

FEDERAL UNIVERSITY OF SAO CARLOS

EXACT SCIENCES AND TECHNOLOGY CENTER

GRADUATION PROGRAM IN COMPUTER SCIENCE

Mining Ontologies to Extract Implicit Knowledge

Author: Lucas Fonseca Navarro

Supervisor: Prof. Dr. Estevam Rafael Hruschka Jr

Sao Carlos - SP

September 2016

FEDERAL UNIVERSITY OF SAO CARLOS

EXACT SCIENCES AND TECHNOLOGY CENTER

GRADUATION PROGRAM IN COMPUTER SCIENCE

Mining Ontologies to Extract Implicit Knowledge

Author: Lucas Fonseca Navarro

Thesis for the Post-Graduation Program in Computer Science of Federal University of São Carlos, as part of the requirements to reach the title of Master in Computer Science, specific field: Artificial Intelligence

Sao Carlos - SP

September 2016

Ficha catalográfica elaborada pelo DePT da Biblioteca Comunitária UFSCar
Processamento Técnico
com os dados fornecidos pelo(a) autor(a)

N322m Navarro, Lucas Fonseca
 Mining ontologies to extract implicit knowledge /
 Lucas Fonseca Navarro. -- São Carlos : UFSCar, 2016.
 82 p.

 Dissertação (Mestrado) -- Universidade Federal de
 São Carlos, 2016.

 1. Mineração de dados. 2. Ontologia. 3. Bases de
 conhecimento. 4. Algoritmos. I. Título.



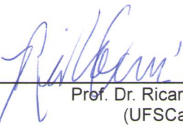
UNIVERSIDADE FEDERAL DE SÃO CARLOS
Centro de Ciências Exatas e de Tecnologia
Programa de Pós-Graduação em Ciência da Computação

Folha de Aprovação


Assinaturas dos membros da comissão examinadora que avaliou e aprovou a defesa de dissertação de mestrado do candidato Lucas Fonseca Navarro, realizada em 07/04/2016.



Prof. Dr. Estevam Rafael Hruschka Júnior
(UFSCar)



Prof. Dr. Ricardo Cerri
(UFSCar)



Prof. Dr. Alexandre Luís Magalhães Levada
(UFSCar)

Prof. Dr. Tom Mitchell
(CMU/Pittsburg)

Profa. Dra. Ana Paula Appel
(IBM Research/SP)

Certifico que a sessão de defesa foi realizada com a participação à distância dos membros Prof. Dr. Tom Mitchell e Profa. Dra. Ana Paula Appel e, depois das arguições e deliberações realizadas, os participantes à distância estão de acordo com o conteúdo do parecer da comissão examinadora redigido no relatório de defesa do aluno Lucas Fonseca Navarro.



Prof. Dr. Estevam Rafael Hruschka Júnior
Presidente da Comissão Examinadora
(UFSCar)

*To my parents, Raquel and Jose, my brothers, Vinicius and Gabriel, my girlfriend Julia
and everyone that supported me during this phase*

Acknowledgements

Firstly, I would like to express my sincere gratitude to my advisor Prof. Dr. Estevam Rafael Hruschka Jr. for all of his support during our research since 2010. If it wasn't for him I wouldn't even started a Master's Degree program right after my bachelor's degree, and I believe that this was the best choice I ever make in my whole life so far.

Besides my advisor, I would like to thank the rest of my thesis committee: First Prof. Dr. Ricardo Cerri and Prof. Dr. Alexandre M. Levada, for their comments and encouragement at my proposal. I would like to thank Prof. Dr. Tom M. Mitchell for inviting me to visit Carnegie Mellon University from 60 days during my Master's degree, and to guide me there at Pittsburgh. It's impossible to describe how much I learned there in just a couple of weeks, it was really amazing.

I would also like to make a special thanks to Dr. Ana Paula Appel, that was an advisor for me along with Prof. Estevam during my whole bachelor's course, besides that she was always helping me in every project that I did so far. The experience and knowledge she passes me is probably bigger to any course that I have attended so far.

Last but not the least, I would like to thank my family: my parents and to my brothers and my girlfriend for supporting me spiritually throughout writing this thesis and my life in general.

“Someone has to make it. Why shouldn’t it be you?”

Josh Homme

Abstract

With the exponentially growing of data available on the Web, several projects were created to automatically represent this information as knowledge bases(KBs). Knowledge bases used in most projects are represented in an ontology-based fashion, so the data can be better organized and easily accessible. It is common to map these KBs into a graph to apply graph mining algorithms to extract implicit knowledge from the KB, knowledge that sometimes is easy for human beings to infer but not so trivial to a machine. One common graph-based task is link prediction, which can be used not only to predict edges (new facts for the KB) that will appear in a near future, but also to find misplaced edges (wrong facts present in the KB). In this project, we create algorithms that uses graph-mining (mostly link-prediction based) approaches to find implicit knowledge from ontological knowledge bases. Despite of common graph-mining algorithms, we mine not just the facts on the KB, but also the ontology information (such as categories of instances and relations among them). The implicit knowledge that our algorithms will find, is not just new facts for the KB, but also new relations and categories, extending the ontology as well.

Resumo

Com o crescimento exponencial dos dados disponíveis na Web, diversos projetos foram criados para automaticamente representar esta informação como bases de conhecimento (KBs). As bases de conhecimento utilizadas na maioria destes projetos são representadas através de uma ontologia, então os dados ficam melhor organizados e facilmente acessíveis. É comum mapear estes KBs utilizando grafos para aplicação de algoritmos de mineração em grafos com o intuito de extrair conhecimento implícito do KB, conhecimento que as pessoas podem ser capazes de inferir mas não são tão triviais para uma máquina. Uma tarefa comum é a predição de arestas, que pode ser usada para encontrar arestas (fatos no KB) que vão aparecer em um futuro próximo, e além disso para encontrar arestas mal alocadas (fatos incorretos no KB). Neste projeto, criamos algoritmos que utilizam mineração em grafos (na maioria baseados em predição de arestas) para encontrar conhecimento implícito em bancos de conhecimento ontológicos. Apesar do uso comum de algoritmos de predição de arestas, vamos minerar também informações da ontologia (como categorias das instâncias e relações entre elas). O conhecimento implícito que nossos algoritmos vão encontrar, serão não somente novos fatos para o KB, mas também novas relações e categorias, estendendo também a ontologia.

Contents

Acknowledgements	ii
Abstract	iv
Resumo	v
Contents	vi
List of Figures	ix
Abbreviations	xi
1 Introduction	1
1.1 Context	1
1.1.1 Link-Prediction Task	2
1.1.2 Using Ontological Information to Find Implicit Knowledge	3
1.2 Motivation	4
1.3 Methodology	7
1.4 Organization	8
2 Literature Review	9
2.1 Background	9
2.2 NELL	11
3 Ontological Networks	13
3.1 Base Definitions	13
3.2 Defining an Ontological Knowledge Base (OKB)	14
3.3 Defining an Ontological Network (N^o)	15
3.4 Setting a second example	18
3.4.1 OKB_{sports}^2	18
3.4.2 $N_{sports}^o = (G_m, G_i)$	19
4 Finding new facts	22
4.1 The Graph Rule Learner	23
4.1.1 GRL Algorithm	25
4.1.2 Example	26
4.1.3 Multi Relation CTCGs	29

4.1.4	Grouping and Ranking Rules	30
4.1.5	GRL Implementation	31
4.2	Experiments	32
4.2.1	Finding Inference Rules with GRL	32
4.2.1.1	GRL applied to NELL's KB	32
4.2.1.2	Grouping and Ranking Repeated Rules.	33
4.2.1.3	GRL applied to YAGO KB	33
4.2.1.4	Validating rules	34
4.2.2	Using different link-prediction metrics	35
4.2.3	Comparing GRL with similar state of the art Rule Learners	37
4.2.4	Scalability	39
4.3	Conclusion	39
5	Finding new relations	40
5.1	Prophet	42
5.1.1	Prophet Algorithm	42
5.1.2	Prophet Example	43
5.1.3	New Prophet Implementation	45
5.2	OntExt	46
5.2.1	OntExt Example	47
5.3	PrOntExt	50
5.4	Results	51
5.5	Conclusion	52
	Finding new categories	53
5.6	The Sub-Categories Finder	56
5.6.1	The SubCat-Finder Algorithm	58
5.6.1.1	N-K-Means++ Algorithm	59
5.6.1.2	Naming the clusters	61
5.6.2	Experiments	62
5.6.2.1	Clustering a Category to Find Sub-Categories	62
5.6.2.2	Naming the Clusters	64
5.7	Conclusion	64
	Future Works	66
5.8	Extending the Ontological Network Model N^o	66
5.9	Future Classification Task	67
5.10	Sensibility of the algorithms, ignoring out of pattern facts	69
	Final Conclusion	71
	Projects Results	73
.1	Running GRL with NELL OKB it. 820	73
.2	GRL vs PRA vs AMIE	74
.3	PrOntExt Results running with NELL	74

Bibliography

List of Figures

1.1	Link-Prediction used in "People You May Know" recommendation in social network	2
1.2	Link-Prediction applied to propose a new relation(<i>athletePlaysLeague</i>) . . .	3
1.3	Athlete category, and the sub-groups (or sub-categories) that could be found with a community detection algorithm	4
2.1	NELL initial core structure (figure presented at [1])	11
3.1	Ontological Model Graph Example	16
3.2	Ontological Instances Graph Example	17
3.3	Ontological Model Graph G_m of Sports Network	20
3.4	ontological instances graph G_i of Sports Network	21
4.1	Example of a pattern of closed triangles between MaleP, Person and FemaleP	23
4.2	GRL running example	27
4.3	Applying a rule to infer new facts	28
4.4	GRL example for multi relation CTCGs	29
4.5	GRL Execution Diagram	32
4.6	Precision curve using different LP metrics	36
5.1	Example: Finding a new relation	41
5.2	Example of Prophet	44
5.3	Example of Prophet	45
5.4	Prophet Execution Diagram	46
5.5	Co-Occurency matrix m	48
5.6	Normalized m	49
5.7	Final Normalized m	49
5.8	The PrOntExt	50
5.9	Example: Finding a new category	53
5.10	Incrementing the social network ontology with a new category	54
5.11	Social network instances graph (G_i) after we add a new category and relations	54
5.12	Relation teammate in N_{sports}^o	55
5.13	Finding sub-categories for Athlete category	55
5.14	Sub-Category Finder	59
5.15	Sub-graph of the Ontological Model representing the Bombing Event	67
5.16	Sub-graph of the Ontological Model representing the Bombing Event	68

17	GRL outputed rules for NELL's KB	73
18	GRL outputed rules for YAGO's KB	74
19	PRA outputed rules for NELL's OKB (iteration 885)	74
20	AMIE outputed rules for NELL's OKB (iteration 885)	75
21	Relations found by PrOntExt running with NELL	75
22	Relations found by PrOntExt running with NELL	76

Abbreviations

KB	K nowledge B ase
OKB	O ntological K nowledge B ase
NELL	N ever E nding L anguage L earner
LP	L ink P rediction
RL	R ule L earner
GRL	G raph R ule L earner
CTCG	C lose T riangles C ategory G roup
OTCG	O pen T riangles C ategory G roup

Chapter 1

Introduction

1.1 Context

Currently, a number of different research projects focus on building large scale ontological knowledge bases (also called ontologies), such as Knowledge Vault [2], Freebase [3], YAGO [4-6], Gene Ontology¹ and a continuous learning program called NELL (Never Ending Language Learner)[1, 7]. An ontological knowledge base (OKB) is usually used to organize and store knowledge based on two different parts, here described as: **i)** an ontological model, where categories (*city*, *company*, *person*, etc.) and relations (**worksFor**(*person*, *company*), **headQuarteredIn**(*company*, *city*)) are defined, and **ii)** a set of facts which are instances of categories (**city** (*New York*), **company**(*Disney*), **person**(*Walt Disney*), as well as instances of relations (**headQuarteredIn**(*Disney*, *Orlando*)).

The ontology structure above described is a very simple model, thus in many projects such as YAGO [4] the initial ontology tends to evolve, being more and more sophisticated and complex. In projects targeting to extract information (inference process) from Knowledge Bases (KBs and also from OKBs), such as the few above mentioned, it is common to see an approach that maps data from the KBs into graphs². Such mapping allows the use of graph-mining approaches to the KB. In this sense, techniques such as

¹<http://geneontology.org/>

²sometimes the mapping process doesn't require any adaptation so it results just in using the KB as a graph - the mapping is trivial in such cases

link-prediction (that can help finding and extracting useful implicit information from the graph) can play an important role.

1.1.1 Link-Prediction Task

One of the most studied and disseminated topics in the graph-mining literature is the link-prediction task. It can be defined as a task to estimate the likelihood of future existence of an edge, between two nodes, based on the current graph information [8]. One traditional approach, used in several works, is the mapping of a social network into a graph [9, 10], and then applying link prediction algorithms to find new edges (friendship) among nodes (friends).

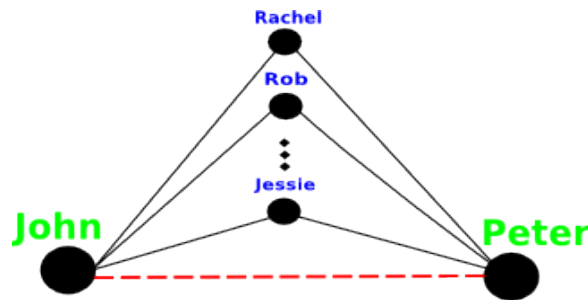


FIGURE 1.1: Link-Prediction used in "People You May Know" recommendation in social network

Figure 1.1 shows an example of a traditional link-prediction method, which is called the common-neighbors approach, and is used to recommend a new connection between two people. In the depicted example, John and Peter (nodes of the graph) have a large number of friends in common, because of that, there will be a high probability of them knowing each other. Thus, they can be recommended to become friends in the social network [9].

Mapping a Knowledge Base into a graph, allows literals (or instances) to be represented as nodes, and the relations between them (the predicates) to be represented as edges. By simply applying link prediction to the graph we can infer new facts (not previously present in the KB), as shown in Figure 1.1. But, if we also add ontological information, for example information about the category of literals (instances), we can use link-prediction techniques to extend the ontological model itself by finding possible new relations.

Figure 1.2 shows an example of how link-prediction can be used to find a possible new relation to extend the original ontological model. The idea is similar to the one used to find new facts (instances). The difference is that, counting edges is performed to groups of nodes (from the same category), instead of to single nodes. Suppose, for instance, that many nodes from *athlete* category are related to nodes from *sport* category. In addition, suppose also that many nodes from *sport* category are related to nodes from *sportsLeague* category. In such a scenario, a relation between *athlete* and *sportsLeague* categories can be recommended, in this case the new relation could be named *athletePlaysInLeague*.

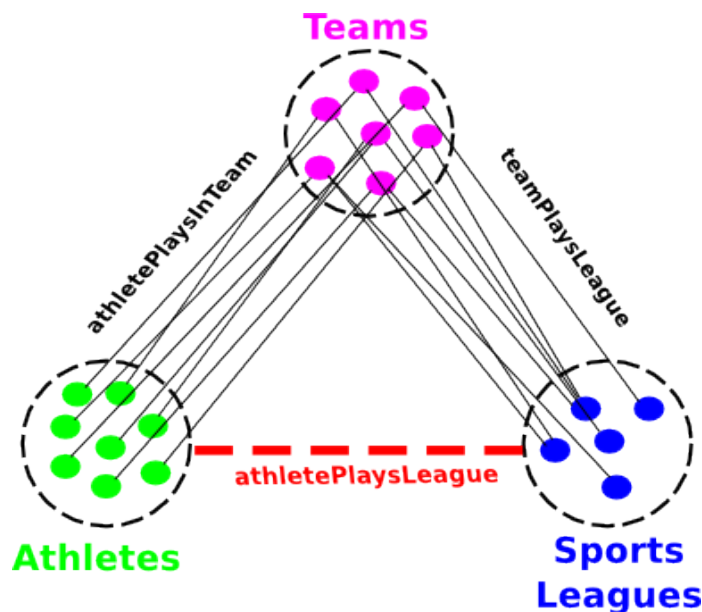


FIGURE 1.2: Link-Prediction applied to propose a new relation(*athletePlaysLeague*)

1.1.2 Using Ontological Information to Find Implicit Knowledge

The main focus of our research is to investigate algorithms that would explore the use of ontological information from an OKB to discover/extract implicit knowledge to augment the original OKB itself. Starting from the definition of a simple OKB, such as presented in this section (containing a set of categories, a set of relations among these categories, instances of the categories and relations among the instances), we intend to proceed our investigation and, also to propose approaches that can take advantage of such ontological information to find new instances and also to extend the original ontology. By ontology extension, we mean, the approach should focus also on finding new categories, as well as new relations among the categories.

In the last subsection we presented some link prediction techniques examples that help to find instances (Figure 1.1), and also to find new possible relations among categories (Figure 1.2). Beyond that, we also want to explore, in this research work, techniques to extend the ontology finding new categories. For that purpose, we could use for example another graph mining task, called community detection[11], to divide or find sub-categories for a given category already in the OKB (see Figure 1.3 for an example).

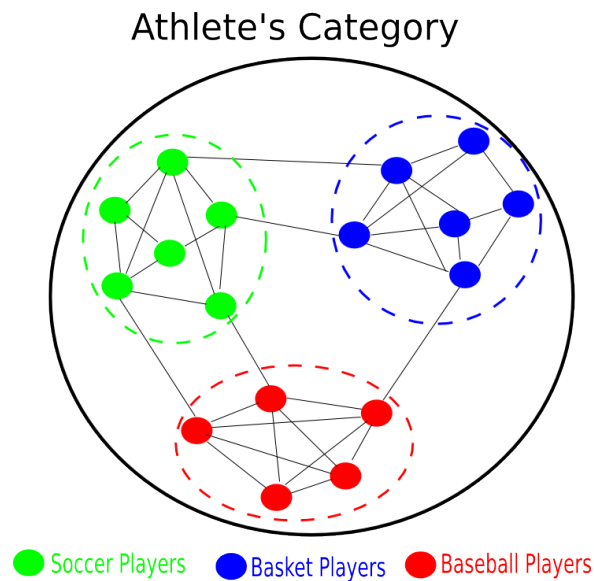


FIGURE 1.3: Athlete category, and the sub-groups (or sub-categories) that could be found with a community detection algorithm

Figure (1.3) shows an example of the task of find new categories (more specifically in this case, sub-categories of an already existing category) through community detection. In that figure, category *athlete* is present, and a community detection algorithm could find *soccer player*, *basketball player* and *baseball player* subcategories, using the topology of the graph. We can observe that the nodes of these groups are dense connected to each other, while the connection with node of other groups are very sparse.

1.2 Motivation

As we show in the beginning of this chapter, there is a lot of interest on building large OKBs, generally, by gathering data from large corpora of text, websites such as Wikipedia, or the web in general. Despite of the gigantic size of these OKBs, the techniques used to extract knowledge from text (and from semi-structured sources, like Wikipedia) generally are not enough to make a machine (computer) capable of inferring

implicit knowledge like human beings do. Thus, in addition to harvest these sources to extract knowledge, a lot of implicit knowledge might be present in these OKBs, which require inference-based algorithms to extract them. For example, consider the two follow statements: “*athlete* Neymar Jr. plays on *team* Barcelona” and “*athlete* Neymar Jr. plays on *league* Champions League”. Considering both statements, there an implicit knowledge: “*team* Barcelona plays on *league* Champions League”. A human being probably would easily infer this implicit knowledge just by reading the first two facts, but for a machine, it is not so easy to learn to infer it.

Considering its never-ending learning characteristics, NELL [1, 7] is the main motivation of the research work here proposed. To build its continuously growing knowledge base, NELL reads the web 24 hours per day, 7 days per week with two specific goals: populate its own knowledge base and improve its learning abilities. A lot of research was made to allow NELL performing these tasks based on different components (most of the research is listed in its website³) and different theoretical approaches. NELL’s different components work together extracting information and learning from the web (such as CPL and CSEAL), as well as from NELL’s own knowledge base (such as CMC, RL and PRA[12–14]).

An important characteristic of an OKB (such as NELL’s) is that the ontology structure itself can be considered to have specific patterns that can be used to infer new knowledge, thus, to learn new categories or new possible relations to be added to the OKB, would mean to obtain new knowledge. There are not many research works focused on extending an ontology by finding implicit knowledge in the form of new categories and new relations. Considering NELL, for example, there were not any active component to extend its own ontology, before this work. There were, however, two components that were previously designed to find new relations, namely Prophet[15] and OntExt[16], and there was previous research on finding sub-categories for categories already present in NELL’s own OKB based on the use of matrix factorization to cluster instances from each category[17].

The first attempt to automatically extend NELL’s OKB was designed based on a distant supervision learning approach and is named OntExt (Ontology Extender)[16]. OntExt was designed to match all pairs of categories present in NELL’s ontology. For each pair

³<http://rtw.ml.cmu.edu/rtw/publications>

of categories, the idea is to gather all instances and use them to extract features from an external textual corpus or database (such as subject-verb-object files, also known as SVO). After that, OntExt creates a *feature X feature* co-occurrence matrix and cluster these pairs of features into groups. Each group is intended to represent a new possible relation between two categories. The name of the relation would be the top ranked feature in the group (or the group centroid).

The biggest problem with OntExt is that it does not scale well. To generate the features co-occurrence matrix, the complexity is $O(n^3)$, and to cluster the matrix, OntExt uses K-Means clustering algorithm[18], that has a considerably high complexity too. Because of this high complexity issue, it wouldn't be feasible to run OntExt for all category pairs combinations in an acceptable time window, to make it iterative and never-ending like NELL.

Prophet [15] was designed to be one of NELL's ontology extension components. The main idea behind Prophet is to map NELL's own OKB into a graph structure, and then, apply link-prediction techniques into the generated graph. It executes a link-prediction task using a metric called extra-neighbors[15] to extend NELL's ontology by finding new possible relations. Also, Prophet can be used to find new instances of the proposed relations and some possible misplaced facts present on the KB.

Prophet's first implementation has some software engineering issues, like, it doesn't scale well if the input graph grows too much on the number of nodes and edges. Also, the score for the proposed relations (in its first implementation) are not precise, thus a great part of those proposed relations might not be correct. Another limitation is that the first implementation was directly coded into NELL's KB by SQL queries, having his use restricted to NELL. Besides all of that, Prophet just points two categories that might be related, and it doesn't give a name to the relation and can't tell if there is more than one possible relation among the two related categories (e.g playedAgainst(athlete, athlete), teammate(athlete, athlete)).

Based on the lack of ontology extension components to NELL (and also to other OKB-based projects), the main goals of this research were, given an Ontological Knowledge Base as input:

I Design and Implement an algorithm to find new relations in NELL's OKB;

II Design and Implement an algorithm to find new categories in NELL's OKB.

III Design and Implement an algorithm to find new facts for the OKB using the Ontology Information.

1.3 Methodology

To accomplish all goals, we focused on a graph mining approach, so the main related research works are Prophet[19] and PRA[12, 13]. Another research that worth attention, mainly for the ontology extension algorithms, is OntExt[16]. Besides the main focus being on these three related works, other contributions reported on the literature were also studied during our work.

To achieve goal I, a new version of Prophet was designed and implemented in C++ to be more generic, allowing other projects to use it (we intend to let it freely available for download in the web), and also more scalable than the original one. After that, a new OntExt version was also created and integrated with Prophet to take advantage of their complementary characteristics.

It is important to mention that, because of Prophet's intrinsic characteristics, its algorithm can be easily adapted to be used as an OKB inference method. In this sense, if instead of focusing only on *open triangles* (as done in Prophet), we focus on *closed triangles*, the algorithm can be adapted to generate first order inference rules from the OKB. Thus, as a side effect of the re-implementation of Prophet, a new inference rule extractor for OKBs was designed and implemented. Considering that the new inference algorithm is also based on link-prediction graph mining techniques, it is called the *Graph Rule Learner* - GRL. Therefore, in spite of not having the main goal of proposing a new OKB inference algorithm, this masters work contributes GRL as a side effect of its first goal, and defined as the third goal.

To achieve the second goal, an algorithm that clusters each category to find sub-categories was created. This algorithm is similar to OntExt, because it will build a features-based matrix to cluster the instances of the given category to find new sub-categories. And to achieve the third goal, as already mentioned, an algorithm that is

very similar to Prophet, was created to find patterns that are present at the graph to create inference rules to extract implicit facts.

A summarization of the methodology is presented below:

- Investigation of past research works about ontology knowledge bases, and ontology extension algorithms;
- Analysis of previous works about Prophet[15, 19] and OntExt[16] and Chunlei's category extension approach [17];
- Design and implementation of a new version of Prophet, as well as a new version of OntExt, and integrate both of them to achieve goal I;
- Investigation the use of elements of OntExt and Prophet (like the Extra-Neighbors Index) to create algorithms to achieve goal II and III;
- Implementation and validation of the algorithms designed to solve goals I, II and III;
- Integration of these algorithms to run within NELL.

1.4 Organization

In the Chapter 2, the review of the literature for the research is present, containing also one section briefly presenting NELL with its initial architecture. Chapter 3 presents the Ontological Network structure and how to map an Ontological Knowledge Base using it. This structure is used in the next chapters to explain and exemplify our projects: in Chapter 4 the one that finds new facts, in Chapter 5 the one that finds new relations and in Chapter 5.5 the one that finds new categories, all of them using the ontology to find implicit knowledge. After that in Chapter 5.7 we present some future extensions to our projects and following in Chapter 5.10 the final conclusion. Next to that are an Appendix presenting some results of the projects and following that the bibliography references.

Chapter 2

Literature Review

In this chapter we present the background theories and methods important to our masters project, and also Nell's architecture and how its OKB is defined.

2.1 Background

As aforementioned, our proposal is to investigate, design and implement algorithms to (semi)automatically extend OKBs. Our intention is to explore the possibility of mapping an input OKB into a graph, and then use graph mining techniques. The main approach in which our research is based on is link-prediction. In the last years a large number of methods has been proposed on this topic, and a review on those can be seen in [20].

Most link prediction methods are based on graph structural properties [21] in which the goal is to assign connection values, called $score(u; w)$, to pairs of nodes $\langle u, w \rangle$ based on a graph G . Traditionally, the assigned scores are later ranked in a list in decreasing order of $score(u; w)$, and then, predictions are made according to this list.

For a node u in the graph G , let $\Gamma(u)$ denote the set of neighbors of u in G . A number of link prediction approaches are based on the idea that two nodes in G (e.g u and w) are more likely to link to each other, in the future, if their sets of neighbors $\Gamma(u)$ and $\Gamma(w)$ largely overlap.

The most straightforward implementation of this link-prediction idea is the common-neighbors metric [9], under which the *scores* are defined as $score(u; w) := |\Gamma(u) \cap \Gamma(w)|$.

Common-neighbors predictor captures the basic notion, inspired in social networks, that two strangers who have a common friend may be introduced by that common friend and, thus, become friends themselves. When analyzing his *introduction* act in a graph-based context, it has the effect of “closing a triangle” in the graph and feels like a common mechanism in real life [22].

Link prediction can make use not only of graph structural information but also relational characteristics, for example attributes related with graph’s node as presented in [23]. This kind of approach is more used in relational or multi-relational learning [8, 24–26].

Besides the amount of different approaches described in the literature for the link prediction task, results presented in [27] indicate that the simplest measure, namely common neighbors has, in general, the best overall performance. Also, this very simple approach has the property of making it easy to calculate a cumulative number of neighbors if nodes were associate to a class.

Another technique that we widely use in this research is clustering. A review on those can be seen in [28]. Clustering algorithms are mostly used to solve unsupervised learning problems[29], so, as every other problem of this kind, it deals with finding a structure in a collection of unclassified data. A loose definition of clustering algorithm could be “the process of organizing objects into groups whose members are similar in some way”, so, a cluster is a collection of objects which are “similar” between them and are “dissimilar” to the objects belonging to other clusters.

One of the most famous clustering algorithm is called k-means[18]. It works iteratively creating centroids(data points in which each feature is the mean of the group’s data points feature) then relocating the data points to the closest centroid, until is error is lower to a given threshold. Generally, k-means is used to cluster data points with numerical features, so the most common distance functions used is Euclidean Distance or Manhattan distance[30]. The distance function can be different to work with discrete data (like text).

We also talk a lot about rule learners and inference rules. An inference Rule (or just **rule**) is a logical form, consisting of a conclusion r , and premises p_1, p_2, \dots, p_n . One possible representations is $r \Leftarrow p_1 \wedge p_2 \wedge \dots \wedge p_n$. The premises and the conclusion are literals that can be predicates (p), a logical function $p(x_1, x_2, \dots, x_n)$ that can only

return true or false. For example, we can have the following rule: $grandfather(A, C) \Leftarrow father(A, B) \wedge father(B, C)$, that indicates that if A is father of B , and B father of C , then A is grandfather of C .

2.2 NELL

The Read The Web project's system is called the Never Ending Language Learner (NELL) [1]. NELL is a long life learning system that reads the web 24h a day, 7 days a week, since January, 2010. In each iteration NELL main goals are to learn new facts, extends its ontology finding new categories and relations, and to improve its learning abilities.

NELL's knowledge Base can be considered an ontological knowledge base (OKB): composed by categories (e.g. person, sportsTeam, fruit, emotion, etc.) and relations (e.g. musicianPlaysInstrument(musician, instrument)). Category instances (e.g. person(Barack Obama), sportsTeam(Pittsburgh Steelers)), as well as relation instances (e.g. athletePlaysFroTeam(Ward, Pittsburgh Steelers)) are facts. NELL's OKB is divided in two main types of facts, *Candidate Facts*, are those facts that NELL has weak evidence of their truth, but not enough to be confident about them. On the other hand, *Beliefs* are facts that NELL is confident enough about their truth.

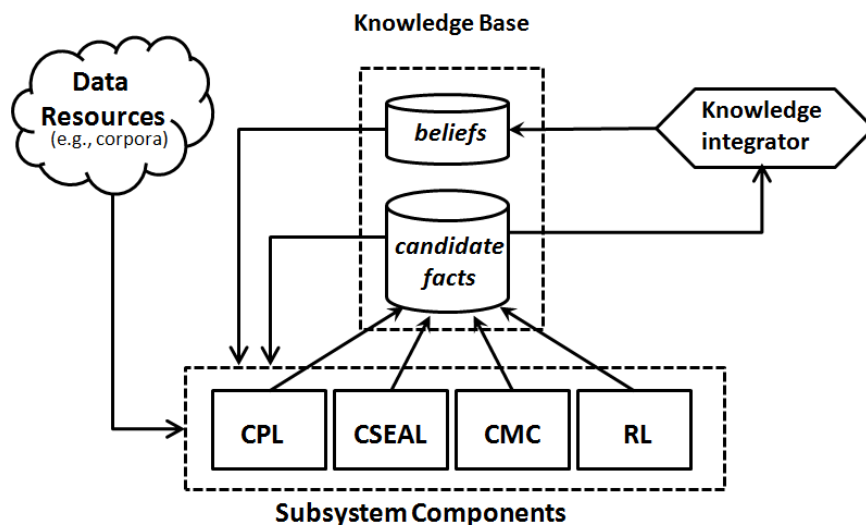


FIGURE 2.1: NELL initial core structure (figure presented at [1])

Figure 2.1 shows the core structure of NELL, its OKB and the main four components that read the web and transform the information available in knowledge. This four components are briefly describe below:

CPL: extracts knowledge using contextual patters like “mayor of x” and “X plays for Y”;

CSEAL: semi-structured extractor which queries the Internet with sets of beliefs from each category and relation, and then mines lists and tables to extract novel instances of the corresponding predicate;

CMC: based on simple set of binary L2-regularized logistic regression models which classify noun phrases based on various morphological features (words, capitalization, affixes, parts-of-speech, etc.);

RL: first-order relational learning algorithm similar to FOIL, which learns probabilistic Horn clauses from the ontology. It is possible to compare it to a procedure that identifies sets of relations (sets of three relations) that are already present in the KB and can be connected in a transitivity-based approach such as: if $\text{relation1}(A,B)$ and $\text{relation2}(B,C)$ both hold, then $\text{relation3}(A,C)$ also holds.

The main motivation of this whole Masters research project is NELL and all of the algorithms that are presented in the next chapters are intended to work as NELL’s components, automatically and iteratively finding new facts (such as the ones presented above), and also new relations and categories to extend its ontology.

Chapter 3

Ontological Networks

In this chapter we will present the Ontological Network structure, that was designed to formally map an ontological knowledge base (OKB) into a graph. This structure was designed mainly to help in the design and description of the algorithms of the projects of this research. But we believe that it can also be very helpful to other recent researches that use OKBs.

There are libraries such as `graphOnt`[31] that maps ontologies into graphs for better visualization or analysis, and also a class called `OWLGraphWrapper`¹ from `OWLTools`. These libraries are very useful to directly apply some graph-like operations over ontological knowledge bases, but none of them described a formal definition.

First there's a definitions section to formally present some notations that will be used in this chapter and in the algorithms in the next chapters either, then we formally define an ontological KB to them formally define an ontological network and the mapping process. We also present some examples in this chapter that will be used to exemplify the algorithms in the next chapter either.

3.1 Base Definitions

Let $G = (V, E)$ be an undirected graph with a set of nodes V and a set of edges E . $\Gamma(u) := \{v \in V : \exists \{v, u\} \in E\}$ of node u is defined to be the set of nodes in V that are

¹http://owltools.googlecode.com/svn/trunk/docs/api/owltools/graph/OWL_GraphWrapper.html

adjacent to u . The number of common neighbors between two nodes (u and v) can be defined as $\aleph(u, v) = |\Gamma(u) \cap \Gamma(v)|$.

A closed triangle $\Delta(u, v, w)$ of a graph $G = (V, E)$ is a set of three complete connected nodes where $u, v, w \in V$ and $\Delta(u, v, w) = \{ \langle u, v \rangle, \langle v, w \rangle, \langle w, u \rangle \} \in E$. An open triangle $\Lambda(u, w)$ of a graph $G = (V, E)$ is three connected node where $\Lambda(u, w) = \{(u, v), (v, w)\} \in E \wedge \{u, w\} \notin E$. The $\Delta_c(c_1, c_2, c_3)$ represents all the closed triangles composed by node's of categories c_1, c_2 and c_3 and $\Lambda_c(c_1, c_2)$ represents all the open triangles composed by node's categories of c_1 and c_2 (in this case, the middle nodes categories doesn't matter). Any Δ_c is called a **closed triangles category group** and any Λ_c is called a **open triangles category group**. The c_u of a node u is the set of the categories that u belongs to.

3.2 Defining an Ontological Knowledge Base (OKB)

To define an ontological network created from an OKB, we need to formally define an OKB first. In this section an OKB format we call OKB^2 is defined. To do so it is used predicate logic elements.

Definition 1. An Ontological Knowledge Base $OKB^2 = (C, H, R, I)$ is composed by four components: a set of categories(C), a set (H) of predicates “ako(a kind of)” that express the hierarchy against the categories, a set (R) of predicates that express relation among the categories.

- \forall predicates $p(t_1, t_2) \in H, R$ the arity of p is 2 ($p/2$) and $t_1, t_2 \in C$;
- The predicate “ako” $\in H$ are transitive: if $ako(t_1, t_2), ako(t_2, t_3) \in H$, then $ako(t_1, t_3) \in H$;
- It must not exist predicates with equal terms in H : $ako(t, t) \notin H$;

At last, a set of instances(I) of the categories(C) and relations(R). The instances set(I) are the facts of the OKB^2 . The set (I) can be divided in two subsets, the categorization set and the relational set ($I = I_c \cup I_r$)

- The categorization set (I_c) is composed by predicates $p_c(t)$ with names of the categories in C (p_c in C) and the arity of p_c is 1 ($p_c/1$).

- The relational set (I_r) is composed by predicates $p_r(t_1, t_2)$ such that: $p_r(c_1, c_2) \in R$ and $c_1(t_1), c_2(t_2) \in I_c$.

The number 2 in the name of our OKB model (OKB^2) was chosen because all of the relations on this model can only have arity 2. Above we present an example to better illustrate how OKB^2 works.

Example 1. Consider an ontological knowledge base $OKB_{social}^2 = (C, H, R, I)$ used to store data from a social network that have different types of relations.

- $C = \{\text{Person, MaleP, FemaleP}\};$
- $H = \{\text{ako(MaleP, Person), ako(FemaleP, Person)}\}$
- $R = \{\text{friends(Person, Person), fatherOf(MaleP, Person), motherOf(FemaleP, Person), descendant(Person, Person), relationship(MaleP, FemaleP), relationship(FemaleP, MaleP)}\}$
- $I = I_c \cup I_r = \{\text{Person(Lucas), MaleP(Lucas), Person(Jose), MaleP(Jose), Person(Raquel), FemaleP(Raquel), Person(Julia), FemaleP(Julia), Person(Vinicius), MaleP(Vinicius), Person(Leo), ...}\} \cup \{\text{friends(Lucas, Leo), friends(Vinicius, Leo), friends(Julia, Raquel), relationship(Lucas, Julia), relationship(Raquel, Jose), relationship(Jose, Raquel), descendant(Lucas, Jose), descendant(Vinicius, Jose), descendant(Vinicius, Raquel), fatherOf(Jose, Lucas), motherOf(Raquel, Lucas), motherOf(Raquel, Vinicius), ...}\}$

3.3 Defining an Ontological Network (N^o)

In this subsection we'll define an ontological network (N^o) created from an arbitrary ontological knowledge base OKB^2 .

Definition 2. An Ontological Network $N^o = (G_m, G_i)$ is composed by two graphs, an ontological model graph (G_m) and an ontological instances graph (G_i).

An Ontological Model Graph $G_m = (V_m, E_m)$ has the same purpose of the model parts on the OKB^2 , to determine the categories that the instances can be and the possible relationships between categories. The set of nodes V_m is composed by labeled nodes

that represents categories, and the set of edges E_m is composed by labeled edges that represents a relation name.

Definition 3. Given an Ontological Knowledge Base $OKB^2 = (C, H, R, I)$, an Ontological Model Graph $G_m = (V_m, E_m)$ can be created to map the ontological model of OKB^2 such as:

- $V_m \equiv C: \forall c \in C \exists v \in V_m \mid v.label = c;$
- $E_m \equiv H \cup R: \forall p(t_1, t_2) \in H \cup R \exists e \in E_m \mid e = p < t_1, t_2 >;$

To better illustrate an ontological model graph, above an example is present, mapped from the OKB^2 of Example 1.

Example 2. Given the OKB^2_{social} presented at Example 1, the ontological model graph $G_m = (V_m, E_m)$ mapped by it would be:

- $V_m = \{Person, MaleP, FemaleP\};$
- $E_m = \{ako < MaleP, Person >, ako < FemaleP, Person >, friends < Person, Person >, fatherOf < MaleP, Person >, motherOf < FemaleP, Person >, husbandOf < MaleP, FemaleP >, wifeOf < FemaleP, MaleP >, \dots\};$

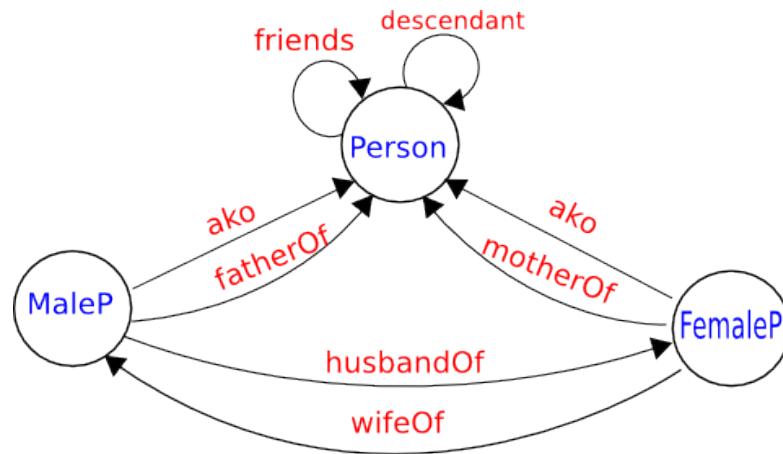


FIGURE 3.1: Ontological Model Graph Example

The **Figure 3.1** is the graphic representation of the graph presented in Example 2.

An Ontological Instances Graph $G_i = (V_i, E_i, X)$ is the graph that will be the network of the instances of the KB. The set of nodes V_i is composed by labeled nodes, representing the instances (the parameters of the predicates $\in I$). The set of edges E_i is composed by labeled edges that represents relations among two instances. The set X is the category list, having the category information for each node.

Definition 4. Given an Ontological Knowledge Base $OKB^2 = (C, H, R, I_c \cup I_r)$ and Ontological Model Graph $G_m = (V_m, E_m)$ mapped from OKB^2 , an ontological instances graph $G_i = (V_i, E_i, X)$ can be created to map the instances of OKB^2 such as:

- $\forall p_c(t) \in I_c \exists v \in V_i \mid v.label = t$;
- $E_i \equiv I_r: \forall p_r(t_1, t_2) \in I_r \exists e \in E_i \mid e = p_r < t_1, t_2 >$;
- $X \equiv I_c: \forall p_c(t) \in I_c \exists x \in X \mid x = (t, p_c)$;

To better illustrate an ontological instances graph, above an example is present, mapped from the OKB^2 of Example 2.

Example 3. Given the OKB^2_{social} presented at Example 1, the ontological instances graph $G_i = (V_i, E_i)$ mapped by it would be:

- $V_i = \{\text{Lucas, Jose, Raquel, Vinicius, Leo, Julia, ...}\}$;
- $E_i = \{\text{friends(Lucas, Leo), descendant(Lucas, Jose), fatherOf(Jose, Vinicius), relationship(Lucas, Julia), ...}\}$;
- $X = \{(\text{Lucas, Person}), (\text{Lucas, MaleP}), (\text{Raquel, Person}), (\text{Raquel, FemaleP}), (\text{Leo, Person}), \dots\}$

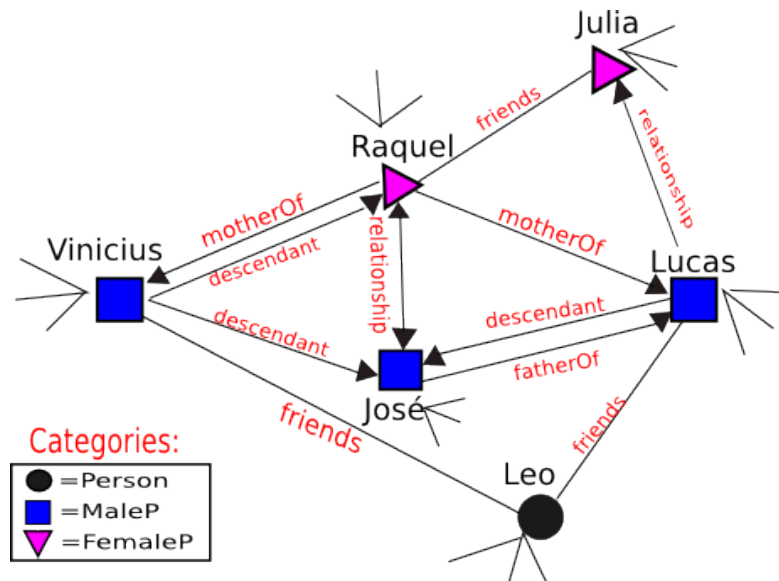


FIGURE 3.2: Ontological Instances Graph Example

The **Figure 3.2** is the graphic representation of the portion of the graph presented in Example 3 (mapped from I of Example 1). The categorization set X is represented by the node colors and shapes.

An alternative format for a N^o , that can be also very interesting to apply graph-mining algorithms, is a format with just one unified graph:

$$N^{o'} = (V_m \cup V_i, E_m \cup E_i \cup X)$$

In $N^{o'}$ there is just one graph, unifying the ontological model with the instances. In practice this network generally will look like a small world network[32], with the category nodes being the hubs.

We use an OKB^2 to build (and define) and Ontological Network, but an N^o can exist by itself, and not just mapped from an OKB. Actually, OKB^2 and N^o are equivalent models. It is possible to represent anyone of the two of them using the other.

3.4 Setting a second example

Now that OKB^2 and N^o models are already defined, in this section a second example is presented. It will be an OKB containing sport-related data. This example and the social network example will both be used in the next chapters to exemplify the algorithms created in this project. OKB^2 and N^o will be presented to better exemplify the mapping process.

3.4.1 OKB_{sports}^2

Consider an ontological knowledge base $OKB_{sports}^2 = (C, H, R, I)$ used to store data from knowledge related to sports:

- $C = \{\text{Athlete, SportsTeam, SportsLeague, Stadium, Continent}\};$
- $H = \{\}$ (*empty*)
- $R = \{\text{athletePlayedAtCountry}(\text{Athlete, Contry}), \text{athletePlaysForTeam}(\text{Athlete, Team}), \text{athletePlayedAtStadium}(\text{Athlete, Stadium}),$

`teamPlaysLeague(SportsTeam, SportsLeague),`
`teamPlayedAtCountry(SportsTeam, Country), leagueUsesStadium(SportsLeague,`
`Stadium), stadiumLocatedAtCountry(Stadium, Contry),`
`athletePlayedAtStadium(Athlete, Stadium),`
`leaguePlacedAtCountry(SportsLeague, Country)}`

- $I = I_c \cup I_r = \{Athlete(Neymar Jr.), Athlete(Jorge Valdivia), Country(Spain),$
 $Country(Brazil), Stadium(Camp Nou), Stadium(Vila Belmiro), Stadium(Allianz$
 $Pq) SportsTeam(Barcelona), SportsTeam(Palmeiras), SportsLeague(Champions$
 $League), SportsLeague(Brazillian Cup), athletePlayedAtStadium(Neymar Jr.,$
 $Vila Belmiro), athletePlayedAtStadium(Jorge Valdivia, Vila Belmiro),$
 $athletePlaysForTeam(Neymar Jr., Barcelona), athletePlaysForTeam(Jorge$
 $Valdivia, Palmeiras), athletePlayedAtCountry(Neymar Jr., Spain),$
 $athletePlayedAtCountry(Neymar Jr., Brazil), athletePlayedAtCountry(Jorge$
 $Valdivia Brazil), stadiumLocatedAtCountry(Camp Nou, Spain),$
 $stadiumLocatedAtCountry(Vila Belmiro, Brazil),$
 $stadiumLocatedAtCountry(Allianz Pq, Brazil), teamPlaysLeague(Barcelona,$
 $Champions League), teamPlaysLeague(Palmeiras, Brazillian Cup),$
 $teamPlayedAtCountry(Barcelona, Spain), teamPlayedAtCountry(Palmeiras,$
 $Brazil), leagueUsesStadium(Brazillian Cup, Allianz Pq.),$
 $leagueUsesStadium(Champions League, Camp Nou),$
 $leaguePlacedAtCountry(Champions League, Spain),$
 $leaguePlacedAtCountry(Brazillian Cup, Brazil), ... \}$

3.4.2 $N_{sports}^o = (G_m, G_i)$

Given the OKB_{sports}^2 , the ontological model graph $G_m = (V_m, E_m)$ mapped by it would be:

- $V_m = \{Athlete, SportsTeam, SportsLeague, Stadium, Continent\};$
- $E_m = \{athletePlayedAtCountry\langle Athlete, Contry \rangle,$
 $athletePlayedForTeam\langle Athlete, Team \rangle, athletePlaysForTeam\langle Athlete, Team \rangle,$
 $athletePlayedAtStadium\langle Athlete, Stadium \rangle, teamPlaysLeague\langle SportsTeam,$
 $SportsLeague \rangle, leagueUsesStadium\langle SportsLeague, Stadium \rangle,$

stadiumLocatedAtCountry<Stadium, Contry>,
 athletePlayedAtStadium<Athlete, Stadium>;

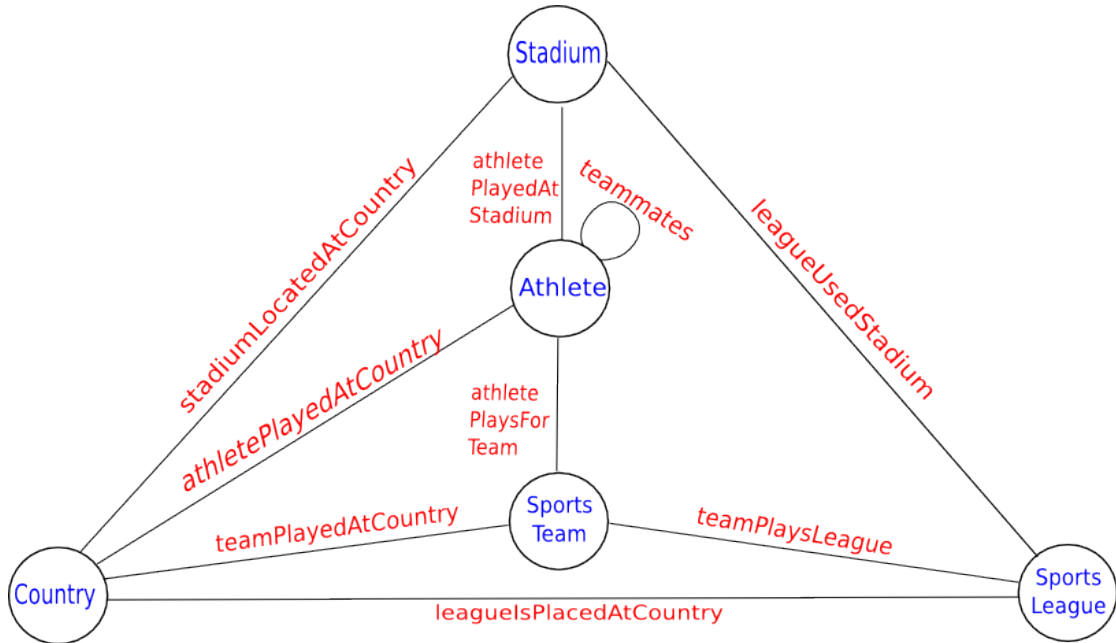


FIGURE 3.3: Ontological Model Graph G_m of Sports Network

Given the OKB_{sports}^2 , the ontological instances graph $G_i = (V_i, E_i)$ mapped by it would be:

- $V_i = \{\text{Neymar Jr.}, \text{Jorge Valdivia}, \text{Spain}, \text{Brazil}, \text{Camp Nou}, \text{Vila Belmiro}, \text{Allianz Pq}, \text{Palmeiras}, \text{Barcelona}, \text{Champions League}, \text{Brazillian Cup}, \dots\}$;
- $E_i = \{\text{athletePlayedAtStadium}\langle \text{Neymar Jr.}, \text{Vila Belmiro} \rangle,$
 $\text{athletePlayedAtStadium}\langle \text{Jorge Valdivia}, \text{Vila Belmiro} \rangle,$
 $\text{athletePlaysForTeam}\langle \text{Neymar Jr.}, \text{Barcelona} \rangle,$ $\text{athletePlaysForTeam}\langle \text{Jorge Valdivia}, \text{Palmeiras} \rangle,$
 $\text{athletePlayedAtCountry}\langle \text{Neymar Jr.}, \text{Spain} \rangle,$
 $\text{athletePlayedAtCountry}\langle \text{Neymar Jr.}, \text{Brazil} \rangle,$ $\text{athletePlayedAtCountry}\langle \text{Jorge Valdivia}, \text{Brazil} \rangle,$
 $\text{stadiumLocatedAtCountry}\langle \text{Camp Nou}, \text{Spain} \rangle,$
 $\text{stadiumLocatedAtCountry}\langle \text{Vila Belmiro}, \text{Brazil} \rangle,$ $\text{stadiumLocatedAtCountry}\langle \text{Allianz Pq}, \text{Brazil} \rangle,$
 $\text{teamPlaysLeague}\langle \text{Barcelona}, \text{Champions League} \rangle,$
 $\text{teamPlaysLeague}\langle \text{Palmeiras}, \text{Brazillian Cup} \rangle,$
 $\text{teamPlayedAtCountry}\langle \text{Barcelona}, \text{Spain} \rangle,$ $\text{teamPlayedAtCountry}\langle \text{Palmeiras}, \text{Brazil} \rangle,$
 $\text{leagueUsesStadium}\langle \text{Brazillian Cup}, \text{Allianz Pq.} \rangle,$
 $\text{leagueUsesStadium}\langle \text{Champions League}, \text{Camp Nou} \rangle,$

leaguePlacedAtCountry<Champions League, Spain>,
 leaguePlacedAtCountry<Brazilian Cup, Brazil>, ...};

- $X = \{(Neymar Jr., Athlete), (Jorge Valdivia, Athlete), (Spain, Country), (Brazil, Country), (Camp Nou, Stadium), (Vila Belmiro, Stadium), (Allianz Pq, Stadium), (Barcelona, SportsTeam), (Palmeiras, SportsTeam), (Champions League, SportsLeague), (Brazilian Cup, SportsLeague), \dots\}$

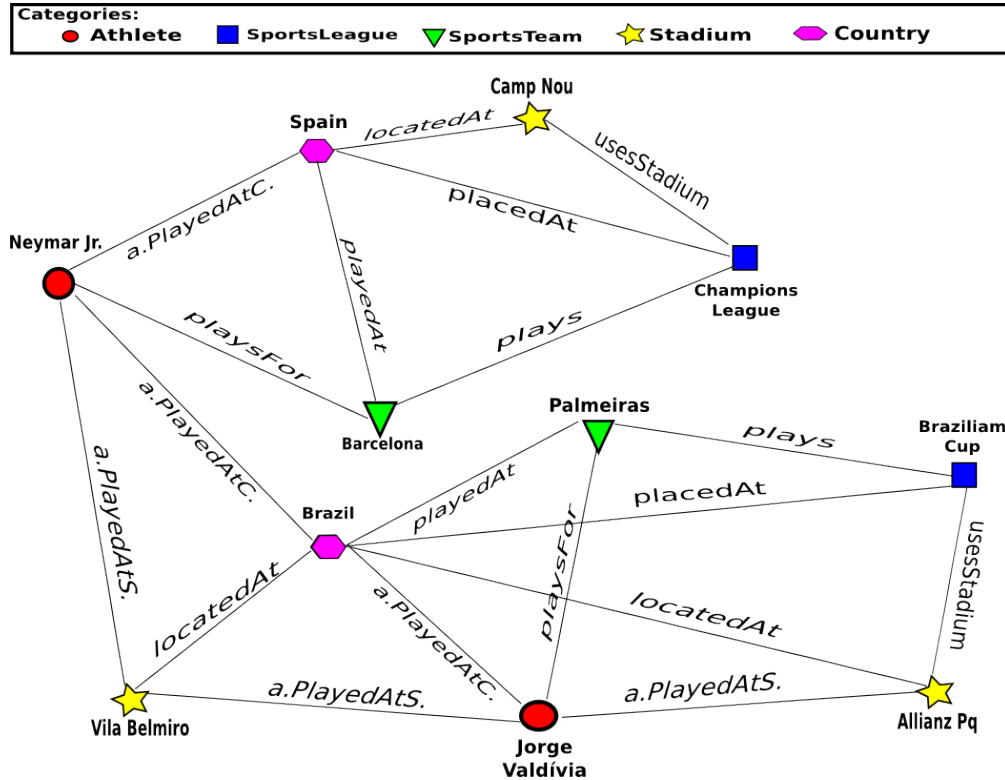


FIGURE 3.4: ontological instances graph G_i of Sports Network

In this section we presented the sports OKB OKB_{sports}^2 then we formally present the Ontological Network N_{sports}^o with both in text and in figures to the better visualization of it. One interest observation is that mostly of the projects that uses OKBs such as NELL, there are a lack of facts (and some wrong facts either). Observing the N_{sports}^o example, a human being might know that Neymar Jr. played at Camp Nou lots of times because it plays for Barcelona and Camp Nou is Barcelona's home stadium, but this fact is missing. This is very common in the OKB of a continuous learning program such as NELL because it cannot read everything instantaneously, it will be learning more and more over time, but always it will have more to learn.

Chapter 4

Finding new facts

Even in a never-ending learning approach, the question of how to develop methodologies to help populating OKBs with facts and improving their coverage is still a challenge [14]. Thus, the use of a rule-based inference approach (here called *Rule Learner* - RL) can have relevant impact in the KB population task. In general, the goal of a RL is to induce inference rules from structured or unstructured data[33].

An Inference Rule (or just **rule**) is a logical form, consisting of a conclusion r , and premises p_1, p_2, \dots, p_n . One possible representations is $r \Leftarrow p_1 \wedge p_2 \wedge \dots \wedge p_n$. The premises and the conclusion are literals that can be predicates (p), a logical function $p(x_1, x_2, \dots, x_n)$ that can only return true or false. For example. we can have the following rule: $grandfather(A, C) \Leftarrow father(A, B) \wedge father(B, C)$, that indicates that if A is father of B , and B father of C , then A is grandfather of C .

In the context of graphs mapping knowledge bases we can use techniques similar to link-prediction to find patterns of *closed triangles* instead of *open triangles*. Following there's an example using the model of N^o proposed to explain how a RL can use these patterns to find inference rules.

If we create an algorithm that exploits patterns of *close triangles*, in a graph based on three category nodes and the relations among them, it could be applied to the graph of the Figure 4.1 and it could find $\Delta MaleP, Person, FemaleP$ as an interesting group to promote a rule. It would happen because there are a lot of triangles with these 3 categories in the same "position" in terms of relations. In this case, the following rule could be proposed:

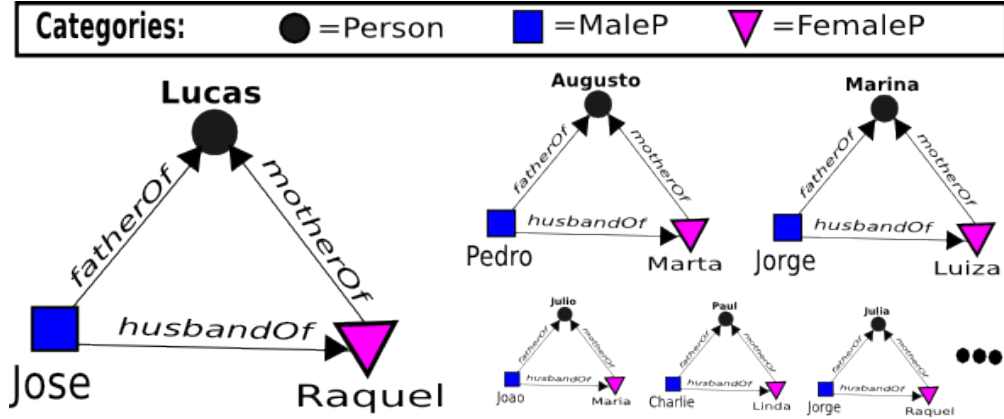


FIGURE 4.1: Example of a pattern of closed triangles between MaleP, Person and FemaleP

$$\begin{aligned} \text{husbandOf}(\text{MaleP} : X, \text{FemaleP} : Y) \Leftarrow \\ \text{fatherOf}(\text{MaleP} : X, \text{Person} : Z) \wedge \text{motherOf}(\text{FemaleP} : Y, \text{Person} : Z) \end{aligned}$$

The rule says that if a MaleP X is father of a Person Z and a FemaleP Y is the mother of the same Person Z , then MaleP X is the husbandOf FemaleP Y .

This rule can be used to find new instances of the relation husbandOf to the N_{social}^o , adding edges to G_i (the same thing as adding facts to the OKB).

4.1 The Graph Rule Learner

The Graph Rule Learner (GRL) is an algorithm designed to extract inference rules from ontological knowledge bases. GRL uses a link-prediction metric called extra-neighbors[15] to rank possible rules, and also to determine the antecedents and consequents of each induced rule.

Rule induction from data is not a novel task and many different approaches have been proposed. Due to space constraints, in this section we focus on more recent approaches and which are closely related to GRL. The *Online Rule Learner* (ORL) [33] mines inference rules from explicit information extracted from large corporas using automated information extraction (IE) systems [34, 35]. ORL is similar to GRL in the sense that it maps input corpora into a graph-based representation. Differently from GRL, however,

ORL uses the topology of the created graph to extract rules, instead of link prediction techniques used in GRL. The Universal Schema proposed in [36] focuses on the benefits of using latent features for increasing coverage of KBs. Key differences between that approach and the one proposed in the work described in our paper include our use of graph-based link prediction measurements as opposed to surface-level patterns in theirs, and also the ability of the proposed GRL method to generate useful (and comprehensible) inference rules which is beyond the capability of the matrix factorization approach.

A traditional approach to extract inference rules is the inductive logic programming (ILP), which deduces rules from ground facts. According to [37], current ILP systems cannot be applied to KBs who gathers data from web with a large scope of categories (anything in the world), such as NELL, mainly because they usually require negative statements as counter-examples, and these projects just hold instances that they consider correct or have some confidence¹. Also, the ILP-based approach don't scale to the huge amount of data that these kind of KBs store.

Regarding NELL's, when considering its KB as input to induce inference rules, there are other previously proposed approaches. In [38] a Markov Logic approach is used to allow inference over subsets of categories and relations. PRA[12], and the Latent PRA [14] and Prophet[15] are graph-based approaches. PRA (Path Ranking Algorithm) uses a combination of constrained, weighted, random walks through NELL's KB graph to reliably infer new beliefs for it's KB. PRA performs such inference by automatically learning semantic inference rules over the KB [13]. Latent PRA proposes the addition of edges labeled with latent features mined from a large dependency parsed corpus of 500 million Web documents to improve performance of previous PRA. Recently the PRA was combined with a vector space representations of surface forms to increase its performance and be capable of execute with the sparsity of textual representations from surface text[39].

Prophet, is not an inference component. However, it is important to mention this component because GRL approach is closely related to Prophet graph mining approach. Differently from PRA approaches (which are based on random walks), Prophet counts on NELL's KB (represented as a graph) as input to (semi-)automatically extend NELL's initial KB. As aforementioned, Prophet does not induce inference rules from the KB,

¹Ontology properties such as mutual exclusion can be used to solve part of this problem as done in [1, 4]

but the same link prediction idea used in Prophet to extend the KB, is adapted in GRL to induce inference rules.

According to [33], there are two problems with most of the existing inference rule learners: [40] [41] [42] [43], they do not scale when based on large corpora and they tend to assume that the training data is largely accurate and complete. However, to be coupled to a never-ending learning system, such as NELL, a RL must overcome both issues. It happens mainly because NELL's KB is continuously growing and continuously being updated and revised by NELL's components.

GRL assumes that the training data is mostly (but not completely) accurate and complete. However, it is not a problem if the KB is either imperfect, or incomplete. Actually, link prediction algorithms assume that the missing links are due to the KB evolution in the near future, thus it is currently incomplete. To take advantage of more accurate knowledge, GRL limits its learning process to the specific part of NELL's OKB called the set of *beliefs* (composed just by high confidence facts). Regarding scalability, GRL scales with large graphs (the same as large KB's), using a graph disk structure called GraphDB-Tree [44].

4.1.1 GRL Algorithm

GRL needs an ontological instances graph as input, and its output is a list of induced inference rules.

Algorithm 1 The GRL

Require: $G_i = (V, E, X)$

Ensure: List of Inference Rules

- 1: Find all $\Delta(u, v, w)$ in G_i
 - 2: **for all** closed triangle $\Delta(u, v, w)$ **do**
 - 3: Calculate $\aleph(u, v)$, $\aleph(v, w)$ and $\aleph(w, u)$
 - 4: Group $\Delta(u, v, w)$ in $\Delta_c(c_u, c_v, c_w)$
 - 5: Group $\Lambda(u, v)$ in $\Lambda_c(c_u, c_v)$, $\Lambda(v, w)$ in $\Lambda_c(c_v, c_w)$ and $\Lambda(w, u)$ in $\Lambda_c(c_w, c_u)$
 - 6: **end for**
 - 7: **for all** $\Lambda_c(c_i, c_j)$ **do**
 - 8: Calculate $\aleph_c(c_i, c_j)$
 - 9: **end for**
 - 10: **for all** $\Delta_c(c_u, c_v, c_w)$ **do**
 - 11: Find the category pair with highest \aleph_c :
 $(c_i, c_j) = \text{MAX}(\aleph_c(c_u, c_v), \aleph_c(c_v, c_w), \aleph_c(c_w, c_u))$
 - 12: **if** $\aleph_c(c_i, c_j) \geq \xi$ **then**
 - 13: Validate the rule: $r_{c_i c_j}(c_i, c_j) \Leftarrow r_{c_i c_k}(c_i, c_k) \wedge r_{c_k c_j}(c_k, c_j)$
 - 14: **end if**
 - 15: **end for**
-

In **line 1**, GRL finds and lists all closed triangles $\Delta(u, v, w)$ present in graph G_i . Then, for each triangle, the number of neighbors \aleph between each pair of nodes is calculated (e.g $\aleph(u, v)$), and grouped in the respective open triangle category group Λ_c ² (e.g $\Lambda(c_u, c_v)$). The closed triangle is also grouped in the closed triangle category group $\Delta_c(c_u, c_v, c_w)$. In **line 8**, for each open triangle category group $\Lambda_c(c_i, c_j)$, the number of extra neighbors \aleph_c is calculated. \aleph_c is the sum of the $\aleph - 1$ of all instances $\Lambda(i, j)$ in the group. If $\aleph_c(c_i, c_j) = 0$ it indicates that all pair of nodes $\Lambda(i, j)$ in the group have only one neighbor in common. In **line 11**, for each closed triangle category group $\Delta_c(c_u, c_v, c_w)$, the pair of categories with the highest extra neighbors value \aleph_c will be selected (e.g (c_u, c_v)). Then, if the extra neighbor value of this pair is greater or equal than a given threshold ξ , the rule $r_{c_u c_v}(c_u, c_v) \Leftarrow r_{c_u c_w}(c_u, c_w) \wedge r_{c_w c_v}(c_w, c_v)$ is validated. One literal $r_{c_x c_y}(c_x, c_y)$ indicates a relation(predicate) $r_{c_x c_y} \in E_c$ between the categories c_x and c_y , and its parameters must be instances of categories c_x and c_y respectively.

4.1.2 Example

In **Figure 4.2**, a simple example of the GRL algorithm for an arbitrary graph is presented.

We have the closed triangle category group Δ_c (*Athlete, Stadium, Country*), and its six instances:

$$\begin{aligned} &\Delta(\textit{Neymar Jr.}, \textit{Camp Nou}, \textit{Spain}), \\ &\Delta(\textit{Neymar Jr.}, \textit{Vila Belmiro}, \textit{Brazil}), \\ &\Delta(\textit{Jorge Valdivia}, \textit{Allianz Parque}, \textit{Brazil}) \textit{ and} \\ &\Delta(\textit{Neymar Jr.}, \textit{Pacaembu}, \textit{Brazil}), \\ &\Delta(\textit{Jorge Valdivia}, \textit{Morumbi}, \textit{Brazil}), \\ &\Delta(\textit{Le'veon Bell}, \textit{Heinz Field}, \textit{USA}). \end{aligned}$$

²Despite the three nodes are connected, GRL considers that the edge between the pair of parameters of Λ does not exist in each group

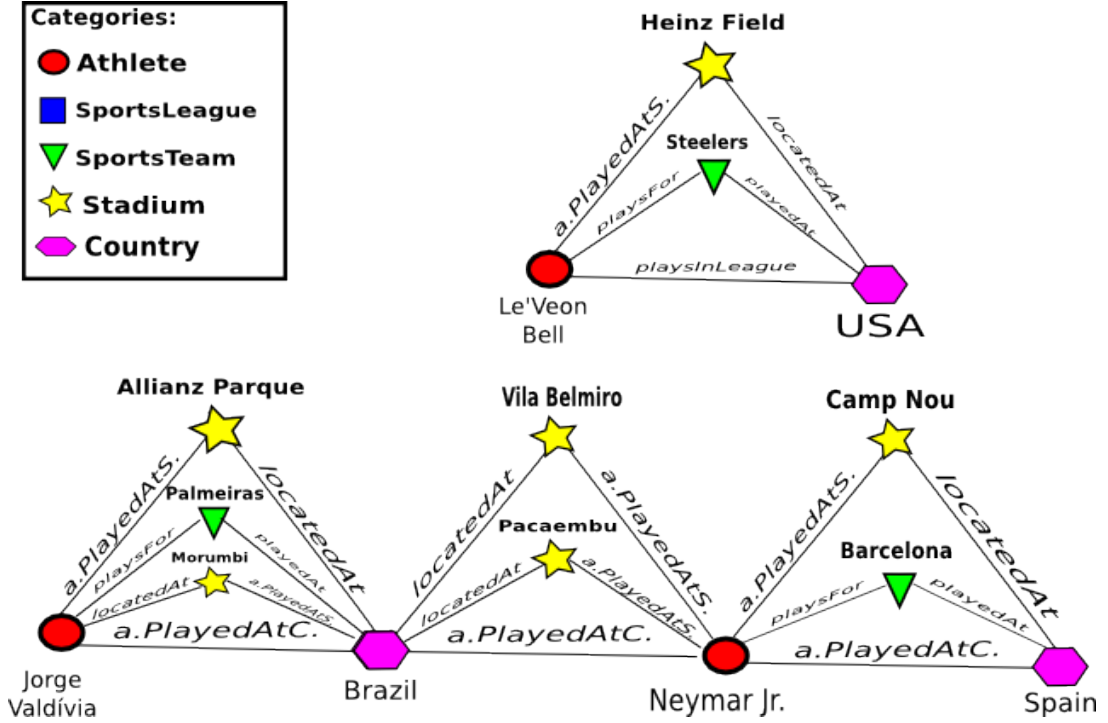


FIGURE 4.2: GRL running example

Now we have to pick the pair of categories among this three categories with the greatest extra-neighbor value, and the pair with the greatest is Athlete and Country:

$$\begin{aligned}
 \aleph_c(\textit{Athlete}, \textit{Country}) &= \aleph(\textit{Neymar Jr.}, \textit{Brazil}) \\
 &+ \aleph(\textit{Neymar}, \textit{Spain}) \\
 &+ \aleph(\textit{JorgeValdivia}, \textit{Brazil}) \\
 &+ \aleph(\textit{Le'veonBell}, \textit{USA}) \\
 &- |\Lambda_c(\textit{Athlete}, \textit{Country})|
 \end{aligned}$$

$$\aleph_c(\textit{Athlete}, \textit{Country}) = 2 + 2 + 3 + 2 - 4 = 5 \geq \xi (\xi = 5)$$

The other two pairs have a lower EN value ($\aleph_c(\textit{Stadium}, \textit{Country}) = \aleph_c(\textit{Stadium}, \textit{Athlete}) = 0$). If we consider a threshold equals to five ($\xi = 5$) then we can validate the rule among the three categories (Athlete, Stadium and Country) with Athlete and Country being the head because $\aleph_c(\textit{Athlete}, \textit{Country}) = 5$. Since there's just one relation between the

three categories of the group, the rule will be:

$$\begin{aligned} &athletePlayedAtCountry(Athlete, Country) \Leftarrow \\ &athletePlayedAtStadium(Athlete, Stadium)) \\ &\wedge stadiumIsLocatedAtCountry(Stadium, Country) \end{aligned}$$

The rule says that if an athlete **X** played at a Stadium **Y** and this Stadium **Y** is located at Country **Z**, then athlete **X** already played at Country **Z**. The generalization of the rule to X,Y and Z process will be present in a subsection above.

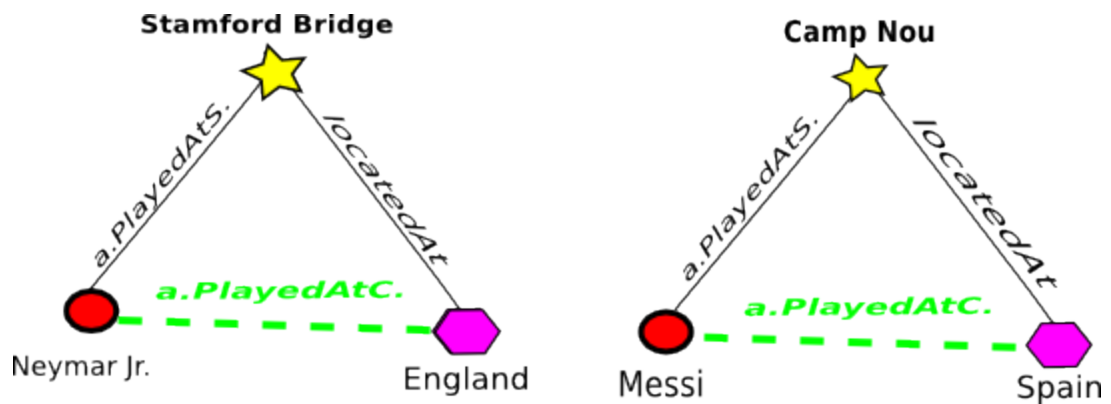


FIGURE 4.3: Applying a rule to infer new facts

After GRL found a rule, then it is possible to use the rule to infer new facts to the input OKB. Let's suppose we have open triangles $\Lambda(NeymarJr., England)$ and $\Lambda(Messi, Messi)$ (see Figure4.3), using the rule GRL just discover it is possible to add two facts to the OKB ($athletePlayedAtCountry(NeymarJr., England)$ and $athletePlayedAtCountry(Spain)$), closing the triangles because we have:

$$\begin{aligned} &1.athletePlayedAtStadium(NeymarJr., S.Bridge) \\ &\quad \wedge stadiumIsLocatedAtCountry(S.Bridge, England) \\ &\quad \Rightarrow athletePlayedAtCountry(NeymarJr., England) \\ &2.athletePlayedAtStadium(Messi, CampNou) \\ &\quad \wedge stadiumIsLocatedAtCountry(CampNou, Spain) \\ &\quad \Rightarrow athletePlayedAtCountry(Messi., Spain) \end{aligned}$$

In this example, there is also another possible closed triangle category group: Δ_c (*Athlete*, *SportsTeam*, *Country*) but no pair of categories (Λ) that belongs to these groups achieves an \aleph_c greater or equal to 5 in this example.

4.1.3 Multi Relation CTCGs

In a closed triangle category group, more than one triple of relations is possible, let's suppose that the relation *athletePlayedAgainstTeam* \langle *athlete*, *SportsTeam* \rangle is added to our sports ontological network (N_{sports}^o) with the edges *playsAgainst* \langle *Neymar Jr.*, *Real Madrid* \rangle and *playedAt* \langle *Real Madrid*, *Spain* \rangle . With this changes, the graph of Figure 4.2 will be changed: see at Figure 4.4 for the changes.



FIGURE 4.4: GRL example for multi relation CTCGs

If those two edges exist, then the closed triangle Δ_c (*Athlete*, *SportsTeam*, *Country*) will have a pair for which \aleph_c is higher than the threshold:

$$\begin{aligned} \aleph_c(\textit{Athlete}, \textit{Country}) &= \aleph(\textit{Neymar Jr.}, \textit{Spain}) \\ &\quad + \aleph(\textit{JorgeValdivia}, \textit{Brazil}) \\ &\quad + \aleph(\textit{Le'veonBell}, \textit{USA}) \\ &\quad - |\Lambda_c(\textit{Athlete}, \textit{Country})| \end{aligned}$$

$$\aleph_c(\textit{Athlete}, \textit{Country}) = 3 + 3 + 2 - 3 = 5 \geq \xi (\xi = 5)$$

We can then, validate a rule having categories Athlete and Country as the head, but we have multiple relations among the three categories of this group (see instances: Δ (*Neymar Jr., Barcelona, Spain*) and Δ (*Neymar Jr., RealMadrid, Spain*)). The triple of relations for the first one is (athletePlaysForTeam, teamPlayedAtCountry, athletePlayedAtCountry) and for the second is (athletePlayedAgainstTeam, teamPlayedAtCountry, athletePlayedAtCountry). Having multiple triples of relations inside the same group, indicates that multiple rules can be created, so it is necessary to decide how to choose among these multiple triples of relations.

To solve the problem of choosing one rule from all the possible ones (for a single group) GRL, for now, just counts the occurrence of each triple inside a closed triangle category group and pick just the one that occurred more frequently. For the example above, we have the two rules:

- $$\begin{aligned}
 (a) \quad & \text{athletePlayedAtCountry}(\text{Athlete}, \text{Country}) \Leftarrow \\
 & \quad \text{athletePlaysForTeam}(\text{Athlete}, \text{SportsTeam}) \\
 & \quad \wedge \text{teamPlayedAtCountry}(\text{SportsTeam}, \text{Country}) \\
 (b) \quad & \text{athletePlayedAtCountry}(\text{Athlete}, \text{Country}) \Leftarrow \\
 & \quad \text{athletePlayedAgainstTeam}(\text{Athlete}, \text{SportsTeam}) \\
 & \quad \wedge \text{teamPlayedAtCountry}(\text{SportsTeam}, \text{Country})
 \end{aligned}$$

Rule (a) relations combination occurred three times - with (Neymar Jr., Barcelona, Spain),(Le’Veon Bell, Steelers, USA) and (Jorge Valdivia, Palmeiras, Brazil), while rule (b) relations combination occurred just once, with (Neymar Jr., Real Madrid, Spain). Because of that, rule (a) will be validated and rule(b) will be discarded. In this case rule (a) makes more sense than rule (b), but in some cases more than one rule could be correct, and because of that, in the future we plan to explore other possibilities to pick up rules in case of Multi Relation CTCGs.

4.1.4 Grouping and Ranking Rules

When using NELL’s KB (as well as any other OKB) as input, it is expected to find several repeated rules in terms of predicates, but with different categories as parameters. This

is expected mainly because in such OKBs, there are hierarchy among categories and multi-categorized instances. See the example below of GRL's output on NELL's graph:

$$\begin{aligned} &teamPlaysSport(sportsTeam, sport) \Leftarrow athletePlaysSport(sport, athlete) \\ &\quad \wedge athletePlaysForTeam(athlete, sportsTeam) \\ &teamPlaysSport(sportsTeam, sport) \Leftarrow athletePlaysSport(sport, personAsia) \\ &\quad \wedge athletePlaysForTeam(personAsia, sportsTeam) \\ &teamPlaysSport(sportsTeam, sport) \Leftarrow athletePlaysSport(sport, personUsa) \\ &\quad \wedge athletePlaysForTeam(personUsa, sportsTeam) \end{aligned}$$

Following along these lines, a **grouping** and **ranking** process is applied on the GRL output. This process consists on simply grouping all rules, which share repeated predicates, in one single generic rule with variables (X , Y and Z) as parameters, and ranking these rules by the number of occurrences on GRL's output list. After this process, we can use this *rank* to increase confidence in some of the rules. Experiments over this process are present in Section [5.6.2](#).

4.1.5 GRL Implementation

One common problem when implementing a graph mining algorithm is scalability, mainly when working with graphs having a growing number (from hundreds to millions and sometimes billions) of nodes and edges. To cope with this scalability issue, GRL stores the graph representation in disk using a structure called GraphDB-Tree [\[44\]](#).

The GraphDB-Tree is a structure created for fast storage and recovery of a graph on secondary memory. The complexity to recover the neighbor list for any node is $O(1)$, so this structure is very efficient to graph algorithms that uses just the locality of the nodes (e.g., find graph cliques – such as triangles -, calculate Adamic/Adar and Jaccard index, etc).

To achieve high performance in the node's locality algorithms, GraphDB-Tree stores the graph partitioned in disk pages, and the entire set of nodes being numeric, sorted and continuous from 1 to $|V|$. Most of the graphs, such as the ones used in the experiments,

does not have such specific characteristics (sorted numeric nodes), so a preprocessing process is necessary before the storage on GraphDB-Tree.

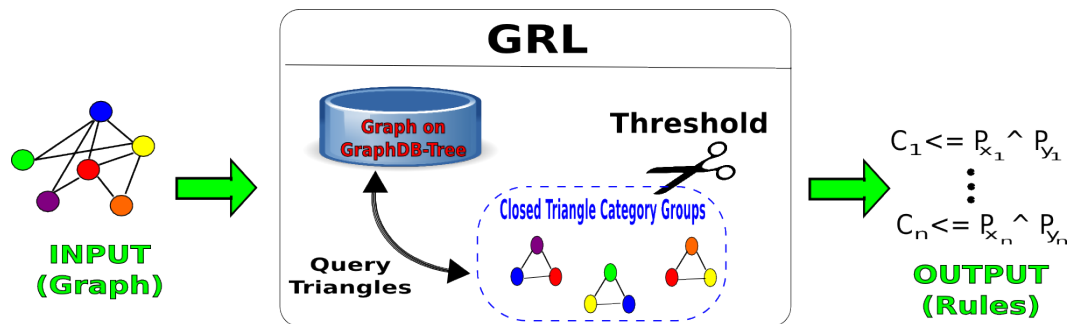


FIGURE 4.5: GRL Execution Diagram

In **Figure 4.5**, we show a simple diagram of GRL implementation: It receives as input an OKB file (mapped as a graph), pre-processes the file and stores the graph in the GraphDB-Tree data structure. Then, GRL initiates its execution querying for closed triangles in the graph and grouping them into their respective category groups. After all triangles were found, extra neighbors values are calculated and GRL validates the extracted rules based on the given threshold value. The output will be the list of inference rules.

4.2 Experiments

In this section we show some results (inference rules) from a GRL experiment using both NELL's KB, as well as YAGO's KB as input. Also, we present experiments running GRL based on different link-prediction scores in place of extra-neighbors to validate rules. Last, it is performed a comparative analysis of GRL, AMIE[37] and PRA[12]³.

4.2.1 Finding Inference Rules with GRL

4.2.1.1 GRL applied to NELL's KB

In this experiment, NELL's KB, also called *rtwgraph*, was used as input for GRL. NELL's KB is automatically extended and populated in an iterative fashion. For this experiment we use the KB from iteration 820. At iteration 820, *rtwgraph* has around 700.000

³All the experiments were performed using a personal computer with Intel(R) Core™ i72.49Hz with 6GB of RAM and on Linux Ubuntu 12.04 (32 bits)

nodes and 500.000 edges. Using threshold equal to ten ($xi = 10$), the output rule list rl_1 contains 3.780 rules before the grouping process. Two examples extracted from rl_1 rules are presented below, more can be seen in the appendix.

$$\begin{aligned}
 R_1. \text{teamplaysport}(\text{sportteam}, \text{sport}) &\Leftarrow \text{athleteplaysport}(\text{sport}, \text{personUsa}) \\
 &\quad \wedge \text{athleteplaysforteam}(\text{personUsa}, \text{sportteam}) \\
 R_2. \text{headquarteredin}(\text{city}, \text{company}) &\Leftarrow \text{atlocation}(\text{company}, \text{buildingfeature}) \\
 &\quad \wedge \text{atlocation}(\text{buildingfeature}, \text{city})
 \end{aligned}$$

4.2.1.2 Grouping and Ranking Repeated Rules.

As we are working with an ontological graph, lots of rl_1 rules are repeated, where only the parameters of relations are different. Thus, as previously mentioned, one extra grouping step is needed. Grouping this rules into generic ones and ranking them (based on the number of times it is repeated in rl_1) generates a new rule list rl_2 , with 870 rules. Two of the top ranked rl_2 rules are presented below, see more in the appendix.

$$\begin{aligned}
 R_3. \text{athleteplaysport}(X, Z) &\Leftarrow \text{teammate}(X, Y) \wedge \text{athleteplaysport}(Y, Z) \\
 R_4. \text{animalistypeofanimal}(X, Z) &\Leftarrow \text{animalistypeofanimal}(X, Y) \\
 &\quad \wedge \text{animalistypeofanimal}(Y, Z)
 \end{aligned}$$

When running this experiments with NELL's KB, some rules with generic relations, such as *everypromotedthing* and *proxyfor* were induced. This kind of rules is automatically removed from the output, because it tends to be noisy. As there is just a few generic relations, it is better to manually create rules for them.

4.2.1.3 GRL applied to YAGO KB

As mentioned before, YAGO[4] is an OKB mined from wordnet⁴ and wikipedia⁵. In this experiment, YAGO's KB was used as input for GRL, its ontology is organized in a different way than NELL's, but it also has categories for the instances, so it is suitable to

⁴<https://wordnet.princeton.edu/>

⁵<http://en.wikipedia.org/>

be used as input for GRL. The only problem we had was that YAGO’s ontology has a big hierarchy with tens of thousands of categories (while NELL has less than a thousand)⁶, so, GRL grouping process was not very effective. It is interesting to notice that YAGO is currently in its third version, called YAGO3[6], where multi language knowledge was gathered. In its second version, called YAGO2[5], it had gathered temporal relations (and instances)⁷.

We could extract around 350.000 categorized nodes and 550.000 edges from YAGO’s KB. Using threshold equal to ten ($xi = 10$), the output rule list rl_1 contains 286 rules before the grouping process. After the grouping and ranking process, the output rule list rl_2 contains 88 rules. Two examples from rl_2 are presented below, more can be seen in the appendix.

$$R_5. \textit{locatedIn}(X, Z) \Leftarrow \textit{hasCapital}(X, Y) \wedge \textit{locatedIn}(Y, Z)$$

$$R_6. \textit{hasPredecessor}(X, Z) \Leftarrow \textit{hasPredecessor}(X, Y) \wedge \textit{hasPredecessor}(Y, Z)$$

Again, the low number of rules, may be due to the detailed (deep hierarchy) ontology, having lots of weak category groups instead.

4.2.1.4 Validating rules

GRL algorithm (and most other link-prediction algorithms) does not present 100% precision. We add the group process to make the output rules more generic, and we use the rank given on this process to help enhance confidence in some rules. In table 4.1 the precision curve over this rank is present for NELL and YAGO experiment.

Table 4.1 contains statistics captured by selecting rules on the grouped list by the *rank* given on the group process. The columns are respectively: the values of *rank* used, the number of total rules of rl_2 with greater or equal first columns value and the percentage of correct rules (precision), the two last columns are repeated, one time for GRL running on NELL, and a second time for GRL running on Yago.

⁶Yago has very specific categories, such as: “Bob Dylan albuns” and “String quartets by Ludwig van Beethoven”

⁷See in its web site for more <http://www.mpi-inf.mpg.de/departments/databases-and-information-systems/research/yago-naga/yago/>

TABLE 4.1: Applying rank as threshold (GRL’s with $\xi = 10$)

GRL’s output precision by rank				
rank \geq	NELL		YAGO	
	Total Rules	Correct Rules	Total Rules	Correct Rules
20	13	61.64%	3	100.00%
15	20	60.00%	3	100.00%
10	31	54.84%	7	71.42%
9	38	60.53%	8	62.25%
8	44	61.36%	9	66.66%
7	51	56.86%	9	66.66%
6	74	51.35%	12	58.33%
5	100	47.00%	15	53.33%
4	141	43.97%	17	52.94%

Despite the fact that the proposed rank can help to give more confidence to some rules (since for lower ranked rules the precision tends to fall as shown in Table 4.1), it is easy to see that using the rank as a fixed threshold tends to promote the extraction of wrong rules. It is important to recall that in a never-ending learning environment (such as NELL), wrong knowledge (generated by wrong rules) can be propagated and deteriorate the whole KB (because of semantic drift).

To automate the identification of correct and wrong rules it is possible to train a simple classification model using algorithms such as decision trees or logistic regression, but to achieve good results in this task, enough labeled training examples is needed. Therefore, manually classifying rules is an interesting option. In NELL, GRL will be running every x iteration (probably $x=5$, which would take about 10-15 days), so we estimate that if we validate around 50 rules from each iteration of GRL, we would not expend more than one hour/month⁸.

4.2.2 Using different link-prediction metrics

In addition to the extra-neighbors (EN) metric, we’ve also performed experiments using *rtwgraph* as input, and common-neighbors (CN), Jaccard(Jac) and Salton(Sal) metrics during the rule extraction process.

In **Table 4.2** we show the manually defined threshold value used in GRL for each metric, the number of Total Rules returned and the number of rules left after the grouping

⁸NELL already has around 15 minutes/week that can be used by the system to ask questions through human supervision

TABLE 4.2: Experimenting different LP metrics: Overall Statistics

Statistic	Metrics			
	EN	CN	Jac	Sal
Threshold used	10	16	0.2	0.5
Rules Found	3780	3996	3404	3517
Grouped Rules	870	884	852	874

TABLE 4.3: Experimenting different LP metrics: Precision statistics using rank as threshold

-	Extra-Neighbors		Common-Neighbors		Jaccard		Salton	
	Rules	Precision	Rules	Precision	Rules	Precision	Rules	Precision
rank \geq 30	6	50.00%	6	33.33%	5	40.00%	5	20.00%
25	8	50.00%	8	37.50%	9	55.56%	5	20.00%
20	13	61.54%	12	50.00%	11	54.54%	13	46.15%
15	20	60.00%	20	60.00%	15	60.00%	13	46.15%
10	31	54.84%	35	45.71%	31	48.39%	30	50.00%
9	38	60.53%	41	51.22%	37	48.65%	38	52.63%
8	44	61.36%	47	51.06%	44	43.18%	42	52.38%

process. The threshold values vary mainly because we wanted all experiments to generate the closest number of rules possible, to better compare the precision of each one. Also, each one of this metrics has different range(magnitude) of values. In **Table 4.3** we present the precision of each metric selecting grouped rules using *rank* as threshold.

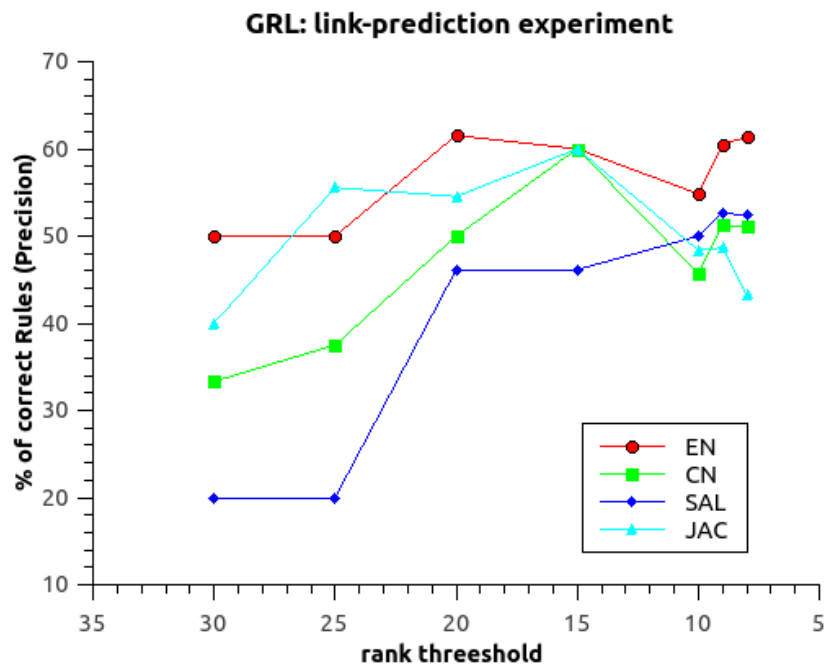


FIGURE 4.6: Precision curve using different LP metrics

-	<i>RulesFound</i>	<i>Precision</i>	<i>Gen.Facts</i>	<i>Format</i>
GRL	459	42/100 = 42%	115.59	$r \Leftarrow p_1 \wedge p_2$
AMIE	982	35/100 = 35%	89.25	$r \Leftarrow p_1 \wedge p_2 \vee r \Leftarrow p$
PRA	49966	33/100 = 33%	–	$r \Leftarrow p_1 \wedge p_2 \dots \wedge p_n \ (n \geq 1)$

Figure 4.6 depicts the precision curve over the rank threshold for each used metric. The extra-neighbors(EN) is in red, the common-neighbors(CN) in green, the Jaccard(Jac) in light blue and the Salton(Sal) in dark blue. Observing **Table 4.3** and **Figure 4.6** numbers we can see that the extra-neighbors metric achieved the best overall precision. If we compare by the number of correct rules, this greater precision may not look like a big deal, but when the number of rules grows such difference can be more relevant.

4.2.3 Comparing GRL with similar state of the art Rule Learners

In this subsection, there's a comparison of GRL with the state of the art Rule Learner called AMIE[37] and also with the PRA[12] because it is a component of NELL. This three algorithms find inference rules from ontological knowledge bases.

In Table ?? we present the results of GRL, AMIE and PRA running in NELL's OKB iteration 885⁹. In the first column there are the algorithm's, in the second the total number of rules the algorithm finds. In the third column the precision of the rules, to measure that we randomly pick 100 rules from each algorithm and manually classified each one as correct(c) or wrong(w), the precision is the number of correct rules(|c|) divided by 100 (|c|/100). The fourth column is the potential to generate facts that the rules provided by each algorithm has, to measure that we apply the correct rules in NELL OKB and then divided the total resulting facts(*f*) for the number of correct rules (|f|/|c|). In the last column is the format of the rules each algorithm generates.

Regarding the total number of rules, AMIE found double of the amount GRL found and PRA found a largely great amount than the other two, but if we look at the precision, GRL maintain a its closely half to half like in the experiments above while both PRA and AMIE had poor precision. PRA is older, and currently AMIE is one of the state of the art rule learner, but they might have achieved a similar value at this experiment because PRA was designed specially to work with NELL.

⁹PRA found a larger amount of rules than the other two, because the version we used is acoplated with NELL and uses the whole OKB and not just the (much smaller) set of beliefs

When concerning the generated facts, GRL rules generate on average more facts than AMIE's.¹⁰

It is possible to notice that GRL has only one single format. We consider that as an advantage, mainly because it produces very readable rules that are also easy to apply on the OKB. AMIE has the same format used by GRL and, also, another one having only one relation at the body of the rule, this might be interesting too, with rules such as the one below:

AMIE.1. ismultipleof(X, Y) \Leftarrow *animalistypeofanimal*(X, Y)

AMIE.2. synonymfor(X, Y) \Leftarrow *synonymfor*(Y, X)

The first rule above uses the generalization concept (like the relation *ismultipleof* is a generalization of *animalistypeofanimal*), and the second presents the symmetric relation. Both factors can be very interesting to another projects, but NELLs ontology relation file already have specification for these characteristics, for each relation is indicated if it is reflexive, symmetric and its generalizations, so this rules doesn't have a greater use to it.

PRA itself, has the most non-standardized rule format, because its body might have a great number of relations, and it also has categorization relations (e.g. *athlete*(X)). This might be the reason why it found too many rules, but also if have a great number of incorrect rules, despite the fact that is very hard to classify them. We found a correct rule found by PRA and GRL that generates the same fact(has the same head), but with a different body:

PRA. awardtrophytournamentisthechampionshipgameofthenationalsport(X, Z) \Leftarrow
trophywonbyteam(X, Y) \wedge *athleleledsportsteam*(W, Y) \wedge *agentcontrols*(W, Y)
 \wedge *teamplaysport*(Y, Z)

GRL. awardtrophytournamentisthechampionshipgameofthenationalsport(X, Z) \Leftarrow
athletewinsawardtrophytournament(Y, X) \wedge *athleteplayssport*(Y, Z)

¹⁰PRA was not used in this experiment because it's rules generally have relations in the body that doesn't have instances in beliefs set

Looking at this two rules it is possible to see how GRL rule is more readable and why it might be better to NELL to generate new facts. It produces the same fact but needs a very simpler “condition”.

4.2.4 Scalability

Experiments evaluating scalability performance of the GRL implementation were not performed. Considering that most of GRL computational effort is related to the task of finding all closed triangles present in the graph, we can base our scalability analysis upon results presented in [44]. We can conclude that GRL follows GraphDB-Tree scalability performance, thus being a good option even for big graphs. The execution time of the GRL with NELL’s graph as input was about one minute.

4.3 Conclusion

GRL, differently from most of the other current approaches, explores ontological knowledge to get better results in inference rule extraction precision and scalability. Empirical results show that GRL can cope with NELL’s KB characteristics of being a big and never-ending growing KB, thus, not being noise free, neither being complete. Also, GRL can be considered generic enough to be used having any other ontological knowledge base as input.

Considering NELL’ KB is constantly evolving, as future work, we plan to investigate how frequently (every iteration, or every 5 iterations, etc.) should GRL be used to get most relevant results. Also, GRL can be coupled with other NELL’s components to help in self-reflection (a key property for a never-ending learner). Last, we will evaluate GRL with more ontological KBs besides NELL and YAGO.

Chapter 5

Finding new relations

Link-prediction techniques, can be referred as the task of finding edges that would appear in a near future of a graph. In the last years a large number of approaches has been proposed in link prediction tasks, an interesting review on those can be seen in [20].

There are also techniques that combine community detection with link prediction as the one presented in [45], where the authors use a naive community detection approach to help in link prediction. However, this approach only works for higher clustered networks. In spite of this, the proposed idea is very useful, since the probability of an edge existing between nodes in the same community is higher than between nodes from different communities.

Link prediction can make use not only of graph structural information but also relational characteristics, for example attributes related with graph's node as presented in [23]. This kind of approach is more used in relational or multi-relational learning [8, 24–26].

In the context of graphs mapping knowledge bases these techniques are mostly used to find new facts, but they can be used as well to find new edges on the ontological model graph that is equivalent to new relations to an OKB as well, using the model of N^o proposed.

Relations-Predictor Algorithm:

1. Find all open triangles $\Lambda(u, w)$;
2. Calculate a score (such as CN[9]) for each triangle $score(u, w)$;

3. Group them at open triangle category groups $\Lambda_c(c_u, c_w)$;
4. Calculate a score for each group

$$score_c(c_u, c_w) = \sum_{\Lambda(x,y) \in \Lambda_c(c_u, c_w)} score(x, y)$$

5. Look for groups with high scores to propose relation between pairs of categories (c_u, c_w) ;

Relations-Predictor Algorithm is a generic algorithm created just to exemplify the task of finding new relations using link-prediction techniques in an N_o . It follows the idea presented in figure 1.2, by grouping open triangles it found based on the nodes in the edge, to suggest relations among the pair of categories if the pattern occurs various times in the graph.

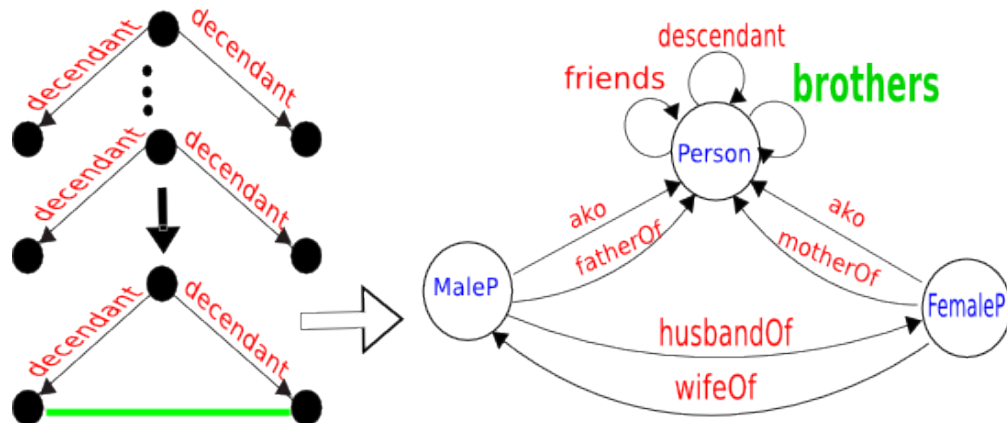


FIGURE 5.1: Example: Finding a new relation

In **Figure 5.1** we use our imaginary social network to illustrate how the Relations-Predictor algorithm can be used. In that case, if it finds too many open triangles with a specific pattern, where the two edges are the relation “descendant” and the the nodes of the category “Person” ($\Lambda_c((Person, descendant), (Person, descendant))$), it may find that there exists a relation, in this case “*brothers* $\langle Person, Person \rangle$ ”. We can observe that the relations found have generic names, another approach (or human supervision) is needed if we need to name the relation like in the example.

In the next sections of this chapter, we will present the Prophet algorithm that is somehow similar to the Relations-Predictor algorithm, OntExt, an algorithm that will be used to find names of relations proposed by Prophet and to finish PrOntExt, the project that

connects this two algorithms and is currently running iteratively with NELL expanding its ontology.

5.1 Prophet

The Prophet algorithm is similar to GRL, it finds all open triangles from the input OKB (mapped as a graph), then, it groups them into groups based on the category of the border nodes, and then, uses the link prediction-metric called Extra-Neighbors to promote new possible relations and instances for them.

5.1.1 Prophet Algorithm

Prophet uses the beliefs present in NELL's KB as an undirected and unweighted graph to apply a link prediction algorithm with three main tasks: finding new relations, finding new facts (instances of these new found relations) and finding wrong facts (also called misplaced edges).

To find new relations, Prophet first finds all the open triangles $\Lambda(u, v, w)$ in the graph, and then, groups them by the categories of the extremity nodes in $\Lambda_c(c_u, c_w)$. After that, it calculates the number of extra neighbors of each group (1), which is the sum of common neighbors, of each open triangle $\aleph(u, w)$, subtracted by the size of the group (sum for all $\Lambda(u, w)$ instances). The bigger this value is, it indicates that the extremity nodes of the open triangles in the group have more and more common neighbors. If every open triangle extremity node has degree 1, the value of the extra neighbor measure will be 0. If the extra neighbors value is greater than a given threshold (ξ), the relation among c_u and c_w categories (c_u, c_w) will be consider a new valid relation.

$$\aleph_c(c_u, c_w) = \sum_{\Lambda(u,w) \in \Lambda_c(c_u, c_w)} \aleph(u, w) - |\Lambda_c(c_u, c_w)|$$

Prophet also finds new facts, that indeed are instances of the new relations, found on the respective open triangle category group. There are two ways to a pair $\Lambda(u, w)$ be considered a valid fact:

First: If the number of distinct categories of the middle nodes $\delta(\Lambda(u, w))$ is equal to the total number of distinct middle node categories in the group $\delta(\Lambda_c(c_u, c_w))$;

Second: If the number of neighbors between u and w $\aleph(u, w)$ are greater than, or equal than the threshold (ξ).

The two conditions for an instance to be valid are shown in equation (2), that is the set of all instances considered valid of a group $\Lambda_c(c_u, c_w)$.

$$I_{\Lambda_c(c_u, c_w)} = \{\langle u, w \rangle | (\delta(\Lambda(u, w)) == \delta(\Lambda_c(c_u, c_w))) \vee (\aleph(u, w) \geq \xi)\}$$

Prophet Algorithm is presented in Algorithm 2:

Algorithm 2 Prophet Algorithm

Require: $G(V, E, X)$

Ensure: List of pairs of categories (c_u, c_w)

```

1: Find all  $\Lambda(u, w)$  in  $G$ 
2: for all open triangle  $\Lambda(u, w)$  do
3:   Calculate  $N(u, w) = |\Gamma(u) \cap \Gamma(w)|$ 
4:   Group  $\Lambda(u, w)$  in  $\Lambda_c(c_u, c_w)$ 
5: end for
6: for all  $\Lambda_c(c_u, c_w)$  do
7:   Calculate  $N_c(c_u, c_w)$ 
8:   if  $N_c(c_u, c_w) \geq \xi$  then
9:     Propose the pair:  $(c_u, c_w)$ 
10:    for all  $\Lambda(u, w) \in \Lambda_c(c_u, c_w)$  do
11:      Calculate  $\Sigma(\Lambda(u, w))$ 
12:      if  $\Sigma(\Lambda(u, w)) == \Sigma(\Lambda_c(c_u, c_w))$  then
13:        Validate Edge  $\langle u, w \rangle$ 
14:      else
15:        if  $N(u, w) \geq \xi$  then
16:          Validate Edge  $\langle u, w \rangle$ 
17:        end if
18:      end if
19:    end for
20:  end if
21: end for

```

5.1.2 Prophet Example

In Figure 5.2, an example of the Prophet is presented. We have the open category group $\Lambda_c(\textit{Athlete}, \textit{SportsLeague})$, and the three instances $\Lambda(\textit{Neymar Jr.}, \textit{Champions League})$, $\Lambda(\textit{JorgeValdivia}, \textit{Brazilian Cup})$ and $\Lambda(\textit{Lebron James}, \textit{ChampionsLeague})$.

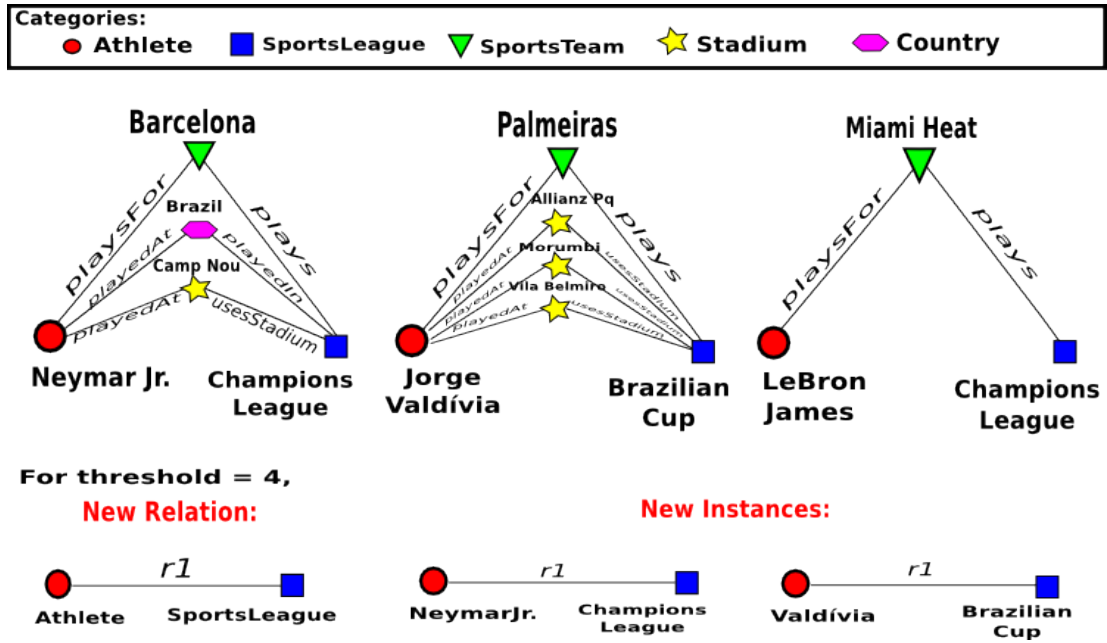


FIGURE 5.2: Example of Prophet

To validate the relation between *Athlete* and *SportsLeague*, the number of extra neighbors have to be greater than, or equal than the given threshold that is equal to four. In this example the relation will be created.

$$\begin{aligned}
 \aleph_c(\textit{Athlete}, \textit{SportsLeague}) &= \aleph(\textit{Neymar Jr.}, \textit{ChampionsLeague}) \\
 &+ \aleph(\textit{JorgeValdivia}, \textit{BrazilianCup}) \\
 &+ \aleph(\textit{LebronJames}, \textit{ChampionsLeague}) \\
 &- |\Lambda_c(\textit{Athlete}, \textit{SportsLeague})|
 \end{aligned}$$

$$\aleph_c(\textit{Athlete}, \textit{SportsLeague}) = 3 + 4 + 1 - 3 = 5 \geq \xi(\xi = 4)$$

Since the extra neighbor value is greater than the threshold ($5 \geq 4$), the relation will be validated. After that, each instance should be tested, and the result is:

$$\delta(\Lambda(\textit{Neymar Jr.}, \textit{ChampionsLeague})) = \delta(\Lambda_c(\textit{Athlete}, \textit{SportsLeague})) = 3$$

$$\aleph(\textit{JorgeValdivia}, \textit{BrazilianCup}) = 4 \geq \xi(\xi = 4)$$

$$I_{\Lambda_c(c_u, c_w)} = \{\langle Neymar Jr., ChampionsLeague \rangle, \langle JorgeValdivia, BrazilianCup \rangle\}$$

The relation created on the example is named `r1`, but this is just a label given by Prophet, because it cannot identify the *semantic meaning* of the relations that it finds. Let's suppose we use another program to find the name for that relation, and it found that `athletePlaysLeague` would be a emphsemantic meaningful name, then we can add the relation to extend our sports ontological network, and name it (see at Figure 5.3).

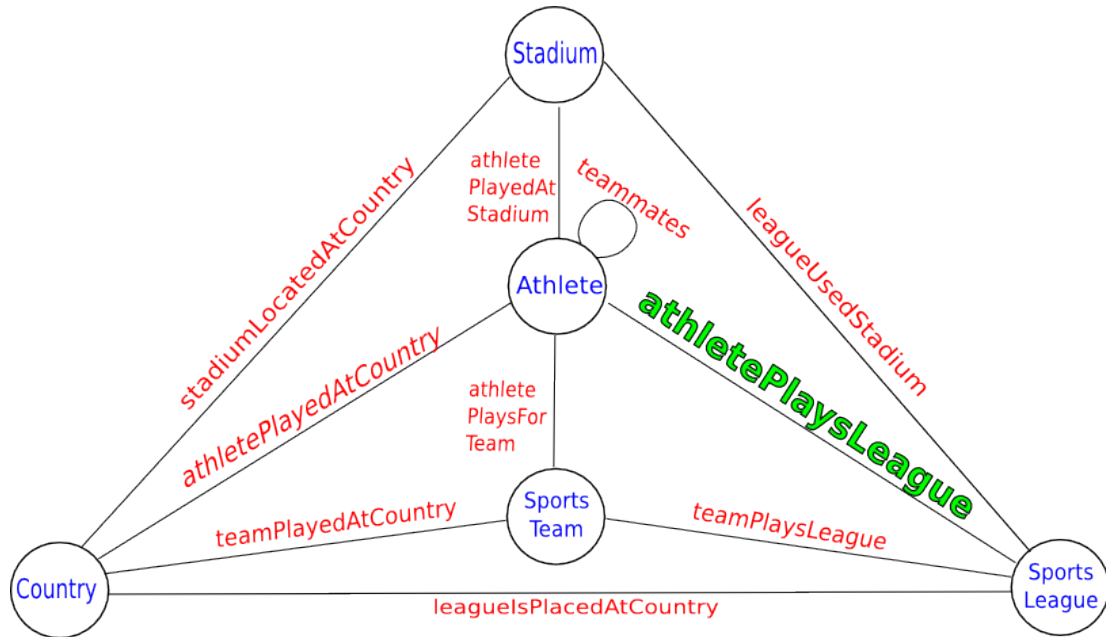


FIGURE 5.3: Example of Prophet

5.1.3 New Prophet Implementation

There were two problems over the first Prophet implementation: the first one is that it was implemented to query directly NELL's KB stored on a database with pure SQL, so it was not very suited to the use outside NELL and the second is that its performance (execution time) doesn't scale very well as the graphs vertex and edge number grows. These problems were the main motivation to the creation of a New Prophet implementation and, in addition, to propose the graph structure GraphDB-Tree [44] used in the Graph Rule Learner, that is also widely used by New Prophet¹.

¹More information about GraphDB-Tree is present at Chapter 4

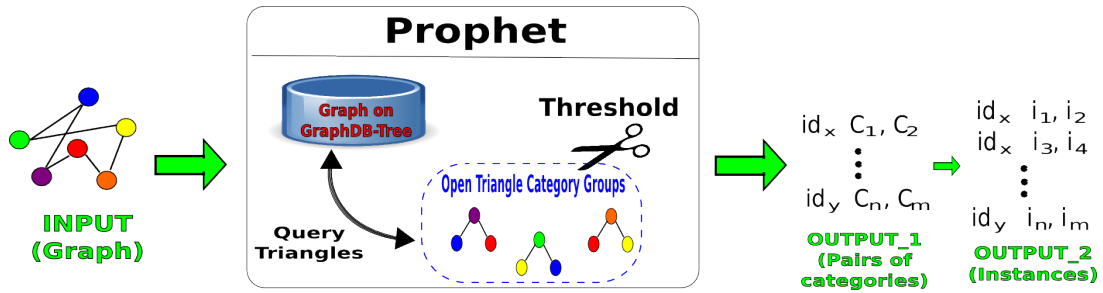


FIGURE 5.4: Prophet Execution Diagram

The New Prophet execution flow can be seen in Figure 5.4. The input is a graph file in edge list format, then this file will be pre-processed if necessary, stored in GraphDB-Tree, then later Prophet queries the GraphDB-tree graph to assemble the open triangles and mount the open triangles category groups. After all the open triangles are found, the Prophet passes through each group trying to validate the relation and the instances of it. The output consists in a list of new relations(as pairs of categories) and a list of instances(as pairs of facts) for these new found relations.

5.2 OntExt

As it was mentioned in the last section, Prophet can not name the relations it found, it just suggests the category pairs that might be related, so we can use OntExt to do the naming task.

OntExt receives as input two lists of instances representing instances of two categories(ex: sportsLeague and athlete) and a (generally huge) data set with triples in the subject-predicate-object (spo) format ², and an object. Lets call the “predicate” in the spo triple a feature between the subject and the object, OntExt will mount a matrix to record the number of co-occurrence between the features which links two categories. Each cell of the matrix corresponds to the number of instance pairs of categories in which both contexts co-occur (Matrix(i,j) value to contexts i and j - e.g. the sentences ”Neymar Jr. plays at Champions League” and ”Neymar Jr. dispute the Champions League” provide a case where the 2 contexts ’plays at’ and ’dispute’ co-occur with an instance pair [Neymar Jr., Champions League]). After that, the matrix is normalized: first each cell value is divided by the total count of its line(row), and after that each

²An spo format file have triples with a subject, a predicate describing the relation (ex: ;Jorge W. Bush, **was the president of**, USA;)

Algorithm 3 OntExt Algorithm**Require:** Two list of instances L_1 and L_2 and a set of svo triples $\langle s, f, o \rangle$ S **Ensure:** A list of features and its clusters

```

1:  $fn$  = Number of different features  $f$  among the triples of  $S$ 
2: Allocate matrix  $m[fn][fn]$ 
3: for all triples  $\langle s_1, f_1, o_1 \rangle$  in  $S$  do
4:   if  $(s_1 \in L_1) \wedge (o_1 \in L_2) \vee ((s_1 \in L_2) \wedge (o_1 \in L_1))$  then
5:     for all triples  $\langle s_2, f_2, o_2 \rangle$  in  $S$  do
6:       if  $((s_2 == s_1) \wedge (o_2 == o_1)) \vee ((s_2 == o_1) \wedge (o_2 == s_1))$  then
7:          $m[f_1][f_2] ++$ 
8:          $m[f_2][f_1] ++$ 
9:       end if
10:    end for
11:  end if
12: end for
13:  $zn$  = number of empty lines in  $m$ 
14:  $fn2 = fn - zn$ 
15: Delete the  $zn$  empty lines and  $zn$  empty columns from  $m$ 
16: for  $i = 0$  to  $fn2$  do
17:   for  $j = 0$  to  $fn2$  do
18:
19:   end for
20: end for
21: for  $i = 0$  to  $fn2$  do
22:    $cont = 0$ 
23:   for  $j = 0$  to  $fn2$  do
24:     if  $m[i][j] > 0$  then
25:        $cont ++$ 
26:     end if
27:   end for
28:   for  $j = 0$  to  $fn2$  do
29:
30:   end for
31: end for
32: Allocate vector  $clusters[fn2]$ 
33:  $clusters = clusteringFunction(m, fn2)$ 
34: RETURN clusters

```

$$m[i][j] = \frac{m[i][j]}{\sum_{j=0}^{fn2} m[i][j]}$$

$$m[i][j] = m[i][j] * \frac{fn2}{cont}$$

cell will be multiplied by the fraction of the total number of features over the number of elements greater than 0 in the same line(row). Doing this normalization, higher weight is given to contexts which co-occur with only a few contexts (predicates), to promote less generic contexts. OntExt algorithm is presented in 3.

5.2.1 OntExt Example

Let's consider the following input:

- $S = \{ \langle \text{Neymar Jr., plays at, Champions League} \rangle, \langle \text{Jorge Valdivia, plays at, Brazilliam Cup} \rangle, \langle \text{Nathaniel Clyne, doesn't play, Champions League} \rangle, \langle \text{Neymar Jr., participates in the, Champions League} \rangle, \langle \text{Nathaniel Clyne, wants to play at, Champions League} \rangle, \langle \text{Neymar Jr., commented about, Champions League} \rangle, \langle \text{Nathaniel Clyne, commented about, Champions League} \rangle \};$
- $L_1 = \{ \text{Neymar Jr., Jorge Valdivia, Nathaniel Clyne} \}$
- $L_2 = \{ \text{Champions League, Brazilliam Cup} \}$

Instances at L_1 represent instances from category Athlete, and the ones at L_2 come from SportsLeague category. $fn = 5$ because we have four different features among all the triples in S . Given this lists, OntExt will then mount the co-occurrence features matrix (see at 5.5.

m	plays at	participate in the	doesn't play at	wants to play at	commented about
plays at	2	1	0	0	1
participate in the	1	1	0	0	2
doesn't play at	0	0	1	1	1
wants to play at	0	0	1	1	1
commented about	1	2	1	1	2

FIGURE 5.5: Co-Occurrence matrix m

After this point, the algorithm presented in 3 will remove empty lines and empty columns from the matrix. In this example, there are not any because all the triples in S have subject and object present in L_1 and L_2 (so $fn_2 = fn = 5$). OntExt will normalize the matrix dividing each cell by the total count of its line, the resulting matrix is present at figure 5.6.

Then each cell(i, j) will be multiplied for the fraction between the total number of features ($fn = 5$) over the number of elements greater than 0 at the current line(i). The resulting matrix is present at figure 5.7.

m	plays at	participate in the	doesn't play at	wants to play at	commented about
plays at	0.5	0.25	0	0	0.25
participate in the	0.25	0.25	0	0	0.5
doesn't play at	0	0	0.33	0.33	0.33
wants to play at	0	0	0.33	0.33	0.33
commented about	0.14	0.29	0.14	0.14	0.29

FIGURE 5.6: Normalized m

m	plays at	participate in the	doesn't play at	wants to play at	commented about
plays at	0.83	0.42	0	0	0.42
participate in the	0.42	0.42	0	0	0.83
doesn't play at	0	0	0.55	0.55	0.55
wants to play at	0	0	0.55	0.55	0.55
commented about	0.14	0.29	0.14	0.14	0.29

FIGURE 5.7: Final Normalized m

Having the matrix 5.7, OntExt will apply a clustering algorithm³. After that, the final results, the clusters vector, will probably be something like: $clusters=\{(plays\ at,\ 1), (participates\ in\ the,\ 1), (doesn't\ play\ at,\ 2), (want\ to\ play\ at,\ 2), (commented\ about,\ 3)\}$

It's possible to observe that we have 3 clusters, suggesting that there might exist 3 relations between categories Athlete and Sports League. At cluster 3, it suggests just one name (commented about), in the other two, there's more than one possible name for the relation, another program or an human being must look at it and choose the best fitting name.

³Currently using n-k-means++ presented in the next chapter

5.3 PrOntExt

Prophet and OntExt complete each other, because OntExt itself generally is not viable to run for all the possible category pairs of the input OKB (as it takes too much time to run), so Prophet can help it to focus on specific pairs of categories, increasing its precision and greatly reducing the execution time. On the other hand, Prophet alone just suggest category pairs, but is not able to name them, and the most important thing: it cannot determine when there exist more than one possible relation between a pair. As we could just see in OntExt example above, OntExt can be used to try solve this problem.

Since the old Prophet and the old OntExt were not very portable, we implemented new versions of both, already looking towards their integration. The OntExt algorithm is changed a bit in the clustering part, wherever the original used a simple k-means we apply the K-means++^[46] algorithm n times (n can be determined by user, we are currently using $n = 20$ for NELL). Then, we use an algorithm we created to gather these results and rank the resulting clusters⁴. This process was created to improve the final clustering results and the ranks help the new OntExt to determine when there might be more than one possible relation between a pair of categories.

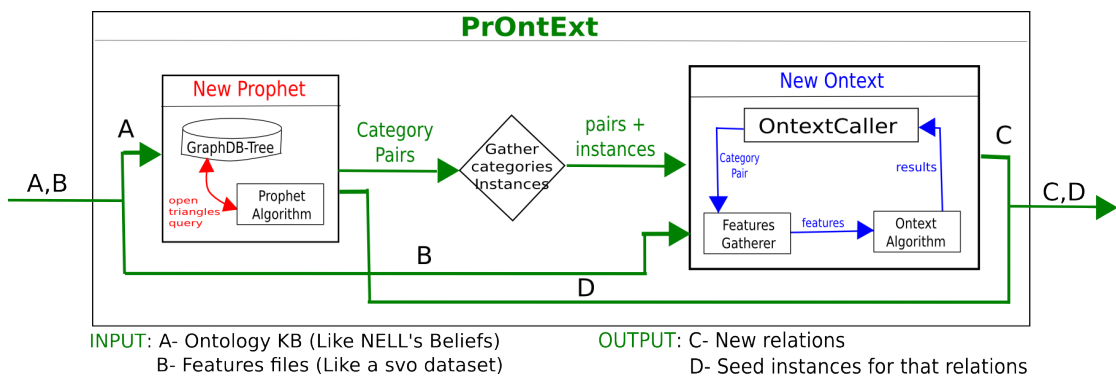


FIGURE 5.8: The PrOntExt

Figure 5.8 shows an overview of the PrOntExt algorithm. PrOntExt will receive an OKB as Input(A) (like NELL's OKB), apply the Prophet algorithm in this OKB, having a list of pairs of categories (indicating new possible relations). After that, PrOntext will gather all the instances from all the categories (among the pairs) from the input OKB and pass them to the New OntExt. New OntExt will then use the instances to gather

⁴this whole algorithm is called n-k-means++, and its presented in the next chapter

features from an input(**B**) features file (like a spo data set) to mount its co-occurrence matrix. After the matrix is mounted, it will cluster the matrix n times and gather the results to define the final set of clusters. The algorithm to gather the clusters will calculate a rank for each one of them, OntExt will always suggest a relation from the best ranked cluster, and for the other clusters, if their rank is at least 75% of the best one, a relation will be suggest for it too, in the end PrOntExt will output a list of relations(**C**). PrOntExt also outputs a list of proposed seed instances(**D**) (for the new relations) that comes from prophet ($I_{\Lambda_c(c_u, c_w)}$).

In other words, in the output, there will be for each relation (represented by a category pair, given by from Prophet), a list of proposed names and a list of proposed instances for the relation. For now, we are using human supervision to decide on the best name and to adjust it to NELL's standards, and also for choosing which seed instances to use or not (if there are just a few valid ones, we manually add some more).

5.4 Results

The list of possible relations that prophet generates (it's output list of relations) is not a hundred percent correct. In fact, good part of it usually consists in not very trustful relations. Inserting wrong knowledge in the KB is not a wise decision. If we insert wrong knowledge on NELL's KB, this knowledge might be propagated and generate dangerous semantic drifting[15].

To mitigate this factor, we are currently pick around 5-15 relations from each iteration of PrOntExt and manually insert them into NELL. This is clearly not the best way to proceed as NELL is supposed to be an autonomous system, but because of NELL's need of an ontology extension component to evolve, we are currently using this process.

NELL had 328 relation before this Masters project started adding relations to its ontology. Currently, ProntExt has already added 70 relations, so it already increased NELL's relations set in more than 20%.

In Table 5.1 it's possible to see how PrOntExt adds a relation to NELL. It starts with Prophet that outputs a pair of categories(first and second column). Then, OntExt will

TABLE 5.1: Some Relations Found by PrOntExt

Prophet		<i>OntExt</i>	<i>HumanSupervision</i>
Category 1	Category 2	<i>Name</i>	<i>FinalName</i>
insect	grain	<i>eats</i>	<i>insectEatsGrain</i>
company	videogamesystem	<i>obtain</i>	<i>companyProducesVideoGameS.</i>
agriculturalprod.	invertebrate	<i>benefit</i>	<i>ag.ProductIsEatenByInvertebrate</i>
island	skiarea	<i>have</i>	<i>islandHasSkiArea</i>
company	stadium	<i>disclose</i>	<i>companyDisclosesStadium</i>
building	park	<i>isfind</i>	<i>buildingWasBuiltAtPark</i>

run and propose a list of names in which we pick the best (third column). And later, we have to manually adjust the name proposed by OntExt (last column).

5.5 Conclusion

We didn't present any scalability experiments in this section because in [44] has already shown that local operation over graphs stored in GraphDB-Tree scales linearly. Experiments over Prophet and OntExt will not be present either because it can be found in their respective papers: [16] and [15].

Despite of that we have to shown some kind of precision measure of PrOntExt and made experiments comparing it to another related work. Since we are going to add a classification task in the future 5.7 to make PrOntExt more autonomous, we thought it will be fair to do all of these experimentation after that.

Finding new categories

Another disseminated topic of graph-mining is the community detection task[47], briefly explaining, this task uses the topology information of the graph to divide it in clusters of nodes (called communities).

To find new categories in an N_o , we can use community detection algorithms in a similar way of the link-prediction. We apply the algorithm in the graph, then we analyze each community that was found (such as the triangle category groups), and, if the community attends to a given condition, we can create a new category from it. Another option would be, if the communities shares common features (same set of relations, for example), then each community could be an instance of a new possible category.

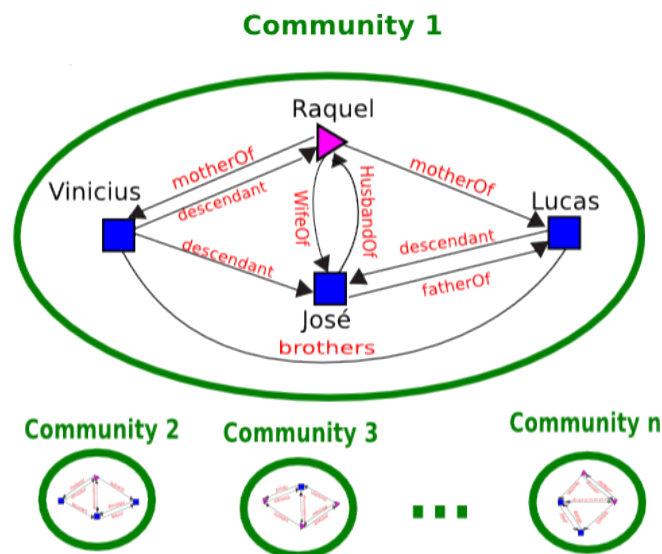


FIGURE 5.9: Example: Finding a new category

In **Figure 5.9** we exemplify how can we find a new category using community detection in our imaginary social network. Imagine that we apply a community detection algorithm that returns communities where nodes (inside the a community) are mostly related by the

following relations: *descendant*, *fatherOf*, *motherOf*, *brothers*. Observing this pattern, we can infer that these communities are families, so the category “family” can be added to our N_o (Figure 5.10).

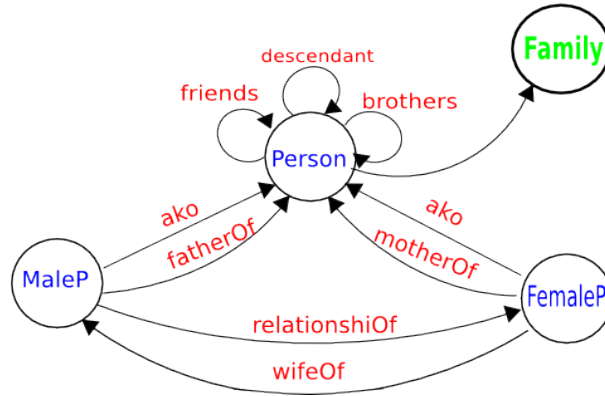


FIGURE 5.10: Incrementing the social network ontology with a new category

Same as in Prophet, this algorithm would not be capable of defining the name of the community or initial relations with other categories. Thus, some human supervision (or another approach) would be used to do that.

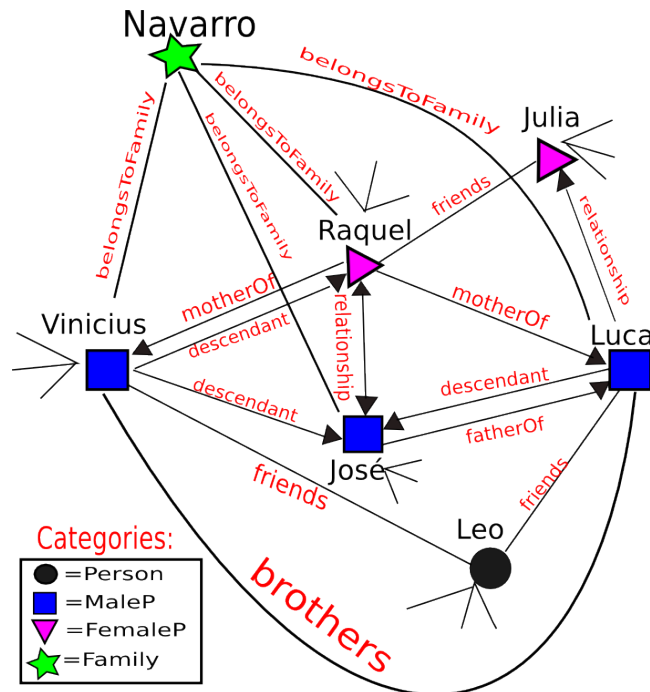


FIGURE 5.11: Social network instances graph (G_i) after we add a new category and relations

In Figure 5.11 we present an example of a fact of the new category (family:Navarro) added to the instances graph of N_{social}^o , and also a fact of the relation brother, discovered

in the last Chapter⁵.

Using the topology of the communities within the ontology information, there is a lot of ways to detect patterns to find new categories. We can focus, for example, at the instances of a specific category to try to find sub-categories. Let's look at the relation teammate between two athletes in N_{sports}^o (see at Figure 5.12).



FIGURE 5.12: Relation teammate in N_{sports}^o

We can look for a community structure composed only by instances of the relation teammate between athlete category instances, like at Figure 5.13 example.

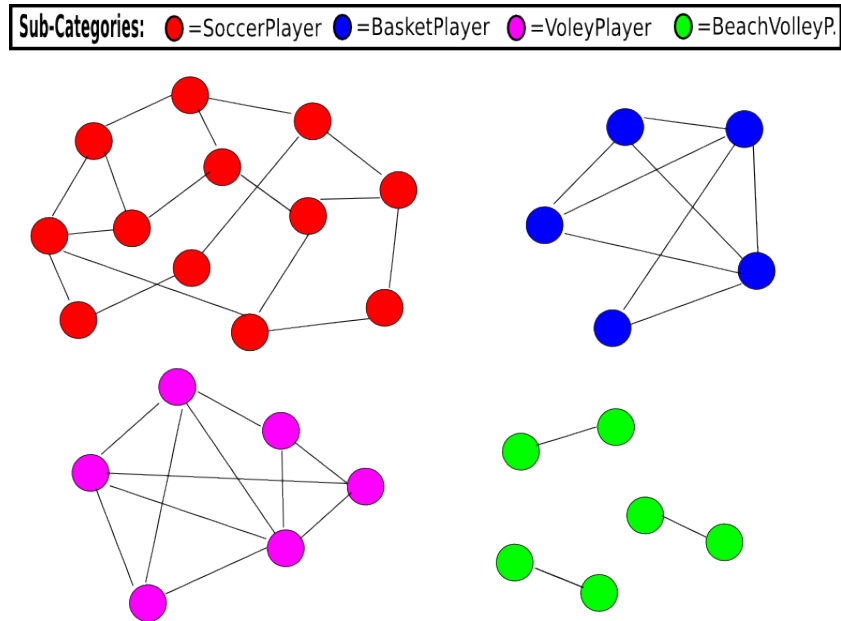


FIGURE 5.13: Finding sub-categories for Athlete category

It is possible to see that, based on the topology, we might differ soccer players, from basketball players, from volleyball players, etc., because each of this sports have different number of players in a team. If we have sports with the same number of players in a team or a close number such as Volleyball and Basketball, we can also use another information like the team of the players or the sports league that they play.

⁵we manually added this facts to exemplify how the instances graph could become more complete after the discovery of new relations and categories

Despite we had all of these ideas using community detection, we cannot actually apply them in NELL as it is right now. The main reason for this constraint is because NELL's OKB is currently too sparse (lacking lots of facts that it has not learned yet). Because of the fact that NELL is lacking in ontology extension components, we thought that it would be better to create a component that might find new categories faster so we use another approach that uses external data sets within NELL's. The component we create using this approach is called the SubCategory Finder and it is presented in the next sections of this chapter.

5.6 The Sub-Categories Finder

With the growth of the web-semantic research community, some projects developed techniques to extract ontologies (also called taxonomic structures) from text (a good revision of them can be seen at [48]). In [49], the OntoLearner is presented, an algorithm that extracts terms, definitions and hypernyms from text, create a hypernym graph and induces the ontology from this graph via optimal branching and a novel weighting policy. A very interesting fact is that OntoLearner could obtain high-quality results not just when used to build brand-new ontologies, but also to reconstruct sub-hierarchies in already existing ontologies. In [50], a system to automatically learn concept hierarchies was proposed (which is the same of refining categories into sub-categories), it is based on Formal Concept Analysis (FCA; a method used to investigate and process explicitly given information). It parses a corpus to extract verb/prepositional phrase (PP)-complement, verb/object and verb/subject dependencies, then uses FCA to produce a lattice that is converted into a special kind of partial order constituting a concept hierarchy. Our work is focused in implicit information (present in the KB) instead of explicit information like [50].

In this section the algorithm we created to find sub-categories, will be present. We call this algorithm the Sub-Category Finder, it can be divided in two main tasks:

- 1- Cluster the instances of a given category
- 2- Give names to these clusters (new sub-categories)

As mentioned before, one of the most common ways to detect category structures is using clustering techniques, some works used similarity vector for instances to cluster them into categories (also called concepts) [51] [52] [53] [54]. The clustering task is also very important to this work, since it is responsible for grouping instances of a category to find possible sub-categories (a review on clustering algorithms can be seen in [28]). This algorithms are used to solve unsupervised learning problems[29], so, as every other problem of this kind, it deals with finding a structure in a collection of unclassified data. A loose definition of clustering algorithm could be “the process of organizing objects into groups whose members are similar in some way”, so, a cluster is a collection of objects which are “similar” between them and are “dissimilar” to the objects belonging to other clusters. The k-means[18] is one of the most famous clustering algorithms, it works iteratively creating centroids (data points in which each feature is the mean of the group’s data points feature), then reallocating the data points to the closest centroid, until the obtained error (of the objective function) is lower than a given threshold. Generally, k-means is used to cluster data points with numerical features (using distance functions such as Euclidean distance or Manhattan’s distance[30]), but it can be adapted to work with discrete data as well [55, 56]). The sub-category finder, uses a version of k-means.

A concept that is important in sub-category finder is hyponymy. It shows the relationship between the more general terms (hypernyms) and the more specific instances of it (hyponyms). The paper [57] presents the traditionall approach of using specific extraction patterns to gather instances for hyponym relations. Below we present some extraction patterns proposed by Hearst (also known as Hearst patters):

1. NP_0 such as $\{NP_1, NP_2, \dots, (and|or)\}NP_n$
Ex: ... mammals such as cats, dogs and wales ...
2. $NP_1\{, NP_2, NP_3, \dots\}$ and other NP_0
Ex: ... mosquitoes, spiders, beetles and other insects ...
3. $NP_0\{, \}$ including $\{NP_1, NP_2, \dots, (and|or)\}NP_n$
Ex: ...basketball players including Michael Jordan, Kobe Bryant and Shaquille O’Neal ...

In the patterns above, NP_0 is the hypernym noun-phrase and the noun-phrases NP_1, NP_2, \dots, NP_n are the hyponym. After each pattern, there is an example to better illustrate how this patterns works, and it also helps us to understand how hyponymy is useful to automatically name new categories (in our case sub-categories).

The work of this paper is mostly based in the work presented in [17]. It was created to find sub-categories (specifically for NELL) such as this work. It uses **Non-negative matrix factorization(NMF)** to cluster the instances of the input category, finding the sub-categories. To do that, it creates a co-occurrence matrix of features of the instances where the features are the relations present in NELL's ontology. The method presented in that work was called **coupled non-negative matrix factorization(CNMF)**, that is a NMF method that clusters a matrix with the features coupled with side information about the instances coupled. The side information used was hyponym relations, for example: `animalSuchAsAnimal`, `animalIsTypeOfAnimal`, etc.

5.6.1 The SubCat-Finder Algorithm

The input for SubCat-Finder Algorithm is a list of instances I of a given category and features F for these instances (could be the category instances relations in NELL KB or/and features extracted from external data sets). The Sub-Category Finder will create a matrix in which each cell $n_{i,j}$ will be the number of times instance I_i is related to feature F_j , and after that, it will cluster this matrix in function of the instances I . The clustering results will be the new sub-categories, after that an algorithm will be applied to find the names of each sub-category (to do this, we were using generalization relations of NELL's KB, such as `istypeOf` and `suchAs`).

Figure 5.14 shows the Sub-Category Finder execution in four main tasks. First, (1) is the acquisition of the given category(c) instances. Next, (2) the features for this instances are gathered and the instances/features matrix is created. In part 3, the clustering task is applied in the matrix, having the (k) clusters as output (see the algorithm 4), then in part 4, the algorithm to find names for the clusters runs (see the algorithm 5) and the final results will be the new k sub-categories of the input category(c), and the instances set clustered into this new sub-categories.

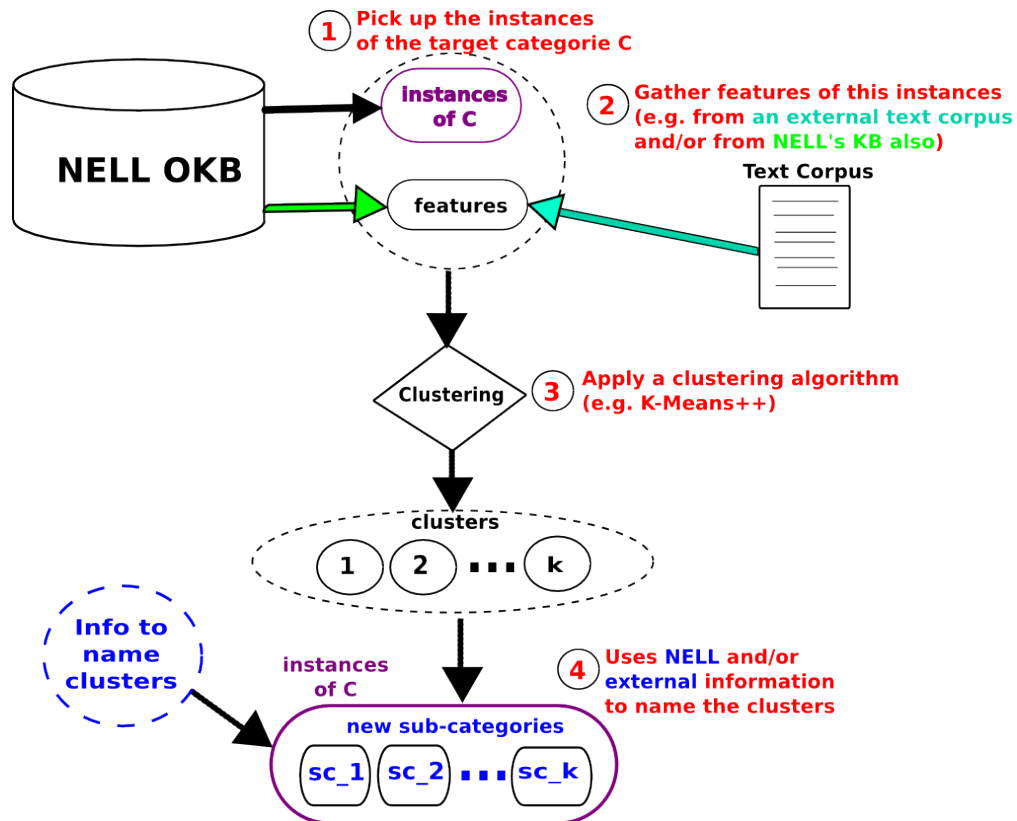


FIGURE 5.14: Sub-Category Finder

The first(1) and second(2) part are just algorithms to filter the input (NELL or external data sets) and to extract the instances/features of the wanted category. The most important part is the third(3), where the sub-categories will be found by the clustering task, and the last(4), where the clusters will be named. We discuss this tasks in the following subsections.

5.6.1.1 N-K-Means++ Algorithm

K-means++ algorithm is the clustering algorithm used in this project. It is a version of K-means, where the initial cluster seeds (centroids) are not chosen totally at random, but with a probability distribution weighted by the distance of the points to already chosen seeds, preferring points more distant (distinct) from already chosen points. In[46], the advantages of this approach are discussed.

K-means++ algorithm (as well as K-Means) does not guarantee that the (global) optimal solution will be found, it just guarantees the local optimum, so we decided to create an algorithm that will merge the results of various executions of the k-means++ algorithm,

called **n-k-means++**, where the k is the number of clusters wanted like the original k -means algorithm, and the n , is the number of times that the k -means++ will be executed.

Algorithm 4 N-K-Means++

Require: Data matrix $data$, number of instances d , number of features f , number of clusters wanted k , number of times of clustering task n

Ensure: Instances clustered in k clusters c_1, c_2, \dots, c_k

```

1: int  $cluster[n][d]$ 
2: int  $coo[d][d]$ 
3: for  $i = 0$  to  $n$  do
4:    $cluster[i] = k\text{-means++}(data, d, f, k)$ 
5: end for
6: for all pairs of instances  $i, j$  do
7:    $coo[i][j] =$  number of times instance  $i$  and  $j$  appears in the same cluster in the  $n$  results in  $cluster[]$  (from 0 to  $n$  times)
8: end for
9: Defines a set  $S$  of  $k$  seed instances for the  $k$  clusters, where:

```

$$S = \{s_1, s_2, \dots, s_k \mid \min(\sum_{i=0}^k \sum_{j=i}^k coo[s_i][s_j])\}$$

```

10: for  $i = 0$  to  $d$  do
11:   Add instance  $i$  to the cluster  $c_x$  where:

```

$$\max(\sum_{j=0}^{|c_x|} \frac{coo[i][c_x[j]]}{|c_x|})$$

```

12: end for

```

The N-K-Means++ algorithm is presented in 4. First it creates a matrix to store the results of each k -means++ execution ($cluster[n][d]$) and a matrix ($coo[d][d]$) to store the co-occurrence between the instances in each K -means++ output. It then executes k -means++ algorithm n times storing the results in $cluster$, following it will pass through each pair of instances (i, j) counting how many times they appear in the same cluster (in the n configurations), storing the value in $coo[i][j]$. After that, it chooses one seed instance for each of the k clusters (c_1, \dots, c_k) picking the combination which has the minimal co-occurrence (in the best scenario the k instances were never in the same cluster in the n configurations, so the sum in line 15 will be 0). After the seeds were chosen, it passes through each instance (d) and put it into the cluster c_x where the mean co-occurrence with c_x instances are greater (in best scenario, the sum in line 17 will be equal to n).

The N-K-Means++ algorithm have two very important advantages over the regular K -means++:

1. It reduces the chance of the algorithm being stucked into local optimal solution;
2. It allows us to use two different features matrix for the instances to be clustered by different k-means++ and then gathered in the end (we will do that in some experiments using features from NELL and from external data sets).

5.6.1.2 Naming the clusters

To be capable of automatically naming the clusters is a desirable characteristic to be used in NELL, and also in any autonomous learner system. To implement this task, we intend to use hyponym relations, some example of this relations in NELL: *athleteSuchAsAthlete*, *professionSuchAsProfession*. And some example of instances for these relations: $\langle \text{basketballplayers}, \text{KobeBryant} \rangle$, $\langle \text{footballplayers}, \text{BenTate} \rangle$, $\langle \text{healthprofessionals}, \text{doctors} \rangle$.

Algorithm 5 Sub-Category Nominator

Require: Instances clustered in k clusters c_1, c_2, \dots, c_k , generalization(hyponym) relation $hrel$

Ensure: Names for each of the k clusters n_1, n_2, \dots, n_k

```

1: for  $i = 0$  to  $k$  do
2:    $N =$  new associative array
3:   for all instances  $y \in c_i$  do
4:     for all  $hrel(x, y) \in hrel$  do
5:        $N(x) ++$ 
6:     end for
7:   end for
8:   name  $n_i = \bar{x} \mid \forall x \in N, N(\bar{x}) \geq N(x)$ 
9: end for

```

With enough instances of this kind of relations, its easy to name the clusters. Let's say almost every athlete instance in NELL's KB participates in a instance of relations *athleteSuchAsAthlete*, so we cluster the instances, and in the end we just need to look at each cluster and count the most frequent hypernym. The algorithm 5 is the implementation this task. It receives a set of clusters and the hyponym relation ($hrel$) used to the task, then for each cluster c_i , it counts the ocurrency of each hyperonym(x) present in $hrel$ involving instances of c_i (lines 4 and 5). After that, it names the cluster c_i (line 8) with the most frequent hyperonym(\bar{x}).

The algorithm itself is very simple, the idea is very neat, but the problem is that currently in NELL' KB there are just a few hyponym relations, and very few instances for them, so its not possible to apply this algorithm.

NELL populates the relations using different approaches, like seed examples given within the relations and also extraction patterns, example for relation `athletePlaysSportsTeam-Position`: "X is the team Y", "X is a Y" and "X plays Y". We can easily create a dozens of hyponyms relations manually and we believe that giving Hearst patters as the extraction patterns to them, NELL will be able to quickly populates this relations so that the naming algorithm can be used, that's our next steps to this project.

5.6.2 Experiments

In this section some experiments performed using the Sub-Category Finder algorithm presented in the last section is presented. To test this algorithm, we performed experiments in the following categories of NELL: `athlete`, `chemical`, `profession`, `vehicle`, `mammal` and `celebrity`. The reasons why we chose these categories, is because they all have a hyponym relation (ex: `athleteSuchAsAthlete`, `vehicleIsTypeOfvehicle`), so that the algorithm can name the clusters at the end, and they are the categories with the more instances of hyponym relations currently in NELL's KB.

The overall results of the experiments weren't very good, the only great results came from the `athlete` category, and the reasons for that is because `Athlete` is the category with the biggest number of direct relations (relations with `athlete` as parameter, such as `teammate`, `athleteplayssport`, `athleteplaysforteam`, `athleteCoach`, etc.), these relations have a great impact in the description of each `athlete` instance, helping a lot in the clustering process. The `athlete` category have around 15 direct relations, while most of the rest of categories in NELL today have less then 3 and no other category have even half of the quantity of `athlete` category. Because of this factor, we will show the results for the `athlete` category in the experiments below, since the results for the other 5 categories were very poor.

5.6.2.1 Clustering a Category to Find Sub-Categories

First we will present results involving the part 3 (see Figure 5.14), that is the clustering of the instances of the input category using the `n-k-means++` algorithm (see algorithm 4). In the process of gathering the results for the final clustering configurations at the `n-k-means++`, the instances are ranked by frequency with each other in the n configurations.

TABLE 5.2: 20x K-means(NELL)

Cluster	Instances	Frequency
c1	Tennis-Players	9/10
c2	Golf-Players	9/10
c3	Football-Players	10/10
c4	Basketball-Players	4/10
c5	Baseball-Players	10/10
Average Frequency		8.4

TABLE 5.3: 10x K-means(NELL) + 10x K-means(Svo)

Cluster	Instances	Frequency
c1	Baseball-Players	10/10
c2	Tennis-Players	10/10
c3	Basketball-Players	6/10
c4	Football-Players	4/10
c5	Baseball-Players	9/10
Average Frequency		7.8

So to evaluate precision of the final clustering results in the experiments of this section, we looked for the top ranked instances of each cluster.

To the **first experiment** NELLs KB relations involving the instances of the input category was used as features to compose the matrix to be clustered. The N-K-Means++ were executed with $n = 20$ and $k = 6$. Its result (for athlete category) is present in Table 5.2.

To the **second experiment**, NELL's KB was used (just as the first experiment) to a n-k-means++ execution running with $n = 10$ and $k = 6$, and also a N-K-means++ was applied into features of the instances of the given category gathered from a svo (subject verb object) data set with 114 million triples (with also $n = 10$ and $k = 6$). The final clusters where gathered (using the same algorithm 4) from the results of this 20 cluster tasks. The result is present in Table 5.3.

Despite the athlete category having 15 direct relations, it also has dozens of relations that are not direct, and in the first two experiments all of those had the same weight in the clustering task. To the **third experiment** NELL's KB was used as features again, with $n = 20$ and $k = 6$ just like the first experiment, but in this experiment the ontology information was used by changing the k-means++ distance function to give more weight to direct relations. The result is present in Table 5.4.

TABLE 5.4: 20x K-means(NELL weighted)

Cluster	Instances	Frequency
c1	Tennis-Players	10/10
c2	Basketball-Players	6/10
c3	Football-Players	10/10
c4	Golf-Players	10/10
c5	Baseball-Players	9/10
Average Frequency		9

By observing the results from the three experiments above, we can see that the svo we used did not bring much contributions to the obtained results. Actually, for the other 5 categories there was some improvement despite none good enough. But for Athlete category we believe that it make the results worse because this category is also well described in NELL (because of the amount of direct relations), and since we used just high confident instances of NELL (called beliefs), it tends to have less noisy instances than the svo data set.

The third experiment achieved the bigger clusters configuration, and this strengthens the importance of the direct relations for description of a category in the clustering task. We can conclude that to achieve a better clustering over a category's instances using NELL's KB, the category needs to have a good amount of direct relations.

5.6.2.2 Naming the Clusters

In the 4th part of the project (see Figure 5.14, the algorithm created to name the clusters using hyponym relation (algorithm 5), was implemented and empirically tested for the 6 categories used in the clustering experiments (from section above). Unfortunately the results were very poor. We chose the six categories with the biggest number of instances in the hyponym relations, but even for these ones, less than 10% of instances participates in hyponym relations. We intend to populate NELL's KB with more hyponym relations and instances and to try out this algorithm again.

5.7 Conclusion

To a self-learning system such as NELL, to be capable of automatically extending its ontology, such as finding sub-categories for a given category is very important. It can

play an important role in the capability of self-organizing instances into more specific categories by itself when a category is well mature.

This work is not yet concluded and only initial steps were taken. But, the obtained results so far helped us conclude that to achieve a better clustering in one of its category instances, the category needs to be better described having a good amount of direct relations. The athlete category, that is the most mature currently on NELL, make us believe that work is very promising.

We also have some problems into the naming of the new clusters because of the lack of hyponym relations (and its instances) in NELL's KB. But we studied methods to help populates this kind of relations, and its one thing that we should focus on the next steps.

Because of this conclusions, the next steps we have to do for the future of the Sub-Category Finder (in NELL) can be resumed in two two main goals:

- Populates NELL's KB with more relation to achieve better results in the clustering a category: to this task, we had a project that finds new relation to NELL that started to run in April/2015 and it should significantly increase NELL's relations in just a couple of months;
- Create and populate more hyponym relations on NELL's KB: as we commented in the end of section [5.6.1](#), we believe that using Hearst patterns NELL might be able to populate hyponym relations faster;

Future Works

5.8 Extending the Ontological Network Model N^o

The N^o model maps a very simple ontology consisting just of categories and relations, but it's possible to extend the model by adding more detailed components, such as events. Currently some events are being added to NELL's ontology, thus we are going to use one of them as an example to better illustrate how this works with our model.

Let's consider NELL's OKB mapped into the N^o model:

$$N_{NELL}^o = (G_m, G_i)$$

An event, as NELL is implementing it, will be a new category to identify the event (name) and a set of relations with the node of this category to describe the event.

To exemplify, one of the events recently added to NELL's OKB, the Bombing event⁶:

- Category of the event={BombingEvent};
- Relations that describe the event={witnessOfBombing(Person, BombingEvent), victimOfBombing(Person, BombingEvent), bombingCausedByExplosive(BombingEvent, Explosive), timeOfBombing(time, BombingEvent), locationOfBombing(Location, BombingEvent)}

We can observe that the event model is a sub-graph ($G_{Bombing}$) contained at G_m , and each instance of the event will be a sub-graph with the exactly same format of $G_{Bombing}$. The Sub-graph will be like:

⁶NELL's bombing event and its relations can be observed at <http://rtw.ml.cmu.edu/rtw/kbbrowser/>

$$G_m \subset G_{Bombing}$$

$$G_{Bombing} = (V_{Bombing}, E_{Bombing})$$

$$V_{Bombing} = \{BombingEvent, Person, Explosive, time, Location\}$$

$$E_{Bombing} = \{witnessOfBombing\langle Person, BombingEvent \rangle, victimOfBombing\langle Person, BombingEvent \rangle, bombingCausedByExplosive\langle BombingEvent, Explosive \rangle, timeOfBombing\langle time, BombingEvent \rangle, locationOfBombing\langle Location, BombingEvent \rangle\}$$

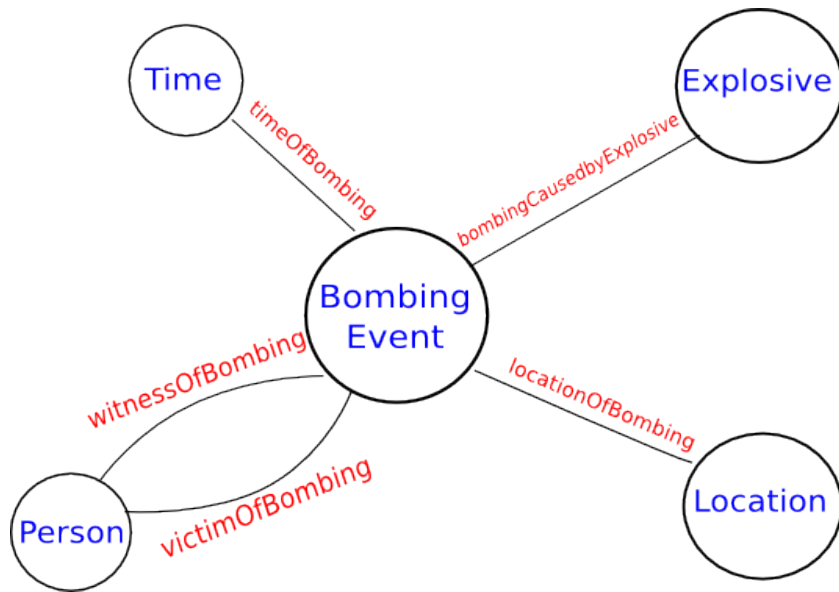


FIGURE 5.15: Sub-graph of the Ontological Model representing the Bombing Event

Figure 5.15 illustrates how the sub-graph $G_{Bombing}$ of the ontological model G_m^{NELL} will look like.

Figure 5.16 present the instance of a bombing event from the model of $G_{Bombing}$.

5.9 Future Classification Task

Every time PrOntExt (and also GRL) runs, it generates a file within the output relations (category pairs) with a set of statistics that we take from it. The list of this statistics is described below:

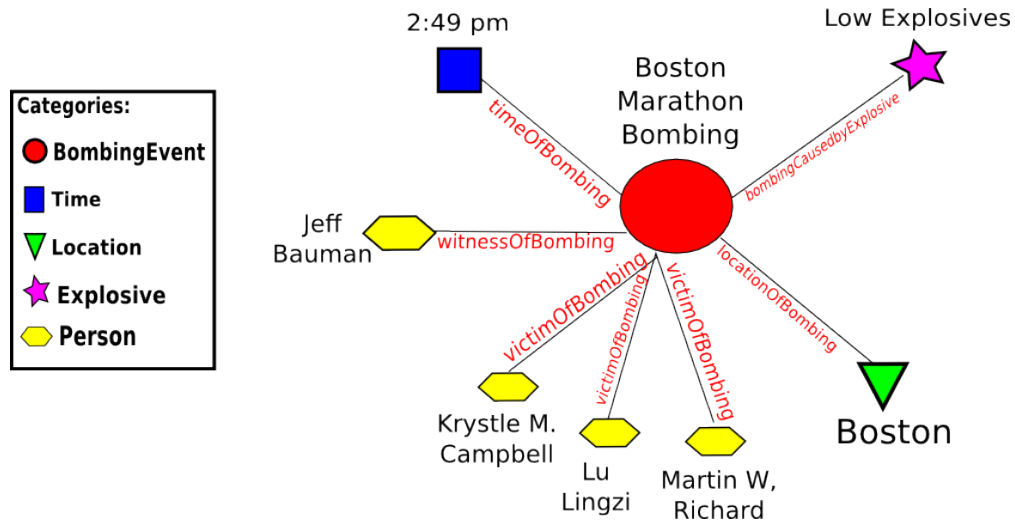


FIGURE 5.16: Sub-graph of the Ontological Model representing the Bombing Event

1- Jaccard Index.[58] The sum of the Jaccard index of each instance of an open triangle category group is calculated, then it is divided for the total number of instances in the group to normalize the value, see in equation below.

$$\sum_{\Lambda(u,w) \in \Lambda_c(c_u, c_w)} \frac{score_{Jac}(u; w)}{|\Lambda_c(c_u, c_w)|}$$

2- Adamic/Adar Index.[10] Analogously to the jaccard index, the sum of the adamic index in an open triangle category group is calculated and normalized, see in equation below.

$$\sum_{\Lambda(u,w) \in \Lambda_c(c_u, c_w)} \frac{score_{AA}(u; w)}{|\Lambda_c(c_u, c_w)|}$$

3- Extra Neighbors. This attribute is based on the previously presented extra neighbor index. The equation below is used as an attribute, with the extra neighbor value divided by the size of the open triangle category group.

$$\frac{\aleph_c(c_u, c_w)}{|\Lambda_c(c_u, c_w)|}$$

4- Promoted Instances. This attribute is based on the number of valid instances of the group, in fact, it's the percentage of instances of the group that will be promoted.

See in equation below.

$$\frac{|I_{\Lambda_c(c_u, c_w)}|}{|\Lambda_c(c_u, c_w)|}$$

5- Size of category group. The last attribute used is directly based on the size of the open triangle category group, which was used to normalize all the other attributes. But to his value doesn't be so much disperse and to maintain the same range as the other metrics, it's applied a MIN-MAX normalization. See in the equation below.

$$\frac{|\Lambda_c(c_u, c_w)| - |\Lambda_c(c_u, c_w)|_{MIN}}{|\Lambda_c(c_u, c_w)|_{MAX} - |\Lambda_c(c_u, c_w)|_{MIN}}$$

We save these statistics in every iteration of PrOntExt execution. Since we are currently using human supervision to promote the relations, in a couple of months we are going to have a lot of highly trust promoted relations and also negative examples, so we plan to gather this examples within their statistics (the attributes presented above) to train a classifier (using an algorithm we didn't define yet). If we succeed we will make PrOntExt task a step further to be an autonomous system⁷.

5.10 Sensibility of the algorithms, ignoring out of pattern facts

All of the algorithms presented in this Masters' thesis are sensitive to wrong facts previously stored in the OKB. A few wrong instances can generate lots of incorrect outputs, so we have always the need of human supervision or a classification task. We design this algorithms to work with NELL that is a never ending system that read from the web. It is expected that NELL promotes some wrong facts, other famous OKBs are not immune to this problem either, so we tough it may be good if our algorithms could "filter" the facts to reduce the chance of using the wrong ones.

For our algorithms that tends to use groups of specific categories and relations isolated, we thought that it will make sense to use a measure based on the degree distribution of a category like the mean, median or the mode to determine a range of degrees that

⁷we already made some experiments using weka with a set of manually classified relation with some classification algorithms and we achieve some reasonable precision (greater than 75%)

the instances of the category should have to increase the confidence that the instances is really correct. Let's stick to the example of the N_{sports}^o .

If we try to filter probably wrong athletes, we might use the mode between all athletes instances to establish a pattern of "probably valid" athletes. Considering the ontology presented in 3.3, and thinking of how things work in the real world. Probably an athlete will have around 1-20 teammates, played at around 50 stadiums and 10 countries, plays for 1 team, and plays at around 3 leagues, so the mode here will be some value around 85-105 (depending on the sports). We can use that, creating a soft interval of acceptable degrees of each instance node of athlete category (let's say from 40-200) to contemplate exceptions, doing that, athlete instances with more than 200 or less than 30 connections will be thrown away.

We are still experimenting what is the best measure and the best way to filter less probably instances, but PrOntExt already achieve very better results with a filter by the mode like in the example above, and it is actually already using it in its current implementation.

Final Conclusion

The main focus of our whole work is to use the ontology information of an ontological knowledge base to extract implicit knowledge in the form of new categories, with the Sub-Category Finder, and relations, with the PrOntExt, this way extending the ontology itself. We also find new facts with the Graph Rule Learner. The main motivation for this work is the lack of researches that uses the ontology directly to find implicit knowledge, plus nowadays there's a lot of projects that uses OKBs to store its knowledge.

Our project was largely motivated by NELL and the algorithms we based most of our own was NELL's components Prophet and OntExt, but instead of just reprogram them in another programming language, we enhance them to find new relations, and we use its core ideas to create algorithms to extract different kinds of knowledge(categories and facts). Also in the implementation, we seek to achieve good performance that will scale with NELL's never ending growing OKB.

Despite all of the algorithms presented in this work seems to be totally focused on NELL, they are all very generic. With simple pre-processment any ontological knowledge base with a simple ontological model like the one described in Chapter 1 should be able to be used as input to these programs.

A structure called ontological network(N^o) that can be used to map and ontological knowledge base(OKB) for the execution of graph-mining algorithms was formally presented (Chapter 3). It was used to help at the design of the algorithms, and also to describe the algorithms used to find implicit knowledge.

We presented a lot of results over GRL(Chapter 4), because it is the most "mature" project we currently have. It was possible to shown that is

PrOntExt is lacking of experimentation, and it is using human supervision, but despite of that it has shown to be very promising, even more if we consider the classification task and filters to reduce sensibility that might be implemented in the future (Chapter 5.7). PrOntExt already increases NELL's relation base in more than 20%.

Some ideas within examples of how to use community detection techniques to find new categories for an OKB were presented in Chapter 5.5. After that we present a project we work for to find sub-categories using clustering techniques, this work is not very mature yet but initial results has show that it is promising if we can increases NELL's relation base(what we are doing with PrOntExt).

After we present all the components, we propose some future work to enhance and improve them in Chapter 5.7. There aren't currently much works involving ontologies to discover implicit knowledge, it can be considered a very recent area, and with our whole work it's possible to imagine that there's a plenty room for it to evolve.

Projects Results

.1 Running GRL with NELL OKB it. 820

Table of **Figure 17** presents rules found by GRL running over NELL's KB. Table of **Figure 18** presents rules found by GRL running over YAGO(1)'s KB. All rules presented in this appendix are from the final list (rl_2), so, they were grouped (to generic rules) and ranked.

rank	NELL's extracted inference rules
29	$\text{teamplyssport}(X,Z) \Leftarrow \text{athleteplayssport}(Y,Z) \wedge \text{athleteplaysforteam}(Y,X)$
26	$\text{agriculturalproductincludingagriculturalproduct}(X,Z) \Leftarrow \text{agriculturalproductincludingagriculturalproduct}(X,Y) \wedge \text{agriculturalproductincludingagriculturalproduct}(Y,Z)$
25	$\text{athleteplayssport}(X,Z) \Leftarrow \text{teammate}(X,Y) \wedge \text{athleteplayssport}(Y,Z)$
19	$\text{animalistypeofanimal}(X,Z) \Leftarrow \text{animalistypeofanimal}(X,Y) \wedge \text{animalistypeofanimal}(Y,Z)$
18	$\text{organizationhireperson}(X,Z) \Leftarrow \text{subpartoforganization}(Y,X) \wedge \text{organizationhireperson}(Y,Z)$
16	$\text{teamplyssport}(X,Z) \Leftarrow \text{teamplyssport}(X,Y) \wedge \text{teamplyssport}(Y,Z)$
14	$\text{countryalsoknownas}(X,Z) \Leftarrow \text{countrylocatedingeopoliticallocation}(X,Y) \wedge \text{countrylocatedingeopoliticallocation}(Z,Y)$
10	$\text{awardtrophytournamentisthechampionshipgameofthenationalsport}(X,Z) \Leftarrow \text{athleteplayssport}(Y,Z) \wedge \text{athletewinsawardtrophytournament}(Y,X)$
9	$\text{professionistypeofprofession}(X,Z) \Leftarrow \text{professionistypeofprofession}(X,Y) \wedge \text{professionistypeofprofession}(Y,Z)$
7	$\text{headquarteredin}(X,Z) \Leftarrow \text{companyalsoknownas}(X,Y) \wedge \text{headquarteredin}(Y,Z)$
4	$\text{worksfor}(X,Z) \Leftarrow \text{companyalsoknownas}(Y,Z) \wedge \text{worksfor}(X,Y)$
4	$\text{ismultipleof}(X,Z) \Leftarrow \text{ismultipleof}(X,Y) \wedge \text{ismultipleof}(Y,Z)$
1	$\text{personbornincity}(X,Z) \Leftarrow \text{personborninstateorprovince}(X,Y) \wedge \text{citylocatedinstate}(Z,Y)$

FIGURE 17: GRL outputed rules for NELL's KB

rank	YAGO's extracted inference rules
20	hasSuccessor(X,Z) <= created(X,Y) ^ created(Y,Z)
14	isAffiliatedTo(X,Z) <= hasPredecessor(X,Y) ^ isAffiliatedTo(Y,Z)
10	hasChild(X,Z) <= hasChild(Y,Z) ^ isMarriedTo(Y,X)
5	hasPredecessor(X,Z) <= hasPredecessor(X,Y) ^ hasPredecessor(Y,Z)
4	isAffiliatedTo(X,Z) <= hasSuccessor(X,Y) ^ isAffiliatedTo(Y,Z)
2	interestedIn(X,Z) <= hasWonPrize(X,Y) ^ hasWonPrize(Z,Y)
1	locatedIn(X,Z) <= hasCapital(Y,X) ^ locatedIn(Y,Z)

FIGURE 18: GRL outputed rules for YAGO's KB

.2 GRL vs PRA vs AMIE

In this section, some rules found by PRA and AMIE from the experiment comparing them to GRL are present. The Table of **Figure 19** presents some of the rules found by PRA and the Table of **Figure 20** presents some of the rules found by AMIE. Both running over NELL's OKB of iteration 885.

PRA's output for NELL 885 OKB	
<hasbrother>(X,Z) <=	<personhasfather>(Y,X) ^ <agentinteractswithagent>(Z,Y) ^ <brotherof>(Z,X)
<citylocatedingeopoliticallocation>(X,Z) <=	<locatedat>(X,Z) ^ <agentcontrols>(Y,X)
<inverseofanimalsuchasinvertebrate>(Z,W) ^ <animalsuchasinvertebrate>(W,Z) ^	<agriculturalproducttoattractinsect>(X,Z) <= <inverseofinvertebratefeedonfood>(X,Y) ^ <animalsuchasinsect>(W,Z)
<organizationheadquarteredincity>(X,Z) <=	<hasofficeincity>(X,Z) ^ <worker>(Y) ^ <agentinteractswithagent>(Y,X)
<hasofficeincity>(X,Z) <=	<radiostation>(X) ^ <radiostationincity>(X,Z)
<fatherofperson>(X,Z) <=	<personhasfather>(Y,X) ^ <personhasparent>(Z,X) ^ <agentinteractswithagent>(X,Z)
<awardtrophytournamentisthechampionshipgameofthenationalsport>(X,Y) <=	<trophywonbyteam>(X,Y) ^ <agentcontrols>(W,Y) ^ <athleteledsportsteam>(W,Y) ^ <teamplyssport>(Y,Z)
<teamplysinleague>(X,Z) <=	<teamplyssport>(X,Y) ^ <subpartoforganization>(Y,Z) ^ <leagueteams>(Z,Y) ^ <subpartoforganization>(X,Z)

FIGURE 19: PRA outputed rules for NELL's OKB (iteration 885)

.3 PrOntExt Results running with NELL

In this section we present some results of PrOntExt. The relation presented are all already inserted at NELL's OKB. In the table of **Figure 21** we present a list with some relations: the first column is the relation name, the second is the relation that is

AMIE's output for NELL 885 OKB	
<ismultipleof>(X, Y) <= <animalistypeofanimal>(X, Y)	
<synonymfor>(X, Y) <= <synonymfor>(Y, X)	
<animaleatvegetable>(X, Z) <= <agentcompeteswithagent>(X, Y) ^ <animaleatfood>(Y, Z)	
<arthropodandotherarthropod>(X, Z) <= <arthropodandotherarthropod>(X, Y) ^ <arthropodcalledarthropod>(Y, Z)	
<animalsuchasinvertebrate>(X, Z) <= <invertebratefeedonfood>(X, Y) ^ <invertebratefeedonfood>(Z, Y)	
<plantincludeplant>(X, Z) <= <plantgrowinginplant>(Z, Y) ^ <plantincludeplant>(X, Y)	
<animalpreyson>(X, Z) <= <animalpreyson>(Y, Z) ^ <mammalsuchasmammal>(Y, X)	
<acquired>(X, Y) <= <subpartoforganization>(Y, X)	
<teamwontrophy>(X, Z) <= <athleteledsportsteam>(X, Y) ^ <athletewinsawardtrophytournament>(Y, Z)	

FIGURE 20: AMIE outputed rules for NELL's OKB (iteration 885)

the “father” of the current relation⁸, the third and fourth is the two categories that the relation connects.

relation name	generalization	range	domain
countryIsTheHomeOfSportsteam	locationRelatedToAgent	country	sportsteam
languageSpokenAtIsland	relatedTo	language	island
astronautWasOnAMissionWithAstronaut	relatedTo	astronaut	astronaut
vegetableIsProducedAtCountry	itemExistsAtLocation	vegetable	country
meatComesFromAnimal	relatedTo	meat	animal
stateorprovinceAttainsTrainstation	relatedTo	stateorprovince	trainstation
coachStudiedAtUniversity	relatedTo	coach	university
buildingIsPartOfUniversity	relatedTo	building	university
beverageCanSoftendisease	relatedTo	beverage	disease
arthropodSuchAsCrustacean	generalizationOf	crustacean	arthropod
bakegoodIsTypicalInCountry	relatedTo	bakedGood	country
buildingFeatureBuiltAboveLake	relatedTo	buildingfeature	lake
mountainAtgeopoliticallocation	locatedAt	mountain	geopoliticallocation
wineryAtFarm	relatedTo	winery	farm
grainUsedToMakeCandy	relatedTo	grain	candy
invertebrateTurnIntoInsect	relatedTo	invertebrate	insect
mammalInducesEmotion	relatedTo	mammal	emotion
clothMadeOfPlant	relatedTo	clothing	plant
invertebrateHasColor	relatedTo	invertebrate	color
newspaperHasWebsite	organizationHasOfficialWebsite	newspaper	website
mammalEatsInsect	animalThatFeedOnInsect	mammal	insect
diseaseCausesPhysiologicalCondition	relatedTo	disease	physiologicalcond

FIGURE 21: Relations found by PrOntExt running with NELL

In the table of **Figure 22** are present four relations and its seed instances. As mentioned before, some comes from Prophet, another ones we manually added.

⁸It's like the hierarchy of categories, but NELL also have a hierarchy for the relations either

relation name	seed instances
languageSpokenAtIsland	{“Spanish”, “Santa Cruz del Islote”} {“Cantonese”, “Ap Lei Chau”} {“Bantu Swahili”, “Mingango Island”} {“French”, “Fadiouth”} {“Maldivian”, “Malé”} {“Marshallese”, “Ebeye”} {“Portuguese”, “Ilha de Moçambique”} {“English”, “Manhattan”} {“Marathi”, “Salsette Island”} {“French”, “Île Saint-Louis”} {“Arabic”, “Arwad”} {“Finnish”, “Lilla Essingen”} {“Cantonese”, “Tsing Yi”} {“English”, “Roosevelt Island”} {“German”, “Lübeck”}
vegetableIsProducedAtCountry	{“Dry Bean”, “India”} {“Dry Bean”, “Brazil”} {“Onion”, “China”} {“Onion”, “India”} {“Garlic”, “India”} {“Garlic”, “China”} {“Spinach”, “China”} {“Spinach”, “usa”} {“Soybean”, “usa”} {“Soybean”, “Brazil”} {“Tomato”, “China”} {“Tomato”, “India”} {“Ginger”, “India”} {“Pulses”, “Mozambique”} {“Pulses”, “India”} {“Cabbage”, “China”} {“Cabbage”, “India”} {“Cassava”, “Nigeria”}
invertebrateTurnIntoInsect	{“caterpillar”, “butterfly”} {“caterpillar”, “moth”} {“planula”, “cnidarians”} {“grub”, “bettle”} {“maggot”, “flies”} {“maggot”, “bee”} {“maggot”, “wasp”} {“wiggler”, “mosquito”} {“larva”, “snakefly”} {“larva”, “alderfly”} {“larva”, “scorpionfly”} {“larva”, “caddisfly”} {“larva”, “fishfly”} {“larva”, “fleas”} {“larva”, “Twisted-winged parasite”} {“larva”, “antlion”}
mammalInducesEmotion	{“kids”, “stress”} {“kids”, “happiness”} {“kids”, “tenderness”} {“kids”, “fatigue”} {“dogs”, “fatigue”} {“dogs”, “happiness”} {“dogs”, “fear”} {“cats”, “happiness”} {“cats”, “tenderness”} {“dogs”, “stress”} {“tiger”, “fear”} {“lion”, “fear”} {“wale”, “impressed”} {“wale”, “amazed”} {“elephant”, “amazed”} {“elephant”, “impressed”} {“rat”, “nausea”} {“bat”, “nausea”} {“bat”, “fear”}

FIGURE 22: Relations found by PrOntExt running with NELL

Bibliography

- [1] Andrew Carlson, Justin Betteridge, Bryan Kisiel, Burr Settles, Estevam R. Hruschka Jr., and Tom M. Mitchell. Toward an architecture for never-ending language learning. In *Proceedings of AAAI*, 2010.
- [2] In Luna Dong, K Murphy, E Gabrilovich, G Heitz, W Horn, N Lao, Thomas Strohmamm, Shaohua Sun, and Wei Zhang. Knowledge vault: A web-scale approach to probabilistic knowledge fusion, 2014.
- [3] Kurt Bollacker, Colin Evans, Praveen Paritosh, Tim Sturge, and Jamie Taylor. Freebase: a collaboratively created graph database for structuring human knowledge. In *In Proceedings of SIGMOD*, 2008.
- [4] Fabian M Suchanek, Gjergji Kasneci, and Gerhard Weikum. Yago: a core of semantic knowledge. In *In Proceedings of WWW*, 2007.
- [5] Johannes Hoffart, Fabian M. Suchanek, Klaus Berberich, and Gerhard Weikum. Yago2: A spatially and temporally enhanced knowledge base from wikipedia. *Artificial Intelligence*, 194:28–61, 2013.
- [6] Farzaneh Mahdisoltani, Joanna Biega, and Fabian M. Suchanek. Yago3: A knowledge base from multilingual wikipedias. *Conference on Innovative Data Systems Research*, 2015.
- [7] T. Mitchell, W. Cohen, E. Hruschka Jr., P. Talukdar, J. Betteridge, A. Carlson, B. Dalvi, M. Gardner, B. Kisiel, J. Krishnamurthy, N. Lao, K. Mazaitis, T. Mohamed, N. Nakashole, E. Platanios, A. Ritter, M. Samadi, B. Settles, R. Wang, D. Wijaya, A. Gupta, X. Chen, A. Saporov, M. Greaves, and J. Welling. Never-ending learning. In *Proceedings of the Twenty-Ninth AAAI Conference on Artificial Intelligence (AAAI-15)*, 2015.

-
- [8] L. Getoor and C. P. Diehl. Link mining: a survey. *ACM SIGKDD Explorations Newsletter*, 7:3–12, 2005.
- [9] F. Lorrain and H. White. Structural equivalence of individuals in social networks. *Journal of Mathematical Sociology*, 1:49–80, 1971.
- [10] Lada Adamic and Eytan Adar. Friends and neighbors on the web. *Social Networks*, 25(3):211–230, 2003.
- [11] A. Lancichinetti and S. Fortunato. Community detection algorithms: A comparative analysis. *Physical Review E*, 80(5), 2009.
- [12] Ni Lao and William W. Cohen. Relational retrieval using a combination of path-constrained random walks. *Machine learning*, 81(1):53–67, 2010.
- [13] Ni Lao, Tom Mitchell, and William W. Cohen. Random walk inference and learning in a large scale knowledge base. In *Proceedings of the 2011 Conference on Empirical Methods in Natural Language Processing*, pages 529–539, Edinburgh, Scotland, UK., July 2011. Association for Computational Linguistics. URL <http://www.aclweb.org/anthology/D11-1049>.
- [14] Matt Gardner, Partha Pratim Talukdar, Bryan Kisiel, and Tom Mitchell. Improving learning and inference in a large knowledge-base using latent syntactic cues. In *Proceedings of the 2013 Conference on Empirical Methods in Natural Language Processing (EMNLP 2013)*, 2013.
- [15] Ana Paula Appel and Estevam Rafael Hruschka Junior. Prophet – a link-predictor to learn new rules on nell. In *Proceedings of the 2011 IEEE 11th International Conference on Data Mining Workshops, ICDMW '11*, pages 917–924, Washington, DC, USA, 2011. IEEE Computer Society. ISBN 978-0-7695-4409-0. doi: 10.1109/ICDMW.2011.142. URL <http://dx.doi.org/10.1109/ICDMW.2011.142>.
- [16] T. Mohammed, E. R. Hruschka Jr., and T. M. Mitchell. Discovering relations between noun categories. *EMNLP, ACL:1447–1455*, 2011.
- [17] L. Chunlei and T. Mitchell. Semi-supervised data clustering with coupled non-negative matrix factorization: Sub-category discovery of noun phrases in nell’s knowledge base, 2011.

- [18] J. B. MacQueen. Some methods for classification and analysis of multivariate observations. *Proceedings of 5th Berkeley Symposium on Mathematical Statistics and Probability*, (1):281–297, 1967.
- [19] A. P. Appel and E. R. Hruschka Jr. Centaurs a component based framework to mine large graphs. In *XXV BRAZILIAN SYMPOSIUM ON DATABASES*, pages 1–8, Belo Horizonte, MG, Brazil, 2010.
- [20] Linyuan Lu and Tao Zhou. Link prediction in complex networks: A survey. In *Physica A: Statistical Mechanics and its Applications*, volume 390, pages 1150 – 1170, 2011. doi: DOI:10.1016/j.physa.2010.11.027.
- [21] David Liben-Nowell and Jon Kleinberg. The link prediction problem for social networks. In *Proceedings of CIKM*, pages 556–559, New York, NY, USA, 2003. ACM. ISBN 1-58113-723-0. doi: <http://doi.acm.org/10.1145/956863.956972>.
- [22] D. J. Watts and S. H. Strogatz. Collective dynamics of small-world networks. *Nature*, 393(6684):440–442, June 1998. ISSN 0028-0836. doi: <http://dx.doi.org/10.1038/30918>.
- [23] Lars Backstrom and Jure Leskovec. Supervised random walks: predicting and recommending links in social networks. In Irwin King, Wolfgang Nejdl, and Hang Li, editors, *Proceedings of WSDM 2011*, pages 635–644. ACM, 2011. ISBN 978-1-4503-0493-1. doi: <http://doi.acm.org/10.1145/1935826.1935914>.
- [24] Mohammad Al Hasan, Vineet Chaoji, Saeed Salem, and Mohammed Zaki. Link prediction using supervised learning. In *In Proc. of SDM 06 workshop on Link Analysis, Counterterrorism and Security*, 2006.
- [25] B. Taskar, M. Wong, P. Abbeel, and D. Koller. Link prediction in relational data, 2004.
- [26] Alexandrin Popescul, Rin Popescul, and Lyle H. Ungar. Statistical relational learning for link prediction, 2003.
- [27] Tao Zhou, Linyuan Lu, and Yi-Cheng Zhang. Predicting missing links via local information. *The European Physical Journal B - Condensed Matter and Complex Systems*, 71(4):623–630, October 2009. ISSN 1434-6028. doi: 10.1140/epjb/e2009-00335-8.

- [28] Rui Xu and II Wunsch, D. Survey of clustering algorithms. *Neural Networks, IEEE Transactions on*, 16(3):645–678, 2005.
- [29] Tom M. Mitchell. *Machine Learning*. McGraw Hill, 1997.
- [30] Elena Deza and Michel Marie Deza. Encyclopedia of distances. *Springer*, page p.94, 2009.
- [31] Stephen Kelley, Mark Goldberg, Malik Magdon-Ismael, Konstantin Mertsalov, William Wallace, and Mohammed Zaki. graphont: An ontology based library for conversion from semantic graphs to jung. *Intelligence and Security Informatics, ISI'09*, pages 170–172, 2009.
- [32] S. Milgram. The small-world problem. *Psychology Today*, pages 62–67, 1967.
- [33] Sindhu Raghavan and Raymond J. Mooney. Online inference-rule learning from natural-language extractions. *Statistical Relational Artificial Intelligence, AAAI 2013 Workshop*:57–63, 2013.
- [34] J. Cowie and W. Lehnert. Information extraction. *CACM*, 39(1):80–91, 1996.
- [35] S Sarawagi. Information extraction. *Foundations and Trends in Databases*, 1(3): 261–377, 2008.
- [36] Sebastian Riedel, Limin Yao, Andrew McCallum, and Benjamin M. Marlin. Relation extraction with matrix factorization and universal schemas. In *Proceedings of the 2013 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pages 74–84, Atlanta, Georgia, June 2013. Association for Computational Linguistics. URL <http://www.aclweb.org/anthology/N13-1008>.
- [37] Luis Galarraga, Christina Teflioudi, Katja Hose, and Fabian M. Suchanek. Amie: Association rule mining under incomplete evidence in ontological knowledge bases. *International World Wide Web Conference(WWW')*, 2013.
- [38] Raphael G. S. Dos Santos and Estevam R. Hruschka Jr. Markov logic scalability in a never-ending language learning system. In *Proceedings of the NewsKDD Workshop: Data Science for News Publishing. Workshop at the 20th ACM-SIGKDD Conference on Knowledge Discovery and Data Mining - KDD2014*, pages 21–25, 2014.

- [39] Matt Gardner, Partha Talukdar, and Tom M. Mitchell. Combining vector space embeddings with symbolic logical inference over open-domain text. *AAAI Spring Symposium*, 2015.
- [40] J. R. Quinlan. Learning logical definitions from relations. *Machine Learning*, 5(3): 239–266, 1990.
- [41] E. Mccreath and A. Sharma. Lime: A system for learning relations. In *Ninth International Workshop on Algorithmic Learning Theory*, Springer-Verlag:336–374, 1998.
- [42] A. Srinivasan. The aleph manual. In <http://web.comlab.ox.ac.uk/oucl/research/areas/machlearn/Aleph/>, 2001.
- [43] K. Kersting and L. De Raedt. Basic principles of learning bayesian logic programs. *Berlin, Heidelberg: Springer-Verlag*, 2008.
- [44] Lucas Fonseca Navarro, Ana Paula Appel, and Estevam Rafael Hruschka Jr. Graphdb - storing large graphs on secondary memory. *ADBIS Special session on Big Data: New Trends and Applications (BiDaTA) in conjunction with the 17th East-European Conference on Advances in Databases and Information Systems (ADBIS)*, 17:177–186, 2013.
- [45] Aaron Clauset, Cristopher Moore, and M. E. J. Newman. Hierarchical structure and the prediction of missing links in networks. *Nature*, 453(7191):98–101, Nov 2008. ISSN 0028-0836. doi: 10.1038/nature06830.
- [46] D. Arthur and S. Vassilvitskii. K-means++: the advantages of careful seeding. *SODA '07 Proceedings of the eighteenth annual ACM-SIAM symposium on Discrete algorithms*, pages 1027–1035, 2007.
- [47] Schaeffer E. S. Graph clustering. *Computer Science Review*, 1(1):27–64, 2007.
- [48] Wilson Wong, Wei Liu, and Mohammed Bennamoun. Ontology learning from text: A look back and into the future. *ACM Computing Surveys (CSUR)*, 44, August 2012.
- [49] Paola Velardi, Stefano Faralli, and Roberto Navigli. Ontolearn reloaded: A graph-based algorithm for taxonomy induction. *Computational Linguistics Journal*, 2013.

-
- [50] Philipp Cimiano, Andreas Hotho, and Steffen Staab. Learning concept hierarchies from text corpora using formal concept analysis. *Journal of Artificial Intelligence Research*, 24(1):305–339, July 2005.
- [51] D. Hindle. Noun classification from predicate-argument structures. *Proceedings of the 14th International Conference on Computational Linguistics (COLING)*, pages 268–275, 1990.
- [52] S. Caraballo. Automatic construction of a hypernym-labeled noun hierarchy from text. *Proceedings of the 38th Annual Meeting of the Association for Computational Linguistics (ACL)*, pages 120–126, 1999.
- [53] D. Faure and C. Nedellec. A corpus-based conconceptual clustering method for verb frames and ontology. *Proceedings of the LREC Workshop on Adapting lexical and corpus resources to sublanguages and application*, pages 5–12, 1998.
- [54] G. Bisson, C. Nedellec, and L. Canamero. Designing clustering methods for ontology building: The mo'k workbench. *Proceedings of the ECAI Ontology Learning Workshop*, pages 13–19, 2000.
- [55] Zhexue Huang. Extensions to the k-means algorithm for clustering large data sets with categorical values. *Data Mining and Knowledge Discovery*, 2(3):283–304, 1998.
- [56] A. Ahmad and L. Dey. A k-mean clustering algorithm for mixed numeric and categorical data. *Data & Knowledge Engineering*, 63:502–527, 2007.
- [57] Marti A. Hearst. Automatic acquisition of hyponyms from large text corpora. *COLING '92 Proceedings of the 14th conference on Computational linguistics*, 2: 539–545, 1992.
- [58] Paul Jaccard. Etude comparative de la distribution florale dans une portion des alpes et des jura. *Bulletin de la Société Vaudoise des Sciences Naturelles*, 37: 547–579, 1901.