PERFORMANCE PREDICTION OF OWL REASONERS

A dissertation submitted to the University of Manchester
for the degree of Master of Science
in the Faculty of Engineering and Physical Sciences

2013

By
Viachaslau Sazonau
School of Computer Science

Contents

Word count: 20578

List of Tables

List of Figures

Abstract

In the last decade, ontologies have become a widely used paradigm for knowledge representation. In particular, growing numbers and sizes of ontologies are evident in biomedical applications. Ontologies are encoded in ontology languages, such as the common Web Ontology Language, OWL.

A variety of efficient reasoners, optimized implementations of algorithms for reasoning problems of high worst-case complexity, have been developed which allow obtaining logical inferences from the represented knowledge. However, the same ontology can be processed in seconds by one reasoner and in hours or even days by another reasoner. This is very hard to predict even for experienced ontology engineers.

This work investigates the computational performance of reasoners across ontologies. We explore ontology properties and their impact on the performance. We define new ontology features in addition to already studied features and estimate how they correlate with the computation time per reasoner. We also investigate correlation between ontology features.

We introduce a novel approach for the performance prediction based on samples of limited sizes retrieved from an ontology. We design ontology sampling algorithms and assess their suitability for providing indicative samples for the performance prediction which is given by regression analysis of samples. We also investigate how prediction results depend on sample sizes.

We study and test a common approach to the performance prediction which is based on supervised machine learning. Finally, we evaluate the proposed approach and compare it with the existing one.

Declaration

No portion of the work referred to in this dissertation has been submitted in support of an application for another degree or qualification of this or any other university or other institute of learning.

Copyright

Acknowledgements

I would like to thank Ulrike Sattler for her precious advises and prompt feedback. She has contributed a lot to understanding the results and changed my vision of critical analysis of experiments. I am also grateful to Gavin Brown for his feedback and inspiring me to focus on the final goal. I would like to thank Nicolas Matentzoglu for providing ontologies for the corpus, Chiara Del Vescovo for her explanations of the Atomic Decomposition, Dmitry Tsarkov for the Atomic Decomposition implementation, and Rafael Gonçalves for his help with performance measurement techniques. All these people have made this thesis possible.

Chapter 1

Introduction

1.1 Motivation

An ontology is "a formal explicit specification of a shared conceptualization of a domain of interest" [10]. In other words, an ontology is a "machine-processable" representation of knowledge about a domain of interest. The knowledge includes terms and relationships between them. Ontologies are encoded using knowledge representation languages which are formally based on mathematical logic.

A common ontology language is the Web Ontology Language [14], OWL, which is a World Wide Web Consortium, W3C, recommendation. A significant number of OWL ontologies have been developed for academia and industry. In particular, this is evident in such areas as medicine, bioinformatics, and geography where many new ontologies have been created while old ones have been considerably extended. The NCBO BioPortal repository[1] contains a collection of such ontologies.

Numerous reasoners, specialised software tools, have been developed to support ontology engineering. Reasoners allow to check the consistency of the encoded knowledge and provide logical inferences. Logical inferences are consequences that follow from the encoded knowledge. For example, a reasoner can determine the relationship between OWL classes in an ontology by inferring that one class is a subclass of another, even if it is not encoded explicitly. This reasoning service is called classification. We will discuss it later.

Among commonly used reasoners are FaCT++ [29] and its Java version JFact, Hermit [22], Pellet [27], ELK [18]. Technically, each reasoner uses a set of specific

[1] `http://bioportal.bioontology.org/`

techniques, tuned calculi and implementation tricks combined into one tool which aims to reason over ontologies as efficiently as possible. It means that different reasoners may perform differently on the same input because some techniques may succeed on this particular input while others may fail. In practice, this difference can be enormous [9]. In other words, the same ontology can take seconds for one reasoner to process while it can require minutes or even hours for another, depending on internal implementation tricks.

The things become even more complicated due to multiple versions of the reasoner implementation. Running the reasoner on a particular ontology, one can realize that the reasoner performs poorly on this input and may attempt to refine the implementation. However, it may make the reasoner more specialized on this particular type of ontologies which means that the reasoner may become performing worse over other ontologies.

All these factors make performance prediction of a reasoner on a given input challenging and often impossible at all, even for experienced and skilled ontology engineers. However, the ability to predict the performance of a reasoner on a given ontology with some degree of precision is attractive.

Firstly, reasoner developers, equipped with such kind of a tool, would be able to estimate the hardness of an ontology for a particular reasoner and to find out which ontologies are harder for the reasoner before literally processing them all. This can fasten tests of reasoners and assessment of new optimizations.

Secondly, an approximate estimate of the computation time might be helpful because it could considerably reduce efforts within ontology development allowing an ontology engineer to decide whether the reasoning task is worth waiting or not. For example, Protégé[2], a common ontology editor, has a status bar showing the progress of reasoning task execution. Unfortunately, this progress bar does not work properly in many cases. It can be significantly enhanced by the performance predictor estimating the computation time for a chosen reasoner on a given ontology.

Thirdly, understanding which ontology properties cause its hardness for reasoners is a potential opportunity to avoid such properties during ontology design and development. In addition, this knowledge could also help to develop new techniques in order to cope with computational difficulties and facilitate new optimizations of reasoners.

[2] `http://protege.stanford.edu/`

Thus, predicting the performance of reasoners on ontologies is interesting from both theoretical and practical prospectives. This inspires recent attempts to predict the performance of reasoners using machine learning techniques [16].

1.2 Aims and objectives

The aim of this project is to explore the computational performance of reasoners over ontologies, investigate the sources of the computational hardness and develop an approach that can predict the computation time of a reasoner on an ontology based on the ontology properties.

First of all, we prepare an ontology corpus for investigations. We investigate ontology properties and their influence on the computation time of a reasoner. We seek the properties which are indicative for the performance prediction.

Some ontology properties can be explored by extracting a set of features from an ontology which reasonably describe it. This raises the question which features can be extracted from an ontology and which features correlate with its computation time more than others. In order to investigate this question, impact values of features can be estimated and the most influential features can be identified.

We also investigate how ontology features are intercorrelated. This involves applying common methods of data mining and visualisation.

We study a common approach to the performance prediction of reasoners based on ontology features. We plan to design a new performance predictor. For these purposes, we study standard methods of prediction and forecasting and check their suitability for the performance prediction. This can additionally help us to gain insights on the behaviour of reasoners on existing ontologies.

The deliverables include a developed tool that processes a given ontology, explores its properties and predicts the computation time of a reasoner on it.

1.3 Thesis overview

The thesis contains 6 chapters. Chapter 1 introduces the problem of predicting the reasoning performance, discusses why it is worth investigating and which methods can help to gain insights. It also contains a brief summary of main findings.

Chapter 2 introduces basic concepts of ontology reasoning and describes the

research background for the problem. It discusses related works in the area and describes existing techniques which we use in our investigation.

Chapter 3 starts with open questions and research goals. Then, it describes the process of designing performance predictors, designed methods and proposed solutions in details.

Chapter 4 describes the ontology corpus as a data set for experiments and lists used software tools.

Chapter 5 contains analysis of ontology features, performance prediction experiments on the ontology corpus and their outcomes.

Chapter 6 summarises results and gained insights, makes conclusions and discusses future work.

1.4 Main findings

We defined a set of 52 ontology features by adding 28 new features to already studied ones [16]. These features aimed to capture different information and describe an ontology from different sides of the global view. However, we discovered that ontology features are highly intercorrelated. Surprisingly, all ontology features can be replaced by just one feature, the ontology size, with insignificant loss of information.

There are recent attempts to apply supervised machine learning to the performance prediction using multiple ontology features [16], which we call global approaches. We followed this way and trained a machine learning model using 52 ontology features. Then, we trained the model using just one feature, the ontology size, and compared the results. We found out that the model with the single feature showed better, sometimes considerably better, prediction accuracy than the one with all features. This means that extracting multiple ontology features for the performance prediction makes no or little sense.

We attempted to explore an ontology from the local view. We sought internal properties which were indicators of the reasoning performance using graph-based ontology representations. For these purposes, we chose a suitable graph. We proposed the idea of building small ontology subsets, called samples, able to reflect the performance of a reasoner on the full ontology. We designed three sampling algorithms providing those samples. We found out that pursuing certain directions in this graph could bear "good" samples of small sizes for a reasoner. We

also found out that these samples were indicative for the performance prediction.

In this work, we introduce a novel approach for the performance analysis and prediction. We call it a local approach because it is based on small samples retrieved from an ontology part. We applied regression analysis to predict the performance of a reasoner on the full ontology using samples retrieved from it. We discovered that a simple polynomial curve of degree 1 or 2, fitted to these samples, could deliver better prediction results than the common machine learning approach using either profiles or only sizes of multiple ontologies in the corpus.

Chapter 2

Background and Related Work

2.1 Ontology and OWL reasoning basics

Many ontology languages are originated from *Description logics*, DLs [4], which are decidable fragments of first order logic, FOL. Hence, an ontology is a set of logical formulae, so-called *axioms*. A simple ontology Bird is shown in Example 2.1.1.

Example 2.1.1. Let Bird be the following ontology in DLs:

$$\begin{aligned}\{\texttt{Rodent} &\sqsubseteq \texttt{Animal},\\ \texttt{Owl} &\sqsubseteq (\texttt{Animal} \sqcap \exists \texttt{eats}.\texttt{Rodent}),\\ \texttt{Carnivore} &\equiv (\texttt{Animal} \sqcap \exists \texttt{eats}.\texttt{Animal})\}.\end{aligned}$$

This ontology can be translated into FOL as follows:

$$\begin{aligned}\{&\forall x(\texttt{Rodent}(x) \Rightarrow \texttt{Animal}(x)),\\ &\forall x(\texttt{Owl}(x) \Rightarrow (\texttt{Animal}(x) \wedge \exists y(\texttt{eats}(x,y) \wedge \texttt{Rodent}(y)))),\\ &\forall x(\texttt{Carnivore}(x) \Leftrightarrow (\texttt{Animal}(x) \wedge \exists y(\texttt{eats}(x,y) \wedge \texttt{Animal}(y))))\}.\end{aligned}$$

An ontology has its *signature* which is a set of named classes, named properties and named individuals appearing within the ontology.

Definition 2.1.2. (Bench-Capon *et al.*, 1999) A *signature* of an axiom $\alpha \in \mathcal{O}$, denoted as $\tilde{\alpha}$, is a set of terms occurring in α.
Consequently, a signature of an ontology $\mathcal{O}$, denoted as $\tilde{\mathcal{O}}$, is a set of terms occurring in $\mathcal{O}$.

Thus, the ontology signature describes the used vocabulary. For instance, the signature of the Bird ontology in Example 2.1.1 is

$$\tilde{\mathcal{O}} = \{\texttt{Animal}, \texttt{Rodent}, \texttt{Owl}, \texttt{Carnivore}, \texttt{eats}\}.$$

Here, `Animal`, `Rodent`, `Owl`, `Carnivore` are class names, `eats` is a property name.

The represented knowledge about the domain of interest should be unambiguous and should contain no contradictions. Therefore, a representation, namely an ontology, should conform to these restrictions. Recalling that an ontology is a set of logical formulae, one can apply reasoning services to check the consistency of an ontology. Moreover, these services can reveal additional consequences, logical inferences, called *entailments*, from the set of axioms. For instance, the reasoning service can derive the following entailment η from the Bird ontology shown in Example 2.1.1:

$$\{\eta \ : \ \texttt{Owl} \sqsubseteq \texttt{Carnivore}\}.$$

In other words, reasoning transforms implicit relationships into explicit ones and, hence, supports the discovery of new knowledge about the domain that the ontology encodes. Tools that implement reasoning services are called *reasoners*. Thus, reasoners are important and helpful tools that facilitate an ontology engineering process due to intermediate checks of the knowledge consistency and derived logical inferences, or discovered hidden relationships between terms.

Ontology languages generally differ in expressivity. The family of commonly used DL languages contains inexpressive but tractable languages when the reasoning task can be completed in polynomial time [2, 5]. At the same time, it also includes highly expressive languages whose decision procedure belongs to the NExpTime complexity class [17]. The NExpTime complexity class contains decision problems that can be solved by an algorithm in $O(2^{p(n)})$ time, or in the number of iterations bounded by $2^{p(n)}$ where $p(n)$ is a polynomial of the input size n. It is said that the decision procedure has the worst-case computational complexity of $O(2^{p(n)})$ in the size n of the input. The size of an ontology can be defined as follows: $n = \sum_{\alpha \in \mathcal{O}} length(\alpha)$.

Ontologies encoded in highly expressive languages can be hard to reason over even if they have a relatively small size. In other words, larger ontologies can

be computationally easier than smaller ones, sometimes significantly. Recent research supports this fact and shows that the size of an ontology is not the only source of its computational hardness [9] for reasoners, which points to the amount of time necessary to execute a reasoning procedure.

There are different inference services that reasoners provide. A standard service invoked by ontology engineers is the *ontology classification*. The ontology classification is a reasoning procedure that comprises:

- checking the ontology consistency;
- testing the satisfiability of each *named* class in the ontology that requires up to k tests, where k is the number of named classes in the ontology;
- checking subsumption relationships for each possible pair of named classes that requires up to k^2 tests, where k is the number of named classes in the ontology.

Thus, the ontology classification needs at most $(1 + k + k^2)$ tests in total for k named classes. The worst-case complexity of each test depends on the expressivity of the logic underpinning the ontology language. Hence, a single test can be computationally hard for a reasoner.

We focus on the ontology classification in this work. Wherever computation time of a reasoner is mentioned, the time required to complete the ontology classification is meant.

The ontology classification should not be confused with *machine learning classification* where the most suitable class is predicted for a given input by a machine learning classifier being learned on training examples. The machine learning classification exploits a statistical-based model.

Let $\mathcal{CT}(\mathcal{O}, \mathcal{R})$ be a computation time estimate given ontology $\mathcal{O}$ and reasoner $\mathcal{R}$, such that $\mathcal{CT}(\mathcal{O}, \mathcal{R})$ measures the computation time of running $\mathcal{R}$ over $\mathcal{O}$ until the termination.

2.2 Performance of reasoners across ontologies

2.2.1 Performance variability phenomena

The huge difference in the performance of reasoners on the given input often surprises ontology designers. The same ontology may take just several seconds to

reason over for some reasoners and require hours to process for others [9]. There is no single reasoner that performs best on all given inputs. Recent results of the competition between 14 reasoners taking place at the OWL Reasoner Evaluation Workshop, ORE2013[1], show that there is no single winner in all ontology categories. Some reasoners perform better on particular kinds of inputs while fail on others. These phenomena are similar to ones observed in evaluation of machine learning classifiers where there is no single classifier working best for all data sets that can be partially explained by the *"no free lunch" theorem* in search and optimization [31].

Phenomena observed from repercussions of ontology maintenance can be even more confusing. For example, adding several axioms which look simple can make the ontology significantly harder for reasoning. As a result, it can increase the computation time considerably. The situation with removing some axioms in order to make the ontology easier and "help" the reasoner to perform faster is even more surprising. Common expectations of such a procedure are that it should cause a drop of the computation time since the ontology size is decreased. In fact, axiom removals can easily rise the computation time in contrast to intuitive expectations.

The Description Logics $\mathcal{SHOIQ}$ and $\mathcal{SROIQ}$ underpinning OWL are expressive DLs whose core reasoning problems belong to the NExpTime complexity class [17]. Despite this fact, many reasoners perform impressively well across real-world ontologies available on the Web. This is achieved by a wide range of optimizations and implementation tricks which reasoners are equipped with.

Nevertheless, there still exist hard ontologies some of which have been found impossible to process for a decade. Galen is one example of such ontologies. Such a problem can be solved in two possible ways. The first one is to develop more efficient reasoners. The second one is to make ontologies easier to process, for example, by reducing the expressivity of languages in which ontologies are encoded. However, nothing comes without a price. Simplified expressivity implies that fewer constraints can be represented in the given language.

Nonetheless, computational benefits are always attractive for practical applications that inspire active research in tractable Description Logics such as $\mathcal{EL}$ family [3] and DL-lite [5], for which reasoning is decidable in polynomial time and which exhibit robustness to an arbitrary input. Some reasoners, for example,

[1] http://ore2013.cs.manchester.ac.uk/competition/âĂŰ

ELK [18] for $\mathcal{EL}$, are specifically tuned for ontologies in such inexpressive but tractable languages. It should be noted though, that an inexpressive language itself does not guarantee easiness of processing. There exist inputs in tractable languages which can be surprisingly hard to process [8].

One possible workaround to cope with hard inputs is to approximate the reasoning procedure itself which is in some sense similar to shifting to less expressive ontology languages. It usually means to sacrifice either soundness, the property of the procedure to return no wrong "yes" answers, or completeness, the property that the procedure always answers "yes" if it should do so. Approximate reasoning can increase the performance and make it more predictable but the cost is uncertainty of obtained results [24].

2.2.2 Performance prediction problem

Significant performance variability of reasoners across ontologies stimulates interest in the performance prediction. Indeed, predicting how long an ontology will take to process for a reasoner will serve well for ontology engineers and researchers, see Section 1.1. However, how hard is the performance prediction problem?

In fact, predicting the performance of OWL reasoners is hard in practice. For example, it is much harder than predicting the satisfiability of a propositional k-CNF formula, so-called k-SAT problem [26]. A propositional k-CNF formula C can be described by just two main parameters: the number of clauses $|C|$ and the number of distinct variables $|\widetilde{C}|$. It is observed experimentally [19] that the k-SAT problem instance is highly determined by just one parameter, *formula density*, which is the ratio:

$$\rho(C) = \frac{|C|}{|\widetilde{C}|}. \tag{2.1}$$

The larger the density is, the more formulae with this ratio are unsatisfiable [19]. Thus, in order to predict satisfiability of a given formula, it suffices to know the formula density.

In addition, the point where 50% of the formulae are unsatisfiable, so-called phase transition point, is where the *computationally hardest* formulae appear. The hardness declines as the distance to this point increases. Hence, the distance of $\rho(C)$ to the phase transition point allows to predict the performance. It should

be noticed that we do not need to execute any reasoning algorithms, or solvers, on a formula to make a prediction. The formula structure solely gives indicative information for predictions and this information is easy extractable.

However, the situation changes if we shift to more expressive yet still relatively simple logics as $\mathcal{ALC}$ and $\mathcal{ALCQ}$ because the number of parameters that have to be considered grows rapidly [13]. Hence, extracting indicative information from formulae becomes significantly harder or even infeasible at all. Theoretical explanations and performance predictions are significantly complicated due to complex influence of multiple parameters. For example, one can consider a signature of an ontology. Although the signature size can provide some useful information about how hard the ontology can be for reasoners to process, elements occurring in the signature can possibly be combined in numerous different ways and can form both easy and hard sets of formulae.

As it is pointed above, each reasoner has its own optimization and implementation tricks that makes it powerful on some inputs and poor on others. Given an ontology, it is difficult to predict which reasoner tricks will succeed and which will fail on this input in advance. It is even harder to estimate how these "wins" or "loses" will affect the final computation time.

All these facts and observations show that the performance prediction problem is hard and cannot be solved in any trivial way. In Section 2.4.1.3 recent attempts [16] to predict the reasoning performance by extracting different types of information from an ontology are discussed.

2.2.3 Performance homogeneous and heterogeneous ontologies

Recent studies of the performance variability suggest that there are ontologies which are *performance homogeneous* and others are *performance heterogeneous* for a reasoner [9].

Definition 2.2.1. (Gonçalves *et al.*, 2012) An ontology $\mathcal{O}$ is *performance heterogeneous* if there exists $\mathcal{M} \subset \mathcal{O}$ such that

1. $|\mathcal{M}| \ll |\mathcal{O}|$
2. $\mathcal{CT}(\mathcal{O} \backslash \mathcal{M}, \mathcal{R}) \ll \mathcal{CT}(\mathcal{O}, \mathcal{R})$.

Such a subset $\mathcal{M}$ is called a *hot spot*.
Otherwise, an ontology $\mathcal{O}$ is *performance homogeneous*.

In other words, an ontology $\mathcal{O}$ is performance heterogeneous if there exists at least one relatively small subset of this ontology whose removal results in a significant decrease of the computation time.

From the ontology engineering perspective, identification and localisation of hot spots may bear practical profits. In order to improve the efficiency, one can merely remove the hot spot from the ontology. It should be noted though, that there is no evidence that hot spots on their own are much harder than the remaining part of the ontology [9]. The latter suggests that interaction effects, or *entanglements*[2], come into play when the hot spot is a part of the ontology and, consequently, loose their impact on the performance when the hot spot is removed.

The most important finding of [9] for the objectives of this work is how the size of an ontology subset relates to the performance of a reasoner. In experiments, an ontology is divided into multiple parts *randomly*. An ontology subset is formed by combining several parts into one. For instance, if an ontology is divided into 8 parts, an ontology subset is formed by consequently adding the parts one by one and, hence, can be of size 1/8, 2/8, 3/8, 4/8, 5/8, 6/8, 7/8, and 8/8 of the whole ontology. An ontology is separated into several parts randomly many times. Each time subsets are combined as it is described and the reasoning time is measured for each subset. Results show that the computation time grows with the size on average. However, the study reveals that there are significant variations, sharp rises and drops of the reasoning time, as the subset size increases. The latter additionally shows that the problem of reasoning performance prediction across various inputs is hard. In particular, Figure 2.1 illustrates that, as the subset size grows, the computation time can drop significantly. This supports an assumption that the size of an ontology is probably not the only reason of its hardness for a reasoner.

Thus, prominent variability phenomena in the performance of reasoners on OWL ontologies raise an important and intriguing question, answering which could make significant influence on the area of ontology engineering: which phenomena and/or ontology properties are the sources of such variability and can

[2]The term "entanglements" further means interaction effects that occur between ontology subsets and cause rising reasoning time

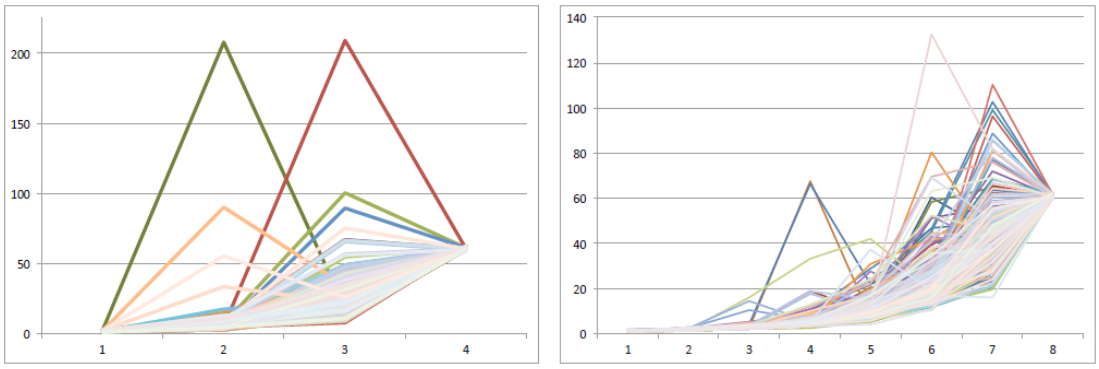


Figure 2.1: Computation time of the EFO ontology part against its size, the Pellet reasoner [9]

we predict the reasoner performance on the given input based on this *a priori* information?

2.3 Modular structure of ontologies

2.3.1 Modules

Users sometimes interact with just a small part of the ontology and do not need the remaining part. Ontologies can be built of hundreds of thousands of axioms and contain highly specialized parts which represent a particular topic within a domain of interest. For instance, SNOMED is an ontology of medical terms, the Systematized Nomenclature of Medicine [28], which consists of 500,000 axioms and covers a broad range of areas in medicine. However, a single doctor usually has just one specialization and, therefore, he is interested in very limited number of terms that the ontology encodes. The remaining and large part of the ontology appears irrelevant for his purposes and can be safely ignored. In fact, these specialized parts of an ontology may be weakly related by their meanings because they merely encode different topics.

These observations led to the idea that an ontology has a modular structure. First, the definition of a *logical module* needs to be introduced.

Definition 2.3.1. (Garson, 1989) Given a logical theory $\mathcal{T}$, a fragment $\mathcal{T}'$, not necessarily a subset, of $\mathcal{T}$ is a *logical module* if, for some background logic, the following two conditions hold:
1. $\mathcal{T}'$ is *locally correct*, i.e. any sentence provable in $\mathcal{T}'$ should be provable in $\mathcal{T}$.

2. $\mathcal{T}'$ is *locally complete*, i.e. every sentence in the signature of $\mathcal{T}'$ that is provable in $\mathcal{T}$ should be provable also in $\mathcal{T}'$.

The first condition means that all entailments, that follow from $\mathcal{T}'$, also follow from $\mathcal{T}$. The second one says that a logical module $\mathcal{T}'$ preserves all entailments of $\mathcal{T}$ over the signature $\tilde{\mathcal{T}}'$ and, therefore, can be used instead of $\mathcal{T}$ wherever the signature $\tilde{\mathcal{T}}'$ is considered. Thus, the latter proposes another opportunity to extract modules from an ontology: finding small subsets that preserve entailments instead of "slicing" it based on topic-specific terms. The logical property of preserving entailments is called *coverage*. However, a logical module may be not unique for a given signature. This is a reason of extending Definition 2.3.1 of a logical module with additional conditions which ensure that only one module exists for a given signature. Such a module is called Σ-module [25].

Although representing an ontology as a set of modules is interesting on its own, it does not provide any information on how these modules are interconnected and whether they have any relationships at all. On the other hand, those relations may be connected with entanglements. In turn, entanglements contribute to the performance variability of reasoners [9] and, hence, are of interest for this work. Thus, modules can help to investigate entanglements which are useful for the performance prediction.

2.3.2 Atomic Decomposition

One of methods of the ontology decomposition proposed recently is *Atomic Decomposition*(AD) [7]. The AD of an ontology is essentially a set of pairwise disjoint ontology subsets, called *atoms*, together with dependency relationships between them. The AD is the modular structure of the ontology induced by Σ-modules. Only Σ-modules are considered for the AD. To understand the AD, it is necessary to be familiar with some notions.

Definition 2.3.2. (Del Vescovo *et al.*, 2013) Given an ontology $\mathcal{O}$ and the set $\mathcal{F}(\mathcal{O})$ of all its modules, we define a *fake module* to be a module $\mathcal{M} \subseteq \mathcal{O}$ such that there are at least two modules $\mathcal{M}_1$ and $\mathcal{M}_2$ of $\mathcal{O}$, with neither $\mathcal{M}_1 \subseteq \mathcal{M}_2$ nor $\mathcal{M}_2 \subseteq \mathcal{M}_1$, and $\mathcal{M} = \mathcal{M}_1 \cup \mathcal{M}_2$. A module is called *genuine module* if it is not fake.

An important consequence of this definition is that every module can be obtained as a union of suitable genuine modules. In other words, the set of genuine

modules forms a basis for all modules.

Definition 2.3.3. (Del Vescovo *et al.*, 2013) Let $\mathcal{F}(\mathcal{O})$ be a set of all modules of an ontology $\mathcal{O}$. Then, an *atom* is a maximal set of axioms, such that for each module $\mathcal{M} \in \mathcal{F}(\mathcal{O})$, either they all occur in $\mathcal{M}$, or none of them does.

As a consequence, atoms $a \in AD$ are disjoint and their union is the whole ontology $\mathcal{O}$, or $\mathcal{O} = \dot{\bigcup}_{a \in AD} a$. The maximal possible number of atoms in an ontology equals to the number of axioms in it. The AD can be represented by a Hasse diagram which can be drawn as a *directed acyclic graph*. There is a one-to-one correspondence between genuine modules and atoms. Relationships between atoms are based on preserving entailments.

An example of the AD of the toy ontology Koala is illustrated on Figure 2.2. The ontology contains 42 axioms. There are 19 atoms in its AD.

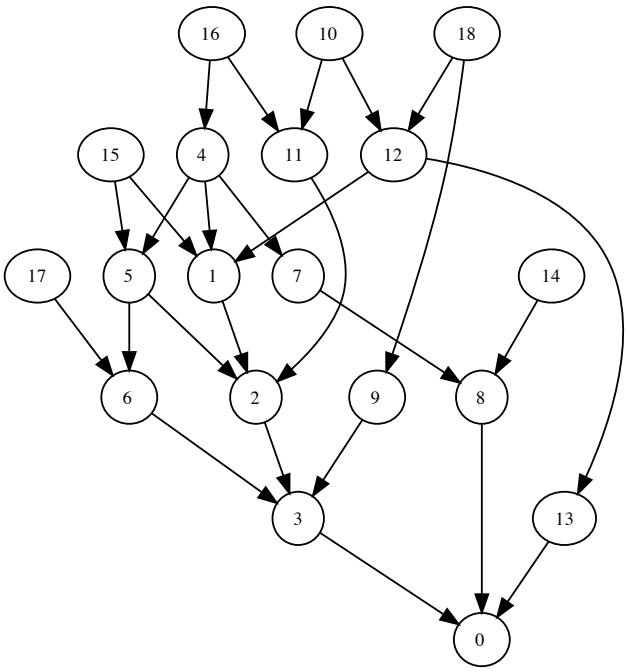


Figure 2.2: Atomic Decomposition of the ontology Koala [7].

Recently developed reasoners CHAINSAW [30] and MORe [1] attempt to make use of the modular structure of the ontology to improve their efficiency. Both of them exploit modularization by splitting ontologies into fragments which are

either sufficient to complete the reasoning task (CHAINSAW) or delegate the subset to a more efficient but specific reasoner (MORe).

The modular structure can also be useful for reasoning performance evaluation and prediction that this work investigates.

2.4 Prediction and forecasting

2.4.1 Machine learning

2.4.1.1 Classification and prediction

Assume there is a finite collection of objects of the same type: texts, images, ontologies, etc. Each object from the collection has its own label which is not necessarily unique. Hence, the objects can be separated into different categories, or classes, based on their labels. Then, a new object of the same type appears and the task is to predict its label, or to classify it into an appropriate category. It is called the *machine learning classification* problem.

In order to make a decision about an unknown object, some information needs to be extracted from the object and should describe its properties. This information should support decision making. It is commonly represented as a multidimensional vector. The vector variables are called *features* and the vector is called a *feature vector*. Then, to classify the unknown object, one can compare it with objects with known labels. Obviously, feature vectors of the known objects must be known for that. The feature vectors of known objects form training examples. A feature vector of an unknown object is a testing example.

Let us denote a set of N feature vectors of dimensionality d in matrix form:

$$\mathbf{X} = \begin{bmatrix} x_{11} & x_{12} & \dots & x_{1d} \\ x_{21} & x_{22} & \dots & x_{2d} \\ \vdots & \vdots & \ddots & \vdots \\ x_{N1} & x_{N2} & \dots & x_{Nd} \end{bmatrix}$$

where $\mathbf{x}_i = (x_{i1}, x_{i2}, \dots, x_{id})$, $i = 1 \dots N$, is i-th feature vector in the set. Then, the corresponding label is denoted as y_i, $i = 1 \dots N$ and the vector of all labels as $\mathbf{y}$. Thus, the data set is given by $[\mathbf{X} \mid \mathbf{y}^T]$.

Training examples together with corresponding labels are used to construct

a machine learning model automatically from the data. This process is called learning, or training. It is a common machine learning approach to train a model through "supervision", so-called *supervised machine learning*. In essence, a model is a mathematical data structure, for instance, a graph or probability distribution which is built from the data set.

In order to train a model, a variety of machine learning algorithms can be applied. Once the model is learned on the training data through a machine learning algorithm, it can be used to make predictions for testing examples. The basic supervised machine learning pipeline is shown on Figure 2.3.

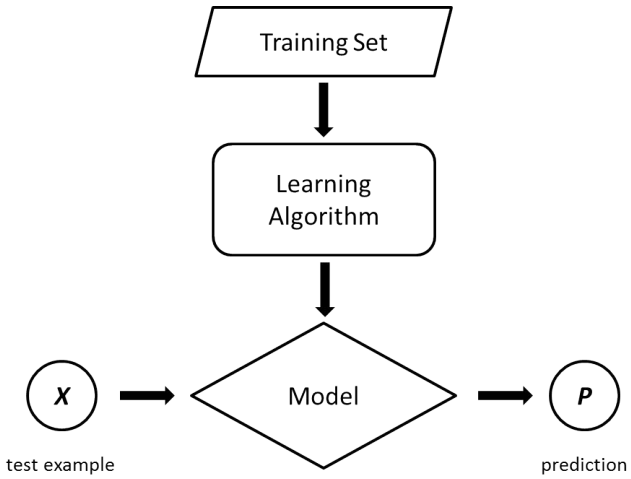


Figure 2.3: Supervised machine learning

Two points should be highlighted in relation to the above. First of all, a set of training examples has to be prepared in advance and a model must be trained on it. The process of training can take long time for hard problems. The second point is that accuracy of a model is largely determined by the training set. In fact, the model is just a generalization of the training data. Therefore, a training set should contain sufficient amount of data and should cover the range where testing examples can appear.

The object labels are not necessarily classes or categories that form a finite set. The labels can be continuous. In this case, a prediction is not a value from the set of classes but a continuous variable which can take any value from the allowed range.

2.4.1.2 Nearest neighbour classifier

The nearest neighbour classifier [6], NN, is among the simplest machine learning classifiers. It is essentially based on finding the closest training example for a testing example. It is done by calculating distances from the testing example to each training example using feature vectors and then taking the training example with the minimal distance. The Euclidean distance between feature vectors is commonly used as a metric. Once the closest training example is found, the testing example is just assumed having the same label.

A simple extension of the NN is the k-nearest neighbour classifier, k-NN [6]. In contrast to the NN, it finds k closest training examples and predicts the testing example to have the most common label among those k. There are different aggregation schemes to produce a prediction for continuous labels. The simplest one of them is calculating the average label of k nearest neighbours and predicting it to be the testing example label.

The k-NN has only one input parameter k. Its value should be chosen carefully because too small values of k cause the classifier to be excessively relying on the training set, or *overfitting*, and, hence, losing its generalization abilities. In contrast, too large values of k tend to insufficient use of the training data, or *underfitting*.

Despite its simplicity, the k-NN produces accurate predictions for many problems. In fact, it approximates the distribution of labels in the feature space and accuracy of this approximation is determined by parameter k. Similarly to all other machine learning classifiers, its prediction abilities are largely determined by the training set.

2.4.1.3 Machine learning application to the performance prediction

There exist recent attempts to apply supervised machine learning methods to the performance prediction of a reasoner on a given ontology [16]. In order to compute a descriptive vector for an ontology, features have to be extracted from it.

Two groups of features, called metrics, are introduced in [32]: ontology level (ONT) and class hierarchy level (CH) features. These features make use of a "graph-centric" ontology representation. Besides conventional ontology characteristics, such as the size of vocabulary, ONT features include properties of a

graph extracted from an ontology: the edge node ratio, tree impurity, the entropy of a graph. *CH* features measure the characteristics of the ontology class hierarchy: the number of children, depth of inheritance, class in-degree, class out-degree.

Kang et al. [16] introduce two new groups of features in addition to *ONT* and *CH* features described in [32]: anonymous class expressions (*ACE*), and properties (*PRO*). *ACE* metrics count the number of negations, conjunctions, disjunctions, universal/existential quantifiers, min/max/exact cardinalities. *PRO* features measure the number of property declarations: object/data type properties, functional, symmetric, transitive, inverse, inverse functional properties, and property equivalences. Overall, a set of 27 features is proposed.

Extracted features can then be used for predictions. *Kang et al.* suggest to discretize ontologies by their computation time. They define 5 performance bins, or classes, whose boundaries are given in seconds: $T \in (0s, 0.01s)$, $A \in [0.01s, 1s)$, $B \in [1s, 10s)$, $C \in [10s, 100s)$, $D \in [100s, +\infty)$. They exclude trivial ontologies $T \in (0s, 0.01s)$ from their experiments. Hence, we consider only 4 remaining bins further. The trained model attempts to predict a bin of a testing ontology given its feature vector. A separate model is trained for each reasoner. They assess a model by its classification accuracy.

They also investigate impacts of individual features from their set on the reasoning performance. According to their results, the most influential features are the number of existential quantifiers, the size of vocabulary, properties of the ontology graph, and characteristics of named classes. Their features capture structural aspect of an ontology. They do not consider a modular structure of an ontology and do not attempt to investigate entanglements.

The approach proposed by *Kang et al.* attempts to predict the performance based on an ontology "profile", or observe it *externally* as a whole. In other words, just one feature vector is computed for an ontology and this vector is the only data source for the prediction, except the trained model itself.

2.4.2 Regression analysis

2.4.2.1 Regression analysis versus machine learning

Prediction and forecasting tasks often make use of regression analysis [12], a set of methods aimed to explore the relationship between a dependent variable and

independent variables in a data set. In this interpretation, regression analysis has overlapping with machine learning. Indeed, a machine learning algorithm fits the data by a model via training. Doing so, it implicitly estimates the relationship between features, independent variables, and a label, a dependent variable. This relationship is encoded in a trained model.

A relevant to this thesis difference is related to the range of independent variables where predictions of the dependent variable are expected to be made. Supervised machine learning mainly focuses on predictions being made inside the interval where observations have been made, called *interpolation*. For these purposes machine learning allows to avoid making any assumptions about a function that determines the relationship between independent variables and a dependent variable. It merely approximates this function by a model fitted to the data. The cost of that is poor abilities to cope with examples appearing far from the observed interval where the model has been trained.

In contrast, some problems require forecasting beyond the observed interval, called *extrapolation*. This can be managed by regression analysis. However, the latter relies on some assumptions about the function of the variables relationship. The quality of predictions generally depends on the distance of a tested point from the observed interval: further points have greater uncertainty of predictions. In turn, the distance of meaningful predictions from the observed interval is determined by how reasonable the assumptions about the function are.

2.4.2.2 Regression models

A general regression model has form of a function with unknown coefficients that defines the relationship between independent variables and a dependent variable [12]:

$$y = f(\mathbf{x}, \beta), \tag{2.2}$$

where $\mathbf{x}$ is an input vector of independent variables, y is a dependent variable, β is a vector of unknown parameters.

Some assumptions need to be made about function f in order to apply regression analysis to the problem. The linear regression model is often used in

practice. It assumes that function f is linear:

$$y = \beta_0 + \sum_{j=1}^{d} \beta_j x_j. \tag{2.3}$$

One of the most popular methods for estimating unknown parameters β is *least squares* which minimizes the sum of residuals [12]:

$$RSS(\beta) = \sum_{i=1}^{N} (y_i - f(\mathbf{x}_i))^2 = \sum_{i=1}^{N} (y_i - \beta_0 - \sum_{j=1}^{d} \beta_j x_{ij})^2. \tag{2.4}$$

The residuals sum can be rewritten in matrix notations as follows:

$$RSS(\beta) = (\mathbf{y} - \dot{\mathbf{X}}\beta)^T (\mathbf{y} - \dot{\mathbf{X}}\beta), \tag{2.5}$$

where $\dot{\mathbf{X}} = [\mathbf{1}^T \mid \mathbf{X}]$.

Then, the unique solution [12] is obtained as:

$$\beta = arg\ \min_\beta\ RSS(\beta) = (\dot{\mathbf{X}}^T \dot{\mathbf{X}})^{-1} \dot{\mathbf{X}}^T \mathbf{y}. \tag{2.6}$$

Thus, the linear function f is fitted to the data by estimating its parameters β and the regression model is a hyperplane in the feature space. Hence, predictions beyond the observed interval become possible.

If there is only one independent variable x, the regression function f can be a simple polynomial of degree L:

$$y = \beta_0 + \sum_{l=1}^{L} \beta_l x^l. \tag{2.7}$$

Then, the polynomial can be fitted to the data by the *non-linear least squares* method [21]:

$$\beta = arg\ \min_\beta\ RSS(\beta), \tag{2.8}$$
$$\text{where } RSS(\beta) = \sum_{i=1}^{N} (y_i - \beta_0 - \sum_{l=1}^{L} \beta_l x_i^l)^2.$$

It is worth noting again that the meaningfulness of predictions strongly relies

on the assumptions about f. Therefore, unwise assumptions may cause meaningless predictions.

2.4.3 Dimensionality reduction

More features may allow to capture more information about the object. This may increase the precision of a model trained on examples capturing more properties of the problem space. However, extracting many features and then training more sophisticated model can be computationally expensive.

Furthermore, extracting more features does not necessarily lead to the model improvement. The high dimensionality of a feature vector often has an opposite effect: it degrades the accuracy of a trained model. These surprising but true facts are commonly observed in practice and referred to the term *"curse of dimensionality"*[3] proposed by Richard Ernest Bellman. The latter means that there exists an optimal number of features for a data set below which the accuracy of the model increases but above which it starts to decrease.

This happens due to several factors. A non-uniform data sampling across the dimensions is one of them. There may exist noisy or corrupted features. Finally, features may strongly correlate with each other. In other words, one feature may capture information that other features already contain. Thus, a smaller set of features may be more efficient than a larger one from both perspectives of computational expenses and model accuracy.

Decreasing the number of features, or dimensions of a feature vector, is known as dimensionality reduction. It can be achieved either by eliminating weakly useful features from the original set or by transforming the original set of features into a lower dimensional space with minimal loss of information. The first group of methods is known as feature selection while the second one is considered as projection methods.

One of common feature selection approaches is based on measuring Pearson's correlation coefficient [11] between labels and features in the data set:

$$r_{\mathbf{X}^j,\mathbf{y}} = \frac{\sum_{i=1}^{N}(x_{ij} - \bar{\mathbf{X}}^j)(y_i - \bar{\mathbf{y}})}{\sqrt{\sum_{i=1}^{N}(x_{ij} - \bar{\mathbf{X}}^j)^2}\sqrt{\sum_{i=1}^{N}(y_i - \bar{\mathbf{y}})^2}}, \tag{2.9}$$

where $\mathbf{X}^j$ is j-th feature, $\mathbf{y}$ is a vector of labels, $\bar{\mathbf{X}}^j$ and $\bar{\mathbf{y}}$ are mean feature

[3] `http://en.wikipedia.org/wiki/Curse_of_dimensionality`

and mean label values correspondingly. Thus, $r_{\mathbf{X}^j,\mathbf{y}}$ measures correlation of j-th feature with labels $\mathbf{y}$ in the data set $[\mathbf{X} \mid \mathbf{y}^T]$. The correlation coefficient can only have values between -1 and 1 inclusive, or $-1 \leq r \leq 1$. Values close to 1 imply very high correlation while values close to -1 show anticorrelation. Values near 0 indicate very low correlation or its absence if it is exactly 0.

Thus, features can be ranked according to their correlation values. However, a disadvantage of Pearson's correlation coefficient is that it does not allow to find out how the groups of features are correlated between each other in feasible way.

One of widely used projection methods is Principal Component Analysis, PCA [23, 15]. PCA projects feature vectors of the data set into a new coordinate system maximizing the variances in the projection space. In order to do that, first, the covariance matrix of the data set is computed:

$$\mathbf{S} = \frac{1}{N}\sum_{i=1}^{N}(\mathbf{x}_i - \bar{\mathbf{x}})^T(\mathbf{x}_i - \bar{\mathbf{x}}), \tag{2.10}$$

where $\bar{\mathbf{x}}$ is an average row vector over $\mathbf{X}$. Since the dimension of a feature vector is d, covariance matrix $\mathbf{S}$ is a $d \times d$ square matrix.

Then, found eigenvectors $\mathbf{u}_j = (u_{1j}, \ldots, u_{dj})^T,\ j = 1 \ldots d$ of the covariance matrix $\mathbf{S}$ are *principal components*, PCs, and eigenvalues $\lambda_j,\ j = 1 \ldots d$ are corresponding variances in the projection space:

$$\mathbf{S}\mathbf{u}_j = \lambda_j \mathbf{u}_j. \tag{2.11}$$

Feature vectors $\mathbf{x} \in \mathbf{X}$ can be considered as points in the coordinate system where features serve as coordinates. Then, a feature vector $\mathbf{x}$ can be projected into a new coordinate system via $\mathbf{U} = [\mathbf{u}_1\ \mathbf{u}_2\ \ldots\ \mathbf{u}_d]$ as follows:

$$\hat{\mathbf{x}} = (\mathbf{x} - \bar{\mathbf{x}})\mathbf{U}. \tag{2.12}$$

An advantage of this projection is that new coordinates are uncorrelated since eigenvectors are orthogonal, $\mathbf{u}_j\mathbf{u}_k = \delta_{jk}$, and, hence, can serve as a set of axes. However, one can notice that the dimensionality of a feature vector is not reduced by this projection.

Found eigenvalues λ_j are variances along new coordinates. For many data sets the amount of variance along the PC describes how much information the new feature contains [15]. It often occurs in practice that only few new coordinates

account for the most of variance. If such a case is observed, it indicates that the original features are highly intercorrelated. For example, if the dependence between two features is $\mathbf{X}^j = \mathbf{X}^k$, these two original coordinates can be replaced by only one new coordinate which accounts for all data variance with no loss of information.

Therefore, eigenvectors $\mathbf{u}_j$ can be sorted in *descending* order by their eigenvalues λ_j. Then, the first principal component, PC1, is one that has the largest eigenvalue, the second principal component, PC2, is one that has the second largest eigenvalue, and so on. Thus, we rank obtained eigenvectors $\mathbf{u}_j$ and replace their original indices with corresponding ranks.

In order to reduce the dimensionality, we can now select top $d' < d$ components and combine them into the matrix which is called a *projection* matrix:

$$\mathbf{U}_{d'} = [\mathbf{u}_1\ \mathbf{u}_2\ \ldots\ \mathbf{u}_{d'}],\ \text{such that}\ \lambda_1 > \lambda_2 > \cdots > \lambda_{d'}. \tag{2.13}$$

Then, the point $\mathbf{x}$ of the original feature space with dimensionality d can be projected into a new space of $d' < d$ coordinates as follows:

$$\mathbf{z} = (\mathbf{x} - \bar{\mathbf{x}})\mathbf{U}_{d'}. \tag{2.14}$$

where $\mathbf{z} = (z_1, z_2, \ldots, z_{d'})$ is a new feature vector. It follows from Equation 2.14 that z_1 is a new feature obtained via PC1 projection of $\mathbf{x}$, z_2 is a new feature obtained via PC2 projection of $\mathbf{x}$, and so on:

$$\begin{aligned} z_1 &= (\mathbf{x} - \bar{\mathbf{x}})\mathbf{u}_1 \\ z_2 &= (\mathbf{x} - \bar{\mathbf{x}})\mathbf{u}_2 \\ &\ldots \\ z_{d'} &= (\mathbf{x} - \bar{\mathbf{x}})\mathbf{u}_{d'}. \end{aligned} \tag{2.15}$$

To summarise, PCA can help to gain insights on the amount of correlation inside the set of features and provide a new set, which is smaller and where features are uncorrelated, with minimal loss of information.

Chapter 3

Designing Performance Predictors

3.1 Goals and open questions

Although to predict the computational performance of a reasoner on a given ontology is hard, see Section 2.2.2, it bears prominent benefits for both theoretical studies and practical applications, see Section 1.1. Let us discuss goals and open questions this thesis considers.

1. An ontology, as well as its part, can be described by a set of features, called a feature vector. Features should correlate per reasoner with the computation time, called a label. How to define and extract performance descriptive features from ontologies is investigated.

2. Some features may have greater impact on the computation time and, hence, correlate more with recorded time measurements. Therefore, each feature significance will be measured. The correlation between features is also worth investigating because it can show how the dimensionality of a feature vector can be reduced finally.

3. In order to test a supervised machine learning approach for the performance prediction, a data set of examples with labels have to be prepared for the ontology corpus. One example is prepared for one ontology and consists of its computed feature vector with an associated time measurement, one per reasoner. Then, a machine learning model for each reasoner can be trained and evaluated on the data set. This is further referred as *global approaches* which consider the application of supervised machine learning techniques to the performance prediction problem.

4. An ontology is a set of logical formulae, axioms. Its logical structure and entanglements may cause computational hardness. An ontology can be viewed as a set of axiom "clumps" with some common properties or interrelations between them. We plan to find out which notion of a clump and which method of assembling clumps together are indicative for entanglements and suitable for the performance prediction. This is the reason to explore graph-based representations of an ontology. The goal is to find an indicative one.

5. Once a suitable notion of a clump and a method of assembling clumps are found, we plan to design a performance predictor which allows to predict the computation time of the full ontology using only its part, ideally small, or to extrapolate, see Section 2.4.2.1. These attempts are further called *local approaches* and make use of regression analysis for the performance prediction.

6. Once such local approaches are specified, we plan to assess their utility for performance predictions on the selected corpus of ontologies. It should show whether this strategy is suitable for the performance prediction.

7. It is worth then to compare global and local approaches on the selected ontology corpus and find out which one and under which conditions performs better. These observations can help to gain insights on the performance prediction problem and behaviour of reasoners across different inputs.

3.2 Ontology features

3.2.1 Global features

As noted in Section 2.4.1.3, recent studies consider four groups of features: ontology level (ONT), class hierarchy level (CH), anonymous class expressions (ACE), and properties (PRO). We extend the set of features proposed by *Kang et al.* [16] adding our own features.

At the same time, we skip some features from their set. In particular, we discard ONT features which are based on the translation of an ontology to a specific graph because there are different possible ways of such a translation and there is no "best" translation.

We also skip *CH* features because we plan to extract features not only from the full ontology but from ontology subsets as well. In the latter case, the class hierarchy is not indicative because the subsets can be small and their class hierarchy can be trivial in most cases.

Thus, the set of proposed features overlaps with the set in [16]. Table 3.1 shows 52 ontology features including 28 new features which are marked with "*".

ID	Notation	ID	Notation
1	#ClassNameOccurrences*	27	#TransitiveObjectPropertyAxioms
2	#ObjectPropertyNameOccurrences*	28	#FunctionalDataPropertyAxioms
3	#DataPropertyNameOccurrences*	29	#DataPropertyRangeAxioms*
4	#ClassNames	30	#DataPropertyDomainAxioms*
5	#ObjectPropertyNames	31	#ObjectPropertyDomainAxioms*
6	#DataPropertyNames	32	#ObjectPropertyRangeAxioms*
7	#ObjectSomeValuesFrom	33	#SubObjectPropertyOfAxioms*
8	#ObjectAllValuesFrom	34	#SubDataPropertyOfAxioms*
9	#ObjectMinCardinality	35	#ReflexiveObjectPropertyAxioms*
10	#ObjectMaxCardinality	36	#EquivalentObjectPropertiesAxioms
11	#ObjectExactCardinality	37	#EquivalentDataPropertiesAxioms
12	#ObjectIntersectionOf	38	#DisjointObjectPropertiesAxioms*
13	#ObjectUnionOf	39	#DisjointDataPropertiesAxioms*
14	#DataExactCardinality	40	#DifferentIndividualsAxioms*
15	#DataAllValuesFrom	41	#SameIndividualAxioms*
16	#DataSomeValuesFrom	42	#ClassAssertionAxioms*
17	#DataMinCardinality	43	#ObjectPropertyAssertionAxioms*
18	#DataMaxCardinality	44	#DataPropertyAssertionAxioms*
19	#SubClassOfAxioms*	45	#DisjointUnionAxioms*
20	#DisjointClassesAxioms*	46	#HasKeyAxioms*
21	#EquivalentClassesAxioms*	47	#IrreflexiveObjectPropertyAxioms*
22	#AsymmetricObjectPropertyAxioms	48	#NegDataPropertyAssertionAxioms*
23	#SymmetricObjectPropertyAxioms	49	#NegObjectPropertyAssertionAxioms*
24	#InvObjectPropertiesAxioms	50	#SubPropertyChainOfAxioms*
25	#InvFunctionalObjectPropertyAxioms	51	#SWRLRules*
26	#FunctionalObjectPropertyAxioms	52	parsingDepth*

Table 3.1: The set of experimental ontology features

#ClassNames, #DataPropertyNames, #ObjectPropertyNames refer to the size of vocabulary that was studied before. Therefore, we consider these features as already studied ones, despite the fact they were not separately studied earlier according to our knowledge.

We introduce new features *#ClassNameOccurrences, #DataPropertyNameOccurrences, #ObjectPropertyNameOccurrences* that collect the number of *all occurrences* of named classes, data properties, and object properties accordingly without considerations whether there exist equally named occurrences or not.

This is based on the assumption that axioms with different numbers of occurrences of the same terms may have considerably different hardness.

We replace discarded *CH* features by *#SubClassOfAxioms, #DisjointClassesAxioms, #EquivalentClassesAxioms* which capture the same information.

We also include disjointness in the set of experimental features: *#DisjointClassesAxioms, #DisjointDataPropertiesAxioms, #DisjointObjectPropertiesAxioms*. A reasoner may require a lot of calculations to check all disjoint declarations in case of complex relations between terms. We observed this effect experimentally during execution of different reasoners on the IMGT ontology from the NCBO BioPortal repository[1].

We add assertion features, such as *#ClassAssertionAxioms*, *#ObjectPropertyAssertionAxioms*, *#DataPropertyAssertionAxioms* in order to investigate how assertions correlate with the performance.

We count the number of object and data property subsumption axioms, *#SubObjectPropertyOfAxioms* and *#SubDataPropertyOfAxioms*, because a complex hierarchy of object and data properties may cause many additional calculations that a reasoner has to perform. We also count reflexivity on object properties by *#ReflexiveObjectPropertyAxioms* and *#IrreflexiveObjectPropertyAxioms*.

We introduce a new feature for an axiom, a *parsing depth*, which is the depth of an axiom expression parse tree, or a maximal number of tree levels starting from a root expression to a leaf expression.

An axiom with several nested expressions is shown in Example 3.2.1.

Example 3.2.1.

$$\{\alpha : \texttt{CaliforniaHandRoll} \sqsubseteq (\texttt{HandRolls} \sqcap \forall \texttt{hasPart}.(\texttt{Temaki} \sqcap \forall \texttt{hasFilling}.(\texttt{Avocado} \sqcup \texttt{Crab} \sqcup \texttt{Mayonnaise} \sqcup \texttt{SesameSeeds})))\}.$$

The parsing depth of the axiom is 3.

The parsing depth of a set of axioms is the sum of parsing depths of each axiom. Numerous axioms with many levels of nested expressions may facilitate complex entanglements between occurring terms to appear. The parsing depth feature attempts to capture this information.

Overall, the features in Table 3.1 cover all types of logical axioms extractable from an ontology (19-51), an ontology signature (4-6), and an ontology length

[1] `http://bioportal.bioontology.org/`

(1-3, 7-18, 52). It should be noticed that adding features like *#Axioms*, *SignatureSize*, and *OntologyLength* to Table 3.1 does not make sense because their values are completely determined by already counted features. Therefore, features *#Axioms*, *SignatureSize*, and *OntologyLength* add no new information to a feature vector. The latter point does not imply, however, that the set of chosen features consists of uncorrelated features. This will be investigated further in this work.

3.2.2 Other features

Besides the features explicitly extractable from an ontology and listed in Table 3.1, there are ontology features which are hard to measure in such a clear way. The entanglements that an ontology exhibits are among those features. There is no definition and no procedure to extract entanglements from an ontology. Considering their usefulness for the performance prediction, this work attempts to develop an approach for implicitly measuring entanglements based on the reasoning performance on an ontology part.

3.3 Global approaches

3.3.1 Ontology profile

We call a single feature vector extracted from an ontology by processing *all* its axioms an *ontology profile*. Thus, it collects information from every axiom exhaustively which means that each variable in an ontology profile has its maximal value for a given ontology.

An instance of a profile, computed for the Vaccine ontology from the NCBO BioPortal repository, $\mathbf{x}$(Vaccine) is given by Example 3.3.1.

Example 3.3.1.

(17437	4031	173	3530	72	5	12835	10	1	0	1	3971	309
6	0	0	1	0	6126	1677	284	0	0	13	0	4
16	4	4	4	7	7	35	0	0	0	0	0	0
3	0	148	2	154	0	0	0	0	0	0	0	4960)

One can notice many zero components in this vector. Indeed, this means

that certain entities, for example, symmetric, ID=22, and asymmetric, ID=23, object properties, do not occur in the ontology at all. In fact, ontology profiles with many zero components are common and Example 3.3.1 is not a coincidence. Observing the data set used by *Kang et al.*, we have found that most feature vectors contain many zero components with their features, too.

3.3.2 Predictions by machine learning using profiles

Given an ontology corpus $\{\mathcal{O}_i\}_{i=1}^N$, we are able to compute a profile for each ontology. If, in addition, we know the computation time for each pair ontology/reasoner, then a set of profiles $\mathbf{x}_i = \mathbf{x}(\mathcal{O}_i)$ with labels $y_{ik} = \mathcal{CT}(\mathcal{O}_i, \mathcal{R}_k), i = 1 \dots N, k = 1 \dots M$ is a data set for model training and testing:

$$[\mathbf{X} \mid \mathbf{Y}_1^T \dots \mathbf{Y}_M^T], \tag{3.1}$$

where $\mathbf{Y}_k = (y_{1k}, \dots, y_{Nk})$ are performance measurements of a reasoner $\mathcal{R}_k$ across the ontology corpus.

Kang et al. separate all ontologies into bins, or classes, according to the computation time of a reasoner, see Section 2.4.1.3. However, how reasonable these boundaries are, how many bins suffice, and whether such binning makes sense at all are debatable questions, see Example 3.3.2.

Example 3.3.2. Assume that the model, which uses binning, predicts bin $D \in [100s, +\infty)$ for an ontology with true computation time of 95s. We know that the same model, which does not use binning, predicts 105s for the same ontology. Then, is it reasonable to count a prediction in the case with binning as a model failure?

On the other hand, assume that the model with binning predicts bin $C \in [10s, 100s)$ for another ontology with true computation time of 12s and we know that the same model without binning predicts 92s for this ontology. Then, should we count the prediction in the case with binning as a model success?

Therefore, we consider machine learning models which are able to predict both bins and exact computation times. The k-NN classifier is an example of such a model, see Section 2.4.1.2.

Once the data set is prepared, a machine learning model can be trained on it via a machine learning algorithm and then tested. We will discuss later how to do this reasonably given a single data set.

3.4 Local approaches

3.4.1 Two principles of a local approach

A local approach is aimed to provide performance predictions for the whole ontology, or "global" predictions, given the data gathered from its part, or "local" data. The local data are extracted from ontology subsets, ideally small, which we call *samples*. Each sample supplies its feature vector and a computation time measurement associated with the feature vector.

Principle 1. A local approach finds samples suitable for the performance prediction.

To satisfy this condition, found samples have to be indicators of entanglements. Otherwise, they are easy to process for all ontologies. This implies they provide information forcing to concern all ontologies as easy ones. Therefore, they are not suitable in this case.

For example, a trivial way is to take arbitrary subsets. However, they are not acceptable indicators of entanglements because they can be so only by chance, which is low, considering that the size of an ontology is sufficiently large.

Another possibility is to define the notion of a clump of axioms and relationships between clumps. A reasonable way to do that is to represent an ontology as a directed graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$, where $\mathcal{V}$ is a set of vertices and $\mathcal{E}$ is a set of directed edges. Each vertice $v \in \mathcal{V}$ is labelled with some set of axioms, $o(v) \subseteq \mathcal{O}$. Then, this set of axioms is called a *clump*. If, in addition, clumps are disjoint and their union is an ontology, or $\mathcal{O} = \dot{\bigcup}_{v \in \mathcal{V}} o(v)$, then the ontology representation is called a *decomposition*. Thus, a graph-based representation induces the notion of a clump that should be suitable for the performance prediction.

Obtaining such a representation, we can assemble clumps in many possible ways. Then, a sample is a union of some clumps. Clump assembling may cause indicative and not indicative samples. Therefore, a suitable way of clump assembling should be found in order to satisfy *Principle 1.*

We introduce *growing-based sampling* which is a method of clump assembling by iterative extensions of a sample via adding new clumps, or "growing" a sample. Its advantage is that it can preserve certain properties, possibly indicative for the performance prediction, from smaller to larger samples.

Principle 2. A local approach provides the performance prediction for the whole ontology using its samples.

In order to provide a meaningful prediction for the full ontology, or extrapolate, a local approach should use a suitable method. Considering that we attempt to use small samples, supervised machine learning is not suitable here because it mainly focuses on interpolation, see Section 2.4.2.1. Therefore, we apply regression analysis for the performance prediction via a local approach.

3.4.2 Growing-based sampling

3.4.2.1 Candidates for clumps

In order to achieve suitable for the prediction samples, we should satisfy two requirements: the notion of a clump should be appropriate and the method of clump assembling should be appropriate. Arbitrary subsets are not suitable candidates for clumps because reasonable relationships between clumps are hard to specify. Therefore, assembled samples are equal to arbitrary subsets of greater sizes.

Another notion of a clump is a set of all axioms $\alpha \in \mathcal{O}$ which contain a named ontology class Cl in their signature. We call such a clump *Class usage* of Cl:

$$CU(Cl) = \bigcup_{Cl \in \tilde{\alpha}} \alpha.$$

Then, $CU(Cl_1)$ is related to $CU(Cl_2)$, if Cl_2 appears in the signature of $CU(Cl_1)$. Thus, an ontology is represented as a directed graph:

$$\mathcal{G} = (\mathcal{V}, \mathcal{E}), \text{ where } v_i = Cl_i,\ v_i \in \mathcal{V},\ Cl_i \in \tilde{\mathcal{O}} \text{ and } o(v_i) = CU(Cl_i)$$
$$\text{with relationships } e(v_i, v_j) \in \mathcal{E} \text{ iff } Cl_j \in \widetilde{CU}(Cl_i).$$

Since CU clumps can overlap, the CU representation is not a decomposition.

Yet another reasonable way to define clumps is Atomic Decomposition, AD, see Section 2.3.2. In this case, clumps are atoms $a \in AD_{\mathcal{O}}$ which are pairwise disjoint sets of axioms and $\mathcal{O} = \dot{\bigcup}_{a \in AD_{\mathcal{O}}} a$. The relationships between clumps are based on preserving entailments. Then, an ontology is represented as a directed

acyclic graph:

$$\mathcal{G} = (\mathcal{V}, \mathcal{E}), \text{ where } v_i = a_i,\ v_i \in \mathcal{V},\ a_i \in AD_{\mathcal{O}} \text{ and } o(v_i) = AD_{\mathcal{O}}(a_i)$$
$$\text{with relationships } e(v_i, v_j) \in \mathcal{E} \text{ iff } e_{AD_{\mathcal{O}}}(a_i, a_j) \in \mathcal{E}_{AD_{\mathcal{O}}}.$$

Given $\mathcal{G} = (\mathcal{V}, \mathcal{E})$, we are able to design methods of assembling clumps together based on relationships between them.

3.4.2.2 Basic algorithm

Growing-based sampling is based on iterative extensions of an ontology subset by adding clumps in some reasonable way. It is a method of assembling clumps together. The sequence of constructed subsets forms a suite of ontology samples. We propose possible ways of growing-based sampling below.

Given a graph-based ontology representation $\mathcal{G} = (\mathcal{V}, \mathcal{E})$, the process of iterative extensions begins from a single clump which is an initial subset $o_1 = o(v_1) \subset \mathcal{O}$. A next subset o_{i+1} is constructed by extending the current subset o_i with a new clump $o(v_{i+1})$ of axioms:

$$o_{i+1} = o_i \cup o(v_{i+1}), \tag{3.2}$$
$$\text{where } v_{i+1} \in \mathcal{V} \text{ and } v_{i+1} \notin \{v_1, \ldots, v_i\}.$$

This is a general definition. A new clump at each step is allowed to be chosen from all set of clumps with the only constraint that it cannot be chosen more than once. However, this process does not involve relationships $\mathcal{E}$ to be used while selecting a next clump.

A subset extension strategy exploiting information about relationships between clumps gives opportunities to associate a computation time measure $\mathcal{CT}(o_j, \mathcal{R})$ with pursuing some directions in the graph. A sensible way of extending ontology subsets may cause significant increases of $\mathcal{CT}(o_j, \mathcal{R})$. If this situation takes place, it indicates the presence of entanglements in a given ontology.

We propose three algorithms of iterative extensions below. First, we propose a general algorithm 1 which implements the strategy of iterative extensions defined by Equation 3.2.

Sorted size limits $\{c_j\}_{j=1}^{n}$ determine the number and moments of time measurements $\{\mathcal{CT}(o_j, \mathcal{R})\}_{j=1}^{n}$. New clumps of axioms are added to a subset o_i until a size

Algorithm 1 Iterative extensions

Input: $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ an ontology graph-based representation, $\mathcal{R}$ a reasoner, $\{c_j\}_{j=1}^{n}$ subset size limits sorted in increasing order, m an optional parameter

Output: $\{\mathcal{CT}(o_j, \mathcal{R})\}_{j=1}^{n}$ a set of n time measurements by $\mathcal{R}$
$\mathcal{V}_0 \leftarrow \emptyset$
$o_0 \leftarrow \emptyset$
$H_0 \leftarrow \emptyset$
$v_1 \leftarrow random(\mathcal{V})$
$i \leftarrow 0$
for each j from 1 to n **do**
  while $|o_i| < c_j$ **do**
    $i \leftarrow i + 1$
    $\mathcal{V}_i \leftarrow \mathcal{V}_{i-1} \cup v_i$
    $o_i \leftarrow o_{i-1} \cup o(v_i)$
    $v_{i+1} \leftarrow nextClump(\mathcal{V}_i, H(v_i)?, m?)$
  end while
  $\mathcal{CT}(o_j, \mathcal{R}) \leftarrow measure(o_i, \mathcal{R})$
end for
return $\{\mathcal{CT}(o_j, \mathcal{R})\}_{j=1}^{n}$

limit c_j is exceeded, or $|o_i| \geq c_j$. Then, the computational time $\{\mathcal{CT}(o_j, \mathcal{R})\}_{j=1}^{n}$ is measured. After that, growing of the subset continues without time measurements until next size limit c_{j+1} is exceeded.

The subroutine $nextClump(\mathcal{V}_i, H(v_i)?, m?)$ specifies the behaviour of the general algorithm. It has one mandatory and two optional input arguments which will be explained below. A version of the subroutine $nextClump()$ is determined by these input parameters. As a result, it produces three different strategies of iterative extensions which are considered as three different algorithms for simplicity and clarity of referencing. These algorithms are described below.

3.4.2.3 RE, HE, NE algorithms

Random extensions, RE The RE algorithm 2 is based on the iterative process described by Equation 3.2. The only constraint for choosing a clump is that it should not have been selected previously. At each step, the algorithm randomly selects a new clump from the remaining set of clumps and adds it to the current ontology subset. The subroutine $nextClump()$ has only one input argument $\mathcal{V}_i$.

Algorithm 2 Random extensions

Subroutine *nextClump*($\mathcal{V}_i$)
Input: $\mathcal{V}_i$ an ontology subset as a set of clumps
Output: v_{i+1} a next chosen clump
$v_{i+1} \leftarrow random(\mathcal{V} \setminus \mathcal{V}_i)$
return v_{i+1}

History-based extensions, HE The HE algorithm 3 imposes an additional constraint on how the next clump is selected. At each step, it records which clumps the current clump relates to, $E(v_i)$. Here, a relation means that there is an edge connecting two clumps without regard to its direction. As the process progresses, related clumps are iteratively added to the history of seen relations if they are not selected before and if they are not already included in the history. This process is described by iterative Equation 3.3.

$$\begin{aligned} o_{i+1} &= o_i \cup o(v_{i+1}), \\ &\text{where } v_{i+1} \in H(v_i), \\ &H(v_i) = \{E(v_k)\}_{k=1}^{i} \text{ such that } H(v_i) \cap \{v_1, \ldots, v_i\} = \emptyset. \end{aligned} \tag{3.3}$$

Thus, $H(v_i)$ is a set of all faced relations up to the current clump v_i in the order of visits, excluding visited nodes. A next clump is selected randomly from the current history. The subroutine *nextClump*() has two input arguments $\mathcal{V}_i$ and $H(v_i)$.

Algorithm 3 History-based extensions

Subroutine *nextClump*($\mathcal{V}_i, H(v_i)$)
Input: $\mathcal{V}_i$ an ontology subset as a set of clumps, $H(v_i)$ a history of clump relations
Output: v_{i+1} a next chosen clump
$E_i \leftarrow relations(v_i)$
for each $v^e \in E_i$ **do**
  if $v^e \notin \mathcal{V}_i$ *and* $v^e \notin H(v_i)$ **then**
    $H(v_i) \leftarrow H(v_i) \cup v^e$
  end if
end for
$v_{i+1} \leftarrow random(H(v_i))$
return v_{i+1}

Neighbourhood-based extensions, NE The NE algorithm 4 imposes one more constraint on the way how a next clump is chosen in comparison to the HE algorithm. It restricts the maximal size of history $H(v_i)$ at any point fixing it to a particular number, m. The resulting set of clumps is called a *neighbourhood*, $H(v_i, m)$, and contains at most m *last* added clumps. As in the HE, a set of related clumps, $E(v_i)$, is determined at each step. New clumps are added to the neighbourhood following the same rules as in the HE until the size of $H(v_i)$ exceeds m. Once it exceeds m, the neighbourhood list begins to be served as a queue. In other words, new clumps are added to the beginning while older ones are removed from the end of the queue such that its size remains constant and equals m within the algorithm step. The process updates the subset according to Equation 3.4.

$$
\begin{aligned}
& o_{i+1} = o_i \cup o(v_{i+1}), \\
& \quad \text{where } v_{i+1} \in H(v_i, m), \\
& \quad H(v_i, m) = \{h_l\}_{l=|H(v_i)|-m}^{|H(v_i)|} \text{ and } h_l \in H(v_i).
\end{aligned}
\tag{3.4}
$$

A next clump is selected randomly from the current neighbourhood $H(v_i, m)$. The subroutine *nextClump*() has three input arguments $\mathcal{V}_i$, $H(v_i)$, and m.

Algorithm 4 Neighbourhood-based extensions

Subroutine *nextClump*$(\mathcal{V}_i, H(v_i), m)$
Input: $\mathcal{V}_i$ an ontology subset as a set of clumps, $H(v_i)$ a history of clump relations, m a maximal size of a neighbourhood
Output: v_{i+1} a next chosen clump
$E_i \leftarrow relations(v_i)$
for each $v^e \in E_i$ **do**
  if $v^e \notin \mathcal{V}_i$ *and* $v^e \notin H(v_i)$ **then**
    if $size(H(v_i)) < m$ **then**
      $H(v_i) \leftarrow H(v_i) \cup v^e$
    else
      $H(v_i) \leftarrow removeLast(H(v_i))$
      $H(v_i) \leftarrow addFirst(v^e, H(v_i))$
    end if
  end if
end for
$v_{i+1} \leftarrow random(H(v_i))$
return v_{i+1}

If the NE algorithm encounters the case, when there are no choices in $H(v_i, m)$ which are not taken yet, it simply tracks back to the closest relation in the history $H(v_i)$ which is not tried yet.

Thus, RE, HE, NE algorithms specialise the general algorithm of iterative extensions by how a next clump, or the extension, is selected. It is implemented by the subroutine *nextClump*() whose behaviour is determined by input parameters.

3.4.2.4 Theoretical analysis of HE and NE algorithms

In fact, the HE algorithm is a particular instance of the NE algorithm, if neighbourhood size m is assigned infinity, $m \leftarrow +\infty$. Hence, restricting m makes all difference.

Let us consider the case when there exist relations of the current node among all its relations $E(v_i)$ which are not in the history $H(v_i)$ yet. Then, it can be seen that at least one relation of the current node is always included into the neighbourhood $H(v_i, m)$ and at most m relations can be added. Whether the neighbourhood additionally contains relations of previous nodes is determined by parameter m.

Assume, that at each step all relations $E(v_i)$ of the current node v_i are not in the history $H(v_i)$ yet. Let the size of $E(v_i)$ be $E_i = |E(v_i)|$ and the probability distribution of $E_i,\ i = 1 \ldots |V|$ be $\mathbb{P}(E)$. For any ontology representation $\mathcal{G} = (\mathcal{V}, \mathcal{E})$, $\mathbb{P}(E)$ can be approximated by collecting E_i for all nodes $v_i,\ i = 1 \ldots |V|$ and then normalizing the resulting statistics. After that, the mean value of the size can be estimated as $\mu_E = \sum_{0<E<+\infty} E \cdot \mathbb{P}(E)$.

Let the size of $H(v_i)$ be $H_i = |H(v_i)|$. Then, the probability of "chaining" from the current node v_i for the HE algorithm roughly falls with increasing H_i as:

$$p_{HE}(v_i) \ \sim \ \frac{\mu_E}{H_i} \tag{3.5}$$

Now, let us estimate the relationship between the probability of "chaining" $p_{NE}(v_i)$ and the size of the history H_i for the NE algorithm. Two cases are possible:

1. The size of the relations set exceeds the neighbourhood size, $E_i > m$. This occurs with the probability $Pr[E > m]$ and, when it occurs, the "chaining" is always performed on the current node, or with the probability 1.

2. The size of the relations set does not exceed the neighbourhood size, $E_i \leq m$. This occurs with the probability $Pr[E \leq m] = \sum_{E \leq m} \mathbb{P}(E)$ and, when it occurs, the "chaining" is performed with the probability $\frac{\mu_{E \leq m}}{m}$, where $\mu_{E \leq m} = \sum_{E \leq m} E \cdot \mathbb{P}(E)$ is a mean size value for sizes less than or equal to m.

Consequently, the probability of "chaining" $p_{NE}(v_i)$ is the sum of contributions made by each case because either case 1 or case 2 can take place but never both. Thus, the probability of "chaining" for NE is derived as follows:

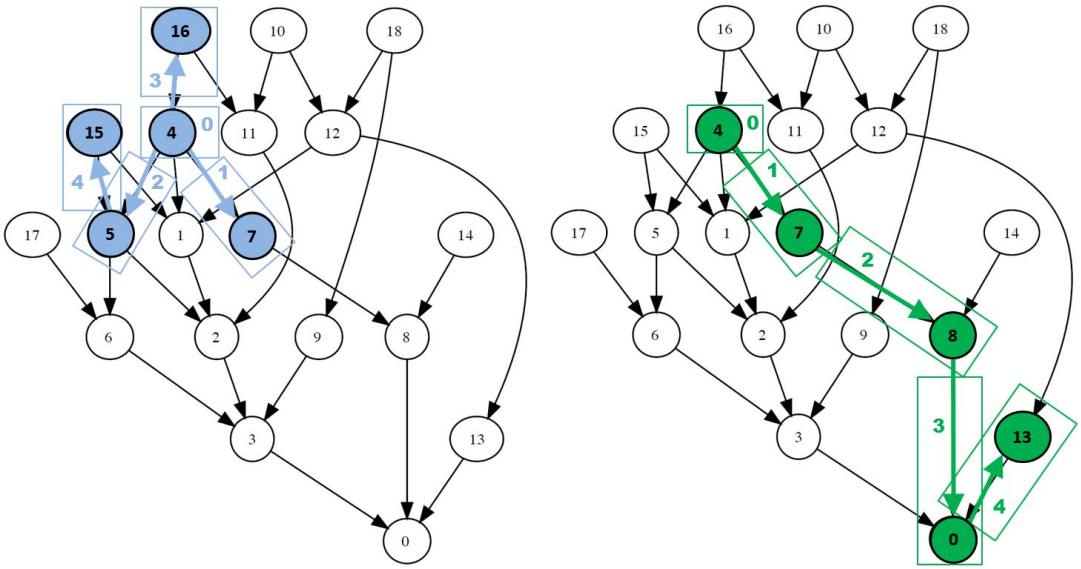
$$\begin{aligned} p_{NE}(v_i) &\sim Pr[E > m] \cdot 1 + Pr[E \leq m] \cdot \frac{\mu_{E \leq m}}{m} \\ &= 1 - Pr[E \leq m] + Pr[E \leq m] \cdot \frac{\mu_{E \leq m}}{m} \\ &= 1 - (1 - \frac{\mu_{E \leq m}}{m}) \cdot Pr[E \leq m]. \end{aligned} \tag{3.6}$$

As it is seen from the derived dependency, Equation 3.6, in contrast to the HE algorithm, the probability of "chaining" $p_{NE}(v_i)$ does not depend on the history size H_i for the NE algorithm and, hence, remains constant as H_i grows.

Thus, our theoretical analysis justifies that the HE algorithm less frequently performs "chaining" than the NE algorithm and the probability to "chain" for the HE algorithm decreases inversely proportional to the increasing size. In contrast, the probability of "chaining" for the NE algorithm does not depend on the history size. As it follows from Equation 3.6, the degree of "chaining" is controlled by parameter m such that the probability to "chain" decreases as the neighbourhood size m increases. Hence, the probability of "chaining" is maximal when the neighbourhood size m is minimal, or $m = 1$.

This explains why the NE algorithm forces building long "chains" while the HE algorithm exhibits "branching". Figure 3.1 illustrates possible behaviour of the HE and NE.

Although the obtained in this section derivations and conclusions are justified by experiments, it should be pointed that probability estimates $p_{HE}(v_i)$ and $p_{NE}(v_i)$ are more precise for some ontologies and less precise for others. The reason is that for some ontologies the assumption that at each step all relations $E(v_i)$ of the current node v_i are not in the history $H(v_i)$ yet can be too strong because the growing history size decreases the number of possible choices. The latter results in declining mean values μ_E and $\mu_{E \leq m}$ with growing H_i. As a result, $p_{HE}(v_i)$ can decrease faster than it is estimated by Equation 3.5 and $p_{NE}(v_i)$ can



(a) HE "branching" (b) NE "chaining"

Figure 3.1: HE and NE behaviour on the AD of Koala, $m = 1$ for NE

start to decline at some point. However, these are rather unusual cases due to the fact that most ontologies produce representations with sufficiently large number of nodes.

3.4.3 Extrapolation by regression analysis using samples

Growing-based sampling produces a set of samples with performance measurements. These data can be used to make predictions for the whole ontology via regression analysis.

Regression analysis estimates the relationship between a dependent variable and independent variables, Section 2.4.2. Independent variables are features that form a feature vector. The same features can be extracted from the whole ontology and its subsets, Table 3.1. A dependent variable is the computation time.

Let $o_j = o_j(\mathcal{O})$ be j-th sample extracted from an ontology $\mathcal{O}$. Then, given a suite of samples $\{o_j\}_{j=1}^{n}$ for an ontology $\mathcal{O}$, features $\mathbf{x}_j = \mathbf{x}(o_j)$ are extracted from each sample and a time measurement $y_{jk} = \mathcal{CT}(o_j, \mathcal{R}_k), j = 1 \dots n, k = 1 \dots M$ is associated with the feature vector $\mathbf{x}_j$. This creates a data set for a given ontology:

$$[\mathbf{X}(\mathcal{O}) \mid \mathbf{Y}_1^T(\mathcal{O}) \dots \mathbf{Y}_M^T(\mathcal{O})], \tag{3.7}$$

where $\mathbf{Y}_k(\mathcal{O}) = (y_{1k}, \dots, y_{nk})$ are performance measurements of a reasoner

$\mathcal{R}_k$ on samples $\{o_j\}_{j=1}^{n}$.

As it is noted before, some assumptions about regression function f need to be made in order to predict a dependent variable beyond the interval of observations, or extrapolate $y = f(\mathbf{x}, \beta)$. Then, parameters β can be estimated by minimizing the sum of residuals and, hence, fitting function f to the data points, see Section 2.4.2.2. Once the regression function is fitted, it is able to make a performance prediction given the ontology profile.

3.5 Differences between global and local approaches

A major difference between global and local approaches is that global approaches rely only on an ontology profile and never attempt to have a look inside the ontology. In contrast, local approaches explore parts, ideally small, of an ontology and use gathered data for the performance prediction on the full ontology.

Global approaches train a supervised machine learning model using ontology profiles and exploit this model to make performance predictions. In contrast, local approaches do not use ontology profiles for building a model but use a suite of samples extracted from an ontology. Hence, local approaches compute multiple feature vectors for each ontology while global ones compute just one.

In order to train a model, global approaches require a prepared training set which includes multiple ontologies. The quality of a trained model strongly depends on a training set, hence, it should be combined cautiously. The set has to cover possible ranges of features where testing examples can appear. Ideally, the training set should be unbiased concerning both ontology sizes and computation times. It has to include sufficient number of examples to provide good generalization for unseen examples.

In contrast, local approaches do not require a training data set of multiple ontologies at all. Only a testing ontology matters and a regression function is fitted to this ontology solely using its sampling data. Hence, there is no general model except the assumed form of a regression function which parameters are estimated for each ontology individually.

Thus, global approaches gather the data from many ontologies and interpolate for unseen ontologies while local approaches collect the data only from a part of a testing ontology and extrapolate to its full size.

Chapter 4

Data and Tools

4.1 Ontology corpus

We have chosen a collection of state-of-the-art ontologies from the NCBO BioPortal[1], NCI Thesaurus[2] and TONES Ontology Repository[3]. These are freely accessible and widely used resources among ontology researchers.

NCBO BioPortal contains 343 ontologies with 5,980,874 terms in total. The ontologies vary in computation time from easy to hard ones [9], [8]. The NCI Thesaurus repository encodes information on cancers and related diseases. It mainly consists of computationally hard ontologies. The TONES repository contains ontologies for testing purposes. Most ontologies in repositories NCBO BioPortal and TONES are easy ones.

We select the ontology corpus from the above repositories such that it is as less biased to easy ontologies as possible because a significantly biased corpus can lead to wrong conclusions.

Our corpus consists of 121 ontologies selected from NCBO BioPortal, NCI Thesaurus, and TONES repositories. These ontologies are listed in Appendix 6.4. We measure the computation time of each reasoner on each of these ontologies. We run a reasoner multiple times on each ontology and record the average execution time for this ontology. The timeout of 1000 seconds is used for all reasoners such that a reasoner is terminated if its job is not finished within 1000 seconds. The performance statistics is shown in Table 4.1.

[1] `http://bioportal.bioontology.org/`
[2] `http://ncit.nci.nih.gov/`
[3] `http://owl.cs.manchester.ac.uk/repository/`

Reasoner	**(0s,1s)**	**[1s,10s)**	**[10s,100s)**	**[100s,1000s)**	**[1000s,+∞)**
JFact	50	17	22	26	6
Hermit	54	28	13	25	1
Pellet	60	25	10	24	2

Table 4.1: Performance statistics on the ontology corpus

Table 4.1 shows that the corpus still has some bias to easy ontologies with $\mathcal{CT}(\mathcal{O},\mathcal{R}) \in (0s, 1s)$: it contains 41% of easy ontologies for JFact, 45% for Hermit, 50% for Pellet. However, this bias is acceptable concerning that the corpus contains half or more not easy ontologies for each reasoner.

To compare, the ontology corpus used by *Kang et al.* is significantly more biased to easy ontologies with $\mathcal{CT}(\mathcal{O},\mathcal{R}) \in [0.01s, 1s)$: 69% for Fact++, 72% for Hermit, 66% for Pellet.

4.2 Data handling

The data are extracted from ontologies as the sequence of feature vectors with computation time measurements. A specific XML format is developed in order to specify the structure for the gathered data for further analysis and processing. This format is used for both ontology subsets and full ontologies. A feature vector of a sample with a performance measurement in the XML format is shown in Example 4.2.1.

Example 4.2.1.

```
<ontology id="http://purl.obolibrary.org/obo/vo.owl">
  <vector s00="2202.0" s01="523.0" s02="154.0" s03="616.0" s04="36.0
      " s05="1.0" s06="1872.0" s07="0.0" s08="0.0" s09="0.0" s10="0.0
      " s11="602.0" s12="79.0" s13="0.0" s14="0.0" s15="0.0" s16="0.0
      " s17="0.0" s18="667.0" s19="206.0" s20="93.0" s21="0.0" s22="
      0.0" s23="4.0" s24="0.0" s25="2.0" s26="5.0" s27="0.0" s28="0.0
      " s29="0.0" s30="4.0" s31="3.0" s32="10.0" s33="0.0" s34="0.0"
      s35="0.0" s36="0.0" s37="0.0" s38="0.0" s39="3.0" s40="0.0" s41
      ="44.0" s42="2.0" s43="154.0" s44="0.0" s45="0.0" s46="0.0" s47
      ="0.0" s48="0.0" s49="0.0" s50="0.0" s51="674.0" />
  <times hermit="3.4788223" />
</ontology>
```

4.3 Development tools

Eclipse Eclipse[4] is a popular Java integrated development environment, IDE. It has a customizable plug-in system that makes it extensible for various purposes. Major implementations are carried out in this environment using the Java programming language. We use version 4.2.1.

OWL API The OWL API[5] is "a Java API and reference implementation for creating, manipulating and serialising OWL Ontologies"[6]. Ontology loading from a source file, ontology processing, ontology feature extraction are implemented using the OWL API, version 3.4.3.

OWL Reasoners This project is focused on exploring behaviour of widely used reasoners Hermit [22], Pellet [27], and JFact, a Java version of FaCT++ [29]. All these reasoners are integrated into OWL API and can be invoked through it as far as appropriate implementation libraries are included. We use Hermit 1.3.8, JFact 1.0.0, Pellet 2.3.0.

Protégé Protégé is "a free, open source ontology editor and knowledge-base framework"[7]. It is a Java-based editor customizable and extensible via plug-ins. It is used to visualise ontologies, invoke reasoners, make changes to ontologies and investigate outcomes. We use version 4.1.0.

4.4 Hardware characteristics

All experiments are executed on the same machine which is a laptop. The operating system is Windows 7, 64-bit. CPU is Intel Core i3-2330M, 2 cores, 2.2 GHz. RAM is DDR2, 4 GB. All algorithms are implemented in Java, JDK 1.7.

[4] http://www.eclipse.org/org/
[5] http://protege.stanford.edu/plugins/owl/api/
[6] http://owlapi.sourceforge.net/
[7] http://protege.stanford.edu/

Chapter 5

Performance Prediction Experiments

5.1 Performance measurement techniques

Performance prediction experiments require precise time measurements. However, the precision of gathered measurements is affected by two factors. Firstly, ontology subsets of small size can be processed in short time by a reasoner which should not be overestimated. Secondly, all reasoners are implemented as threads. These implementations are forced by OWL API where a reasoner is viewed as an executed Java thread and it completes all its job inside this thread. Although it is a reasonable way to parallelise running of multiple reasoners, it bears difficulties for measuring computation times by standard ways because threads are generally controlled by the Java Virtual Machine, JVM. Hence, some threads may be started before others finish their job.

In order to solve these problems, we use the *ThreadMXBean*[1] interface. Its implementation allows to measure the CPU time of a current thread by invoking the method *getCurrentThreadCpuTime()*. It has nanoseconds precision which is sufficient for measure minor times if a reasoner processes a small set of axioms.

Concerning the performance variability phenomena, some ontology subsets, not necessarily large, may be hard for a reasoner to handle and may take too long to process. To fasten the process of data gathering, we terminate the reasoner execution if it exceeds some *timeout*.

[1] `http://docs.oracle.com/javase/6/docs/api/java/lang/management/ThreadMXBean.html`

5.2 Feature analysis

5.2.1 Feature ranking

As discussed in Section 2.4.3, reducing the dimensionality of a feature vector can bear both computation and accuracy benefits for prediction methods. This can be achieved in two different ways. The first way is to rank the existing features by their relevance and then just eliminate weakly relevant ones.

First, let us investigate feature ranking. The features can be ranked by their correlation with the computation time. The correlation can be measured by Pearson's correlation coefficient, Section 2.4.3. However, the raw value of the coefficient r estimates both correlation, positive values of r, and anti-correlation, negative values of r. Then, if we rank features by r in decreasing order, features having zero correlation will be ranked higher than anti-correlated features. One can notice, however, that negating an anti-correlated feature gives a positively correlated feature. Therefore, correlated and anti-correlated features are equal in ranking. A simple way to ensure this equality is using the absolute value $|r|$ instead of the raw value r.

Features should be ranked for each reasoner separately. Figure 5.1 shows absolute values $|r|$ for JFact, Hermit, Pellet. Feature identification numbers coincide with the identification numbers in Table 3.1.

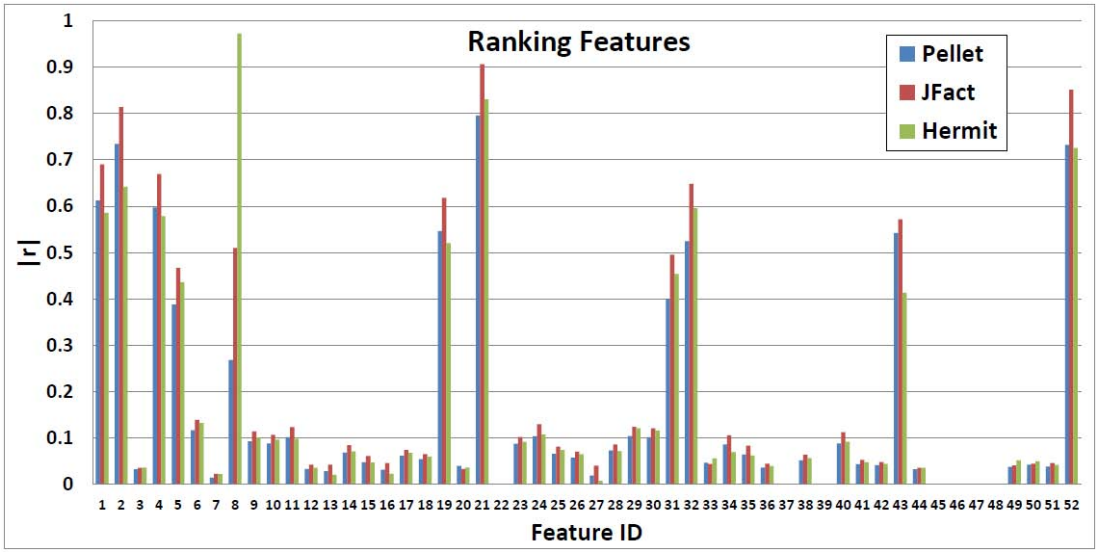


Figure 5.1: Absolute coefficient values of the features correlation with the computation time for JFact, Hermit, Pellet

As Figure 5.1 shows, mostly the same group of features appears to be ranked higher than others for all three reasoners. However, there do exist differences. A summary of feature ranking with 10 top-ranked features[2] for each of three reasoners is given in Table 5.1.

Rank	**JFact**	**Hermit**	**Pellet**
1	#EqClasAx	#ObjAllValFrom	#EqClasAx
2	parsingDepth	#EqClasAx	#ObjProNOcc
3	#ObjProNOcc	parsingDepth	parsingDepth
4	#ClassNOcc	#ObjProNOcc	#ClassNOcc
5	#ClassN	#ObjProRangeAx	#ClassN
6	#ObjProRangeAx	#ClassNOcc	#SubClassOfAx
7	#SubClassOfAx	#ClassN	#ObjProAssertionAx
8	#ObjProAssertionAx	#SubClassOfAx	#ObjProRangeAx
9	#ObjAllValFrom	#ObjProDomainAx	#ObjProDomainAx
10	#ObjProDomainAx	#ObjProN	#ObjProN

Table 5.1: 10 top-ranked features for JFact, Hermit, Pellet

As it is seen from Table 5.1, overall, *#EquivalentClassesAxioms*, *parsingDepth*, *#ObjectPropertyNameOccurrences* have the greatest correlation values with the computation time. In contrast, *#ObjectAllValuesFrom* is ranked highest for Hermit with $|r| \approx 0.97$. It means that this feature has almost perfect correlation with the computation time for Hermit. Unlike Hermit, JFact and Pellet have *#ObjectAllValuesFrom* much lower in the ranking table. For Pellet, this feature is beyond top-10 ranks at all.

Another difference of Hermit is that it has no *#ObjectPropertyAssertionAxioms* in top 10 ranks unlike JFact and Pellet. In addition, *#ClassNameOccurrences* and *#ClassNames* are ranked lower for Hermit.

Feature ranking shows that the computation time correlates significantly stronger with features from nearly the same and small group of 11 out of 52 features while all other features have small values $|r| < 0.14$ and can possibly be removed. It should be pointed out though that correlation between two variables does not necessarily imply causation[3]. There may be other factors affecting the computation time, independent from the feature which merely appears at the same moments.

[2]Full feature names are shorten for simplicity and easier comparisons

[3]`http://en.wikipedia.org/wiki/Correlation_does_not_imply_causation`

5.2.2 Correlation between features

The second possibility to reduce the dimensionality of a feature vector is projecting features to a lower dimensional space that means extracting new features from existing ones. One of methods to achieve that is PCA described in Section 2.4.3. We do not use any information about the computation time here but only feature values, in contrast to feature ranking. Thus, PCA deals only with the set of feature vectors and ignores computation times.

In order to investigate the amount of correlation between features, we apply PCA to the data set constructed form the ontology corpus. First, let us explore component variances, Figure 5.2.

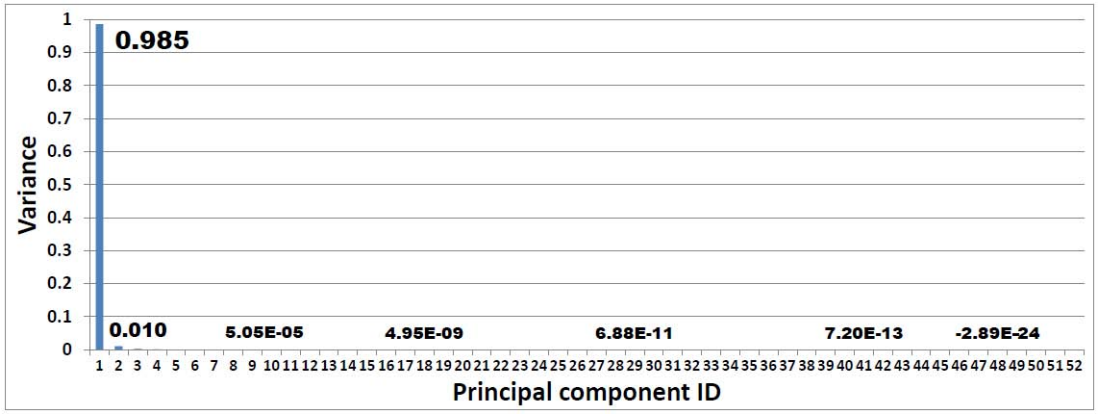


Figure 5.2: Normalized principal component variances

Surprisingly, the first principal component accounts for $\approx 98.5\%$ of all variance, Figure 5.2. The second component has a variance $\approx 1\%$ of the total value. The variances of other components are insignificant. It means that we can replace 52 original features by just *one* feature and preserve $\approx 98.5\%$ of variance. The latter implies that the original features are *highly intercorrelated*.

In order to explore ontology spreading in the projection space, we can visualise them in two dimensional space (z_1, z_2) using PC1 and PC2 projections, see Equation 2.15. Following this way, we preserve $\approx 99.5\%$ of total variance. The results are shown on Figure 5.3.

As Figure 5.3 illustrates, all ontologies are mainly spread along the first coordinate with smaller deviations along the second coordinate. In addition, we have achieved clustering of the ontology corpus, see Appendix 6.4. Figure 5.3 shows that NCI releases are assembled in clearly separable clusters.

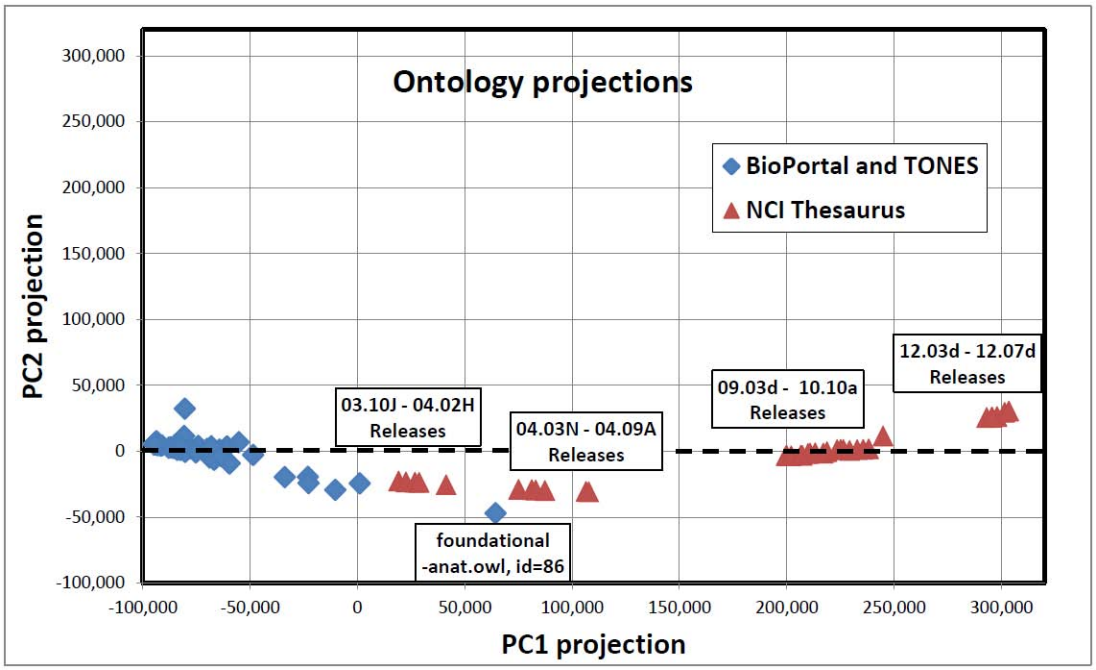


Figure 5.3: Projecting ontologies of the corpus into two dimensional space using PCA

Observing such an efficient representation of variance by a single feature, a reasonable question is how this feature is related to ontology features. As it follows from Equation 2.15, the value of the PC1 projection is given by:

$$z_1 = \sum_{j=1}^{d} (x_j - \bar{x}_j) u_{1j} \tag{5.1}$$

where $u_{1j},\ j = 1 \ldots d$ are PC1 projection coefficients which measure contributions of original features to the PC1 projection. This shows the relationship between the original features and the new feature z_1 obtained through the PC1 projection. The results are illustrated on Figure 5.4.

Figure 5.4 shows that only 7 out of 52 original features contribute significantly to the PC1 projection. These features are sorted in descending order by their contribution value and listed in Table 5.2.

As it is seen from Table 5.2, *#ClassNameOccurrences* is the most contributing feature to the PC1 projection with the contribution value significantly larger than others. Overall, all these features are related to the ontology size, except the fifth feature *#ClassNames*, which is a vocabulary feature. Hence, this shows that the PC1 projection is mostly affected by ontology size.

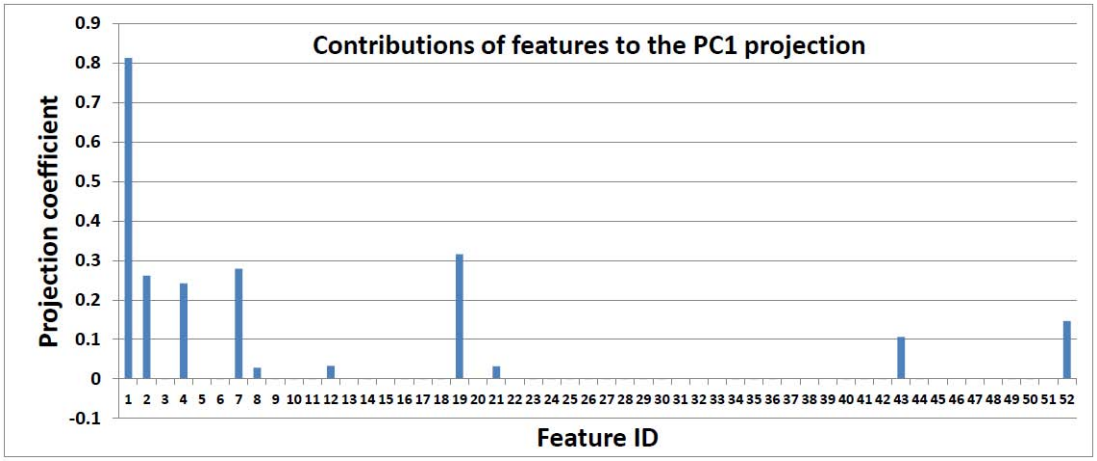


Figure 5.4: PC1 coefficients as contributions of original features to the PC1 projection

Order	Feature	Contribution
1	#ClassNameOccurrences	0.81
2	#SubClassOfAxioms	0.32
3	#ObjectSomeValuesFrom	0.28
4	#ObjectPropertyNameOccurrences	0.26
5	#ClassNames	0.24
6	parsingDepth	0.15
7	#ObjectPropertyAssertionAxioms	0.11

Table 5.2: Contributing features with coefficient > 0.1 for the PC1 projection

Hypothesis. The PC1 projection correlates with ontology size.

Test. In order to test the hypothesis, we estimate the correlation between the PC1 projection and the ontology size by calculating the absolute value of Pearson's correlation coefficient $|r|$, as it is done for feature ranking. In addition, we calculate the correlation for all other components to ensure that the PC1 projection has a greatest value of $|r|$ and to find out how big it is in comparison to others. The results of this test are shown on Figure 5.5.

Figure 5.5 shows that the PC1 projection has the absolute value $|r| \approx 90\%$ while other components are marked with the values $|r| < 9\%$. The latter justifies that *the PC1 and only the PC1 projection* is highly correlated with the ontology size and, hence, justifies the *Hypothesis*. As a consequence, the original set of 52 ontology features can be replaced by just one feature, the ontology size, with minor loss of information. The loss of information is minimal if 52 features are

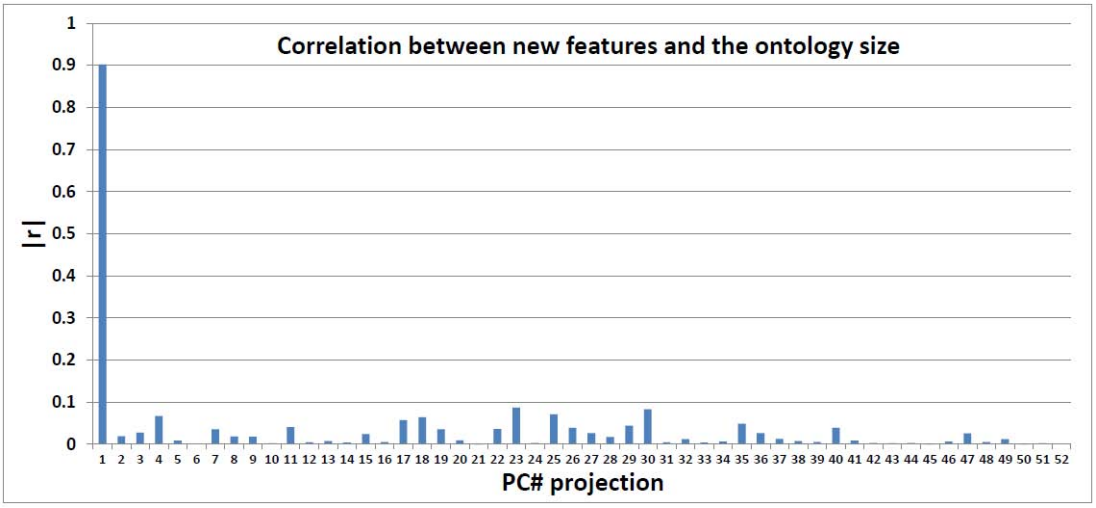


Figure 5.5: Absolute values of the correlation between each PC projection and the ontology size

replaced by the PC1 projection. However, the ontology size has its advantages: it is easy obtainable and clearly explainable.

Thus, we have established a connection between the PC1 projection and the ontology size. Despite the high correlation between them, however, one can argue that uncorrelated parts might cause placing a hard ontology close to easy ones and vice versa because there is no relation to the computation time. Hence, this might lead to different performance clusters across the same ontologies. In order to investigate this, we link the PC1 projection and the size to computation times of ontologies in the corpus. First, we sort all corpus ontologies by their size and display their computation times on Figure 5.6.

Now, let us sort ontologies by the PC1 projection, see Figure 5.7. Comparing Figure 5.6 and Figure 5.7, we are able to see that performance curves are similar in both cases for each reasoner which means the same ontologies occupy the same positions in the ordering.

We have shown above, that 52 ontology features can be replaced by the ontology size alone, concerning the information content. In addition, Figures 5.6, 5.7 show that the replacement seems reasonable, concerning the performance, because, at least, we can be sure that the ontology ordering will not be corrupted severely. Hence, we can hypothesize that extracting 52 ontology features for the performance prediction, as the common global approach does, makes little sense because they can be replaced by the ontology size alone. Nevertheless, we cannot

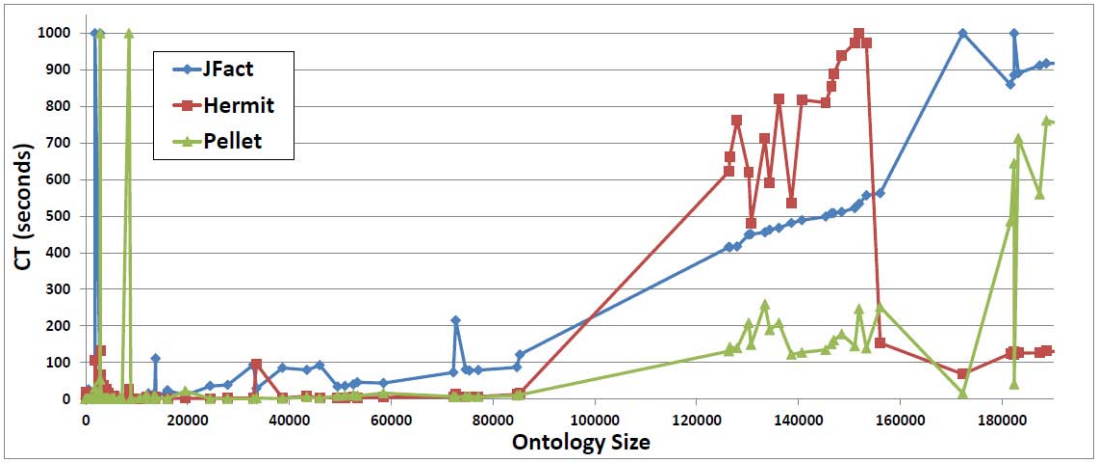


Figure 5.6: Computation times of JFact, Hermit, Pellet on the corpus ontologies sorted by their size

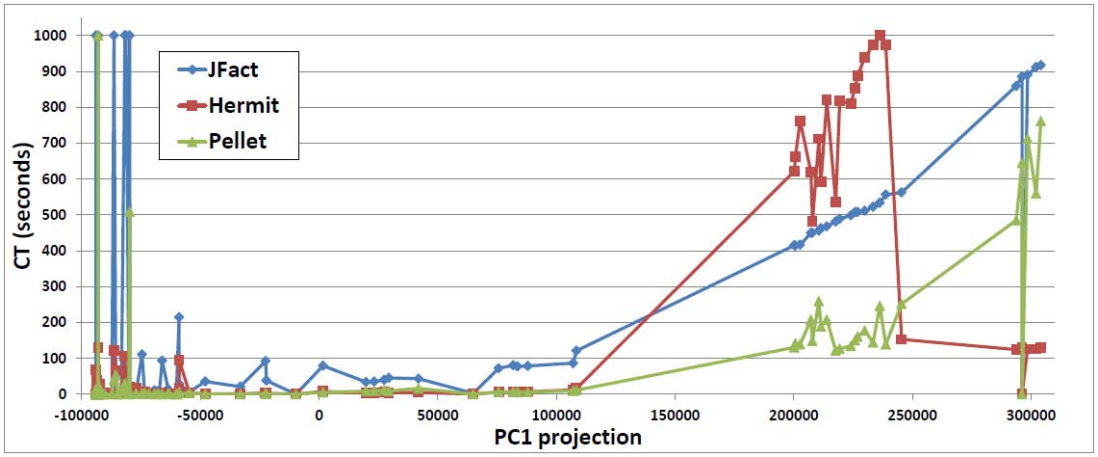


Figure 5.7: Computation times of JFact, Hermit, Pellet on the corpus ontologies sorted by their PC1 projection

be sure until we compare performance prediction results of the model, which uses 52 features, with the same model, which uses the size feature alone.

In addition, Figures 5.6, 5.7 show that neither the PC1 projection nor the size are perfect "global" performance indicators because there are massive spikes at the beginning of their ranges. Hence, the global approach will inevitably make errors trying to predict the performance using either the PC1 projection or size values from this interval because it merely builds an approximation of observed computation times, a machine learning model.

5.3 Performance predictor evaluation

There are multiple ways to assess a model by its predictions. Here, we consider these ways in context of the performance predictions.

The first possibility is based on discretization of the computation time to finite number of intervals, or bins, see Section 2.4.1.3. Then, given a testing set of ontologies $\mathcal{O}_i,\ i = 1 \dots N$, a model predicts a bin $\hat{b}(\mathcal{O}_i, \mathcal{R})$ for each ontology from the set. If the model predicts a wrong bin for an ontology, or $\hat{b}(\mathcal{O}_i, \mathcal{R}) \neq b(\mathcal{O}_i, \mathcal{R})$, it is counted as an error. Once the model is tested on all ontologies, an average error is calculated which we call the *average classification error*, ACE:

$$ACE(\mathcal{R}) = 1 - \frac{1}{N}\sum_{i=1}^{N} \delta(\hat{b}(\mathcal{O}_i, \mathcal{R}), b(\mathcal{O}_i, \mathcal{R})). \tag{5.2}$$

ACE is used by *Kang et al.* to evaluate a model. However, this measure has its disadvantages. Firstly, it is affected by binning, see Example 3.3.2, that may cause incorrect conclusions while assessing a model. Secondly, it counts all wrong predictions equally. For example, to predict bin 5 and bin 2 for true bin 1 are equally wrong regardless of the huge difference in computation times.

One more way to assess a model is to measure the raw differences of true computation time and the predicted time for each testing ontology. This measure avoids ontology binning whose helpfulness for evaluating performance predictors is an open question. The *mean squared error*, MSE, is a common way to measure the differences between values. In application to the performance prediction, it can be written as follows:

$$MSE(\mathcal{R}) = \frac{1}{N}\sum_{i=1}^{N} (\hat{y}(\mathcal{O}_i, \mathcal{R}) - \mathcal{CT}(\mathcal{O}_i, \mathcal{R}))^2, \tag{5.3}$$

where $\hat{y}(\mathcal{O}_i, \mathcal{R}),\ i = 1 \dots N$ is the predicted computation time of a reasoner $\mathcal{R}$ on an ontology $\mathcal{O}_i$, $\mathcal{CT}(\mathcal{O}_i, \mathcal{R})$ is the true computation time.

However, the MSE has its own disadvantage because the hardest ontologies, which are rare occasions, can contribute more to the MSE than all remaining ontologies together. In this case, the MSE measures the model fitness to hard ontologies but not to the data set.

Thus, both the ACE and MSE have disadvantages. Therefore, we use both of them for model evaluation. Both ACE and MSE are suitable for assessing both

global and local approaches. However, a global approach of training a machine learning model on ontology profiles has its specificity.

A global approach makes use of a supervised machine learning model that is trained on a training set and tested on a testing set. However, only one set of examples is usually available. Hence, we have to separate this set into two parts, a training set and a testing set. How to select examples for training and testing is an open question and depends on a data set. Related issues are discussed in [20].

One of common approaches for evaluating a supervised machine learning model using the same set of examples is *cross-validation*, CV [20]. It separates the data set into n complementary parts, or folds, of equal sizes. Then, it leaves one fold for validation and uses remaining $n-1$ folds for training. It does so for each fold and calculate the fold error each time. Either ACE or MSE can be used for a fold depending on whether bins or exact values are used as predictions. Finally, the average value of the fold error, called *cross-validation error*, CVE, is computed.

In order to achieve the best accuracy with a model, we need to select its *optimal parameters*. The CVE is used to select the optimal parameter k for the k-NN model such that the best model has the lowest CVE.

However, the CVE is an optimistic estimate of the future generalization error. In other words, the model is assessed better than it is. Therefore, first, one of n folds is separated for final testing and then the CV is performed on remaining $n-1$ folds. Once the best model over these folds is identified, it is tested on the separated fold and its error on this fold, either ACE or MSE, is recorded. In order to eliminate the dependency on fold choices, the whole procedure is repeated many times shuffling the examples each time. Finally, the errors on a testing fold are averaged. We call this average error a *generalization error*, GE. In addition, the standard deviation of the error on a testing fold, GD, is calculated.

The GE and GD quantify utility of a supervised machine learning model on a given data set. The first criterion assesses the model fitness and the second criterion assesses the model stability. We use the GE and GD for evaluating a global approach on the ontology corpus.

It suffices to evaluate a local approach by the ACE and MSE because it does not use the ontology corpus, excluding a testing ontology, and never observes ontology profiles and true computation times before making a prediction.

5.4 Global approaches

5.4.1 Predictions by machine learning using profiles

As a global approach, machine learning classification uses a set of ontology profiles to train a model. We combine a training set from the ontology corpus as it is described in Section 3.3.2 and, hence, make use of all 52 features from Table 3.1. In addition to bins specified by *Kang et al.*, see Section 2.4.1.3, we add one more bin E in order to distinguish hardest ontologies with the computation time longer than 1000 seconds and include all ontologies with $\mathcal{CT}(\mathcal{O}, \mathcal{R}) < 1s$ in bin $A \in (0s, 1s)$. Thus, the ontology corpus is classified into 5 bins by the computation time of its ontologies as follows: $A \in (0s, 1s)$, $B \in [1s, 10s)$, $C \in [10s, 100s)$, $D \in [100s, 1000s)$, $E \in [1000s, +\infty)$.

We use the k-NN classifier as a model, see Section 2.4.1.2. The choice of a right machine learning model is not critical here, because we test overall suitability of supervised machine learning models for the performance predictions and plan to compare these results with local approaches. The k-NN classifier allows to obtain sufficiently good predictions for many training sets. An advantage of the k-NN is its simplicity that makes analysis of its predictions easy and transparent. In addition, it can be used to predict both bins and exact computation times.

The k-NN algorithm has just one input parameter k. In order to obtain the best prediction results, the optimal value k should be estimated for each reasoner. We apply cross-validation with 10 folds, see Section 5.3, for selecting optimal k and evaluating the model.

The results of the performance prediction for each reasoner by the global approach with the estimated optimal parameter, when binning is used, are shown in Table 5.3. The results without binning are shown in Table 5.4.

Reasoner	**Optimal k**	**GE**	**GD**
JFact	5	0.278	0.127
Hermit	9	0.258	0.128
Pellet	10	0.287	0.117

Table 5.3: Performance prediction by the k-NN with the optimal parameter and *with* binning

There is a single optimal value k for each reasoner, see Tables 5.3, 5.4, because both evaluating criteria have their minimal values.

Reasoner	**Optimal** k	**GE** (seconds)	**GD** (seconds)
JFact	1	161	84
Hermit	1	191	53
Pellet	10	123	96

Table 5.4: Performance prediction by the k-NN with the optimal parameter and *without* binning

Table 5.3 shows performance prediction results *with* ontology binning. Hence, the GE is the average ACE over testing folds, see Section 5.3. The model for JFact has $GE \approx 27.8\%$ that implies the average accuracy of $\approx 72.2\%$. The k-NN shows $GE \approx 25.8\%$ for Hermit and $GE \approx 28.7\%$ for Pellet. It should be noticed that all three models with their optimal parameters have significant values of the deviation: $12.7\%, 12.8\%, 11.7\%$ correspondingly.

Table 5.4 shows performance prediction results *without* ontology binning. Hence, the GE is the average MSE in seconds over testing folds, see Section 5.3. The model for JFact, Hermit, and Pellet shows $GE \approx 161s$, $GE \approx 191s$, and $GE \approx 123s$ accordingly. Similarly to Table 5.3, all models have significant values of the deviation that means the models are unstable.

5.4.2 Predictions by machine learning using the ontology size

According to findings in Section 5.2.2, all original 52 features can be replaced by a single feature, which is the ontology size, with minor loss of information. On the other hand, considering the curse of dimensionality, see Section 2.4.3, reducing the dimensionality of a feature vector can lead to the model improvement. Thus, replacing all features by one, we can retain the most of information and increase the model precision. Therefore, let us now investigate the performance prediction using the ontology size alone instead of 52 original features.

Hypothesis. The performance prediction using ontology size alone is comparable to or slightly better than the performance prediction using all 52 features.

Test. We use exactly the same procedure as above to evaluate the performance prediction for each reasoner. However, an ontology profile is not a vector but a single value now. We investigate both cases: with and without ontology

binning. The results are given by Tables 5.5, 5.6 respectively.

Reasoner	**Optimal** k	**GE**	**GD**
JFact	5	0.241	0.122
Hermit	2	0.257	0.125
Pellet	3	0.215	0.107

Table 5.5: Performance predictions using the ontology size alone *with* binning

Reasoner	**Optimal** k	**GE** (seconds)	**GD** (seconds)
JFact	6	136	78
Hermit	2	171	53
Pellet	4	144	98

Table 5.6: Performance predictions using the ontology size alone *without* binning

Tables 5.5, 5.6 show that performance prediction results are better in both cases for each reasoner in comparison to Tables 5.3, 5.4, except Pellet without binning whose results are slightly worse. In contrast, the results for Pellet with binning are significantly better with the size alone, 21.5% against 28.7%. Hence, the results justify the *Hypothesis*. An explanation of this outcome is that the loss of useful information due to replacing 52 original features by a single feature is outweighed by removing useless information due to dimensionality reduction.

Thus, we have achieved the model improvement in both computational efficiency and prediction ability.

5.5 Local approaches

5.5.1 Tests of sampling algorithms

The proposed RE, HE, NE algorithms are different ways of constructing ontology samples, see Section 3.4.2.3. Although the process is iterative and similar in all three cases, the method of assembling clumps together differs.

First, let us explore how a reasoner reacts on samples produced by each algorithm. In other words, we target to find out which algorithm per reasoner produces hardest samples given a size restriction, without any relation to the prediction accuracy for now.

We assume that the AD is a suitable ontology representation. Then, clumps are atoms. We apply the NE algorithm with the minimal neighbourhood size $m = 1$ to ensure that the probability of "chaining" is maximal, see Section 3.4.2.4.

A sampling algorithm builds 20 samples within each run. We restrict the size of a maximal sample to be not more 10% of the full ontology. Once the size limit is reached, the algorithm terminates. It is important to note that, if the ontology is hard and large, we extract randomly only a part of it sufficient to retrieve samples of maximal sizes. In other words, we use nearly 11% of the full ontology to be able to obtain samples of 10% sizes and ignore remaining 89% of the full ontology due to computational considerations.

Since the algorithms are stochastic by their nature, we run each algorithm for each ontology multiple times and collect all retrieved samples in a single suite per ontology. The computation time of each reasoner $\mathcal{CT}(o_j, \mathcal{R})$ is measured for each sample. Time measurements of each suite of samples are aggregated according to their sizes and averaged across all ontologies in the corpus.

The results of algorithm comparisons for JFact, Hermit, Pellet are shown on Figures 5.8, 5.9, 5.10 respectively. The relative sample size of 0.1 means 10% of the full ontology.

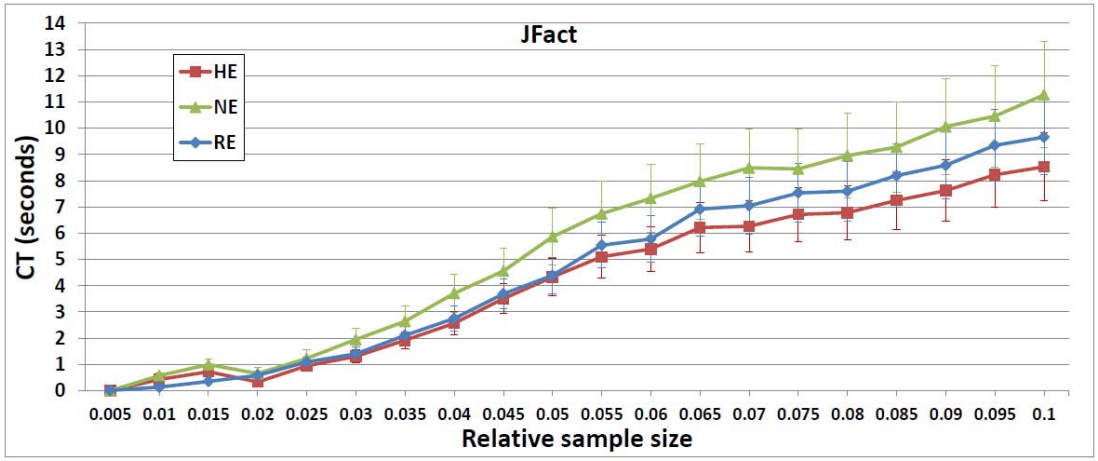


Figure 5.8: Average computation time of JFact on RE, HE, NE samples

Figures 5.8, 5.9, 5.10 show that the NE algorithm produces the hardest samples on average for each reasoner. The HE samples are the easiest ones and the RE samples are in the middle.

Now, let us recall the conclusions of our theoretical analysis of the HE and

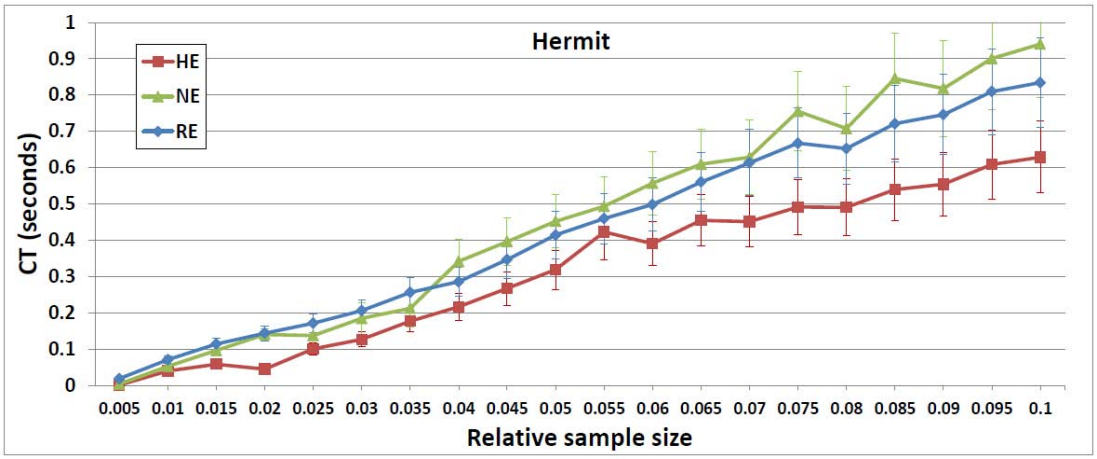


Figure 5.9: Average computation time of Hermit on RE, HE, NE samples

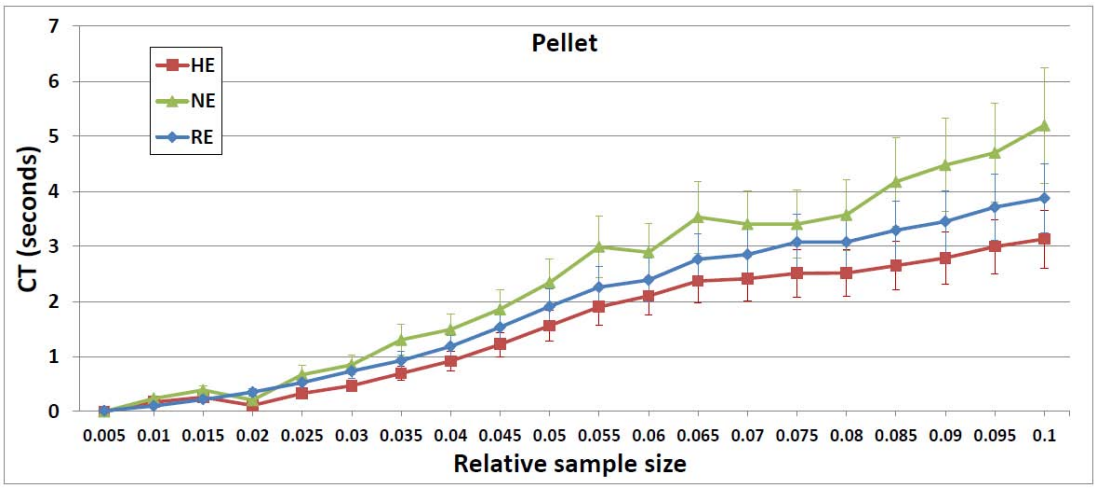


Figure 5.10: Average computation time of Pellet on RE, HE, NE samples

NE algorithms, see Section 3.4.2.4. We have shown by estimating the probabilities of "chaining" $p_{HE}(v_i)$ and $p_{NE}(v_i)$ that the HE algorithm exhibits "branching" by spreading in multiple directions in the ontology graph while the NE algorithm forces "chaining" by building long chains of visited nodes. Concerning Figures 5.8, 5.9, 5.10, we can conclude that "chaining" produces significantly harder samples than "branching" for all reasoners. It should be noticed that the RE samples appear between the NE and HE samples. Consequently, "branching" builds easier samples than random selection.

5.5.2 Predictions by regression analysis using samples

The results of Section 5.5.1 show which algorithm produces hardest samples per reasoner and which one produces the easiest. However, there is no evidence which samples, hard, middle or easy ones, are the most suitable for the performance prediction. We target to find that out. We consider samples as the most suitable if they deliver the best prediction accuracy.

Considering our findings in Section 5.2.2, we record only sample size instead of 52 features. Hence, there is only one independent variable, sample size x. We assume that $f(x)$ is a polynomial of degree L at most 2. This is supported by the performance observations in [9] with the note that ontology subsets are not random now and have much smaller sizes.

We exploit the same samples as before with the size limit 10%. Sample sizes and time measurements are used to fit the regression function $f(x)$ to the sampling data by minimizing the sum of residuals, see Section 2.4.2.2, for each ontology individually.

We evaluate each algorithm by the ACE and MSE of predictions based on its samples across all ontologies in the corpus. The results of performance predictions for JFact, Hermit, Pellet using samples are shown in Tables 5.7, 5.8, 5.9.

Algorithm	L=1		L=2	
	ACE	**MSE**	**ACE**	**MSE**
RE	0.239	173	0.438	338
HE	0.247	160	0.421	309
NE	**0.231**	**64**	0.446	311

Table 5.7: Predictions by regression analysis for JFact

Algorithm	L=1		L=2	
	ACE	**MSE**	**ACE**	**MSE**
RE	0.338	285	0.281	294
HE	0.371	289	0.247	283
NE	0.322	285	**0.214**	**252**

Table 5.8: Predictions by regression analysis for Hermit

Overall, the NE-based predictions have the highest accuracy in all cases, see Tables 5.7, 5.8, 5.9. The lowest errors are in bold. It should be noticed that a linear function delivers the best performance predictions for JFact and Pellet while a quadratic function serves best for predicting the performance of Hermit.

Algorithm	L=1		L=2	
	ACE	**MSE**	**ACE**	**MSE**
RE	0.223	131	0.429	375
HE	0.223	140	0.454	378
NE	**0.198**	**96**	0.429	364

Table 5.9: Predictions by regression analysis for Pellet

Thus, we can conclude that the fact of producing the hardest samples by the NE algorithm is not a coincidence but an indicator of ontology entanglements that cause the performance degradation and useful for the performance prediction.

5.5.3 The ontology representation: Atomic Decomposition against Class Usage

So far, we assume that the AD is a suitable ontology graph. Nevertheless, this is not obvious. This experiment is aimed to test suitability of the AD for the performance prediction. We cannot show that the AD is the most appropriate graph in general because there are many possible ways to represent an ontology as a graph. However, we can compare it against another reasonable candidate, the CU, see Section 3.4.2.1.

According to the results of Section 5.5.2, the NE samples deliver the best prediction accuracy. Therefore, we apply the NE sampling to the CU of each ontology. Then, we use a suite of samples for each ontology to predict its computation time per reasoner via regression analysis. The polynomials of the same degree as for the AD, $L = 1$ for JFact, Pellet and $L = 2$ for Hermit, give the best prediction results for the CU. These results in comparison to the AD results per reasoner are given in Table 5.10.

Reasoner	**AD**		**CU**	
	ACE	**MSE**	**ACE**	**MSE**
JFact	0.231	64	0.289	240
Hermit	0.214	252	0.338	281
Pellet	0.198	96	0.239	157

Table 5.10: Comparison of performance predictions using the AD and CU

Table 5.10 shows that the AD allows to retrieve NE samples delivering significantly better accuracy for all reasoners than the CU does. Consequently, the

AD is more suitable ontology representation for the performance prediction than the CU.

5.5.4 Analysis of data volume required for accurate predictions

A local approach makes use of regression analysis to predict the performance by extrapolating beyond the range of observed data. The distance from the observed interval to a testing point determines the accuracy of extrapolation such that far points cause greater errors. The reason is uncertainty growing with the distance from the observed interval, see Section 2.4.2.1.

Therefore, it is worth investigating how prediction errors depend on the sample size limit. In addition, we target to find out which sample size limit per reasoner suffice to achieve satisfiable predictions.

Hypothesis. The prediction error declines as the sample size limit increases.

Test. We use the NE samples because they provide the best prediction accuracy. We impose a size restriction on these samples, filter out the samples which violate this restriction and then calculate the prediction error, either ACE or MSE, on remaining samples. As before, we average errors across all ontologies in the corpus. We use a polynomial of degree either $L = 1$ or $L = 2$ depending on which one bears the best prediction accuracy for a reasoner, see Section 5.5.2.

The results of the ACE and MSE dependency on the sample size limit for each reasoner are shown on Figures 5.11, 5.12 respectively. As before, we consider relative sizes of samples such that 0.1 means 10% of the full ontology.

Figure 5.11 shows unexpected behaviour of the ACE as the sample size limit grows. ACE curves of JFact and Pellet are similar. For small limits from 0.005 to 0.03 or 0.035, the ACE tends to grow and reaches its maximum at the end of this interval. Only after that, it starts to decline with the increasing limit as the *Hypothesis* claims. Thus, we observe a "hill". One can notice that the ACE reaches its nearly lowest values for sufficiently small limits, just after the rapid fall from the hill top: 0.045 for JFact and 0.065 for Pellet. Then, it slowly decreases.

The abnormal behaviour of the ACE for JFact and Pellet for small limits is explained by the existing bias of the corpus towards easy ontologies. Although the

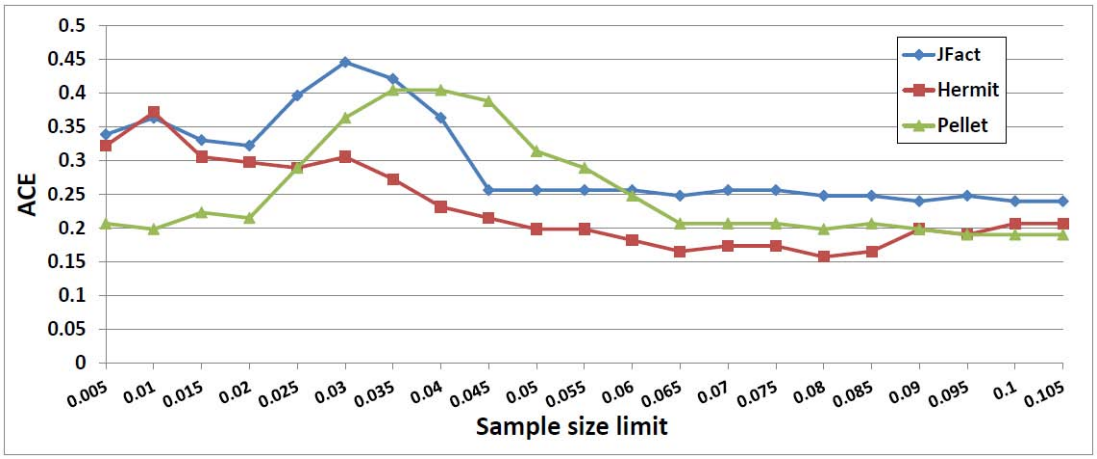


Figure 5.11: ACE against the sample size limit for JFact, Hermit, Pellet

bias is not significant, see Section 4.1, it plays its role because smallest samples can be not big enough to reflect the performance and, hence, they force to predict all ontologies as easy ones. Therefore, the ACE has low values. Then, as the sample size grows, the uncertainty of predictions takes its place and its influence gradually outweighs the effect of the biased corpus. This causes the rise of the ACE until reaching its maximum at the hill top. Starting from this point, the uncertainty of predictions begins to decline because samples reach indicative sizes and, hence, so does the ACE. Thus, the *Hypothesis* holds starting only from this point.

The ACE for Hermit behaves differently. First, it shows just a little hill at the very beginning and then steadily declines until the limit of 0.08. After that, it unexpectedly starts to grow. As a result, restricting the size limit to 0.08, we achieve the lower $ACE = 0.157$ than for bigger samples with the limit 0.1 which induce the $ACE = 0.206$. A possible reason of that is predicting the Hermit performance using the nonlinear function, a polynomial of degree $L = 2$, in contrast to JFact and Pellet which are predicted by the linear function. This nonlinearity causes greater uncertainty of results due to imprecise time measurements of large samples because these imprecise measurements are amplified by the quadratic term.

Figure 5.12 shows that the MSE behaves similarly to the ACE for all reasoners. The difference for JFact and Pellet is that the MSE does not have the nearly constant region after falling from its maximum but continues to decrease. This is explained by the fact that the MSE measures raw differences between the

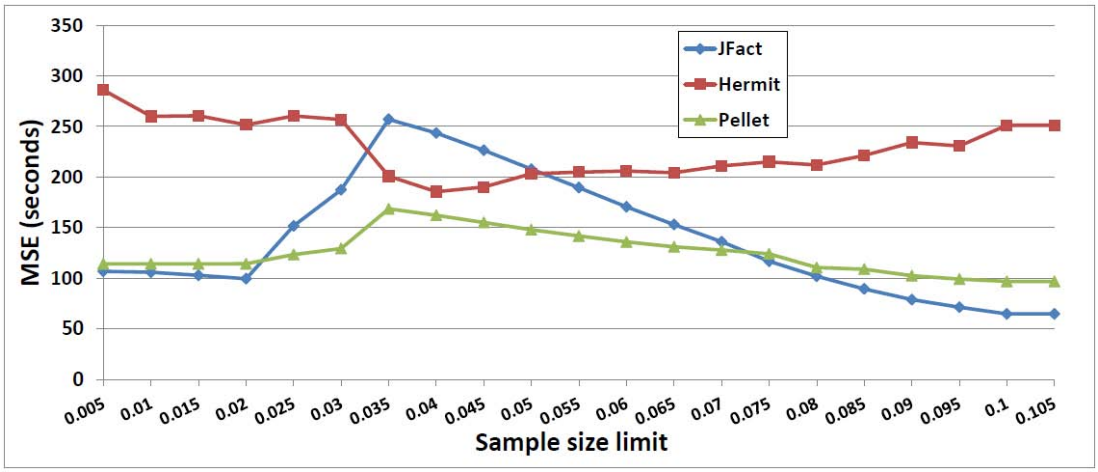


Figure 5.12: MSE against the sample size limit for JFact, Hermit, Pellet

prediction and true computation time in contrast to the ACE which only counts wrong bins.

The MSE curve for Hermit shows the familiar "hole" but, unlike the ACE, it has its minimum for significantly lower limit of 0.04 and steadily grows starting from this point. The reason is that the nonlinear term starts to influence earlier in this case because the MSE measures raw differences but not incorrectly predicted bins.

Thus, we can conclude that the *Hypothesis* holds only starting from certain size limit for JFact and Pellet errors and it holds only before certain size limit for Hermit errors.

5.6 Comparison of global and local approaches to the performance prediction

We have tested a global and local approach on the same ontology corpus and obtained the performance prediction results for each reasoner. Now, let us put everything together and compare the results.

A global approach suggests training a machine learning model on the ontology corpus and then using this model for performance predictions. This idea is proposed by *Kang et al.*, see Section 2.4.1.3. They suggest extracting 27 features from an ontology and, hence, computing its profile, as we call it. They also propose ontology binning by the computation time and specify performance bins.

Then, a supervised machine learning model predicts the performance bin for an ontology given its profile.

We specify a new set of 52 features which overlaps with the set of features by *Kang et al.*, see Section 3.2.1. We also add the additional bin $E \in [1000s, +\infty)$ to their performance bins. Our ontology corpus is different. Therefore, we do not check their results here and do not claim that our global approach with 52 features is better than theirs. Our goal is to test the idea, but not a particular implementation, of the global approach to the performance prediction and compare its results with the local approach which we introduce in this work.

One of our main findings is that all 52 ontology features can be replaced by just one feature, the ontology size, with insignificant loss of information, see Section 5.2.2. In addition, we have discovered that the global approach model with only one feature, the ontology size, delivers better prediction results than one with all 52 features.

We have designed and tested the local approach to the performance prediction that suggests building ontology samples and then using these samples to predict the computation time of a reasoner on this ontology via regression analysis. We have found out that the NE is the most suitable sampling algorithm out of three proposed for the performance prediction and it allows to obtain satisfactory prediction results with samples of sufficiently small sizes, not more than 10% of the full ontology. We have discovered that the NE "chaining" produces hardest samples for all reasoners. We have also shown that the AD is a suitable ontology graph by comparing it with another reasonable candidate, the CU. Finally, we have investigated the prediction accuracy against the sample size limit.

Now, let us gather the best prediction results of the global approach and local approach and compare them in a single table. The global approach is evaluated by the GE which is either averaged ACE or averaged MSE, see Section 5.3. For the sake of simplicity and clarity of comparisons, we denote the GE by either ACE or MSE accordingly. The results are shown in Table 5.11, where Glob.52f is the global approach using 52 features and Glob.1f is the global approach using ontology size as the only feature.

As Table 5.11 shows, the local approach delivers better prediction accuracy than the global one, using either 52 features or one feature, for all three reasoners if we compare by the ACE. However, the local approach does not outperform the global one for Hermit if we compare by the MSE. In contrast, it outperforms the

Reasoner	ACE			MSE		
	Glob.52f	**Glob.1f**	**Local**	**Glob.52f**	**Glob.1f**	**Local**
JFact	0.278	0.241	**0.231**	161	136	**64**
Hermit	0.258	0.257	**0.214**[1]	191	**171**	252[2]
Pellet	0.287	0.215	**0.198**	123	144	**96**

Table 5.11: Comparison of global and local approaches to the performance prediction

global approach significantly, by 4.3%, if we compare by the ACE. This can be caused by using the quadratic polynomial to predict the performance of Hermit because the quadratic term amplifies imprecise time measurements of samples nonlinearly that affects the MSE greater than the ACE.

We should remind the reader that the local approach never observes true computation times of ontologies and uses only small ontology samples, up to 10%, to make predictions. Hence, due to extrapolation, it has greater uncertainty of predictions, particularly when we evaluate it by the MSE. Nonetheless, this uncertainty is successfully reduced by the suitable sampling algorithm such that the local approach shows surprisingly better MSEs for JFact and Pellet and fails to do so only for Hermit. Computation time predictions for each reasoner on each ontology in the corpus via the local approach are given in Appendix 6.4.

[1] **0.157** for 8% size limit
[2] **185** for 4% size limit

Chapter 6

Conclusions

6.1 Results and gained insights

Ontologies are widely used for knowledge representation. To support ontology engineering, many specialised software tools, called reasoners, have been developed. New reasoners are permanently extending the list of existing ones. Reasoners are very useful because they are able to check the consistency of the encoded knowledge and provide logical inferences.

However, the computational performance of reasoners varies enormously across ontologies. The same ontology can take seconds to process for one reasoner and hours for another. These variability phenomena are caused by the interrelation of two facts. Firstly, ontologies can be complex and diverse structures, in contrast to propositional formulae, for example. Secondly, each reasoner has its own optimizations and internal implementation tricks which can succeed on one input and fail on others. Therefore, predicting the computational performance of reasoners across ontologies is a hard problem.

Nonetheless, this problem is worth investigating. Finding ontology properties which cause performance degradation is an opportunity to use this information during ontology engineering for avoiding undesirable properties. Additionally, reasoner developers can benefit from predicting the performance of reasoners on unknown inputs without the necessity to literally process the full ontology that can take too long. Ontology engineers can be supplied with the "progress bar", which uses the predicted computational time of a reasoner. It can help them to decide whether the reasoner should be interrupted, if its computation time is too long, or it should complete its task, if it is worth this time.

The problem was studied before. A common way to investigate ontology properties is to extract multiple global features from it. Then, supervised machine learning can be applied to predict the performance of a reasoner given an ontology feature vector. We call this vector an ontology profile. In order to learn a model, an ontology corpus is required. Ontology profiles have to be extracted for all ontologies in the corpus to prepare a data set for model training and testing. This is a known method which we call a global approach because it observes ontologies only "externally" by computing their profiles and uses the computation times of full ontologies to train a model. It never attempts to look inside the ontology structure or explore its parts.

In this thesis we investigate the computational performance of three commonly used reasoners, JFact, Hermit, Pellet, on the ontology corpus combined from 121 ontology of various hardness, see Appendix 6.4.

We started from the global approach and defined the list of 52 ontology features, adding 28 new ontology features to already studied ones [16]. These features aimed to capture different sorts of information extractable from an ontology.

Then, we evaluated our 52 features by their impact on the computation time of a reasoner across ontologies in the corpus. We ranked features by the absolute value of their correlation with the computation time per reasoner. We found out that only a small group of 11 features affected the computation time significantly for each reasoner. Most influential features amongst them were *#EquivalentClassesAxioms*, *parsingDepth*, *#ObjectPropertyNameOccurrences* for JFact and Pellet. Hermit was most affected by *#ObjectAllValuesFrom* in contrast to others. These results can be interesting for ontology designers as warnings during ontology modelling. For example, using equivalent classes axioms too often is likely to degrade the reasoning performance significantly. Therefore, they should be avoided in cases when it is possible. Thus, our feature ranking can support wiser ontology design.

After that, we investigated how the features were intercorrelated in the corpus. For these purposes, we applied PCA to the set of profiles computed for all ontologies. As a result, we discovered that all 52 features can be replaced by just one feature, the PC1 projection, preserving 98.5% of all variance. This means that ontology features are highly intercorrelated. In addition, we projected all ontologies into two dimensional space with the PC1 and PC2 projections as coordinates and achieved ontology clustering with clearly separable clusters of NCI

Thesaurus releases.

Achieving such a good representation of all ontology features by just one variable, the PC1 projection, we raised the question how this variable was connected to the original features. To answer this question, we investigated how the original features contributed to this variable value and found out that only 7 out of 52 features had significant contribution values. The most contributing feature amongst them was #ClassNameOccurrences which was an ontology size feature.

In fact, 6 out of these 7 features were ontology size features. Therefore, we hypothesized that the PC1 projection correlates with the ontology size. We tested our hypothesis by calculating the absolute value of the correlation coefficient between the ontology size and all 52 PCA projection coordinates. As a result, we discovered that the PC1 projection had 90% correlation value with the ontology size while all others had insignificant values less than 9%. Thus, the PC1 and only the PC1 projection had significant correlation with the ontology size and, hence, we justified our hypothesis.

As a consequence of our feature analysis, all 52 ontology features can be replaced with minor loss of information by just one feature, the ontology size, which is simple and easily extractable. However, one can argue that this analysis does not consider any relations to the computation time and the performance prediction. This was the reason why we targeted to compare the prediction accuracy of the same global approach model with the single feature, the ontology size, and with all 52 features. We found out that almost in all cases the model with one feature delivered better, sometimes significantly better, prediction results than the model with 52 features. Only in one case it did just slightly worse.

Thus, we discovered that, in addition to the efficient representation of 52 features by the ontology size alone, the prediction model using this feature alone showed a better prediction accuracy than one with all 52 features. This implies that extracting multiple ontology features, as a common approach, makes no or little sense for the performance prediction.

In this thesis, we introduce a novel approach to the performance prediction of reasoners, called a local approach, because it is based on gathering the data from ontology subsets, ideally small, which we call samples, and then using these data to predict the reasoning performance on the full ontology.

Firstly, these samples are generally hard to find such that they cause significant computation difficulties for reasoners. For example, arbitrary small subsets

are usually easy for all reasoners, even if the full ontology is hard, because they do not reflect entanglements. Secondly, just hard samples do not suffice either because they should additionally be performance indicative. We know, however, that solely easy samples force all ontologies, even hard ones, to be considered as easy. Hence, they are not indicative in this case. Thus, we raised the question how to select samples suitable for the performance prediction. In order to find the answer, we explored graph-based ontology representations.

To construct suitable samples, we proposed the idea of growing-based sampling using the ontology graph. We suggested three possible ways of growing-based sampling: History-based extensions (HE), and Neighbourhood-based extensions (NE), and Random extensions (RE). We designed three algorithms, called correspondingly, to build these samples. We also performed theoretical analysis of the HE and NE behaviour by estimating the probabilities of "chaining" and showed that the NE algorithm forced building long "chains" of visited nodes on the ontology graph while the HE exhibited "branching" in some local area.

In order to find out how a reasoner performs on RE, HE and NE samples, we tested sampling algorithms on the ontology corpus. As a result, we discovered that the NE algorithm produced hardest samples on average for all three reasoners and the HE algorithm constructed the easiest ones. This implies that "chaining" builds significantly harder subsets than "branching" for each reasoner.

To predict the computation performance via a local approach, we suggested to apply regression analysis which extrapolated time measurements of built samples to the full ontology size. As a consequence of our feature analysis, we suggested to extract only one feature from a sample, the sample size. Hence, there was only one independent and one dependent variable. We also suggested to use a simple polynomial of degree 1 or 2 as a regression model for the performance prediction using samples.

Although we found out which algorithm produced hardest samples and which one produced easiest samples for a reasoner, this did not show us which samples were the most suitable for the performance prediction. To investigate this, we predicted the performance of all reasoners on the ontology corpus via regression analysis of RE, HE, NE samples and then calculated prediction errors. As a result, we found out that NE samples delivered the best prediction accuracy in all cases and, hence, these samples were not only the hardest but also most indicative for the performance prediction.

Thus, we obtained performance prediction results of the designed local approach. We also gained from these results that JFact and Pellet were better predictable by a linear function while Hermit was better predictable by a quadratic function. We also tested different ontology graphs and showed that the Atomic Decomposition, AD, was more suitable for the performance prediction than the Class Usage, CU.

In all performance experiments of the local approach, we restricted sample sizes to be not more than 10% of the full ontology. Therefore, we targeted to test how the prediction error depended on the size limit in the range from 0.5% to 10.5%. We hypothesised that the error declined as the limit increased. However, we observed unexpected behaviour of the error for JFact and Pellet when the limit was low. In contrast, the Hermit errors showed unexpected behaviour for higher limits where errors were greater than for smaller limits. The latter implied that samples of lower limits were able to bear a better prediction accuracy than ones of higher limits for Hermit. Hence, its results for 10% limit were not the best, in contrast to two other reasoners.

Finally, we compared the best results achieved by the global approach, which used all 52 features and the ontology size alone, with the results of the local approach, which used 10% limitation of sample sizes. Consequently, we found out that the local approach produced better, sometimes significantly better, performance predictions in all cases except one.

We should notice that the local approach never observes the computations times of full ontologies and does not require multiple ontologies to fit its model. Nonetheless, it outperforms the global approach with the ontology size as the only ontology feature and, hence, outperforms even more the common global approach with all features. Overall, the local approach achieves nearly 80% bin prediction accuracy for all reasoners.

A great advantage of the local approach is that it does not need an ontology corpus to fit its model, except choosing the optimal polynomial degree. Only one testing ontology suffice. In contrast, the global approach completely relies on the ontology corpus to train its model. The corpus should be combined cautiously such that it has sufficient generalization abilities and is biased as little as possible which is a hard task in general. In addition, we have to know the computation time for all ontologies in order to train a global approach model.

However, the local approach has also disadvantages. Firstly, it requires reasoning to be done on samples in order to predict the computation time of the full ontology. Secondly, it relies on the ontology graph, the AD, which can take too long to compute for large and hard ontologies, such as NCI Thesaurus releases. We overcome this issue by computing the AD only on a randomly selected ontology part of feasible size and ignoring the remaining part completely. Obviously, this worsens the prediction abilities. Despite that, the local approach achieves a higher prediction accuracy than the global one.

6.2 What is not included

Some ideas and experiments are not included into this thesis because we wish to keep the reader focused on our main findings and most successful methods.

As an alternative to growing-based sampling, we also developed module-based sampling which uses ontology modules with certain properties instead of building samples iteratively using the ontology graph. Considering that the computational performance of a propositional formula can be predicted by the formula density, we similarly defined the density of a module as the number of axioms in it divided by its signature size. The modules extracted from an ontology based on the specified density range were called density modules. We targeted to find similar phase transition points for ontology modules as in the propositional case. However, it was hard to specify the density range such that to obtain indicative modules because ontologies appeared too diverse in their structure. As a result, growing-based sampling allowed to retrieve more suitable samples for the performance prediction because the search space was significantly reduced by the ontology graph.

We also tried genuine modules as samples because they are self-contained ontology subsets. Surprisingly, we found out that genuine modules were too easy for all reasoners to process and, hence, not indicative. We can make parallels here with the HE samples which, in some sense, are forced towards the self-contained subsets and are easiest for a reasoner.

We also tested a linear regression model with all 52 features for local approach predictions. Exactly as the global model did, it showed a worse prediction accuracy than the local approach model with the size feature alone.

6.3 Contributions

1. Ontology features are highly intercorrelated. As a consequence, multiple ontology features can be represented by just one feature, which is the ontology size, with insignificant loss of information.

2. A global approach model, using the ontology size as the only feature, is able to deliver more accurate performance predictions than one, using multiple ontology features. Therefore, extracting multiple ontology features to predict the performance makes no or little sense.

3. We propose the local approach for the performance analysis and prediction which is based on small ontology subsets, samples. It gathers the data from these samples and provide performance predictions for a reasoner on the full ontology.

4. We propose growing-based sampling using an ontology graph and design three algorithms to build samples for the performance prediction. We provide the theoretical analysis for two of these algorithms. As experiments show, "chaining" on the ontology graph produces harder and more suitable samples for the performance prediction than either "branching" or random selection for all reasoners.

5. We justify that the Atomic Decomposition is a suitable ontology graph for growing-based sampling.

6. We suggest to use regression analysis of samples for performance predictions via a local approach. We show that a simple polynomial of degree 1 or 2 suffices as a regression model for the performance prediction.

7. As experiments show, the proposed local approach, despite its simplicity, is able to produce more accurate predictions than the common global approach. The local approach has prominent advantages but also disadvantages.

6.4 Discussions and future work

One of our main findings is that multiple ontology features are efficiently represented by the ontology size alone. Why does this surprising result take place? One

of possible explanations can be given by considering the way multiple features are extracted from an ontology. In fact, all features are extracted from ontology axioms and then aggregated for the full ontology. All axioms of the ontology constitute its size. Hence, the bigger the ontology size is, the bigger feature values are. However, it is still surprising how efficiently the size alone replaces all 52 features, preserving 98.5% of all variance.

As a consequence, it is an expectable outcome that 52 features add little useful information for a performance predictor in comparison to the size alone. Therefore, the global approach model with the size alone shows comparable and even better prediction accuracy than the same model with all 52 features. The better prediction accuracy is explained by benefits of the dimensionality reduction from 52 to 1 dimension that outweigh minor losses of useful information.

We propose the local approach to the performance prediction. It uses small samples of the size not more than 10% of the full ontology and tries to extrapolate far beyond the observed interval, to the full ontology size. The target ontology is the only available data source. The local approach never observes other ontologies and, hence, it does not have any information of how big or small the computation time of the full ontology can be according to its size. Finally, it uses just a simple polynomial of degree 1 or 2 as its prediction model. Thus, why does the local approach succeed at all?

The reason of its success is that it uses not random samples for the performance prediction. These samples are performance indicative because they are built by a suitable sampling algorithm, the NE, which is executed on a suitable ontology graph, the AD.

Although the local approach provides satisfactory results of the performance prediction for all three reasoners, there is still much room for improvement.

Firstly, we impose the 10% sample size limit for all ontologies. However, this limit can be estimated for each ontology accordingly because for some ontologies samples with such a size restriction are not enough indicative to obtain a precise prediction for a reasoner on the full ontology.

Secondly, we use a simple linear or quadratic function as a regression model. There probably exist more suitable models for extrapolation of sample measurements to the full ontology.

Thirdly, a regression model can be chosen for each ontology adaptively in addition to fitting its parameters to the gathered data. As a simplest example,

we can determine an optimal degree of a polynomial for each ontology separately. This should not, however, be estimated using the standard least squares method because a polynomial of higher degree is always fitted better to the data than a polynomial of lower degree. Some other criteria, which rely on indicative ontology properties, should be designed.

Bibliography

[1] A. Armas Romero, B. Cuenca Grau, and I. Horrocks. MORe: Modular combination of OWL reasoners for ontology classification. In *Proceedings of the 11th International Semantic Web Conference (ISWC-12)*, volume 7649 of *Lecture Notes in Computer Science*, pages 1–16, 2012.

[2] F. Baader. Terminological cycles in a Description Logic with existential restrictions. In *Proceedings of the 18th International Joint Conference on Artificial Intelligence (IJCAI-03)*, pages 325–330, 2003.

[3] F. Baader, S. Brandt, and C. Lutz. Pushing the $\mathcal{EL}$ envelope. In *Proceedings of the 19th International Joint Conference on Artificial Intelligence (IJCAI-05)*, pages 364–369, 2005.

[4] F. Baader, D. Calvanese, Diego andMcGuinness, D. Nardi, and P. F. Patel-Schneider, editors. *The Description Logic Handbook: Theory, Implementation, and Applications*. Cambridge University Press, 2003.

[5] D. Calvanese, G. De Giacomo, D. Lembo, M. Lenzerini, and R. Rosati. DL-Lite: Tractable description logics for ontologies. In *Proceedings of the 20th National Conference on Artificial Intelligence (AAAI-05)*, pages 602–607, 2005.

[6] T. Cover and P. Hart. Nearest neighbor pattern classification. *Information Theory, IEEE Transactions on*, 13(1):21–27, 1967.

[7] C. Del Vescovo. *The Modular Structure of an Ontology: Atomic Decomposition and its Applications*. PhD thesis, The University of Manchester, 2013.

[8] R. S. Gonçalves, N. Matentzoglu, B. Parsia, and U. Sattler. The empirical robustness of description logic classification. In T. Eiter, B. Glimm,

Y. Kazakov, and M. Krötzsch, editors, *Description Logics*, volume 1014 of *CEUR Workshop Proceedings*, pages 197–208. CEUR-WS.org, 2013.

[9] R. S. Gonçalves, B. Parsia, and U. Sattler. Performance heterogeneity and approximate reasoning in description logic ontologies. In *Proceedings of the 11th International Semantic Web Conference (ISWC-12)*, volume 7649 of *Lecture Notes in Computer Science*, pages 82–98, 2012.

[10] T. R. Gruber. A translation approach to portable ontology specifications. *Knowledge Acquisition*, 5(2):199 – 220, 1993.

[11] I. Guyon and A. Elisseeff. An introduction to variable and feature selection. *J. Mach. Learn. Res.*, 3:1157–1182, Mar. 2003.

[12] T. Hastie, R. Tibshirani, and J. J. H. Friedman. *The elements of statistical learning*, volume 1. Springer New York, 2001.

[13] J. Hladik. A generator for description logic formulas. In I. Horrocks, U. Sattler, and F. Wolter, editors, *Proceedings of the 2005 International Workshop on Description Logics (DL2005), Edinburgh, Scotland, UK, July 26-28, 2005*, volume 147 of *CEUR Workshop Proceedings*. CEUR-WS.org, 2005.

[14] I. Horrocks, P. F. Patel-Schneider, and F. van Harmelen. From SHIQ and RDF to OWL: The making of a web ontology language. *Journal of Web Semantics*, 1(1):7–26, 2003.

[15] I. T. Jolliffe. *Principal Component Analysis*. Springer, second edition, Oct. 2002.

[16] Y.-B. Kang, Y.-F. Li, and S. Krishnaswamy. Predicting reasoning performance using ontology metrics. In *The Semantic Web - ISWC 2012*, volume 7649 of *Lecture Notes in Computer Science*, pages 198–214. Springer Berlin Heidelberg, 2012.

[17] Y. Kazakov. Consequence-driven reasoning for Horn $\mathcal{SHIQ}$ ontologies. In *Proceedings of the 22nd International Workshop on Description Logics (DL-09)*, volume 477 of *CEUR (`http://ceur-ws.org/`)*, 2009.

[18] Y. Kazakov, M. Krötzsch, and F. Simancik. Concurrent classification of $\mathcal{EL}$ ontologies. In *Proceedings of the 10th International Semantic Web Conference (ISWC-11)*, volume 7031 of *Lecture Notes in Computer Science*, pages 305–320, 2011.

[19] S. Kirkpatrick and B. Selman. Critical behavior in the satisfiability of random boolean expressions. *Science*, 264(5163):1297–1301, 1994.

[20] R. Kohavi. A study of cross-validation and bootstrap for accuracy estimation and model selection. pages 1137–1143. Morgan Kaufmann, 1995.

[21] C. L. Lawson and R. J. Hanson. *Solving least squares problems.* 3 edition, 1995.

[22] B. Motik, R. Shearer, and I. Horrocks. A hypertableau calculus for $\mathcal{SHIQ}$. In *Proceedings of the 20th International Workshop on Description Logics (DL-07)*, pages 419–426. Bozen/Bolzano University Press, 2007.

[23] K. Pearson. On lines and planes of closest fit to systems of points in space. *Philosophical Magazine*, 2(6):559–572, 1901.

[24] S. Rudolph, T. Tserendorj, and P. Hitzler. What is approximate reasoning? In D. Calvanese and G. Lausen, editors, *Web Reasoning and Rule Systems*, volume 5341 of *Lecture Notes in Computer Science*, pages 150–164. Springer Berlin Heidelberg, 2008.

[25] U. Sattler, T. Schneider, and M. Zakharyaschev. Which kind of module should I extract? In *Proceedings of the 22nd International Workshop on Description Logics (DL-09)*, volume 477 of *CEUR (`http://ceur-ws.org/`)*, 2009.

[26] T. J. Schaefer. The complexity of satisfiability problems. In *Proceedings of the tenth annual ACM symposium on Theory of computing*, STOC '78, pages 216–226, New York, NY, USA, 1978. ACM.

[27] E. Sirin, B. Parsia, B. Cuenca Grau, A. Kalyanpur, and Y. Katz. Pellet: A practical OWL-DL reasoner. *Journal of Web Semantics*, 5(2):51–53, 2007.

[28] K. A. Spackman, K. E. Campbell, and R. A. Côté. SNOMED RT: A reference terminology for health care. *Journal of the American Medical Informatics Association*, pages 640–644, 1997.

[29] D. Tsarkov and I. Horrocks. FaCT++ Description Logic reasoner: System description. In *Proceedings of the 3rd International Joint Conference on Automated Reasoning (IJCAR-06)*, volume 4130 of *Lecture Notes in Computer Science*, pages 292–297. Springer-Verlag, 2006.

[30] D. Tsarkov and I. Palmisano. Divide et impera: Metareasoning for large ontologies. In *Proceedings of the 9th International Workshop on OWL: Experiences and Directions (OWLED-12)*, volume 849 of *CEUR (http://ceur-ws.org/)*, 2012.

[31] D. H. Wolpert and W. G. Macready. No free lunch theorems for optimization. *IEEE TRANSACTIONS ON EVOLUTIONARY COMPUTATION*, 1(1):67–82, 1997.

[32] H. Zhang, Y.-F. Li, and H. B. Kuan Tan. Measuring design complexity of semantic web ontologies. *The Journal of Systems and Software*, 83:803–814, 2010.

Appendix A.
Ontology corpus

id	Ontology	Repository	JFact (sec)	Hermit (sec)	Pellet (sec)
0	amodel	mowlcorp	0.01	0.01	0.02
1	02.00-Release	ncit	33.92	3.45	8.52
2	juegos	mowlcorp	25.80	7.08	0.36
3	chment	mowlcorp	0.01	0.01	0.01
4	03.10J-Release	ncit	35.52	3.81	9.57
5	03.12A-Release	ncit	39.81	7.03	11.58
6	03.12E-Release	ncit	45.49	3.77	8.95
7	04.02H-Release	ncit	43.42	4.64	16.33
8	04.03N-Release	ncit	72.40	5.97	6.59
9	04.04J-Release	ncit	80.65	5.87	6.59
10	04.05F-Release	ncit	77.70	7.11	6.86
11	04.06i-Release	ncit	78.93	6.75	5.93
12	04.08B-Release	ncit	86.87	12.45	9.51
13	04.09A-Release	ncit	121.29	16.23	11.30
14	3139	mowlcorp	0.06	0.06	0.09
15	owl-dl	mowlcorp	8.71	1.37	1.09
16	roides	mowlcorp	0.01	0.01	0.01
17	09.03d-Release	ncit	415.38	622.33	130.71
18	09.04d-Release	ncit	414.55	662.00	141.75
19	09.05d-Release	ncit	416.55	762.37	139.34
20	09.06e-Release	ncit	449.56	619.60	207.90
21	09.07e-Release	ncit	450.32	481.15	148.53
22	09.08e-Release	ncit	455.91	711.36	259.48
23	09.09c-Release	ncit	462.74	591.21	188.81
24	09.10d-Release	ncit	467.77	819.44	208.15
25	09.12d-Release	ncit	481.30	536.10	121.84
26	091216	mowlcorp	0.58	0.11	0.20
27	10.01d-Release	ncit	488.74	817.92	126.95
28	10.02d-Release	ncit	498.41	810.12	134.25
29	10.03h-Release	ncit	507.82	853.79	150.21

id	Ontology	Repository	JFact (sec)	Hermit (sec)	Pellet (sec)
30	10.04f-Release	ncit	508.21	888.31	160.92
31	10.05d-Release	ncit	511.27	938.71	177.31
32	10.06e-Release	ncit	522.21	972.89	144.55
33	10.07d-Release	ncit	533.87	1000.00	246.09
34	10.08e-Release	ncit	556.52	972.69	138.67
35	10.10a-Release	ncit	562.56	153.14	251.72
36	12.03d-Release	ncit	859.31	124.65	485.35
37	12.04e-Release	ncit	885.50	129.20	644.36
38	12.05d-Release	ncit	890.36	125.79	712.90
39	12.06d-Release	ncit	911.75	125.92	559.25
40	12.07d-Release	ncit	917.27	131.03	761.53
41	zas2-2	mowlcorp	0.01	0.01	0.01
42	O3DVT	mowlcorp	0.01	0.01	0.04
43	pro	mowlcorp	0.04	0.06	0.04
44	tdl-3	mowlcorp	0.01	0.01	0.01
45	artist	mowlcorp	0.01	0.01	0.01
46	Phone	mowlcorp	0.01	0.02	0.01
47	orTI-1	mowlcorp	1000.00	131.08	1000.00
48	cmo	mowlcorp	0.01	0.01	0.06
49	genpol	mowlcorp	0.03	0.04	0.05
50	go-bp	mowlcorp	79.52	8.17	5.40
51	243861	mowlcorp	0.01	0.01	0.01
52	rotege	mowlcorp	1.26	0.87	1.75
53	zation	mowlcorp	0.01	0.03	0.03
54	tp3	mowlcorp	0.33	18.66	0.80
55	axioms	mowlcorp	0.01	0.01	0.01
56	ex-1.0	mowlcorp	0.01	0.02	0.03
57	cs-ont	mowlcorp	0.01	0.01	0.01
58	products	mowlcorp	0.01	0.01	0.01
59	n-slim	mowlcorp	0.48	1.11	0.37
60	onship	mowlcorp	0.01	0.01	0.01
61	apollo-akesios	bioportal	0.01	0.01	0.01
62	bioassay	bioportal	0.18	5.05	0.39
63	bone-dysplasia	bioportal	92.40	3.56	1.90
64	c-elegans-gross	bioportal	14.92	1.33	0.35
65	cekaw	mowlcorp	0.01	0.01	0.01
66	WSRF	mowlcorp	0.01	0.01	0.01
67	new-1	mowlcorp	0.01	0.01	0.01
68	cardiac-electroph	bioportal	1000.00	67.99	15.33
69	foaf-basic	mowlcorp	0.01	0.01	0.01
70	cell-line	bioportal	2.49	0.50	0.35
71	clinical-meas	bioportal	0.09	0.04	0.02
72	wine	mowlcorp	0.01	0.01	0.01
73	orTI-2	mowlcorp	1000.00	106.72	30.14
74	dengue-fever	bioportal	3.77	7.31	3.32
75	dermlex	bioportal	35.49	1.31	0.48
76	drosophila	bioportal	9.71	2.30	22.54
77	Cerveau	mowlcorp	0.07	0.01	0.01
78	eagle-i	bioportal	2.15	16.49	1.78
79	event	mowlcorp	1.81	0.56	0.95

id	Ontology	Repository	JFact (sec)	Hermit (sec)	Pellet (sec)
80	event-inoh	bioportal	1.16	0.35	0.63
81	n-0918	mowlcorp	0.14	0.03	0.14
82	sp-all	mowlcorp	0.01	0.01	0.02
83	ract2	mowlcorp	1.26	0.49	8.70
84	fda-med-devices	bioportal	0.83	0.38	0.05
85	fly-taxonomy	bioportal	2.80	0.44	0.28
86	foundational-anat	bioportal	1000.00	19.32	508.86
87	gene	bioportal	214.83	14.05	5.12
88	glycomics	bioportal	28.74	95.32	2.46
89	hom-dxvcodes2	bioportal	21.52	1.23	0.38
90	hom-icd9-dxv	bioportal	23.32	1.34	0.35
91	hom-oshpd	bioportal	38.44	2.61	0.70
92	hom-ucare	bioportal	0.01	0.01	0.01
93	hugo	bioportal	93.32	2.56	0.55
94	human-dev	bioportal	1.59	0.37	6.36
95	human-phe	bioportal	10.62	0.94	0.28
96	hymenoptera	bioportal	0.47	0.41	0.62
97	nursing	bioportal	3.00	5.58	10.42
98	mahco	bioportal	6.34	4.55	4.03
99	mouse-exp	bioportal	0.01	0.01	0.01
100	mouse-anat	bioportal	110.70	3.68	0.98
101	nci-thesaurus	bioportal	1000.00	121.00	40.12
102	neural-electrom	bioportal	0.64	65.48	53.89
103	neural-immune	bioportal	2.15	0.52	0.44
104	nif-cell	bioportal	1.03	37.89	2.18
105	nif-subcellular	bioportal	0.93	27.69	2.10
106	parasite-lifecycle	bioportal	0.36	0.87	1.35
107	phylogenetic	bioportal	0.01	0.01	0.01
108	physico-chem	bioportal	0.02	0.04	0.03
109	plant-env-cond	bioportal	0.01	0.03	0.01
110	pma-2010	bioportal	0.01	0.01	0.01
111	hoehndorf-mesh	bioportal	1000.00	23.06	11.70
112	sanou	bioportal	0.01	0.01	0.01
113	solanaceae-phe	bioportal	0.01	0.02	0.01
114	taxonomic	bioportal	0.01	0.01	0.01
115	teleost-anat	bioportal	2.50	1.00	1.11
116	teleost-taxonomy	bioportal	85.05	2.49	1.16
117	thesaurus-altern	bioportal	0.05	0.05	0.07
118	time-event	bioportal	0.76	1.65	0.45
119	zebrafish	bioportal	0.85	0.34	1.96
120	vaccine	bioportal	7.59	27.53	1000.00

Appendix B. Performance predictions via regression analysis of samples

id	Ontology	JFact (sec)		Hermit (sec)		Pellet (sec)	
		time[1]	predict[2]	time	predict	time	predict
0	amodel	0.01	0.02	0.01	0.04	0.02	0.03
1	02.00-Release	33.92	33.55	3.45	4.15	8.52	1.17
2	juegos	25.80	0.29	7.08	304.10	0.36	0.18
3	chment	0.01	0.01	0.01	0.01	0.01	0.01
4	03.10J-Release	35.52	36.27	3.81	5.44	9.57	1.13
5	03.12A-Release	39.81	37.96	7.03	4.51	11.58	1.20
6	03.12E-Release	45.49	33.02	3.77	4.45	8.95	1.01
7	04.02H-Release	43.42	44.39	4.64	4.74	16.33	1.21
8	04.03N-Release	72.40	97.76	5.97	6.78	6.59	34.25
9	04.04J-Release	80.65	101.71	5.87	6.62	6.59	35.08
10	04.05F-Release	77.70	95.60	7.11	5.72	6.86	32.96
11	04.06i-Release	78.93	108.01	6.75	6.95	5.93	37.45
12	04.08B-Release	86.87	116.41	12.45	6.76	9.51	41.18
13	04.09A-Release	121.29	126.23	16.23	7.22	11.30	45.44
14	3139	0.06	0.01	0.06	0.03	0.09	0.02
15	owl-dl	8.71	1.04	1.37	1.93	1.09	1.11
16	roides	0.01	0.01	0.01	0.01	0.01	0.01
17	09.03d-Release	415.38	373.54	622.33	30.23	130.71	203.76
18	09.04d-Release	414.55	306.09	662.00	32.03	141.75	161.51
19	09.05d-Release	416.55	372.46	762.37	394.18	139.34	173.28
20	09.06e-Release	449.56	377.07	619.60	44.93	207.90	216.63
21	09.07e-Release	450.32	377.03	481.15	409.11	148.53	192.67
22	09.08e-Release	455.91	400.01	711.36	43.24	259.48	219.29
23	09.09c-Release	462.74	370.69	591.21	34.73	188.81	192.34
24	09.10d-Release	467.77	400.07	819.44	373.32	208.15	180.69
25	09.12d-Release	481.30	472.41	536.10	25.94	121.84	204.29

[1] measured computation times
[2] performance predictions given by the model
[3] the model predicts exceeding the timeout of 1000 seconds

id	Ontology	JFact (sec)		Hermit (sec)		Pellet (sec)	
		time	predict	time	predict	time	predict
26	091216	0.58	0.07	0.11	0.06	0.20	0.15
27	10.01d-Release	488.74	392.22	817.92	275.52	126.95	176.18
28	10.02d-Release	498.41	374.80	810.12	20.11	134.25	162.40
29	10.03h-Release	507.82	422.59	853.79	282.02	150.21	178.76
30	10.04f-Release	508.21	378.24	888.31	257.78	160.92	163.49
31	10.05d-Release	511.27	437.72	938.71	330.63	177.31	223.15
32	10.06e-Release	522.21	377.82	972.89	289.30	144.55	193.62
33	10.07d-Release	533.87	411.70	1000	277.24	246.09	171.15
34	10.08e-Release	556.52	547.09	972.69	25.07	138.67	247.03
35	10.10a-Release	562.56	602.08	153.14	341.60	251.72	258.09
36	12.03d-Release	859.31	808.42	124.65	515.13	485.35	365.08
37	12.04e-Release	885.50	939.75	129.20	546.10	644.36	406.99
38	12.05d-Release	890.36	943.70	125.79	545.91	712.90	398.50
39	12.06d-Release	911.75	759.61	125.92	447.13	559.25	318.44
40	12.07d-Release	917.27	833.40	131.03	724.63	761.53	355.20
41	zas2-2	0.01	0.02	0.01	0.03	0.01	0.02
42	O3DVT	0.01	0.03	0.01	0.05	0.04	0.05
43	pro	0.04	0.05	0.06	0.05	0.04	0.06
44	tdl-3	0.01	0.01	0.01	0.01	0.01	0.01
45	artist	0.01	0.01	0.01	0.01	0.01	0.01
46	Phone	0.01	0.01	0.02	0.01	0.01	0.01
47	orTI-1	1000	1000*[3]	131.08	36.25	1000	1000*
48	cmo	0.01	0.12	0.01	0.11	0.06	0.25
49	genpol	0.03	0.07	0.04	0.10	0.05	0.13
50	go-bp	79.52	22.01	8.17	3.98	5.40	1.63
51	243861	0.01	0.01	0.01	0.01	0.01	0.01
52	rotege	1.26	0.50	0.87	0.55	1.75	1.34
53	zation	0.01	0.01	0.03	0.02	0.03	0.03
54	tp3	0.33	0.01	18.66	0.01	0.80	0.01
55	axioms	0.01	0.01	0.01	0.01	0.01	0.03
56	ex-1.0	0.01	0.01	0.02	0.01	0.03	0.03
57	cs-ont	0.01	0.01	0.01	0.01	0.01	0.01
58	products	0.01	0.01	0.01	0.01	0.01	0.01
59	n-slim	0.48	0.35	1.11	0.36	0.37	0.25
60	onship	0.01	0.01	0.01	0.01	0.01	0.01
61	apollo-akesios	0.01	0.01	0.01	0.01	0.01	0.01
62	bioassay	0.18	0.09	5.05	1.64	0.39	1.89
63	bone-dysplasia	92.40	11.91	3.56	0.12	1.90	0.02
64	c-elegans-gross	14.92	2.62	1.33	1.34	0.35	0.29
65	cekaw	0.01	0.01	0.01	0.04	0.01	0.04
66	WSRF	0.01	0.01	0.01	0.05	0.01	0.01
67	new-1	0.01	0.01	0.01	0.01	0.01	0.01
68	cardiac-electroph	1000	545.30	67.99	237.09	15.33	38.69
69	foaf-basic	0.01	0.01	0.01	0.01	0.01	0.01

id	Ontology	JFact (sec)		Hermit (sec)		Pellet (sec)	
		time	predict	time	predict	time	predict
70	cell-line	2.49	5.69	0.50	1.29	0.35	0.80
71	clinical-meas	0.09	0.01	0.04	0.08	0.02	0.01
72	wine	0.01	0.01	0.01	0.01	0.01	0.01
73	orTI-2	1000	1000*	106.72	24.56	30.14	6.10
74	dengue-fever	3.77	0.84	7.31	6.59	3.32	1.86
75	dermlex	35.49	138.79	1.31	0.92	0.48	0.51
76	drosophila	9.71	2.96	2.30	1.58	22.54	1.24
77	Cerveau	0.07	0.05	0.01	0.01	0.01	0.01
78	eagle-i	2.15	1.09	16.49	47.94	1.78	1.02
79	event	1.81	0.32	0.56	0.44	0.95	0.23
80	event-inoh	1.16	0.24	0.35	0.40	0.63	0.07
81	n-0918	0.14	0.01	0.03	0.01	0.14	0.03
82	sp-all	0.01	0.01	0.01	0.02	0.02	0.05
83	ract2	1.26	0.53	0.49	0.52	8.70	0.48
84	fda-med-devices	0.83	0.12	0.38	0.40	0.05	0.06
85	fly-taxonomy	2.80	0.21	0.44	0.88	0.28	0.36
86	foundational-anat	1000	1000*	19.32	25.12	508.86	677.39
87	gene	214.83	95.20	14.05	4.81	5.12	0.90
88	glycomics	28.74	7.12	95.32	39.78	2.46	0.97
89	hom-dxvcodes2	21.52	1.42	1.23	1.47	0.38	0.24
90	hom-icd9-dxv	23.32	0.83	1.34	0.89	0.35	0.13
91	hom-oshpd	38.44	1.93	2.61	1.54	0.70	0.27
92	hom-ucare	0.01	0.01	0.01	0.01	0.01	0.01
93	hugo	93.32	2.80	2.56	2.11	0.55	0.35
94	human-dev	1.59	1.00	0.37	0.95	6.36	0.83
95	human-phe	10.62	0.50	0.94	0.56	0.28	0.10
96	hymenoptera	0.47	0.21	0.41	0.38	0.62	0.34
97	nursing	3.00	0.78	5.58	1.35	10.42	1.81
98	mahco	6.34	1.12	4.55	1.81	4.03	1.70
99	mouse-exp	0.01	0.01	0.01	0.01	0.01	0.01
100	mouse-anat	110.70	6.64	3.68	1.53	0.98	0.27
101	nci-thesaurus	1000	750.01	121.00	582.01	40.12	349.10
102	neural-electrom	0.64	4.38	65.48	97.19	53.89	200.43
103	neural-immune	2.15	0.84	0.52	0.85	0.44	0.25
104	nif-cell	1.03	3.92	37.89	83.80	2.18	1.60
105	nif-subcellular	0.93	4.58	27.69	115.15	2.10	1.51
106	parasite-lifecycle	0.36	0.18	0.87	1.90	1.35	0.10
107	phylogenetic	0.01	0.01	0.01	0.03	0.01	0.02
108	physico-chem	0.02	0.03	0.04	0.06	0.03	0.03
109	plant-env-cond	0.01	0.01	0.03	0.03	0.01	0.01
110	pma-2010	0.01	0.01	0.01	0.01	0.01	0.01
111	hoehndorf-mesh	1000	1000*	23.06	19.21	11.70	4.96
112	sanou	0.01	0.15	0.01	1.04	0.01	0.16
113	solanaceae-phe	0.01	0.07	0.02	0.08	0.01	0.05
114	taxonomic	0.01	0.01	0.01	0.01	0.01	0.01
115	teleost-anat	2.50	0.65	1.00	0.62	1.11	0.25
116	teleost-taxonomy	85.05	7.39	2.49	2.41	1.16	0.40
117	thesaurus-altern	0.05	0.01	0.05	0.01	0.07	0.02
118	time-event	0.76	0.63	1.65	2.56	0.45	0.22
119	zebrafish	0.85	0.25	0.34	0.36	1.96	0.08
120	vaccine	7.59	1.39	27.53	198.49	1000	260.52