Univerza v Ljubljani
Fakulteta za Računalništvo in Informatiko
Fakulteta za Matematiko in Fiziko

Klemen Simonič

Strukturne lastnosti v omrežjih in njihova uporaba v napovedovanju manjkajočih lastnosti

Diplomsko delo

na interdisciplinarnem univerzitetnem študiju

Mentor: Vladimir Batagelj
Somentor: Primož Škraba

Ljubljana, 2013

University of Ljubljana
Faculty of Computer and Information Science
Faculty of Mathematics and Physics

Klemen Simonič

Network structural properties and their application to missing property prediction

Diploma Thesis

University Study Program of Computer Science and Mathematics

Mentor: Vladimir Batagelj
Co-mentor: Primož Škraba

Ljubljana 2013

Univerza v Ljubljani
Fakulteta za Računalništvo in Informatiko
Fakulteta za Matematiko in Fiziko

Klemen Simonič

Strukturne lastnosti v omrežjih in njihova uporaba v napovedovanju manjkajočih lastnosti

Diplomsko delo

na interdisciplinarnem univerzitetnem študiju

Mentor: Vladimir Batagelj
Somentor: Primož Škraba

Ljubljana, 2013

Besedilo je oblikovano z urejevalnikom besedil LaTeX.

Univerza v Ljubljani
Fakulteta za računalništvo in informatiko

Tržaška 25
1001 Ljubljana, Slovenija
telefon: 01 476 84 11
01 476 83 87
faks: 01 426 46 47
01 476 87 11
www.fri.uni-lj.si
e-mail: dekanat@fri.uni-lj.si

Št. naloge: 00044/2013
Datum: 28.01.2013

Univerza v Ljubljani, Fakulteta za računalništvo in informatiko ter Fakulteta za matematiko in fiziko izdaja naslednjo nalogo:

Kandidat: **KLEMEN SIMONIČ**

Naslov: **STRUKTURNE LASTNOSTI V OMREŽJIH IN NJIHOVA UPORABA V NAPOVEDOVANJU MANJKAJOČIH LASTNOSTI**

NETWORK STRUCTURAL PROPERTIES AND THEIR APPLICATION TO MISSING PROPERTY PREDICTION

Vrsta naloge: Diplomsko delo univerzitetnega študija

Tematika naloge:

V diplomski nalogi preučite problem napovedanja manjkajočih lastnosti (tipi relacij) v danem omrežju. Pri tem se oprite na strukturne lastnosti omrežja in na tej osnovi opredelite ustrezno mero različnosti, ki bo osnova za postopek napovedovanja. Postopek naj učinkovito deluje tudi za velika omrežja. Postopek izvedite v izbranem programskem jeziku in ga preizkusite na izbranih omrežjih iz zbirke Linked Open Data.

Mentor:

prof. dr. Vladimir Batagelj

Somentor:

znan. sod. dr. Primož Škraba

Dekan Fakultete za računalništvo in informatiko:

prof. dr. Nikolaj Zimic

Dekan Fakultete za matematiko in fiziko:

akad. prof. dr. Franc Forstnerič

Univerza v Ljubljani
Fakulteta za računalništvo in informatiko

Tržaška 25
1001 Ljubljana, Slovenija
telefon: 01 476 84 11
01 476 83 87
faks: 01 426 46 47
01 476 87 11
www.fri.uni-lj.si
e-mail: dekanat@fri.uni-lj.si

No. of dissertation: 00044/2013
Date: 28.01.2013

University of Ljubljana, Faculty of Computer and Information Science issues the following dissertation:

Candidate: **KLEMEN SIMONIČ**

Title: **NETWORK STRUCTURAL PROPERTIES AND THEIR APPLICATION TO MISSING PROPERTY PREDICTION**

Type of dissertation: Undergraduate dissertation

Topic of the thesis:

In your diploma thesis make a study of the problem of missing properties (types of relation) prediction in a given network. To solve this problem, define a suitable dissimilarity measure based on network structural properties, that will be the basis for a prediction algorithm. The algorithm should perform efficiently also on large networks. Implement and test the algorithm on selected networks from Linked Open Data datasets.

Mentor:

Prof. Vladimir Batagelj, PhD

Dean of Faculty of Computer and Information Science

Prof. Nikolaj Zimic, PhD

Comentor:

Assist. Prof. Primož Škraba, PhD

Dean of Faculty of Mathematics and Physics

Acad. Prof. Franc Forstnerič, PhD

Izjava o avtorstvu diplomskega dela

Spodaj podpisani Klemen Simonič, z vpisno številko **63080156**, sem avtor diplomskega dela z naslovom:

Strukturne lastnosti v omrežjih in njihova uporaba v napovedovanju manjkajočih lastnosti.

S svojim podpisom zagotavljam, da:

- sem diplomsko delo izdelal samostojno pod mentorstvom Vladimirja Batagelja in somentorstvom Primoža Škrabe,
- so elektronska oblika diplomskega dela, naslov (slov., angl.), povzetek (slov., angl.) ter ključne besede (slov., angl.) identični s tiskano obliko diplomskega dela,
- soglašam z javno objavo elektronske oblike diplomskega dela v zbirki "Dela FRI".

Ljubljana, 24. april 2013 Podpis avtorja:

First, I would like to thank to my mentor prof. Vladimir Batagelj, who led me through the process of writing a diploma and gave me many important advices and comments.

I would especially like to thank to two of my colleges Jan Rupnik and Primož Škraba. We spent numerous hours working together on a problem "Predicting Missing Properties." Many times our discussions went beyond the main problem and their experiences and wisdom shaped my view of the science. Jan's and Primož's advices were crucial for the outcome of this research.

Marko Grobelnik, project manager of the department where I have been working in the last four years, is the person who introduced me to the research field. I am grateful to him for giving me this opportunity, which initiated my "research career."

Help and support from my mother Marjetka were crucial in the last years. Her effort and hard work have always inspired me and helped me move forward. Close relationships with my extended-family, grand-parents, uncles, aunts and cousins have motivated me even further to work hard and pursue my dream.

Contents

Acronyms and Abbreviations

LOD	Linked Open Data
RDF	Resource Description Framework
DB	DBpedia
WCC	Weakly Connected Component
SCC	Strongly Connected Component
Dir	Directed
Nbh	Neighbor
SP	Shortest Paths
DK	Diffusion Kernel

Povzetek

Dandanes ustvarimo ogromne količine podatkov, ki prihajajo iz različnih virov: znanstvenih raziskav, senzorjev in socialnih komunikacij v obliki blogov, novic in sporočil Twitter. Podatki so razumljivejši, če so medsebojno povezani – povezave med podobnimi članki na Wikipediji nam omogočajo lažje razumevanje področja, iskalniki bazirajo na povezavah med spletnimi stranmi. Ideja metode Linked Data je povezovanje še nepovezanih stvari s pomočjo interneta.

Povezave v podatkih definirajo strukturiranost podatkov – podatkovne baze so primer "popolno" strukturiranih podatkov, kjer podatkovna shema predpiše strukturo posamezni entiteti. Veliko podatkovnih zbirk v Linked Data nima definirane sheme – podatki bi lahko bili bolje opisani. Pogosto se tudi uporablja različna terminologija za opisovanje semantično enakih podatkov.

Linked Data vsebuje več sto podatkovnih zbirk, ki so jih objavili posamezniki ali organizacije v obliki formata RDF (Resource Description Framework). Vsaka podatkovna zbirka vsebuje množico trditev ali povezav, ki povezujejo entitete oziroma vire v podatkih. Kvaliteta in velikost podatkovnih zbirk je različna od primera do primera, vendar so podatki o entitetah pogosto pomanjkljivi. V luči te "nepopolne" strukture se pojavi naravno vprašanje: do kakšne mere lahko samodejno identificiramo manjkajoče podatke?

Konkreten primer grafa RDF je prikazan na sliki 1.1 (Figure 1.1), kjer so izpostavljene tri entitete oziroma objekti, *Audi, Mercedes-Benz* in *Fiat*.

Opazimo lahko, da ima *Fiat* naslednje lastnosti: *location, founder, industry, manufacturer*. Objekti imajo nekaj skupnih lastnosti *(founder, manufacturer)*, vendar so tudi nekatere lastnosti *(parentCompany, name)*, ki jih imata samo *Audi* in *Mercedes-Benz*. Primeri manjkajočih lastnosti objekta *Fiat* bi lahko bili: *name, parentCompany, subsidiary, formationYear*.

V diplomskem delu je predstavljen pristop k reševanje tega problema, ki temelji na iskanju podobnih objektov in uporabe njihovih lastnosti za napovedovanje morebitnih manjkajočih lastnosti. V primeru objekta *Fiat*, bi lahko bil seznam podobnih objektov in njihovih podobnosti sledeč: *(0.7, Mercedes-Benz), (0.6, Audi)*. Seznam manjkajočih lastnosti in njihovih uteži pa bi lahko izgledal: *(0.62, name), (0.54, parentCompany), (0.32, formationYear), (0.13, subsidiary)*.

Primer uporabe našega pristopa je, da pomaga identificirati manjkajoče lastnosti v dani podatkovni zbirki. S pomočjo priporočanja verjetno manjkajočih lastnosti, bi lahko uporabniki vodeno dodajali nova dejstva v podatkovno zbirko. Popularen primer je "Google Knowledge Graph" [21], ki pomaga poiskati uporabniku smiselnejše rezultate. Naš priporočilni sistem, bi lahko učinkovito in strukturirano predlagal uporabniku tip informacije, ki bi jo nato uporabnik vnesel v obstoječi graf znanja.

Glavna ideja našega pristopa je, da danemu objektu predlagamo manjkajoče lastnosti na podlagi njemu podobnih objektov – podobno kot priporočanje uporabnikovih preferenc na podlagi drugih uporabnikov preferenc. Ogrodje našega pristopa je sledeče: vhod v metodo je objekt skupaj z množico njegovih obstoječih lastnosti. Izhod metode je uteženo zaporedje manjkajočih lastnosti za vhodni objekt, kjer utež določa kako pomembna je manjkajoča lastnost. Pristop je sestavljen iz treh glavnih korakov: najprej poiščemo množico podobnih objektov za dani objekt. Ta korak je odvisen od naše definicije podobnosti med objekti. Nato za vsakega izmed podobnih objektov določimo obstoječe lastnosti. V tretjem koraku pa uporabimo obstoječe lastnosti podobnih objektov, da izračunamo manjkajoče lastnosti vhodnega objekta.

POVZETEK

Pomembno vprašanje v reševanju našega problema je: kako poiskati podobne objekte? Definirali smo številne strukturne lastnosti grafa oziroma omrežja, ki inducirajo različnost oziroma podobnost med objekti. Primera enostavne strukturne lastnost omrežja so porazdelitev stopenj točk in porazdelitev oznak na povezavah. Povezanost omrežja opisujejo najkrajše poti med dvema točkama. Lokalno strukturo grafa opisujejo podgrafi, kot so polno povezani podgrafi ali omrežni motivi. Nekatere imed teh strukturnih lastnosti se ne da učinkovito izračunati oziroma shraniti, še posebno ne na velikih podatkovnih zbirkah, kar je naš primer. Zato smo za računanje podobnosti med objekti uporabili več "enostavnih" metod, ki smo jih razdelili v dve kategoriji. Lokalne metod opisuje lokalno strukturo grafa tako, da upoštevajo porazdelitev oznak na povezavah, ki so (ne)posredno povezane z objektom, medtem ko globalne metode izkoristijo globalne lastnosti grafa, kot so najkrajše poti med točkami.

Za evaluacijo smo uporabili tri podatkovne zbirke iz Linked Data: DBmapped, DBraw in Freebase, tabela 5.1 (Table 5.1). Velikost posamezne podatkove zbirke sega tudi čez 140 milijonov točk in 600 milijonov povezav. Strukturiranost oziroma čistost podatkov niha od zbirke do zbirke, kar nakazujejo število različnih lastnosti, ki jih je v primeru DBraw zbirke več kot 40 tisoč.

Opravili smo obsežno evaluacijo s prej omenjenimi metodami na vseh treh podatkovnih zbirkah. V evaluaciji smo uporabili številne temeljne oziroma osnovne metode in preizkusili njihovo delovanje. Evaluacija je potekala na način, da smo vhodnemu objektu izbrisali nekaj povezav in smo jih nato poskušali napovedati z dano metodo kot manjkajoče lastnosti. Višje so bile uvrščene izbrisane lastnosti, boljšo oceno je dobila metoda. V povprečju so bile lokalne metode boljše od globalnih metod, vendar je bila med njima prisotna izrazita nekoreliranost – za določene objekte so globalne metode delovale boljše kot lokalne in obratno.

Veliko dela smo posvetili tudi skrbni programski izvedbi, ki je omogočila izvajanje eksperimentov na tako velikih podatkovnih zbirkah. Uporabili

smo številne pohitritve (npr. K-najbljižjih sosedov, razbitje prostora na manjše dele), ki so nam omogočile izvajanje eksperimentov v doglednem času. Končni izdelek je demonstrativni program LODminer `http://lodminer.net`, kjer lahko uporabnik preizkusi delovanje sistema.

Ključne besede: Linked Data, analiza grafov, omrežja, strukturne lastnosti, manjkajoče lastnosti, napovedovanje.

Abstract

The volume of available structured data is increasing, particularly in the form of *Linked Data*, where relationships between individual pieces of data are encoded by a graph-like structure. Despite increasing scales of the data, the use and applicability of these resources is currently limited by mistakes and omissions in the linking data.

In this diploma thesis, we look at the problem of predicting potential instance properties (types of relations). Given a specific *query node* in our multigraph dataset, can we correctly rank possibly omitted properties? We propose a method based on leveraging properties from similar nodes in our dataset.

In order to compute similar nodes, we define various network structural properties, which induce dissimilarities between nodes. These structural properties are based on either *local* or *global* processing of the underlying network. Since their complexity highly varies, a special treatment needs to be considered when dealing with networks containing hundreds of millions of nodes and edges.

In our tool *LODminer*, we use weighted averages of property frequency vectors over a set of similar nodes to determine the most likely missing instance property. We investigate the performance of different dissimilarities and compare them to several other methods on three large-scale datasets, two based on DBpedia and one based on Freebase.

ABSTRACT

Mathematics Subject Classification [MSC2010]: 68R10 [Graph theory], 68T30 [Knowledge representation], 05C82 [Small world graphs, complex networks], 91D30 [Social networks].

CCS Categories and Subject Descriptors [1998 system]: G.2.2 [Graph Theory], I.2.4 [Knowledge Representation Formalisms and Methods]: Semantic networks, H.2 [Database Management]: Database Applications – Data Mining.

Keywords: Linked Data, Graph Mining, Network, Structural Properties, Missing Properties, Prediction.

Chapter 1

Introduction

In today's world, we are producing data at an astounding rate: scientific data, sensor data, as well as social data in the form of blogs, news articles and Twitter. Data is much easier to interpret when it is interconnected — the internal links on Wikipedia make the understanding of topics much easier due to easy access to references; when searching the Web, we find relevant content through links. The idea behind Linked Data is to connect previously unlinked, but related data, using the Web.

Links in data also define how structured it is — databases, for example, are "perfectly" structured data where a schema defines the descriptions for each of the entities. Many organic datasets in Linked Data do not have a predefined schema and entities are rather poorly described. Often, a varied vocabulary is used to describe entities, with semantically identical entities having different descriptions.

An often used framework to publish data on the web is the *Resource Description Framework (RDF)* [34]. RDF data is published in a collection of triples, each consisting of a subject, a predicate and an object. Each triple can also be illustrated as a $node_1$-*arc*-$node_2$ structure, where $node_1$ and $node_2$ represent subject and object, respectively, and *arc* links the $node_1$, $node_2$ and is labeled with a predicate. A set of such structures (triples) is also called an *RDF graph*.

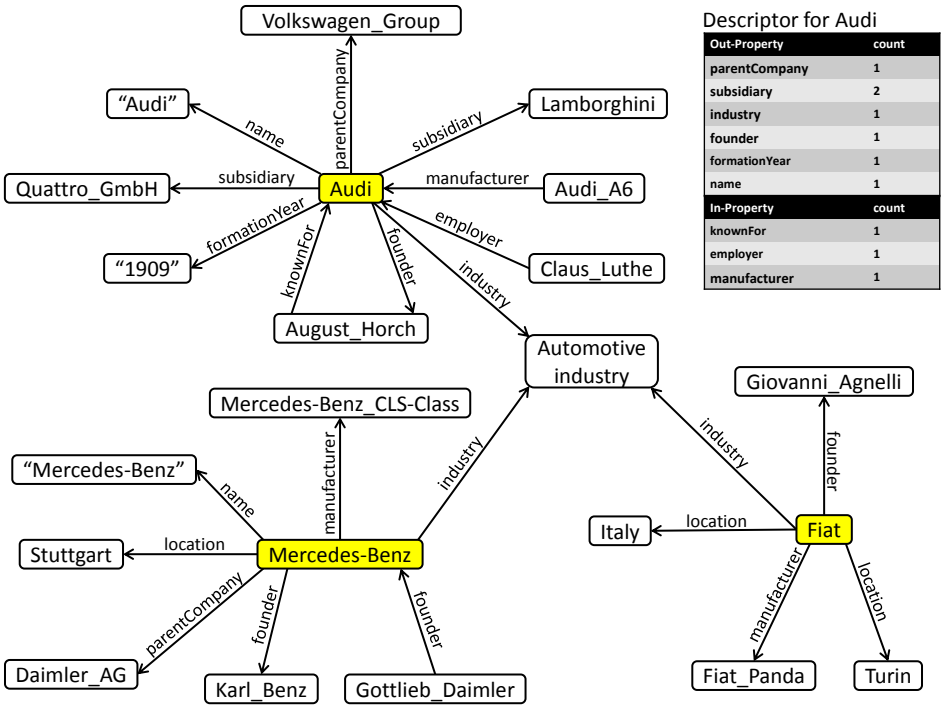


Figure 1.1: An example of a small LOD dataset. The graph describes three resources, *Audi, Mercedes-Benz and Fiat*, and presents the associated properties.

Linked Data [7] now contains more than a hundred datasets published by individuals and organizations in a RDF format. Each dataset contains a set of assertions or links connecting facts or resources in the dataset. The datasets vary in size and quality, but information about resources is often incomplete. In light of this imperfect structure, a natural question is: to what extent can we automatically identify what data is missing?

As an example, consider the RDF graph in Figure 1.1. The graph highlights three resources or objects, *Audi, Mercedes-Benz and Fiat*. We can see that *Fiat* has the following four properties: *location, founder, industry, manufacturer*. The objects share some common properties, *founder, manufacturer*, but there are some properties (*parentCompany, name*) that the *Audi* and *Mercedes-Benz* objects have and the *Fiat* object is missing. A

few examples of *Fiat's* missing properties could be *name, parentCompany, subsidiary, formationYear*.

In this thesis, we present an approach to this problem based on finding similar objects and using their properties to predict possible missing properties. In the case of *Fiat*, the list of similar objects and their similarities could be *(0.7, Mercedes-Benz), (0.6, Audi)*. The list of missing properties and their relevance could be *(0.62, name), (0.54, parentCompany), (0.32, formationYear), (0.13, subsidiary)*.

One application of this approach is to help datasets "ask" for missing data. By recommending likely missing properties, users could be guided in adding additional assertions to the datasets. For example, Google recently published the Knowledge Graph [21], which it is hoped will help people find more meaningful results. This type of recommendation system could be used to suggest what type of additional knowledge can be inserted into the graph in an efficient way.

Our contributions are summarized as follows:

- We describe various network structural properties that can induce dissimilarities between nodes.
- We present a general approach to predicting missing properties based on finding similar objects in the data.
- A scalable implementation of the approach above allowing for the analysis of very large graphs with over 600M edges and 140M nodes.
- An empirical analysis of the performance of different dissimilarities along with an extensive comparison with performance with other methods on three different datasets.
- The analysis also provides insights into the structural similarities and differences of the three datasets.
- A demonstration recommendation system for producing ranks of missing properties (`http://lodminer.net`) on the DBpedia dataset.

The thesis is organized as follows. We first give an overview of related

work in Chapter 2. Network structural properties are presented in Chapter 3. Chapter 4 presents the general approach taken in solving our problem. The datasets that we consider are presented in Chapter 5. Extensive evaluation and experiments are presented in Chapter 6. Chapter 7 describes crucial implementation details allowing us to solve the problem at scale. Finally, discussion and future work are presented in the chapter 8.

This work is an extended version of the paper *Predicting Missing Properties in Linked Data Datasets* [31], which was submitted to *19th ACM SIGKDD Conference on Knowledge Discovery and Data Mining*. The paper reports the results of the research that was conducted by *Klemen Simonič, Jan Rupnik* and *Primož Škraba* at Jožef Stefan Institute.

Chapter 2

Related Work

There has been an enormous recent interest in Linked Data. Work in this area typically falls into three categories: publishing of domain-specific datasets [11, 17], applications of existing datasets [4, 28] and algorithms for construction, correction or cleaning of Linked Data [13, 24].

The work in this area generally concentrates more on semantic approaches, however our work is closely related to the work presented in [26]. The authors addressed the same problem of predicting missing links in data and proposed two approaches to instance property recommendation: a classification-based and co-occurrence-based approach. The classification-based approach could be seen as a special case of our approach, where the smoothing parameter is sufficiently large. The co-occurrence based approach is based on association rule mining for recommendation and we included it in our evaluation. An important difference between our work and [26] lies in the scale and level of detail of the evaluation. The largest LOD in our study includes 6×10^8 triples and 1.4×10^7 nodes, whereas the largest set considered in [26] is 2.8×10^4 triples and 4.4×10^3 nodes.

The problem of predicting missing links occurs in several other contexts. There is the related problem of inferring the domain range of binary relations occurs in schema induction [33, 27]. A key difference is that we focus on recommending properties for individual instances (with no class information),

rather than object class based predictions. A data driven approach for inferring unseen triples was recently proposed in [25]. The approach is based on a low rank tensor decomposition of the triplet cube. Inferring possible triples is a harder and more general problem than predicting possible properties for a given resource. The tensor decomposition based approach is limited to RDF graphs with hundreds of types of properties (networks in this paper include tens of thousands of types of properties).

Similar work includes predicting or recommending links in social network, where the problem is to predict the interaction between the members [3, 22].

Chapter 3

Structural Properties

In this chapter we present various structural properties or characteristics that can be computed in a given graph or network. Some of these characteristics can be efficiently computed (with a single pass through the graph), while for others, the best known algorithms run in higher polynomial or even exponential time. First we define basic graph and network concepts that are used through out this work and afterward we present the structural properties in the order of their complexities.

3.1 Basic Definitions

We say a **graph** is an ordered pair $G = (V, L)$, where V is a set of **nodes** (or **vertices**) and L is a set of **links**. Sometimes, we denote the set of nodes and the set of links of a graph $G = (V, L)$ also as $V(G)$ and $L(G)$, respectively. A link can be either undirected – an **edge**, or directed – an **arc**. The set of all edges is denoted by E, and the set of all arcs is denoted by A, therefore $L = E \cup A$. If $E = \emptyset$, the graph is **directed**, and if $A = \emptyset$, the graph is **undirected**. Every link has two **end-nodes**. In an arc one is the **initial** node or **source** and the other is the **terminal** node or **target**. We can move only from initial to terminal node (the direction of the arc). With $e(u, v)$ we express the fact that using link e, we can move from node u to node v.

A link with one node as both the source and the target is called a **loop**. We only consider graphs with finite number of nodes and links, denoting the cardinalities $n = |V|, m = |L|$.

A **Network** $N = (V, L, \mathcal{P}, \mathcal{W})$ consists of:

- A graph $G = (V, L)$, where V is the set of nodes and L is the set of links,
- $\mathcal{P}$ is the set of **node value functions**, $p : V \to A$,
- $\mathcal{W}$ is the set of **link value functions**, $w : L \to B$.

Informally, we can say that **Network = Graph + Data**, a combination of topological structure and data about the nodes and links.

The **in-neighbors** and **out-neighbors** of a node v are defined as:

$$\mathit{inneigh}(v) = \{u : \exists e \in L : e(u, v)\}, \quad \mathit{outneigh}(v) = \{u : \exists e \in L : e(v, u)\}.$$

The **neighbors** of a node v are defined as: $\mathit{neigh}(v) = \mathit{inneigh}(v) \cup \mathit{outneigh}(v)$.

The **in-degree** and **out-degree** of a node v are defined as:

$$\mathit{in\text{-}degree}(v) = |\mathit{inneigh}(v)|, \quad \mathit{out\text{-}degree}(v) = |\mathit{outneigh}(v)|.$$

The **degree** of a node v is defined as: $\mathit{degree}(v) = |\mathit{neigh}(v)|$.

Graph S is a **subgraph** of a graph G, if $V(S) \subseteq V(G)$ and $L(S) \subseteq L(G)$. If S is a subgraph of G, then G is said to be a **supergraph** of S. A subgraph S of a graph G is called an **induced subgraph** by a set of nodes $W \subset V(G)$, if $V(S) = W$ and $L(S) = \{e \in L(G) : \exists u, v \in W : e(u, v)\}$.

A **walk** from u to v is a sequence $u = v_0, e_1, v_1, e_2, v_2, \ldots, e_k, v_k = v$, where e_i links nodes v_{i-1} and v_i, $i = 1, \ldots, k$. The sequence is a **semiwalk** (or **chain**) from u to v, if e_i connects (direction is not important) nodes v_{i-1} and

v_i, $i = 1, \ldots, k$. A **path** is a walk where all the nodes in the sequence are different.

Nodes u and v are **weakly connected** if there exists a semiwalk between u and v. Nodes u and v are **strongly connected** if there exists a walk from u to v and a walk from v to u. Both relations are equivalence relations. The subgraphs induced by equivalence classes are called **weakly / strongly connected components**.

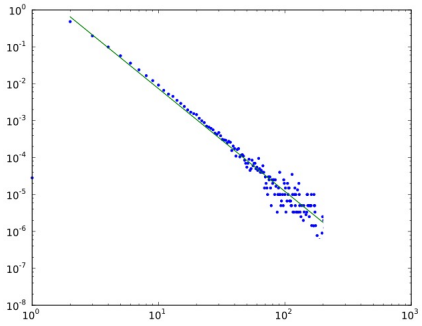
3.2 Network Structure

The simplest properties of a graph are its **number of nodes**, n, and **number of links**, m. These are rough measures of "connectivity" of the graph. For example, if the number of links is less than the number of nodes, then we know that we have more than one weakly connected component. For simple unlabeled directed graphs (graphs with no loops and no parallel arcs), it holds $m \leq n(n-1)$. For simple planar graphs with $n \geq 3$ nodes (graphs that can be drawn on the plane in such a way that its links intersect only at their endpoints), it holds that the number of links is bounded by $3n - 6$.

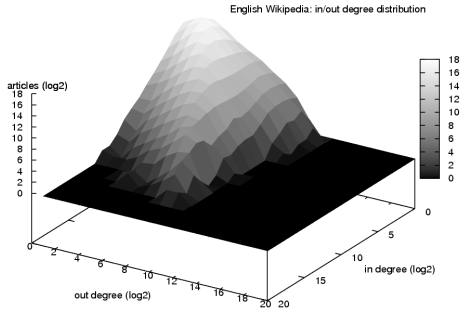
In general, we can define families of graphs based on the the number of nodes and links. Two very popular families are **dense** and **sparse** graphs, where the main idea is that dense graph is a graph where the number of links is close to the number of links in a complete graph, while the average node degree in a sparse graph is much smaller than the number of links in the graph.

A more detailed quantitative relation between the number of nodes and links is **degree distribution** of the graph, which is the probability distribution of the node degrees over the graph. Formally, the degree distribution $P_{deg}(d)$ of a graph is defined to be the fraction of nodes in the graph with degree d:

$$P_{deg}(d) = \frac{n_{deg}(d)}{n}, \tag{3.1}$$



(a) Degree distribution of the Barabási-Albert Model for generating random scale-free networks (log-log scale).



(b) In and out degree distribution of English-Wikipedia hyperlink graph (log-log scale).

where n is the number of nodes and $n_{deg}(d)$ is the number of nodes having the degree d. In the case of directed graph, we can compute the **in-degree** and **out-degree distribution** separately.

A common phenomenon in **real-world** networks is a highly skewed degree distribution, which often follows the **power law** $P_{deg}(d) \sim d^{\lambda}$, where λ is some constant. These types of networks are called **scale-free networks** and have been studied in details in the last 15 years. Figure 3.1a shows how the degree distribution of a random generated scale-free network using Barabási-Albert Model follows the power-law. Figure 3.1b shows the in/out degree distribution for a real-world network of English-Wikipedia hyperlink graph. We can see that a majority of nodes (Wikipedia articles) have only a few links to other nodes and there are only a few nodes with very high degree.

A related property to the degree distribution is the **degree sequence**. This is a non-increasing sequence of a graphs node degrees. Degree sequences have a nice property: isomorphic graphs have the same degree sequence; the converse is not true: there exist non-isomorphic graphs with the same degree sequence.

A natural step further is the **node label** and **link label distributions** for labeled graphs – networks. The frequency distributions of node and link labels are denoted by $n_{\mathcal{P}}(l)$ and $n_{\mathcal{W}}(l)$, where $n_{\mathcal{P}}(l)$ is the number of nodes

with node label $l \in A$, and $n_{\mathcal{W}}(l)$ is the number of links with link label $l \in B$, and the corresponding probability distributions are denoted by $P_{\mathcal{P}}(l)$ and $P_{\mathcal{W}}(l)$.

Another interesting concept is **connectivity** of the graph. The simplest connection between two nodes in a graph is a link. An often used measure on the graph is the **length of the shortest path** between two nodes. In the case two nodes are not connected, we can set the length of the path to some large number, possibly infinity. There are many applications where shortest paths are used intensively. Everyday examples include road routing, where nodes are cities and links are roads between the cities with number label, specifying the distance between the cities. Depending on the type of the shortest path problem, we can use one of the standard algorithms: *Floyd–Warshall algorithm* [18] (all pairs shortest paths), *Bellman–Ford algorithm* [6] (single source problem with possibly negative weights), or *Dijkstra's algorithm* [14]) (single source problem with nonnegative weights).

For large graphs, the algorithms above quickly become impractical. In this case, a good heuristic is the A^* *algorithm* [9] which works well in practice. If we plan to run the shortest paths node-to-node many times or we need real-time performance, then some auxiliary data (linear amount in the size of the graph) can be stored to speed up the queries. The authors in [19, 20] present several different techniques, such as contraction hierarchies, transit node, hub labeling and highway dimension, to achieve state of the art performance. In cases when we are interested only in k-nearest neighbors, we can simply run the *breadth-first search algorithm* and stop at a certain depth.

Connected components partition the graph into disjoint sets of nodes and links. They give us an insight into how well the graph is connected overall. There are two types of connected components: **weakly connected components (WCCs)** and **strongly connected components (SCCs)**. We can compute several interesting distributions related to connected components, such as distribution of:

- number of nodes in *WCCs*,
- number of links in *WCCs*,
- number of nodes in *SCCs*,
- number of arcs in *SCCs*.

Experiments have shown that many real-world graphs, consists of one giant *WCC*, which contains most of the nodes and links from the graph. Therefore a lot of research focuses on investigating only the giant component. When dealing with directed graphs, it is interesting to compute the ratio between the sizes of the largest *SCCs* and *WCCs*, which tells us how well is the largest connected component bidirectionally connected.

WCCs can be easily computed with a single *depth-first search algorithm*, resulting in $O(m)$ time and $O(n)$ space complexity. *SCCs* can be computed with the two well-known algorithms: *Tarjan's algorithm* [32] and *Kosaraju's algorithm* [12]. They both have the same time $O(n+m)$, which is optimal, and $O(n)$ space complexity.

The **diffusion kernel** is based on computing the matrix exponential of the graph adjacency matrix [36]. The entries of the adjacency matrix A are defined as $a_{i,j} = k$, if nodes i and j are linked with k links. The exponential diffusion kernel is defined as

$$K = e^{\alpha A} = \sum_{i=1}^{\infty} \frac{\alpha^i A^i}{i!} \quad , \quad \alpha > 0,$$

where the parameter α controls how local/global the similarities are and is estimated from the data. The kernel sums contributions from all paths between two nodes, discounting paths by their lengths. Therefore, diffusion takes into account both the total number of paths between nodes along with their respective lengths.

Centrality of a node determines the relative importance of a node within the graph. An example of a simple centrality measure of a node can be its degree. The **closeness** and **betweenness** centrality measures rely on the identification and length of the shortest paths among nodes in the network.

The idea of using the shortest distances is that the intermediary nodes increase or delay the time taken for the interaction between the two nodes and can distort the information as it travels between the nodes. Depending on the given graph, directed or undirected, the next two definitions assume that the graph is connected or strongly connected, in order to avoid division with zero.

Closeness centrality measures the length of the shortest paths from a node to all other nodes in the graph and is defined as the inverse of total length. Formally,

$$C_C(v) = \frac{n-1}{\sum_{u \in V} d_{v,u}},$$

where $d_{v,u}$ is the shortest distance between nodes v and u.

Betweenness centrality measures the number of shortest paths that pass through a node [8]. Formally,

$$C_B(v) = \frac{1}{\lambda} \sum_{s \neq v \neq t \in V} \frac{\sigma_{st}(v)}{\sigma_{st}},$$

where σ_{st} is the total number of shortest paths from node s to node t, and $\sigma_{st}(v)$ is the number of those paths that pass through v. The constant λ is a normalization factor and for directed graphs is $(n-1)(n-2)$ and for undirected graphs is $(n-1)(n-2)/2$.

Richer characteristics of the graph can be encoded in the **form of subgraphs**. Small subgraphs give us a deeper insight into the structure of the graph and can enable us to uncover the structural design principles of a more complex graph. Typically, "frequent" subgraphs are called *building blocks* or *network motifs* of larger networks [23].

Computing all different **connected induced size-k subgraphs** (by a set of nodes of size k) and their frequencies enable us to further understand the structure of the graph. For example: if our graph represents a real-world process, then connected induced size-k subgraphs capture the "relation" between the k individuals (nodes) in the process. Enumerating all connected induced size-k subgraphs of a given graph is typically a very time consuming process due to the large number of connected induced size-k subgraphs in a

given graph. The upper bound on the number of connected induced size-k subgraphs is $\binom{n}{k}$, where n is the number of nodes in the graph.

We studied the algorithm *ESU*, proposed by [35], which enumerates all connected induced size-k subgraphs and each exactly once. This algorithm can be trivially changed into a randomized algorithm, which unbiasedly enumerates only a certain fraction of all connected induced size-k subgraphs. For large graphs, enumerating all connected induced size-k subgraphs is prohibitive. In these situations, randomization or sampling is of crucial importance to solve the problem. In case we are interested only in size-3 subgraphs (triads), authors of [5] presented a subquadratic algorithm $O(m)$ for counting triads in a large sparse network with a small maximum degree.

Recently, there has been an interesting paper on Enumerating Subgraph Instances Using Map-Reduce [1], which describes how to find all instances of a given "sample" graph in a larger "data graph," using a single round of map-reduce. This enables us to significantly speed up the process of enumeration with additional computing resources.

Network motifs in a given network are small connected induced subgraphs (subnetworks) that occur with significantly higher frequencies than would be expected in random networks [23]. Subgraph significance is typically determined by generating a set of random graphs under a given random graph model. It is often required that a random graph model generates graphs with the same degree sequence as the original network.

The authors of [23] computed network motifs on ecosystem food webs dataset. Nodes represent groups of species and arcs point from nodes representing predators to nodes representing its preys. A four-node motif was found (bi-parallel motif), Figure 3.1, which indicates that two species that are prey of the same predator both tend to share the same prey.

A **clique** in an undirected graph is a subset of its nodes such that every two nodes in the subset are linked – complete subgraph of a graph. A **maximal clique** is a clique that can not be extended by including an adjacent node. Finding cliques in a given graph is of great importance in many appli-

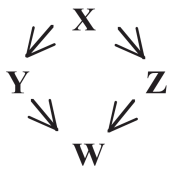


Figure 3.1: Bi-parallel network motif.

cation areas. An often studied example are communities in social networks, in which cliques are an approximation of dense-interacting communities of people. In bioinformatics, clique finding procedures have been used to find frequently occurring patterns in protein structure [16].

Listing all maximal cliques in a given graph is an NP-complete problem and thus can take exponential time, because in the worst-case scenario there may be an exponential number of cliques. Even so, there has been a dedicated study in the algorithms for finding maximal cliques. The best practical algorithms are based on the *Bron–Kerbosch algorithm* [15, 10].

We introduced several structural properties that can be computed in a given graph or network. Some of these structural properties are expensive to compute, especially on large datasets, which is our case. Based on these circumstances, we decided to use a variation of the distribution of node and link data, shortest paths and diffusion kernel. The next few chapters describe in detail the problem itself, approach, datasets, experiments and implementation details.

Chapter 4

Approach

The main premise of this work is that given an object, we can predict its missing properties based on the properties of similar objects — much like predicting preferences through the preferences of social connections.

The framework of our approach is summarized as follows: the input to our method is an object with a set of known properties that are associated with it. We refer to the object as the query object. The output is represented as a sequence of **relevance scored** missing property candidates, where the relevance scores (positive numbers) reflect how relevant a property is to the given object. The relevance scores are encoded as a vector, denoted as the **property relevance vector**.

Our approach consists of three steps: first, we find a set of objects that are the most similar to the query object. This step relies on the definition of how similarity is measured. In the second step each of the selected objects is used to obtain a **property frequency vector**. Finally the property relevance vector is obtained by aggregating the property frequency vectors, where we use a weighted sum. We will first introduce some notation and then present the technical details of the approach.

4.1 Definitions

The LOD datasets are stored as a set of **assertions** of the form (u, v, p), where u and v are nodes and p is a property linking the two nodes. Datasets of such form can be naturally represented as networks.

The input to our method is a network $N = (V, A, t, w)$, where:

- $t : V \to \{object,\ literal\}$,
- $w : A \to P$.

The network contains two types of nodes, **objects** and **literals**. Literals are used to identify values such as numbers and dates, so in our problem, we only recommend properties for objects. This differentiation is specified by input function t. We denote the set of objects by $O = t^{-1}(object) \subseteq V$. Since each node in the network is an identifier of a concrete object or literal, each node is a unique element in V.

We refer to the arc labels as **properties** and denote the indexed set of possible properties by $P = \{p_1, \ldots, p_{|P|}\}$. A pair of nodes can be linked by several arcs. For every arc $a \in A$ in the network it holds that: $w(a(u, v)) = p \leftrightarrow (u, v, p)$, where (u, v, p) is an assertion from the dataset. The elements of the triple (u, v, p) are called the **source**, **target** and **label** respectively. Finally, two nodes are **neighbors** if they are linked — here the link direction is unimportant.

To each object $o \in O$ we assign two vectors: **in-property frequency vector** $f_{in}(o) \in \mathbb{N}^{|P|}$ and **out-property frequency vector** $f_{out}(o) \in \mathbb{N}^{|P|}$. The i-th component of the in-property frequency vector is defined as:

$$\begin{aligned} f_{in}(o)[i] &= |\{u \in V : (u, o, p_i)\}| \\ &= |\{u \in inneigh(o) : \exists a(u, o) \in A : w(a) = p_i\}| \end{aligned}$$

The out-property frequency vector is defined analogously. There are two possible ranking tasks: ranking missing in-properties and ranking missing out-properties. To simplify the presentation, we will focus on predicting out-properties.

4.2 Computing the property relevance vector

Let us assume we are given a **dissimilarity** between objects, $d : V \times V \to \mathbb{R}_0^+$, with the following properties:

- $d(o, o) = 0$,
- $d(o_1, o_2) \geq 0$,
- $d(o_1, o_2) = d(o_2, o_1)$,

where $o, o_1, o_2 \in O$. The property relevance vector for recommending missing out-properties for the given query object o is defined as:

$$g(o) := \sum_{o_i \in S_k} e^{\frac{-d(o,o_i)^2}{\sigma^2}} f_{out}(o_i), \tag{4.1}$$

where $S_k \subset O$ denotes the set of the closest k objects to o and σ represents a Gaussian bandwidth parameter which is estimated from the data.

Determining the set S_k corresponds to the first step in our approach, $-d(o, o_i)^2$ represents the choice of the dissimilarity, $f_{out}(o_i)$ represents the local property relevance vector and $e^{\frac{-d(o,o_i)^2}{\sigma^2}}$ corresponds to the weight assigned to the i-th object in S_k. For the task of recommending missing in-properties we use $f_{in}(o_i)$ as the local property relevance vectors.

It is often convenient to define d in terms of a normalized Mercer's kernel function [1] [29], $k : V \times V \to \mathbb{R}$. Since a Mercer kernel induces a metric, we can define our dissimilarity as:

$$d_k(o_1, o_2) = \sqrt{k(o_1, o_1)^2 - 2k(o_1, o_2) + k(o_2, o_2)^2}.$$

[1] All dissimilarities used in this thesis are kernel based.

Since the kernel k is normalized ($k(x,x) = 1, \forall x \in V$), it follows that $d_k(o_1,o_2)^2 = 2 - 2k(o_1,o_2)$. Finally, replacing $2 - 2k(o_1,o_2)$ with $-k(o_1,o_2)$ produces scores with equivalent rankings.

The property relevance vector clearly heavily depends on the choice of dissimilarity d. In the following section, we will introduce several structural descriptions of objects which induce dissimilarities between objects. We use sometimes the term **similarity** instead of dissimilarity, which can be simply thought of as the opposite of dissimilarity.

4.3 Structural Descriptions of Objects

In this section, we describe the structural descriptions of objects which can be divided into the two following categories. **Local descriptions** encode the node's local network structure by storing various feature vectors based on property distributions, while **global descriptions** exploit the global graph properties such as graph distances between objects.

4.3.1 Local Descriptions

In our local approaches we will define dissimilarities in terms of normalized kernels, explicitly constructed by using feature mappings. A **feature map** is a function that maps objects to N-dimensional **feature vectors**.

$$\phi : O \to \mathbb{R}^N.$$

Using the kernel based formulation of dissimilarity, a feature map ϕ is used to define a normalized kernel:

$$k(o_1,o_2) = \frac{\langle \phi(o_1), \phi(o_2) \rangle}{\sqrt{\| \langle \phi(o_1), \phi(o_1) \rangle \langle \phi(o_2), \phi(o_2) \rangle \|}},$$

where $\langle \cdot, \cdot \rangle$ denotes the standard inner product.

We present three different feature vectors based on local graph information. We focus on a local descriptions that are based on property distributions

and can be computed efficiently. The main idea is that objects with similar properties (in the neighborhood) represent similar things. We implemented the following feature maps:

PropertyCount (Count): the property count feature map assigns to each object o a vector of counts of properties associated with the object o (for each possible property, we also store the number of times it appears with respect to the object o). For a given o, we define its feature map as the sum of its property frequency vectors,

$$\phi_{pc}(o) := f_{in}(o) + f_{out}(o).$$

Note that this feature map is insensitive to the directionality of the property.

DirPropertyCount (DirCount): the directed property count feature map is a direction sensitive variant of *PropertyCount* feature map. The feature map is defined as

$$\phi_{dpc}(o) := [f_{in}(o), f_{out}(o)].$$

NbhPropertyCount (NbhCount): neighborhood property count feature map is defined as

$$\phi_{npc}(o) = \phi_{pc}(o) + \frac{1}{|neigh(o) \cap O|} \sum_{o' \in neigh(o) \cap O} \phi_{pc}(o').$$

The first two maps count only properties of the given object, while this map counts the properties of the neighbors of the given object as well. It represents a locally smoothed version of *PropertyCount* and is suited for objects with a low number of distinct properties.

4.3.2 Global Descriptions

Local descriptions assume a richness of the local structure of the network making them well-suited for objects with high property support (*DirProp-*

ertyCount) or objects with well-connected neighbors (*NhbPropertyCount*). One alternative is to exploit the topological structure of the network – graph component of the network.

To capture the global structure accurately, we remove literals from the graph. We do not want to objects to be connected through literals (e.g. value "100"), but rather if they share common objects. Consider the following example: *(Ljubljana, 1000, postalCode)* and *(kilometer, 1000, equalsMeters)*. If we do not remove literals, unrelated objects, such as a city and unit of distance can be similar. To ensure a well-connected graph, we also remove directionality from the graph. We describe two global dissimilarities based on graph and diffusion distances on the graph.

ShortestPaths: the length of the shortest path (or distance) between two objects in the graph is a simple indicator of dissimilarity between two objects (e.g. the distance between similar instances in a taxonomy is often low). A problem with this method is that for a query object, the objects at a distance 1 typically represent objects of different types. However, such objects will inevitably be assigned the highest weights in the scoring function, possibly yielding poor results. Therefore, we decided to compute two dissimilarities for a given object:

- **ShortestPaths1 (SP1)**: the length of the shortest path between a given object and any other object.
- **ShortestPaths2 (SP2)**: the length of the shortest path of length at least 2 between a given object and any other object.

DiffusionKernel (DK): is based on the diffusion distance in graph. We defined the diffusion kernel in Section 3.2. In our experiments, we use an unnormalized kernel to define the dissimilarity, because the normalization is computationally prohibitively expensive.

4.4 Algorithm

This section presents the entire approach described so far in a short algorithm written in a form of pseudo code. The algorithm consists of five phases:

1. compute the set of objects using the network input function t,
2. compute property frequency vectors (f_{in} or f_{out}) for every object in O,
3. compute the dissimilarities between object o and all the objects in O,
4. compute the property relevance vector for object o,
5. omit the existing properties of object o.

Note that the first, second and third phase – object set determination, property frequency vectors and dissimilarity computation – can be computed in the preprocessing step. This is important in the process of learning the parameter σ, because it enables us to significantly speed up the learning procedure.

Algorithm 1 Missing Properties

Input: Network $N = (V, L, t, w)$,
Dissimilarity d,
Query object $o \in t^{-1}(object)$,
Sigma parameter σ.

Output: Missing properties of object o.

{Phase 1: compute the set of objects O using the input function t.}
$O = t^{-1}(object)$

{Phase 2: compute property frequency function for every object.}
vector P
for all $u \in O$ **do**
  $P[u] = f(u)$
end for

{Phase 3: compute dissimilarities between object o and other objects.}
vector D
for all $u \in O$ **do**
  $D[u] = d(u, o)$
end for

{Phase 4: compute property relevance vector for object o.}
vector R
for all $u \in O$ **do**
  $R \mathrel{+}= e^{-D(u)^2/\sigma^2} P[u]$
end for

{Phase 5: remove existing properties of object o from R.}
vector $M = R \setminus P[o]$

return M

Chapter 5

Datasets

In our experiments, we consider three datasets from the LOD cloud diagram: two variants of DBpedia and Freebase. These were selected based on scale and applicability to the missing property problem.

5.1 DBpedia

The DBpedia project [2] extracts information from Wikipedia and combines this information into a large, cross-domain knowledge base. The data is provided in the Resource Description Framework (RDF) model, which is easily transformed into a graph/network. We consider two DBpedia datasets, **DBraw** and **DBmapped**. Both datasets are based on Wikipedia *infoboxes* — the tables at the top right-hand corner of Wikipedia articles. Each infobox contains a set of properties and the corresponding property values.

DBraw (raw properties): contains all the properties from all the infoboxes and templates within the English Wikipedia articles. The properties are not cleaned or merged and there is no consistent ontology for the dataset. (e.g. an infobox describing a person named Bob uses the *dateOfBirth* property, while some other infobox describing a person named Alice uses the *birithOfDate* property).

Feature	DBmapped	DBraw	Freebase
#nodes	5.8M	11.1M	**141M**
#links	17.2M	47.2M	**607M**
#objects	2.2M	3M	**23M**
#properties	1296	**44463**	19700
avgDegree	5.92	8.45	8.58
avgInDegree	3.77	5.01	4.29
avgOutDegree	9.42	18.97	11.53

Table 5.1: General characteristics for the three datasets we used (note that M stands for million).

DBmapped (mapped properties): each property is mapped onto a DBpedia ontology (e.g. semantically equal properties, such as in the example above, are mapped to a single ontology property – *DateOfBirth*). The data is therefore much cleaner and better structured than the raw properties dataset. The ontology is not complete and properties not in the ontology are omitted. Consequently, DBmapped is much smaller than DBraw.

5.2 Freebase

Freebase is a large collaborative structured knowledge base of general human knowledge, which is composed mainly by its community members. It is the underlying database for Google Knowledge Graph. It is the largest dataset we considered containing hundreds of millions of assertions in RDF format.

5.3 Dataset characteristics

In order to get a better understanding of the three datasets, Table 5.1 presents an overview of dataset characteristics. We highlight the following observations:

- We are dealing with relatively large networks – the Freebase network contains more than half a billion links.
- DBraw has the largest number of properties, although Freebase has much more links. This is a sign that DBraw is a messier dataset than either Freebase or DBmapped.
- The average degree grows with the size of the dataset.
- In all three datasets, the avgOutDegree is two or three times larger than avgInDegree. This is due to the nature of the RDF description – an object is described by numerous literals, which do not have any out-properties.

Chapter 6

Experiments

In our experiments, we only show results for out-properties. We performed experiments on in-properties as well but the out-properties where the more interesting case, since objects generally have more out than in-properties.

6.1 Evaluation protocol

The basic unit of evaluation is a tuple $(o, \{p_1, p_2, ..., p_{k_o}\})$, containing an object o in the network and its properties $\{p_1, p_2, ..., p_{k_o}\}$ which are deleted to object o (k_o is specific to the object o and can be defined as some percentage of all the object o's properties). Given a query object o, the algorithm outputs a ranked list of possible missing properties. We compute evaluation metrics on this list to measure the quality of the outputted ranking.

Given a ranking of the possible missing properties, we compute two evaluation metrics to measure the quality of the outputted ranking:

- **inverse rank (IRank)** = 1 / rank of deleted property
- **top k** $= \begin{cases} 1, & \text{rank of deleted property} \leq k \\ 0, & \text{otherwise} \end{cases}$

Note that IRank metric is a stable metric in a sense that if only a few among many of deleted properties are ranked very low, then this does not affect the

average IRank much (e.g. if the deleted property placeOfBirth is ranked at the 1000th place, this contributes to the IRank average only 1/1000. This is a commonly used measure for evaluating a ranking system.

When deleting more than one property ($k > 1$), we compute the rank of deleted property $p_i, 1 \leq i \leq k$ by disregarding the other deleted properties $p_j, 1 \leq i \leq k, j \neq i$ from the outputted ranking (i.e. if for 5 deleted properties, they are returned as the first 5 results, they all have an effective rank of 1).

This evaluation methodology allows us to compare the outputted ranking with the "ground truth" (the datasets only provide which properties the object o has; there is no explicit information about the missing properties for the object o).

6.1.1 Sampling

It is computationally prohibitive to perform the evaluation on all possible $(o, \{p_1, p_2, ..., p_{k_o}\})$ tuples due to the size of the datasets. Therefore, we need to construct a representative sample of $(o, \{p_1, p_2, ..., p_{k_o}\})$ tuples, on which we evaluate the algorithms. Because the distributions of the dataset are skewed (power-law), we propose two sampling methods:

Power Sample. In this method, we first sample the objects and then sample its properties. Using a uniform distribution over the objects we select a random subset $S_O \subset O$ of a specified size. For each object $o \in S_O$, we randomly select a given percentage p of its properties according to a uniform distribution. The parameter p defines how many of the properties will be deleted to the objects in S_O.

While this is a natural approach, it skews towards more frequently occurring properties. These "power properties" are easy to recommend, as we illustrate with one of the baseline methods (PropertySupport – Section 6.2).

Uniform Sample. In this case, we first sample the properties and then

the objects. Using a uniform distribution, we select a random subset of properties $S_P \subset P$ of a given size. For each $p \in S_P$, we uniformly select N objects which have property p. This ensures us that properties have been chosen independently of their support and that we have exactly N objects per every property in S_P. We use this sampling method because it covers a wide range of properties. We chose $N = 10$ so that there were a sufficient number of distinct properties with this support while ensuring a selected property occurred often enough to make results statistically significant.

6.2 Baselines

We first present four methods for computing missing properties, which we call baselines. The purpose of these methods is the following:

- Measure the performance of simple approaches for the missing property problem.
- Simple methods and their failure modes reveal additional information about the structure of the given datasets.
- Comparison of the baselines performance against our method.

PropertySupport (Sup). Our initial baseline is to return a fixed ranking of the properties. This ranking is created by sorting the properties in a non-increasing order by their property support, i.e. the number of objects that have a particular property. This method should perform well on properties that occur frequently in the network.

RandomObjects (Rnd). The next method uses our general approach of ranking properties based on summing the property frequency vectors across a set of objects. However, the set of "similar" objects is chosen at random rather than via similarity. This baseline gives an insight into the structure of the dataset: the method should perform well on highly homogeneous datasets where objects are associated with highly similar sets of properties.

PropertyIntersection (Inter). The next baseline is a simplified version of our approach based on local descriptions. It computes similarity by considering the number of properties that are shared with the query object. This number is taken as the weight for the object and the rank is computed as the weighted sum of objects' property counts. Note that the method is computationally as complex as using the other local descriptions. The performance of this method illustrates the utility of kernel smoothing and more complex object descriptions.

PropertyCooccurrence (Cooc). For our final baseline we compare to the algorithm described in [26]. The approach is based on association rule mining and the idea is to approximate resource similarities through the co-occurrence of properties. For example, if object o has property *presidentOf*, then it is likely that it also has properties like *precededBy*, *succeededBy*, and *politicalParty*.

Let o represent an object and $L(o) = \{p_1, \ldots, p_m\}$ represent the set of out-going properties associated with o. Let $C(p)$ denote the set of all out-going properties that co-occur with p at least once. The authors define a scoring function, $confidence(p, o)$ for each possible property $p \notin L(o)$ as:

$$\begin{aligned} confidence(p, o) &= \prod_{p_i \in C(p) \cap L(o)} confidence(p_i \Rightarrow p), \\ confidence(p_i \Rightarrow p) &= \frac{coocc(p_i, p)}{occ(p_i)}, \\ coocc(p_i, p) &= |\{o \in O | p_i \in L(o) \wedge p \in L(o)\}|, \\ occ(p_i) &= |\{o \in O | p_i \in L(o)\}|. \end{aligned} \tag{6.1}$$

6.3 Comparison of Approaches

We first give an overview of the results of our experiments. Table 6.1 shows the complete results for described methods on all three datasets. The results

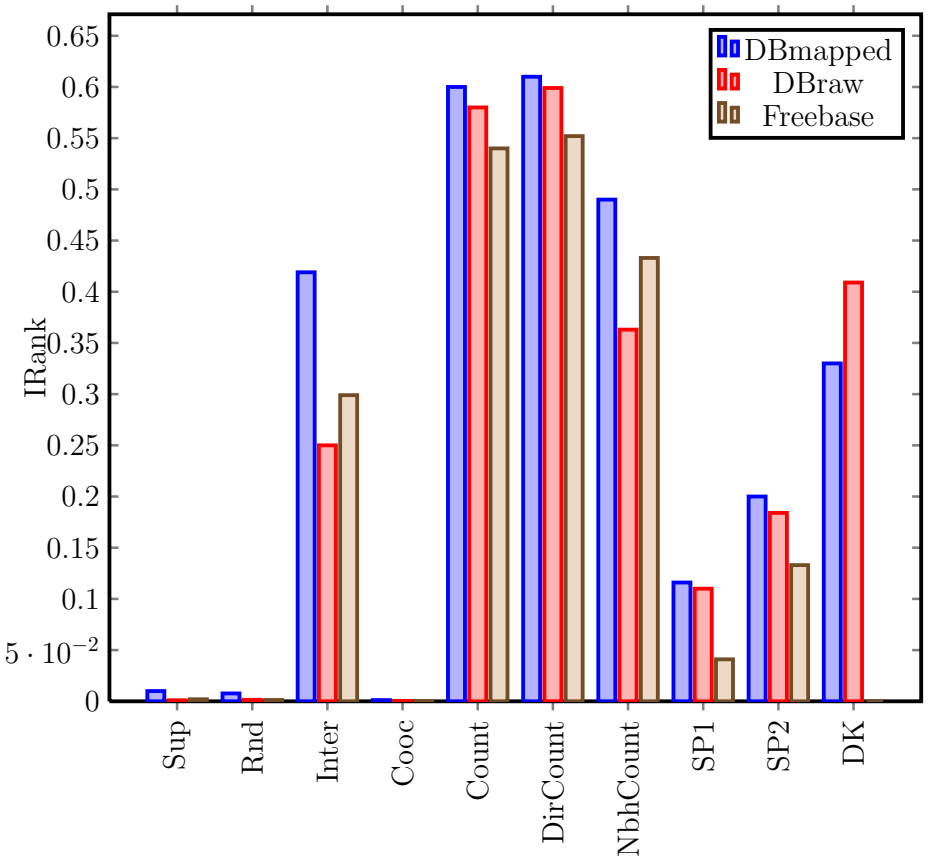
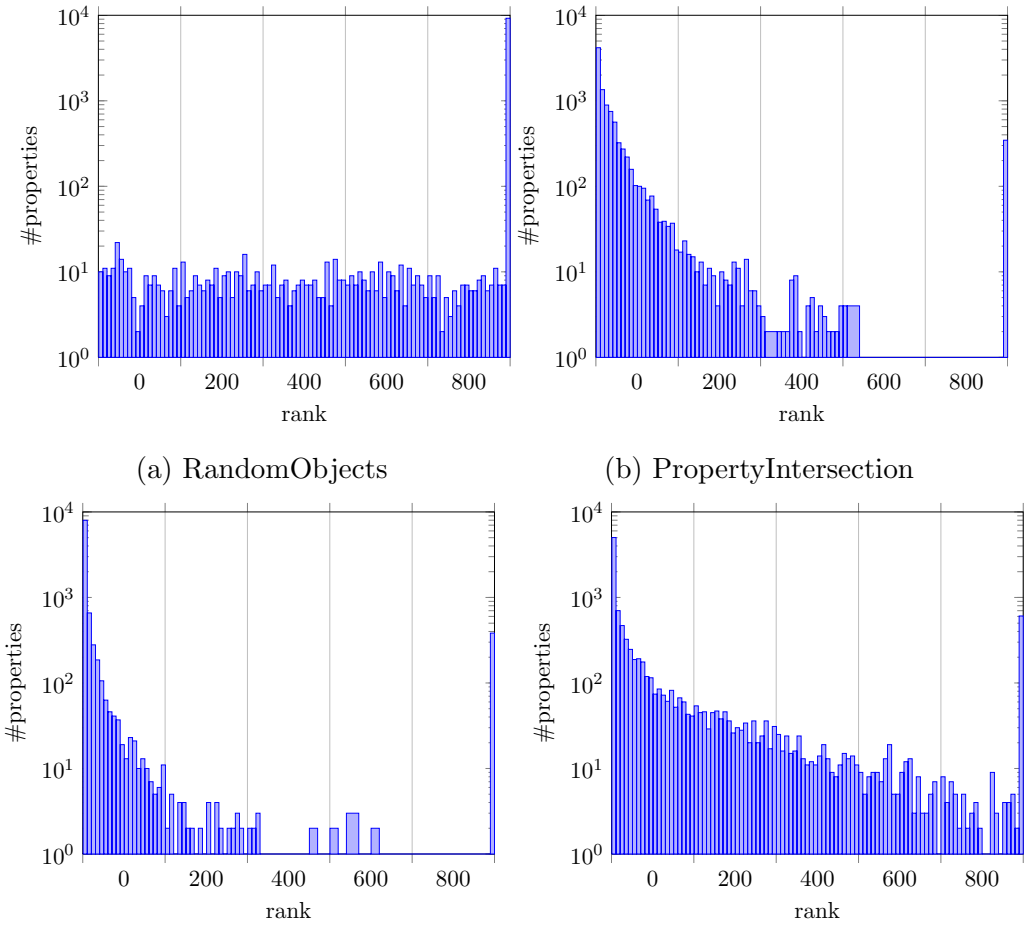


Figure 6.1: The averaged IRank for the three datasets over all the methods. Note the extremely low values for *Sup, Rnd* and *Cooc*.

are grouped according to the methods in three categories: baselines (first four methods), local descriptions (fifth, sixth and seventh method) and global descriptions (last three methods). We omit the results for *DiffusionKernel* method on Freebase dataset, because of the expensive computation of the kernel. In the Table 6.1, we show the IRank averaged over all runs and the ratio of experiments where the deleted property was ranked within the top five (Top5) and top fifty (Top50). To get a better sense of the meaning of these numbers, we show the rank distributions for four of the methods in Figure 6.2. Note that we show only the first thousand rankings with all other rankings compressed into the final column. Note that other than *RandomObjects*, the distributions all have a heavy tail which makes the standard



(a) RandomObjects

(b) PropertyIntersection

(c) DirPropertyCount

(d) DiffusionKernel

Figure 6.2: The rank distributions cut off at 1000 properties for four methods over baselines, local and global descriptions.

deviations meaningless. Rather than simply show the average IRank, we show Top5 and Top50 to give an indication of the spread of these distributions. For local descriptions, in nearly all cases (over 80%) we see the deleted property was among the top 50 entries. For global descriptions, the figure was lower but for DiffusionKernel, the deleted property was among the top 50 in 60% of the cases.

Overall, the local descriptions performed best followed by the global descriptions. Unsurprisingly the baselines performed the worst although not in the order expected. In addition to average performance, we also computed the correlations between the performance of different methods. This is shown graphically in Figure 6.3 with lighter color indicating a higher correlation. Within a category, the performance of different methods was quite highly correlated, but interestingly, the performance using local and global descriptions did not correlate, implying that they perform well in different cases. This is especially interesting in the case of *Neighborhood Property Count* and the global descriptions. Both methods attempt to compensate for objects with few properties. However, the *Neighborhood Property Count* correlated highly with the other local descriptions, while global descriptions did not. This suggests that global descriptions are a better way to deal with objects with few properties. In the following sections, we compare the methods in greater detail within the categories.

6.3.1 Baselines

Examining the performance of the baselines individually, we found that the *PropertySupport* method performed badly on all three datasets since we used a uniform sample. That is the sample consisted of many non-frequent properties. The method performed better on the power sample, which can be seen in Section 6.4. The *RandomObjects* method exhibited poor performance. This is an indication that the problem is nontrivial. This holds for all three of datasets we considered. The IRank was always very low, which is not surprising since the datasets are closely related to Wikipedia and so are highly

heterogeneous.

The baseline *PropertyIntersection*, although simple, performed quite well. This is especially true for the smaller and cleaner datasets (DPMapped). The performance of the method degraded on the larger and noisier datasets (Freebase and DBraw) although not critically. Recommendation on Freebase resulted in roughly 40% of the deleted properties being ranked in the top 5.

Finally, the co-occurrence approach (*Cooccurrence*) performed badly on all three datasets. This is surprising, since in [26] it is reported to perform well on a variety of datasets. A possible explanation lies in the difference between the scale of the datasets considered. We also not that the decomposition of the $confidence(p, r)$ makes an assumption of conditional independence: $P(p_1, \ldots, p_m|p) = \prod_{i=1}^{m} P(p_i|p)$. Applying the Bayes' rule and ignoring the factors which do not involve p (since they do not affect the ranking), we get

$$P(p_1, \ldots, p_m|p) \propto P(p)^{m-1} \prod_{i=1}^{m} P(p|p_i).$$

Omitting the factor $P(p)^{m-1}$ drastically changes the ranking and could lead to bad performance on graphs with power-law-like property distributions. The second issue with the co-occurrence approach is that no smoothing of probability estimates is used. Consequently, one must ignore the properties p_i which do not co-occur with p, using the set $C(p)$.

6.3.2 Local Descriptions

In Table 6.1, we see that *PropertyCount* and *DirPropertyCount* methods have very similar performance (average IRank ≈ 0.6 on DBmapped) with highly correlated results. This indicates that storing the direction of the properties does not significantly improve the performance. Both methods performed very well, with more than 70% of deleted properties ranked among the top 5 properties on the DBmapped and DBraw datasets. The performance on Freebase was slightly lower but still above 60%.

NbhPropertyCount uses the property distribution of the query nodes graph neighbors in addition to its own. The results of this method are lower than in the case of either *PropertyCount* and *DirPropertyCount*. This is expected because neighborhood property distributions of objects are smoothed version of property distribution of objects, therefore objects become more similar to each other. The method works better in structured datasets (*DBmapped* and *Freebase*), than in a messier datasets (*DBraw*).

We investigated further what occurs if we continue to smooth the neighborhood property distribution. Let us define:

$$\phi_{n2pc}(o) = \phi_{npc}(o) + \frac{1}{|neigh(o) \cap O|} \sum_{v \in neigh(o) \cap O} \phi_{npc}(v),$$

$$\phi_{n3pc}(o) = \phi_{n2pc}(o) + \frac{1}{|neigh(o) \cap O|} \sum_{v \in neigh(o) \cap O} \phi_{n2pc}(v).$$

We refer to the first description as *Nbh2PropertyCount (Nbh2-Count)*, which is the sum of the *NbhPropertyCount* vectors of neighboring objects of a given object. Similar, the second description is *Nbh3PropertyCount (Nbh3Count)*, which is the sum of the *Nbh2PropertyCount* vectors of neighboring objects of a given object. Table 6.2 shows the results of applied smoothing in several steps. We can see that the IRank and Top5 performance degraded with increased smoothing of the property distribution. Going beyond immediate neighbors in the smoothing case proved infeasible due to an explosion in the number of neighbors over which we had to average.

6.3.3 Global Descriptions

In Table 6.1 we can see that *ShortestPaths2* outperformed *ShortestPaths1* method on all three datasets, which means that skipping immediate neighbors (often objects of different type) improves the performance of the system.

The table shows that *DiffusionKernel* method performed better than either of the methods based on shortest paths. This is because *DiffusionKernel* takes into account both the length of the path as well as the number of paths

of a given length between two objects. It is therefore a more robust description than simple shortest path methods. The distribution of ranks for the *DiffusionKernel* are presented in Figure 6.2d. This shows a much heavier tail for *DiffusionKernel* distributions than the distributions for local descriptions, indicating a larger degree of smoothing occurring.

Performance across the data sets was consistent with the behavior we saw with the local descriptions: performance on DBmapped and DBraw was comparable although marginally better on DBmapped. Interestingly, for the *DiffusionKernel* method, the situation was reversed. The method performed better on DBraw than on DBmapped, suggesting that averaging over the noisy links can improve performance. Performance on the Freebase dataset was generally lower, although computing the diffusion kernel unfortunately proved too computationally difficult, so we cannot compare results..

Finally, as a whole global descriptions were not as informative as local descriptions. It is meant as a form of smoothing so its most meaningful comparison is with *NhbPropertyCount*. Comparing the correlation of *NhbPropertyCount* and the global descriptions with the best local description, *DirPropertyCount*, we see that *NhbPropertyCount* is relatively highly correlated indicating that it deteriorates performance while the low correlation with global descriptions capture different information.

6.4 Deleting several properties

A natural question is what happens if we need to predict more than one property (as may be the case in general). We perform this evaluation by removing a fixed fraction of properties (both in and out) for each object in the sample, but the ranking is only tested for out-properties. We refer to this as query object degradation. Here, we use the power sample method since it produces tuples of properties associated with one object. We focused on *PropertyCount* method since it was among top methods in previous experiments.

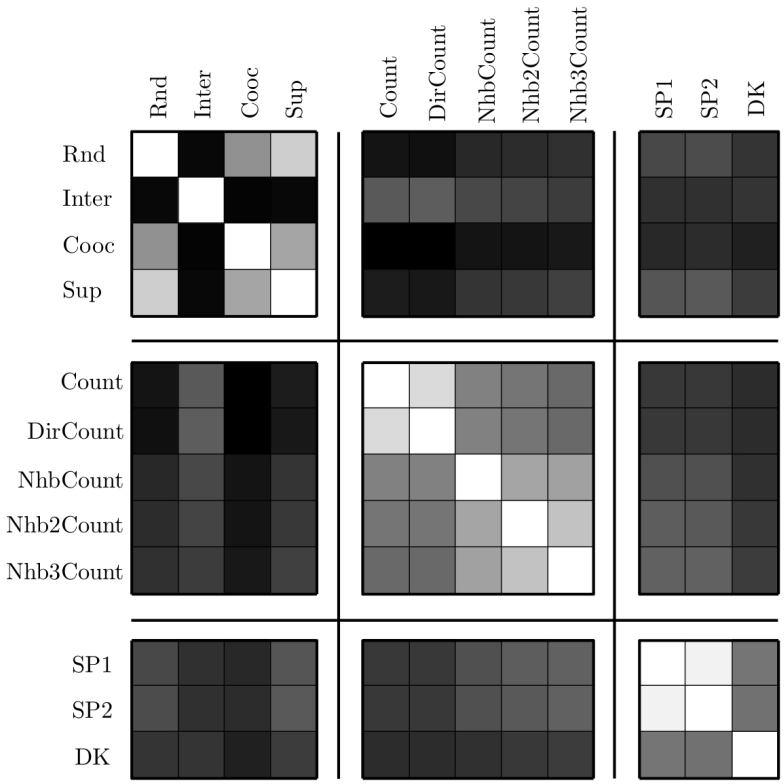
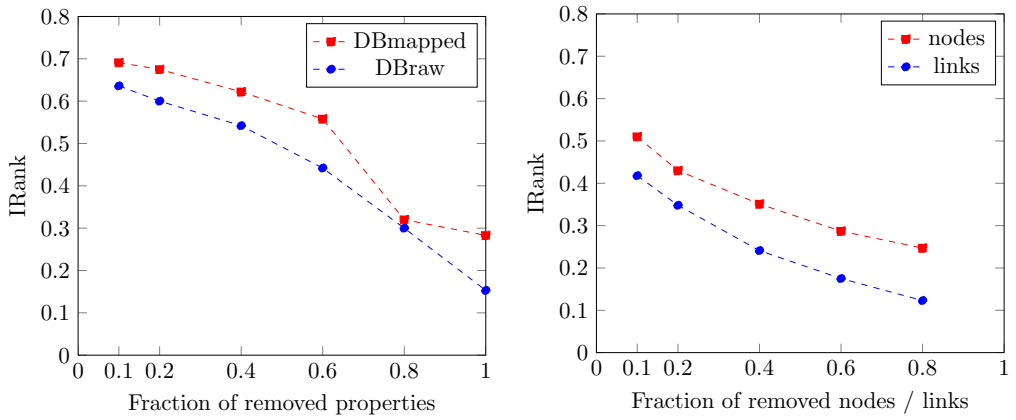


Figure 6.3: Graphical representation of the correlation matrix between the performance of the methods on the DBmapped dataset. A lighter color indicates the higher correlation.

Several different fractions were considered, as we illustrate in Figure 6.4a. Note that a fraction of 1.0 corresponds to all but one in-property being deleted (we can not compute similarities on objects with no properties). We observe a similar negative effect on IRank when deleting properties on both DBmapped and DBraw, where the effect is more pronounced on the noisier dataset DBraw. The degradation occurs gradually soon becoming almost linear for DBraw. For DBmapped the rate of degradation is slower but increases at higher fractions. The interesting phenomenon is that even with only one in-property, the average IRank is between 0.15 and 0.28. One factor is that many properties that we are trying to predict are ubiquitous in the dataset (such as "name") and are easy to predict because many (similar) object have



(a) Deleting properties.

(b) Deleting nodes and links.

Figure 6.4: Effect of degrading the datasets using the *PropertyCount* method: (a) degrading the query objects; (b) degrading the network.

this property. The other factor is that after deletion of properties, objects with only one in-property often have a very specific property (such as "associatedMusicalArtist") that belongs to a specific type of objects. And, when computing the similarities to other objects, the objects of the same type are the most similar and thus provide a relative good property recommendation. However, this also indicates that on average even a single property contains relevant information about the object. This is useful for the early stages of data entry, where an object has only a few links.

Table 6.3 shows the IRank of *PropertySupport* baseline method for different fractions of delete properties to objects.

6.5 Degradation of datasets

The results of predicting properties show that the three datasets (DBmapped, DBraw and Freebase) are highly structured. An interesting question is: how does changing the structure of the network affect the recommendation quality. To test this, we degrade the datasets by removing nodes, links and the most popular properties. We perform this experiment on DBraw dataset

using the *PropertyCount* method and a uniform sample.

Deleting nodes / links. First, we uniformly at random delete some fraction of nodes or links from the graph, and then compute the recommendation quality. The effects of deleting nodes and links are presented in Figure 6.4b. We see a clear correspondence of performance degradation with information removal. This effect is especially pronounced when deleting links.

Deleting popular properties. With the next experiment, we try to show that deleting the most popular properties from the graph does not significantly affect the results of recommending missing properties. For example, deleting a property "rdf:type" from the dataset should not affect the performance of our method if it is robust. The effect of deleting K most frequent properties from the network is reported in the Table 6.4. We can see that in fact the performance improves, because we remove properties that contribute to the local descriptions only because they are frequent.

Method	Dataset	IRank	Top5	Top50
Property Support	DBmapped	0.01	0.006	0.05
	DBraw	0.001	0.001	0.003
	Freebase	0.00201	0.002	0.0182
Random Objects	DBmapped	0.00769	0.0068	0.052
	DBraw	0.00148	0.0006	0.0021
	Freebase	0.0013	0.0002	0.0148
Property Intersection	DBmapped	0.419	0.565	0.86
	DBraw	0.25	0.315	0.773
	Freebase	0.299	0.411	0.787
Property Cooccurrence	DBmapped	0.00117	0.0001	0.0001
	DBraw	0.000023	0.0	0.0
	Freebase	0.000069	0.0	0.0
Property Count	DBmapped	0.60	0.750	0.927
	DBraw	0.58	0.71	0.914
	Freebase	0.540	0.629	0.818
DirProperty Count	DBmapped	0.605	0.760	0.933
	DBraw	0.599	0.72	0.921
	Freebase	0.552	0.640	0.829
NbhProperty Count	DBmapped	0.490	0.624	0.900
	DBraw	0.363	0.443	0.765
	Freebase	0.433	0.548	0.809
Shortest Paths1	DBmapped	0.116	0.140	0.520
	DBraw	0.110	0.135	0.411
	Freebase	0.041	0.056	0.338
Shortest Paths2	DBmapped	0.200	0.261	0.524
	DBraw	0.184	0.232	0.520
	Freebase	0.133	0.185	0.479
Diffusion Kernel	DBmapped	0.330	0.370	0.602
	DBraw	0.409	0.445	0.676
	Freebase	–	–	–

Table 6.1: Numerical results for the different methods over our three datasets. We show the averaged IRank along with the Top5 and Top50 statistics.

Method	IRank	Top5	Top50
PropertyCount	0.60	0.750	0.927
NbhPropertyCount	0.490	0.624	0.900
Nbh2PropertyCount	0.477	0.604	0.902
Nbh3PropertyCount	0.435	0.550	0.868

Table 6.2: Numerical results for smoothing on the DBmapped dataset.

Fraction	0.1	0.2	0.4	0.6	0.8	1.0
DBmapped	0.289	0.293	0.292	0.293	0.293	0.293
DBraw	0.122	0.119	0.121	0.120	0.118	0.116

Table 6.3: The effect on averaged IRank for the *PropertySupport* method when we delete different fractions of objects' properties (query object degradation).

K	1	5	10	20	50	100
DBraw	0.602	0.614	0.621	0.633	0.652	0.651

Table 6.4: The effect on averaged IRank when we delete the K most frequent properties from the graph using the *PropertyCount* method.

Chapter 7

Implementation

One of the key contributions of this work is an implementation which makes the analysis given in the previous section feasible at the scale of the datasets we consider. The data, code and a interactive demo is available at `http://lodminer.net` [30]. In this section, we highlight some of the key implementational details.

K-nearest objects. Computing similarities between a given object and all the other objects results in a similarity vector, which contains similarities of all the objects to the given object. The size of similarity vector is linear in the number of objects in the dataset. Since our datasets contains millions or even tens of millions of objects, we can not afford to store the entire similarity vector for every object we evaluate. We decided to take at most K-nearest objects, where K is a parameter. Since the value of parameter K can significantly affect the final results, we conducted the experiment in which we varied the value of parameter K and measured the IRank of the method. We decided to set $K = 1000$, which was experimentation showed sufficient and kept memory usage to manageable levels.

Parameters. In Section 4.2, we introduced the bandwidth parameter σ, which regulates the kernel smoothing. We automatically estimated the pa-

rameter from data, using standard cross-validation.

Shortest Paths. For the global descriptions, computing all the shortest path distances of a query object is prohibitively expensive, especially in larger datasets. Furthermore, the discrete nature of graph distance means there are many objects with identical distances. Therefore, we used the following notion of K-nearest objects with the methods *ShorthestPaths1*, *ShorthestPaths2*:

- We took all the objects with shortest paths of length 1 from a given object.
- We took 1000 objects with progressively larger length of shortest paths.

This allows us to efficiently compute (time and space wise) the shortest paths. It also guarantees, that we have at least 1000 objects which have the length of the shortest path from a given object at least 2, which is what we need for the *ShortestPaths2* global description.

Space partitioning. Using only K nearest neighbors significantly reduced space usage. However, for a given query object, we still have to compute the similarity to all other objects. Again, for large datasets this can be prohibitively expensive. To speed up the algorithm, we first cluster the objects into some number of clusters and then compute the dissimilarities for a given object only to the objects that are in the same cluster as the object. Since cluster size were much smaller than the total number of objects, we obtained a significant speed up in the similarity computation phase. We used k-means clustering on the datasets and compared the results with and without clustering for a range of number of clusters, and found the performance differences to be negligible.

LODminer. As mentioned above, there is an interactive demo of this approach available at `http://lodminer.net`. The demo computes similar objects and missing properties for a query object using *PropertyCount* method

on the DBmapped dataset. For example, the query object **Aston Martin** returns the results:

- top 5 missing properties - *(location, parentCompany, numberOfEmployees, product, revenue)*,
- top 5 similar objects - *(Porsche, Piper Aircraft, TVR, Kia Motors, Lamborghini)*.

LODminer runs online – for a given query object, similar objects and missing properties are computed in real-time. To support this, each dataset has to be preprocessed. The preprocessing step prepares and computes the basic data for the demo. It also includes some of the above mention implementation details. For two of our datasets, DBmapped and DBraw, we present in Table 7.1 time and space information for the preprocessing step.

Dataset	Time	Space
DBmapped	14 min	2.3 GB
DBraw	29 min	4.4 GB

Table 7.1: Time and space information of the preprocessing step in LODminer.

Chapter 8

Discussion and Future Work

We recount the highlights and draw conclusions based on the extensive evaluation we performed. Overall, the most effective descriptors where the local descriptions followed by the global descriptions and finally the baselines. While it is unsurprising that some baselines performed poorly, it was surprising that cooccurrence performed badly at scale. This shows how assumptions must be adapted given the scale of datasets. Further, the initial baseline of returning a fixed ranking highlights the importance of performing the sampling in an appropriate way. Overall the directed property count (*DirPropertyCount*) performed the best, but we saw that computing similarity based on smoothing over larger neighborhoods had a negative effect on performance. For global descriptions, however, smoothing had a positive effect on performance (*DiffusionKernel*).

Comparing the performance of the methods across experiments, we found that the performance of global descriptions and local descriptions was uncorrelated, suggesting that combinations of these methods could boost overall performance. While simple convex combinations did not lead to improvements, taking the best ranking for both global and local descriptions, we see that for 90% of the cases, the deleted property was among the top 10 recommended. In future work, we may investigate how to combine the different descriptions in order to improve performance. Finally, we also investigated

the effect of degrading the quality of the datasets and found that property information is robust across all three datasets.

Interesting future work lays in the area between assertions (concrete facts) and rules (mechanisms to encode general phenomena). One possible approach is to use a reasoner or an inference engine in combination with a rich ontology and a set of facts. This approach would enable us to do "smarter" queries than the search engines we are using nowadays.

Bibliography

[1] F. N. Afrati, D. Fotakis, and J. D. Ullman, "Enumerating subgraph instances using map-reduce," *CoRR*, vol. abs/1208.0615, 2012.

[2] S. Auer, C. Bizer, G. Kobilarov, J. Lehmann, R. Cyganiak, and Z. G. Ives, "Dbpedia: A nucleus for a web of open data," in *ISWC/ASWC*, 2007, pp. 722–735.

[3] L. Backstrom and J. Leskovec, "Supervised random walks: predicting and recommending links in social networks," in *WSDM*, 2011, pp. 635–644.

[4] P. Barnaghi and M. Presser, "Publishing linked sensor data," in *The 3rd International workshop on Semantic Sensor Networks 2010 (SSN10)*, 2010.

[5] V. Batagelj and A. Mrvar, "A subquadratic triad census algorithm for large sparse networks with small maximum degree," *Social networks*, vol. 23, no. 3, pp. 237–243, 2001.

[6] R. Bellman, "On a routing problem," DTIC Document, Tech. Rep., 1956.

[7] T. Berners-Lee *et al.*, "Linked data-the story so far," *International Journal on Semantic Web and Information Systems*, vol. 5, no. 3, pp. 1–22, 2009.

[8] U. Brandes, "A faster algorithm for betweenness centrality," *Journal of Mathematical Sociology*, vol. 25, pp. 163–177, 2001.

[9] I. Bratko, *Prolog: programming for artificial intelligence*. Addison-Wesley, 2001.

[10] C. Bron and J. Kerbosch, "Algorithm 457: finding all cliques of an undirected graph," *Commun. ACM*, vol. 16, no. 9, pp. 575–577, Sep. 1973. [Online]. Available: http://doi.acm.org/10.1145/362342.362367

[11] A. Carlson, J. Betteridge, B. Kisiel, B. Settles, E. R. H. Jr., and T. M. Mitchell, "Toward an architecture for never-ending language learning," in *AAAI*, 2010.

[12] T. H. Cormen, C. E. Leiserson, R. L. Rivest, and C. Stein, *Introduction to algorithms*. MIT press, 2001.

[13] P. Cudré-Mauroux, P. Haghani, M. Jost, K. Aberer, and H. de Meer, "idmesh: graph-based disambiguation of linked data," in *WWW*, 2009, pp. 591–600.

[14] E. Dijkstra, "A note on two problems in connexion with graphs," *Numerische mathematik*, vol. 1, no. 1, pp. 269–271, 1959.

[15] D. Eppstein, M. Löffler, and D. Strash, "Listing all maximal cliques in sparse graphs in near-optimal time," *CoRR*, vol. abs/1006.5440, 2010.

[16] D. Eppstein and D. Strash, "Listing all maximal cliques in large sparse real-world graphs," *CoRR*, vol. abs/1103.0318, 2011.

[17] O. Etzioni, M. Banko, S. Soderland, and D. S. Weld, "Open information extraction from the web," *Commun. ACM*, vol. 51, no. 12, pp. 68–74, Dec. 2008. [Online]. Available: http://doi.acm.org/10.1145/1409360.1409378

[18] R. W. Floyd, "Algorithm 97: Shortest path," *Commun. ACM*, vol. 5, no. 6, pp. 345–, Jun. 1962. [Online]. Available: http://doi.acm.org/10.1145/367766.368168

[19] A. V. Goldberg, "Shortest paths in road networks," *Slides, Microsoft Research*, 2011.

[20] A. V. Goldberg and C. Harrelson, "Computing the shortest path: A search meets graph theory," in *Proceedings of the sixteenth annual ACM-SIAM symposium on Discrete algorithms*, ser. SODA '05. Philadelphia, PA, USA: Society for Industrial and Applied Mathematics, 2005, pp. 156–165. [Online]. Available: http://dl.acm.org/citation.cfm?id=1070432.1070455

[21] Google, "Google knowledge graph," http://www.google.com/insidesearch/features/search/knowledge.html.

[22] J. Leskovec, D. Huttenlocher, and J. Kleinberg, "Predicting positive and negative links in online social networks," in *Proceedings of the 19th international conference on World wide web*, ser. WWW '10. New York, NY, USA: ACM, 2010, pp. 641–650. [Online]. Available: http://doi.acm.org/10.1145/1772690.1772756

[23] R. Milo, S. Shen-Orr, S. Itzkovitz, N. Kashtan, D. Chklovskii, and U. Alon, "Network motifs: simple building blocks of complex networks," *Science Signaling*, vol. 298, no. 5594, p. 824, 2002.

[24] N. Nakashole, M. Theobald, and G. Weikum, "Scalable knowledge harvesting with high precision and high recall," in *Proceedings of the fourth ACM international conference on Web search and data mining*, ser. WSDM '11. New York, NY, USA: ACM, 2011, pp. 227–236. [Online]. Available: http://doi.acm.org/10.1145/1935826.1935869

[25] M. Nickel, V. Tresp, and H.-P. Kriegel, "Factorizing yago: scalable machine learning for linked data," in *WWW*, 2012, pp. 271–280.

[26] E. Oren, S. Gerke, and S. Decker, "Simple algorithms for predicate suggestions using similarity and co-occurrence," in *Proceedings of the 4th European conference on The Semantic Web: Research and Applications*, ser. ESWC '07. Berlin, Heidelberg: Springer-Verlag, 2007, pp. 160–174. [Online]. Available: http://dx.doi.org/10.1007/978-3-540-72667-8_13

[27] S. Rudolph, "Acquiring generalized domain-range restrictions," in *Proceedings of the 6th international conference on Formal concept analysis*, ser. ICFCA'08. Berlin, Heidelberg: Springer-Verlag, 2008, pp. 32–45. [Online]. Available: http://dl.acm.org/citation.cfm?id=1787746.1787749

[28] M. Sabou, M. Dzbor, C. Baldassarre, S. Angeletou, and E. Motta, "Watson: A gateway for the semantic web," in *Poster session of the European Semantic Web Conference, ESWC*, 2007.

[29] J. Shawe-Taylor and N. Cristianini, *Kernel Methods for Pattern Analysis*. Cambridge University Press, 2004.

[30] K. Simonič, J. Rupnik, and P. Škraba, "Lodminer," http://lodminer.net/.

[31] ——, "Predicting missing properties in linked data datasets," http://lodminer.net/pubs/missing_properties_KDD.pdf, 2013.

[32] R. Tarjan, "Depth-first search and linear graph algorithms," *SIAM journal on computing*, vol. 1, no. 2, pp. 146–160, 1972.

[33] J. Völker and M. Niepert, "Statistical schema induction," in *ESWC (1)*, 2011, pp. 124–138.

[34] W3C, "Resource description framework (rdf): Concepts and abstract syntax," http://www.w3.org/TR/2004/REC-rdf-concepts-20040210/.

[35] S. Wernicke, "Efficient detection of network motifs," *IEEE/ACM Trans. Comput. Biol. Bioinformatics*, vol. 3, no. 4, pp. 347–359, Oct. 2006. [Online]. Available: http://dx.doi.org/10.1109/TCBB.2006.51

[36] L. Yen, A. Pirotte, and M. Saerens, "An experimental investigation of graph kernels on collaborative recommendation and semisupervised classification," *Information Systems Research*, pp. 1–39, 2009. [Online]. Available: http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.157.8467&rep=rep1&type=pdf