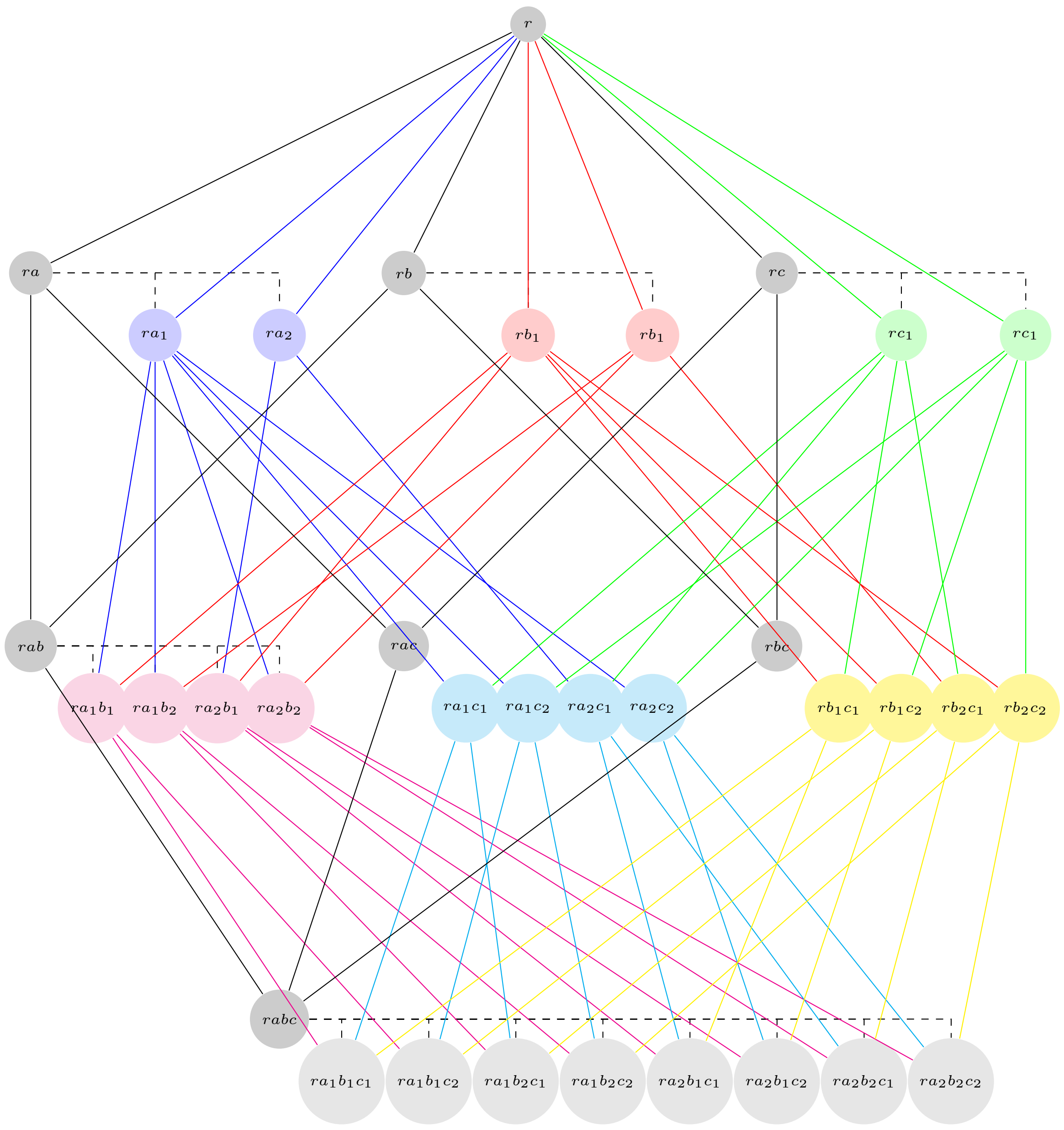
$$Rule_2 : \quad b_1 \leftarrow a_1 c_1 r$$
$$supp_i(Rule_2) = h_i(r a_1 b_1 c_1)$$
$$conf_i(Rule_2) = \frac{h_i(r a_1 b_1 c_1)}{h_i(r a_1 c_1)}$$

$$Rule_3 : \quad c_1 \leftarrow a_1 b_1 r$$
$$supp_i(Rule_3) = h_i(r a_1 b_1 c_1)$$
$$conf_i(Rule_3) = \frac{h_i(r a_1 b_1 c_1)}{h_i(r a_1 b_1)}$$

Subsequently we can analyze the frequency and confidence distributions and determine whether any of the rules are interesting, using any of the techniques discussed in [???] and find possible interesting intervals.

4.2.2.4

4.3 ILP Core Algorithm



Algorithm 1: Checks whether two relations are joinable for a given join pattern

```
Input: Relation $r_i$, $r_j$, Argument $arg_i$, $arg_j$
Output: True if $arg_i$ from $r_i$ joins with $arg_j$ from $r_j$, False otherwise
if $r_i.arg_i = r_j.arg_j$ or subsumes($r_i.arg_i$,$r_j.arg_j$) or subsumes($r_j.arg_j$,$r_i.arg_i$) then
    return true;
else
    return false;
```

Algorithm 2: Checks whether join support exceeds threshold

```
Input: Relation $r_i$, $r_j$, Argument $arg_i$, $arg_j$, Float $supportThreshold$
Output: True if join support exceeds threshold, False otherwise
switch $(arg_i, arg_j)$ do
    case $(1, 1)$
        $query$ ← “select count distinct ?$x$ where { ?$x$ <$r_i$> ?$y$ . ?$x$ <$r_j$> ?$z$ }” ;
    case $(1, 2)$
        $query$ ← “select count distinct ?$x$ where { ?$x$ <$r_i$> ?$y$ . ?$z$ <$r_j$> ?$x$ }” ;
    case $(2, 1)$
        $query$ ← “select count distinct ?$x$ where { ?$y$ <$r_i$> ?$x$ . ?$x$ <$r_j$> ?$z$ }” ;
    case $(2, 2)$
        $query$ ← “select count distinct ?$x$ where { ?$y$ <$r_i$> ?$x$ . ?$z$ <$r_j$> ?$x$ }” ;
$joinSupport$ ← executeQuery($query$);
if $joinSupport \geq supportThreshold$ then
    return true;
else
    return false;
```

List of Figures

List of Tables

Bibliography