Link Prediction in Large Directed Graphs

Dario Garcia-Gasulla

LSI

Universitat Politècnica de Catalunya - BarcelonaTECH

A thesis submitted for the degree of

Ph.D. in Artificial Intelligence

2015

Acknowledgments

Back in 2009, while finishing my Computer Science degree, I woke up in the middle of the night with an idea. The idea that semantics, that true semantics, could only be found in the relations among entities. Indeed, for me a book is defined by concepts like page, reading, paper, word, author, library, burn, "I, Robot", and many others. And the removal of any of those concepts from my mind would change what a book actually means. The simplicity and clarity of this idea (the semantics are in the relations!) has haunted me for six years. My wife, who was with me on that fateful 2009 night and had to endure six years of the same conversation topic, is witness. Gràcies Sara.

In 2009 I believed that idea was key for the development of AI. And after finishing my CS degree, doing a master on AI, and writing this thesis, I still believe it. My faith, my obsession with this idea is the only reason why this thesis exists. It's a piece of myself. This thesis is not the answer to AI however, not even close. It is, I believe, a step in the right direction. My thesis director is also to blame here. He trusted me and gave me freedom to make lots of mistakes, providing the means to get me where I am now. Gracias Ulises.

During these six years I've been talking about my thesis with anyone who would be too polite to refuse. I'd like to thank all of them as well. My friends. My family. Specially my parents, without whom I wouldn't have made it to the university in the first place. My colleagues at KEMLg, where I leanrt everything I know about research. Working there was really fun, thank you all. Jesús and Eduard from BSC, who introduced me into the HPC world and coped my ignorance. And to you, who is about to read this thesis.

Have fun,

Dario

This work has been partly supported by the following projects

- CLOUD project - Co-financed by the Spanish Minister of Science and Innovation - IPT-2011-0813-020000
- SUPERHUB project. SUstainable and PERsuasive Human Users moBility in future cities - Co-financed by the European Commission - ICT-FP7-289067
- Severo Ochoa grant. Through the Barcelona Supercomputing Center, Computer Science department.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

In the early XXI century a connectivist approach to Machine Learning (ML) gained scientific attention, where data is interpreted as a set of highly interconnected elements composing a network. While Data Mining (DM) and ML had traditionally focused on *intra-entity* patterns for knowledge discovery (*i.e.*, instances and their attributes), this new trend focused on *inter-entity* patters for that same purpose (*i.e.*, entity-entity relations). The change of perspective from individuals to communities, revealed a set of problems closely related with the concept of network that had not been successfully solved so far. For example, discovering communities of interconnected entities, finding new relations among entities, or determining the role and relevance of entities within a community. These are all problems fitting this new learning paradigm.

The community-based approach to learning has been adopted by numerous branches of science and has received equally numerous names. Graph-based data mining (Holder and Cook, 2009; Washio and Motoda, 2003), Statistical Relational Learning (Getoor and Taskar, 2007; Nickel et al., 2011), Link Mining (Getoor and Diehl, 2005; Lü and Zhou, 2011), Network or Link Analysis (Guimerà and Sales-Pardo, 2009; Mantrach et al., 2010), Network Science (Lichtenwalter et al., 2010) or Structural Mining (Cook et al., 2001) are all names used to define knowledge discovery tools based on the same basic precept: exploiting structural properties of high-dimensional, inter-connected data sets to learn from the relational patterns of its entities. For the sake of simplicity from now on we refer to these methods using the general term *graph mining*.

Not by chance, the popularization of graph mining came hand-in-hand with the explosion of the Internet. As the web graph, social networks and other large digitalized data sets became available, ML and DM researchers realized there was a lack of tools specifically designed for exploiting information represented as a network of interconnected entities. Many different problems and algorithms have been proposed since then (as we will review in §2), and what is more important, many different applications have been found, making of graph mining a hot topic nowadays. Life sciences (Airoldi et al., 2006; Guimera and Amaral, 2005; Von-Mering et al., 2002), sociology and social networks (Adamic and Adar, 2003; Murata and Moriyasu, 2008; Watanabe and Suzumura, 2013), collaboration analysis (Aggarwal et al., 2013; Lü and Zhou, 2011; Tylenda et al., 2009), business and product recommendation (Burke, 2002; Huang et al., 2005), and even law enforcement and anti-terrorism (Al Hasan et al., 2006; Clauset et al., 2008; Krebs, 2002) are examples of domains to which graph mining has been recently applied.

A frequent feature of graph-based data sets is their large size. In traditional attribute-based data representations each specific instance may include a large amount of information by itself through the definition of features. This allows attribute-based DM and ML methods to achieve successful learning even when data sets are composed by a relatively small number of instances. Topology-based mining on the other hand typically focuses on the existence of relations among entities, and rarely exploits many (if any) internal attributes of either the entities or their relations. Hence, graph mining algorithms tend to require graphs with many vertices and edges to achieve learning, as only networks with complex topologies can express complex domains. The coming of the digitalization age has helped in that regard, making available huge amounts of information in what is popularly known as the Big Data. However, given their *preference* for large data sets, graph mining algorithms are particularly affected by the computational problems and limitations arising from processing huge amounts of data.

In graph mining, knowledge emerges from the combination and interaction of multiple entities. Nevertheless, the patterns discovered by graph mining methods are in most cases local to some degree (*i.e.*, they refer to a subset of the whole graph topology). Discovered knowledge referring to the whole graph is either too abstract or too imprecise as to be useful, and is rarely sought. Instead, graph mining leans towards the distributed analysis of sub-parts of the graph, to discover partially local models. When dealing with huge data sets [1], this scaled approach becomes a key advantage as it allows graph mining tasks to be efficiently parallelized. In an analogous decentralizing process,

[1]Today *huge* is a fuzzy concept ranging from several gigabytes to petabytes of data

High Performance Computing (HPC) is partly focused on parallelization through task distribution on multiple autonomous computing units (*e.g.*, current supercomputers). This synergy between graph mining and HPC on *how* problems are tackled benefits both sides, as graph mining can aim at processing large scale graphs through HPC, and HPC finds in graph mining a field of application for its infrastructure and tools. In that regard, given the vast and rich knowledge sources available from graph mining, and the potential benefits graph mining can produce, we expect *high performance graph mining* to become a hot topic in the near future.

1.1 Scope of this Thesis

In this thesis we focus on a graph mining task known as Link Prediction (LP) when applied to directed graphs. The goal of LP is to discover new edges among the vertices of a graph given their previous relations (*i.e.*, the graph topology). We have studied the current state-of-the-art of graph mining and of LP in particular, to identify the current limitations of the field. We have found that LP is close to become a powerful tool applicable to a wide variety of domains, if only two main issues can be tackled. Those issues are *precision* and *scalability.*

Let us first consider precision. Since graphs built from real world data are typically sparse (in our current experience graphs rarely have more than ten edges per vertex on average), LP in a directed graph can be understood as a classification problem in a highly imbalanced domain. On one hand we have a small positive class of edges, containing those edges which do or should exist in the graph, normally linear in size with respect to the number of vertices (*e.g.*, ten times the number of vertices). On the other hand we have a huge negative class of edges, containing those edges which could exist but do not and should not, quadratic with respect to the number of vertices. The difference in size between both classes (*i.e.*, the class imbalance) grows linearly with the number of vertices, such that for graphs with 1 million vertices it is common to have 100,000 edges in the negative class for each edge found in the positive class. The problem of LP then becomes that of telling apart actual edges of the small class from non-existing edges of the large class. Keeping a good precision in this type of needle in a haystack problem turns out to be very difficult, as the smallest rate of false positive acceptance will amount to a huge absolute number of wrongfully predicted edges (*i.e.*, false positives). As a result, the precision of LP algorithms quickly decreases as the size of the graphs being computed grows. This condition has restrained the broad applicability of LP so far, particularly on large scale graphs. In this thesis we will discuss various techniques for increasing precision in LP, such as the design of algorithms matching the inherent properties of graphs, and focusing on high confident predictions at the cost of a smaller recall.

The other main issue key for the future application of LP is scalability. Data-intensive tasks (one where most of the computational time is spent fetching data from memory instead of processing it) such as the one of performing LP on a large graph is a challenging topic in computation. This type of task becomes particularly complex when storing and accessing large amounts of data. As we will discuss in this thesis, LP may be an exemplifying case of those problems, and much can be learnt from its exploration. We will discuss how to represent data, structure code and handle memory in order to make the mining of large graphs feasible. We approach the HPC field by discussing the implementation of LP algorithms in various parallel programming models, and by testing them on a HPC environment such as the MareNostrum[1] and TSUBAME[2] supercomputers.

While exploring these two topics, precision and scalability, we will also overview applicability. By testing different data sets we intend to exemplify the utility of LP in general and of our proposed approach in particular. We also hope to illustrate the potential applications of LP by applying it to a variety of domains.

1.2 Main Goals

At this point we can outline the main goals of this thesis. With this work we intend to

1. study the limitations of current solutions for LP on large directed graphs, from both the perspective of predictive performance and of computational scalability.

[1]MareNostrum supercomputer, managed by the Barcelona Supercomputing Center. http://www.bsc.es

[2]TSUBAME supercomputer, managed by the Tokyo Institute of Technology. http://www.gsic.titech.ac.jp/en/tsubame

2. propose, implement and evaluate novel LP methodologies specifically designed for mining directed graphs, by using properties often found on the topology of large directed networks.

3. explore the usability of LP, particularly when applied to large directed graphs. Identify potential domains of application and the challenges those represent.

4. contribute to the integration of graph mining and HPC, by applying state-of-the-art HPC models, tools and infrastructure to the LP problem.

1.3 Research Questions

To avoid getting lost in this document, one must be aware that this thesis revolves around the concept of hierarchy. The original motivating research question of this work was, *what is the role of hierarchies in the topology of large graphs?* This rather abstract question spawned others as we considered how to formally measure the importance of hierarchies, like, *can hierarchies be used to predict relations within a graph? Are hierarchies implicitly represented within natural, informal graphs?* While trying to solve those, other operational questions arose, questions we needed to tackle in order to solve the previous ones. Questions such as, *is it feasible nowadays to massively run graph mining algorithms on large graphs? Are HPC infrastructure and tools ready to deal with the challenges of the emerging graph mining problems?*

Partially or fully answering all these questions is eventually the scientific purpose of this thesis. Nevertheless, as we will see in the remaining of the document, solving these questions brings forth more questions. Some of those we will try to solve as well, many others will remain as future challenges for the scientific community and ourselves.

1.4 Plan of this Thesis

We start this thesis with a study of the current state-of-the-art of LP in §2, describing the various approaches available and focusing on those most relevant for the scope of this thesis. This will serve to fix the vocabulary and the limits of discussion. In §3 we discuss the problem of LP in detail, analyzing the different interpretations and difficulties the problem poses. After that we argue about the relation between hierarchies, large directed graphs and LP, and introduce our proposed LP score in §4. This includes a novel algorithm for LP, two modifications for it, as well as a hierarchical version of the current best algorithms of LP according to the bibliography. Chapter §5 is entirely dedicated to discuss how LP scores are evaluated. We present a study of the most frequently used methods of performance evaluation, as well as our own proposal which targets applicability. We describe the data sets we have used for testing in Chapter §6, list their properties and explain how we built a directed graph from them. This Chapter pays special attention to the family of webgraphs, as these will be one of the main targets in our evaluation. Then we test several LP scores on all graphs presented, in §7. Tests include an evaluation of the current best algorithms of LP, of their hierarchical version previously presented, and of all the variants of our novel algorithm proposal. Predictive performance for hyperlink prediction (*i.e.*, LP in webgraphs) is studied here as well. In §8 we look at the problem of LP on large graphs from the perspective of computation, tackling topics such as optimization and parallelization. In this Chapter we discuss the emerging parallel programming models for graphs, and present two different implementations we have developed for the same LP problem. We discuss some possible lines of future research in §10, outline our main conclusions in §9 and end with a detailed summary of this thesis in §11. Appendix §A contains tables and figures with additional results of the tests performed in §7.

Chapter 2

Background

Certainly not by coincidence, graph mining became a popular research field by the time the Internet grew and gathered world-wide attention. Previous contributions to the field exist, but it was not until the early XXI century that a wide community of ML and DM researchers collaboratively explored how to extract knowledge through the structural properties of high-relational data (Adamic and Adar, 2003; Getoor and Diehl, 2005; Inokuchi et al., 2000; Kleinberg, 1999; Newman, 2001; Taskar et al., 2003; Washio and Motoda, 2003). Behind this new approach to ML was the idea that relations among entities could be used to understand data, a kind of understanding that could not be achieved through traditional attribute-based proposals. The availability of large, topology based data sets like the web graph provided support to that idea as it showed, first the lack of ML tools natively appropriate to deal with that kind of knowledge, and second the existence of important problems to be solved through those tools. As a result the interest on exploiting relations among entities for learning purposes has been continuously growing.

In graph mining one assumes information is represented through relations. That makes graph mining coherent with the core principles of relational data. Currently popular solutions on how to structure and represent relational data however remain focused on internal attributes, differing from the graph mining focus on external relations. In that regard, alternatives following the same direction that graph mining are emerging in other fields. Data storage systems, for example, have witnessed the appearance of graph databases, challenging traditional solutions; where relational databases (composed by tables and data items) are idoneous for representing lists of entities and their properties, graph databases (containing index-free adjacency) are the natural representation of topology-centric data. Hence, graph databases outperform relational databases in those operations where structure is centric, like traversing the graph (Vicknair et al., 2010).

One of the earliest and more detailed summaries of topology-based algorithms for ML was presented in (Getoor and Diehl, 2005). In this paper authors identify eight different tasks within the research field they called *Link Mining*, splitting those tasks in three categories depending on which part of structural data they focus: Object-related tasks, link-related tasks, and graph-related tasks. This work proved to be prescient, as most of the described problems have received growing attention since then. Next we outline the ones we consider to be more relevant nowadays.

- **Link-based object ranking** (LBR), is the task of calculating an ordering of entities within a network. LBR algorithms produce a relevance score for each entity based on the patterns of its relations. Vertices are then compared and ordered according to this score to produce a global rank of entities. The importance of LBR lies mostly in its direct applicability to real world problems, as it has become a key component in the design of web search engines. The most relevant LBR algorithms are HITS (Kleinberg, 1999) and PageRank (Page et al., 1999).

- **Group detection**, also referred to as *community detection*, is a clustering task in which sets of entities are identified based on their structural properties. This can be used to split a network into closely related groups, or to find the community neighbours of a given vertex. Several multidisciplinary solutions have been proposed for this problem (Fortunato, 2010), being stochastic block modeling (Karrer and Newman, 2011) the most followed approach nowadays.

- **Subgraph discovery**, also known as *frequent subgraph mining*; is a discovery task seeking recurrent patterns of relations within a relational set. Most proposals are based on the Apriori algorithm (Agrawal et al., 1994), such as AGM (Inokuchi et al., 2000), an algorithm that finds subgraphs satisfying a minimum support based on the concept of subset induction. Subgraph discovery algorithms can be used for graph classification, sorting graphs based on which frequent subgraphs they contain, and more specifically to find biological modules (Hu et al., 2005).

- **Link prediction**; is a learning task in charge of finding new relations within a relational data set. The rest of this chapter will be dedicated to analyze the current state-of-the-art of LP in detail, as it will be a main topic of this thesis.

2.1 Link Prediction

According to (Lü and Zhou, 2011) link prediction (LP) algorithms can be classified in three different families: similarity-based algorithms, maximum likelihood algorithms and probabilistic models. This last family however is also named statistical relational models in the bibliography (Getoor and Taskar, 2007), term used to include other statistical but non-probabilistic solutions, like tensor factorization.

Statistical relational models typically build a joint probability distribution representing the graph, based on all the edges found in it. Given this distribution, and through the application of different types of inference, these algorithms can estimate the likelihood with which edges not found in the graph may actually exist (Getoor and Taskar, 2007). Algorithms within this field are frequently based on Markov Networks (for directed graphs) or Bayesian Networks (for undirected graphs). The sub-type of statistical relational models that do not build a probabilistic model and instead are based on purely statistical properties are those based on tensor factorization (which is sometimes also used in combination with probabilistic models (Sutskever et al., 2009)). Tensor factorization solutions are particularly relevant due to their capability of being directly applied to heterogeneous networks (*i.e.*, networks composed by more than one type of relation) (Franz et al., 2009; Nickel et al., 2011).

Maximum likelihood algorithms of LP assume a given structure within the graph (*e.g.*, a hierarchy, a set of communities, *etc.*) and build a model according to it. Then, based on this model, one can calculate the likelihood of edges not found in the original graph through maximum likelihood algorithms. The most prominent contribution of these algorithms is their capacity to provide insight into the composition of the graph (*i.e.*, how is its topology defined and why), information that can be used for other purposes beyond LP. An example of this kind of algorithms is the Hierarchical Random Graph, see (Clauset et al., 2008), which builds a dendrogram model representing a hierarchical abstraction of the graph. A set of connection probabilities is inferred from the consensus dendogram that most accurately represents the graph hierarchically.

Based on those one can then estimate which edges are most likely to exist in the original graph according to its hierarchical structure.

The third and last family of LP methods are *similarity-based algorithms*. These algorithms compute a similarity score between each pair of vertices in the graph based on their relations. Then, by comparing the similarity scores of all edges one obtains the likelihood with which an edge between a given pair of vertices exists. In similarity-based algorithms each score is independent from the rest, allowing them to be calculated separately and in parallel. Details on how are scores calculated will be provided in the following section (§2.1.1).

Similarity-based algorithms are the simplest of LP algorithms in computational terms. They require no overall model and can evaluate edges individually and independently without supervision. This simplicity may represent a handicap in certain data sets, *e.g.*, providing lower precisions (Clauset et al., 2008), but it becomes a key advantage when considering scalability. As discussed in §1, the graphs targeted by LP are continuously growing in size, increasing at the same time the computational cost of processing them. Eventually, when graphs reach tens of millions of vertices, building a model for the whole graph simply becomes unfeasible due to the limitation of resources. In this context, where maximum likelihood algorithms and statistical relational models cannot be even tested, similarity-based algorithms gain relevance. Not only are those the only algorithms applicable to large scale graphs, but also their performance remains remarkable even though the LP problem becomes more complicated as graphs grow (see §3.6).

Outside the previously described and well delimited categories of LP algorithms, there are other, more heterogeneous solutions that cannot be categorized accordingly, but which are relevant contributions to the field. Given the complexity of the LP problem (discussed in detail in §3) it is frequent when one tries to solve a particular problem to reach particular solutions. Such is the approach proposed in (Adafre and de Rijke, 2005), an algorithm combining clustering, natural language processing and information retrieval with the goal of finding missing edges among Wikipedia pages. Another interesting example of heterogeneous solution is the work presented in (Aggarwal et al., 2013), where the LP process is decomposed in two steps: a first macro-processing step obtains clusters in the graph given structural properties, while a second micro-processing step tries to find new edges on those clusters based on attribute properties.

This work is particularly designed for heterogeneous networks (*i.e.*, networks composed by more than one type of vertex and/or relation) that grow over time.

2.1.1 Similarity-based Link Prediction

Similarity-based LP algorithms seek properties shared by a given pair of vertices in order to evaluate their similarity independently. Finding the number and length of existing paths from one vertex to the other is nowadays the most popular approach, as a path between two vertices represents a type of commonness. Similarity-based algorithms are categorized in three types based on the maximum path length they explore. *Local similarity-based algorithms* of LP consider only the direct relations of the pair of vertices to evaluate their similarity. *Global similarity-based algorithms* consider paths without length constrain (*i.e.*, all the reachable vertices). And as a trade-off between both, *quasi-local similarity-based algorithms* consider paths with a variable number of steps.

The application of global similarity-based algorithms to large graphs is unfeasible (Lü and Zhou, 2011) due to their computational cost (see §3.5). Regardless, quasi-local and local scores have been shown to achieve similar (Liben-Nowell and Kleinberg, 2007) or even better results (Liu and Lü, 2010; Lü et al., 2009) than global scores in certain domains. As a result global similarity-based LP algorithms are rarely a choice today.

Quasi-local indices try to be a compromise between global an local scores, between efficacy and efficiency, by executing a variable number of steps. Or what is the same, by exploring a sub-graph of variable radius size centered on the edge being evaluated. Quasi-local indices intend to achieve higher precisions by taking into account a larger portion of the graph, while keeping a practical computational cost. The most popular quasi-local indices are based on the random walk model (Lichtenwalter et al., 2010; Liu and Lü, 2010) and on the number of paths between the pair of vertices (Lü et al., 2009). However, in essence all quasi-local indices originate from local scores: when the number of steps is set to one, quasi-local indices are equal to some local score. For example, the quasi-local index *local path index* (LPI)[1] reduces to the local score *common neighbours* (Zhou et al., 2009), while quasi-local indices *superposed random walk* and *local random walk* reduce to the local score *resource allocation* (Liu and Lü, 2010).

[1]Local path index is named LP by its authors. In here we refer to it as LPI to distinguish it from the abbreviation of Link Prediction (LP)

Quasi-local indices often include variables to be fit, such as how is the weight of graph evidence reduced based on its increasing separation from the original vertex. This particular variable follows the topological idea that as we travel further from our source vertex, information found becomes less relevant for it. An algorithm which implements this decaying factor is LPI (Lü et al., 2009), a quasi-local index originally fixed on two steps (*i.e.*, exploring only the neighbours and the neighbours of those neighbours), which includes a free parameter ϵ to reduce the importance of evidence found at step two w.r.t. evidence found at step one. As shown in (Lü et al., 2009) the optimal value for ϵ is around 0.01, which suggests the idea that in large graphs the importance of evidence quickly decays with distance.

A more frequent variable of quasi-local indices that must be fit is the optimal number of steps to be performed (*i.e.*, the maximum depth of graph explored). Even though not all quasi-local indices include it (some have fixed, predefined depths like the previously mentioned LPI algorithm), all quasi-local indices can potentially include it. This kind of LP algorithms must therefore find a way to estimate the optimal number of steps before running on large scale. A straightforward, but computationally expensive solution to be applied on large graphs, is to estimate the optimal value through sampling (Liu and Lü, 2010). A promising and cheaper alternative is to estimate the optimal value of variables from the graph topology; results indicate that the optimal value of variables defining the behavior of quasi-local indices may be correlated with graph features such as the clustering coefficient or the average topological distance (Lü et al., 2009). Independently of the methodology used to determine the number of steps to be performed, it must be kept in mind that every further step taken by quasi-local algorithms increases the cost of the algorithm exponentially (see §3.5). Thus, for real applications in large domains these algorithms may only be capable of performing a very small number of additional steps, if any, regardless of what the optimum may be. This fact eventually constrains the application of quasi-local indices to large scale graphs (Liu and Lü, 2010; Lü and Zhou, 2011), and motivates the use of local scores since, as previously said, quasi-local indices reduce to local scores when the number of steps is one. Another limitation of quasi-local indices are domains with a low average distance among vertices, such as protein-protein interactions where the average distance is 2 (Cao et al., 2013). In these domains only local scores are relevant.

2.1.2 Local similarity-based algorithms

To determine if an edge between to given vertices shall exist, local similarity-based algorithms of LP only take into account those vertices directly connected with the pair of vertices composing the evaluated edge. The first detailed analysis and evaluation of these algorithms was presented in (Liben-Nowell and Kleinberg, 2007), where five local scores were compared among themselves and against four global scores on five different scientific co-authorship graphs in the field of physics. Although no method clearly outperformed the rest in all datasets, three methods consistently achieved the best results; local algorithms Adamic/Adar (AA) and Common Neighbours (CN), and global algorithm Katz (Katz, 1953). In (Murata and Moriyasu, 2008) a similar test was conducted for graphs obtained from social networks, and the same results were obtained with AA and CN achieving the best results among local algorithms.

A new local algorithm called Resource Allocation (RA) was proposed and compared with other local similarity-based algorithms in (Zhou et al., 2009). Testing on six different datasets showed once again that AA and CN provide the best results among local algorithms, but it also showed how RA was capable of improving them both. Next we describe these top three local similarity-based algorithms (CN, AA and RA) since these are the ones that have been proven to provide the best results so far. For this same reason these three algorithms will be used as base-line to evaluate the performance of our algorithm proposed in §4.1, when applied to the set of graphs described in §6.

The Common Neighbours (CN) algorithm (Newman, 2001) computes the similarity $s_{x,y}$ between two vertices x and y as the size of the intersection of their neighbours. Formally, let $\Gamma(x)$ be the set of neighbours of x

Definition 1

$$s^{CN}_{x,y} = |\Gamma(x) \cap \Gamma(y)|$$

The Adamic/Adar (AA) algorithm (Adamic and Adar, 2003) follows the same idea than the CN, but it also considers the *rareness* of edges. To do so, shared neighbours are weighted by their own degree and the score becomes

Definition 2

$$s^{AA}_{x,y} = \sum_{z \in \Gamma(x) \cap \Gamma(y)} \frac{1}{log(|\Gamma(z)|)}$$

The Resource Allocation (RA) algorithm (Zhou et al., 2009) is motivated by the resource allocation process of networks. In the algorithm's simpler implementation, each vertex is interpreted as a transmitter of a single resource unit, having this resource unit evenly distributed among its neighbours. In this case, the similarity between vertices x and y becomes the amount of resource obtained by y from x

Definition 3

$$s^{RA}_{x,y} = \sum_{z \in \Gamma(x) \cap \Gamma(y)} \frac{1}{|\Gamma(z)|}$$

2.1.3 Present and future of similarity scores

Research is currently focusing on quasi-local scores, as these seem to be the most promising kind of similarity-based LP algorithms. Global algorithms, in addition to their computational complexity issues, have not proven to be particularly effective (Liben-Nowell and Kleinberg, 2007; Liu and Lü, 2010; Lü et al., 2009). This disappointing lack of performance by global scores is probably caused by issues related with the inherent noise found in the data set; by considering the whole graph to evaluate the likelihood of a single edge, all irrelevant or nearly irrelevant parts of the graph will eventually block out the contributions of the small relevant parts, therefore affecting precision. On the other side of the spectrum, local scores employ a very limited amount of graph data: that which is directly related with the edge being evaluated. Precise and complex predictions seem hard to achieve with so scarce information.

It is our impression that quasi-local indices will eventually become the most relevant family of similarity-based LP algorithms. These scores are capable of considering a limited but variable in size part of the graph, balancing the trade-off between exactness and computational feasibility. Furthermore, these scores can include in-graph distance as a variable to weight the impact of data, extending the semantics that can be captured. However, before quasi-local indices can become fully functional for large scale graphs one must find a way of making them computationally feasible, reducing their cost and parallelizing them efficiently. As we will see in §3.5 this is not an easy task due to the computational cost added by each further step performed by quasi-local scores. Nevertheless one can expect a lot of research effort to be made in that area in the near future. Before that happens though, we believe local scores should remain at the focus of research. As said before, quasi-local indices eventually represent fine-drawn versions of local scores, expanding their reach and refining their predictions. Thus, defining more precise local scores which can be later transformed into quasi-local should be a priority of the LP field. In that regard, in §4.1 we propose a local score of LP, and evaluate it against other local scores in §7.1.3. All while keeping in mind its future potential transformation into a quasi-local index, which we already explore in §4.1.2.

Chapter 3

The Problem of Directed Link Prediction

The problem of LP can be outlined as *finding new relations within a relational data set*. Beyond this apparent simplicity LP includes features which may change the way the problem is interpreted. In here we discuss some of these features with the goal of properly defining the problem we try to solve, and also to defend the decisions we have made in proposing a solution.

One of the most shared views of the LP problem is that of a binary classification problem, motivated by decades of interpreting problems through the perspective of classic ML and DM tools. Indeed, LP can be reduced to a problem of determining which edges belong to a positive class (those that do exist) and which belong to a negative class (those that do not exist). However, in the context of huge, high-dimensional data sets this type of interpretation may be an oversimplification. We discuss that point in §3.1.

An early decision one must make when working on LP refers to the type of graph that will be mined. Graphs can be directed, weighted, acyclic, labeled *etc.*, and each of those features has a strong impact on how LP is performed. Most LP research has been applied to the simplest type of graph: *i.e.*, an undirected, unweighted and unlabeled graph. In this thesis we focus on directed, unweighted and unlabeled graphs instead. We include directions because these are very frequent in graphs, and directionality provides semantics which can be used for improving predictions (see §3.2). We do not include weights because these serve mostly as a refining tool rather than providing radically new information about the graph topology (see §3.3). Nevertheless we discuss the integration of weights into our approach in the future work chapter (see §10.4). We do not consider labels because these are an *ad hoc* property of graphs which requires one to focus on a particular domains, thus crippling the universal applicability of LP algorithms. Furthermore, the most appropriate solution for labeled graphs are heterogeneous solutions like the ones presented at the end of §2.1.1. Finally, even though it is frequent in the LP bibliography, we do not consider time grounded graphs (where edges are assumed to have an associated time of addition) for the lack of a purpose, as discussed in §3.4.

In the last two sections of this chapter we explore the complexity of the LP problem. First, in §3.5 we study the computational complexity of LP scores. Since one of the topics of this thesis is scalability we need to understand what is the complexity of LP algorithms, and based on which properties the complexity increases. By doing so we will be able to predict how can the performance of LP algorithms be improved without hampering its applicability. Then, in the last section §3.6, we discuss the presence of class imbalance in LP and how it may affect the performance of the predictions.

3.1 Link Prediction as a Binary Classification Problem

In essence, LP can be understood as a binary classification problem. Given a directed graph $G = (N, E)$, and all the possible edges in the graph (of size $|N * (N-1)|$), the problem LP is that of distinguishing between edges that exist, $e \in E$, and edges that do not exist, $e \notin E$.

The application of classic classification algorithms to this problem, such as decision trees, neural networks or support vector machines, seems inadequate for various reasons. To start with, the number of features available for each instance (which in this case would be each possible edge) is limited to topological features, a rather small and redundant source of information. Also, most classification algorithms (*e.g.*, the three previously mentioned) assume that the data sample is independent and identically distributed, whereas in a graph that is clearly not the case as vertices and edges directly define the topological features of one another. Finally, the computational cost of calculating the features of all possible instances ($|N * (N-1)|$), training an algorithm with them and then performing classification on every edge not found in the graph seems unfeasible.

Regardless of all these issues, (Lichtenwalter et al., 2010) proposes, implements and successfully tests an assemble of classification algorithms (C4.5, J48, Naïve Bayes) on a graph composed by over 5 million vertices. These authors identify a set of 12 topological features on directed weighted graphs, including the CN score (see §2.1.2 for its formal definition). To make the approach feasible, training for each edge is not performed considering the whole graph. Instead only those vertices found at a variable distance between 2 and 4 hops of the a given edge are considered to define its features. As a result, these authors perform a quasi-local training and obtain a quasi-local model of the data. Results indicate that their proposed algorithm (HPLP) improves those of similarity-based scores on both a directed and an undirected weighted graph. For doing so HPLP includes one similarity score (CN) as feature, implements an ensemble of classifiers through bagging and random forests, and performs undersampling for correcting the class imbalance. From these results we acknowledge that one can achieve good results in LP with traditional classification algorithms, by combining techniques obtained after decades of research. We consider however that the novel, relational approach of similarity-based LP scores is more coherent with the nature of graphs, and that it will demonstrate its supremacy on graph related tasks after some research effort has been put on it.

Even though LP can and has been reduced to a binary classification problem, it is our perception that this approach results in a dangerous oversimplification. Classification has but one purpose: to distinguish between classes with the highest possible precision and recall. LP on the other hand, has among its goals (or should have, from our perspective) to understand the behavior of connectivity, and to produce high certainty predictions. In large scale graphs the existence or absence of a particular edge is a rather uncertain and irrelevant issue, particularly when there are billions of possible edges and thousands of exceptions (*i.e.*, outliers). Hence, LP needs not to classify most edges correctly in order to be successful. In our interpretation LP needs only to correctly classify a significant set of edges[1] edges with high certainty to become useful. Instead of building two clearly delimited classes of edges, LP ought to focus on building

[1]What represents a set of significant size with regards to edges predicted depends only on the domain of application. We elaborate on that in §5.3.

one high-certainty class of edges for which its existence is well founded, and most importantly, understood. Therefore, LP algorithms ought to focus on building accurate scores which can approximate the edge likelihood and represent their semantics.

3.2 Directed and Undirected Link Prediction

The vast majority of contributions to the LP field focus on undirected graphs, with only a few works addressing the directed LP problem (Lichtenwalter et al., 2010; Mantrach et al., 2010; Zhang et al., 2013). And yet, most of the graphs used in LP and most of those that would be interesting to use, are or could be represented as directed graphs (*e.g.*, web graph, social networks, protein interactions, gene expressions, product recommendation, *etc.*). The motivation to focus on undirected graphs lies is the apparent simplicity of the problem; by considering directed graphs we must add edge directionality to the list of LP issues to solve (Lü and Zhou, 2011). The tendency towards undirected graphs is such that even methodologies clearly based on directed models (*e.g.*, hierarchies) are developed to work on undirected graphs (Clauset et al., 2008). Our perception however is that LP on directed graphs instead of being a more complex problem is a simpler one, thanks to the added semantics provided by directionality. By considering the direction of edges one can further characterize the nature of edges, and thus adapt the predictive methodologies to achieve higher precisions. We further discuss this issue when motivating our proposed directed LP score in §4.1, and present our conclusions in that regard in §9.1.

3.3 Weighted Link Prediction

As happens with edge directionality, the consideration of edge weights in LP is often over-viewed. One of the problems one finds when trying to incorporate weights into LP algorithms is the wide variety of interpretations weights allow. Weights may be natural numbers (*e.g.*, representing a frequency (Lichtenwalter et al., 2010; Murata and Moriyasu, 2007)) or they may be rational numbers within a range (*e.g.*, expressing a probability). Following the work of (Murata and Moriyasu, 2007), (Lü and Zhou, 2010) explored the impact of weights on LP by testing the top three local scores (CN, AA and RA) on three weighted graphs against a weighted version of themselves. To the authors' surprise, the weighted scores performed better only in one of the three tested graphs. Authors discuss the importance of edges with small weights, in accordance with weak ties theory, as an explanation for this behavior.

Our motivation to work with unweighted graphs is one of simplicity. We acknowledge the importance of weights in graphs, and even the necessity of using weights to represent complex domains. However, when the source of information is the graph topology, weights can not provide radically new information, as they do not change the pattern of relations within the graph. By adding weights to edges one can only refine the interpretation of those edges, but not expand them. Acknowledging the future importance of weights for LP, and particularly its impact on our approach, in §10.4 we discuss some preliminary thoughts on how to integrate them.

3.4 Time Grounded Link Prediction

Since many of the graphs used in LP are constantly growing (*e.g.*, social networks, citation networks, web graph), researchers have often considered the problem of LP as one of predicting edges through time (Aggarwal et al., 2013; Lichtenwalter et al., 2010; Lü and Zhou, 2011). From this perspective the problem is that of finding which edges will exist at time t_1 given the edges at time t_0, where t_0 represents the edges in the original graph. This perception of an evolving graph with a temporal grounding may be accurate and relevant to define some of the domains used (*e.g.*, co-authorship networks), but for others it may be clearly inadequate (*e.g.*, metabolic networks, protein-protein interactions).

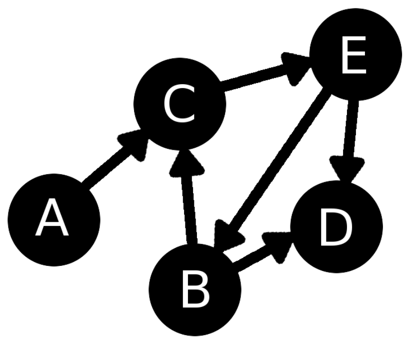
For those domains where time is a meaningful variable, taking time into account may be extremely valuable by providing rich information in the LP setting where information is scarce. For example, in the case of scientific co-authorship graphs, it seems clear that the least time it has past since two authors collaborated on authoring a paper, the more likely these two authors are to collaborate soon again. Unfortunately, the temporal properties that rule one graph may not be directly applicable to other time-based graphs (*e.g.*, the interest of users on products for recommendation may not evolve through time like scientific collaborations do), which would force one to produce a specific temporal analysis for each data set. This problem, together with the reduced number of time annotated graphs have limited the proposal of LP methods which consider edge temporality as a key feature, as even those authors that acknowledge time as a basic feature of LP do not fully integrate it in their methods (Aggarwal et al., 2013; Lichtenwalter et al., 2010; Lü and Zhou, 2011). Nevertheless, we consider this to be an interesting approach that will eventually gain some deserved attention.

3.5 Computational Complexity of Similarity-based Link Prediction

One of the focal issues of this thesis is the scalability of LP scores, a mandatory concept if one intends to process large graphs. To properly understand how do the various LP scores scale one must perform an analysis of their computational complexity, both in terms of space and cost. In this section we tackle both, starting with an analysis of the space complexity and ending with an analysis of the computational cost.

There are various ways of representing a graph, and each of them has its own space and access complexity. The most frequently used representations are *adjacency lists* and *adjacency matrix*. Adjacency lists store, for each vertex of the graph, a list of the vertices directly connected with it (*i.e.*, its neighbours). Adjacency matrix on the other hand store all possible edges in the graph, annotating those that do exist. See Figure 3.1 for a graphical representation of both representations. The adjacency matrix is more efficient than the adjacency list in access operations: while the adjacency matrix allows for direct access (*e.g.*, $get[x][y]$), the adjacency list needs to perform a search process (*e.g.*, $list[x].search(y)$). From the perspective of spatial cost it is the other way around, as the adjacency list is more efficient than the adjacency matrix: while the adjacency matrix cost is $O(N^2)$ for a graph with N vertices, the adjacency list has a cost of only $O(E)$ for a graph with E edges. Clearly, the choice between both representations depends on the type of problem one is trying to solve. In our case, a space complexity of $O(N^2)$ for graphs with millions of vertices is unfeasible due to the limitation of resources (*i.e.*, RAM memory). We therefore settle for the space efficient solution, and use the adjacency list representation.

Before evaluating the computational cost let us first summarize the problem. LP evaluation, as defined in §5.1 through ROC and PR curves, requires one to calculate the score of all possible edges in the graph. This process can be significantly optimized in the context of local scores, as the score of many edges can be directly evaluated



$$\begin{array}{c|ccccc} & A & B & C & D & E \\ \hline A & 0 & 0 & 1 & 0 & 0 \\ B & 0 & 0 & 1 & 1 & 0 \\ C & 0 & 0 & 0 & 0 & 1 \\ D & 0 & 0 & 0 & 0 & 0 \\ E & 0 & 1 & 0 & 1 & 0 \end{array}$$

A → C
B → C D
C → E
D →
E → B D

Figure 3.1: Adjacency matrix representation of a graph (right, top), and adjacency list representation (right, bottom).

as zero. Since local scores require of common neighbours between a pair of vertices to evaluate their edge, one needs only to evaluate those pairs of vertices which are at maximum distance of two hops, as the rest of edges in the graph will certainly have a null similarity (*i.e.*, no common neighbours). In this context we next evaluate the computational cost of the worst case scenario $O(x)$ (when all edges must be evaluated) and the computational cost of a more realistic scenario $o(x)$ (when only a portion of all edges must be evaluated).

The time complexity of accessing all edges in a directed graph of N vertices is $O(N^2)$, as in a directed graph there are $N * (N-1)$ possible edges. The cost of calculating a single local similarity score is $O(N)$ in the worse case, as only those vertices directly related with the edge under evaluation must be accessed (with a maximum of $2(N-1)$ vertices per edge evaluated). The time complexity of calculating all possible edges in the worse case of local similarity-based algorithms is therefore $O(N^3)$. This complexity can be reduced if one dismisses edges as previously considered. The previous complexity analysis ($O(N^3)$) is accurate for the worse case scenario, when the graph is complete. When the graph has on average k neighbours per vertex, the full time complexity of local scores is reduced to $o(N * k^2)$.

Quasi-local similarity-based scores have an increased cost w.r.t. local scores, as these scores explore a deeper section of the graph. If the exact depth of exploration (*i.e.*, the number of steps performed by quasi-local scores) is known beforehand, one can reduce the computational cost in an analogous fashion as with local scores: Evaluate only the edges between vertices found at a distance equal to the number of steps of the score, and dismiss the rest. Formally, the cost of computing the quasi-local similarity score of single edge in a graph with N vertices is $O(N^s)$ in the worse case, where s is the number of steps taken, $s \geq 2$ as $s = 1$ makes quasi-local algorithms analogous to local algorithms. The time complexity of calculating all possible edges in the worse case of quasi-local similarity-based algorithms is therefore $O(N^{s+2})$ where $s \geq 2$. This complexity can be reduced if one knows the number of steps s before hand. In this case, when the graph has on average k neighbours per vertex, the full time complexity of quasi-local scores performing s steps is reduced to $o(N * k^{s+2})$

Finally, global similarity-based algorithms cannot benefit from any cost reducing measure, as these scores need to fully traverse the graph. The cost of these algorithms is therefore always of the form $O(N^x)$ where x depends on the particularities of each global score. If one has resources enough as to store the graph in adjacency matrix form (which is currently unfeasible for large graphs) the cost may be significantly contained. The global score Katz index (Katz, 1953) for example can be calculated through the matrix inversion operator (Lü et al., 2009), making its full computational cost $O(N^3)$ for graphs stored in adjacency matrix form. Nevertheless, even in this optimum scenario $O(N^3) \gg o(N * k^2)$.

3.6 Class Imbalance in Link Prediction

As discussed in §3.1, LP can be reduced to a classification problem. To do so one needs only to categorize edges of the graph in two classes depending on whether they exist in the graph (the positive class) or not (the negative class). Unfortunately, graphs used for LP are typically sparse, which results in a large imbalance: the negative class is very large in comparison to the positive class. As it is well known, class imbalance can be a (severely) complicating factor in classification problems (Chawla et al., 2002; Liu et al., 2009; Wasikowski and Chen, 2010; Weiss, 2004).

Data source	**Number of vertices**	**Edges per vertex**	**positive:negative class ratio**
Wordnet	89,178	7.83	1:11,382
Cyc	116,835	2.95	1:39,496
IMDb	2,930,634	2.56	1:1,140,835
webND	325,729	4.59	1:70,867
webSB	685,230	11.09	1:61,775
webGL	875,713	5.82	1:150,217
hudong	1,984,484	7.49	1:264,848
baidu	2,141,300	8.31	1:257,667
DBpedia	17,170,894	9.70	1:2,151,672

Table 3.1: Average number of edges per vertex and class imbalance of the graphs used for evaluation

Let us start by analyzing the class imbalance found in LP on large graphs from a formal point of view. All the graphs we use have a number of edges per vertex constant with respect to the number of vertices: regardless of how many vertices we add to the graph, the average number of edges per vertex does not vary significantly. Having a number of edges per vertex constant with respect to the number of vertices implies that the positive class (*i.e.*, the number of edges in the graph) grows linearly with the number of vertices ($N * k$). At the same time, the negative class (*i.e.*, the number of possible edges in the graph) grows quadratically with the number of vertices ($N * (N - 1)$). Consequently, class imbalance in LP grows linearly with the number of vertices. The graphs we present in §6 and use in §7 ratify this analysis (see Table 3.1): regardless of the number of vertices in the graph, all graphs have on average less than 10 edges per vertex, while class imbalance grows (more or less) linearly.

The magnitude of the class imbalance found in LP is such that similar examples are rarely found in the bibliography. Consider for example a directed graph with N vertices and $N * 10$ edges. The negative class of such a graph would be composed by $N * (N - 1)$ edges [1], minus those which do exist ($N * 10$). For these estimates, a graph composed by 100,000 vertices has 1 million edges in its positive class, and *9,999 million edges* in its negative class. Hence, in this scenario we would have a 1:10,000 *positive:negative* ratio

[1]For undirected graphs the number of edges is half of that, but it still grows quadratically.

of imbalance. As discussed before, the problem aggravates as graphs grow in number of vertices, making this our best case scenario, as shown by Table 3.1.

In (Japkowicz and Stephen, 2002) the impact of class imbalance on the performance of classifiers was explored, and authors concluded that this impact is largely reduced when all classes are of reasonable size. *Apriori* this should be good news for LP on large graphs, as its classes seem to be of *reasonable* size; the smallest class we have is our tests is composed by 34,559 edges (10% of edges of the Cyc data set, see Table 6.1), a large class from the perspective of classification. Unfortunately, even though according to (Japkowicz and Stephen, 2002) having large small class should neutralize the effect of class imbalance, that is not the case in LP (Lichtenwalter et al., 2010). The reason for this is twofold. On one hand the imbalance found in LP on large graphs is several orders of magnitude larger than any imbalance tested in (Japkowicz and Stephen, 2002). Thus its impact may remain significant. On the other hand, as discussed in §3.1 LP is not a typical classification problem, and given the small amount of information provided by each edge, a class composed 30,000 elements could still be considered to be small.

Informally, class imbalances of 1:10,000 or larger (1:2,151,672 in our worst case) make of the LP problem one of a needle in a haystack. This situation translates as a tendency towards false positive classification mistakes, as incorrectly accepting as positive elements from the negative class is almost inevitable. The size of the negative class (typically composed by billions of edges) does not help either. With so may negative edges it is very likely that there are thousands of non-existing edges with identical properties. Hence, a false positive classification mistake in most cases entails thousands of actual misclassifications. As a result, the great challenge of LP is precision, as we will see in our tests (see §7). In that regard we discuss how LP performance can be fairly evaluated in a setting of large class imbalance in §5.1, and propose an evaluation methodology focusing on precision in §5.3. We also present a brief study on the correlation between LP performance and class imbalance in §9.5.

Chapter 4

Hierarchies and Directed Graphs

So far we have presented the current solutions for LP as well as the challenges one must face when tackling this problem. In this context we introduce the main contribution of this thesis: a new score of LP for directed graphs. Nevertheless, the design of this score was not the result of a premeditated analysis of LP. In fact, the outline of our proposed score was previous to the consideration of its application to the LP problem. Our original thoughts were on the importance of hierarchical information for knowledge representation, according to the following idea: *what generalizes you and what specializes you defines most of what you are.* As a simple example of that idea consider *how much you know about cats just by knowing about felines and about a cat you own.* The LP score here presented is but a practical application of this idea.

Our motivating hypothesis was that semantics of an entity could be informally approximated by combining the semantics of its generalizations and specializations. To test the validity of such hypothesis we required a domain in which knowledge was implemented through directed relations (asymmetric relations are needed by hierarchies), and where there were lots of relations (a requirement of informal, abstract learning). If we could correctly predict new relations in such a domain by applying hierarchical assumptions, we could argue on the validity of our hypothesis.

The practical aspects of our considerations eventually drove us towards the LP problem on large, directed graphs, which nowadays happens to be a hot topic of research. The LP problem perfectly fits our hypothesis requirements; it contains lots of relational, directed knowledge (edges within a graph), and some of its knowledge discovery algorithms are based on a measure of semantic similarity obtained from the exploration of nearby entities (similarity-based scores). Hence, we develop a LP score based on the assumption that an entity (in this context, a vertex) is partly defined by its generalizations (in this context, the vertices it points to) and its specializations (in this context, the vertices that point to it). The limitations of these assumptions are discussed in §9.3.

4.1 The INF score

To introduce our proposed LP score we must first define two different sets of edges found in any directed graph. Given a directed graph $G = (N, E)$, and a given vertex $x \in N$, we name the vertices that x points to ($x \rightarrow$) as the *ancestors* of x ($A(x)$), and the vertices that point to x ($\rightarrow x$) as the *descendants* of x ($D(x)$). Formally

Definition 4

$$\forall x, y \in N : y \in A(x) \leftrightarrow x \rightarrow y \in E$$

$$\forall x, y \in N : y \in D(x) \leftrightarrow y \rightarrow x \in E$$

Given these two sets we assume directionality defines a weak hierarchy, weak in the sense that inheritance is not directly transitive. Instead inheritance is assumed to be weighted (*e.g.*, the result of an average), such that, to determine if a given vertex x must be related to a given vertex y, we take a look at how many of x generalizations ($A(x)$) are related with y. Formally

Definition 5 *Given a directed graph $G = (N, E)$, and a pair of vertices $x, y \in N$, the likelihood of the relation $x \rightarrow y$ given $A(x)$ is:*

$$\frac{|AY(x)|}{|A(x)|}$$

where $AY(x)$ is a subset of $A(x)$, containing the ancestors of x having the relation $\rightarrow y$:

$$\forall a_x \in A(x) : a_x \in AY(x) \leftrightarrow a_x \rightarrow y \in E$$

The previous definition states how information is propagated top-down. That is how the generalizations of a vertex define its relations. We define the analogous process with information propagating in the opposite direction, bottom-up. In this case we define how the relations of a vertex are obtained as a weighted average of the relations of its descendants.

Definition 6 *Given a directed graph $G = (N, E)$, and a pair of vertices $x, y \in N$, the likelihood of the relation $x \rightarrow y$ given $D(x)$ is:*

$$\frac{|DY(x)|}{|D(x)|}$$

where $DY(x)$ is a subset of $D(x)$, containing the descendants of x having the relation $\rightarrow y$:

$$\forall d_x \in D(x) : d_x \in DY(x) \leftrightarrow d_x \rightarrow y \in E$$

The combination of Definitions 5 and 6 can be understood as a formalization of our initial hypothesis: *what generalizes and what specializes you defines most of what you are.* Which in the context of directed graphs has become: *the likelihood of relation $(x \rightarrow y)$ is estimated from the existence of relation $(\rightarrow y)$ in the $A(x)$ and $D(x)$ sets.* Still, this is a rather obvious statement from the point of view of LP, as the union of the $A(x)$ and $D(x)$ sets contains all neighbors of x. To go further on our assumptions let us consider Definitions 5 and 6 separately, and their resemblance to inferential reasoning.

Definition 5 follows a top-down reasoning approach, similar to that of weighted deductive inference: *if my four grandparents are mortal, I will probably be mortal too.* In this case new information $(me \rightarrow mortal)$ regarding a vertex (me) is obtained from the ancestors of the vertex $(me \rightarrow grandparent)$, and weighted based on the amount of times the relation is satisfied ($grandparent \rightarrow mortal$, four out of four). Hence, the more of ones grandparents are dead $(|A(me) \cap D(mortal)|)$, the more certain one can be about his/her own mortality. See Figure 4.1 for a graphical representation of the process. We define a LP similarity score between two vertices based on this idea, called the *deductive* metric (DED for short)

Definition 7

$$s^{DED}_{x \rightarrow y} = \frac{|A(x) \cap D(y)|}{|A(x)|}$$

Definition 6 follows a bottom-up reasoning approach, in this case similar to that of a weighted inductive inference: *if most publications from an author are meticulous, the author will most likely be a meticulous person.* In this case new information

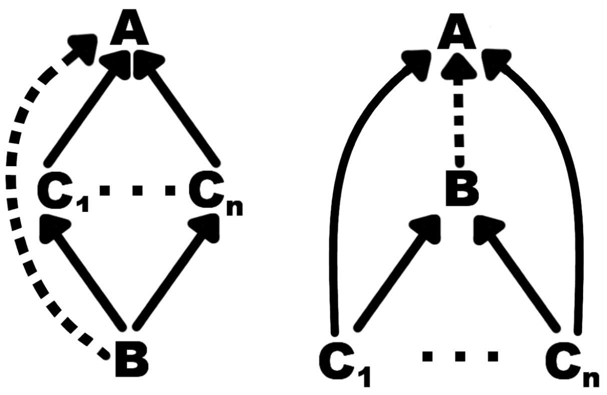


Figure 4.1: On the left: graphic representation of the top-down deductive process for estimating edge likelihood according to DED. On the right: graphic representation of the bottom-up inductive process for estimating edge likelihood according to IND.

($author \rightarrow meticulous$) regarding a vertex ($author$) is obtained from the descendants of the vertex ($publication \rightarrow author$), and weighted based on the amount of times the relation is satisfied ($publication \rightarrow meticulous$). Hence, the more frequently our publications are meticulous ($|D(author) \cap D(meticulous)|$), the more certain we can be about that property applying to ourselves. See Figure 4.1 for a graphical representation of the process. We define a LP similarity score based on this idea called the *inductive* metric (IND for short)

Definition 8

$$s^{IND}_{x \rightarrow y} = \frac{|D(x) \cap D(y)|}{|D(x)|}$$

Since we consider both generalizations and specializations key for defining an entity, we combine both DED and IND into a single score. By doing so we are aggregating the evidence provided by both the ancestors and the descendants of a vertex in order to determine the likelihood with which edges *originating* from that vertex exist. The result is a hierarchical affinity measure for each possible pair of vertices in the graph, and the basic LP score for directed graphs that we propose. We call this score the *inference* score (INF for short).

Definition 9

$$s^{INF}_{x \rightarrow y} = s^{DED}_{x \rightarrow y} + s^{IND}_{x \rightarrow y}$$

INF is a local score of LP (introduced in §2.1.2) according to Definitions 4, 7, 8 and 9. Since sets $A(x)$ and $D(x)$ are defined as local in Definition 4, *i.e.*, only those elements directly connected with x may belong to $A(x)$ or $D(x)$, INF considers only information local to the vertices x and y to determine the likelihood of edge $x \rightarrow y$. This feature guarantees that the computational cost of INF will be that of local scores (see §3.5).

4.1.1 Modifying INF

INF consideration of directed edges, and at a higher level, of hierarchies, makes it more expressive than other local scores such as RA, AA and CN. This added complexity allows us to introduce modifications on INF, adapting the score to properties frequently found on hierarchical domains. In this section we describe two modifications we implemented on the INF score. These tuned versions of INF are evaluated in §7.1.4. By understanding, first the motivation behind these modifications, and second their predictive performance, we can learn about the particularities of the mined domains. This will allow us to increase the algorithm precision and its chances of applicability, as we will see in §7.2.

The first modification we propose is based on how the DED and IND scores are combined in INF. According to Definition 9, the evidence provided by DED and IND is considered equally (INF = DED + IND). In preliminary tests of this thesis we noticed that the DED score typically achieves higher precisions than IND. This is coherent for graphs including edge transitivity, as DED computes a sort of weighted inheritance. On the other hand, the IND bottom-up inductive process is more volatile as specializations are typically less reliable than generalizations. Nevertheless, the combination of DED and IND outperforms DED alone, as it is capable of detecting a wider variety of patterns for LP (some edges will have a DED score of zero, but a high IND score). We therefore decided to define a version of INF where the DED score is given twice the weight of the IND score, but where IND is still taken into account. We name it *INF_2D*

Definition 10

$$s^{INF\_2D}_{x \rightarrow y} = s^{DED}_{x \rightarrow y} * 2 + s^{IND}_{x \rightarrow y}$$

The second modification we propose has the goal of prioritizing those predictions having a larger set of evidence in their favor. According to Definition 7 and 8, the likelihood of an edge is obtained from the *proportion* of satisfying evidence (how many of all my generalizations/specializations have that edge). However, a proportion may not be the best measure of significance in certain domains. In informal domains (*e.g.*, the ones presented in §6.2) the *absolute amount* of evidence can be a decisive factor of reliability. Consider for example a vertex x having only one generalization and one specialization. If both point to vertex y, the INF score evaluates edge $x \rightarrow y$ as 2 (DED = 1, IND = 1). However, for a vertex z which has 100 generalizations and 100 specializations, where 99 of each of them point to vertex y, the INF score evaluates edge $z \rightarrow y$ as 1.98 (DED = 0.99, IND = 0.99). In many domains edge $z \rightarrow y$ seems more reliable than edge $x \rightarrow y$ because of the absolute amount of evidence (198 vs 2), even though it has less proportional evidence (99% vs 100%). To include the notion of absolute evidence we defined a modification of INF in which the amount of satisfying ancestors and descendants, not only their proportion, is taken into account. Formally, we redefine DED and IND

Definition 11

$$s^{DED\_LOG}_{x \rightarrow y} = \frac{|A(x) \cap D(y)|}{|A(x)|} * log(|A(x)|)$$

Definition 12

$$s^{IND\_LOG}_{x \rightarrow y} = \frac{|D(x) \cap D(y)|}{|D(x)|} * log(|D(x)|)$$

Which results in the following definition of the *INF_LOG* score

Definition 13

$$s^{INF\_LOG}_{x \rightarrow y} = s^{DED\_LOG}_{x \rightarrow y} + s^{IND\_LOG}_{x \rightarrow y}$$

Both modifications address different issues, and can be combined. The score resultant of this integration, INF_LOG_2D, is defined next, and will be tested as well in §7.

Definition 14

$$s^{INF\_LOG\_2D}_{x \rightarrow y} = s^{DED\_LOG}_{x \rightarrow y} * 2 + s^{IND\_LOG}_{x \rightarrow y}$$

4.1.2 Quasi-Local INF

In §2.1.1 we introduced quasi-local indices, those considering a variable section of the graph (by increasing the maximum path length explored) for estimating edges. In that same section we saw how quasi-local indices reduce to local scores when that maximum path length is 1. Then, in §2.1.3 we presented the reasons why we foresee these indices will gain relevance in the near future. In this section we address a necessary question at this point: how to turn INF, which was defined as a local score, into a quasi-local score.

The approach we follow is similar to that found in the bibliography. Instead of considering the neighbours shared at distance 1 by vertices x and y in order to determine the likelihood of $x \to y$, we increase that distance. As frequently done in quasi-local scores (Lü et al., 2009), we apply a decaying factor to the information provided at a longer distance, so that importance decreases as path length increases (*i.e.*, my friends define me more accurately than the friends of my friends). For computational reasons we limit the quasi-local version of INF to two-step paths, but notice this can be easily extended to any length. To provide a formalization of the quasi-local INF we start by defining the set $D^2(x)$ as those vertices descendants of the descendants of x which are not direct descendants of x, and $A^2(x)$ as those vertices ancestors of the ancestors of x which are not direct ancestors of x

Definition 15

$$\forall x, y \in N : y \in A^2(x) \leftrightarrow y \in \bigcup_{\forall z \in A(x)} A(z) \wedge y \notin A(x)$$

$$\forall x, y \in N : y \in D^2(x) \leftrightarrow y \in \bigcup_{\forall z \in D(x)} D(z) \wedge y \notin D(x)$$

We next define the two-steps, quasi-local version of deductive and inductive scores. Here we combine the evidence provided at step 1 (local, right part of the addition) and at step 2 (quasi-local, left part of the addition)

Definition 16

$$s_{x \to y}^{DED^2} = \frac{|A^2(x) \cap D^2(y)|}{|A^2(x)| * \alpha} + s_{x \to y}^{DED}$$

$$s_{x \to y}^{IND^2} = \frac{|D^2(x) \cap D^2(y)|}{|D^2(x)| * \alpha} + s_{x \to y}^{IND}$$

The parameter α is the decaying factor, based on which the relevance of evidence found at step 2 is decreased with respect to the evidence found at step 1. In our evaluations (see §7.1.5) we have set $\alpha = 2$. Finally, we define the two-step quasi-local version of INF, the INF_QL score

Definition 17

$$s_{x\to y}^{INF\_QL} = s_{x\to y}^{DED^2} + s_{x\to y}^{IND^2}$$

We briefly discuss the performance of $s_{x\to y}^{INF\_QL}$ in §7.1.5, and consider future work in this regard in §10.3.

4.2 Hierarchical Undirected Scores

The most reliable local similarity-based scores currently available in the bibliography, presented in §2.1.2, were designed for undirected link prediction. For directed graphs, as the ones presented in §6, these algorithms cannot characterize separately symmetrical pairs of edges. In other words, given two vertices x and y these scores evaluate equally edges $x \to y$ and $y \to x$. To improve the performance of these algorithms in certain contexts, some authors have considered the possibility of extending them, for example by adding weights obtained from the re-occurrence of edges (Murata and Moriyasu, 2008). In a similar fashion, in this section we propose an adapted version of local undirected similarity-based algorithms for directed graphs.

To turn undirected scores of LP into directed scores we focus on one of the most easily observable properties of a hierarchy: the *specificity level* of an element. By analyzing the position of a given element within a hierarchy (*e.g.*, how many abstract elements it has above it, how many specific elements below it) one can estimate its degree of specificity within the context of the hierarchy. As hierarchies are typically tree-shaped[1], specific elements low in the hierarchy are more likely to be the origin of edges, while abstract elements high in the hierarchy are more likely to be the destination of edges. This property of hierarchies can be supported both statistically and argumentatively. In detail, we compute the specificity level of a vertex in a graph as

[1]In this context *tree-shaped* does not refer to the mathematical definition of tree structure. It refers to the general shape of having a wide bottom and a thin top, which for example may still include cycles.

the number of distinct vertices from which it can be reached, regardless of the path distance. A sort of recursive descendants set size.

Our proposed hierarchical version of the top three undirected link prediction algorithms CN, AA and RA is rather strict in that it only considers relations going from a given vertex to those which are more abstract than itself. When that is the case it computes the similarity of an edge normally, as CN, AA and RA do. Otherwise, when the edge goes from a more general vertex to a more specific one, our hierarchical version of CN, AA and RA evaluates the similarity as 0 (*i.e.*, dismisses the edge). Formally, the Hierarchical Common Neighbours (HCN) score is

Definition 18

$$s^{HCN}_{x\to y} = \begin{cases} s^{CN}_{x,y} & \textit{if } x \textit{ is more or equally specific than } y \\ 0 & \textit{if } x \textit{ is more abstract than } y \end{cases}$$

We introduce the Hierarchical Adamic/Adar (HAA) score in a similar manner

Definition 19

$$s^{HAA}_{x\to y} = \begin{cases} s^{AA}_{x,y} & \textit{if } x \textit{ is more or equally specific than } y \\ 0 & \textit{if } x \textit{ is more abstract than } y \end{cases}$$

And finally, the Hierarchical Resource Allocation (HRA) score

Definition 20

$$s^{HRA}_{x\to y} = \begin{cases} s^{RA}_{x,y} & \textit{if } x \textit{ is more or equally specific than } y \\ 0 & \textit{if } x \textit{ is more abstract than } y \end{cases}$$

To evaluate the impact of including the concept of specificity level into LP scores, we compare HCN, HAA and HRA against the original CN, AA and RA in §7.1.1, and against our own proposed score INF in §7.1.3. From these results we can argue about the importance of hierarchical properties for predicting edges on directed graphs (see §9.4).

4.3 Abductive Score of LP

Two algorithms inspired by inferential reasoning processes are combined in the definition of the INF score. One resembling a deductive process where knowledge is propagated top-down (DED), and one resembling an inductive process, where knowledge is propagated bottom-up (IND). For supporting a given edge prediction, this dual approach uses higher-level data and lower-level data separately but collaboratively. Consider the analogous process of deciding whether a specific cat will be friendly to us, by reviewing both how friendly animals with claws are (deductively, top-down) and how friendly previous cats we have met were (inductively, bottom-up).

A third inferential reasoning process is often identified besides deduction and induction: *abductive reasoning*. Abduction can be defined as the process of deriving explanations which justify previously known facts (Larrosa and Cortés, 1995). For example, to explain why cats are found in houses, and knowing that animals found in houses are in most cases pets, one could abduce that cats are most likely pets. In Figure 4.2, where a graphic graph-based representation of abductive reasoning is shown, this example would be mapped as $A : animals\_in\_houses$, $B : pets$ and $C : cats$.

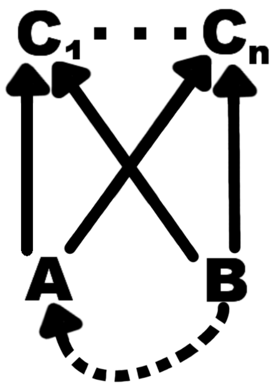


Figure 4.2: Graphic representation of the side-to-side abductive process for estimating edge likelihood.

The analogous LP score ABD, short for abduction, can be seen in Definition 21

Definition 21

$$s^{ABD}_{x \to y} = \frac{|A(x) \cap A(y)|}{|A(x)|}$$

The possibility of including a score based on the idea of abductive reasoning into INF, alongside DED and IND, was contemplated and tested. Results showed that the ABD score was capable of positively evaluating edges not found by either DED or IND. As a result, an INF score including the ABD achieves higher recalls. Unfortunately, the ABD score also has significantly lower precision that DED and IND, thus decreasing the overall precision of the INF score if included. The explanation we find for this is the volatility of abductive reasoning: abductive reasoning must rely on a lot of evidence in order to be made with certainty, making it much more vulnerable to the inconsistencies and sparsity found in most graphs.

Since in this thesis we focus on increasing the precision of link predictions (*e.g.*, paying special attention to high certainty predictions, see §5.3), we decided to leave ABD outside the definition of INF. Nevertheless, we consider ABD to represent a different and relevant type of LP score, and we intend to find a way to successfully integrate it into INF in the near future.

4.4 Chapter Summary

In this chapter we have defined a novel local similarity-based score of LP based on hierarchical inference, called INF. The purpose of INF is to exploit the properties of hierarchies for relational learning, both from a top-down deductive and a bottom-up inductive perspective. INF assumes that edge directionality includes a sense of abstraction/specialization, not necessarily in a formal manner. INF can therefore be applied to any directed graph. The added expressiveness of INF in comparison with other local scores allowed us to define two modifications in it, one increasing the weight of deductive evidence over inductive evidence (2D), and one considering the absolute amount of evidence in combination with the local proportion of evidence (LOG). INF was also transformed into a quasi-local index, empowering the future development of a whole new family of quasi-local algorithms of LP. Finally, details on a third type of hierarchical inference based on abductive reasoning were presented, as well as a formalization so that the derived sub-score ABD can be integrated into future LP scores.

Chapter 5

Evaluating Link Prediction

After introducing the current best local scores of LP and our own proposed score, we want to compare and evaluate them. In this chapter we discuss how to do so, identifying some of the problems one finds when trying to faithfully evaluate LP scores. We start by discussing class imbalance, an almost inherent feature of LP (see §3.6) which significantly affects how scores are evaluated. In §5.1 we present the solutions found in the past for this problem, and how do these relate with LP. Then in §5.2 the state-of-the-art in LP evaluation is discussed. In §5.3 we introduce another contribution of this thesis, an evaluation methodology designed to focus on the potential usability of the LP results. Moving away from imbalance and evaluation scores, in §5.4 we discuss how do we obtain the test sets needed to perform an empirical analysis on the performance of scores, and what impact may that approach have on the overall results. Finally, in §5.5 we discuss the benefits and alternatives of current LP performance analysis.

5.1 Evaluation under Class Imbalance

As we saw in §3.6 the LP problem must face a particularly large class imbalance. Class imbalance is key in classification problems as it implies difficulties at predicting the small class. A small class that is in most cases the main target of the predictive process. Consequently there is a large and growing state-of-the-art on how to deal with class imbalance, typically by trying to build a model well trained for the small class.

A frequent approach of supervised or semi-supervised learning methods to overcome class imbalance is to equilibrate the training set through over-sampling (adding new entities to the small class), under-sampling (removing entities from the large class) or feature selection (Chawla et al., 2002; Liu et al., 2009; Wasikowski and Chen, 2010; Weiss, 2004). Regardless, local similarity-based scores of LP are unsupervised and cannot benefit from these solutions: adding or removing edges from the data set would equal to perform classification outside the algorithm, and there are no features to be removed beyond the existence of edges among vertices. In that regard we focus on those aspects of class imbalance that are relevant for unsupervised methods: deciding which metrics to use when evaluating and comparing the performance of classifiers for data sets with a large class imbalance.

The most frequently used metrics to evaluate the performance of classifiers are based on accuracy. However, these metrics are biased towards the classification of instances within the large classes (*e.g.*, it is relatively easy to determine which edges do not exist for certain), making them inappropriate for imbalanced data sets (He and Garcia, 2009; Liu et al., 2009; Weiss, 2004). Using them for LP would be almost analogous to measuring the capability of algorithms at predicting which edges should not be added to the graph, which is not the goal of LP.

For data sets with large class imbalance, the most frequently used evaluation metric is the Receiver Operating Characteristic (ROC) curve and the derived Area Under the Curve (AUC) measure (Fawcett, 2004). The ROC curve sets the True Positive Rate (TPR, number of positive edges predicted as positive given the total number of positive edges) against the False Positive Rate (FPR, number of negative edges predicted as positive given the total number of negative edges), making this metric unbiased towards entities of any class regardless of their size. The AUC measures the area below the curve in order to compare the overall predictive performance of two different curves.

ROC curves are unbiased in imbalanced contexts, but their consideration of miss-classifications can result in mistakenly optimistic interpretations (Davis and Goadrich, 2006; Yang et al., 2014). When the negative class is very large, showing mistakes as relative to the negative class size (FPR) can hide their actual relevance, and make it complicated to assess the overall performance quality. For someone who intends to find an actual application to classifiers tested in a hugely imbalanced data set (such as us), most of the ROC curve is irrelevant as it represents completely unacceptable performance. For example, one may consider that a classifier achieving a TPR of 0.95 (finding 95% of all positive edges) and a FPR of 0.01 (incorrectly accepting 1% of all negative edges) in the ROC curve demonstrates an excellent performance. However, for a data set with a positive:negative rate of 1:100 those results imply that the classifier accepts more negative edges than positive edges (*i.e.*, it has a precision smaller than 0.5). For domains with a 1:11,000 or worse positive:negative ratio, like the ones we work with (see §3.6), the limitations of the ROC curve become even more striking. In those even a FPR of 0.0001 implies a very poor precision/performance regardless of the TPR achieved.

An alternative evaluation measure to ROC curves are precision-recall (PR) curves. This metric is also resistant to class imbalances as it focuses only on the performance achieved for the positive class (typically the small one), and does not show the number of correct classifications for the negative class. In fact, the ROC and PR curves are strongly related; a curve dominates another (it is above another) in the ROC space if and only if it also dominates it in PR space (Davis and Goadrich, 2006). The main difference between ROC and PR curves is on how errors are represented. While ROC curves show miss-classifications as relative to the total number of negative cases, PR curves show miss-classifications as relative to the total number of predictions done. In detail, PR curves plot precision on the y axis and recall on the x axis, making it more sensitive to mistakes and focused on high precision predictions.

As an illustration on the differences between ROC and PR consider how these two curves represent a random classifier, which always performs poorly in an imbalanced data set. The ROC curve always represents the random classifier as a straight line between points $(0, 0)$ and $(1, 1)$, regardless of class imbalance, with all better than random classifiers represented as lines above that diagonal. PR curves on the other hand represent random classifiers in imbalanced data sets a flat line on the x axis, as their precision in imbalanced settings is always close to zero. We therefore agree with recent research (Yang et al., 2014), and consider the PR curve to be a more appropriate evaluation methodology for LP. It will be the one used in our tests §7. Nevertheless, we have also included the ROC curves of all tests performed in Appendix §A for illustrative purposes.

5.2 Precision-Recall Curves in Link Prediction

Most research on LP use ROC curves(Clauset et al., 2008; Fire et al., 2011; Lü et al., 2009; Lü and Zhou, 2011; Murata and Moriyasu, 2008) or PR curves (Aggarwal et al., 2013; Nickel et al., 2011) for evaluation. This is motivated, as previously discussed, by the huge class imbalance found in LP tasks and the resistance of these evaluation methods to this setting. For the reasons discussed in §5.1 though, we find PR curves to be more appropriate. For a given data set, the PR curve shows the performance of a classifier at various thresholds: at the left part of the curve are the high-certainty predictions where precision is higher, while at the right part of the curve are the low-certainty predictions where recall grows at the expense of a lower precision. Through the PR curve one can directly see which classifier performs better at each different predictive threshold. The derived AUC metric of the PR curve on the other hand determines which classifier performs better overall, when all thresholds are considered at the same time with the same importance. In that regard, when evaluating LP scores in a data set, the AUC of the PR curve is a good indicator of which score performs the best overall.

In practice, we find the PR-AUC score to be sub-optimal for evaluating the applicability of results. Given the imbalance of the graphs used (see §3.6), in most cases a large part of the PR curve will represent low precisions, as high precisions can only be achieved for a small set of high confidence predictions. In fact, as recall grows and the acceptance threshold decreases, precision can quickly reach levels unacceptable from a practical point of view. At this point one must consider which results are worth taking into account when evaluating the performance of a classifier. If we intend to achieve an applicable methodology we should focus on its performance where it matters, when a *reasonable* number of mistakes are being done. At extremely low precisions (*e.g.*, 0.01%) results are likely to be useless, and therefore should not be taken into account (at least not with equal weight) into the evaluation. For this reason in §5.3 we propose a AUC measure focusing on applicability.

5.3 Constrained AUC

Is a prediction where, of 1,000 edges predicted 1 is right and 999 wrong, useful? The answer to that question ultimately depends on what is to be done with those predictions, but in general such an imprecise prediction will hardly be exploitable: there are just too many mistakes. One of the goals of this thesis is to get LP as close as possible to broad applicability. For that purpose we consider the necessity of evaluating performance in the context of usability, so that we can determine which LP score produces the most usable results. The classic AUC measure considers equally the performance of a classifier at any threshold, therefore evaluating which classifier performs the best overall. However, we consider that the performance of LP scores at low thresholds, where millions of mistakes are done, is irrelevant for assessment purposes.

We propose to evaluate LP scores based only on the predictions these produce while keeping an acceptable ratio of mistakes. In order to be fair with the particularities of each domain, we define this *acceptable* ratio of mistakes based on the number of edges originally found in the graph. The more edges there are in a graph, the more predictions we expect to obtain, therefore extending the limit of mistakes considered as *acceptable*; we assume that one will rarely want to predict more new edges than edges already found in the graph. Indeed, generating a larger amount of knowledge than one currently has is rarely sought in AI, due to the unreliability of such an approach. Formally, the proposed Constrained AUC score (CAUC) is obtained by calculating the AUC of the PR sub-curve where the number of non-existing edges mistakenly accepted by the score is equal or lower than the total number of edges in the graph.

The CAUC considers only the part of the PR curve where the score is incorrectly accepting less edges than the number of edges in the graph, dismissing the rest of the area under the curve. CAUC therefore calculates a portion of the AUC starting from the left of the curve, ending when too many mistakes are done. Nevertheless, if a LP score is precise enough the CAUC can be equal to the AUC. Notice that, unlike the AUC, the CAUC can be 0, if the top E edges evaluated by a score in a graph with E edges are incorrect predictions.

As said before, the goal of this performance measure is to focus on the relevant parts of the PR curve. By accepting only mistakes up to the number of edges in the graph one puts a limit to what one considers are potentially useful predictions. Thus, scores which perform better at low precisions are penalized by this score, whereas they are not by the AUC. Furthermore, by making the threshold dependent on graph properties (*i.e.*, on the number of edges in it) the CAUC score adapts to domain specific properties such as sparsity and graph size. This is a interesting novel feature not found in the AUC

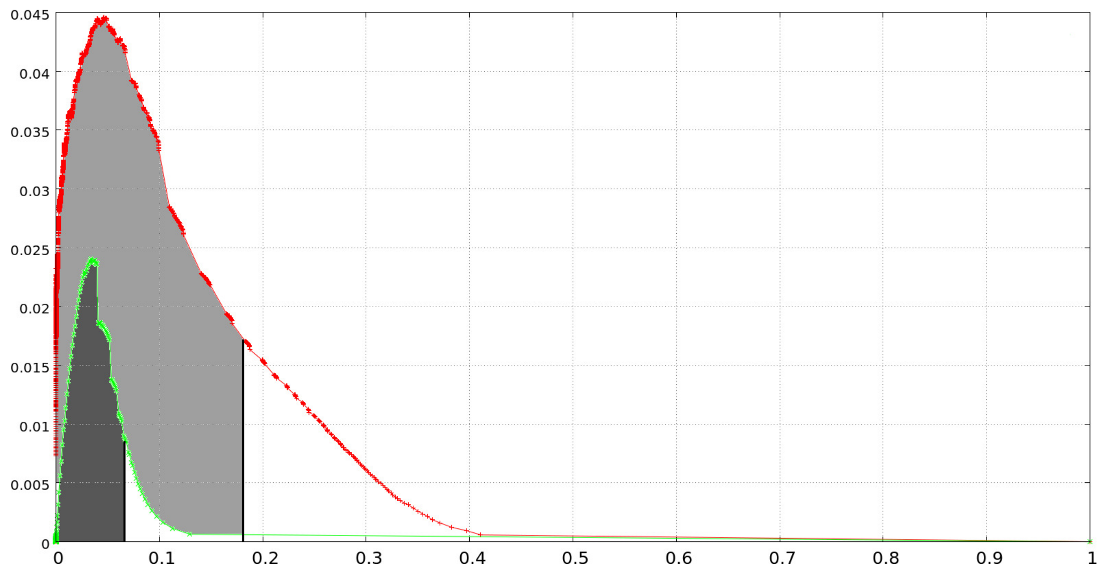


Figure 5.1: PR curve of RA on two different graphs (Cyc and IMDb). Grey area shows the CAUC.

measure, as a contextual evaluation allows the cross-domain comparison of results. As an example consider Figure 5.1, where the PR curves of the RA score are shown for two different graphs. The vertical cut on each curve represents the location of the CAUC threshold for each particular data set and score, limiting the CAUC to the area at the left of the threshold (colored in grey), whereas the AUC considers the whole curve.

5.4 Building Test Sets

To evaluate a predictor empirically we require a test set. In the case of LP the edges proposed by a predictor are compared against those of the test set. Each predicted edge within the test set is considered as a correct prediction, and each predicted edge not found in the test set is considered a mistake. The resultant precision and recalls obtained build the previously discussed PR curve.

The main problem with tests sets is how to obtain them. The best test set one can use is one which represents a natural extension of the graph being tested. Such will be the approach taken with the DBpedia graph, introduced in §6.2.1, in which the original graph is composed by the pagelinks among wikipages of 2012, and the test set by the pagelinks added one year later, in 2013. Unfortunately, this setting where we have two clearly incremental parts of the graph is rare, and we will not be able to apply it to the rest of the graph tested. Instead, as will happen in most cases, we must settle for the more drastic approach of randomly removing a number of edges from the graph in order to use them as test set.

For all graphs introduced in §6 we will remove a random 10% of all edges. We will then try to predict this 10% test set using as a source the graph composed by the remaining 90% of the graph. The problem with this approach is that the topology of the graph may be significantly affected by the removal of a random 10% of edges, therefore hindering the performance of the evaluated scores. Hence, one can expect the actual performance of LP scores to be slightly better than the one achieved here when applied to an integral graph. Regardless of its limitations, this methodology remains the most reliable one (Yang et al., 2014).

5.5 Representativity of Test Sets

The use of PR or ROC curves to evaluate LP implies the assumption that the sets of edges being used for test (the prediction of which is evaluated by the curves) are a significant representation of *all* the edges missing in the graph. Or in other words, that all edges not found in neither the graph nor in the test set, are wrong. In certain cases, where the graph topology is stable, this may be an accurate assessment. For example, the Wordnet graph (introduced in §6.1.1) can be considered as almost perfect, as WordNet relations have been identified, discussed and implemented by linguists for years. In other cases though the test cases are an imperfect measure of which edges are missing from the graph. Consider for example the DBpedia graph, introduced in §6.2.1, in which the pagelinks among Wikipedia articles from 2012 are used as training and the new pagelinks added on 2013 are used as test. This graph is clearly incomplete, even if we consider the links of 2012 plus those of 2013. The Wikipedia grows every day and the fact that a link is not implemented so far does not mean it is wrong. As a result, one must take into account that some of the edges predicted, not found in the test set and labeled as mistakes, will in fact be correct predictions corresponding to edges not yet added to the graph.

Since these limitations in the evaluation of LP scores through test sets affect all scores, it can be argued that the PR curve remains a valid methodology for comparing the relative performance of two or more scores. The shortcoming of test set representativity comes when evaluating the actual true precision of a score in the context of applicability. Unfortunately, there is no solution so far to that problem beyond an impossible hand-made validation. An approximate solution in that direction is to perform a sampling process of all edges predicted, manually evaluating the sampled edges as correct or incorrect predictions, and then extrapolating the performance obtained on the sample to the rest of the graph. There are several important aspects to keep in mind with this solution. First of all, for the extrapolation to be faithful, the sampling needs to be large, which equals to many hours of manual labeling. And second, since the performance of LP scores decrease as the similarity of edges decreases (*i.e.*, high certainty prediction are much more accurate than low certainty ones), the sampling would have to be done at several thresholds so that extrapolations are representative of the whole curve. Sampling is therefore relevant for accurately estimating the predictive performance of a given score on a specific domain. Since we lack the necessary manpower and we do not target one specific domain, we will not perform sampling in our evaluation.

Chapter 6

Graph Domains

LP scores are learning tools completely agnostic from the nature of data being processed: these only requires entities and relations composing a graph to perform learning, regardless of the semantics of those entities and relations. Thanks to this high level of abstraction, LP can be applied more or less successfully to virtually any domain, as one can always find a way to transform knowledge representations into a relational set (*i.e.*, a graph). In here we introduce the data sets we use in §7 for comparing the performance of different LP scores. We first introduce two graphs which are explicitly hierarchical in §6.1, and then several non hierarchical graphs in §6.2. Our goal is to show both the relevance of hierarchical features in graphs (regardless if those graphs are hierarchical or not) and the potential applicability of LP in general to a variety of domains. The basic properties of the graphs used can be seen in Table 6.1.

6.1 Explicitly Hierarchical Graphs

Hierarchies are frequently found in relational sets through properties such as inheritance and transitivity. When these properties are implemented, they may be responsible for the existence of most relations within the data set. In this section we introduce two data sets which explicitly implement hierarchical features: one obtained from the WordNet lexical database and one from the OpenCyc ontology.

Data set source	**Number of vertices**	**Number of edges (Input + Test)**
WordNet	89,178	698,588
Cyc	116,835	345,599
IMDb	2,930,634	7,528,349
webND	325,729	1,497,134
webSB	685,230	7,600,595
webGL	875,713	5,105,039
hudong	1,984,484	14,869,484
baidu	2,141,300	17,794,839
DBpedia	17,170,894	137,028,000

Table 6.1: Size of graphs used for evaluation

6.1.1 WordNet

WordNet (Miller, 1995) is a lexical database originally designed for the English language, now available for several languages. It contains words with associated lexical information, such as the word type (*i.e.*, noun, verb, adjective, *etc.*). Words in WordNet are grouped according to their semantics and compose *synsets*, groups of synonym words. Accordingly, words with more than one meaning belong to more than one synset. Synsets are related one to another through semantic relations such as hypernym/hyponym, meronym/holonym, *etc.*. The English WordNet knowledge base can be downloaded from the project's web page[1], and has been used for various tasks of natural language processing, such as word sense disambiguation (Banerjee and Pedersen, 2002) and machine translation (Knight and Luk, 1994).

To build a graph from WordNet we focus on the *hyponym/hypernym* relations of synsets. The hyponym/hypernym relation associates concepts based on the inclusion of semantics: A is a hyponym of B (*i.e.*, B is a hypernym of A) if the semantics of A are within the semantics of B. For example, *cat* is a hyponym of *feline*, and *artifact* is a hypernym of *pencil*. Hence, this relation defines a type of hierarchy based on semantic specialization: the more specific the semantics of a word, the lower it will be found in this hierarchy, and the more hypernyms and less hyponyms it will have. Based on this *is-a* relation we build a graph by considering each synset as a vertex,

[1]wordnet.princeton.edu

and each *hyponym/hypernym* relation as a directed edge (from hyponym to hypernym). The resultant graph is composed by 89,178 vertices and 698,588 edges. WordNet hyponym/hypernym relations are transitive, and we implement such transitivity in the graph. This means that every synset is directly connected with all its hyponyms and hypernyms. Since WordNet has been formally defined by linguists it does not contain cycles, making the graph we build a directed tree. This will be the only tested graph satisfying this property.

6.1.2 Cyc

The Cyc project was started in 1984, by D. Lenat, with the goal of enabling AI applications with human-like common sense reasoning (Lenat, 1995). In its almost thirty years of existence the Cyc project has developed a large knowledge base containing those facts that define common sense according to its creators. The project uses a declarative language based on first-order logic (FOL) called CycL to formally represent knowledge. Opencyc is a reduced version of Cyc containing most of the same concepts, taxonomic relationships, well-formedness information about predicates and natural language strings associated with each term. In 2012 a version of OpenCyc was released which included the entire Cyc ontology in OWL, implementing RDF vocabulary. It can be downloaded from the project's web page[1].

We build a semi-formal hierarchy from the OWL ontology of OpenCyc by extracting the `rdfs:Class` elements. These will become the vertices of our graph. To implement the edges of the graph we use the RDF relations `rdf:type` and `rdfs:subClassOf`, as these define a type of hierarchy. The Cyc knowledge base includes transitivity for the relation `rdfs:subClassOf`, but not for `rdf:type` (RDF_Working_Group, 2014). Since edges labels are not used in the LP process, edges predicted in this graph will be of a merged `rdf:type` and `rdfs:subClassOf` kind, among `rdfs:Class` elements. Significantly, these two types of relations, `rdf:type` and `rdfs:subClassOf`, account for over the 80% of all relations among `rdfs:Class` entities in the OpenCyc ontology. The resultant graph is directed and unlabeled (we delete the labels of edges), composed by 116,835 vertices and 345,599 edges. This graph is also explicitly hierarchical, first because it is partly transitive, and second because of the specialization/generalization semantics included in both types of relation implemented.

[1] www.cyc.com

6.2 Non-hierarchical Graphs

As discussed in §4 one of our hypothesis is that hierarchical semantics are found at some level in most general knowledge representations which include directionality. Even if no explicit hierarchical properties are defined (*e.g.*, inheritance, transitivity) in a graph, the essence of edge directionality typically includes features which can be exploited in the context of hierarchies (*e.g.*, causality). To evaluate that hypothesis we test our proposed score, which assumes generalization/specialization relations, on some graphs which are not explicitly hierarchical. If a hierarchy is found at some implicit level in them is something that we will discuss in §9.4.

6.2.1 Webgraphs

The World Wide Web naturally represents a large, directed *webgraph*: a graph composed by web pages connected through hyperlinks. When considering LP for the WWW, the first application that comes into mind is to find new hyperlinks among web pages. Through similarity-based methods, this process implements a proximity or similarity analysis among web pages, from which one can easily identify multiple fields of application. For example, one could use hyperlink prediction for increasing the connectivity, navigability and visibility of web sites, or in a more subtle approach, for improving the quality of search engine recommendations given similarity-based analysis. Regardless of the particular application, the capability of computing larger graphs becomes a key factor when trying to generalize them to the large variety of webgraphs available today.

The task of predicting hyperlinks among web pages using similarity-based algorithms has received little attention so far. In contrast, other similar problems like the one of predicting relations in social network graphs have been thoroughly studied using a wide variety of methodologies (Adamic and Adar, 2003; Liben-Nowell and Kleinberg, 2007; Lichtenwalter et al., 2010; Murata and Moriyasu, 2007; Song et al., 2009). We find in the bibliography that webgraphs have in fact been occasionally used to evaluate LP scores in combination with other domains, but never as an independent case of study. Furthermore, when webgraphs have been used for LP, only relatively small graphs have been computed (a few thousand vertices), typically due to the complexity of the models being used (Lü et al., 2009; Tong et al., 2007).

To evaluate the web hyperlink prediction problem we use six large webgraphs obtained from different sources. Three of them are the webgraph of the University of Notre Dame domain in 1999 (webND) (Albert et al., 1999), the webgraph of Standford and Stanford-Berkley in 2002 (webSB) (Khalil and Liu, 2004) and a webgraph provided by Google for its 2002 programming contest (webGL). The other three evaluated webgraphs originate from online encyclopedias. They still represent web pages and the hyperlinks among them, but they have in particular that those web pages correspond to encyclopedic articles. The three webgraphs were obtained from the Chinese encyclopedias Baidu and Hudong (Kunegis, 2013) and the English version of Wikipedia (Lehmann et al., 2014). The size of each webgraph used is shown in Table 6.1.

6.2.1.1 Hierarchies in the WWW

One of the main contributions of this work is the novel application of a hierarchical score (defined in §4.1) for the problem of LP. By applying INF to webgraphs we are implicitly assuming that the WWW topology contains or is partly defined by hierarchical properties. The relation between the WWW and hierarchies has in fact been discussed, empirically evaluated and exploited in the past, as we review in this section. What is novel in our contribution is its application to the specific case of hyperlink prediction, something that has never been done before. As we will see, for this use case we consider hierarchies at a much smaller scale than usual, thus enabling hierarchical properties not so far exploited in the context of the WWW.

Various aspects of the WWW have been shown to include hierarchical properties in the past. The autonomous systems topology of routing units (Pastor-Satorras et al., 2001), and the WWW requests and traffic (Crovella and Bestavros, 1997) were among the first WWW aspects shown to be hierarchically structured. The webgraph defined by web pages and hyperlinks was first found to compose a self-similar structure defined by a set of nested entities, indicating a *"natural hierarchical characterization of the structure of the web"* (Dill et al., 2002). This idea was extended and formalized in (Ravasz and Barabási, 2003), where authors showed how some key properties found in webgraphs and other real networks, such as being scale-free and having a high degree of clustering, could be satisfied through a hierarchical organization. A challenge at which previous models had failed. In their work, Ravasz and Barbási propose a hierarchical network model fitting such networks, to be composed by small, dense groups, recursively combining to compose ever larger and less dense groups, in a fractal-like structure.

The importance of hierarchies to model the structural properties of real networks was finally exploited through its application to generative models: models built to produce artificial, large scale networks mimicking the topological structure and properties of real networks. The work of Leskovec *et al.*(Leskovec et al., 2005) is of particular interest, as authors define various generative models satisfying certain complex properties never properly captured before (*e.g.*, densification, shrinking diameters and a heavy-tailed out-degree). The most complex model proposed by Leskovec *et al.*, called the forest fire model (FFM), builds a network by iteratively adding one new vertex x to a graph. In this model, one first randomly provides an entry point to x in the form of a vertex x will point to (*i.e.*, its ambassador) through a first, random out-going edge. In a second step, a random number of out-going edges are added to x, to be chosen among the out-going and in-going edges of the ambassador (the former with a higher probability of being selected than the latter). This process in then repeated recursively for each new vertex that gets connected with x, with the restriction that each ambassador is considered only once. The FFM and INF share some relevant features, which we further discuss in §9.4.

6.2.2 IMDb

IMDb is one of the largest and most popular online databases of audiovisual entertainment information. It contains lots of related data about movies, TV shows, actors, directors, movie genres, user ratings *etc.*. For building a graph with this information we first crawled the website and obtained all the available titles (movies or TV shows), directors, genres (*e.g.*, Western, Comedy) and tags (*e.g.*, female-protagonist, miami-florida, car-chase). As a result we obtained a list of 2,930,634 vertices. Their types and frequency can be seen in Table 6.2. For those, we extract the following relations found in IMDb:

- *Movie* references *Movie* (520,768 occurrences)
- *Movie* directed by *Director* (1,839,296 occurrences)
- *Movie* belongs to *Genre* (1,289,500 occurrences)

- *Movie* defined by *Tag* (3,878,785 occurrences)

Directionality of edges is defined as shown in the previous list ($e.g.$, $Movie \rightarrow Director$). Interpreting this graph as a hierarchy means movies are some type of specialization of the movies they reference, of the directors that directed them, of the genre they belong to and of the tags that define them. The result is a directed graph with 2,930,634 vertices and 7,528,349 edges. We randomly split those edges 90%-10% into two sets, using the 90% of edges as initial graph, and using the remaining 10% as test set.

Vertex type	**Number of vertices**
Title	2,476,797
Director	321,485
Genre	31
Tag	132,321

Table 6.2: Type and frequency of IMDb vertices

Chapter 7

Empirical Study

In this chapter we compare the performance of the methods presented in §2 and §4 when applied to the nine graphs introduced in §6. To use the evaluation methods discussed in §5 we require a set of edges known to be correct but missing from the graph. This will be the test set, the target of the predictors. As discussed in §5.4, for eight of our graphs we randomly remove a 10% of the original edges from the graph to build the test set. Thus, for all graphs but the DBpedia we evaluate $(N*(N-1))-(E*0.9)$ edges[1], where N is the number of vertices and E the number of edges. The actual number of edges being evaluated for each graph are shown in Table 7.1. The evaluation on the DBpedia webgraph on the other hand could be incrementally performed through time thanks to its periodic data dumps. In detail, our tests consider all articles and hyperlinks added to the Wikipedia by June 2012 as train graph, and all new hyperlinks added by June 2013 as our test graph. Thus, for the DBpedia webgraph we are evaluating the scores ability at predicting hyperlinks that will be added from one year to another. Further details are shown in Table 7.1.

The main benefit of using the family of LP methods inhere proposed is *scalability*. A necessary feature considering the total number of edges to be evaluated. Due to the scalability of our approach we are capable of calculating the score of all possible edges to define the PR curve, avoiding approximate and potentially harmful methodologies such as test sampling. Test sampling reduces the computational complexity of the evaluation process by computing only a subset of all possible edges. Unfortunately

[1]Not perfectly accurate as vertices becoming disconnected in the train graph after removing the test edges were not computed nor considered for score evaluation. See Table 7.1 for the exact amount.

Graph	#edges to be found	#edges evaluated
WordNet	69,858	7,951 million
Cyc	34,559	13,649 million
IMDb	752,834	8,588,605 million
webND	133,279	95,939 million
webSB	756,937	466,034 million
webGL	494,982	741,978 million
hudong	1,446,760	3,786,991 million
baidu	1,701,330	4,370,982 million
DBpedia	2,865,540	6,009,812 million

Table 7.1: Total edges evaluated by graph.

there is no methodology for choosing that subset which guarantees results will not be biased (Yang et al., 2014). Only by evaluating all possible edges we can guarantee a faithful analysis of the actual predictive performance of the computed scores.

Even though our methods are the most scalable ones, the number of edges to be evaluated sets some computational boundaries. In our executions we define a running time limit of 48 hours. We could completely evaluate all the graphs here used within that limit, with the only exception of the DBpedia. The size of this particular graph (which has around $3 \cdot 10^{14}$ possible edges) forced us to simplify the problem in order to compute it within the limit (computational times are shown in Table 8.1). Instead of doing a typical test sampling we decided to focus on the prediction of edges originating from the 350,000 vertices with higher out-degree ($350,000 \cdot (N-1) - E^*$ edges)[1]. These are the richer edges in terms of information, and thus the most expensive ones to compute. As we will discuss in §7.2, this problem reduction method still results in a slightly biased comparison.

As seen in §2.1.1 the complexity of similarity-based link prediction algorithms depends on the diameter of the graph these algorithms explore. Global algorithms are very costly and cannot compute large scale graphs. And quasi-local indices, besides being more expensive than local algorithms, typically need the sampling of parameters such as optimal diameter and decay which increases their cost. In our evaluation we

[1]Where E^* corresponds to the edges in the training graph originating from the 350,000 vertices with higher out-degree.

focus on local similarity-based link prediction algorithms, since we consider INFerence to be our most significant contribution. However, we still want to test some quasi-local (INF_QL, INF_2D_QL, INF_LOG_QL, INF_LOG_2D_QL) and global (HRA, HCN and HAA) scores in order to better understand the overall behavior of INF and to validate our hypothesis regarding the importance of hierarchies.

We start by evaluating all thirteen scores (seven local, three quasi-local and three global) on three non-webgraphs, WordNet, Cyc and IMDb, in §7.1, showing the PR-CAUC achieved by each score on each graph. Then we focus on the particular problem of hyperlink prediction (*i.e.*, LP on webgraphs) through local scores in §7.2, evaluating seven local scores on six webgraphs using the PR-AUC measure. Additional PR-AUC, PR-CAUC measures and PR and ROC curves for all graphs are shown in Appendix §A for reference. Finally in §7.3 we discuss the edges predicted with higher accuracy, in hopes that this informal study will illustrate the type of predictions obtained, and the impact of the results shown here. All tests here presented were executed on a single node of the MareNostrum supercomputer, based on Intel SandyBridge processors, iDataPlex compute racks and an Infiniband interconnection. Further details are found in §8.4.

7.1 General Evaluation

We start the evaluation by comparing the performance of the top three undirected scores of LP (CN, AA and RA), defined in §2.1.2. Then we compare the global hierarchical versions of those scores (HCN, HAA and HRA), proposed in §4.2. Afterwards, we evaluate the unmodified INF score against the best undirected score and the best hierarchical version of an undirected score according to our tests. Then we study the performance of the various modifications on the INF score (INF_2D, INF_LOG and INF_LOG_2D), as proposed in §4.1.1. Finally, we evaluate the quasi-local version of the INF score, defined in §4.1.2.

7.1.1 Undirected Scores

There is a consensus in the bibliography on which scores of LP obtain the best predictive results (see §2.1.2). In accordance with this consensus, in this section we compare the top three scores, formally described in Definitions 1, 2 and 3, on the WordNet, Cyc and IMDb graphs. The PR-CAUC values achieved by CN, AA and RA for this problems

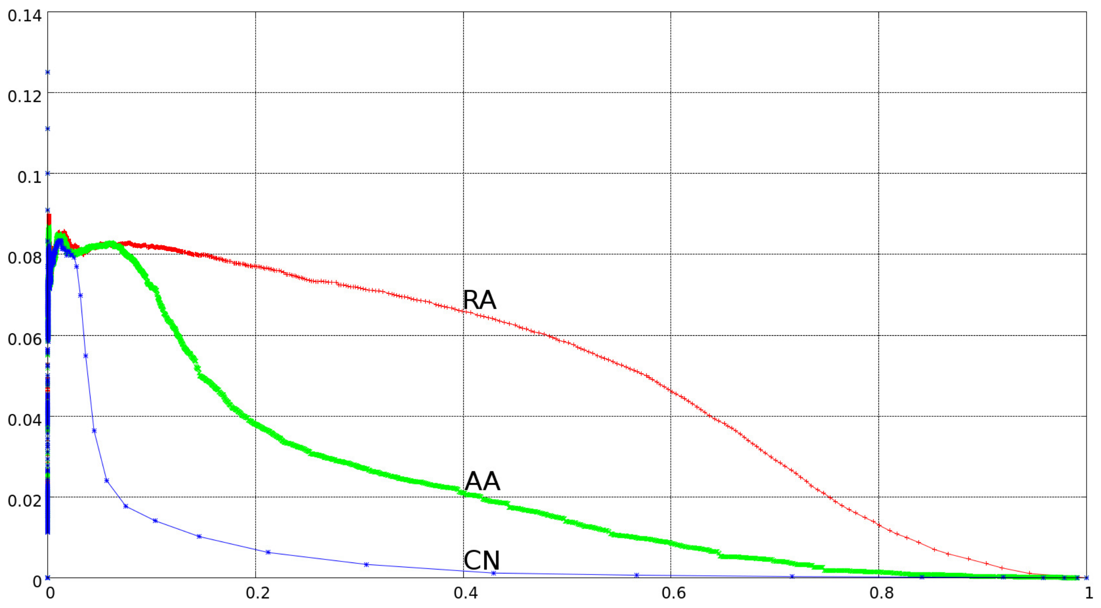


Figure 7.1: PR curve of RA, AA and CN scores on the WordNet graph.

can be seen in Table 7.2. The exact PR and ROC curves, as well as the PR-AUC score for all graphs, can be seen in Appendix §A.

Results indicate that RA outperforms AA and CN by a large margin in all three graphs. Regardless of the graph being hierarchical, non-hierarchical, large, small, dense or sparse, RA seems to be the most appropriate LP score. The performance of undirected scores among graphs seems to be strongly correlated with the size and sparsity of the graph being tested (shown in Table 3.1). Results are for all scores better in WordNet, the graph with the smallest number of vertices (89,178), and the smallest class imbalance (1:11,382). As the number of vertices and the imbalance increases (WordNet→Cyc→IMDb), results get worse. These results indicate the possibility of estimating the performance of predictors on a given graph given its properties.

Score	**WordNet PR-CAUC**	**Cyc PR-CAUC**	**IMDb PR-CAUC**
RA	<u>0.0383679</u>	<u>0.00584383</u>	<u>0.0011261</u>
AA	0.0156469	0.0048808	0.0008811
CN	0.0048877	0.0013585	0.0006182

Table 7.2: PR-CAUC score of RA, AA and CN on all graphs.

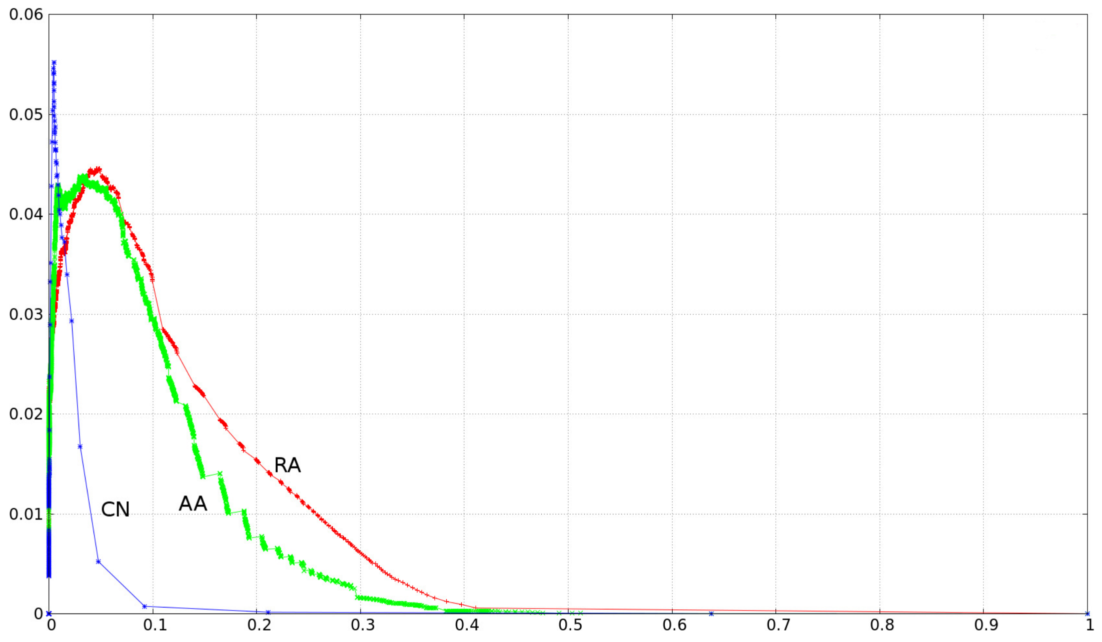


Figure 7.2: PR curve of RA, AA and CN scores on the OpenCyc graph.

In Figure 7.1 we show the PR curve for the WordNet graph, where undirected scores achieve the best results. Notice how the best precision obtained is around 9%. RA obtains approximately the same maximum precision than AA and CN. The difference between their performance is at the range of low thresholds, where RA is capable of maintaining a high precision (between 9% and 4%) as recall increases, while precision decreases more rapidly for the other scores.

Similar results are obtained for the OpenCyc graph, where RA obtains better CAUC than AA (by 19%) and CN (by 330%) by keeping a higher precision as recall grows (see Figure 7.2.) Notice however that CN is the best score for high precision predictions, as it reaches a maximum precision of 5.5% when recall is around 1%. RA on the other hand only reaches a maximum precision of 4.3% at 5% recall.

7.1.2 Hierarchical Undirected Scores

In this section we compare the hierarchical version of the three scores previously tested (HRA, HAA and HCN), as defined in §4.2. The PR-CAUC obtained by these scores on three of the proposed graphs can be seen in Table 7.3. In consistence with the results obtained for the undirected version of these scores, HRA outperforms HAA and

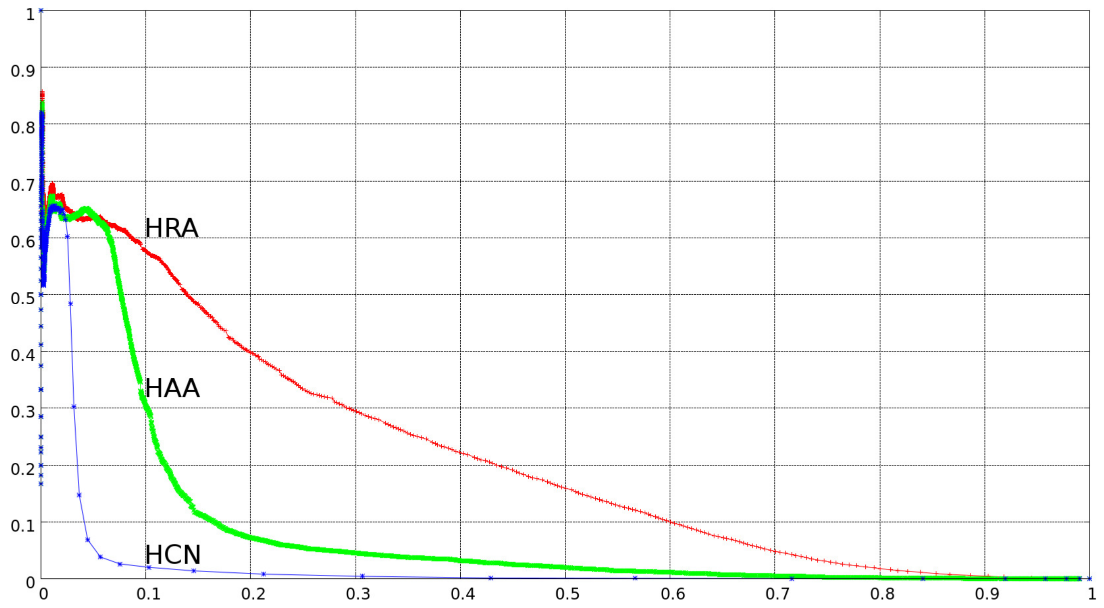


Figure 7.3: PR curve of HRA, HAA and HCN scores on the WordNet graph.

HCN by a large margin in all graphs. The PR curve of HRA, HAA and HCN for the WordNet graph can be seen in Figure 7.3.

Once again all scores achieve similar results at high certainty predictions, and HRA outperforms the others by keeping a higher precision as recall increases. In fact, the curves shown in Figures 7.1 and 7.3 are quite similar in shape. Not in scale though; directed algorithms HRA, HAA and HCN outperform their original undirected version by one order of magnitude in CAUC. Where undirected scores reached maximum precisions around 9% on WordNet, the hierarchical version of these scores achieve precisions up to 65%, a huge improvement. This increase in performance is also found in the other graphs, as shown by Tables 7.2 and 7.3 and Figures 7.3, A.13 and A.16. HRA, HAA and HCN combined outperform RA, AA and CN combined on the three

Score	**WordNet PR-CAUC**	**Cyc PR-CAUC**	**IMDb PR-CAUC**
HRA	<u>0.208804</u>	<u>0.0178101</u>	<u>0.00249108</u>
HAA	0.0789657	0.0143919	0.00205791
HCN	0.0234393	0.0034154	0.00149409

Table 7.3: PR-CAUC score of HRA, HAA and HCN on all graphs.

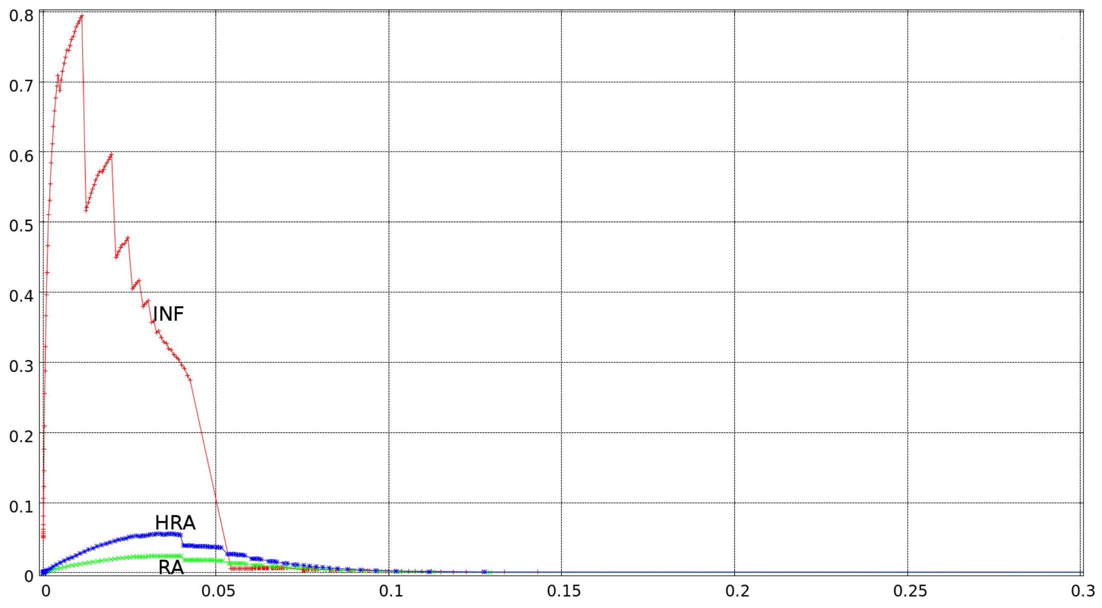


Figure 7.4: PR curve of INF, HRA and RA scores on the IMDb graph.

graphs by a 479%. We discuss the consequences of these results in §9.4, but for now let us remark how our proposed hierarchical versions represent a very effective way of converting undirected scores of LP to directed scores, at least for the graphs tested, achieving a significant performance improvement while doing so.

7.1.3 INF Score Evaluation

In this section we compare the performance of the INF score, proposed in §4.1, with that of the best undirected and hierarchical undirected scores according to the previous tests (*i.e.*, RA and HRA). In Table 7.4 the PR-CAUC scores of INF, HRA and RA are shown for the three graphs. Overall, INF is capable of significantly outperforming both HRA and RA, achieving for example 9 times higher CAUC than HRA and 20 times higher than RA in the IMDb graph. In the particular comparison between INF and RA, INF outperforms RA by one order of magnitude in all graphs: INF obtains better results on all graphs tested, regardless of the graph being explicitly hierarchical or not. This indicates that INF is a competitive score for various types of directed graph, along various domains. As a more detailed example, the PR curves of INF, HRA and RA in the IMDb graph are shown in Figure 7.4.

Score	WordNet PR-CAUC	Cyc PR-CAUC	IMDb PR-CAUC
INF	<u>0.715</u>	<u>0.0685397</u>	<u>0.0227005</u>
HRA	0.208804	0.0178101	0.0024910
RA	0.038367	0.0058438	0.0011261

Table 7.4: PR-CAUC score of INF, HRA, and RA on the three non webgraphs.

Even though INF obtains the best results on all three graphs, its performance on the WordNet graph is particularly good. INF does not make a single prediction mistake on WordNet until reaching a recall of 15.1% (see Figure §A.19), thus keeping a precision of 100% up to that point. As a result, of the total 69,858 edges to be predicted, 10,548 edges are correctly identified without incorrectly accepting any of the 7,951 million non-existing edges. We expected the best results to be achieved on WordNet, as it composes the most strictly hierarchical and dense of the graph tested. Additionally, the DED score can be understood as a weighted transitivity measure, and WordNet is strongly defined through transitivity. Nevertheless, the quality of the results is remarkable when considering the complexity of the task.

7.1.4 Modified INF Scores

After comparing the basic version of INF against the best candidates of LP in the previous section, next we focus on the modifications of INF as defined in §4.1.1. We test the 2D modification (which doubles the weight of DED in INF in comparison with IND, Definition 10), the LOG modification (which besides the proportion of edges supporting the inference it also considers their absolute number, Definition 13) and finally, the combination of both, INF_LOG_2D. The results on all three graphs can be seen in Table 7.5.

These results are more variable, as each modification performs very differently depending on the graph. We interpret this as an opportunity for *score adaptation*. The 2D modification works well on WordNet as INF_2D reaches a remarkable 0.837 PR-CAUC in it, a 17.16% higher than INF, and also in IMDb in combination with the LOG modification. On the other hand, on Cyc the 2D modification does not improve performance, indicating that Cyc is in fact the most taxonomically structured graph where generalization and specialization are equally reliable. For the remaining graphs, also

Score	**WordNet PR-CAUC**	**Cyc PR-CAUC**	**IMDb PR-CAUC**
INF	0.715	<u>0.0685397</u>	0.0227005
INF_2D	0.837709	0.0669549	0.0152854
INF_LOG	0.730075	0.0145715	0.0256254
INF_LOG_2D	<u>0.839372</u>	0.0144879	<u>0.0320782</u>

Table 7.5: PR-CAUC score of INF, INF_2D, INF_LOG, INF_LOG_2D on all graphs.

the webgraphs shown in §7.2, the 2D results indicate that knowledge obtained from generalizations (*i.e.*, top-down) is on average more reliable than that obtained from specializations (*i.e.*, bottom-up). These results are nothing new for transitive graphs, but is a novel and interesting property of informal graphs like IMDb and webgraphs.

The *LOG* modification of the INF score considers the amount of evidence as an important predictive factor, prioritizing those predictions that have more edges supporting them. This implies counterproductive results on graphs where a single edge is formally as much evidence as a set of them. Cyc is an example of that since its structure is ontologically defined: INF_LOG achieves a 78.7% lower PR-CAUC than INF on it. In WordNet, the structure is less formally constrained which allows INF_LOG to achieve a small improvement in the PR-CAUC of 2% when compared to INF. On informal graphs however, the LOG modification provides a significant improvement. On IMDb for example INF_LOG has a 12.8% higher PR-CAUC than INF. An effect that takes place as well on webgraphs. The results obtained by the various modifications seem therefore to be strongly linked with the nature of the graph. A less strict score (one with modifications) being appropriate for less strict graphs. A more detailed analysis of that will be possible after testing INF, INF_2D, INF_LOG and INF_LOG_2D on the webgraphs, in §7.2.

7.1.5 Quasi-local INF Scores

The last PR-CAUC evaluation we perform is on the quasi-local INF scores. In this section we test, for each modified version of INF, the performance of its quasi-local version. Table 7.6 shows the performance results. Surprisingly, the only graph in which a quasi-local scores achieves better results than its local version is WordNet. On the other graphs the local version works better. Regardless, we consider the tests

Score	WordNet PR-CAUC	Cyc PR-CAUC	IMDb PR-CAUC
INF_QL	0.720866	0.0452247	0.005800
INF_2D_QL	0.842386	0.0409292	0.010527
INF_LOG_QL	0.726982	0.0071217	0.016685
INF_LOG_2D_QL	0.836465	0.0077724	0.025504

Table 7.6: PR-CAUC score of INF_QL, INF_2D_QL, INF_LOG_QL, INF_LOG_2D_QL on three graphs.

performed on quasi-local scores to be inconclusive, as further discussed in §10.3, since the computationally expensive sampling process of the number of steps to be performed, and of the decaying parameter α, was not implemented.

Let us summarize the results obtained in this section in table 7.7, by showing the top three scores, out of 14 scores tested, by PR-CAUC achieved on each graph.

Rank	WordNet	Cyc	IMDb
#1	INF_2D_QL	INF	INF_LOG_2D
#2	INF_LOG_2D	INF_2D	INF_LOG
#3	INF_2D	INF_QL	INF_LOG_2D_QL

Table 7.7: Top 3 scores in PR-CAUC for each graph tested

7.2 Hyperlink Prediction Evaluation

In this section we consider the problem of predicting links among webgraphs. We compare the top three undirected scores of LP (CN, AA and RA) with all the local versions on the INF score (INF, INF_2D, INF_LOG and INF_LOG_2D), using six different webgraphs. The PR curves can be seen in Figure 7.5 (a larger version of these curves can be found in Appendix A.6). Evaluation is done using the PR-AUC. The resultant PR-AUC obtained by each of those curves are shown in Table 7.8. Other evaluation metrics (PR-CAUC and ROC-AUC) can be found in Appendix A.6.

First and foremost, INF_LOG_2D obtains the best PR-AUC values on all webgraphs by a large margin. The only score that gets close to INF_LOG_2D in performance is INF_LOG, particularly on the webND and baidu webgraphs. In these, INF_LOG_2D

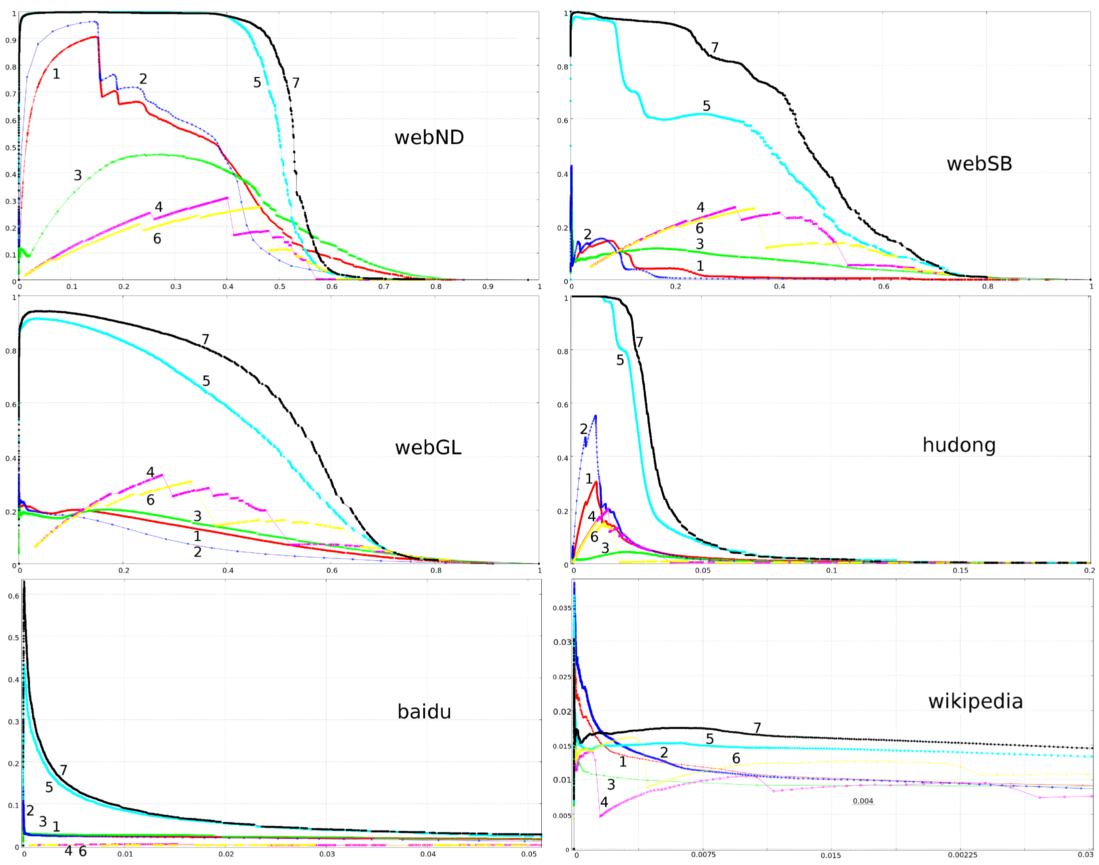


Figure 7.5: PR curves for all webgraphs, 1:AA, 2:CN, 3:RA, 4:INF, 5:INF_LOG, 6:INF_2D, 7:INF_LOG_2D. Recall in x axis, recall in y axis. Hudong, Baidu and DBpedia curves are zoomed in for clarity.

improves the PR-AUC of INF_LOG by *only* a 5%. When compared to the state-of-the-art local similarity-based algorithms (CN, AA and RA), INF_LOG_2D represents an improvement in predictive quality of several orders of magnitude (see Table 7.9). These results indicate that the INF score with both proposed modifications (2D and LOG) is the best local similarity-based algorithms found so far for predicting hyperlinks.

To further discuss the results obtained, let us classify similarity-based algorithms in two types: proportional and accumulative. *Proportional scores*, such as INF, weight the evidence of edges according to their local context, providing a normalized similarity for each edge. In INF for example, all possible edges share a maximum possible score of 2, when DED=1 and IND=1 (see Definitions 7, 8 and 9), regardless of the degree of the related vertices. Proportional scores are therefore unbiased towards edges among high-degree vertices. *Accumulative scores* on the other hand only measure the absolute amount of evidence, ignoring the local context (*e.g.*, number of shared neighbors, regardless of the vertex degree). In these scores, edges are evaluated and ranked from a graph-wide perspective, which clearly benefits edges among high-degree vertices.

Previous results discussed in §2.1.2 have shown that in general, accumulative scores perform better than proportional scores: the top three scores found in the bibliography are accumulative (*i.e.*, CN, AA and RA). This can be explained by the importance of the preferential attachment process (*i.e.*, the rich get richer), which accumulative scores implicitly satisfy, and proportional scores avoid. But the generally poor results of proportional scores is also caused by the volatility of predictions among low-degree vertices, which proportional scores may overestimate. To further validate that point we tested one well-known proportional score on the six webgraphs inhere presented, the *Jaccard's coefficient* score (Liben-Nowell and Kleinberg, 2007), and got coherent results. Jaccard's performance was incomparably worse on all webgraphs; when plotted together with the other scores, Jaccard's PR curves were a flat line on the x axis.

Proportional scores are theoretically and empirically in disadvantage with accumulative scores. However, INF (a proportional score) outperforms CN, AA and RA (the top three accumulative scores) on two of the six webgraphs, while still achieving competitive results on the other four (see Table 7.9). Apparently, INF works around the overestimation of edges among low-degree vertices (an unfortunately frequent type of vertex on real networks such as webgraphs) by fitting edges into a more complex model than just plain neighborhood intersection size. We therefore consider the results

	AA	**CN**	**RA**	**INF**	**LOG**	**2D**	**LOG_2D**
webND	0.31679	0.31855	0.21178	0.09966	0.50096	0.09395	<u>0.52640</u>
webSB	0.02218	0.01669	0.05491	0.10530	0.33605	0.09847	<u>0.45156</u>
webGL	0.08961	0.06227	0.10035	0.12826	0.41954	0.12485	<u>0.49210</u>
hudong	0.00555	0.00743	0.00223	0.00402	0.02953	0.00268	<u>0.03424</u>
baidu	0.00285	0.00176	0.00308	0.00065	0.00504	0.00060	<u>0.00528</u>
DBpedia	0.00056	0.00038	0.00016	0.00049	0.00066	0.00055	<u>0.00071</u>

Table 7.8: AUC obtained by each tested score on the PR curves shown in Figure 7.5.

	AA	CN	RA
webND	+66%	+65%	+148%
webSB	+1935%	+2605%	+722%
webGL	+449%	+690%	+390%
hudong	+516%	+360%	+1435%
baidu	+85%	+200%	+71%
DBpedia	+26%	+186%	+443%

Table 7.9: Percentage of PR-AUC improvement achieved by INF_LOG_2D over the current top three local similarity-based algorithms for each webgraph.

achieved by INF as evidence of two facts. First, that the hierarchical assumptions of INF are key for hyperlink prediction. And second, that proportional scores can be competitive if properly adapted.

To further elaborate on the relation between these two classes of similarity-based scores, let us consider the results obtained by the LOG modification. While INF was a proportional score, INF_LOG is in fact a *hybrid score*: it first normalizes evidence given the local context (proportionally) and then weights that proportion with the absolute size it was based on (accumulatively, see Definitions 11 and 12). The hybrid score INF_LOG outperforms all purely proportional or accumulative scores on the six webgraphs tested by a large margin (see Table 7.8). This indicates that hybrid scores are the most competitive approach for the hyperlink prediction problem.

It is worth noticing that, even though INF_LOG_2D achieves the best overall AUC in the DBpedia webgraph (see Table 7.8), CN and AA outperform INF_LOG_2D at very high precisions on the DBpedia webgraph, something not happening for any other tested graph. This anomaly is likely to be the result of an imperfect evaluation on the DBpedia webgraph: due to time constrains, algorithms computed only the edges originating from the 350,000 vertices with higher degree. This approach, even though coherent with the goal of maximizing the graph data being processed (we focus on *rich* vertices), penalizes proportional and hybrid scores in front of accumulative scores. Accumulative scores obtain their high certainty predictions around high-degree vertices. Thus, within the evaluation performed on DBpedia, limited on the highest degree vertices, accumulative scores like CN and AA will be able to find most reliable predictions. Proportional and hybrid scores on the other hand do not necessarily find their high certainty predictions around high degree vertices, although hybrid scores lean towards them. As a result some of the high confidence predictions of proportional and hybrid scores will not be evaluated in this test setting. These results suggest that accumulative scores may be more useful for contexts with limited computational resources, as one can easily obtain their high certainty predictions by computing only a few, highly connected vertices. Hybrid and proportional scores on the other hand can produce more reliable solutions, but require of a more exhaustive exploration of the graph.

7.3 Predicting Top-links

The tests performed in §7.1 are the standard objective evaluation of LP scores. These tests show which scores achieve the best predictive results under equal conditions. From a practical point of view though, the production of the whole PR curve is rarely useful as low precision predictions can hardly be applied. In practice, the most straight-forward application of link prediction in the short-term is as a support tool, discovering and proposing new high-certainty edges. For this purpose it is not necessary to calculate the likelihood of all edges, as the PR and ROC curve do. Instead one needs only to find and return a set of relations in the graph highly likely to exist. In Tables 7.10, 7.11 and 7.12 samples of the most likely edges obtained for various graphs are shown. For each of those high-certainty predictions it is also shown whether or not they belong to the test set, and therefore, if these predictions are considered to be mistakes or correct guesses in our evaluation.

When high precisions cannot be achieved in LP, the process of integrating the top-edges proposed into the actual graphs requires of supervision. For informal graphs managed by communities like DBpedia that would not be much of a problem, as edges could be suggested to each Wikipedia as they supervise an article. The same would apply for example to social networks, where edges could be shown to users as recommendations. Or to webgraphs, where each webmaster could decide whether or not to add a set of proposed edges for its managed web pages.

In formally defined graphs like WordNet and Cyc, the integration of the top-edges predicted may need to be performed exclusively by experts. In this type of graphs, we find a different purpose for the top-edges predicted by focusing on the apparently

Relation found	**Edges in test set**
embrace.v.02 → hold.v.02	Yes
observation.n.04 → abstraction.n.06	Yes
research.v.02 → analyze.v.01	Yes
bet.v.02 → compete.v.01	Yes
climb.v.01 → travel.v.01	Yes
flatten.v.01 → change.v.01	Yes
hierarchy.n.01 → abstraction.n.06	Yes
escape.v.01 → leave.v.01	Yes
organize.v.02 → control.v.01	Yes

Table 7.10: Sample of top edges predicted by the INF score on the WordNet graph.

Relation found	**Edge in test set**
moss → type of organism	No
tank cannon → type of weapon	No
political instability → event	No
province → non-overlapping geopolitical entity	No
breezy location → weather attribute	No
dancing by humans → intelligent agent activity	Yes
cuisine → edible thing	No
propelling an object → movement	No

Table 7.11: Sample of top edges predicted by the INF score on the Cyc graph.

wrong ones. Determining which edges are definitely wrong takes less time than determining which edges are definitely right. And since INF predictions are based on hierarchies, the detection of certainly wrong top-edges typically implies inconsistencies in the graph hierarchy (*e.g.*, missing or wrong edges). People in charge of building or maintaining these data sets could benefit from such information and use it to improve the quality of the knowledge base. As an example of this application consider the following scenario we encountered when reviewing the results: in the WordNet graph, the edge $chordate.n.01 \rightarrow vertebrate.n.01$ is predicted with high reliability by the INF score. This edge is apparently wrong, as vertebrates are a subtype of chordates. The edge $vertebrate.n.01 \rightarrow chordate.n.01$ is in fact found in the original graph. We investigated on the reasons of this prediction, and realized that even though chordate is a generalization of vertebrate, the vertebrate synset has more specializations than chordate ($|D(vertebrate)| = 3844$, $|D(chordate)| = 3087$), and in fact there are only 12 specializations of chordate which are not vertebrates. This clearly shows a bald spot in WordNet regarding chordate animals that are not vertebrate, and explains the wrong prediction. Fixing the balance in that part of the hierarchy by adding new non-vertebrate chordates in WordNet would also fix the prediction of INF, as the certainty of the wrong edge would decrease. This shows the relations between INF and hierarchical knowledge, and how it can be used to build hierarchically coherent graphs.

Relation found	**Edge in test set**
Stamen → Flowering_Plant	No
Predation → Animal	No
Doctor_(Doctor_Who) → United_Kingdom	No
Battle_of_Iwo_Jima → World_War_II	No
Districts_of_Turkey → Eastern_European_Time	No
2010_United_States_Census→United_States	Yes
Poverty_threshold → Population_density	No
FM_broadcasting → Hertz	No
Batting_average → Home_run	No
Census-designated_place → List_of_sovereign_states	No
NBC → United_States	Yes
DC_Comics → Comic_book	No
Cold_War → United_States	Yes
German_Empire → Germany	No
Trumpet → Jazz	No

Table 7.12: Sample of top edges predicted by the INF score on the DBpedia graph.

Chapter 8

Computing Link Prediction

This thesis focuses on the computation of large scale graphs. Processing graphs which are large and continuously growing is in fact the main motivation for using local similarity-based scores, the most scalable of link prediction methods. However, even with the most *computationally cheap* of LP algorithms, processing large graphs eventually demands of HPC tools and resources. For specific commercial applications, and depending on the exhaustivity required, HPC may not be necessary: one can quickly find ten high-certainty edges to be added to a network. However, if one intends to thoroughly compute lots of edges within a graph, for either formal or commercial purposes, HPC becomes a complete necessity. The use of HPC tools and infrastructure in this thesis is further motivated by issues found in graph related tasks. Issues that reduce the computational efficiency of any graph processing algorithm. Theses issues can be summarized in two topics: data processing problems and parallelism requirements.

Graph data is high-dimensional as vertices are related many to many. Current computers on the other hand, physically store data in one-dimensional memories, assuming sequentially and exploiting features like data locality. When working with graph data one quickly realizes that neither memories, nor basic data structures (*e.g.*, arrays) are designed for efficiently processing high-dimensional data. Which frequently results in data access operations defining bottlenecks within graph algorithms. On top of that, networks originating from real world data, like the ones we work with, are very large and very sparse (see Table 3.1), properties that further complicate the efficient handling of graphs. Focusing on this issue, we explore two optimizations based on *how* data is stored (in §8.1) and *how* precisely is data computed (in §8.2), with the goal of increasing the efficiency of our LP algorithms.

Besides data related issues, the other main topic to consider is parallelism. LP algorithms must compute a huge amount of edges; in this thesis within the order of billions (see Table 7.1). Computing them linearly, one by one, is clearly impractical, which makes High Performance Computing (HPC) parallelism necessary for the feasibility of our work. In §8.3 we review some of the parallel models available today for graph processing, and present the implementation of our LP algorithms on two of those models. Then, in §8.4 we describe the HPC infrastructure we have used in this thesis.

8.1 Data Ordering and Locality

Memory latency is the time it takes memory access operations to obtain data requested from a given storage location. Cache memory latency is short due to the physical embedding of the cache inside the CPU. RAM memory (or main memory) on the other hand is located farther away from the CPU, making its latency much higher. Thus, the number of computational cycles needed by memory access operations strongly depends on which memory is being accessed. When a memory access operation requests a set of data, and this data is not found on cache memory, the operation must wait for the data to be fetched from main memory and copied to cache memory. These situations, known as a memory *misses*, entail a loss of computational efficiency, as several cycles are spent waiting for the data to arrive. LP in large graphs is a *data-intensive* task, as it requires continuous accesses to graph data while its arithmetic operations remain relatively simple (see Definitions in §2.1.2 and §4.1). When a data-intensive task is executed on a large data, set memory misses tend to increase, to the point that they may define the computational cost of the whole task. Moreover when data is high-dimensionally related (*i.e.*, graph-like). For these reasons, when developing graph processing algorithms data structures and data access operations must be implemented with special care.

To decrease cache misses it is important to store and access graph data sequentially in memory. By doing so, when data is fetched from main memory one can be sure than only useful data will be copied into the cache. Our goal is to store the entire graph in a consecutive portion of memory, using containers that allow us to decide not only where is data physically located, but also in what order. Then, to maximize data locality we design the graph processing algorithm to consume the graph data in the same order used for its storage. For example, for our OpenMP implementation further discussed in §8.3.1.3, our approach was to store the graph vertices as adjacency lists, sorting vertices based on their degree (total number of edges) and sorting adjacency lists by vertex id. Hence, the most connected vertex is stored in the first memory location, as a list of all the vertices related with it, followed by the second most connected vertex and its neighbors, and so on. We designed our LP algorithm to access vertex data following the same order, from largest degree vertex to smallest. In our tests this approach achieved better performance than using the inverse ordering (from small degree to large) and than a random ordering.

Regardless of how the graph is stored in memory, misses will happen. One simply cannot store all near-by vertices near-by in memory, since graphs represent a high-dimensional space and memory is indexed as an unidimensional space. However, since the data regarding high degree vertices will be the one accessed more often (at least for the LP problem), by keeping high degree vertices together one can use possible cache misses to bring data into memory which will be used later with a higher probability.

8.2 Precision Reduction

Local algorithms of LP compute the likelihood with which an edge exists by operating on the neighborhood of the two vertices defining the edge. Similarity scores use mathematical operations such as division and logarithm that can produce similarity values within the real numbers, with an infinite sequence of digits. The number of distinct similarity values found in a graph thus grows as the number of possible maximum neighbours grows, and it reaches millions for graphs as large as the ones used here. In this context we argue that calculating edge similarities with the largest possible precision is not necessary or useful, moreover when these similarities define individual points later on aggregated into a higher order curve (PR or ROC, see §5). A reduction of precision in the calculus of similarity scores can thus save both time and space; double-precision floating-point operations require more memory (providing precisions from 15 to 17 decimal digits) and are harder to calculate (requiring more computing time) than for example single-precision floating-point operations.

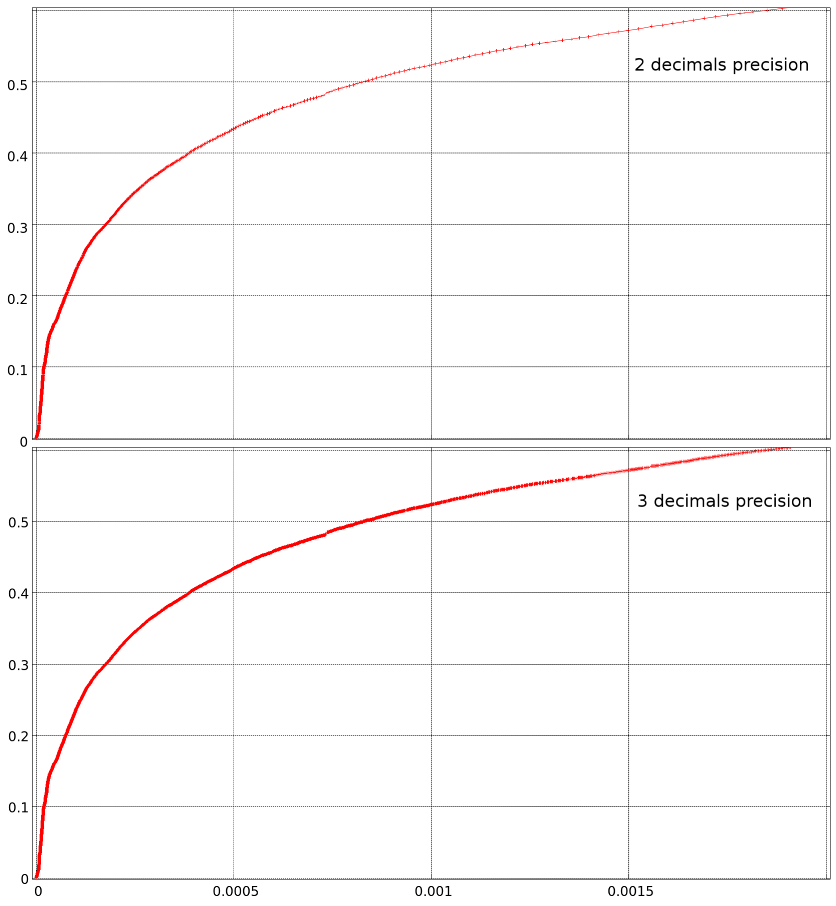


Figure 8.1: ROC curve for the IMDb data zoomed in. The curve on top is calculated with 2 digits precision. The curve at the bottom with 3 digits precision.

Overall, PR and ROC curves are hardly affected by a reduction in the precision, due to the larger scale of the curve in comparison with individual scores. The effect of reducing the precision of score computation on the resultant curves is on the resolution of the curves: the number of points composing the curves is reduced through a process of aggregation, as points which are very close to one another when calculated with high precision become the same one when calculated with a lower precision. This loss can be understood as a process of approximation, as the general curve remains the same. An example on how a loss of precision may not significantly affect an PR or ROC curve consider the charts shown in Figure §8.1. Both ROC curves shown are obtained from the IMDb data set using the same LP score, but the one on top corresponds to a precision of 2 decimals while the one at the bottom corresponds to a precision of 3 decimals. The 2 decimals curve is defined by 3,134 points, while the 3 decimals curve is defined by 16,522 points. Notice that, even though we have applied a large zoom to the curves (notice the axis scale), these are virtually indistinguishable. The benefits of this approximate approach is even clearer for larger graphs, where the number of points of the curves (*i.e.*, the number of distinct similarities found) can easily reach millions, and where a reduction of calculus precision can significantly reduce computational cost.

8.3 Parallelization

Local similarity-based algorithms represent the most scalable approach to LP (see the complexity analysis in §3.5). However, even local algorithms cannot *linearly* compute graphs (*i.e.*, if computation is performed by a single computing unit, one edge at a time) with millions of vertices within an acceptable time. HPC provides both infrastructure (*e.g.*, supercomputers) and tools (*e.g.*, programming models) for parallel computation. And we integrate both in our research, which grants us insight into the potential capabilities of large scale graph mining, as we discuss in §10.1.

One of the most important contributions of HPC for the large scale computation of graphs are parallel programming models. These models have a huge impact not only on computational performance but on algorithm design as well. Due to the particularities of graph processing, the HPC community has recently proposed and released several graph specific parallel programming models. At the same time, the *traditional* programming models have also evolved, becoming more efficient and user friendly for non-HPC developers. We explore the available possibilities for parallel graph processing first by implementing the LP problem using both OpenMP(ARB, 2013) and OmpSs (Duran et al., 2011) programming models in §8.3.1. We describe the software design and optimization process used in this setting, seeking an efficient solution. Then in §8.3.2 we review the emerging graph specific models, and implement the same problem using one of them. We discuss and compare both solutions in §8.3.3.

8.3.1 OpenMP and OmpSs models

The most common parallel programming models are based on the creation of threads spanning over one or more instructions of a program (*i.e.*, fork-join). These threads are responsible of executing subparts of the whole program, and may depend on one another at some point of their execution (*e.g.*, through data dependencies). The main challenge in the implementation of a graph processing algorithm using this model lies in the high-dimensionality of graphs. High-dimensionality implies multiple relations within the data, which easily translates into lots of dependencies, and a consequent loss of efficiency. In §8.3.1.1 we describe how we design our algorithm in this scenario to minimize dependencies, and show samples of it. In §8.3.1.2 we introduce two different memory settings of parallel computation: shared memory and distributed memory. We start by detailing our implementation in a shared memory setting in §8.3.1.3 using OpenMP. We then extend this implementation to a distributed context in §8.3.1.4, thanks OmpSs.

8.3.1.1 Link Prediction as an Embarrassing Parallel Problem

One of the main concerns when implementing an algorithm for a parallel environment is the existence of *data dependencies*. Dependencies determine execution order constrains among portions of code and imply synchronization points, as one portion of code must wait for another portion to be executed first. Through the existence of dependencies, threads see their work flow halted as they wait for other threads. In essence, dependencies define bottlenecks in the parallel execution of code, and reduce the efficiency of computational resources usage.

A related concept within the field of parallel computing is that of *embarrassingly parallel* problems (Foster, 1995). This notion applies to algorithms that can be parallelized without the definition of significant dependencies. These are therefore problems which can achieve a huge efficiency through parallelization, as there will be almost no idle resources in their computation. Embarrassingly parallel problems are capable of decreasing computational time almost linearly with the number of computing units available, as the various threads must not endure waiting times. Clearly, embarrassingly parallel problems are the most appropriate ones to parallelize. As we will see next, LP can be defined as an embarrassingly parallel problem, a feature we exploit in the work presented here.

As said in §2, one of the key features of similarity based LP algorithms is that the score of each edge can be calculated independently from the rest. This particularity gains a huge relevance now as it allow us to define LP as an embarrassingly parallel problem. Fully testing a LP algorithm on a graph equals to calculate the similarity of all possible ordered pairs of vertices (*i.e.*, of each possible directed edge). Since each similarity can be calculated independently, we can evaluate them all simultaneously without dependencies. Considering the huge number of edges to test (at times in the order of billions), the code parallelization design defines the efficiency of the algorithm, and eventually, the size of the graph it can process.

Our algorithmic design is divided into two fully parallel sections. On the first one we calculate the similarity of each possible edge in parallel, storing the results obtained for the edges originating on each vertex in a different structure. Or what is the same, for every vertex n in the graph we define a distinct data structure storing the similarities of edges of the form $n \rightarrow X$. An overview of this code can be seen in Listing 8.1. On the second section of the code, for the purpose of building the similarity evaluation metrics discussed in §5, we need to combine the results obtained. The purpose of the second section is to calculate the total number of correctly identified edges (true positives) and the total number of incorrectly identified edges (false positives) at every distinct threshold found (*i.e.*, for every distinct edge similarity value). To do so we must consider the scores achieved on all edges at the same time, checking both the computed similarity of the edge and whether or not it was a correct prediction (*i.e.*, if it was found in the test set). However, due to the parallelization of the first section of the code, the results obtained for every vertex are stored separately. Thus, the number true positive and false positives achieved at every distinct threshold are distributed in a series of arrays. To fix that, between the first section and the second section of the code we perform a reduction process. This allows us to simplify and speed up the computation of the second section, and to define it without dependencies. For every distinct threshold obtained, this reduction will be in charge of calculating the total number of true positives and false positives found on all vertices.

Listing 8.1: Code skeleton for similarity evaluation of all edges in a graph

```
// Structure to independently store the results of each vertex
vector <vector<class<float ,int ,int> > > graph_results;
for_each (vertex n1 in graph){
  // Structure to store the true positives (1st int) and false positives (2nd int)
  // found for n1 at every distinct similarity (float)
  vector<class<float ,int ,int> > n1_results;
  for_each (vertex n2 in graph, n2!=n1){
    if(n1->n2 exists in graph) continue;
    bool n1_n2_positive=false;
    if(n1->n2 exists in test edges) n1_n2_positive=true;
    float sim_n1_n2 = calculate_similarity(n1,n2);
    if(n1_n2_positive) n1_results[sim_n1_n2,true_pos++,false_pos];
    else n1_results[sim_n1_n2,true_pos,false_pos++];
  }
  graph_results.push_back(n1_results);
}
```

The second part of the code calculates the graph-wide performance of LP algorithms at different thresholds. For each threshold there is a precision and a recall, from which we obtain the final PR curves. Thanks to the reduction process performed between the first and the second part, by the beginning of the second part the total number of true positive and false positives found at each different similarity value is known. With these two values, and knowing the size of the graph, we can calculate the points composing the metrics discussed in §5 through an aggregation process. This task is also embarrassingly parallel, as the performance at each threshold can be calculated independently from the rest of thresholds. It is important to parallelize this task, as the number of distinct similarity values in large graphs can be also large (up to millions of values). An overview of this second section of code can be seen in Listing 8.2.

Listing 8.2: Code skeleton for full graph performance evaluation

```
// Structure to store the results obtained at all thresholds
vector <pair <int ,int> > full_results;
for_each (similarity value sim1 in graph){
  int true_positives_sim1 = 0;
  itn false_positives_sim1 = 0;
  for_each (similarity value sim2 in graph){
      if(sim2>=sim1){
        true_positives_sim1 += true_positives_sim2;
        false_positives_sim1 += false_positives_sim2;
      }
  }
  full_results.push_back(sim1,true_positives_sim1,false_positives_sim1);
}
```

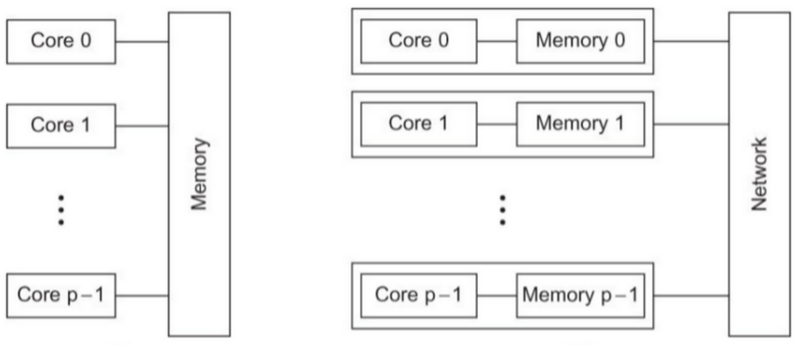


Figure 8.2: On the left, structure the a shared memory architecture. On the right, structure of a distributed memory architecture.

8.3.1.2 Shared vs Distributed memory

One of the most important distinctions in parallel computing is whether the available memory space is shared (*i.e.*, centralized), or distributed. On the shared memory paradigm there is a single memory location which contains all data, and which is directly accessible to all computing units of the machine. An example of that would be a computer with local memory and several processors working in parallel consuming data from the same memory. The alternative, distributed memory, splits memory physically and therefore data among different separate locations, typically each hosting a distinct set of computing units. These distributed components are accessible from one another through a network (*i.e.*, a computing unit can access data not available on its local memory), but often at a higher latency. An example of that would be a set of computers connected through a network, each hosting a part of the whole data set, which communicate the missing parts among themselves at the request of their local processes. See Figure 8.2 for a graphical representation of both paradigms.

In most architectures shared memory designs are simpler and more efficient. Data is centralized on a single location and can it be accessed directly and equally from all computing units. However, when working with very large graphs it may be the case that data simply does not fit into a single memory storage (*i.e.*, insufficient RAM memory). In these cases we will be forced to split data among different locations, as we will see in §8.3.1.4.

8.3.1.3 OpenMP parallelization

The code used for most of the tests presented in this thesis was parallelized using the OpenMP shared memory model API (ARB, 2013). This API provides a set of compiler directives that extend the C/C++ and FORTRAN base languages. OpenMP directives define how code is to be parallelized and how data is to be shared. We chose OpenMP because it is portable, scalable, flexible and the de-facto standard.

We split our code in two parts as described in §8.3.1.1. Within the first section of code, the parallelization was done on the most external loop (line 4 of Listing 8.1), effectively distributing its iterations among different threads. This approach guarantees that all similarities $n \rightarrow X$ of a given vertex n (*i.e.*, a full iteration of the outermost loop) are calculated by a single thread, thus avoiding any dependencies. This was implemented using the OpenMP `parallel` directive, which creates a team of threads, and the `for` directive, which splits the iterations of a loop among the threads of a team. See an example of this directive in Listing 8.3.

Between the first and second sections of the code we implemented a reduction with OpenMP. This was done using a similar directive as the one seen in Listing 8.3, with the only addition of information on which variables where to be reduced. Finally, the second section of the code was also parallelized using the same directives used for the first section. In this case the parallelization was also performed on the most external loop (line 4 of Listing 8.2). This approach guarantees that each possible threshold (*i.e.*, each point within the PR and ROC curves) was calculated by a unique thread, thus avoiding any dependencies.

Listing 8.3: OpenMP directives used to parallelize the first section of code

```
#pragma omp parallel for schedule(dynamic,100)
```

Of the different ways of splitting iterations among a team of threads, we found that the most efficient for our problem was a *dynamic scheduling*. In OpenMP a dynamic schedule splits iterations in chunks of pre-determined size (100 in the example of Listing 8.3), and assigns one chunk per thread as these request it. In this setting each thread will compute 100 consecutive vertices (*i.e.*, 100 iterations of the outermost loop) before asking for more. It is important to define chunk sizes according to the problem size in order to minimize imbalances. If the chunk size is too large, at the end of the computation one or more threads will remain idle for a long time while the rest of threads finish their last chunks. If chunks are too small the constant scheduling of threads, as these finish chunks and requesting for more, may slow down the whole process. In the case of LP, the larger and denser the graph, the smaller the chunks must be, as iterations in a large graph will be more time consuming, thus increasing the possibilities of imbalance. For our code we found chunks between 100 and 2000 iterations to provide the best results.

Given the previous parallel design and use of OpenMP directives, we identified a single bottleneck in each of the two sections of code. That is when results are stored (*i.e.*, pushed back) into the global vectors. These instructions (line 23 of Listing 8.1, and line 15 of Listing 8.2) can only be executed by one thread at a time, as the structure being filled is the same. This data access is protected in OpenMP through a `critical` directive (show in Listing 8.4) guaranteeing that only on thread will execute that line at the same time.

Listing 8.4: OpenMP directive used to protect sequential parts of the code

```
#pragma omp critical (vector_push)
```

8.3.1.4 OmpSs parallelization

Storing graphs entirely in a unique memory space, *i.e.*, in shared memory, simplifies data access and memory management. Unfortunately, capacity of memories is limited which means that eventually a unique memory space will not be sufficient to store a large enough graph. The solution typically used in the HPC field is to distribute memory, providing several autonomous memory spaces where different parts of the data are stored. Obviously, this approach entails a much more complex memory management, as data must be physically split among locations, and data communications must be implemented so that threads within a memory space can access data found on other memory spaces when necessary.

Regardless of the difficulties added by distributed memory models, it is obvious to us that it eventually becomes a necessity. We realized that much when working with the DBpedia graph, composed by $17 \cdot 10^6$ vertices. In our tests we saw how some of the structures required by certain LP algorithms could not fit into a unique memory space with 24GB of RAM, even though we spent a significant amount of time optimizing the memory requirements of our algorithms. For example, the structures needed to calculate the specificity level of every vertex in the graph (*i.e.*, the number of vertices from which each vertex can be reached), needed by our hierarchical version of the undirected scores (see §2.1.1), could not fit into memory. Neither did the set of 2-step neighbours of every vertex in the graph, needed by the 2-step quasi-local scores (see §4.1.2). Since it was not possible to pre-calculate and store these structures into memory during the graph reading process, we tried to calculate them *live* on request from more simple structures which could fit into memory (*i.e.*, adjacency lists). Unfortunately, the constant accesses to main memory (*i.e.*, RAM) required by this operations made the executions unbearably slow. Global and quasi-local scores therefore require distributed memory settings for graphs composed by tens of millions of vertices.

Local scores can compute graphs like DBpedia in a shared memory environment. Eventually though, we will be interested in working with larger graphs for which a distributed memory is needed even by local methods. Consider for example the webgraph defined by Internet with 3.5 billion web pages, or a brain connectome graph composed by billions of neurons. For this kind of data sets the only feasible solution nowadays is distributed memory. And not only because of space requirement, but also because of the time complexity. Hundreds of cores computing in parallel will be needed to mine those graphs, and the number of computing cores accessing a single shared memory space is rarely over a few dozens.

With the goal of running LP methods on a distributed memory environment, we parallelized our code using the OmpSs programming model (Duran et al., 2011). OmpSs is developed by the Barcelona Supercomputing Center (BSC) and supports OpenMP like directives. It has been adapted to work on clusters with distributed memory (Bueno et al., 2011) with a significant reduction on the programmers effort. Thanks to that, to switch our code from OpenMP to OmpSs we only had to change the OpenMP directive parallelizing the loop (see Listing 8.3) with one defining tasks instead (see Listing 8.5). Additional parameters were added to this directive to specify which data had to be copied to and from a given location when executing a given iteration of the loop.

Listing 8.5: OmpSs directive used to define tasks

```
#pragma omp task
```

Even though in this approach graph data has to be communicated among computing entities (as each entities only has direct access to portion of the whole graph), our results show no relevant overhead added by this communication. This is so because in our LP algorithmic design it is easy to predict which edge will be evaluated next, and therefore which graph data will be needed next by each thread (*i.e.*, which vertices and neighbours must be brought to memory). Thanks to this foreseeability, data can then sent before it is needed, thus avoiding idle threads and the consequent communication overhead. From a computational point of view this means that LP may scale almost linearly on distributed memory contexts. A feature with a huge impact that we discuss as future work in §10.2. Finally let us mention that, even though the OmpSs code has been implemented and is being tested, no relevant results have been obtained for it yet. A study in that regard is also a priority line of future work, as discussed in §10.2

8.3.2 Pregel Model Implementation

The inadequacy of traditional computing paradigms for processing graphs becomes more obvious as the popularity and size of graph data sets grows. Aware of that, the HPC community has made an effort to define computing models specific for graphs, with the goal of producing efficient frameworks for the computation of large scale networks. The resultant parallel graph models consider computation in a rather unique way, typically either from a vertex or edge perspective. Vertex centric models (Gonzalez et al., 2012; Low et al., 2010; Malewicz et al., 2010) define computation from the point of view of vertices, as code is to be executed from within each of the vertices of the graph. In these models a vertex typically has access to the data of its nearby vertices, and can also send messages to other vertices. Edge centric models (Roy et al., 2013) are less popular, and can be analogously described by using edges as the basic computing element instead of vertices. As a final example of this family of parallel graph computing models let us mention a recently proposed path centric model (Yuan et al., 2014), that achieves remarkable performance when implementing algorithms following the directionality of graphs (it therefore requires the graph to be directed).

To try this new and promising approach to large scale graph processing we implemented our algorithms using the ScaleGraph API (Dayarathna et al., 2012), which is based on the Pregel model (Malewicz et al., 2010). Pregel was one of the first parallel graph models to be proposed. It is a vertex-centric approach following the Bulk

Synchronous Parallel (BSP) paradigm. Pregel is based on *supersteps*, iterations of parallel computation for every vertex of the graph, separated by global synchronization points (hence BSP). At those synchronization points is where communication takes place. During a superstep each vertex receives the messages sent to it on the previous superstep, modifies its internal state according to those, and finally sends messages to other vertices that will be received on the next superstep. Since within a superstep the computation of each vertex is done in parallel without dependencies, the Pregel model obtains good parallel speedup and scalability on algorithms that define well balanced supersteps. Based on the Pregel model, ScaleGraph provides an open-source API in the X10 programming language. ScaleGraph was conceived to process massive graphs, it is documented and under active development, which makes it idoneous for our use.

Using the Pregel model entails a completely different algorithmic design. While in the OpenMP/OmpSs model the algorithm was designed as two nested loops evaluating all pairs of vertices and parallelized through the distribution of the loop iterations, in the Pregel model, code is designed from the point of view of each individual vertex and parallelized through the distribution of vertices. To reduce dependencies we implement our algorithm such that each vertex must evaluate the edges that have itself as an origin. Since we implement local similarity-based algorithms, the only edges that will have a similarity degree different than zero will be those located at exactly two steps of distance from the source vertex (and thus having at least one shared neighbor). As a result, vertex x will be responsible of evaluating edges of the form $x \rightarrow Y$, for the set of vertices Y located two steps away from x. If every vertex calculates the similarity of its designated edges all relevant edges will be computed once and only once.

The previous design could be executed in parallel without dependencies, since each vertex independently calculates a disjoint set of edges. That is if every vertex has access to the information it needs. For a vertex x to compute all edges of the form $x \rightarrow y$, x needs to know all the paths of length two originating in itself. By doing so x can know which are the target vertices (those vertices at distance two), and which are all the paths that lead to those targets (those vertices at distance one between x and y). However according to the Pregel model one vertex does not know which vertices are two steps away from itself, as it is only aware of its direct neighbors. To make the necessary information available to every vertex in the graph we must use messages

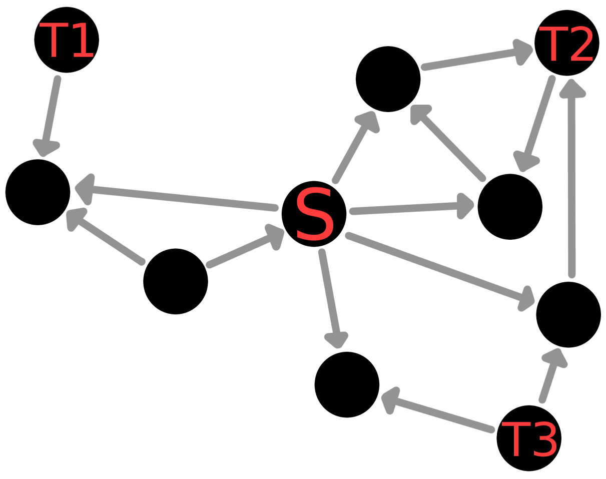


Figure 8.3: Example of two step neighborhood around one source vertex (S). T1, T2 and T3 are potential target vertices to evaluate.

between vertices. These messages are in charge of allowing each vertex to know its two steps neighborhood.

In our Pregel design, in a first superstep vertices send the information of their neighbors to all their neighbors. By doing so, at a second superstep each vertex will receive the information necessary to know its full two steps neighborhood. See Figure 8.3 as an example of that neighborhood, where source vertex S is shown together with all the paths of length 2 originating in S. According to the graph of Figure 8.3 $T1$, $T2$ and $T3$ are the only possible target vertices of S, which means that the only edges to be computed by S will be $S \rightarrow T1$ having only one path, $S \rightarrow T2$ having three paths, and $S \rightarrow T3$ having two paths.

A simplified overview of the ScaleGraph code to be executed on each vertex is shown in Listing 8.6. In this design messages are sent in the first superstep, while all paths to all targets are calculated in the second superstep. Once every path (named $first\_step$ in the example of Listing 8.6) that lead to a given target (named *second_step* in the example) are known, we can calculate the similarity score of the edge pointing to the target. For example, in the case of the CN algorithm the score is simply the number of different paths that lead to a given target. In the case of INF more information is needed, for example the directionality of paths or the degree of vertices (see Definitions of §4.1 for more details).

Listing 8.6: Code skeleton of the ScaleGraph implementation

```
//First superstep, send list of all neighbors to all neighbors
if(superstep == 0){
  sendToAllNeighbors(listOfAllNeighbors);
}
//Second superstep, process recieved lists of neighbors
if(superstep == 1){
  //For each message from neighbor
  for_each (messageRecieved in Messages){
    first_step = messageRecieved.sender;
    //For each vertex at two steps distance
    for_each (second_step in messageRecieved.neighbors){
      //If edge already exists, skip. Otherwise store new path
      if(self.OutNeighbors contains second_step) continue;
      paths.add(first_step, second_step)
    }
  }
}
```

Using a dedicated API such as ScaleGraph has several benefits. To start with, ScaleGraph builds and handles all graph representation structures, allowing the developer to forget about data handling operations. At the same time ScaleGraph is designed for both a shared memory and a distributed memory setting. This is particularly interesting for the programmer, as increasing the number of computational or memory resources becomes completely transparent. If the algorithm is executed in a distributed memory context, ScaleGraph automatically splits the graph among the various computing units. Similarly, ScaleGraph handles all required communication, such as messages sent among vertices that may or may not be within the same physical location.

8.3.3 Graph Programming Models from an AI Perspective

In §8.3.1 and §8.3.2 we described two different implementations of the same problem, that of evaluating the local similarity of all edges missing from a graph. The primary difference between both implementations was the parallel programming model being used. For the first implementation we used the well known OpenMP model, while for the second we used the more innovative Pregel model through its ScaleGraph implementation. The differences between both models lead us to define different algorithmic designs for the same problem.

A formal performance evaluation of graph programming models would be interesting

(Guo et al., 2014). Regardless, a proper evaluation requires implementing a wide variety of models (we only implement two), using the same algorithmic design in them (we use two different designs, as we discuss next) and using the same infrastructure for their computation (we use different hardware, as discussed in §8.4). Instead of evaluating graph programming models we introduce an equally interesting and useful discussion: the evaluation of graph processing models from the perspective of the AI developer. The goal of this evaluation is to reduce the large gap found between HPC researchers and the graph mining community. By testing and comparing graph programming models from a developer perspective on a relevant and realistic problem we can on one hand encourage and advice AI researchers on how to use these powerful tools, while at the same time provide feedback to HPC scientists on how to increase the popularity and utility of their work.

The algorithm design shown used with the OpenMP model loops over all pairs of vertices (thus N^2 where N is the total number of vertices), and for each pair it calculates the intersection of their neighbors. The resultant algorithmic complexity is $O(N^2 * k)$ where k is the average vertex degree. The other algorithm design, used with the Pregel model, runs for every vertex in the graph (thus N) visiting its neighbors and the neighbors of its neighbors (thus k^2 where k is the average vertex degree). The resultant algorithmic complexity of the second design is therefore $O(N * k^2)$. Notice that the difference in complexity between both designs is quite large, as $N \gg k$.

The motivation of the first design, with a complexity of $O(N^2 * k)$, was to maximize locality, a key feature as discussed in §8.1. In this design graph data is stored in a predefined order: first vertex n_1 and its neighbors, then vertex n_2 and its neighbors *etc.* The same order is used to consume the data. Since in the OpenMP and OmpSs models the programmer may have total control over which data structures are used and how are these stored, by using this design we were able to maximize the efficiency of memory related operations. Indeed, our OpenMP implementation achieved a number of instructions per cycle (IPC) between 1.6 and 2.2 for all graphs tested. Achieving a high IPC in such a data-intensive task being particularly significant in terms of performance.

For our ScaleGraph implementation we took into account that the same library efficiently handles how graph data is represented and stored. Maximizing locality in this case was therefore not the goal, which lead us to focus on reducing computational complexity. The proposed design is relatively cheap, with a complexity of $O(N * k^2)$.

However, the cost of the memory access operations (*i.e.*, the number of memory misses) will be much larger as we are unable to store near-by vertices that will be visited consecutively.

Our experience suggests that algorithmic design should depend on the programming model being used, and that one should not be closed to changes of perspective. The particularities of large scale graph mining are so unique (*e.g.*, graph traverse) and relevant (*e.g.*, high-dimensional data structures) that traditional solutions and conventions may not apply to them. To the point where the same problem may have different optimal designs depending on the underlying model being used. This is on one hand a handicap, as it means one may not know which model is better until testing the appropriate design on it. But on the other is an important advantage, as it allows developers to explore the same problem from different perspectives seeking the optimal solution. All in all it is key to understand these models before using them, as an inappropriate design can make the best of programming models seem useless.

Another important lesson learnt is on the programmability of HPC models. From an AI programmer point of view learning current parallel models (at least of the ones tested in this thesis) is rather easy, and one can achieve efficient implementations in relatively short time. Similarly, shifting from a shared memory to a distributed memory context is largely simplified by these models nowadays; using either a shared or a distributed memory setting is transparent while using ScaleGraph, and the effort needed to transform an OpenMP code to an OmpSs code is minimal. As a result we expect both the HPC and graph mining community to benefit from these advances.

8.3.4 Computing Times

As discussed in §8.3.3, the two implementations described in §8.3.1 and §8.3.2 have different complexities and run on different hardware. Producing a consistent comparison between them is thus impossible at this point of our research. Let us however mention that both implementations achieved similar computational times. In order to provide a general idea of the computational time spent in our tests, see in Table 8.1 the computation times of both the OpenMP and the ScaleGraph implementation. The DBpedia graph could not be computed on a single TSUBAME node due to memory restrictions.

Web Graph	**MareNostrum time**	**TSUBAME time**
WordNet	63 seconds	736 seconds
Cyc	67 seconds	84 minutes
webND	513 seconds	250 seconds
webSB	60 minutes	5.3 hours
webGL	85 minutes	12.7 minutes
IMDb	6.3 hours	25 hours
hudong	8.6 hours	5.4 hours
baidu	20.7 hours	8.6 hours
DBpedia	48 hours	——

Table 8.1: Time spent on each graph used in one MareNostrum node for the OpenMP implementation, and in one TSUBAME node for the ScaleGraph implementation. Time includes reading the graph from a file, building internal data structures, calculating the scores of edges, computing the overall PR and ROC curves and writing the results.

The numbers of Table 8.1 show the computation times of both implementations following different functions. While the OpenMP code time grows based on the size of the graph (*i.e.*, in the number of vertices), the ScaleGraph code seems to follow a different rule, as smaller graphs are computed slower than bigger graphs (*e.g.*, Cyc and webSB, IMDb and hudong). We analyzed the properties of these graphs and found that the time for the OpenMP code seems indeed to be closely related with the number of edge missing from the graph. Since this code visits every missing edge, its computational time can be approximated through the number of missing edges. See Figure 8.4 for a chart showing this relation and a linear regression.

The ScaleGraph code does not compute all missing edges, as it focuses only on those that have a score bigger than zero. The size of this set of edges (the once having a score bigger than zero) is closely related to the number of actual edges in the graph; the more links a graph has the more different paths we will find. However we hypothesize that there is yet another graph property key to understand ScaleGraph computing times. Since ScaleGraph has a lower data locality (it does not store or explore the graph data in order), its performance may be affected more strongly by the existence of *superhubs*, vertices with an abnormally high number of relations. Also, the time spent by the ScaleGraph code building the two-steps neighborhood around those superhubs (as designed in §8.3.2) may add a significant overhead. We support

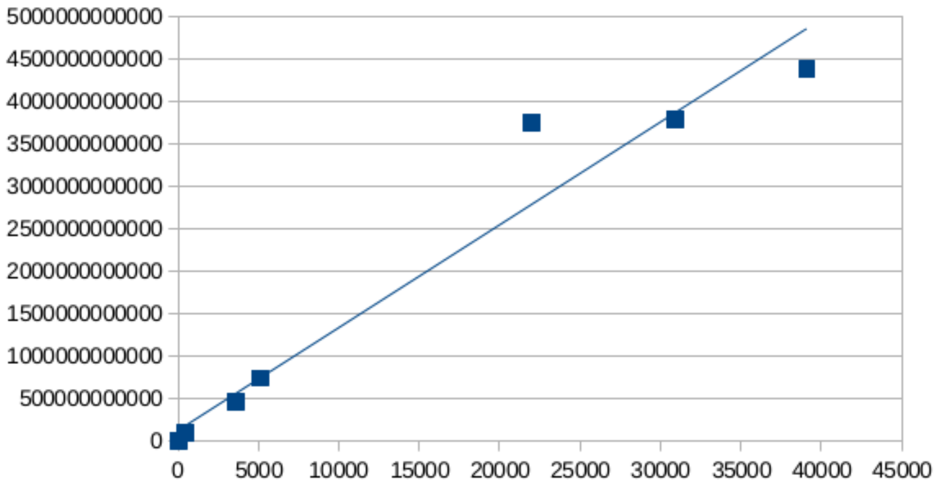


Figure 8.4: Computation time of each graph in the OpenMP implementation (x axis), and number of edges missing on each graph (y axis). Linear regression on both variables.

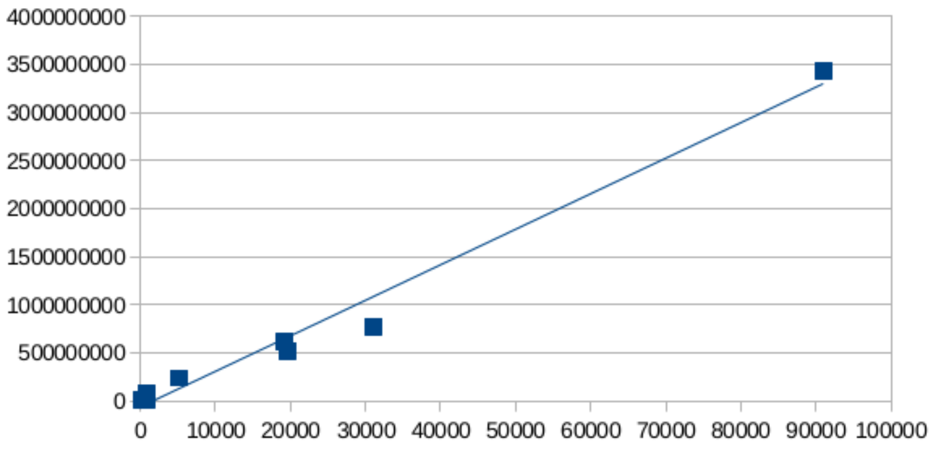


Figure 8.5: Computation time of each graph in the ScaleGraph implementation (x axis), and number of edges on each graph multiplied by the relevance of *superhubs* (y axis). Linear regression on both variables.

this hypothesis with Figure 8.5. In this figure we plot computation times against the number of edges in the graph *times* the importance of superhubs. We compute the importance of superhubs as the percentage of total edges in the graph related with the 1% of vertices with higher degree. The linear regression shown in Figure 8.5 seems to approximate the computational time rather well.

Our final remark on these results is on the impact of the models being used. Depending on the properties of each graph, one parallel programming model, and one algorithmic design may be more efficient. This further reinforces our notion that a thorough analysis of the specific graph problem and the available programming models is necessary prior to development.

8.4 Computational Resources

In this thesis we used two different computational contexts. For running the OpenMP/OmpSs implementation we used the MareNostrum supercomputer, provided by BSC. MareNostrum is based on Intel SandyBridge processors, iDataPlex Compute Racks, a Linux Operating System and an Infiniband interconnection. One MareNostrum node is composed by two Intel SandyBridge-EP E5-2670/1600 20M 8-core at 2.6 GHz, and a limit of 28.1 GB of RAM. This translates into 16 parallel threads. For running the ScaleGraph implementation we used the TSUBAME supercomputer, provided by Tokyo Institute of Technology through the JST CREST project. TSUBAME is based on Intel Xeon processors and NVIDIA Tesla, Infiniband interconnection and both Linux SUSE and Windows HPC Server. TSUBAME nodes are composed by two Intel Xeon X5760 6-core at 2.93 GHz, and a limit of 26 GB of RAM. This translates into 12 parallel threads.

Chapter 9

Conclusions

During the development of this thesis we encountered a wide variety of problems, and had to employ an equally varied set of solutions to tackle them. In this chapter we summarize the conclusions we have reached during this stimulating process. The conclusions regarding the two main goals defined in §1.1 are shown first in §9.1 and §9.2. After that we present other lessons learnt and contributions done in various aspects of the LP problem. Before that however, let us summarize our answers to the research questions made in §1.3.

- *What is the role of hierarchies in the topology of large graphs?* We found hierarchies in several different domains, in some of them explicitly (ontologies, lexical relations) and in some implicitly (webgraphs, movie relations). The universality of hierarchies for defining knowledge makes hierarchical properties available, with more or less significance, in a wide variety of domains. This motivates a case by case analysis of each domain prior to computation.

- *Can hierarchies be used to predict relations within a graph?* Indeed they can. Results show that assuming and exploiting hierarchical properties can provide a huge boost in predictive performance. INF, and particularly INF_LOG_2D, achieved remarkable predictive results, improving the current state-of-the-art by one or more orders of magnitude.

- *Are hierarchies implicitly represented within natural, informal graphs?* In some, they are. Hierarchies are so abstract and generic than many biologic (*e.g.*, neurons connectivity), sociological (*e.g.*, social network relations) and technological (*e.g.*, webgraphs) domains include them. We should be aware however, that not all graphs are hierarchical.

- *Is it feasible nowadays to massively run graph mining algorithms on large graphs?* Yes it is, even though they are computationally challenging. The particularities of graph mining makes it rather appropriate for HPC integration.

- *Are HPC infrastructure and tools ready to deal with the challenges of the emerging graph mining problems?* The HPC community is making a huge effort on its own in providing solutions for large scale graph mining. In this thesis we benefited from this effort. However to further improve this collaboration, the graph mining community should properly specify its needs in an HPC context (*e.g.*, type of algorithms, type of data sets, time constrains, *etc.*). Only then the HPC community will be able to provide more adequate solutions.

9.1 Improving Precision

As discussed in §1.2, one of the main goals of this thesis was to find ways to increase the precision of LP similarity-based scores. Our general approach to increase score precision is to exploit inherent properties of the graph. With this objective in mind we proposed the INF score that assumes the existence of a hierarchical structure within the topology of the graph, and uses this structure for the benefit of its predictions. In §7 we show how INF is capable of obtaining much higher precisions than other scores which do not make the same structural assumptions INF does. Furthermore, we saw how the assumption of a hierarchy is satisfied in a wide variety of domains, even in domains not purposely hierarchical, which extends the applicability of INF.

To further increase precision we propose two modifications to the INF score, 2D and LOG, and evaluated their performance in §7.1.4 and §7.2. The results of these modifications illustrates the importance of understanding the nature of the graph topology for predicting edges: by choosing the adequate score with the adequate modifications one can obtain a huge leap in precision. By analyzing the nature and results of each modification we defined similarity-based LP scores in three categories: proportional, accumulative and hybrid scores. Furthermore, in all our tests on informal graphs (IMDb and six different webgraphs), the hybrid scores (INF_LOG, INF_LOG_2D) always outperformed the proportional (INF, INF_2D) and accumulative (CN, RA, AA) scores.

A different approach to obtaining higher precision predictions is to focus on high-similarity edges. This implies a significant reduction in predictive recall, which is typically acceptable for real world applications. Formal evaluation methodologies however explore predictions up to a complete recall, which often implies a dramatic drop in precision due to the high underlying imbalance of large scale graphs. To reduce the weight of useless predictions we proposed a predictor evaluation methodology, the CAUC (see §5.3), which is biased towards predictors achieving high precisions. Finally, when evaluating link predictors using formal methods like CAUC we must also take into account that in most cases the test sets do not represent a large part of the edges missing from the graph (see §5.5). Hence, the precisions these formal methods calculate for predictors will be frequently underestimated.

9.1.1 Hyperlink Prediction

While trying to increase the precision of LP, we considered the particular problem of hyperlink prediction, *i.e.*, predicting links among web pages. Our results on this particular domain show the importance of hierarchical properties for the definition of hyperlinks. In our tests the INF_LOG_2D hierarchical score outperformed all non-hierarchical scores by a large margin on six different webgraphs (see §7.2). Even INF, a simpler hierarchical score which should a priori be handicapped by its lack of focus on high-degree vertices, achieved competitive results and outperformed all non-hierarchical scores on two of those webgraphs. These results align with the previous work discussed in §6.2.1.1, and provides further evidence on the importance of hierarchies for the topology of webgraphs.

The leap in precision quality obtained by INF_LOG_2D (see Figure 7.5) shows for the first time that even within graphs with millions of vertices, one can predict thousands of edges (those with higher score) with an almost perfect precision (*i.e.*, with very few false positives) and in a scalable manner. According to our results, hyperlink prediction becomes a feasible problem by exploiting hierarchies. This immediately enables multiple interesting cases of application, including but not limited to: increase and improve the connectivity of web pages, optimize the navigability of web sites, tune search engines results through web connectivity analysis, and refine product recommendation through item page linkage. All cases are of immediate application.

9.2 Improving Scalability

Besides improving precision, the other main objective we identified in §1 was scalability. Since some of the most interesting graphs to process are large scale graphs (see §10.1), LP needs to integrate with state-of-the-art HPC techniques and to become as computationally efficient as possible. Our contribution to LP scalability was presented in §8. Some of the techniques we found useful for increasing scalability were based on how data is represented, stored and processed (see §8.1). This is a topic of particular importance due to the data-intensive profile of the LP task. We also significantly improved scalability by reducing the precision of arithmetic operations. In large scale problems, like LP on large graphs, one can be permissive in certain aspects without significantly affecting the results (see §8.2).

However, what eventually defines the feasibility of LP on large graphs is its parallelization. We developed a parallel implementation using two different programming models. For the first, based on OpenMP and OmpSs, we focus on defining the LP problem as an embarrassingly parallel one, thus optimizing the benefits of parallelization. We also exploited data locality (through data storage and sorting) to maximize the efficiency of memory access operations. The second model used was Pregel, through its ScaleGraph implementation. The algorithmic design for this model was radically different, due to the graph oriented nature of Pregel. The code could be parallelized with the definition of a single synchronization point, where all graph traverse operations are done. Both implementations are available for both a shared and a distributed memory context. In OpenMP through OmpSs, and in ScaleGraph through its native support.

9.3 Overall Test Results

From the tests performed and shown in §7 we conclude that RA is the most reliable local LP score of those so far presented in the bibliography, as it achieves the best PR-AUC results on six of the nine graphs tested. If one looks at the ROC-AUC measure instead, AA outperforms RA in some graphs (*e.g.*, WordNet, Cyc and IMDb). Regardless, as discussed in §5.1 PR curves are a more faithful representation of predictive performance than ROC curves, particularly in the context of applicability. The conclusion that RA outperforms AA and CN is coherent with previous analysis (see §2.1.2).

Our tests also show that INF is a very competitive score, as it outperforms CN, RA and AA on five of the nine graphs tested. The good results of INF on the hierarchical graphs can be easily explained, as INF seeks the same hierarchical principles. More interesting is the fact that INF largely outperforms CN, RA and AA on large, informal, apparently non-hierarchical and potentially bi-directed graphs like the webgraph or IMDb. By including the two proposed modifications, the INF_LOG_2D becomes the best score on all graphs tested. These results suggest that INF and its derived scores are appropriate for a wide variety of domains.

To also elaborate on the weaknesses of our approach, we expect the performance of INF to be significantly worse on graphs with more chaotic structures, without a hint of hierarchy in their topology. INF may be inadequate for example to predict road links, as the road system does not include hierarchical properties. Regardless, domains may have to be analyzed one by one in order to decide if INF should be applied or not. In the case of social networks for example, we expect INF to perform very well on a Twitter graph data, as followed/follower relations are rather hierarchical. Other social networks, such as Facebook friendships, may not be so suitable as they include a weaker sense of hierarchy. The application range limitation of INF may also apply to smaller and sparser graphs that the ones used here, where a general hierarchical topology is not properly captured. Nevertheless, given how universal hierarchies are at defining knowledge, we expect to find lots of domains appropriate for INF. Finally, even though INF requires graphs to be directed, methodologies for transforming an undirected graph into a hierarchically directed one exist (Clauset et al., 2007).

9.4 Hierarchical Knowledge for LP

One of the goals of this thesis was to evaluate the importance of hierarchies in defining and predicting the topology of large, directed graphs. For that purpose we compared the results of the undirected scores RA, AA and CN with those of a hierarchical versions of those same scores, HRA, HAA and HCN proposed in §4.2. HRA, HAA and HCN scores are identical to RA, AA and CN in all cases but when edges are *anti-hierarchical* (*i.e.*, edges going from abstract vertices to more specific vertices). Hence we understand the differences in performance between these two sets of scores as a direct indicator of how importance the concept of *node specificity*, and thus hierarchies, is for LP.

Hierarchical scores HRA, HAA and HCN achieve an average improvement in PR-CAUC over RA, AA and CN of 409% on WordNet, and of 183% on Cyc. This could be expected due to the explicitly hierarchical structure of these graphs. More relevant is the difference in performance achieved on the IMDb graph: an average improvement of 131%. This graph composed by movie-related relations is not hierarchical by design, but its topology seems to be based on a hierarchy nonetheless. To understand that behavior consider how the most referenced movies (*e.g.*, Metropolis, The Godfather, *etc.*) become more influential with every added reference, and are therefore more likely to be referenced in the future. This hierarchical behavior is not limited to a particular domain, and can be exported to others such as social networks, biological data or webgraphs.

In the case of webgraphs, the importance of hierarchies has already been studied (see §6.2.1.1). Our results clearly support that claim (see §7.2). Hierarchical knowledge has been used for example by the forest fire model (FFM) proposed by Leskovec *et al.*, briefly described in §6.2.1.1. The methodology through which the FFM adds vertices and edges to the graph contains no explicit hierarchy, and yet it generates data which is hierarchically structured. To explain that we realized the link adding process of FFM for the out-going edges of y is in fact a particular case of the DED sub-score (see Figure 4.1), while the link adding process for the in-going edges of y is a particular case of an *abductive* score (ABD) (defined in §4.3). As discussed before, ABD was not used because preliminary tests showed it to be unstable, adding a lot of noise and bias to DED and IND. Remarkably, the FFM seems to acknowledge the same higher reliability we identified on DED, as out-going edges of y are chosen with a higher probability than the in-going ones. All these unpremeditated similarities shows that both INF and FFM are based on close hierarchical principles.

Regardless of the similarities between FFM and INF, the scope and the scale of both algorithms is completely different. This dissimilar scale results in important differences on their applicability. INF considers in-edges of vertex x to determine its out-edges (*i.e.*, IND score), while the FFM does not. INF calculates a similarity score for each edge by looking at all the available evidence, which allows it to rank edges based on their likelihood. The FFM on the other hand randomly accepts and rejects edges as it does not seek faithfulness at a vertex-level; it seeks topological coherency at a graph level. Finally, the FFM explores edges far away from the ambassador vertex though various iterations, thanks to its computational simplicity. INF cannot do that in acceptable time due to the higher computational cost of calculating similarities, and performs only a one step exploration (*i.e.*, its a local score). Regardless, a a quasi-local version of INF was already defined in §4.1.2.

To sum up, while the FFM and INF share a set of precepts, each model uses those for a different purpose: the FFM uses them to define a large, coherent topology model at graph scale, while INF uses them to define a high confidence and exhaustive edge likelihood score applicable at vertex level. In that regard, the good results achieved by both methods on their respective fields of application seems to partly support the assumptions of the other.

9.5 Impact of Imbalance

Local scores of LP use local graph features to evaluate the existence of new edges. Local features such as the degree of vertices, the number of shared neighbors, and even local hierarchical structures (as used by INF). If one wants to estimate the overall performance of a local score on a given graph without computing every single local feature, one can consider global features instead (*e.g.*, average degree, class imbalance) as these combine and generalize local features. Since global features are unique for the whole graph computing them is typically faster, and can still be used to estimate the predictive quality to be obtained by local scores with rough accuracy.

Estimating the performance of predictors through global structural properties is possible as long as those global properties are coherent with the design of the predictors. For example, a global *graph hierarchical measure* would be most appropriate to estimate the performance of the family of hierarchical scores INF. Hence, each predictor may be better estimated using a different global measure. Nevertheless, one can still use a single generic graph measure to estimate the performance of all predictors at the cost of estimation accuracy. Of the generic graph measures relevant for all LP scores we find that the most appropriate one is class imbalance. In other words, the larger the difference between the number of non-existing and existing edges, the lower predictive

Graph	**class imbalance ratio**	**Max PR-AUC obtained**
Wordnet	1:11,382	0.8393
Cyc	1:39,496	0.0735
webSB	1:61,775	0.4515
webND	1:70,867	0.5264
webGL	1:150,217	0.4921
baidu	1:257,667	0.0052
hudong	1:264,848	0.0342
IMDb	1:1,140,835	0.0326
DBpedia	1:2,151,672	0.0012

Table 9.1: For all graphs, positive:negative ratio and best predictive rate of all tested local similarity-based scores (CN, AA, RA, INF, INF_2D, INF_LOG and INF_LOG_2D).

quality we can expect for any LP score. The relation between class imbalance and maximum predictive performance for all graphs tested here can be seen in Table 9.1.

Class imbalance increases as the maximum PR-AUC decreases, although variations exist. This is coherent with what was discussed in §3.1, as a larger class imbalance makes the classification problem harder, implying worse results. We plotted both variables in Figure 9.1 (see blue squares), and also produced a regression to it, a power law function being the best apparent fit. According to this function, one could argue that predicting edges on graphs having a degree of imbalance larger than a given threshold is useless, as the maximum predictive score trends to zero. We rebate this argument from two different perspectives.

First of all, as discussed in §7.3, the standard evaluation methodologies based on the PR or ROC curves may not be appropriate to estimate the applicability or utility of the LP scores. For example, one may have a bad PR-AUC score and yet produce a set of correct high-certainty predictions large enough as to be of use. Thus, class imbalance may be strongly correlated with the maximum overall predictive results, but not necessarily with the applicability of some of those results.

Secondly, the definition of more precise predictors may significantly modify the trend function, and by extension the estimated performance of LP scores. As an example of this last situation consider Figure §9.1, where the data points resultant of considering only the results achieved by the previous state-of-the-art (*i.e.*, CN, RA and

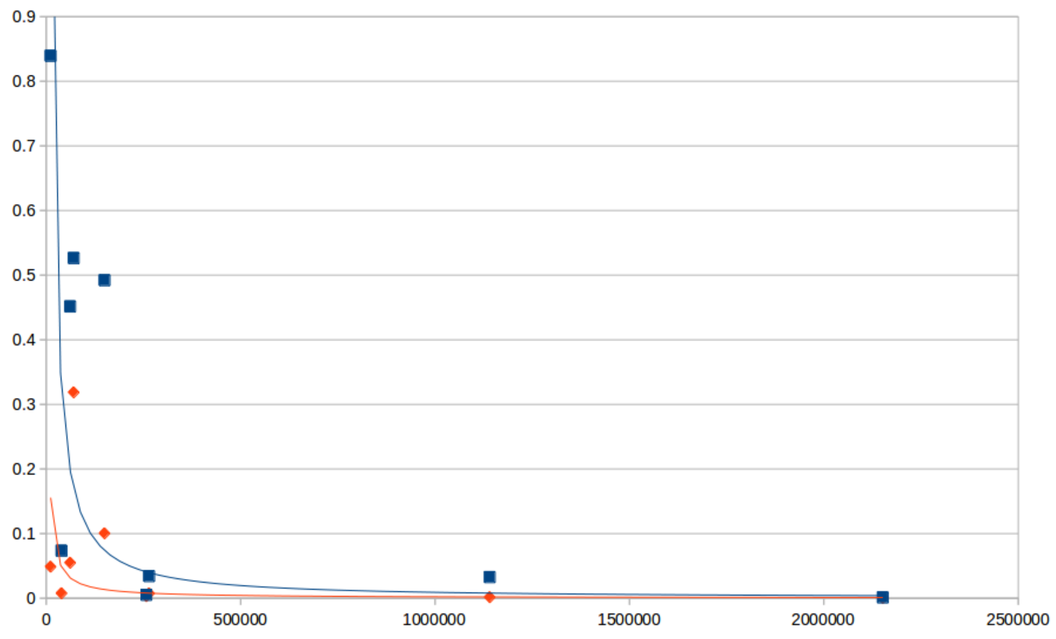


Figure 9.1: Chart of second and last column of Table 9.1 (blue squares). The same chart as orange rhomboids when considering only results of CN, RA and AA. Power law regression for both cases.

AA, not considering the results of INF scores) are also shown as orange rhomboids. The improvement in the trend function provided by our contribution INF is clearly visible, effectively increasing the size of graph that can be fully explored with acceptable results. It is therefore coherent to expect that extremely imbalanced graphs where current predictors perform poorly may be solvable in the near future through new and more precise scores.

9.6 Similarity-based, Maximum Likelihood and Hybrid Scores

The INF score proposed inhere has certain unique particularities. For example, even though INF independently computes a score for every pair of vertices, and is therefore a similarity-based score by definition, it assumes and exploits a model (a hierarchy), like maximum likelihood algorithms do. By using the properties of this underlying model INF can achieve better predictions in domains where those properties are satisfied, as its understanding of the domain topology is increased. However, since INF does not *compute* the model (it takes it for granted), INF does not suffer the computational problems derived from building a model for the whole graph; maximum likelihood algorithms typically scale poorly because of the cost of building a graph model. In this regard we consider that having the predictive advantage of including a model, while avoiding the limitations derived from of building it, is one of the most relevant features of INF. We believe this methodology can be exported to other domains, where different models are inherently represented (*e.g.*, communities) and can therefore be implicitly exploited.

Another particularity of INF is that, while most similarity-based scores are accumulative, INF is proportional. Proportional scores evaluate the likelihood of an edge in proportion to the local evidence available, that is, they normalize the score according to the local context. Accumulative scores on the other hand do not normalize the score, and thus rate edge evidence from a graph wide perspective. This difference has serious implications because, as discussed in §7.2, proportional scores do not prioritize high-degree vertices (which are often more reliable due to the rich get richer principle), and do not neglect low-degree vertices (which are often less reliable due to arbitrariness), like accumulative scores do. INF was defined as a proportional score because its original purpose was to define a measure of hierarchical affinity between two vertices based on principles of inferential reasoning, affinity represented as a numeric value between 0 and 2. Thus, properties only found in proportional scores, like having a maximum hierarchical self-affinity, were desirable: an edge connecting a vertex with itself always has maximum INF score $s^{DED}_{x \to x} = 1$ and $s^{IND}_{x \to x} = 1$ (also $s^{ABD}_{x \to x} = 1$). This property can be explained by the fact that an element perfectly generalizes and specializes itself.

Proportional scores like INF are, for the reasons previously explained, severely penalized when applied to large, sparse, informal graphs obtained from real world networks. Nevertheless we consider both perspectives (accumulative and proportional) to be relevant for the LP problem. Hence, we decided to combine both. We did so through the LOG modification transforming INF into a hybrid score (INF_LOG and INF_LOG_2D), thus taking into account both the vertex context and the graph context. Remarkably these two hybrid scores obtained the best results on every informal graph tested (six webgraphs and IMDb). Although this is clearly insufficient to argue that hybrid scores are superior to the others, it is significant enough as to encourage further

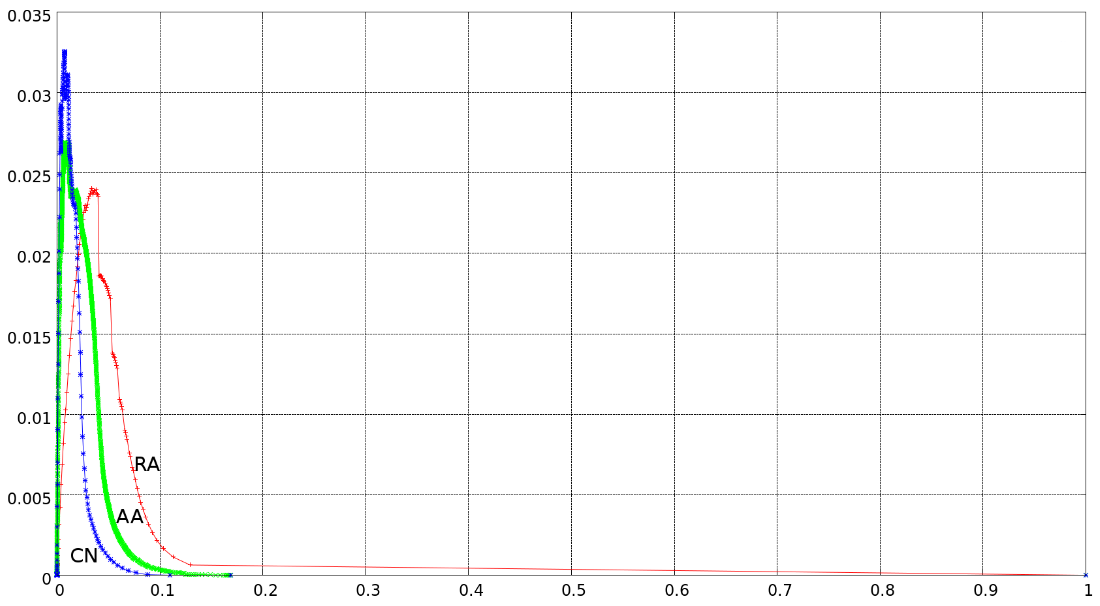


Figure 9.2: PR curve of RA, AA and CN scores on the IMDb graph.

research on this type of LP score, for example by building new hybrid scores from the accumulative scores CN, AA or RA.

9.7 CAUC vs AUC

In §5.3 we proposed a modified evaluation methodology for predictive performance, PR-CAUC, based on the more conservative PR-AUC with the goal of focusing on applicability. To evaluate the effect of our proposal consider the PR curve of local scores RA, AA and CN on the IMDb graph, shown in Figure 9.2. RA outperforms CN in both PR-CAUC and PR-AUC, but the PR curve indicates that CN is better for high certainty predictions. Indeed, while CN achieves precisions of 3.5% RA reaches only precisions around 2.4%. By considering the whole area under the curve of Figure 9.2, RA outperforms CN in PR-AUC by 231%. On the other hand, by focusing on the high certainty predictions RA outperforms CN in PR-CAUC by 182%. Even though the RA beats CN according to both evaluation methodologies, the difference of 49% between the CAUC and the AUC measures shows the potential impact of our proposal.

Another relevant feature of the CAUC measure is that it adapts to the graph under evaluation. CAUC defines the threshold stating which predictions are relevant and which are not according to each graph size. Simply put, CAUC does not take into account predictions made after more mistakes are done than actual edges in the graph. As a result, more predictions will be demanded for graphs with more edges. AUC on the other hand evaluates the predictions done on a graph with N vertices and 1000 edges and the predictions done on a graph with N vertices and 100,000 edges under the same conditions, as if these two problems were equally difficult. A clearly unrealistic assumption that may lead to the underestimation or overestimation of results.

Chapter 10

Future Work

The work presented in this thesis focuses on the application of LP to directed graphs, a problem not thoroughly tackled in the bibliography so far. The questions arisen from this work open up many possible lines of future work, not only in directed LP but in graph mining in general. In this chapter we discuss several of those future lines of work, as identified throughout the development of this thesis. We expect that some of them will gain attention from the scientific community in the coming years.

10.1 Application to Large Graph Domains

In this thesis we tested graphs obtained from different domains: an ontological taxonomy (Cyc), a lexical taxonomy (Wordnet), movie related connections (IMDb) and several webgraphs. Even though the domains were chosen to represent a variety of fields, we acknowledge the necessity to keep exploring more and more domains of application. Thus, one of the planned lines of future work is to test LP on other domains, particularly in those which offer useful applications (*e.g.*, commercial product recommendation, support to biomedical research, *etc.*). Beyond applicability we also intend to focus on domains providing large amounts of data, so that large scale graphs can be built. In this section we discuss two fields we intend to prioritize, as they provide both applicability and large amounts of data: social networks and biological graphs.

Social networks are large scale graphs which keep getting larger. The graph obtained from follower relations in Twitter ($follower \rightarrow followed$), considering each user as a vertex, evolved from 41.9 million vertices and 1,470 million edges on July 2009, to 465.7

million vertices and 28,700 million edges on September 2012 (Watanabe and Suzumura, 2013). Like Twitter, there are many other social networks from which large directed social graphs of interactions could be extracted. In that regard, the application of LP to these graphs would be straightforward by producing follower/friendship/colleague recommendations.

Large scale graphs obtained from biological domains is another field of application we consider. Protein-protein interactions (Cao et al., 2013; Von-Mering et al., 2002) for example define directed graphs, and play an important role in pharmacological drug design. As protein-protein interactions are expensive to validate through experimentation in laboratories, scientists in the field would benefit from a support tool proposing them pairs of proteins particularly likely to be related (*i.e.*, top-link prediction).

Another large scale biological graph can be obtained from connectome data collection (*i.e.*, a map showing a neural wiring diagram). This currently hot topic is already producing very detailed graphs of neurons (Helmstaedter et al., 2013) of small subparts of the brain. Due to the existence of seemingly bidirectional relations (electrical synapses) and to the limitations of current exploratory technology (Bullmore and Bassett, 2011), most connectome graphs are undirected so far. However, connectome directionality has been acknowledged as an important feature, and is one of the target of current technological innovation in connectomics. From our point of view, the possibility of computing a graph representing a significant portion of the brain is inspiring. The brain is the most complex and relevant graph we can currently aspire to work with, and the benefits of understanding and predicting brain connectivity seem endless.

10.1.1 Exploiting Hyperlink Prediction

In §7.2 we tackled the problem of predicting hyperlinks among webpages, achieving remarkable results. Once we have recognized the capability of our proposal at modeling relations among webpages, we intend to bring this knowledge to real world applications. The scalability and parallelization aspects of this thesis are key for that purpose, as the web graph can reach even larger sizes than social networks; a web graph built from a crawl performed in 2012 (Meusel et al., 2014) contained 3,830 million pages as vertices and 3,530 million links among those. The possible lines of application we consider within the field of the WWW are the following:

- *Web search engines.* Current search engines are composed by a wide variety of interacting metrics, which together produce a complete ranking of web relevance. INF and its derived scores provide a measure of hierarchical similarity between webpages, which may enrich this ranking from a different perspective. For example, if the top five ranked webpages are all found to be hierarchically similar to the same webpage, this webpage could be assumed to be more relevant than initially anticipated.

- *Web connectivity.* Hierarchical properties spontaneously emerge in the WWW at a global level (§6.2.1.1), but at a local level things are rather chaotic. Each webmaster must find appropriate webpages to link to, in a domain with billions of possible targets. As humans cannot be aware of every single webpage online, a hyperlink recommender could be used to find relevant web pages to a website given its previous pattern of relations. This tool could improve both the connectivity and coherency of the WWW, as well as enrich the contents of directory webs.

- *Bottom-up taxonomy building.* Taxonomies are frequently used in online shops and encyclopedias (among others) to organize their content. These taxonomies are often externally defined by experts, who then must fit the data (*e.g.*, webpages) to it. According to the results of this thesis it seems feasible to build a system producing an automatic taxonomy of web pages based on their interrelations. Such a taxonomy would have the benefit of being based on the data, making it necessarily relevant for the domain in question. This taxonomy could be used to optimize the commercial organization of an online shop, for example by considering user navigation paths as source for the LP algorithms.

10.2 Graph Mining with HPC

As discussed in §8.3.1.4, the mining of large scale graphs requires of a set capabilities currently available only in HPC environments. The size of the data to be processed requires huge amounts of memory, and the number of independent computations to be performed requires lots of computational units. Thus, one of the most important future lines of work of this thesis will be to strengthen the interaction between the fields of graph mining and HPC. This view is already shared by many HPC researches, as demonstrates the introduction of a benchmark to rank supercomputers based on how they process large scale graphs (Graph500[1]). In this benchmark supercomputers are ranked based on the number of traversed edges per second achieved in a breadth first search, when processing graphs up to 68,000 million vertices (Ueno and Suzumura, 2012). The development of graph-specific parallel programming models (discussed in §8.3.2) also demonstrates the growing interest of the HPC community in graph processing. To strengthen the collaboration between HPC and graph mining we intend to explore other parallel programming models, like the promising path centric model proposed in (Yuan et al., 2014). Evaluating which models works better for which algorithm is an area of great interest, as it will potentially allow us to compute large graphs like the ones proposed in §10.1 in the short term. Further optimizing and extending our OmpSs implementation (see §8.3.1.4) is also a priority.

Beyond the acknowledged necessity of integration between graph mining and HPC, there is as well a natural harmony between both fields. As shown in §8.3, the LP problem ca be parallelized in a very efficient manner, defining an inconsequential few dependencies. This efficient parallelization was done on two different programming models, which indicates that the problem itself is parallel efficient. An important future work to consider is to explore the parallelization of other graph mining tasks and algorithms, as it appears to us that the local and distributed nature of graph mining algorithms (which independently consider subparts of the graph for its computation) will fit the parallel computation paradigms in most cases. This includes challenges such as community detection, frequent sub-graph discovery and node discovery.

10.3 Optimizing INF Parameters and Quasi-local INF

The main goals of the tests performed in this thesis (see §7) were the evaluation of the INF score, the comparison of its performance with other state-of-the-art LP scores, and the analysis on the applicability of hierarchical assumptions to informal, directed graphs. Both the code and the tests were designed for that purpose. As a result, conclusive results on the evaluation of quasi-local scores were not reached. Even though a quasi-local version of the INF score was defined in §4.1.2. At the same time, a detailed study of INF parameters was not completed.

[1]http://www.graph500.org/

The modifications proposed for INF (see §4.1.1) either increase the importance of a part of INF (2D: DED twice over IND) or weight the amount of evidence obtained (LOG: through a logarithm). However these two parameters were not sampled for optimal value, and may seem to be rather ad-hoc. Indeed, we choose the simplest of parameters (multiplying by two, and a logarithm base two) because our general purpose was to show how prioritizing a certain type of hierarchical inference and adding weighted measures could improve performance. We did not intend to find the optimal value of these parameters for all graphs tested, also because results indicate the optimal value depends on the domain, thus requiring a case by case sampling process not fully relevant for our hypothesis. Nevertheless, doing such a study and finding the range of parameters more frequently optimal is a necessary future work we will be doing when applying the INF algorithm to real cases.

As discussed in §2.1.3, and according to the current state-of-the-art, quasi-local scores are the most promising approach to similarity LP. Nevertheless, quasi-local scores originate from local scores, which remarks the importance of the work done in this thesis. To thoroughly evaluate the performance of quasi-local scores one needs too sample the number of steps performed, as well as the optimal value of other parameters such as the decaying factor (discussed in §2.1.1). For large scale graphs though, this work is particularly challenging, as the cost of quasi-local scores may increase exponentially with the number of steps taken (see §3.5). Thus, tests and code should be specifically designed for maximizing the efficiency of quasi-local computation, for example by sacrificing evaluation faithfulness though a sampling process (see §5.5). We intend to make of this line of work a priority in the near future, as quasi-local scores are the most likely ones to be applied to real problems. Their integration with distributed computing systems (as proposed in §10.2) will be an issue to be considered.

10.4 Weights

In §3.3 we discussed the possibility of using weighted graphs in LP, and why, for the sake of simplicity, we discarded it. We nevertheless consider weights to be a key feature for prediction refinement, and intend to integrate them with INF in the future. A first step towards this integration would be to add weights to all the possible edges in the graph (not only the currently existing ones) as provided by Definitions 7, 8 and 21.

Either as three different values or as a weighted average. The resultant weighted graph would capture, for each edge, its hierarchical support.

A LP process performed on that weighted graph could use the pre-calculated weights to balance the evidence provided by each vertex based on its hierarchical affinity. Hence, to determine if vertex y is the ancestor of vertex x one could consider not only how many and in which proportion are the ancestors and descendants of x descendants of y, but also to which degree. A first formulation of a weighted DED score in that manner could be

Definition 22

$$s^{WDED}_{x \to y} = \frac{\sum\limits_{\forall z \in A(x) \cap D(y)} s^{WDED}_{x \to z} \cdot s^{WDED}_{z \to y}}{|A(x)|}$$

This formulation defines the WDED score recursively, making it dependent on the whole graph topology. Notice, however, how the impact that other weights have on a given edge decreases as the distance from that edge increases. Thus, even though the WDED score is a global score by definition, its computational cost would be similar to a quasi-local score where the decaying factor eventually makes further steps irrelevant. This line of research drives us towards the definition of a purely hierarchical graph, where all relations of the graph are weighted according to a hierarchical topology. We further discuss this concept in §10.5

10.5 Purely Hierarchical Graph

Our proposed LP score (see §4) assumes that the semantics of a vertex are defined both by the semantics of its ancestors and its descendants. Results indicate that this idea is partly satisfied in the graphs tested. However, it is clear that none of the graphs used were designed with this intention. The simplicity and yet expressive power of hierarchies motivates a very interesting future line work in which a purely hierarchical knowledge base is built according to specialization and generalization principles.

In that regard we consider the integration of our methodology with a deep learning system to build a transversal reasoning engine. Deep learning systems produce high level, symbolic abstractions from given a set of low level, sub-symbolic inputs. An example consider DeSTIN (Arel et al., 2009), a deep learning system capable of discovering new shapes (symbolic abstractions) from an input of pixels (sub-symbolic data). The symbolic output of a system such as DeSTIN could be used as input of a LP system, which in turn can perform learning at a symbolic level based on contextual relations (*e.g.*, temporal or spatial co-occurrences). The resultant engine would then be able to produce abstractions from low level inputs, and to learn about the relations of those self-produced abstractions. Such an engine would probably need to include a node discovery process (see §10.6) to further populate the symbolic hierarchy.

10.6 Node Discovery

Through the analysis of the LP problem we have identified a closely related task of graph mining: the Node Discovery (ND) problem of finding new vertices within a graph. Predicting edges among entities though LP is an important learning task as it discovers previously unknown relations. However, the capability to find new edges is limited to the immutable number of vertices available in the graph. For extending the graph not only in edges, but in vertices we need to address the ND problem.

In that regard we consider the principles behind the design of the INF score to be applicable to ND. Informally, the idea is to find sets of vertices in the graph which share hierarchical properties (as represented through their edges) which have no generalization implementing that pattern. A ND algorithm could add new vertices to the graph to represent those patterns, thus completing the hierarchy defined by the topology and increasing the coherency of the graph. As an example consider the reasoning process done by a person who sees a road for the first time in her life. All cars, motorcycles, trucks *etc.* are different, but they share a set of properties. These relations (*e.g.*, noisy, fast, has wheels, materials, size *etc.*) will eventually trigger the formation of a new concept within the mind of that person: *motor vehicles*.

Chapter 11

Summary

Let us end this thesis proposal with a summary of the work presented. The first chapter §1 starts by introducing an approach to ML were data is understood as a network of connected entities. This new strategy uses and seeks inter-entity information for knowledge discovery, in contrast with traditional intra-entity approaches where the basic concepts are instances and their features. The importance of this connectivist ML (which we refer to as graph mining) to solve problems naturally expressed through patterns of relations is argued, particularly in the current context where large, topology-based data sets have been made available. The chapter ends by introducing the Link Prediction (LP) problem, one which favors its consideration from a perspective of data interconnectivity. The potential benefits of using LP are outlined, as well as its current computational and performance limitations. The main goals of this thesis proposal are also defined.

Chapter §2 discusses early contributions to graph mining, and introduces problems frequently tackled through this paradigm. After that the chapter focuses on the specific state-of-the-art of LP. It first presents three different approaches to the problem of discovering new links within a relational set, and argues about the importance of the most computationally scalable one: similarity-based algorithms. It further categorizes similarity-based algorithms in three types of LP scores based on the depth of their graph exploration. For the most scalable type, local similarity-based algorithms, the chapter identifies and formally describes the three most competitive proposals according to the bibliography.

Chapter §3 is dedicated to the analysis of the LP problem. LP can be reduced to a classic binary classification problem, and the benefits and dangers of doing so are described at the beginning this chapter. Then a list of graph properties such as directionality, weights and time are discussed in the context of LP. Follows a formal time and space complexity analysis of the similarity-based scores of LP as support for posterior decisions. The chapter ends with an study of the class imbalance typically found in LP problems.

In chapter §4 a novel similarity-based score of LP is introduced. For that purpose the chapter first elaborates on the importance of hierarchies for representing knowledge through directed graphs. Several modifications to the proposed score are also defined, including one which makes the score consider a variable size of the graph for its predictions. Using the same assumptions which motivated the design of our proposed score, this chapter introduces a modified version of the most competitive undirected scores of LP, identified previously, to adapt them to directed graphs.

The evaluation methodologies of LP scores are analyzed in §5. It starts by discussing the problem of evaluating domains with a huge class imbalance, identifying the most appropriate methodologies in that context. A modification of the most appropriate evaluation methodology according to the bibliography is presented, with the goal of focusing on relevant predictions. Follows a discussion of other issues related with the faithful estimation of the precision of predictors.

Chapter §6 describes the graphs used for score evaluation, as well as how the data was transformed into a directed graph. Reasons on why these particular domains were chosen are given, making a special case of webgraphs and their well known relation with hierarchies. The most basic properties of each resultant graph are shown.

All tests performed in this thesis proposal are presented in §7. The three most competitive LP scores currently available are first tested among themselves, and then against a proposed version of those same scores for directed graphs. Our proposed score and its modifications are tested against the scores obtaining the best results in the previous tests. The case of LP in webgraphs is considered separately, testing all computationally feasible algorithms on six different webgraphs. The chapter ends with a discussion on the limitations of this formal analysis, on how other less formal approaches could result in interesting applications, and by showing examples of predictions obtained.

Chapter §8 includes the computational aspects of the work done. The chapter starts with a discussion on the importance of memory management for determining the computational cost of LP algorithms. A proposal on how to reduce this cost through precision reduction is presented afterwards, and argued in the context of LP for large graphs. Follows a section focused on the parallelization of code, which includes two different implementations on one graph-specific programming model and on one generic programming model. The chapter ends with a specification of the computational resources used for the tests done.

The conclusions of this thesis proposal are presented in §9. First the main issues stated in the introduction are tackled, and then several other relevant findings are outlined. Some of those refer to the results obtained by our proposed score and its modifications, while others address the LP problem in general.

Finally, chapter §10 contains several future lines of work.

11.1 Published Contributions in the Context of this Document

- Dario Garcia-Gasulla and Ulises Cortés, *Hierarchical inference on semantic graphs applied to Cyc*, Proceedings of the 16th International Conference of the Catalan Association of Artificial Intelligence, 2013.

  This paper presents preliminary results of the INF score on the Cyc ontology, and argues about the variety and potential impact of the predictions achieved. Its results and conclusions were precursory to the work here presented, and are not shown in this thesis proposal.

- Dario Garcia-Gasulla and Ulises Cortés, *Link Prediction in Very Large Directed Graphs: Exploiting Hierarchical Properties in Parallel*, 3rd Workshop on Knowledge Discovery and Data Mining Meets Linked Open Data - 11th Extended Semantic Web Conference, 2014.

  This paper shows results on WordNet, Cyc, as well as a first test on the DBpedia graph. It also introduces some ideas regarding the optimization of code. Thus, it covers a portion (approximately a 20%) of §7 and of §8.

- Dario Garcia-Gasulla , *La febre i el poder de les dades*, ACIA Nodes num. 55 , 2014.

  This informational article written in Catalan discusses past trends in computer science research, and how current graph mining research such as Link Prediction represents a coherent evolution of those.

- Dario Garcia-Gasulla and Ulises Cortés, *Scalable Hyperlink Prediction for the WWW*, submitted to the IEEE International Conference on Data Mining, 2015.

  This article, currently under evaluation, explores the problem of hyperlink prediction. Explores the relationship between hiearchies and the WWW, and evaluates the performance of the INF_LOG_2D method on several webgraphs.

- Dario Garcia-Gasulla and Ulises Cortés, *Evaluating Link Prediction on Large Graphs*, submitted to the 18th International Conference of the Catalan Association for Artificial Intelligence, 2015.

  This paper, currently under evaluation, explores the issue of faithfully evaluating the prediction of link prediction algorithms when applied to large scale graphs. The utility of measures like 10-fold-CV and ROC curves are discussed, and specific solutions are proposed.

Appendix A

Appendix I

In §7 two sets of tests were shown. The first one evaluated RA, AA, CN, HRA, HAA, HCN, INF, INF_2D, INF_LOG, INF_LOG_2D, INF_QL, INF_2D_QL, INF_LOG_QL and INF_LOG_2D_QL on three different graphs (WordNet, Cyc and IMDb) using the PR-CAUC measure. The second one evalated RA, AA, CN, INF, INF_2D, INF_LOG and INF_LOG_2D on six different webgraphs using the PR-AUC measure. In this appendix we complete those tests by showing as well the complementary PR-AUC, PR-CAUC and ROC-AUC measures for both tests. These results are added for possible reference and evaluation. Figures with the actual PR and ROC curves can also be found in this appendix, including a zoomed-in version when considered necessary.

A.1 ROC/PR curves, ROC/PR-AUC: Undirected Scores

Score	**WordNet PR-AUC**	**Cyc PR-AUC**	**IMDb PR-AUC**
RA	<u>0.048763</u>	<u>0.00764913</u>	<u>0.0015387</u>
AA	0.022618	0.00608318	0.0009651
CN	0.006414	0.00145343	0.0006652

Table A.1: PR-AUC score of RA, AA and CN on three graphs.

Score	**WordNet ROC-AUC**	**Cyc ROC-AUC**	**IMDb ROC-AUC**
RA	0.97189	0.70480	0.564618
AA	<u>0.98538</u>	<u>0.79012</u>	<u>0.567304</u>
CN	0.96740	0.77651	0.566015

Table A.2: ROC-AUC score of RA, AA and CN on three graphs.

A.1.1 Undirected Scores on WordNet

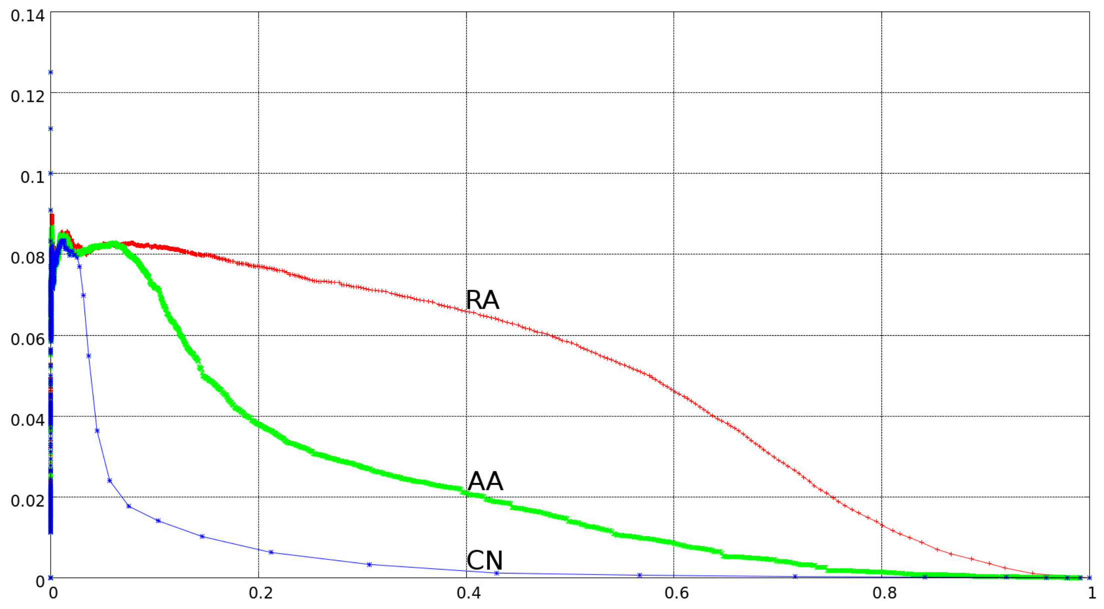


Figure A.1: PR curve of RA, AA and CN scores on the WordNet graph.

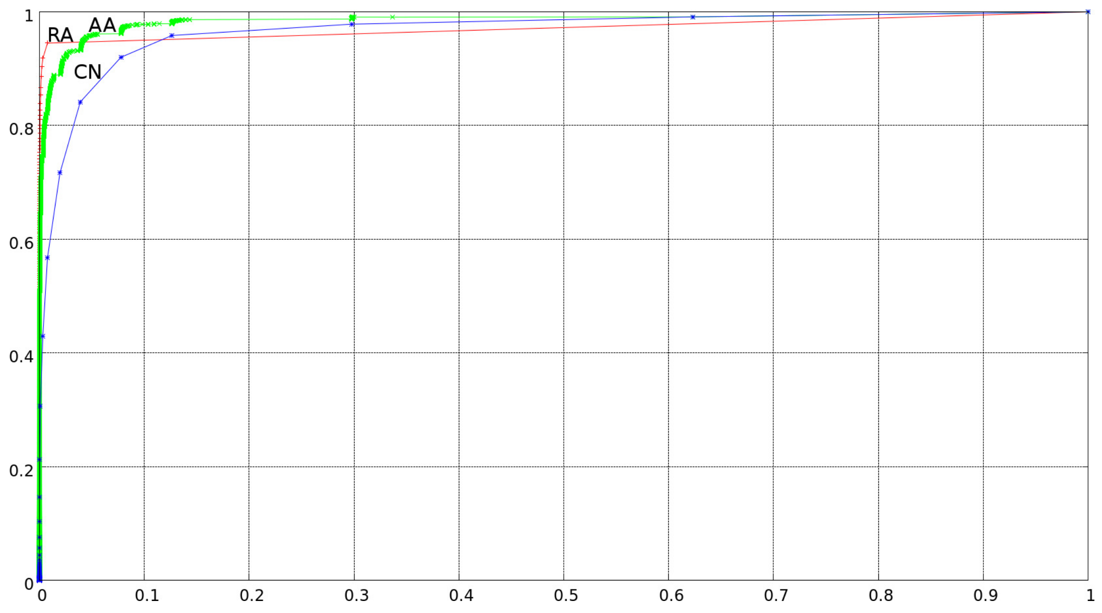


Figure A.2: ROC curve of RA, AA and CN scores on the WordNet graph.

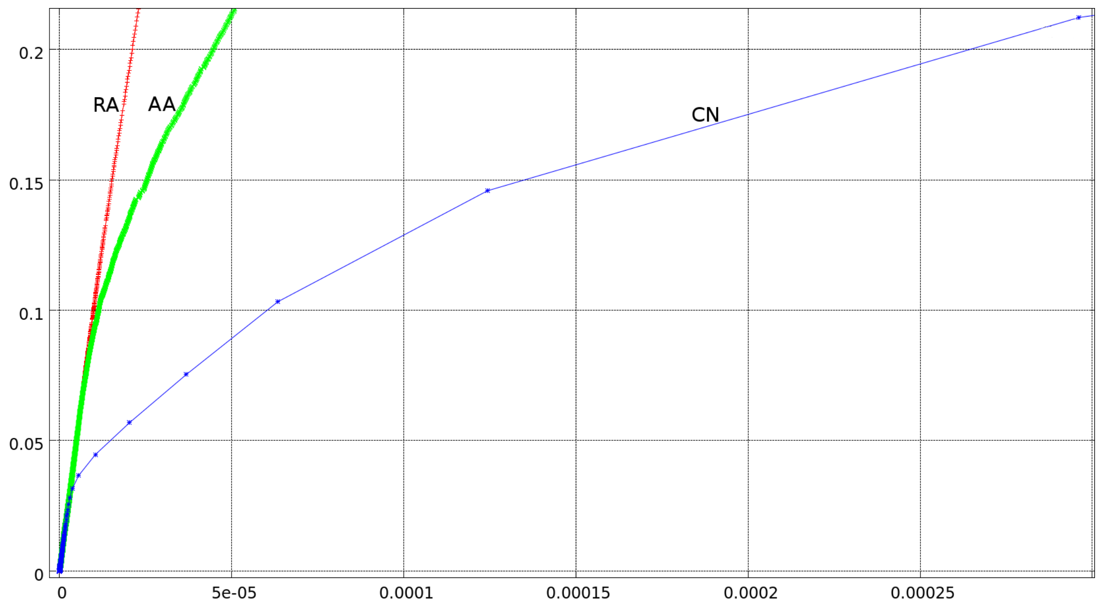


Figure A.3: ROC curve of RA, AA and CN scores on the WordNet graph, zoomed in.

A.1.2 Undirected Scores on Cyc

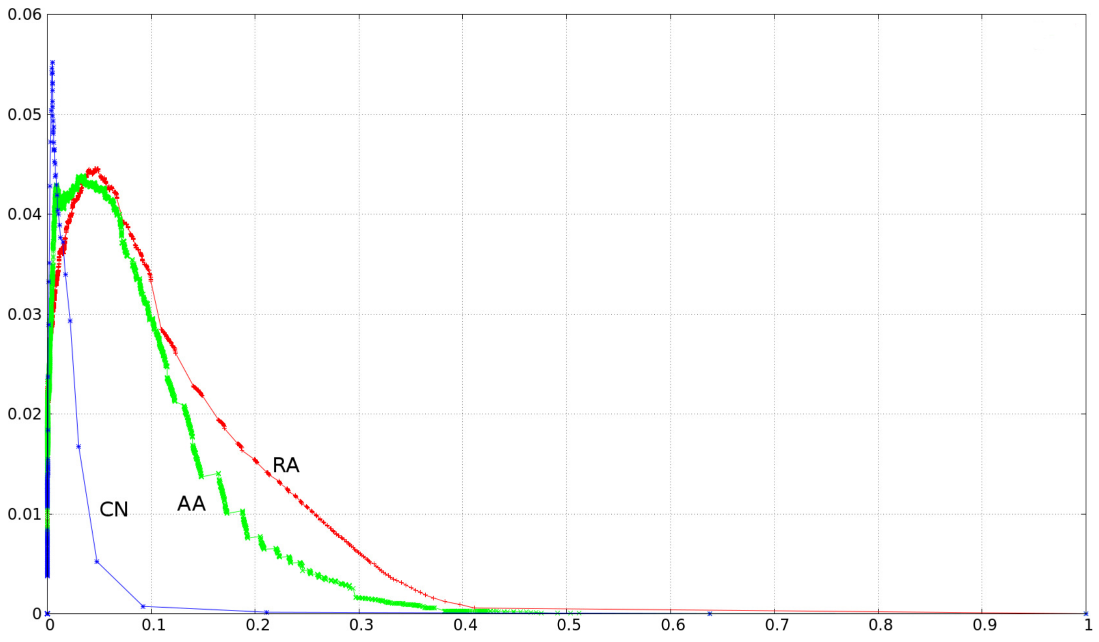


Figure A.4: PR curve of RA, AA and CN scores on the Cyc graph.

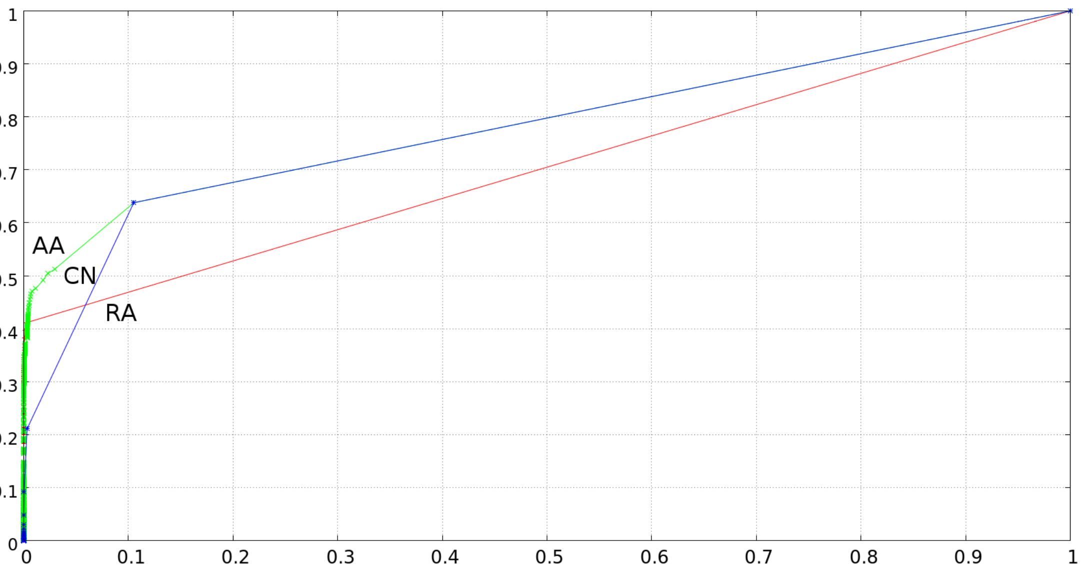


Figure A.5: ROC curve of RA, AA and CN scores on the Cyc graph.

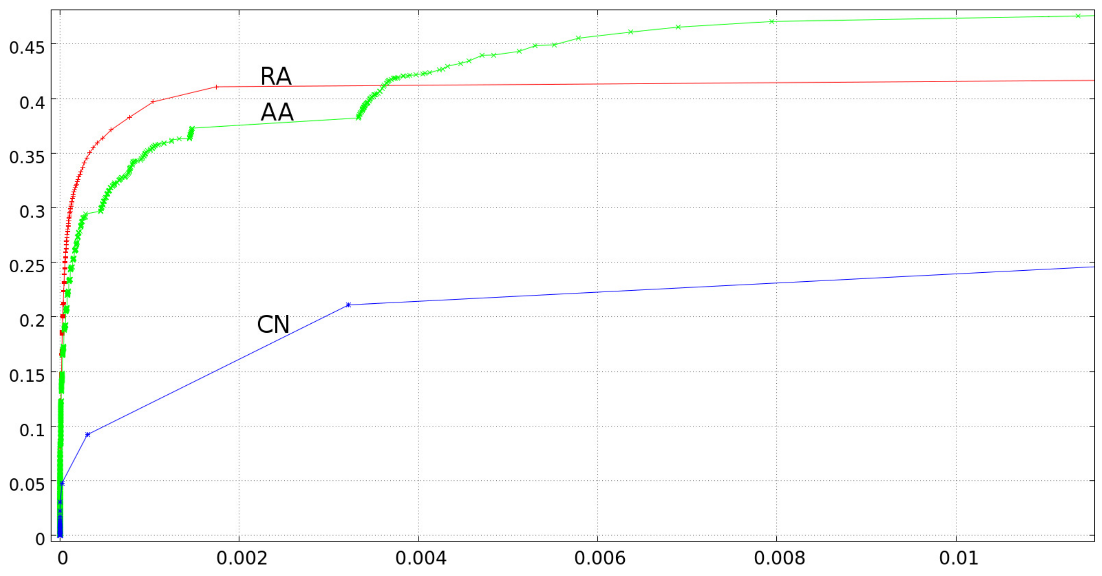


Figure A.6: ROC curve of RA, AA and CN scores on the Cyc graph, zoomed in.

A.1.3 Undirected Scores on IMDb

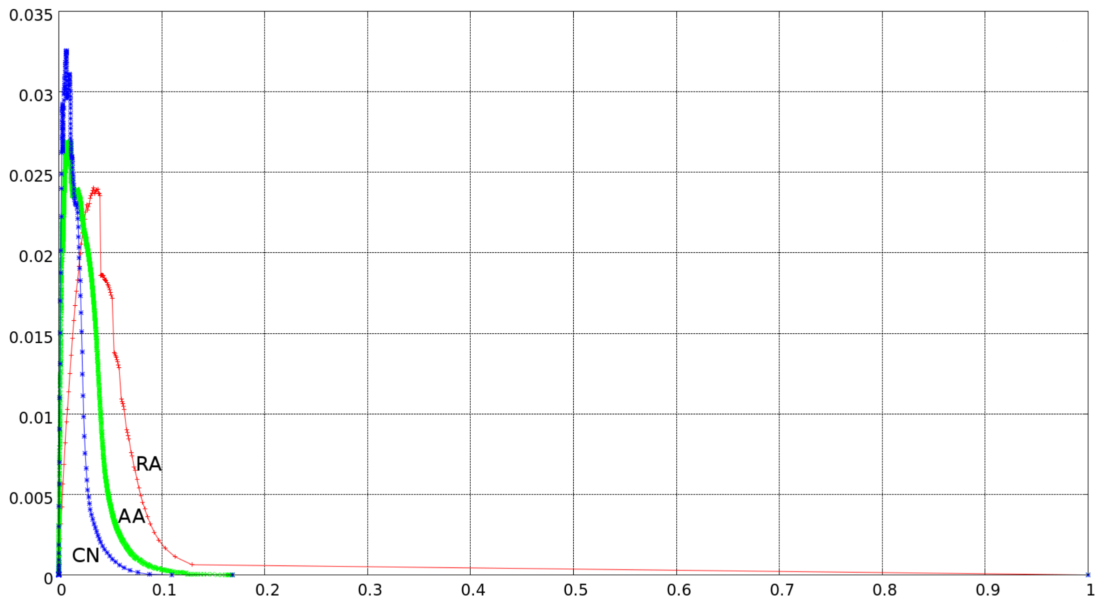


Figure A.7: PR curve of RA, AA and CN scores on the IMDb graph.

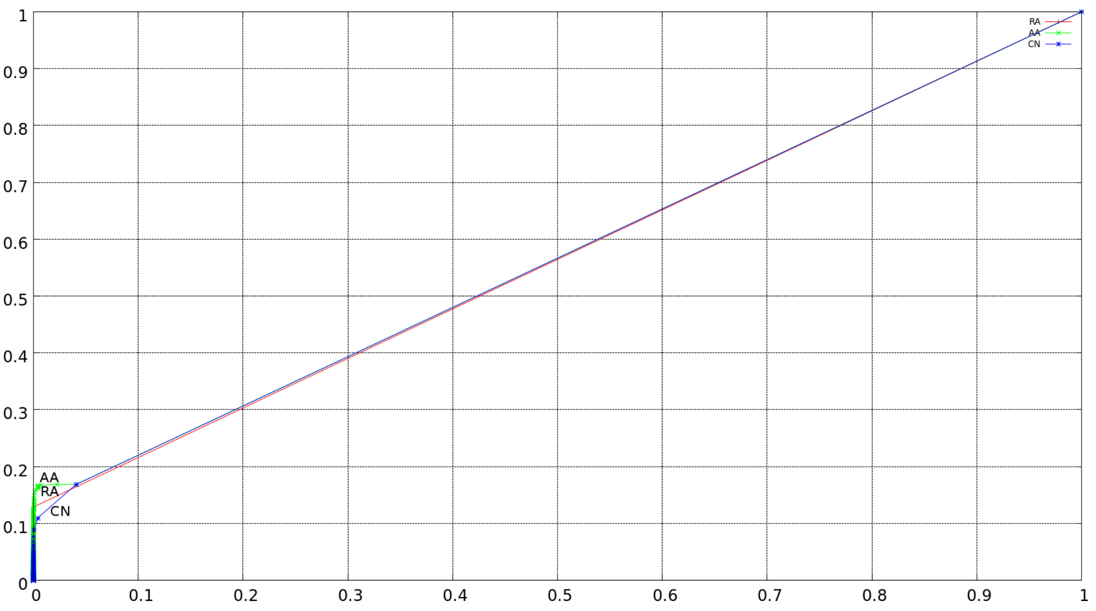


Figure A.8: ROC curve of RA, AA and CN scores on the IMDb graph.

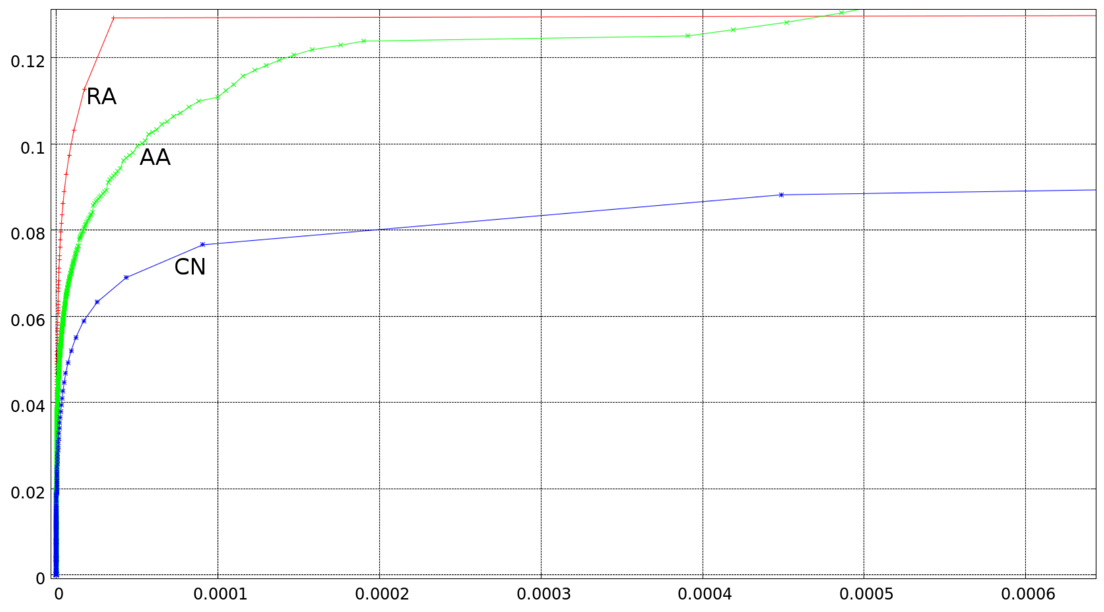


Figure A.9: ROC curve of RA, AA and CN scores on the IMDb graph, zoomed in.

A.2 ROC/PR curves, ROC/PR-AUC: Hierarchical Undirected Scores

Score	WordNet PR-AUC	Cyc PR-AUC	IMDb PR-AUC
HRA	<u>0.215312</u>	<u>0.0195774</u>	<u>0.00305526</u>
HAA	0.085892	0.0156353	0.00215395
HCN	0.025417	0.0035343	0.00154233

Table A.3: PR-AUC score of HRA, HAA and HCN on three graphs.

Score	WordNet ROC-AUC	Cyc ROC-AUC	IMDb ROC-AUC
HRA	0.971901	0.701416	0.563886
HAA	<u>0.987109</u>	<u>0.789144</u>	<u>0.567013</u>
HCN	0.972509	0.779297	0.565826

Table A.4: ROC-AUC score of HRA, HAA and HCN on three graphs.

A.2.1 Hierarchical Undirected Scores on WordNet

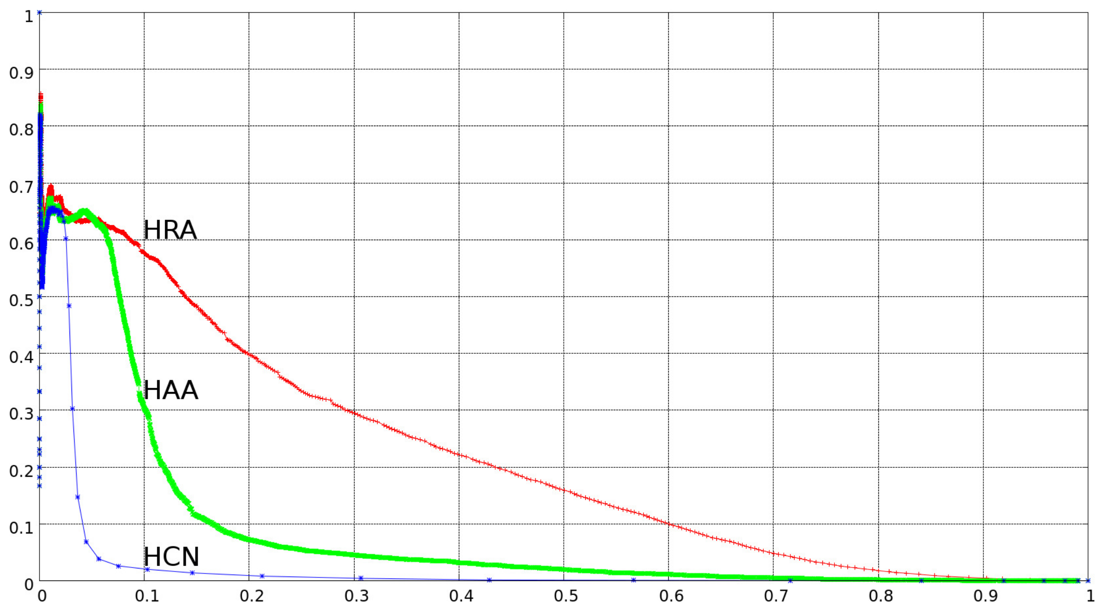


Figure A.10: PR curve of HRA, HAA and HCN scores on the WordNet graph.

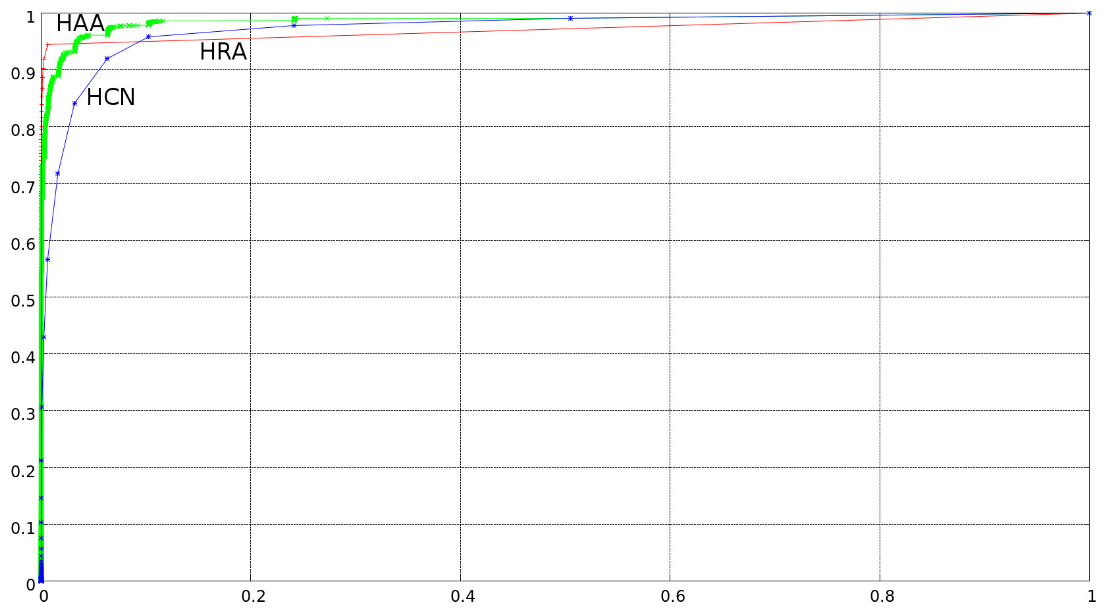


Figure A.11: ROC curve of HRA, HAA and HCN scores on the WordNet graph.

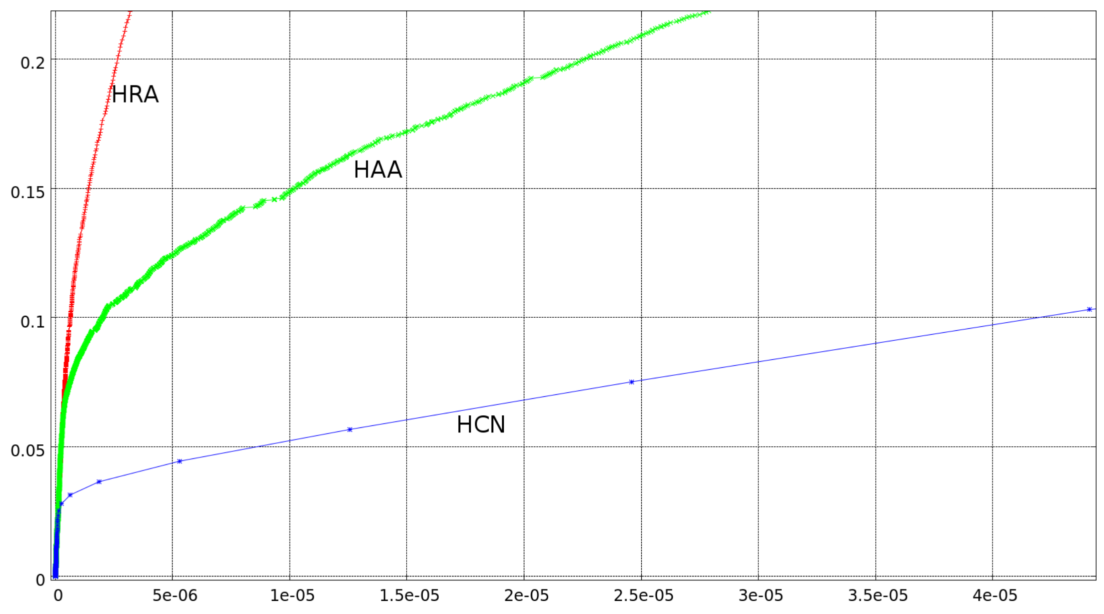


Figure A.12: ROC curve of HRA, HAA and HCN scores on the WordNet graph, zoomed in.

A.2.2 Hierarchical Undirected Scores on Cyc

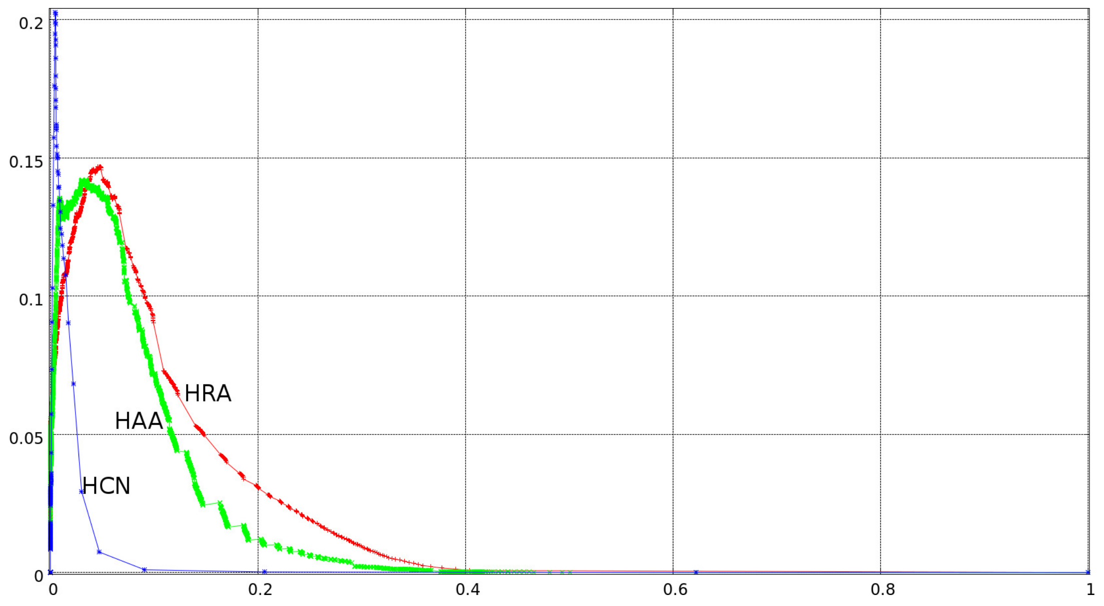


Figure A.13: PR curve of HRA, HAA and HCN scores on the Cyc graph.

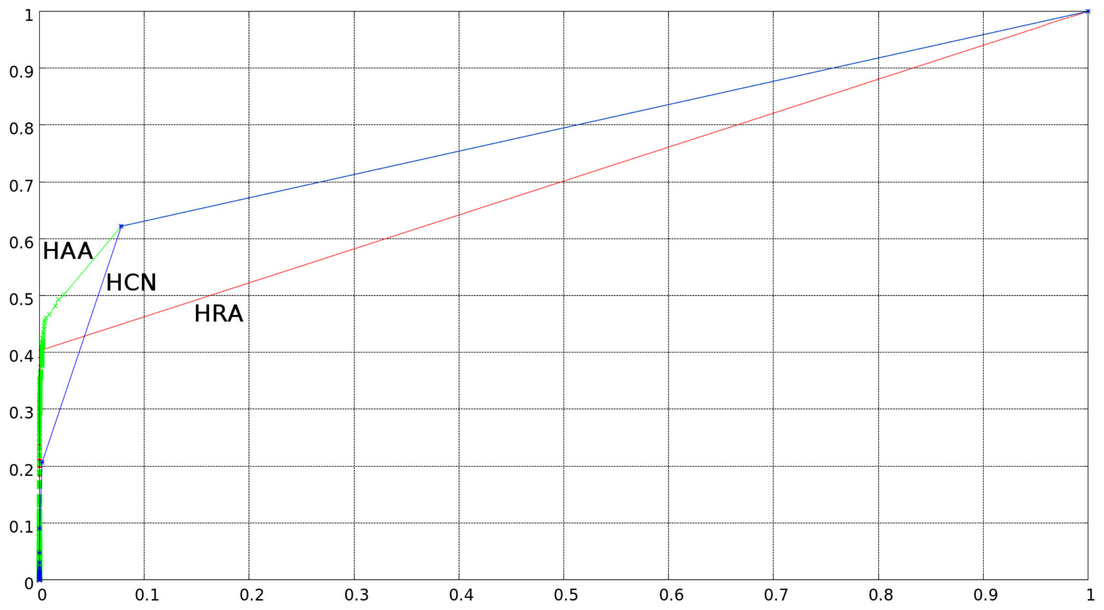


Figure A.14: ROC curve of HRA, HAA and HCN scores on the Cyc graph.

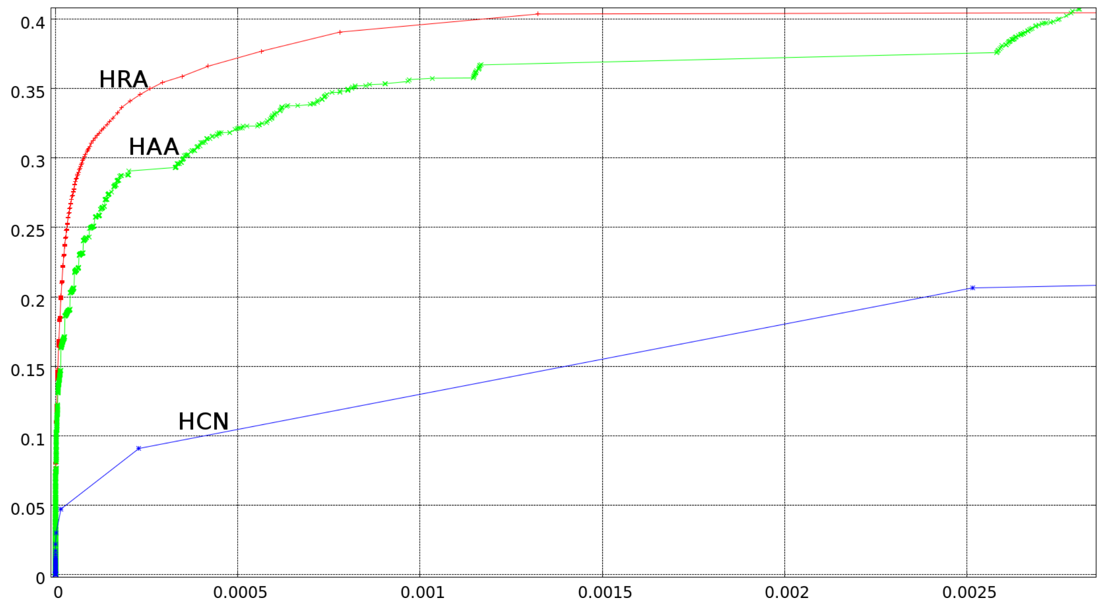


Figure A.15: ROC curve of HRA, HAA and HCN scores on the Cyc graph, zoomed in.

A.2.3 Hierarchical Undirected Scores on IMDb

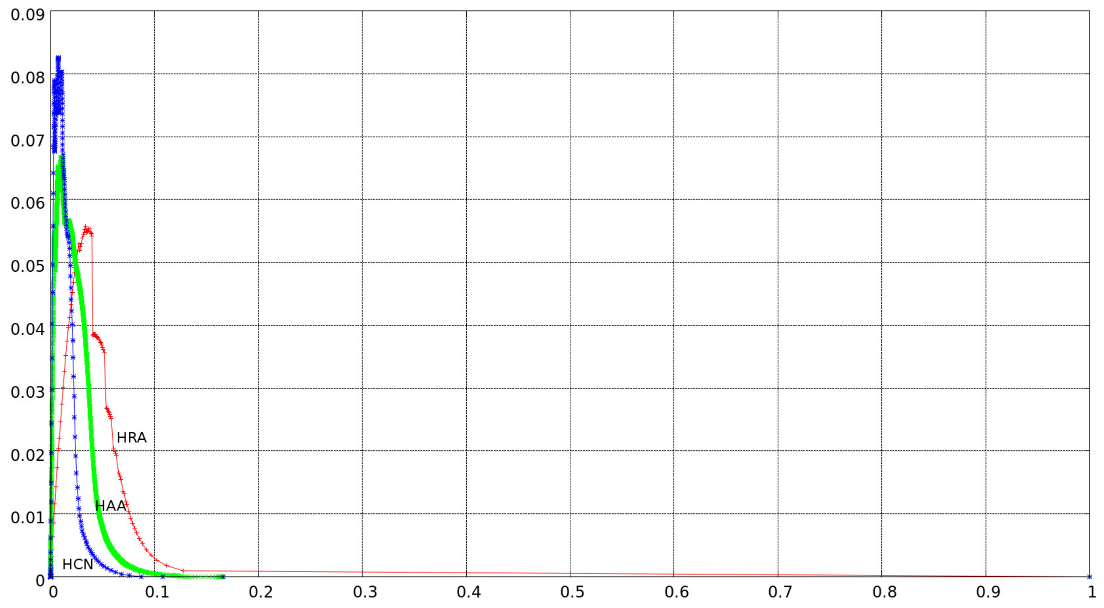


Figure A.16: PR curve of HRA, HAA and HCN scores on the IMDb graph.

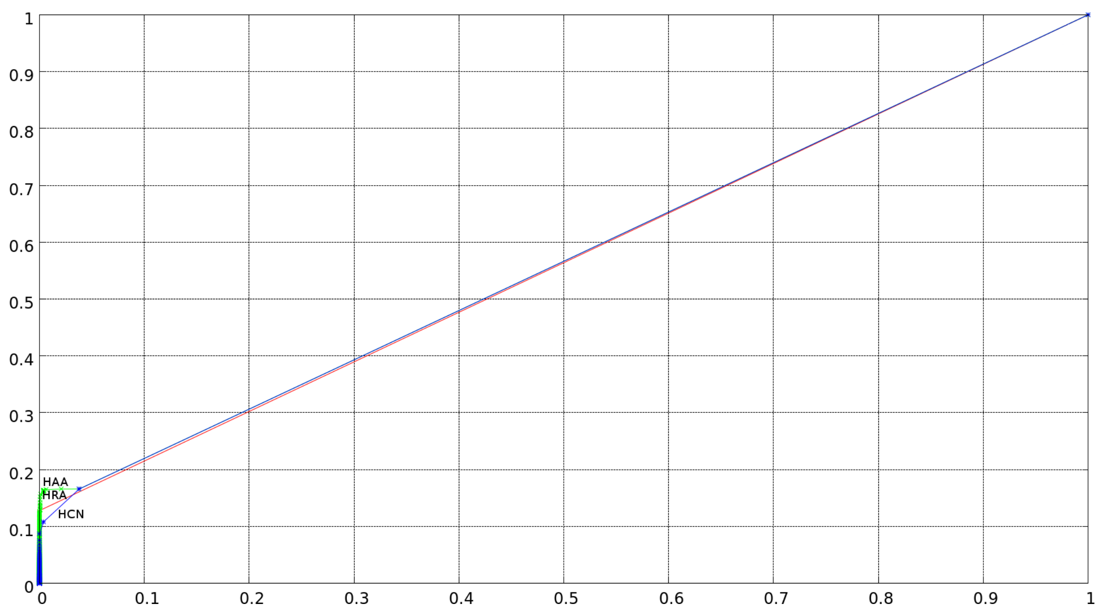


Figure A.17: ROC curve of HRA, HAA and HCN scores on the IMDb graph.

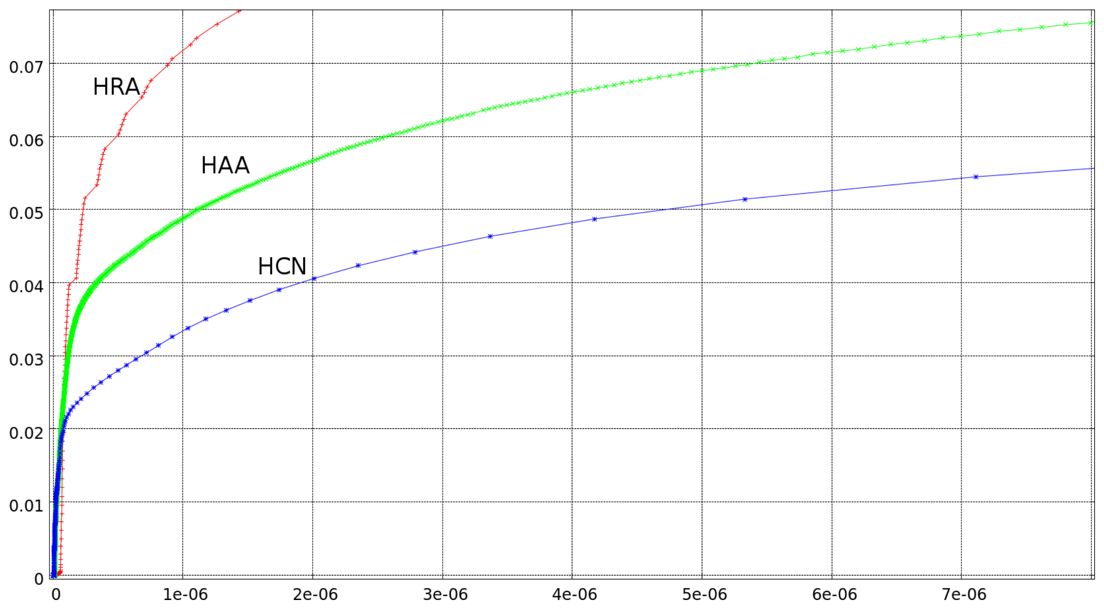


Figure A.18: ROC curve of HRA, HAA and HCN scores on the IMDb graph, zoomed in.

A.3 ROC/PR curves, ROC/PR-AUC: INF evaluation

Score	**WordNet PR-AUC**	**Cyc PR-AUC**	**IMDb PR-AUC**
INF	0.715	0.0735172	0.0233511
HRA	0.215312	0.0195774	0.0030552
RA	0.048763	0.0076491	0.0015387

Table A.5: PR-AUC score of INF, HRA, and RA on three graphs.

Score	**WordNet ROC-AUC**	**Cyc ROC-AUC**	**IMDb ROC-AUC**
INF	0.935562	0.672769	0.571541
HRA	0.971901	0.701416	0.563886
RA	0.971892	0.704801	0.564618

Table A.6: ROC-AUC score of INF, HRA, and RA on three graphs.

A.3.1 INF Evaluation on WordNet

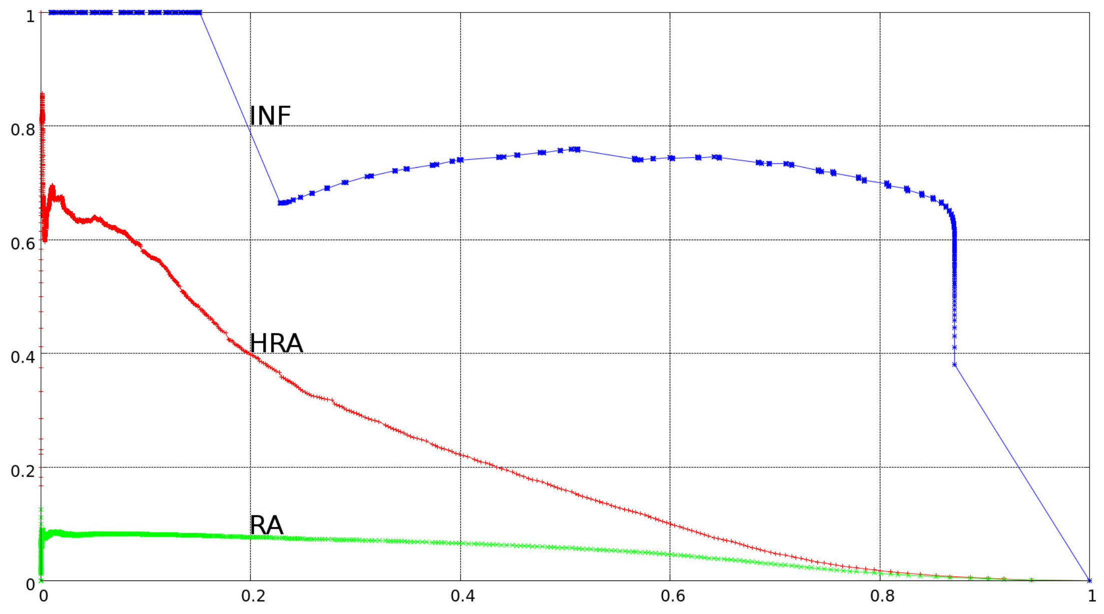


Figure A.19: PR curve of INF, HRA and RA scores on the WordNet graph.

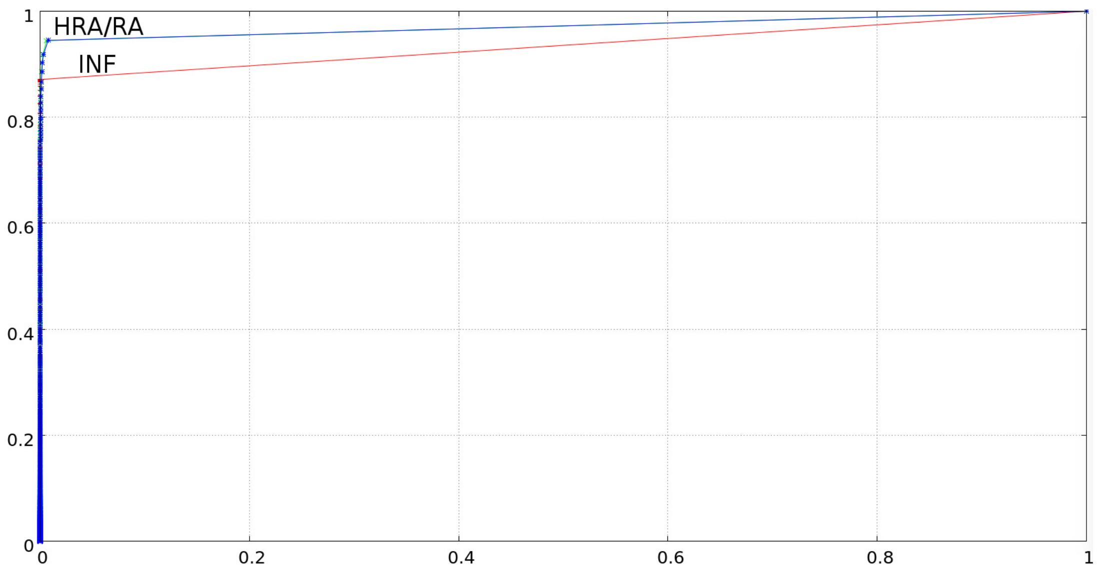


Figure A.20: ROC curve of INF, HRA and RA scores on the WordNet graph.

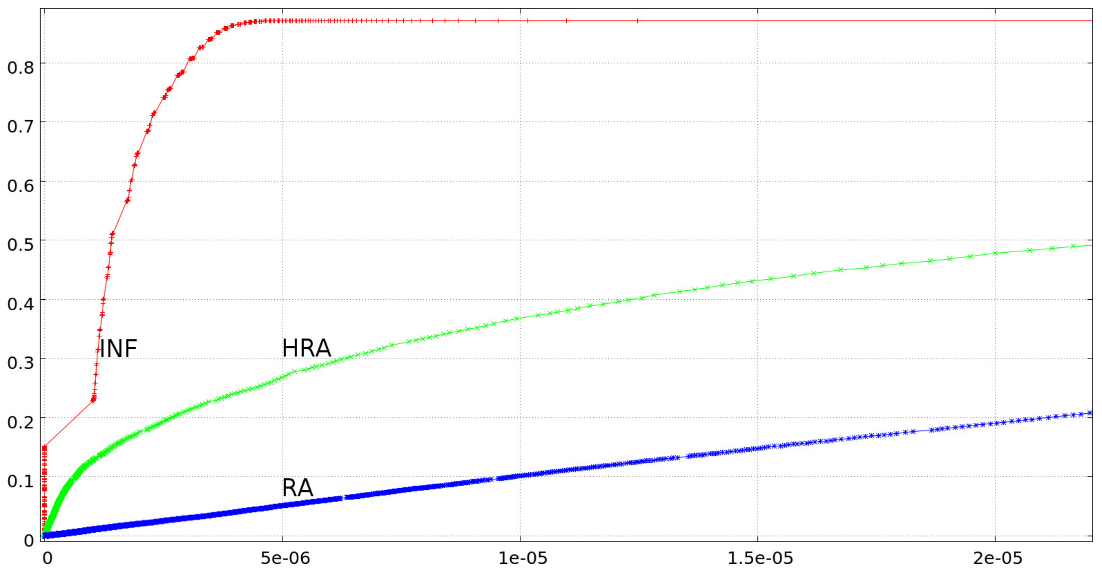


Figure A.21: ROC curve of INF, HRA and RA scores on the WordNet graph, zoomed in.

A.3.2 INF Evaluation on Cyc

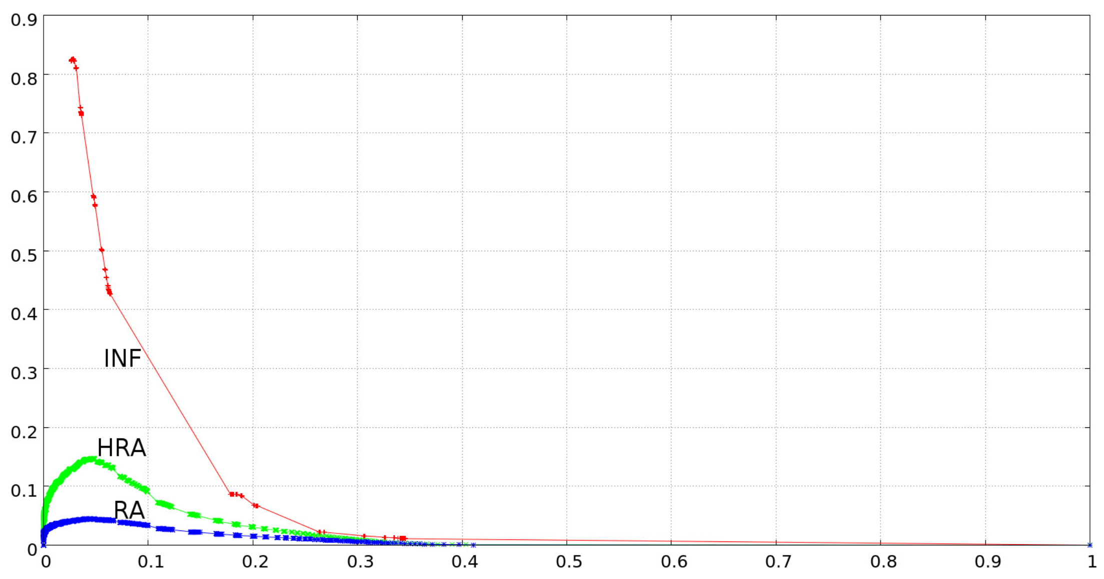


Figure A.22: PR curve of INF, HRA and RA scores on the Cyc graph.

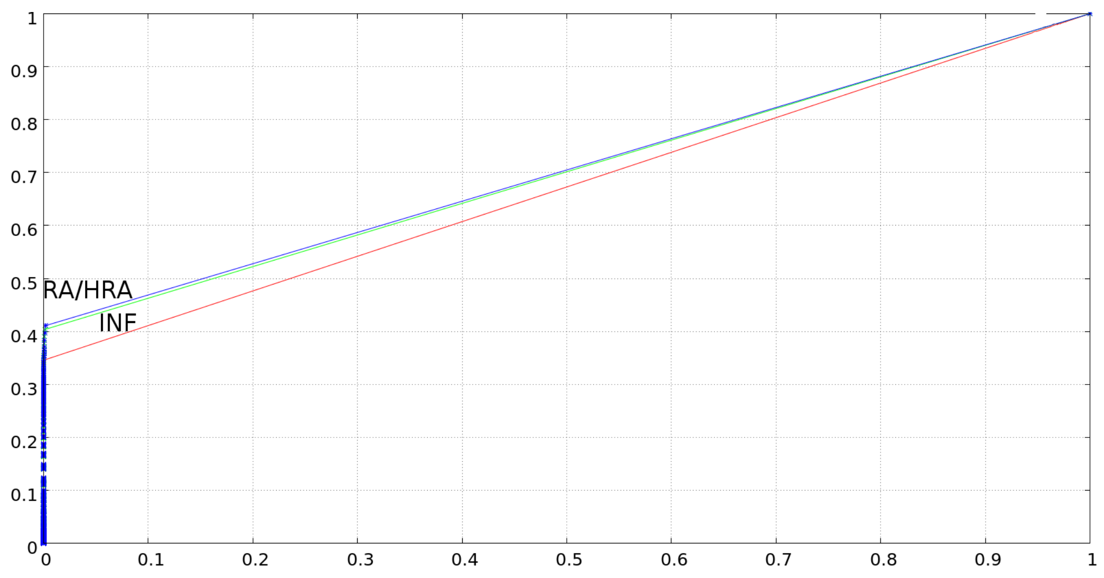


Figure A.23: ROC curve of INF, HRA and RA scores on the Cyc graph.

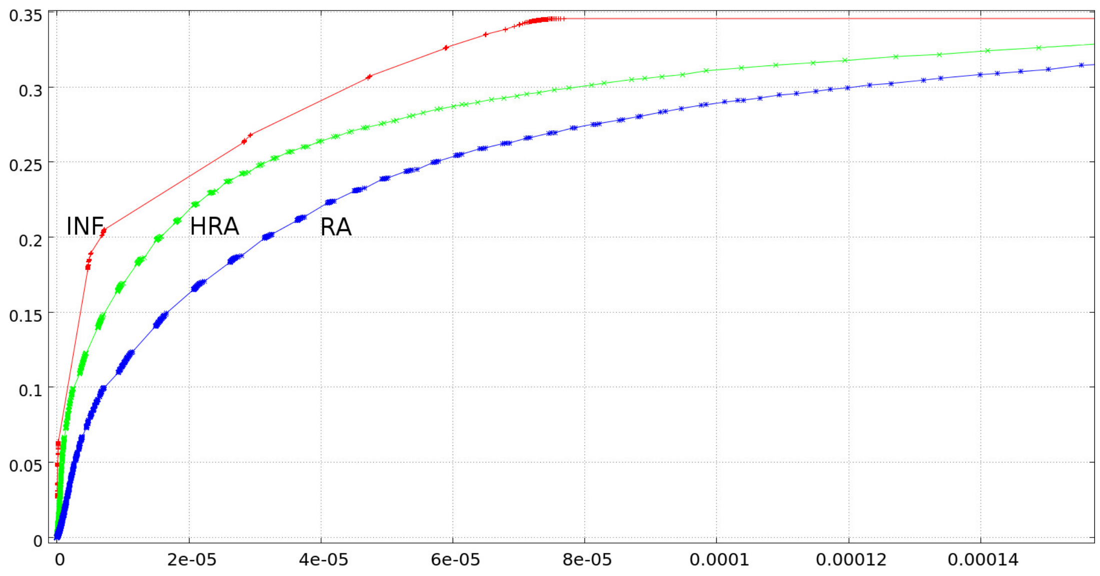


Figure A.24: ROC curve of INF, HRA and RA scores on the Cyc graph, zoomed in.

A.3.3 INF Evaluation on IMDb

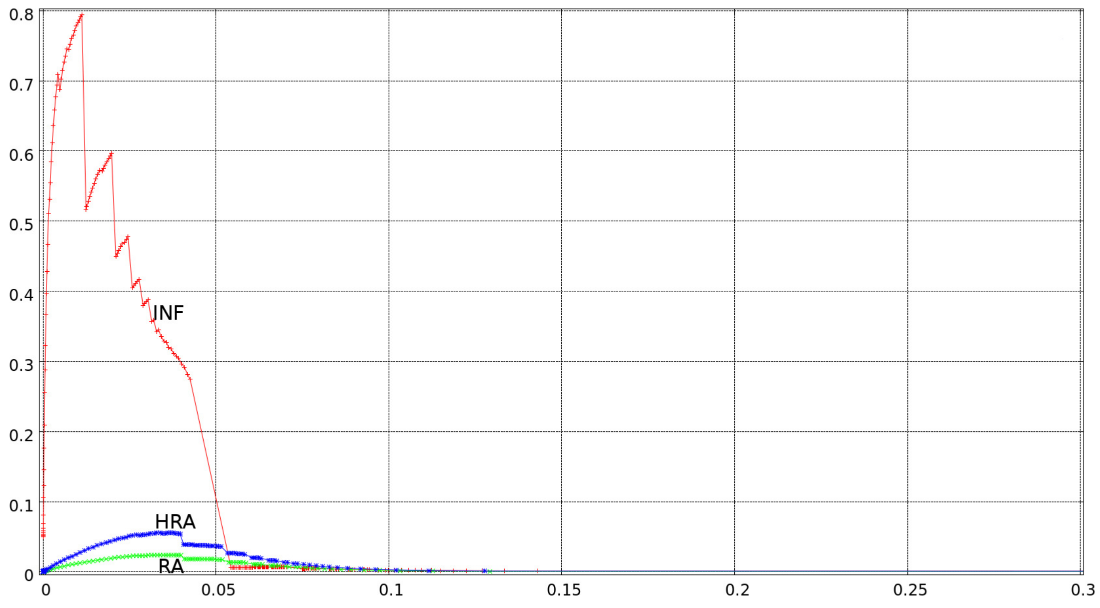


Figure A.25: PR curve of INF, HRA and RA scores on the IMDb graph.

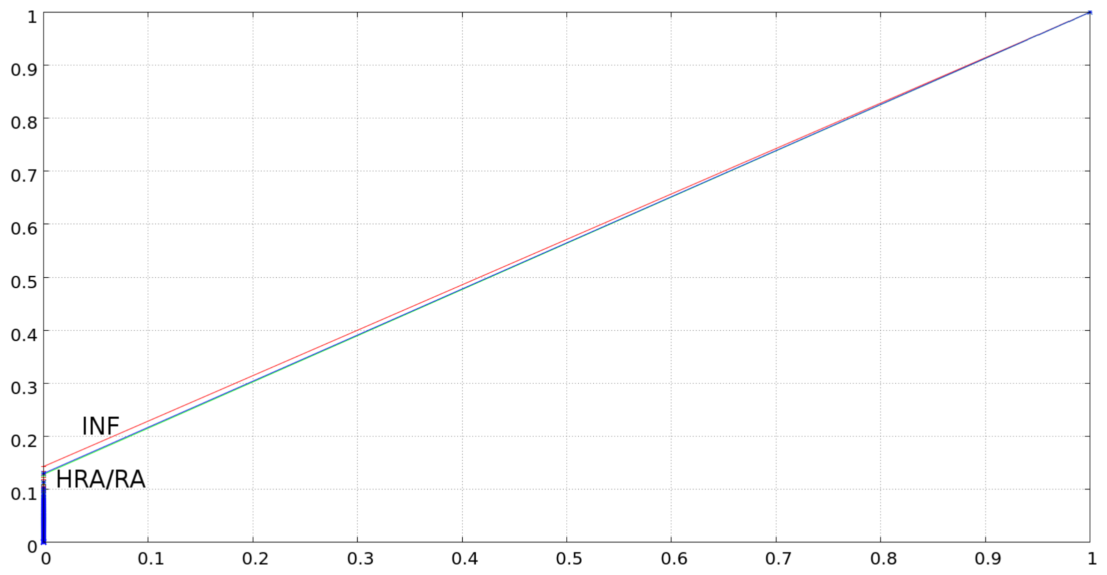


Figure A.26: ROC curve of INF, HRA and RA scores on the IMDb graph.

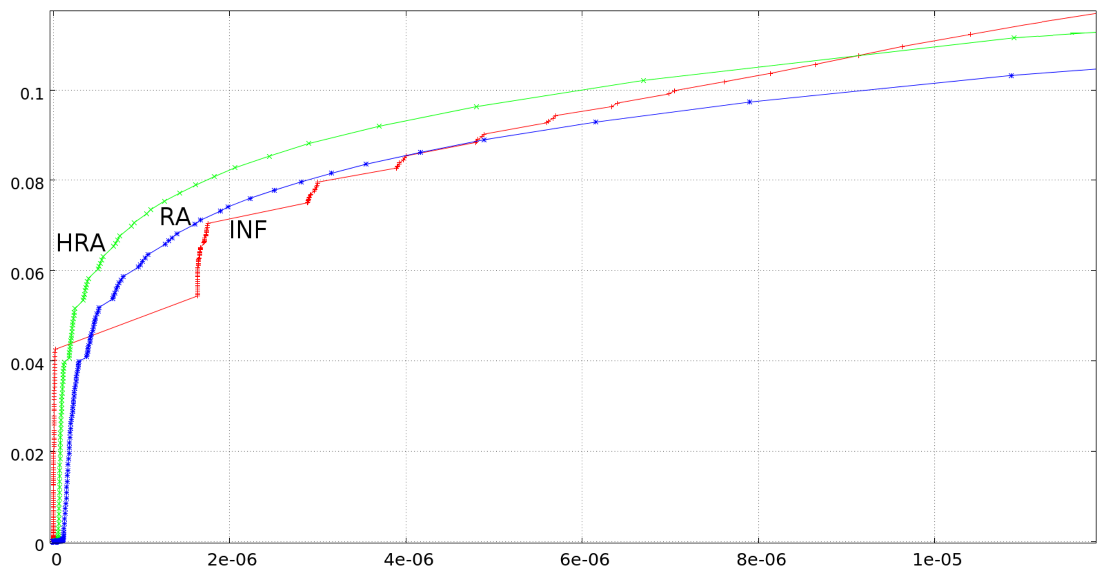


Figure A.27: ROC curve of INF, HRA and RA scores on the IMDb graph, zoomed in.

A.4 ROC/PR curves, ROC/PR-AUC: Tunned INFs

Score	**WordNet PR-AUC**	**Cyc PR-AUC**	**IMDb PR-AUC**
INF	0.715	0.0735172	0.023351
INF_2D	0.837709	0.0721955	0.016052
INF_LOG	0.730075	0.0188687	0.026152
INF_LOG_2D	0.839372	0.0189289	0.0326476

Table A.7: PR-AUC score of INF, INF_2D, INF_LOG, INF_LOG_2D on three graphs.

Score	**WordNet ROC-AUC**	**Cyc ROC-AUC**	**IMDb ROC-AUC**
INF	0.935562	0.672769	0.571541
INF_2D	0.935563	0.672769	0.574928
INF_LOG	0.928184	0.628524	0.57275
INF_LOG_2D	0.928184	0.628524	0.572756

Table A.8: ROC-AUC score of INF, INF_2D, INF_LOG, INF_LOG_2D on three graphs.

A.4.1 Tunned INFs on WordNet

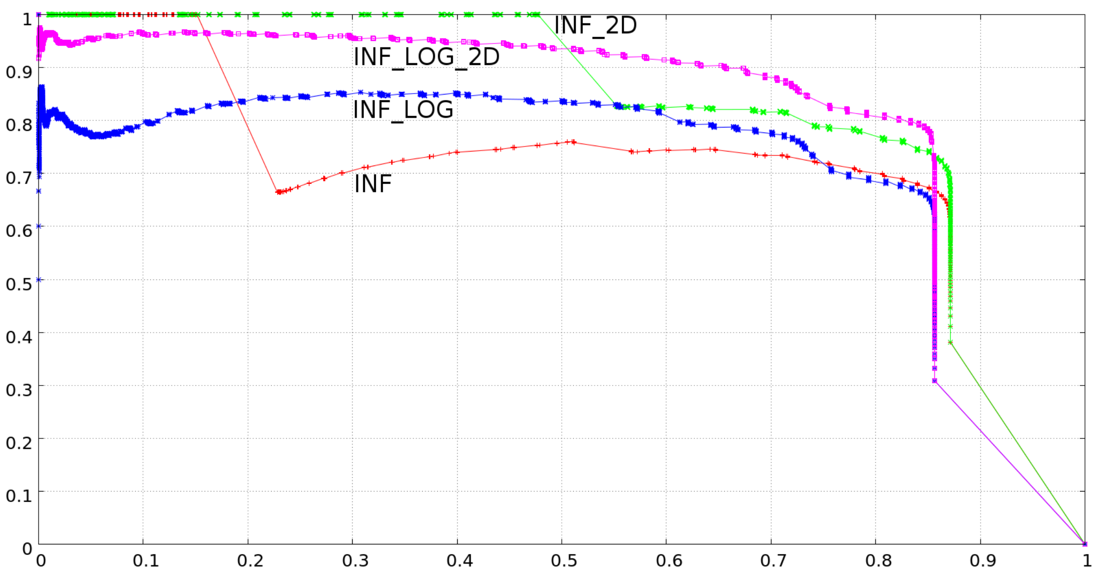


Figure A.28: PR curve of INF, INF_2D, INF_LOG, INF_LOG_2D scores on the WordNet graph.

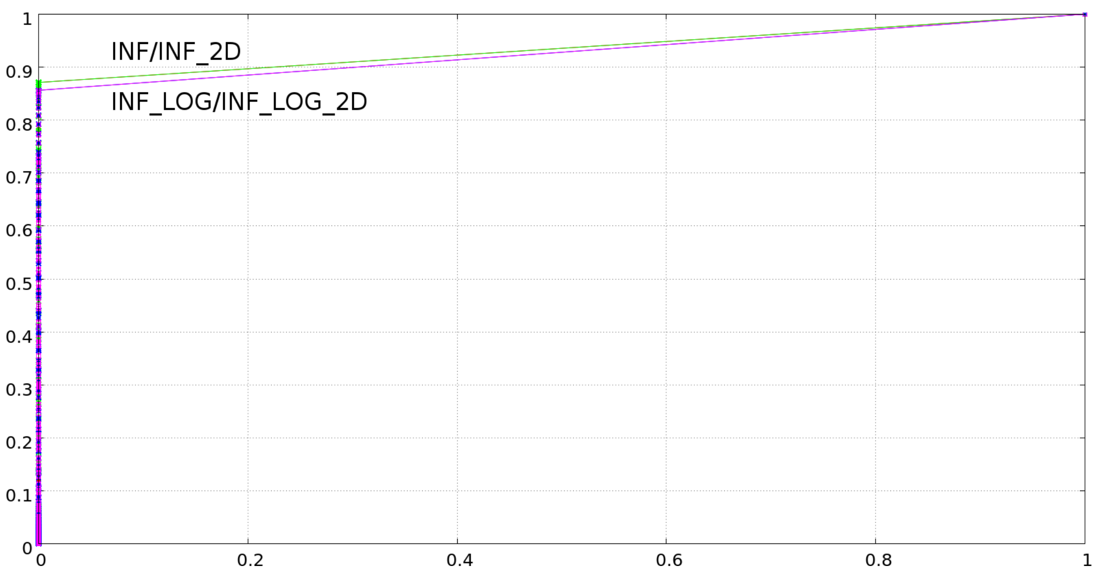


Figure A.29: ROC curve of INF, INF_2D, INF_LOG, INF_LOG_2D scores on the WordNet graph.

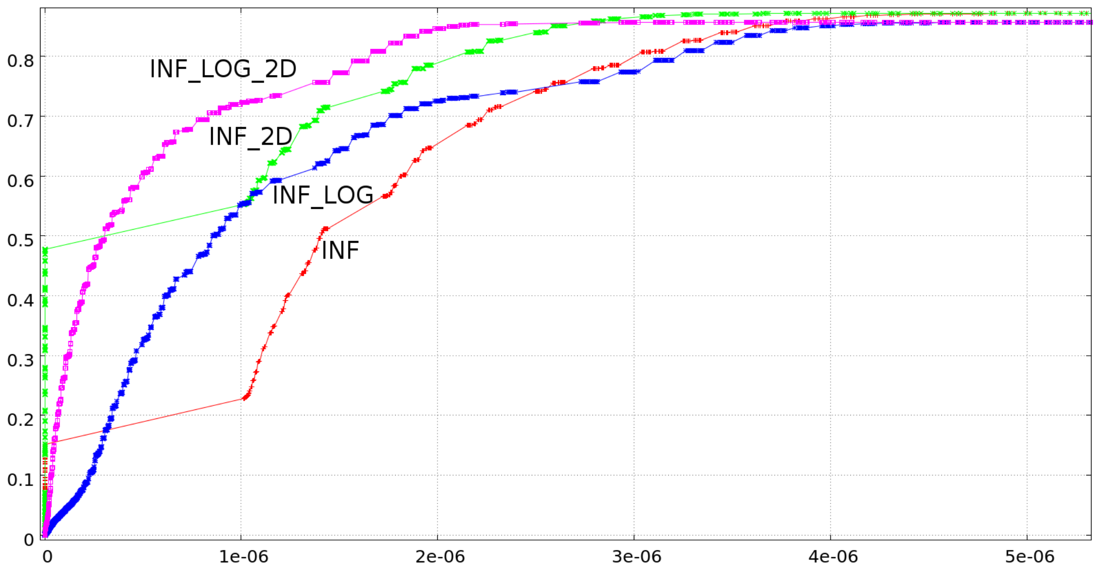


Figure A.30: ROC curve of INF, INF_2D, INF_LOG, INF_LOG_2D scores on the WordNet graph, zoomed in.

A.4.2 Tunned INFs on Cyc

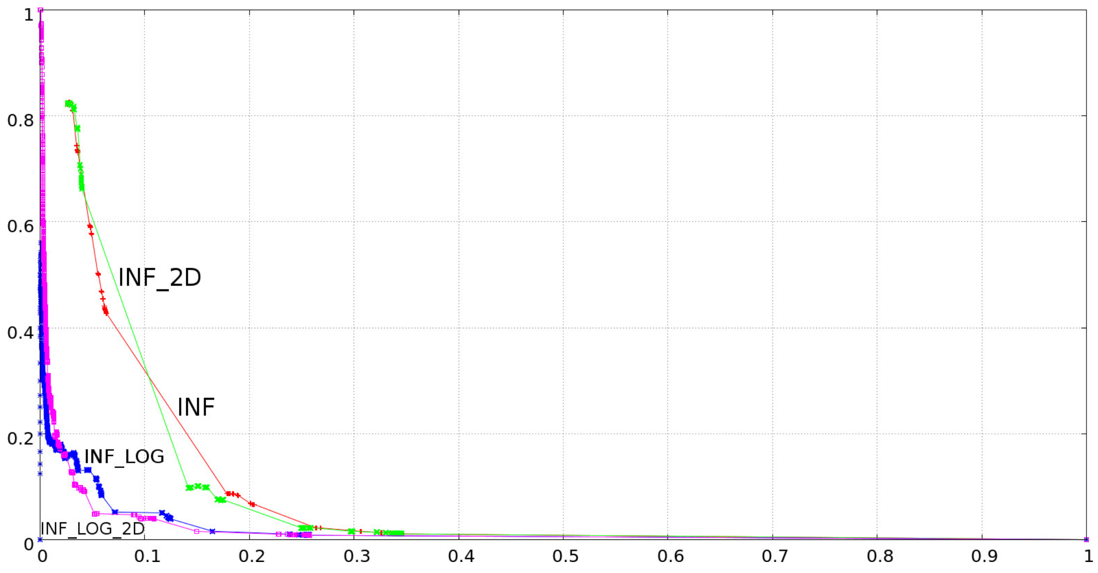


Figure A.31: PR curve of INF, INF_2D, INF_LOG, INF_LOG_2D scores on the Cyc graph.

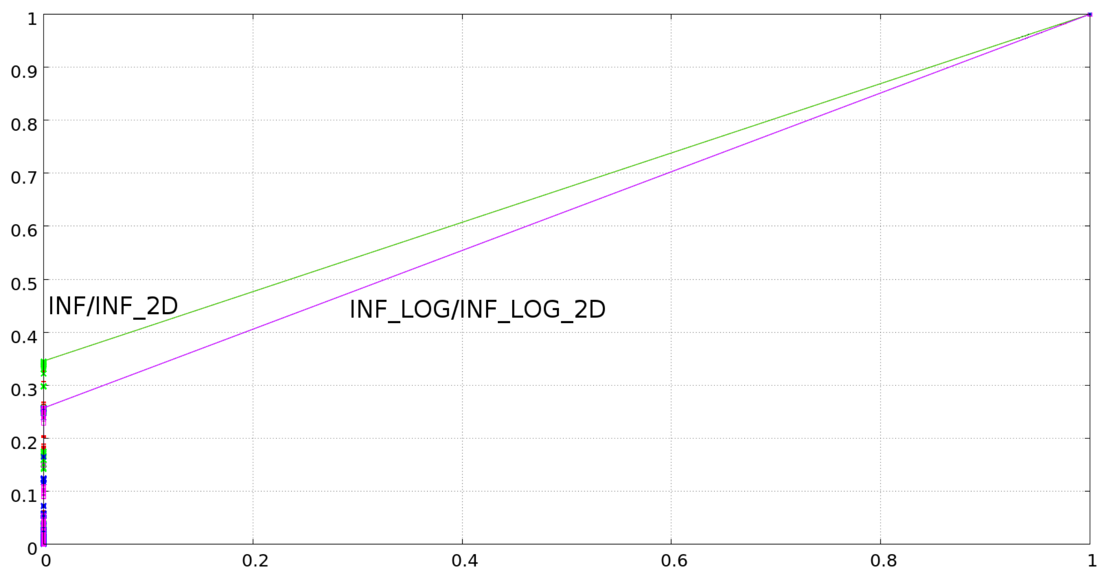


Figure A.32: ROC curve of INF, INF_2D, INF_LOG, INF_LOG_2D scores on the Cyc graph.

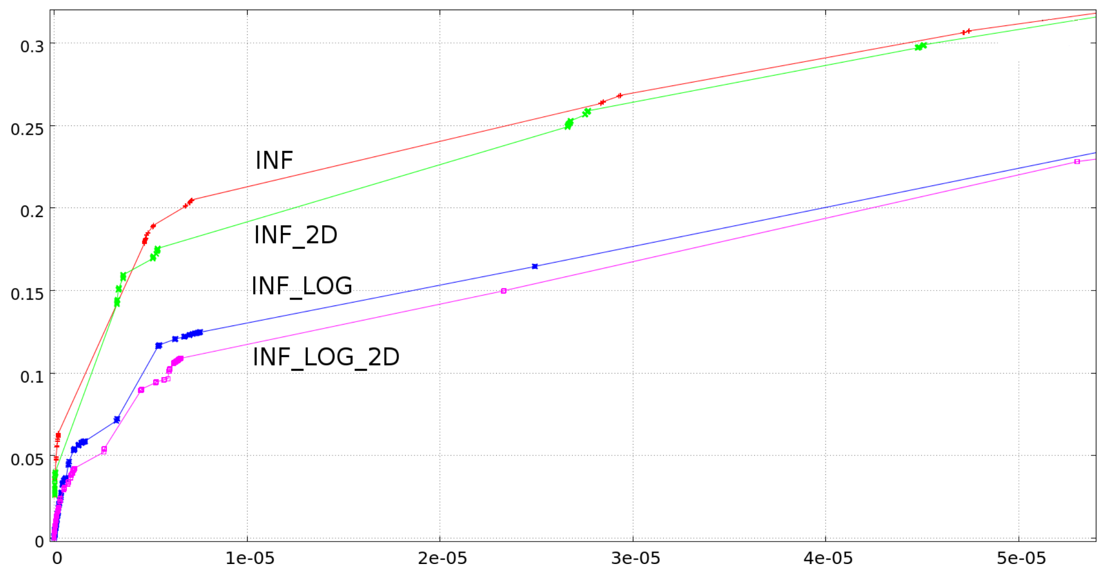


Figure A.33: ROC curve of INF, INF_2D, INF_LOG, INF_LOG_2D scores on the Cyc graph, zoomed in.

A.4.3 Tunned INFs on IMDb

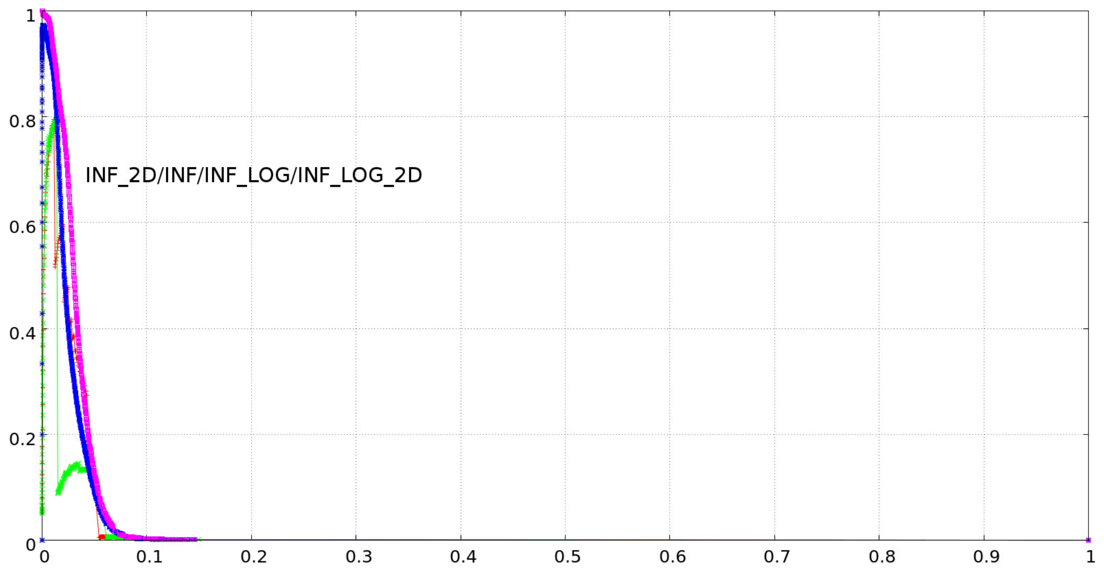


Figure A.34: PR curve of INF, INF_2D, INF_LOG, INF_LOG_2D scores on the IMDb graph.

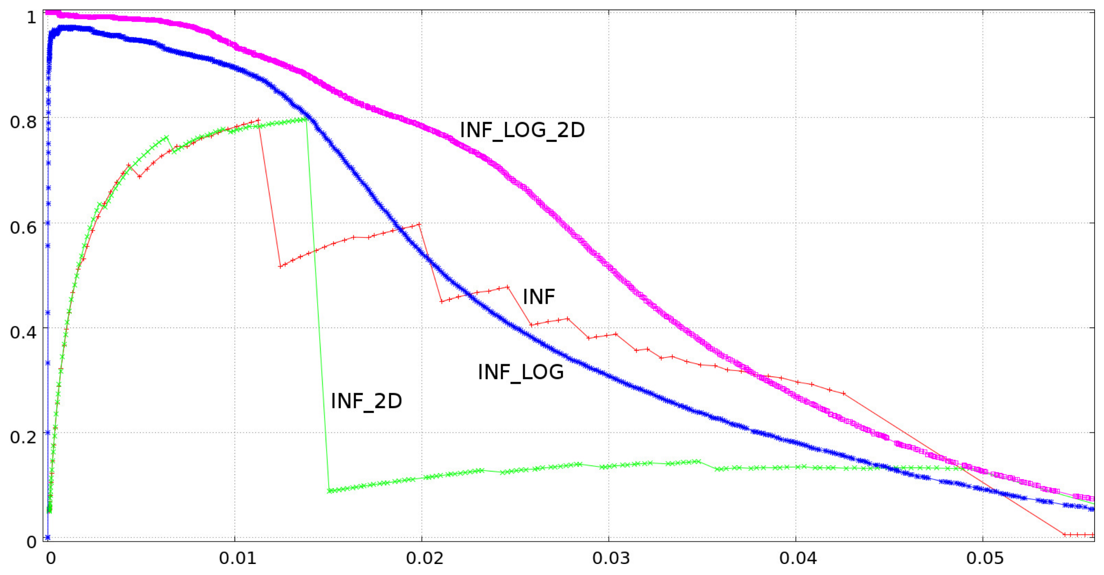


Figure A.35: PR curve of INF, INF_2D, INF_LOG, INF_LOG_2D scores on the IMDb graph, zoomed in.

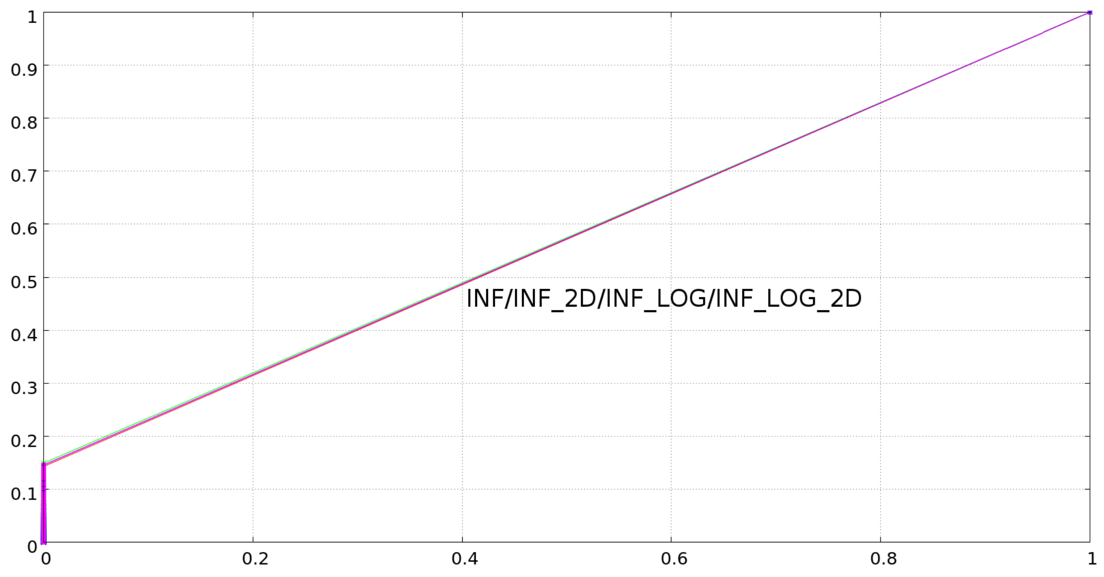


Figure A.36: ROC curve of INF, INF_2D, INF_LOG, INF_LOG_2D scores on the IMDb graph.

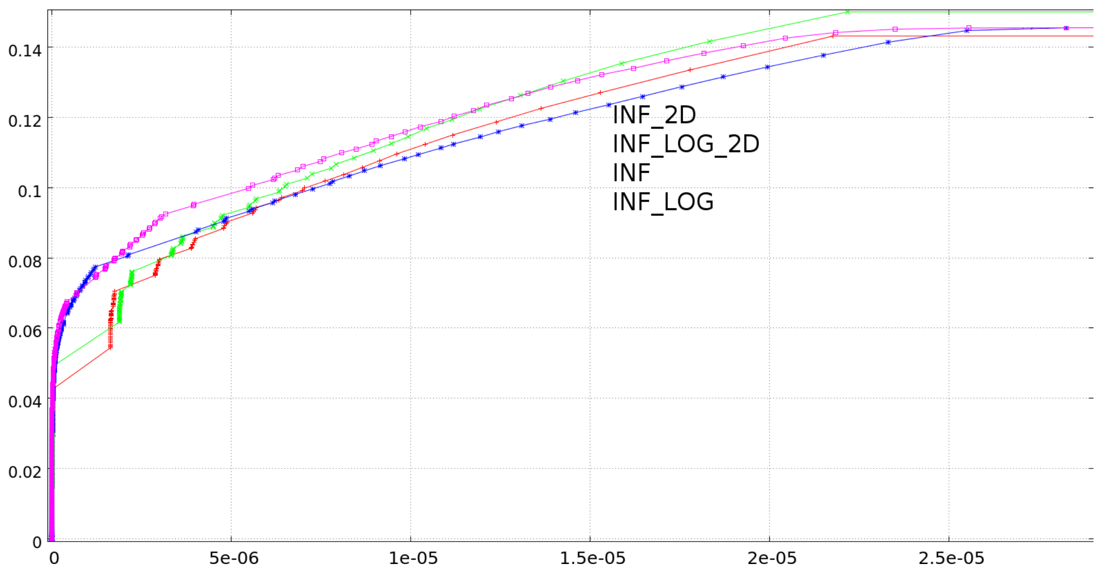


Figure A.37: ROC curve of INF, INF_2D, INF_LOG, INF_LOG_2D scores on the IMDb graph, zoomed in.

A.5 ROC/PR curves, ROC/PR-AUC: Quasi-local INFs

Score	**WordNet**	**Cyc**	**IMDb**
	PR-AUC	**PR-AUC**	**PR-AUC**
INF_QL	0.720866	$\underline{0.0473856}$	0.0060537
INF_2D_QL	$\underline{0.842386}$	0.0432759	0.0107751
INF_LOG_QL	0.726982	0.0091870	0.0168905
INF_LOG_2D_QL	0.836465	0.0099114	$\underline{0.0257291}$

Table A.9: PR-AUC score of INF_QL, INF_2D_QL, INF_LOG_QL, INF_LOG_2D_QL on all graphs.

Score	**WordNet**	**Cyc**	**IMDb**
	ROC-AUC	**ROC-AUC**	**ROC-AUC**
INF_QL	0.935562	$\underline{0.707766}$	0.605394
INF_2D_QL	$\underline{0.935563}$	0.707764	$\underline{0.608768}$
INF_LOG_QL	0.928184	0.651159	0.606042
INF_LOG_2D_QL	0.928184	0.651157	0.608571

Table A.10: ROC-AUC score of INF_QL, INF_2D_QL, INF_LOG_QL, INF_LOG_2D_QL on all graphs.

A.5.1 Quasi-local INFs on WordNet

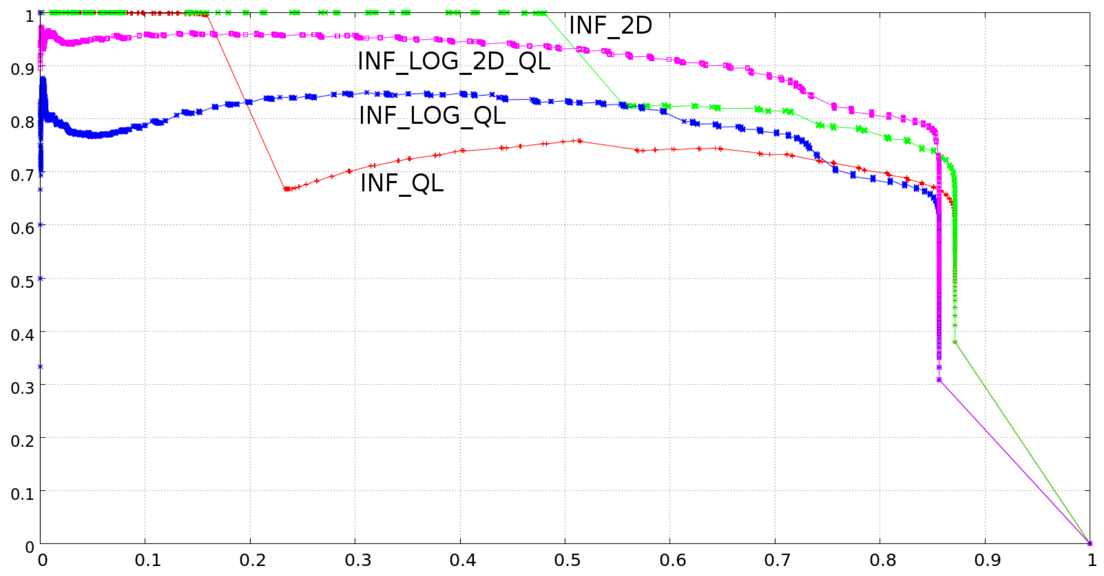


Figure A.38: PR curve of INF_QL, INF_2D_QL, INF_LOG_QL, INF_LOG_2D_QL scores on the WordNet graph.

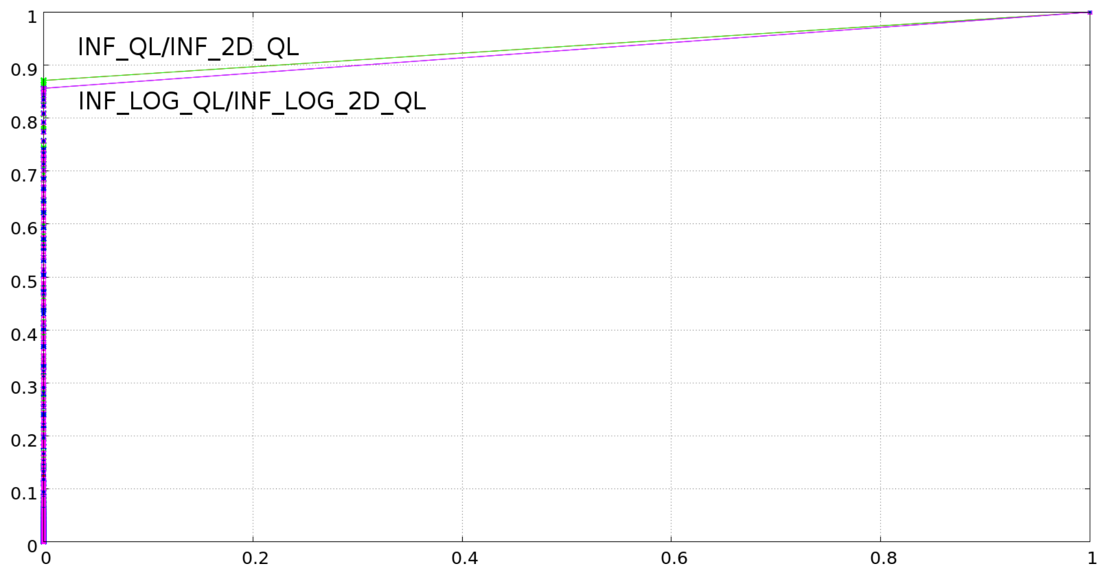


Figure A.39: ROC curve of INF_QL, INF_2D_QL, INF_LOG_QL, INF_LOG_2D_QL scores on the WordNet graph.

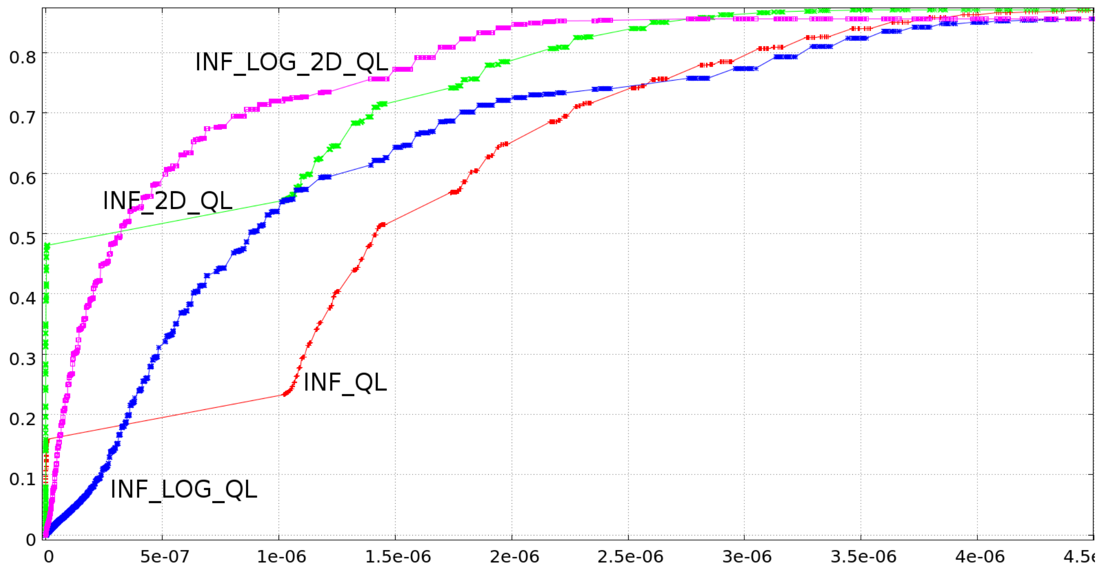


Figure A.40: ROC curve of INF_QL, INF_2D_QL, INF_LOG_QL, INF_LOG_2D_QL scores on the WordNet graph, zoomed in.

A.5.2 Quasi-local INFs on Cyc

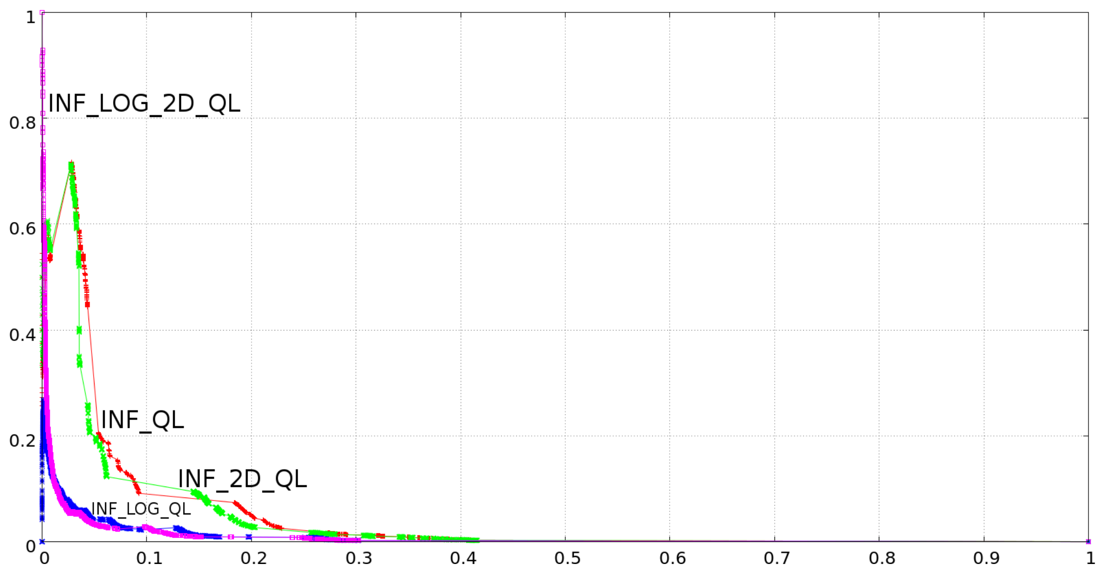


Figure A.41: PR curve of INF_QL, INF_2D_QL, INF_LOG_QL, INF_LOG_2D_QL scores on the Cyc graph.

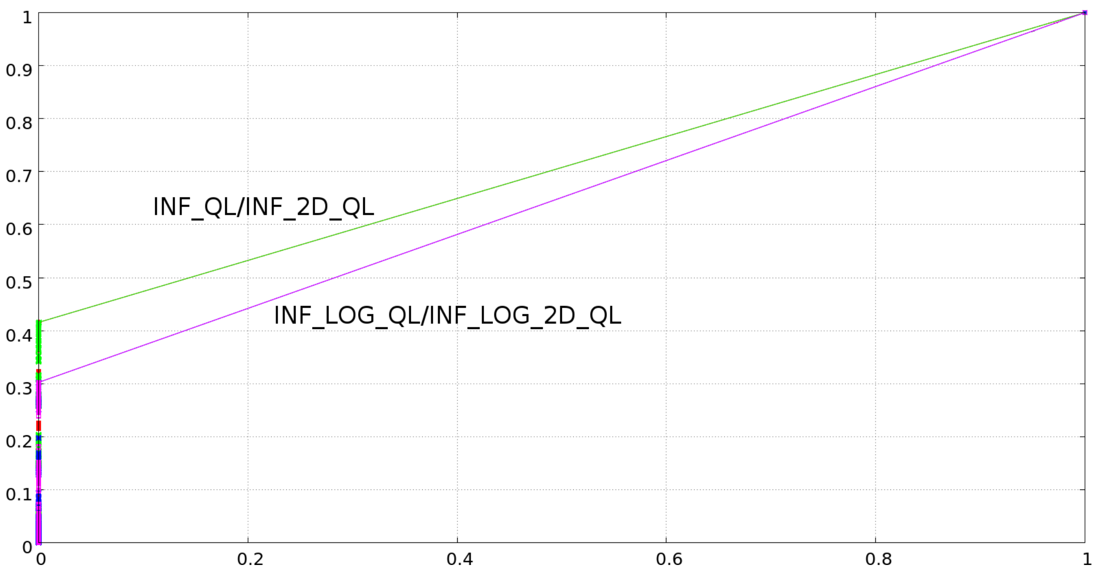


Figure A.42: ROC curve of INF_QL, INF_2D_QL, INF_LOG_QL, INF_LOG_2D_QL scores on the Cyc graph.

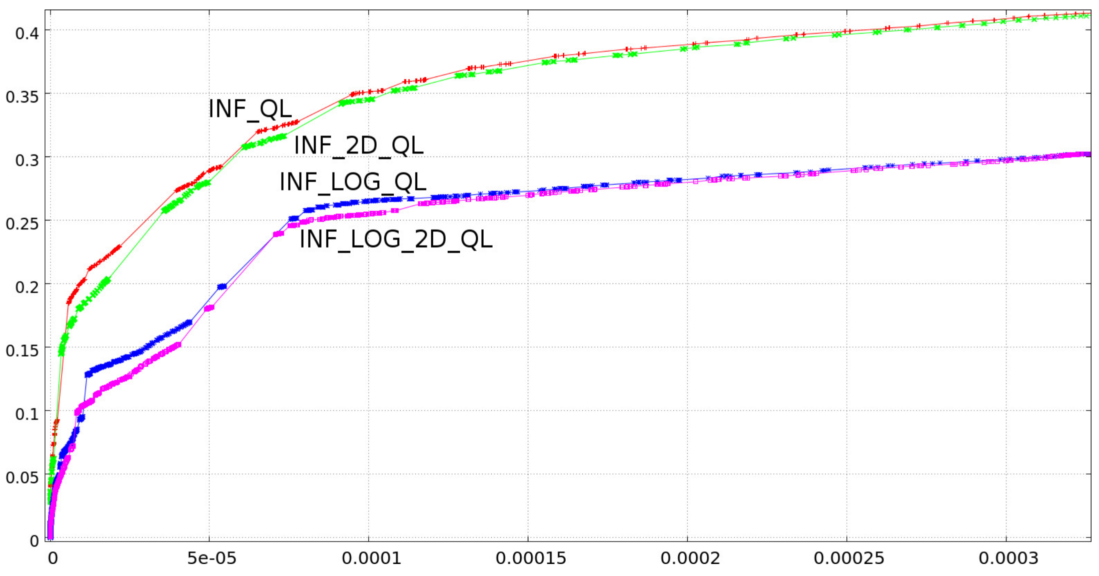


Figure A.43: ROC curve of INF_QL, INF_2D_QL, INF_LOG_QL, INF_LOG_2D_QL scores on the Cyc graph, zoomed in.

A.5.3 Quasi-local INFs on IMDb

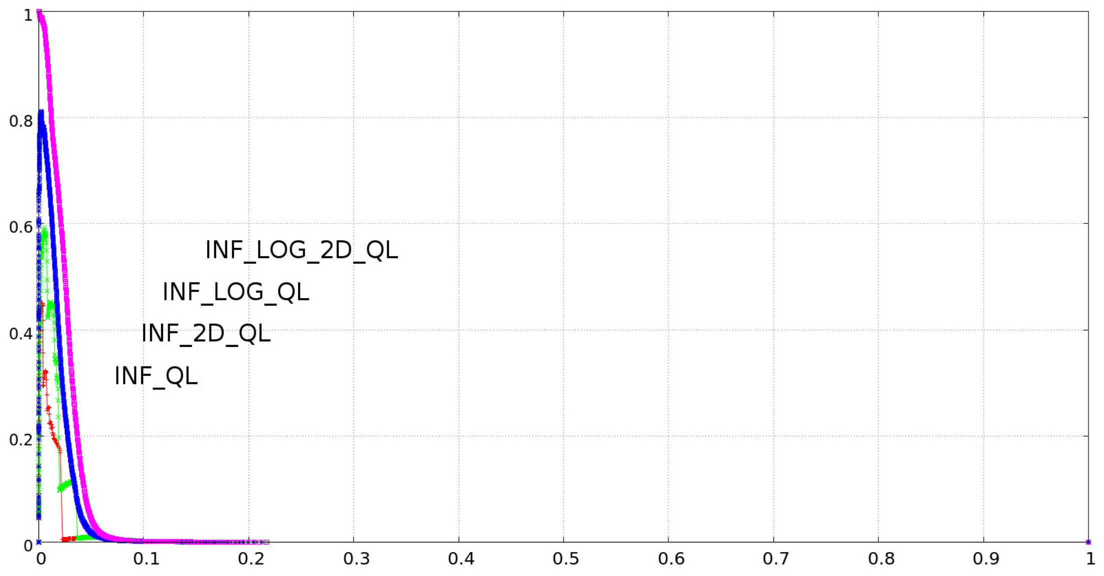


Figure A.44: PR curve of INF_QL, INF_2D_QL, INF_LOG_QL, INF_LOG_2D_QL scores on the IMDb graph.

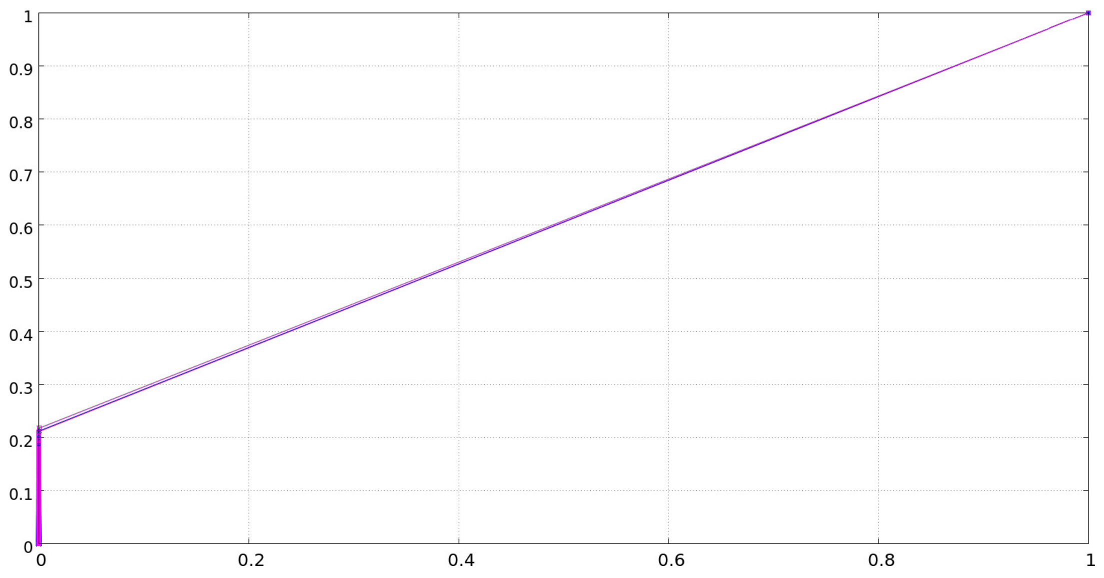


Figure A.45: ROC curve of INF_QL, INF_2D_QL, INF_LOG_QL, INF_LOG_2D_QL scores on the IMDb graph.

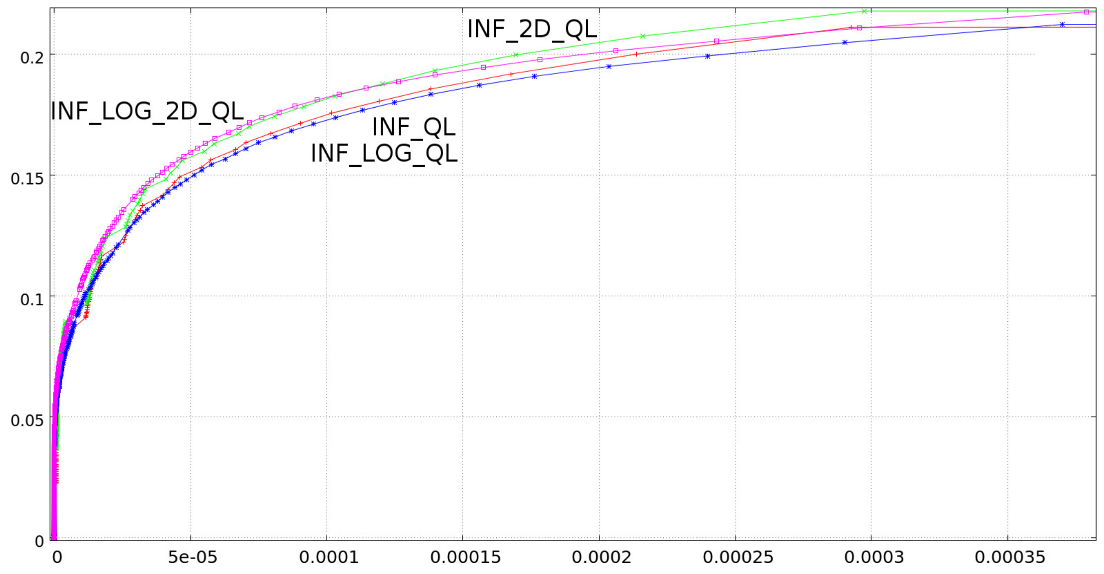


Figure A.46: ROC curve of INF_QL, INF_2D_QL, INF_LOG_QL, INF_LOG_2D_QL scores on the IMDb graph, zoomed in.

A.6 ROC/PR curves, ROC/PR-AUC: Webgraphs

	AA	**CN**	**RA**	**INF**	**LOG**	**2D**	**LOG_2D**
webND	0.312951	0.315807	0.207824	0.099200	0.498572	0.092195	<u>0.524134</u>
webSB	0.018812	0.015056	0.046012	0.094924	0.331292	0.093863	<u>0.447642</u>
webGL	0.081692	0.056055	0.092157	0.121209	0.414556	0.117714	<u>0.487771</u>
hudong	0.005033	0.007216	0.001579	0.003210	0.028695	0.001967	<u>0.033565</u>
baidu	0.001546	0.001088	0.001633	0.000052	0.003902	0.000007	<u>0.004271</u>
DBpedia	0.000355	0.000262	<u>0.001249</u>	0.000310	0.000477	0.000381	0.000524

Table A.11: PR-CAUC obtained by each tested score on six different webgraphs.

	AA	CN	RA	INF	LOG	2D	LOG_2D
webND	<u>0.917274</u>	0.917125	0.90837	0.88486	0.879633	0.885086	0.879639
webSB	<u>0.973477</u>	0.971452	0.92143	0.90377	0.900262	0.903772	0.900262
webGL	<u>0.956008</u>	0.955955	0.94873	0.91474	0.902688	0.914741	0.902688
hudong	<u>0.723551</u>	0.723109	0.67071	0.67069	0.668171	0.670758	0.668171
baidu	<u>0.844153</u>	0.843299	0.80086	0.78843	0.785428	0.788497	0.785428
DBpedia	0.003683	<u>0.005029</u>	0.00275	0.00403	0.004403	0.004187	0.004421

Table A.12: ROC-AUC obtained by each tested score on six different webgraphs.

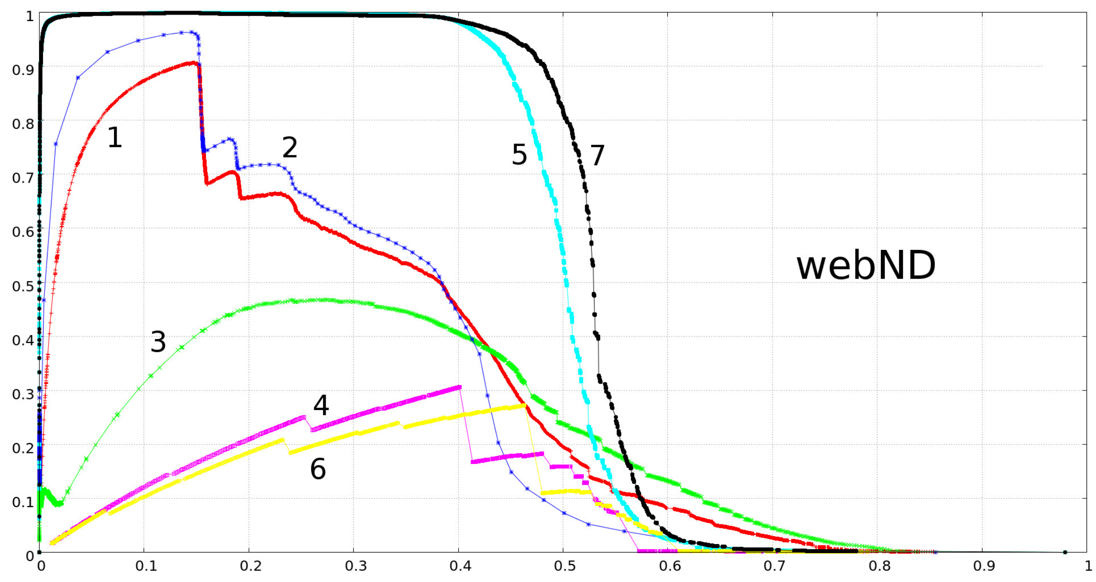


Figure A.47: PR curve of RA, AA, CN, INF, INF_2D, INF_LOG and INF_LOG_2D on the webND webgraph.

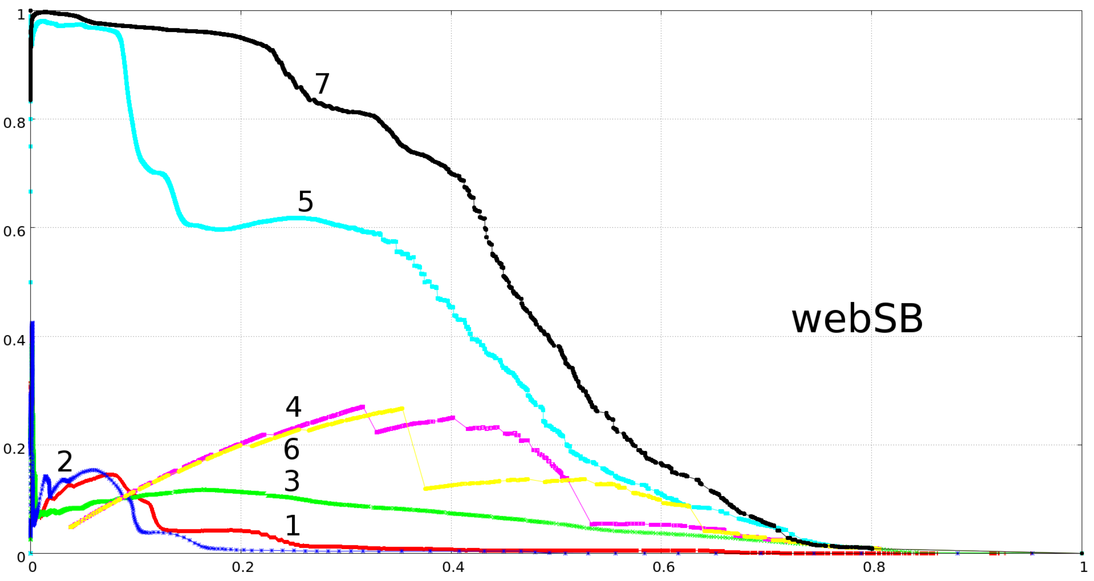


Figure A.48: PR curve of RA, AA, CN, INF, INF_2D, INF_LOG and INF_LOG_2D on the webSB webgraph.

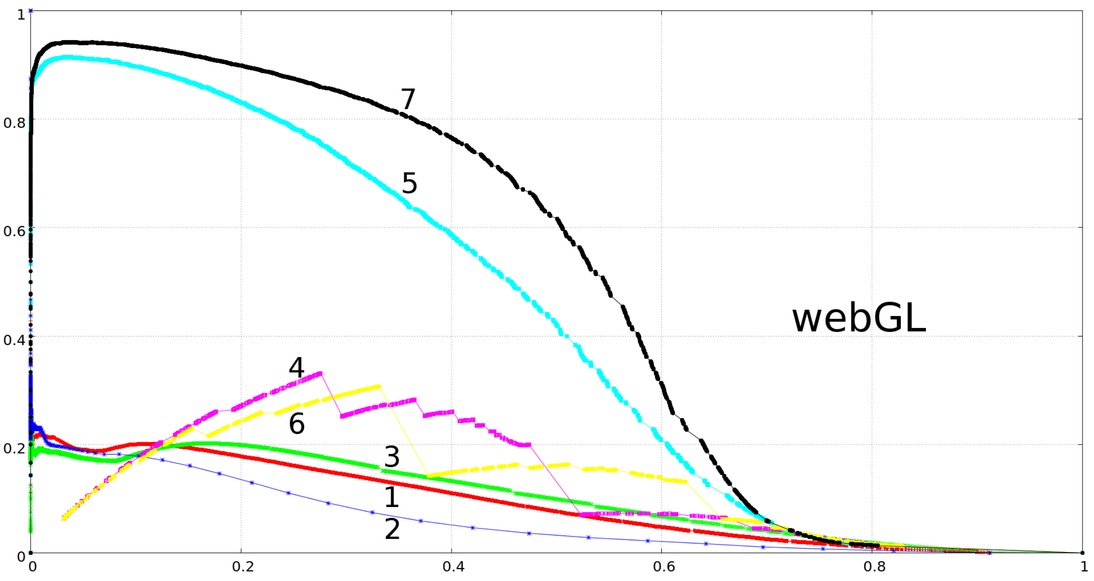


Figure A.49: PR curve of RA, AA, CN, INF, INF_2D, INF_LOG and INF_LOG_2D on the webGL webgraph.

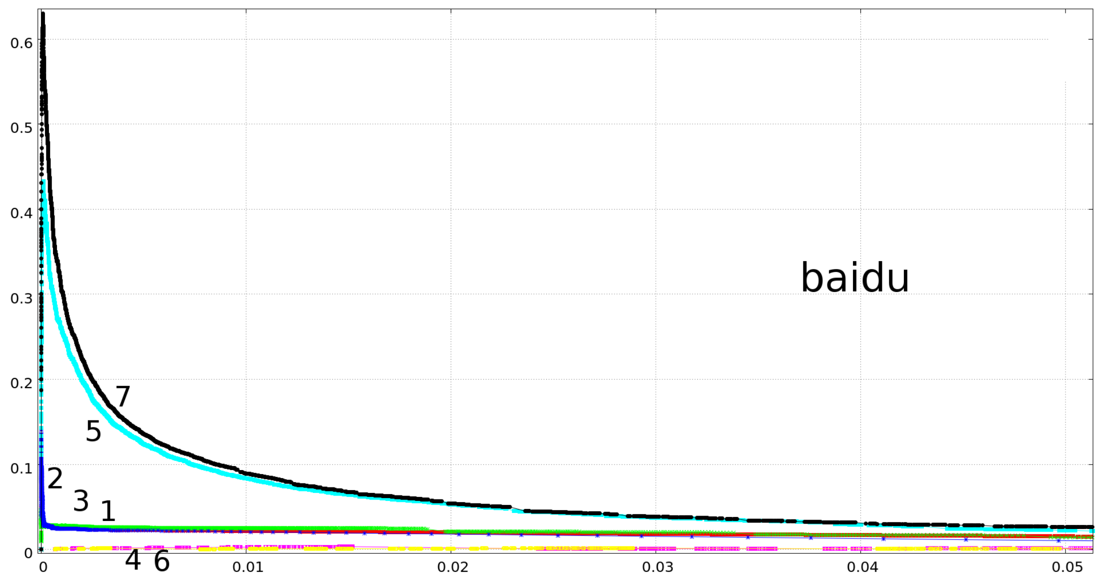


Figure A.50: PR curve of RA, AA, CN, INF, INF_2D, INF_LOG and INF_LOG_2D on the baidu webgraph.

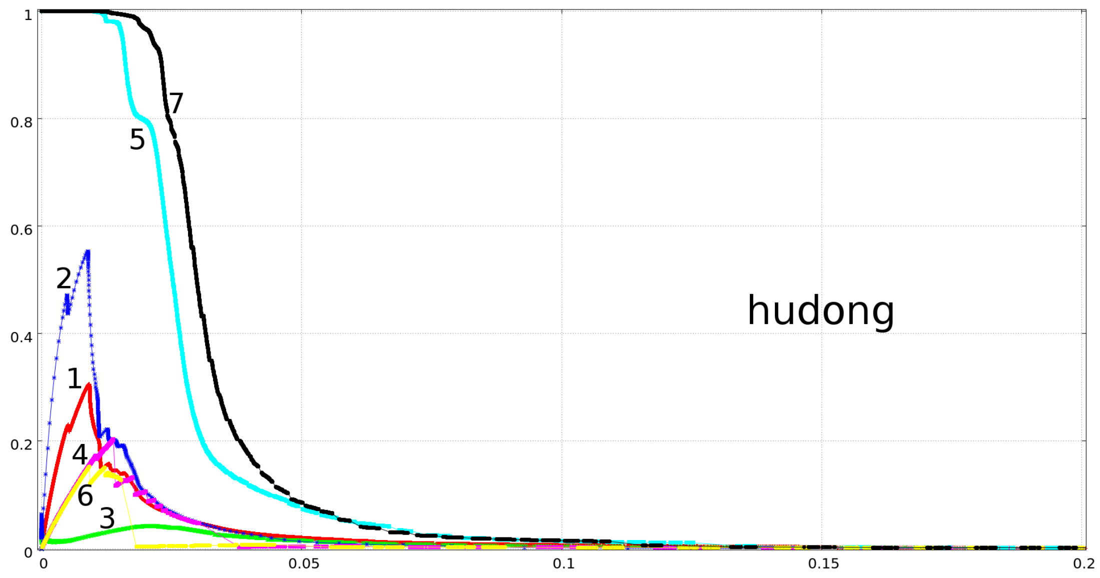


Figure A.51: PR curve of RA, AA, CN, INF, INF_2D, INF_LOG and INF_LOG_2D on the hudong webgraph.

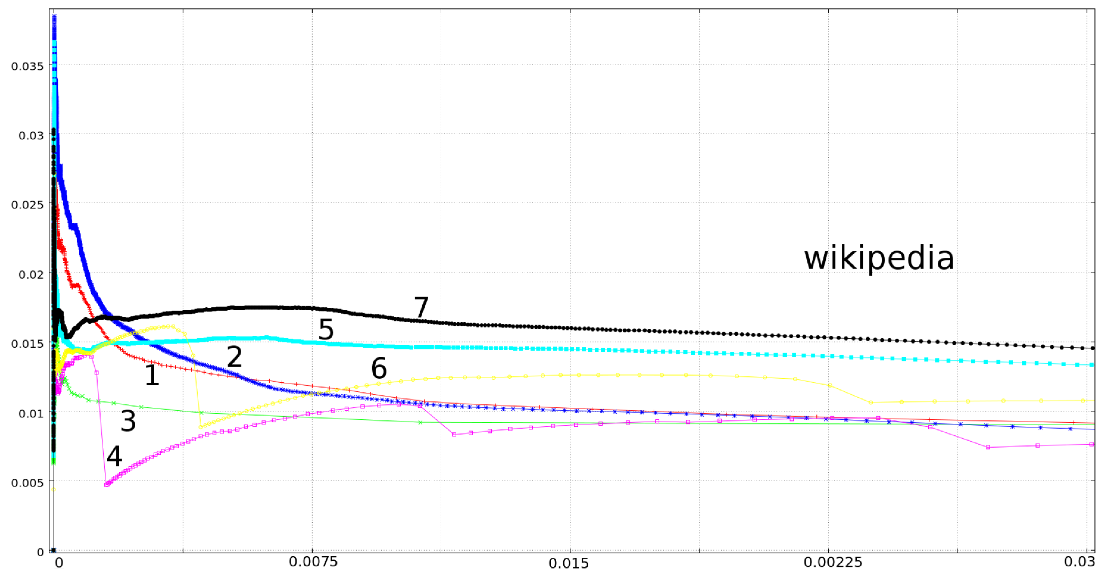


Figure A.52: PR curve of RA, AA, CN, INF, INF_2D, INF_LOG and INF_LOG_2D on the DBpedia webgraph.

References

Adafre, S. F. and de Rijke, M. (2005). Discovering missing links in Wikipedia. In *Proceedings of the 3rd international workshop on Link discovery*, pages 90–97. ACM. 10

Adamic, L. A. and Adar, E. (2003). Friends and neighbors on the web. *Social networks*, 25(3):211–230. 2, 7, 13, 50

Aggarwal, C. C., Xie, Y., and Yu, P. S. (2013). A framework for dynamic link prediction in heterogeneous networks. *Statistical Analysis and Data Mining*. 2, 10, 21, 22, 42

Agrawal, R., Srikant, R., et al. (1994). Fast algorithms for mining association rules. In *Proc. 20th Int. Conf. Very Large Data Bases, VLDB*, volume 1215, pages 487–499. 8

Airoldi, E. M., Blei, D. M., Fienberg, S. E., Xing, E. P., and Jaakkola, T. (2006). Mixed membership stochastic block models for relational data with application to protein-protein interactions. In *Proceedings of the international biometrics society annual meeting*, page I5. 2

Al Hasan, M., Chaoji, V., Salem, S., and Zaki, M. (2006). Link prediction using supervised learning. In *SDM'06: Workshop on Link Analysis, Counter-terrorism and Security*. 2

Albert, R., Jeong, H., and Barabási, A.-L. (1999). Internet: Diameter of the world-wide web. *Nature*, 401(6749):130–131. 51

ARB, O. (2013). OpenMP Application Program Interface, v.4.0. `http://www.openmp.org/mp-documents/OpenMP4.0.0.pdf`. 77, 82

Arel, I., Rose, D., and Coop, R. (2009). Destin: A scalable deep learning architecture with application to high- dimensional robust pattern recognition. In *Proc. AAAI Workshop on Biologically Inspired Cognitive Architectures*, pages 1150–1157. 112

Banerjee, S. and Pedersen, T. (2002). An adapted Lesk algorithm for word sense disambiguation using WordNet. In *Computational linguistics and intelligent text processing*, pages 136–145. Springer. 48

Bueno, J., Martinell, L., Duran, A., Farreras, M., Martorell, X., Badia, R. M., Ayguade, E., and Labarta, J. (2011). Productive cluster programming with OmpSs. In *Euro-Par 2011 Parallel Processing*, pages 555–566. Springer. 84

Bullmore, E. T. and Bassett, D. S. (2011). Brain graphs: graphical models of the human brain connectome. *Annual review of clinical psychology*, 7:113–140. 108

Burke, R. (2002). Hybrid recommender systems: Survey and experiments. *User modeling and user-adapted interaction*, 12(4):331–370. 2

Cao, M., Zhang, H., Park, J., Daniels, N. M., Crovella, M. E., Cowen, L. J., and Hescott, B. (2013). Going the distance for protein function prediction: a new distance metric for protein interaction networks. *PloS one*, 8(10):e76339. 12, 108

Chawla, N. V., Bowyer, K. W., Hall, L. O., and Kegelmeyer, W. P. (2002). SMOTE: Synthetic Minority Over-sampling Technique. *Journal of Artificial Intelligence Research*, 16:321–357. 24, 40

Clauset, A., Moore, C., and Newman, M. E. (2007). Structural inference of hierarchies in networks. In *Statistical network analysis: models, issues, and new directions*, pages 1–13. Springer. 99

Clauset, A., Moore, C., and Newman, M. E. (2008). Hierarchical structure and the prediction of missing links in networks. *Nature*, 453(7191):98–101. 2, 9, 10, 20, 42

Cook, D. J., Holder, L. B., Su, S., Maglothin, R., and Jonyer, I. (2001). Structural mining of molecular biology data. *Engineering in Medicine and Biology Magazine, IEEE*, 20(4):67–74. 1

Crovella, M. E. and Bestavros, A. (1997). Self-similarity in World Wide Web traffic: evidence and possible causes. *Networking, IEEE/ACM Transactions on*, 5(6):835–846. 51

Davis, J. and Goadrich, M. (2006). The relationship between Precision-Recall and ROC curves. In *Proceedings of the 23rd international conference on Machine learning*, pages 233–240. ACM. 40, 41

Dayarathna, M., Houngkaew, C., and Suzumura, T. (2012). Introducing scalegraph: an x10 library for billion scale graph analytics. In *Proceedings of the 2012 ACM SIGPLAN X10 Workshop*, page 6. ACM. 85

Dill, S., Kumar, R., McCurley, K. S., Rajagopalan, S., Sivakumar, D., and Tomkins, A. (2002). Self-similarity in the web. *ACM Transactions on Internet Technology (TOIT)*, 2(3):205–223. 51

Duran, A., Ayguadé, E., Badia, R. M., Labarta, J., Martinell, L., Martorell, X., and Planas, J. (2011). OmpSs: A proposal for programming heterogeneous multi-core architectures. *Parallel Processing Letters*, 21(02):173–193. 77, 84

Fawcett, T. (2004). ROC graphs: Notes and practical considerations for researchers. 40

Fire, M., Tenenboim, L., Lesser, O., Puzis, R., Rokach, L., and Elovici, Y. (2011). Link prediction in social networks using computationally efficient topological features. In *2011 IEEE 3rd international conference on social computing*, pages 73–80. 42

Fortunato, S. (2010). Community detection in graphs. *Physics Reports*, 486(3):75–174. 8

Foster, I. (1995). *Designing and building parallel programs.* Addison Wesley Publishing Company. 78

Franz, T., Schultz, A., Sizov, S., and Staab, S. (2009). Triplerank: Ranking semantic web data by tensor decomposition. In *The Semantic Web-ISWC 2009*, pages 213–228. Springer. 9

Getoor, L. and Diehl, C. P. (2005). Link mining: a survey. *ACM SIGKDD Explorations Newsletter*, 7(2):3–12. 1, 7, 8

Getoor, L. and Taskar, B. (2007). *Introduction to statistical relational learning*. MIT press. 1, 9

Gonzalez, J. E., Low, Y., Gu, H., Bickson, D., and Guestrin, C. (2012). Powergraph: Distributed graph-parallel computation on natural graphs. In *OSDI*, volume 12, page 2. 85

Guimera, R. and Amaral, L. A. N. (2005). Functional cartography of complex metabolic networks. *Nature*, 433(7028):895–900. 2

Guimerà, R. and Sales-Pardo, M. (2009). Missing and spurious interactions and the reconstruction of complex networks. *Proceedings of the National Academy of Sciences*, 106(52):22073–22078. 1

Guo, Y., Biczak, M., Varbanescu, A. L., Iosup, A., Martella, C., and Willke, T. L. (2014). How well do graph-processing platforms perform? an empirical performance evaluation and analysis. In *Parallel and Distributed Processing Symposium, 2014 IEEE 28th International*, pages 395–404. IEEE. 89

He, H. and Garcia, E. (2009). Learning from Imbalanced Data. *Knowledge and Data Engineering, IEEE Transactions on*, 21(9):1263–1284. 40

Helmstaedter, M., Briggman, K. L., Turaga, S. C., Jain, V., Seung, H. S., and Denk, W. (2013). Connectomic reconstruction of the inner plexiform layer in the mouse retina. *Nature*, 500(7461):168–174. 108

Holder, L. B. and Cook, D. J. (2009). Graph-Based Data Mining. *Encyclopedia of data warehousing and mining*, 2:943–949. 1

Hu, H., Yan, X., Huang, Y., Han, J., and Zhou, X. J. (2005). Mining coherent dense subgraphs across massive biological networks for functional discovery. *Bioinformatics*, 21(suppl 1):i213–i221. 8

Huang, Z., Li, X., and Chen, H. (2005). Link prediction approach to collaborative filtering. In *Proceedings of the 5th ACM/IEEE-CS joint conference on Digital libraries*, pages 141–142. ACM. 2

Inokuchi, A., Washio, T., and Motoda, H. (2000). An apriori-based algorithm for mining frequent substructures from graph data. In *Principles of Data Mining and Knowledge Discovery*, pages 13–23. Springer. 7, 8

Japkowicz, N. and Stephen, S. (2002). The class imbalance problem: A systematic study. *Intelligent data analysis*, 6(5):429–449. 26

Karrer, B. and Newman, M. E. (2011). Stochastic blockmodels and community structure in networks. *Physical Review E*, 83(1):016107. 8

Katz, L. (1953). A new status index derived from sociometric analysis. *Psychometrika*, 18(1):39–43. 13, 24

Khalil, A. and Liu, Y. (2004). Experiments with PageRank computation. *Indiana University, Department Computer Science. URL: http://www. cs. indiana. edu/ akhalil/Papers/pageRank. pdf.* 51

Kleinberg, J. (1999). Authoritative sources in a hyperlinked environment. *Journal of the ACM (JACM)*, 46(5):604–632. 7, 8

Knight, K. and Luk, S. K. (1994). Building a large-scale knowledge base for machine translation. In *Proceedings of the National Conference on Artificial Intelligence*, pages 773–773. JOHN WILEY & SONS LTD. 48

Krebs, V. E. (2002). Mapping networks of terrorist cells. *Connections*, 24(3):43–52. 2

Kunegis, J. (2013). KONECT: the Koblenz network collection. In *Proceedings of the 22nd international conference on World Wide Web companion*, pages 1343–1350. International World Wide Web Conferences Steering Committee. 51

Larrosa, J. and Cortés, U. (1995). A Framework for Abductive Rule Formation. *AI Communications*, 8(2):91–100. 36

Lehmann, J., Isele, R., Jakob, M., Jentzsch, A., Kontokostas, D., Mendes, P. N., Hellmann, S., Morsey, M., van Kleef, P., Auer, S., and Bizer, C. (2014). DBpedia - A Large-scale, Multilingual Knowledge Base Extracted from Wikipedia. *Semantic Web Journal*. 51

Lenat, D. B. (1995). CYC: A Large-Scale Investment in Knowledge Infrastructure. *Communications of the ACM*, 38:33–38. 49

Leskovec, J., Kleinberg, J., and Faloutsos, C. (2005). Graphs over time: densification laws, shrinking diameters and possible explanations. In *Proceedings of the eleventh ACM SIGKDD international conference on Knowledge discovery in data mining*, pages 177–187. ACM. 52

Liben-Nowell, D. and Kleinberg, J. (2007). The link-prediction problem for social networks. *Journal of the American society for information science and technology*, 58(7):1019–1031. 11, 13, 14, 50, 66

Lichtenwalter, R. N., Lussier, J. T., and Chawla, N. V. (2010). New perspectives and methods in link prediction. In *Proceedings of the 16th ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 243–252. ACM. 1, 11, 19, 20, 21, 22, 26, 50

Liu, W. and Lü, L. (2010). Link prediction based on local random walk. *EPL (Europhysics Letters)*, 89(5):58007. 11, 12, 14

Liu, X.-Y., Wu, J., and Zhou, Z.-H. (2009). Exploratory undersampling for class-imbalance learning. *Systems, Man, and Cybernetics, Part B: Cybernetics, IEEE Transactions on*, 39(2):539–550. 24, 40

Low, Y., Gonzalez, J., Kyrola, A., Bickson, D., Guestrin, C., and Hellerstein, J. M. (2010). Graphlab: A new framework for parallel machine learning. *arXiv preprint arXiv:1006.4990*. 85

Lü, L., Jin, C.-H., and Zhou, T. (2009). Similarity index based on local paths for link prediction of complex networks. *Physical Review E*, 80(4):046122. 11, 12, 14, 24, 33, 42, 50

Lü, L. and Zhou, T. (2010). Link prediction in weighted networks: The role of weak ties. *EPL (Europhysics Letters)*, 89(1):18001. 20

Lü, L. and Zhou, T. (2011). Link prediction in complex networks: A survey. *Physica A: Statistical Mechanics and its Applications*, 390(6):1150–1170. 1, 2, 9, 11, 12, 20, 21, 22, 42

Malewicz, G., Austern, M. H., Bik, A. J., Dehnert, J. C., Horn, I., Leiser, N., and Czajkowski, G. (2010). Pregel: a system for large-scale graph processing. In *Proceedings of the 2010 ACM SIGMOD International Conference on Management of data*, pages 135–146. ACM. 85

Mantrach, A., Yen, L., Callut, J., Francoisse, K., Shimbo, M., and Saerens, M. (2010). The sum-over-paths covariance kernel: A novel covariance measure between nodes of a directed graph. *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, 32(6):1112–1126. 1, 20

Meusel, R., Vigna, S., Lehmberg, O., and Bizer, C. (2014). Graph Structure in the Web - Revisited. In *23rd International World Wide Web Conference (WWW2014)*. 108

Miller, G. A. (1995). WordNet: A lexical database for English. *Communications of the ACM*, 38(11):39–41. 48

Murata, T. and Moriyasu, S. (2007). Link prediction of social networks based on weighted proximity measures. In *Web Intelligence, IEEE/WIC/ACM International Conference on*, pages 85–88. IEEE. 20, 50

Murata, T. and Moriyasu, S. (2008). Link prediction based on structural properties of online social networks. *New Generation Computing*, 26(3):245–257. 2, 13, 34, 42

Newman, M. E. (2001). Clustering and preferential attachment in growing networks. *Physical Review E*, 64(2):251021–251024. 7, 13

Nickel, M., Tresp, V., and Kriegel, H.-P. (2011). A three-way model for collective learning on multi-relational data. In *Proceedings of the 28th international conference on machine learning (ICML-11)*, pages 809–816. 1, 9, 42

Page, L., Brin, S., Motwani, R., and Winograd, T. (1999). The PageRank citation ranking: bringing order to the web. 8

Pastor-Satorras, R., Vázquez, A., and Vespignani, A. (2001). Dynamical and correlation properties of the Internet. *Physical review letters*, 87(25):258701. 51

Ravasz, E. and Barabási, A.-L. (2003). Hierarchical organization in complex networks. *Physical Review E*, 67(2):026112. 51

RDF_Working_Group (2014). RDF Schema 1.1 (Work in progress). Technical report, W3C, http://www.w3.org/TR/2014/PER-rdf-schema-20140109/. 49

Roy, A., Mihailovic, I., and Zwaenepoel, W. (2013). X-stream: edge-centric graph processing using streaming partitions. In *Proceedings of the Twenty-Fourth ACM Symposium on Operating Systems Principles*, pages 472–488. ACM. 85

Song, H. H., Cho, T. W., Dave, V., Zhang, Y., and Qiu, L. (2009). Scalable proximity estimation and link prediction in online social networks. In *Proceedings of the 9th ACM SIGCOMM conference on Internet measurement conference*, pages 322–335. ACM. 50

Sutskever, I., Salakhutdinov, R., and Tenenbaum, J. B. (2009). Modelling Relational Data using Bayesian Clustered Tensor Factorization. In *NIPS*, pages 1821–1828. 9

Taskar, B., Wong, M. F., Abbeel, P., and Koller, D. (2003). Link prediction in relational data. 7

Tong, H., Faloutsos, C., and Koren, Y. (2007). Fast direction-aware proximity for graph mining. In *Proceedings of the 13th ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 747–756. ACM. 50

Tylenda, T., Angelova, R., and Bedathur, S. (2009). Towards time-aware link prediction in evolving social networks. In *Proceedings of the 3rd Workshop on Social Network Mining and Analysis*, page 9. ACM. 2

Ueno, K. and Suzumura, T. (2012). Highly scalable graph search for the graph500 benchmark. In *Proceedings of the 21st international symposium on High-Performance Parallel and Distributed Computing*, pages 149–160. ACM. 110

Vicknair, C., Macias, M., Zhao, Z., Nan, X., Chen, Y., and Wilkins, D. (2010). A comparison of a graph database and a relational database: a data provenance perspective. In *Proceedings of the 48th annual Southeast regional conference*, page 42. ACM. 8

Von-Mering, C., Krause, R., Snel, B., Cornell, M., Oliver, S. G., Fields, S., and Bork, P. (2002). Comparative assessment of large-scale data sets of protein–protein interactions. *Nature*, 417(6887):399–403. 2, 108

Washio, T. and Motoda, H. (2003). State of the art of graph-based data mining. *ACM SIGKDD Explorations Newsletter*, 5(1):59–68. 1, 7

Wasikowski, M. and Chen, X. W. (2010). Combating the small sample class imbalance problem using feature selection. *Knowledge and Data Engineering, IEEE Transactions on*, 22(10):1388–1400. 24, 40

Watanabe, M. and Suzumura, T. (2013). How social network is evolving? A preliminary study on billion-scale twitter network. In *Proceedings of the 22nd international conference on World Wide Web companion*, pages 531–534. International World Wide Web Conferences Steering Committee. 2, 108

Weiss, G. M. (2004). Mining with rarity: a unifying framework. *ACM SIGKDD Explorations Newsletter*, 6(1):7–19. 24, 40

Yang, Y., Lichtenwalter, R. N., and Chawla, N. V. (2014). Evaluating link prediction methods. *Knowledge and Information Systems*, pages 1–32. 40, 41, 45, 56

Yuan, P., Zhang, W., Xie, C., Jin, H., Liu, L., and Lee, K. (2014). Fast iterative graph computation: a path centric approach. In *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*, pages 401–412. IEEE Press. 85, 110

Zhang, Q.-M., Lü, L., Wang, W.-Q., Zhou, T., et al. (2013). Potential theory for directed networks. *PloS one*, 8(2):e55437. 20

Zhou, T., Lü, L., and Zhang, Y.-C. (2009). Predicting missing links via local information. *The European Physical Journal B*, 71(4):623–630. 11, 13, 14

Author dissemination related with the thesis proposal

Dario Garcia-Gasulla and Ulises Cortés, *Hierarchical inference on semantic graphs applied to Cyc*, Proceedings of the 16th International Conference of the Catalan Association of Artificial Intelligence, 2013.

Dario Garcia-Gasulla and Ulises Cortés, *Link Prediction in Very Large Directed Graphs: Exploiting Hierarchical Properties in Parallel*, 3rd Workshop on Knowledge Discovery and Data Mining Meets Linked Open Data - 11th Extended Semantic Web Conference, 2014.

Dario Garcia-Gasulla , *La febre i el poder de les dades*, ACIA Nodes num. 55 , 2014.

Dario Garcia-Gasulla and Ulises Cortés, *Evaluating Link Prediction on Large Graphs*, Proceedings of the 18th International Conference of the Catalan Association for Artificial Intelligence, 2015.

Dario Garcia-Gasulla and Ulises Cortés, *Scalable Hyperlink Prediction for the WWW*, submitted to the IEEE International Conference on Data Mining, 2015.

Other activities of the author related with the thesis proposal

Programme chair and organizer of the 1st High Performance Graph Mining workshop, co-located with the 21st ACM SIGKDD Conference on Knowledge Discovery and Data Mining (KDD 2015), Sydney, Australia.

Author publications during the research period

Dario Garcia-Gasulla, Sergio Alvarez-Napagao, Arturo Tejeda-Gómez, Luis Oliva-Felipe and Javier Vázquez-Salceda, *Social network data analysis for event detection*, 21th European Conference on Artificial Intelligence (ECAI), 2014.

Javier Vázquez-Salceda, Sergio Alvarez-Napagao, Arturo Tejeda-Gómez, Luis Oliva, Dario Garcia-Gasulla, Ignasi Gómez-Sebastià and Victor Codina, *Making Smart Cities Smarter: Using Artificial Intelligence Techniques for Smarter Mobility*, *Proceedings of the 3rd International Conference on Smart Grids and Green IT Systems*, 2014.

Ignasi Gómez-Sebastià, Luis Oliva-Felipe, Sergio Alvarez-Napagao, Dario Garcia-Gasulla, Arturo Tejeda-Gómez and Javier Vázquez-Salceda, *A functional approach for disruptive event discovery and policy monitoring in mobility scenarios*, 7th European Lisp Symposium, 2014.

Dario Garcia-Gasulla, Arturo Tejeda-Gómez, Sergio Alvarez-Napagao, Luis Oliva-Felipe and Javier Vázquez-Salceda, *Detection of events through collaborative social network data*, 6th International Workshop on Emergent Intelligence on Networked Agents (WEIN), 2014.

Sergio Alvarez-Napagao, Arturo Tejeda-Gómez, Luis Oliva, Dario Garcia-Gasulla, Victor Codina and Javier Vázquez-Salceda, *Urban context detection and context-aware recommendation via networks of Humans as Sensors*, 5th International Workshop on Collaborative Agents – Research & Development (CARE), AAMAS 2014.

AUTHOR PUBLICATIONS DURING THE RESEARCH PERIOD

Hannu Järvinen, Dario Garcia-Gasulla, and Ulises Cortés, *A push-based agent communication model empowering assistive technologies*, International Journal on Artificial Intelligence Tools (IJAIT), **23**, **1**, 2014.

Jonathan Moreno and Ulises Cortés and Dario Garcia-Gasulla and Sergio Alvarez-Napagao and Ignasi Gómez-Sebastià, *Applying COAALAS to SPiDer*, 16th International Conference of the Catalan Assotiation of Artificial Intelligence, 2013.

Jonathan Moreno, Dario Garcia-Gasulla, and Ulises Cortés, *SPiDer: Integrating smart medical dispensing with multiple purpose elder assistance systems*, 14th Conference on Artificial Intelligence in Medicine - VIII Workshop on Agents Applied in Health Care (A2HC), 2013.

Ignasi Gómez-Sebastià, Sergio Alvarez-Napagao, Dario Garcia-Gasulla, and Ulises Cortés, *Situated agents and humans in social interaction for elderly healthcare: the case of COAALAS*, 14th Conference on Artificial Intelligence in Medicine - VIII Workshop on Agents Applied in Health Care (A2HC), 2013.

Hannu Järvinen and Dario Garcia-Gasulla, *Position paper: An Agent-oriented Architecture for Building Automation Systems applied to Assistive Technologies*, 11th international conference on autonomous agents and multiagents systems - 7th Workshop on Agents Applied in Health Care (A2HC), p. 71, 2012.

Ignasi Gómez-Sebastià, Dario Garcia-Gasulla, Sergio Alvarez-Napagao, Javier Vázquez-Salceda, and Ulises Cortés, *Towards an implementation of a social electronic reminder for pills*, 11th International conference on Autonomous Agents and Multiagents Systems - 7th Workshop on Agents Applied in Health Care (A2HC), p. 61, 2012.

Dario Garcia-Gasulla, Manel Poch, Juan Carlos Nieves, Ulises Cortés, and Claudia Turon, *A logic-based environmental decision support system for the management of horizontal subsurface constructed wetlands*, Ecological Engineering **47**, p.44–55, 2012.

Juan Carlos Nieves and Dario Garcia-Gasulla and Montse Aulinas and Ulises Cortés, *An operational approach for implementing normative agents in urban wastewater systems*, Computación y Sistemas, **16**, no. 1, p.27–42, 2012.