Thesis Dissertation

presented in partial fulfilment of the requirements for the degree of

DOCTOR OF COMPUTER SCIENCE

at the University of Tunis El Manar (Tunisia)

and the University of Paris 8 (France)

by

Nourhène Alaya

(Master Degree in Computer Science, FST)

Managing The Empirical Hardness of The Ontology Reasoning using The Predictive Modelling

13 October 2016

Supervised by

Mr. Sadok Ben Yahia	(University Professor, Tunisia)
Mrs. Myriam Lamolle	(University Professor, France)

Jury members

Mr. Olivier Curé,	(MCF,HDR)	(reviewer)
Mrs. Hajer Baazaoui Ep. Zghal,	(Professor)	(reviewer)
Mr. Nhan Le Thanh,	(Professor)	(examiner)
Mrs. Narjes Bellamine Ben Saoud,	(Professor)	(examiner)

T H E S I S

Dédicaces

À ma chère mère, Radia, qui a tant donné, pour qu'on puisse grandir et réussir,
À mon cher père, Abdel Kader, à qui j'espère apporter joie et fièreté,
À l'homme de ma vie, Amen Eddine, mon mari, mon amour et mon partenaire dans ce projet de recherche,
À ma chère sœur, Safa,
À mon cher grand frère, Hassène,
À mon adorable petit frère, Wael,
À ma belle famille,
À tous mes amis,
À tous mes enseignants,
À tous ceux qui m'aiment.

En témoignage de mes sincères reconnaissances pour les efforts qu'ils ont consentis pour l'accomplissement de ma thèse de doctorat. Je leur dédie ce modeste travail en témoignage de mon grand amour et ma gratitude infinie.

Que Dieu vous protège tous, vous préserve la santé et le bonheur

Nourhène Alaya Mili

Acknowledgements

I would like to express my sincere appreciation to all those who helped and supported me throughout various phases of this thesis:

To Pr. **Sadok Ben Yahia** and to Pr. **Myriam Laomlle**, the thesis co-directors, who provided me the opportunity to do research at both the University of Tunis EL Manar and the University of Paris 8. I would like to thank them for believing in me, for their continuous support, patience and assistance throughout my research work. It was a great pleasure to work with them, and without their tremendous help this thesis would not be a reality. You have been tremendous mentors for me. I would like to thank you for encouraging my research and for allowing me to grow as a research scientist

To Faculty of Sciences of Tunis, Computer Sciences Department staff members, my friends and colleagues who gave me a lot of encouragement to finish this thesis.

To IUT of Montreuil, Computer Sciences Department staff members, my friends and colleagues who warmly received me in their institution and supported me throughout my thesis.

For this dissertation I would like to thank my reading committee members: Mr. **Olivier Curé**, Mrs. **Hajer Baazaoui Ep. Zghal**, Mr. **Nhan Le Thanh** and Mrs. **Narjes Bellamine Ben Saoud** for their time, interest, and helpful comments.

A special thanks to my family. Words can not express how grateful I am to my mother, and father for all of the sacrifices that you've made on my behalf. Your prayer for me was what sustained me thus far. I would also like to thank my brothers and sister and my family in law for their encouragement and unconditional support during these years. And most of all for my loving, supportive, encouraging, and patient husband **Amen Eddine**. Your are just my very special person. Your continued and unfailing love, support and understanding have give me strength and motivation to carry on my pursuit of Ph.D degree. Thank you.

Nourhène ALAYA MILI
October 2016

Abstract

Highly optimized reasoning algorithms have been developed to allow inference tasks on expressive ontology languages such as OWL (DL). Nevertheless, DL reasoning remains a challenge in practice. In fact, reasoner evaluation results are often surprising the experts as reasoner efficiency, may remarkably vary on different ontologies standing at the same computational complexity level. Furthermore, the reasoners could disagree over the inferences computed for the same input ontology. In overall, a reasoner could be optimized for some, but not all ontologies.

Given these observations, the main purpose of this thesis is to investigate means to cope with the reasoner performances variability phenomena and ways to gain insights into the ontology primary features likely to be reasoner performances degrading factors. Our final goal is to provide simple, efficiently computable guidelines to novice and expert users. The guidelines are in their basic form a set of predictions covering different aspects of the ontology reasoning. To achieve this purpose, we opted for the supervised learning as the kernel theory to guide the design of our solution. Our main claim is that the output quality of a reasoner is closely depending on the quality of the input, i.e. the ontology and machine learning technique can help us capture these dependencies. Accordingly, we first introduced a novel collection of efficiently computable ontology features, that characterise the design quality of an OWL ontology. Afterwards, we modelled two learning problems: first, predicting the overall empirical hardness of OWL ontologies regarding a group of reasoners; and then, anticipating a single reasoner robustness when inferring ontologies under some online usage constraints. To fulfil this goal, we designed a generic learning framework which integrates the introduced ontology features. The framework employs a rich set of supervised machine learning algorithms and feature selection methods. Owing to this solution, we succeeded to establish highly accurate predictive models. Furthermore, by analysing the learned models, we unveiled a set of key ontology features likely to alter the reasoner empirical robustness. In the final part of the thesis, we proposed two domain applications were the established predictive models could be integrated. First of all, we discussed the issue of reasoner automatic selection for ontology based applications. We introduced a novel reasoner ranking framework. Correctness and efficiency are our main ranking criteria. We applied bucket order rules to model the ranking of reasoners according to these criteria. Then, we proposed two distinct methods to be able to automatically predict these ranking: i) a ranking based on single label prediction, and ii) a multi-label ranking method. Afterwards, we suggested to extract from a predicted hard to manage ontology, those sub-parts that are the most computationally demanding ones. Our method mainly rely on the atomic decomposition of the ontology. We extracted syntactic locality modules using the signature of specific atoms. Then, we identified the *hard* ones using our predictive model of ontology overall hardness. All of our proposals were gathered and modelled as distinct services offered by one system, called ADSOR, the "Advisor System over Ontology Reasoners". The system was designed to allow the different learning operations within an offline stage and to interact with the user requests in an online stage. Excessive experimentations were carried out to prove the worthiness of our approaches and their competitiveness with existing solutions.

Keywords: *ontology, reasoner, empirical performance, supervised learning, prediction, hardness, robustness, ranking, extraction.*

Résumé

Le raisonnement en logique de description (LD) autour des ontologies écrites en OWL est un réel défi en raison du niveau d'expressivité élevé de ce langage. En effet, la complexité dans le pire cas de raisonnement avec une ontologie OWL2 (DL) est 2N-EXPTIME-complet. De multiples approches de raisonnement et de techniques d'optimisation ont été implémentées afin de surmonter le compromis entre la complexité des algorithmes de raisonnement et l'expressivité du langage de formulation de l'ontologie. Cependant les compagnes d'évaluation des raisonneurs continuent de confirmer l'aspect imprévisible et aléatoire des performances de ces logiciels à l'égard des ontologies issues du monde réel.

Partant de ces observations, l'objectif principal de cette thèse est d'assurer une meilleure compréhension du comportement empirique des raisonneurs en fouillant davantage le contenu des ontologies. Nous avons déployé des techniques d'apprentissage supervisé afin de capturer les relations de corrélation entre, d'une part, les attributs qui décrivent les modèles conceptuels des ontologies et d'autre part, l'historique des performances des raisonneurs. Les relations découvertes sont exploitées afin de permettre la prédiction des comportements futurs des raisonneurs. Nos propositions sont établies sous forme d'un système d'assistance aux utilisateurs afin de faire face au phénomène de variabilité de performance des raisonneurs. Quatre composantes principales ont été proposées. La première est un profileur d'ontologie qui décrit son modèle conceptuel par l'intermédiaire de 116 descripteurs. La deuxième composante est un module d'apprentissage supervisé capable d'établir des modèles prédictifs de la robustesse des raisonneurs et de la difficulté empirique des ontologies. La troisième composante est un module de ordonnancement par apprentissage des raisonneurs LD. Cette composante a été mise en place afin d'aider à la sélection du raisonneur le plus robuste étant donnée une ontologie. Nous avons proposé deux approches d'ordonnancement automatique; la première fondée sur la prédiction mono-label et la seconde sur la prédiction multi-label. La dernière composante de notre système offre la possibilité d'identifier les parties potentiellement les plus complexes d'une ontologie. Notre solution repose sur l'approche de décomposition atomique et de l'extraction de modules ontologiques de localité. L'identification des modules complexes est guidée par notre modèle de prédiction du niveau de difficulté d'une ontologie. Chaque composante du système proposée a été validée grâce à une large palette d'expérimentations. Ces composantes ont été intégrées dans un système intelligent d'assistance, intitulé "ADSOR: Advising System Over Ontology Reasoners", destiné aux utilisateurs de raisonneurs d'ontologie.

Mots clé: *ontologie, logique de description, raisonnement, apprentissage supervisé, prédiction, classement, extraction.*

List of Publications

Most of our results have been already published. In the following, we list of our publications that are related to this thesis:

International journal and book papers

Alaya Nourhène, Ben Yahia Sadok, Lamolle Myriam (2016). *Ranking with ties of OWL Ontology Reasoners based on Learned Performances.* Chapter in the Communications in Computer and Information Science Book, Springer. (To appear)

Alaya Nourhène, Ben Yahia Sadok, Lamolle Myriam (2016). Predictive Modelling of OWL Ontologies Towards a Scrutiny of the Reasoners' Empirical Behaviours. In Journal of Web Semantics Science, Services and Agents on the World Wide Web. (Submitted, Ref: JWS-D-16-00069)

International conference and workshop papers

Alaya Nourhène, Ben Yahia Sadok, Lamolle Myriam (2016). *RakSOR: Ranking of Ontology Reasoners Based on Predicted Performances.* In Proceedings of the 28th International Conference on Tools with Artificial Intelligence (ICTAI), 06-08 November, San Jose, USA. *Accepted, to appear*

Alaya Nourhène, Ben Yahia Sadok, Lamolle Myriam (2015). *Predicting the Ontology Reasoner Empirical Robustness Using Machine Learning Techniques.* In Proceedings of the 7th International Joint Conference on Knowledge Discovery, Knowledge Engineering and Knowledge Management, (KEOD-IC3K'15), pp. 61-73, Lisbon, Portugal. ***[Best Student Paper Award]***

Alaya Nourhène, Ben Yahia Sadok, Lamolle Myriam (2015). What Makes Ontology Reasoning So Arduous? Unveiling the Key Ontological Features. In Proceedings of the 5th International Conference on Web Intelligence, Mining and Semantics, (WIMS'15), 4:1–4:12, Larnaca, Cyprus.

Alaya Nourhène, Ben Yahia Sadok (2010). Revisiting large ontologies partitioning techniques. In Knowledge Engineering and Knowledge Management (EKAW 2010) workshop on Ontology Quality, Lisbon, Portugal.

National conference and workshop papers

Alaya Nourhène, Ben Yahia Sadok, Lamolle Myriam (2013) : Nouvelle mesure de degré de relation sémantique pour une meilleur modularisation d'ontologies. Actes de 24èmes Journées francophones d'Ingénierie des Connaissances, (IC'2013), Lille, France.

Alaya Nourhène, Ben Yahia Sadok, Lamolle Myriam (2012) MODLSON: une nouvelle approche de modularisation d'ontologies à grande échelle. Actes de 12e Conférence Internationale Francophone sur l'Extraction et la Gestion des Connaissances, EGC'2012, Bordeaux, France.

Alaya Nourhène, Ben Yahia Sadok (2010) Enrichissement d'ontologies passant à l'échelle à partir de données non-structurées. Actes de la 21es Journées francophones d'Ingénierie des Connaissances (IC'2010), Nîmes, France.

Contents

List of Tables

List of Figures

List of Algorithms

Résumé détaillé

Contents

0.1 Introduction

Lorsque Berners-Lee et al. [BLHL01] ont formulé la notion de Web sémantique, ils ont porté une grande attention aux ontologies. En effet, une ontologie peut fournir un vocabulaire commun, une grammaire de publication de données, offrir une description de pages web, etc. Dès lors, les ontologies sont utilisées comme des modèles conceptuels pour l'intégration de données dans des domaines très divers. Moyennant les ontologies, le futur Web sémantique devrait représenter l'information de telle façon qu'elle soit beaucoup plus porteuse de sens pour les humains mais aussi pour les machines.

Une ontologie est typiquement encodée dans un langage de représentation des connaissances (KR). Une classe majeure des langages d'ontologie est fondée sur les logiques des descriptions (DLs) [BCM+10] qui sont des fragments décidables de la logique du premier ordre (FOL[1]). Le langage d'ontologie le plus notablement connu dans cette classe est le langage OWL [HPSV03]. OWL est très utilisé dans les mondes académique et industriel, sachant que c'est une recommandation du W3C (World Wide

[1]First Order Logic

Web Consortium) comme le standard de création d'ontologie. Une ontologie exprimée en OWL peut être vue comme un ensemble de formules FOL, appelées *axiomes*, et donc comme une théorie logique. Par conséquent, le vocabulaire exprimé en OWL est bien défini et sans ambiguïté. Cependant, il y a souvent un décalage entre la représentation visée et la représentation réelle de la connaissance décrite dans l'ontologie. Alors que la connaissance du domaine d'intérêt est généralement bien maîtrisée ou tout au moins comprise, il n'est pas trivial de prévoir et de comprendre les conséquences logiques d'un ajout, d'un retrait ou d'une modification d'axiome, notamment quand la terminologie est fortement interconnectée. Ainsi, un ingénieur en ontologie a besoin d'utiliser des systèmes de déduction automatisés, communément appelés *raisonneurs*, pour vérifier certaines formes de déduction à partir des axiomes préétablis et de rendre explicites ceux qui sont déclarés implicitement. Les *raisonneurs* sont donc, à l'heure actuelle, des éléments clés pour travailler avec des ontologies. Ayant mis en évidence l'importance et le rôle crucial de ces derniers, beaucoup d'efforts en recherche et en ingénierie ont été faits pour proposer de nouveaux algorithmes de raisonnement sémantique [BS01, MSH09, SKH11] avec des optimisations conséquentes, tels que FaCT++ [TH06], Pellet [SPG+07] et HermiT [GHM+12]. Matentzoglu et al. [MLH+15] ont récemment analysé cet effort dans le domaine de la recherche dans une étude détaillée.

Challenges

Malheureusement, la prolifération des ontologies a un coût. Elles continuent à gagner en importance, que ce soit par leur taille et/ou par leur complexité ; ce qui outrepasse les capacités cognitives de tout homme pour comprendre et pour manipuler manuellement les connaissances modélisées. Cette complexité cognitive a entraîné une dépendance vis-à-vis des raisonneurs pour contrôler et gérer ces ontologies. De plus, les raisonneurs, eux-aussi, ont gagné en complexité. En fait, il existe un réel compromis entre l'expressivité logique du langage des ontologies et la complexité du raisonnement DL ; plus le langage est expressif, plus sa complexité computationnelle sera importante. Horrocks et al. [HKS06] ont montré que les problèmes de raisonnement sur des ontologies écrites en $\mathcal{SROIQ}$, soit la logique de description qui est le fondement de OWL 2[2] sont, dans le pire des cas résolus par un algorithme non déterministe en un temps (doublement) exponentiel dans le respect de la taille de l'ontologie donnée en entrée. Par contre, différents travaux, dont [MRW11, Abb12], ont montré qu'en général, le raisonnement est faisable en pratique et beaucoup moins complexe qu'en théorie. Cet état de fait serait dû à diverses techniques d'optimisation sophistiquées du raisonnement [Hor03, THPS07]. Néanmoins, les auteurs respectifs de [WLL+07, GPS12] ont souligné le fait qu'en pratique aussi, les raisonneurs ont tendance à présenter des comportements *imprévisibles* quand ils travaillent avec des ontologies du monde réel. Leurs performances varient énormément d'une ontologie à l'autre, même

[2]Deuxième version de OWL, OWL 2 [Gro09], est une recommandation du W3C depuis 2009

quand la taille ou l'expressivité de ces dernières restent constantes. En effet, les observations parcimonieuses existantes concernant les performances des raisonneurs ont incité l'organisation de plusieurs bancs d'essai de raisonneurs [DCtTdK11, Abb12] et des compétitions d'évaluation annuelles, telle que "Ontology Reasoner Evaluation Workshop" (ORE) [GBJR+13, BGJ+14, DGG+15]. Tous essaient de comparer les avancées dans le domaine des systèmes de raisonnement et examinent la robustesse de ces derniers face à des ontologies du monde réel. Les résultats empiriques de ces campagnes d'évaluation surprennent souvent les novices et les experts. Les principaux constats ont été résumés par Gonçalves et al. [GPS12] comme suit: i) étant donnée une même ontologie, écarts énormes de temps de raisonnement apparaissent ; ii) performances aléatoires d'un raisonneur pour un ensemble d'ontologies partageant les même caractéristiques ; iii) échecs de tous les raisonneurs à traiter une certaine catégorie d'ontologies. En outre, il a été constaté par Gardiner et al. [GTH06] et plus récemment Lee et al. [LMPS15] que les inférences calculées pour une même ontologie peuvent variées d'un raisonneur à un autre, ce qui remet en question l'exactitude ainsi que l'exhaustivité de leurs méthodes de raisonnement. D'une manière générale, il est largement admis qu'il n'y a pas de moteur de raisonnement qui puisse surclasser tous les autres pour toute catégorie d'ontologie. Il y a simplement un consensus [GTH06, WLL+07, WP07, GPS12, GMPS13, LMPS15] établissant que la complexité empirique du raisonnement autour des ontologies est en relation avec le phénomène de variabilité des performances de ces outils. Ce problème rend l'utilisation des raisonneurs assez fastidieuse, ce qui risque de freiner leur prolifération et par conséquence, le développement du Web Sémantique. Dans cette thèse, nous nous sommes focalisés sur deux principaux aspects de ce phénomène à savoir (i) la variabilité du temps de calcule en fonction de l'ontologie en cours de traitement, (ii) la disparité des déductions calculées par différents raisonneurs sur une même ontologie.

Face à ces constats, il est réaliste d'affirmer que les qualités les plus désirées, i.e. la justesse du résultat et l'efficacité temporelle, ne sont pas garanties empiriquement par tous les raisonneurs et pour toutes ontologies. Un raisonneur pourrait être optimisé pour certaines ontologies mais pas pour toutes. Ces observations identifient la difficulté de compréhension des comportements empiriques des raisonneurs même pour des concepteurs de raisonneur expérimentés et compétents. Comme l'ont précédemment soutenu Wand and Parsia [WP07], le problème réel avec le raisonnement LD est le manque de théorie et d'outil d'assistance aidant tout aussi bien un novice qu'un expert pour analyser les performances empirique des raisonneurs. En outre, faute de manque de connaissances, les règles de décision préétablies par les experts concernant le bien fondé d'un raisonneur pour une ontologie particulière sont moins effectives en pratique, excepté pour des cas triviaux.

Partant de ces constatations, dans cette thèse, nous allons développer et de nouveau examiner deux questions de recherche fondamentales à nos yeux pour un utilisateur final à savoir :

1. Quelles caractéristiques des ontologies sont susceptibles d'avoir un impact sur les performances d'un raisonneur ?
2. Est-ce que c'est faisable de raisonner sur une ontologie écrite en une LD particulière, en utilisant les meilleurs raisonneurs disponibles ?

Cette seconde question pourrait également être détaillée. Premièrement, nous pourrions nous demander s'il est possible *a priori* de prédire les performances de tout raisonneur sur une ontologie quelconque. Ensuite, comment pourrions nous assister les utilisateurs finaux afin qu'ils obtiennent une sélection automatique des raisonneurs les plus *"appropriés"* pour leurs ontologies ?

Objectifs et contributions

L'objectif principal de cette thèse est d'apporter des solutions pour faire face au phénomène de variabilité des performances des raisonneurs ; et donc, de faire au préalable un état de l'art de l'analyse empirique des raisonneurs DL. Nous croyons que fournir un guide sur les comportements empiriques des raisonneurs pourrait améliorer davantage leur optimisation et la conception d'ontologie en évitant les facteurs dégradant les performances de ceux-ci.

Pour apporter des solutions à notre première question, nous avons mené une investigation sur les méthodes et les outils existants destinés à identifier une corrélation potentielle entre le comportement empirique d'un raisonneur et les caractéristiques particulières d'une ontologie. Nous avons essayé de donner aux utilisateurs une vue globale au travers d'un état de l'art dans ce domaine. Les tout premiers travaux ont été analysés et ensuite catégorisés. Nos investigations nous ont amené à penser que la complexité de conception d'une ontologie est le résultat d'une combinaison complexe de multiples caractéristiques dimensionnelles et ne peut être mesurée directement en utilisant une unique métrique. Aussi, nous avons proposé un ensemble de caractéristiques d'ontologie couvrant un large spectre des propriétés structurelles et syntaxiques d'ontologie. Nous revendiquons le fait que ces caractéristiques devraient être des indicateurs clés du niveau de difficulté d'une ontologie par rapport aux tâches de raisonnement.

Notre seconde question concerne deux problèmes : (i) la prédiction des comportements empiriques d'un raisonneur et (ii) la sélection automatique du raisonneur adéquat étant donné une ontologie. La prédiction de la performance d'un raisonneur a aussi été étudiée par Gonçalves et al. [GPS12]. Ils ont suggéré qu'une voie possible pour expliquer pourquoi un raisonneur fonctionne mal sur une certaine ontologie est de prédire si l'ontologie contient certains composants coûteux informatiquement parlant. Leur principale revendication est que si le temps exigé pour classer une ontologie par un raisonneur particulier n'est pas corrélé linéairement à la taille de ses sous-ensembles, alors l'ontologie est dite hétérogène en performance pour ce raisonneur. Dans ce cas, il est vraisemblable que certaines sous-parties de l'ontologie, *i.e.* les modules, sont des *"zones sensibles"* (dit "hotspots") de la performance du raisonneur. Cependant, les expérimentations ont révélé qu'il n'y a pas de corrélation précise entre le temps de raisonnement sur un seul *hotspot* et la réduction en temps de raisonnement quand un tel *hotspot* est enlevé de l'ontologie globale. Ils ont affirmé que d'autres recherches devraient être faites sur des interactions possibles entre hotspots et d'autres caractéristiques de l'ontologie. Une autre plus-value de ces travaux, principalement décrite dans [KLK12, SSB14, KLK14], a utilisé des techniques d'apprentissage supervisé (SML) [Kot07] visant à prédire le temps de calcul d'un raisonneur sur n'importe quelle ontologie. Leurs modèles prédictifs s'appuient sur un grand ensemble de métriques ontologiques pré-calculées. La rationalité de ce choix est d'être capable d'apprendre automatiquement les comportements futurs d'un raisonneur fondé sur ce qui a été éprouvé dans leurs exécutions passées. Les approches de prédiction des performances d'un raisonneur sont encore à leur balbutiement mais ont montré des résultats très prometteurs dans différents champs d'application. Par exemple, les auteurs respectifs de [KLK14, SSB14] ont déployé leurs modèles de prédiction du temps d'exécution de raisonneur pour identifier les *hotspots* d'ontologie. Par ailleurs, nous tenons à souligner que l'idée d'apprentissage pour prédire le temps d'exécution d'un algorithme à partir de données empiriques n'est pas nouveau. Après plus d'une décennie de recherche, Brown et al. [LBHHX14] ont démontré que le temps d'exécution des solveurs sur des problèmes NP-complet SAT peut être prédit précisément fondé sur la facilité de calculer des caractéristiques et en employant des techniques SML. Hutter et al. [HXHLB14] ont effectué une large étude récapitulant tous les travaux pionniers essayant d'apprendre des *modèles empirique de performance* (EPMs) d'algorithme. Nous constatons en cela que les approches basées apprentissage automatique sont d'un grand intérêt pour satisfaire la demande de dépistage des phénomènes de variabilité de performance de raisonneur. En effet, les méthodes d'apprentissage sont génériques et pourraient donc manipuler n'importe quel raisonneur ou n'importe quelle ontologie. D'autre part, elles ont un large éventail d'application. Néanmoins, les solutions existantes de prédiction de performance des raisonneurs sont encore limitées puisque toutes évaluent seulement le temps d'exécution des raisonneurs. Pourtant, selon nous, un raisonneur qui vérifie rapidement mais incorrectement la cohérence d'une ontologie est de peu d'intérêt. Dans cette thèse, nous proposons d'apporter une pierre supplémentaire à la modélisation prédictive des comportements empiriques de raisonneur en recherchant également la qualité des résultats des raisonneurs et en identifiant les ontologies qui vont probablement dégrader ces qualités. Plus spécifiquement, nous essaierons de prédire la correction des raisonneurs ainsi que leur efficacité. Pour terminer, nous établirons une spécification détaillée d'un cadre de travail générique d'apprentissage capable de réaliser plusieurs tâches prédictives de modélisation pour du raisonnement sur des ontologies. Ce cadre utilise la suite proposée de caractéristiques d'ontologie et un large ensemble d'algorithmes d'apprentissage supervisé couplé à de multiples techniques de sélection de caractéristiques. En utilisant ce cadre de travail, nous commencerons à entraîner un modèle capable de prédire si une ontologie est *difficile empiriquement* selon un groupe de raisonneurs. Puis, les performances des ontologies ingérables pourraient être identifiées. Cherchant une analyse plus fine, nous essaierons de décrire la robustesse de chaque raisonneur en combinant des modèles conditionnels prédictifs de sa justesse et de son efficacité. Cette association particulière matérialise notre concept de *profil prédictif de robustesse* de raisonneur. Nous montrerons que, fondés seulement sur des caractéristiques d'ontologie simples et faciles à calculer, les modèles de grande précision traitant à la fois la *difficulté* des ontologies et la *robustesse* des raisonneurs devraient être appris par une exploration intelligente des données empiriques. Enfin, nous fournirons des indications pour interpréter les différents modèles prédictifs entraînés dans le but de découvrir de nouvelles relations entre les caractéristiques d'ontologie et la qualité du raisonneur ; et, par là, favoriser notre compréhension des caractéristiques clés d'ontologie probablement source de la dégradation de la robustesse empirique des raisonneurs. Notre hypothèse est que, plus efficaces sont les caractéristiques dans la description de la complexité de conception des ontologies, plus exacts seront les modèles prédictifs entraînés.

Notre interrogation suivante est liée au problème déjà ancien de sélection d'algorithme, discuté la première fois par John Rice [Ric76] au milieu des années 70. Ce sujet est à la croisée de beaucoup de domaines de problème, tels que l'apprentissage machine (ML) et l'intelligence artificielle (AI). Un trait commun parmi les solutions proposées est qu'elles emploient des techniques d'apprentissage supervisé, de régression ou de classification de ML[3], pour construire un système prédictif efficace des performances des algorithmes candidats pour chaque instance de problème. Par conséquent, le plus adéquat devrait être identifié. Le méta-apprentissage [BGCSV08] et la construction d'un algorithme portfolio [GS01] sont les champs principaux où le modèle de Rice d'algorithme de sélection ont été mis en pratique. Alors que les approches de méta-apprentissage fondamentalement tendent à recommander le meilleur algorithme ML selon un ensemble de données fourni, le but final des approches portfolio est de tirer profit de la complémentarité des algorithmes en les combinant dans le but d'obtenir des performances éprouvées dans le cas moyen. L'idée de construire un portfolio de raisonneurs a séduit Kang et al. [KKL15]. Ces derniers ont utilisé leurs modèles prédictifs de temps d'exécution de raisonneurs précédemment entraînés pour construire R_2O_2, une sorte de *méta-raisonneur* qui combine les raisonneurs existants et essaie de déterminer le plus efficace pour une ontologie donnée. Leurs résultats ont été remarquables, prétendant que R_2O_2 pourrait surclasser le plus efficace des raisonneurs. Cependant, aucune évaluation rigoureuse sous condition particulière n'a été conduite pour confirmer les assertions

[3]Il est important de noter que la "classification DL" représente pour une tâche de raisonnement la tendance à inférer des relations de subsumption entre les classes et les propriétés pour une ontologie donnée, alors que la "classification ML" dénote le processus d'entraînement d'un modèle prédictif à partir d'un ensemble de données étiquetées.

de Kang et al. En fait, le temps supplémentaire induit par les étapes de prédiction n'est pas pris en compte. Dans cette thèse, nous avons choisi de nous intéresser au problème de la sélection de raisonneur d'ontologie selon un point de vue différent. Nous croyons qu'avoir des raisonneurs sélectionnés comme étant les plus adaptés pour une ontologie donnée est un type de recommandation qui serait faite pour des utilisateurs cherchant à être conseillés. Le conseil fourni devrait être, dans sa forme basique, un classement des raisonneurs dans l'ordre de préférence. Comme précédemment souligné par [PdSL11], fournir un classement des algorithmes candidats est une solution plus informative et flexible que suggérer le meilleur algorithme. Par exemple, un utilisateur peut décider de continuer à utiliser son raisonneur favori si sa performance est légèrement en dessous que le meilleur dans le classement. Pour réaliser ceci, nous avons formalisé un nouveau cadre de classement de raisonneurs qui prend en compte la robustesse des raisonneurs candidats, et plus précisément leur niveau de correction et d'efficacité pour une ontologie donnée. Nous avons étudié deux approches distinctes de classement : une première méthode qui permet des liens et suit les règles des principes du "bucket order" et une méthode basée classement de labels qui applique les règles de préférence d'apprentissage. Nous avons prouvé expérimentalement le mérite de chaque approche proposée et avons discuté de leurs avantages et de leurs inconvénients. A notre connaissance, nous sommes les premiers à employer plus d'un critère de classement de raisonneurs et nous croyons que c'est une étape importante dans le classement multi-critères des raisonneurs d'ontologie.

Au delà de la sélection de raisonneur, nous avons proposé un autre type de conseil pour les utilisateurs finaux cherchant à gagner en compréhension de leurs ontologies. En effet, quand une ontologie est reconnue comme étant difficile à gérer, un utilisateur cherchera à détecter les causes de cette difficulté. Dans ce cas, nous suggérons de lui fournir la possibilité d'extraire les sous-parties de son ontologie qui sont potentiellement les plus exigeantes en terme de coût de raisonnement. Comme décrit ci-dessus, ces sous-parties sont appelées des *hotspots* d'ontologie [GPS12]. Les hotspots sont des goulots d'étranglement de performance lors du raisonnement, et ils représentent des opportunités de remaniement pour les concepteurs d'ontologie. Aussi, une identification efficace de ces composants est hautement nécessaire. Nous avons revisité les approches respectives de Gonçalves et al. [GPS12] et Kang et al. [KLK14] d'extraction d'hotspot. Nous avons suggéré une nouvelle méthode qui repose sur la représentation de graphe de dépendance des axiomes d'une ontologie. Notre extraction tire avantage du modèle général entraîné de difficulté d'ontologie et elle est guidée par les caractéristiques de l'ontologie reconnues comme clés et ayant probablement un impact sur la robustesse du raisonneur. Nos premiers résultats expérimentaux sont encourageants, témoignant du potentiel de cette méthode.

Un premier aperçu

Pour résumer, le propos principal de cette thèse est d'examiner les moyens à mettre en oeuvre pour faire face aux phénomènes de variabilité de performances de raisonneur et les voies possibles pour gagner en compréhension sur les caractéristiques principales d'ontologie dégradant probablement les facteurs de performance des raisonneurs. Notre but final est de fournir des orientations simples, calculables efficacement pour des novices et des experts concernant, d'une part, la difficulté empirique des ontologies OWL et, d'autre part, la robustesse empirique des raisonneurs DL modernes. Ces orientations sont basiquement un ensemble de prédictions sur les comportements attendus d'un raisonneur ou un classement d'un groupe de raisonneurs pour une ontologie donnée. Nous croyons que la délivrance de telles indications à un nouvel utilisateur ayant peu d'expérience dans les algorithmes sophistiqués de raisonnement devrait être une solution utile lui faisant économiser un temps précieux en recherche et en analyse. Mais c'est aussi un point de départ également utile pour les experts pour conduire des examens approfondis sur leurs ontologies ou leurs raisonneurs. Pour terminer ce propos, nous avons opté pour l'apprentissage supervisé comme théorie centrale pour guider la conception de notre solution.

Ce mémoire est structuré selon l'étude expérimentale qui a été effectuée. De façon plus détaillée, les chapitres ultérieurs sont centrés sur les sujets suivants :

Chapitre 1 jette les bases de la terminologie employée et les notions de fond centrales utilisées tout au long de cette thèse. Nos questions de recherche sont à la croisée de deux champs disciplinaires : le Web sémantique [BLHL01] et l'apprentissage machine [Sam59]. C'est pourquoi nous rappelons quelques unes de leurs notions fondamentales. En particulier, nous nous focalisons sur les trois notions clés à savoir les ontologies, les raisonneurs et l'apprentissage supervisé.

Chapitre 2 examine les travaux liés au notre. Nous commencerons par aborder les niveaux de complexité de modélisation des ontologies OWL et la façon de caractériser sa conception. Puis, nous focaliserons notre étude sur le phénomène de variabilité de performances des raisonneurs. Nous allons mettre en place des méthodes de comparaison et des outils qui essaient de le détecter et l'analyser. La dernière section de ce chapitre sera d'ailleurs dédiée aux avancées s'adressant au problème de sélection d'algorithme. Nous examinerons les approches connues dans le méta-apprentissage et la construction de portfolio d'algorithme.

Chapitre 3 décrit en détail notre ensemble de caractéristiques proposées pour classer une ontologie. Ces caractéristiques sont classées en quatre catégories de base et couvrent un large spectre des propriétés structurelles et syntaxiques des ontologies. Nous employons nos caractéristiques pour décrire des ontologies OWL sélectionnées à partir d'un corpus bien connu. Enfin, nous avons mené une comparaison sur le coût computationnel de mesure de nos caractéristiques dans le but d'identifier celles qui sont les plus efficacement calculables.

Chapitre 4 détaille notre méthodologie d'apprentissage de raisonneur qui dépend de notre riche ensemble de caractéristiques avancées pour ontologie. Grâce aux avantages de cette proposition, nous avons modélisé le profil de robustesse d'un raisonneur, en terme de modèles prédictifs de sa correction et son efficacité. Pour démontrer le mérite de nos propositions, nous avons conduit un examen rigoureux à partir de six raisonneurs bien connus et sur 2 500 ontologies. Nous avons comparé divers algorithmes d'apprentissage supervisé et des techniques de sélection de caractéristique dans le but d'entraîner les meilleurs modèles de prédiction de raisonneur. Les modèles prédictifs entraînés ont montré qu'il sont des solutions surpassant grandement les autres solutions présentées dans l'état de l'art.

Chapitre 5 décrit les deux domaines d'application où les modèles prédictifs établis pourraient être intégrés. Les deux font partie d'un système générique que nous avons nommé ADSOR pour "Advising System Over Ontology Reasoners". Le propos principal de ce système est de fournir une assitance aux utilisateurs finaux pour une sélection facile du raisonneur le plus approprié pour leur ontologies basées applications. De plus, le système aide les utilisateurs pour gagner en compréhension sur la difficulté globale de leur ontologie. Il suggère de prédire cette difficulté et, plus intéressant, il offre la possibilité d'extraire ses sous-parties les plus coûteuses en temps, connues comme des *hotspots* d'ontologie. Les diverses approches qui fournissent les bases pour la conception centrale de ce systèmes sont détaillées dans ce chapitre. Le mérite de chaque approche a été prouvé expérimentalement et analysé en détail.

Enfin, nous concluons ce mémoire en résumant nos principales contributions et les résultats pertinents, en pointant quelques questions ouvertes et les futures directions.

0.2 Résumé du chapitre 1 : Un peu de théorie

Dans ce chapitre, nous avons présenté les concepts relatifs à notre problématique de recherche. Ces concepts ont été introduits en partie pour rendre ce mémoire plus indépendant et, particulièrement, pour familiariser le lecteur aux notations formelles qui seront utilisées tout au long de cette thèse.

Notre problématique de recherche se trouve à la croisée de deux champs : le Web sémantique [BLHL01] et l'apprentissage automatique (dit *Machine Learning*)[Sam59]. Nous avons exposé, en premier lieu, les notions fondamentales du Web sémantique. Puis, nous avons présenté l'étude des ontologies du point de vue de la représentation des connaissances. C'est pourquoi, nous avons rappelé les caractéristiques principales du langage de création d'ontologie OWL. En particulier, nous avons étudié les logiques de description (LD) dont OWL est dérivé. Les LDs sont particulièrement bien adaptées pour la représentation des ontologies et elles sont pertinentes pour le Web sémantique car elles ont des sémantiques précises et permettent le raisonnement automatique. Les LDs offrent de riches caractéristiques d'expressivité et différents degrés de logiques. Cependant, un compromis devrait être fourni entre l'expressivité DL et la complexité de raisonnement. En effet, les divers moteurs de raisonnement ont été présentés dans la littérature pour traiter les ontologies de la façon la plus fiable et efficace possible. Nous avons établi une comparaison de leurs principaux attributs fonctionnels afin de dresser une idée générale de la couverture et des tendances actuelles autour du développement des raisonneurs d'ontologies.

Dans la seconde partie de ce chapitre, nous avons passé en revue les notions de base du champ d'apprentissage automatique. Nous rappelons que nous avons l'intention d'utiliser l'approche d'apprentissage supervisée pour faire face aux phénomènes de variabilité de performance des raisonneurs. Pour satisfaire cette exigence, nous avons rappelé les notations formelles habituelles. Puis, nous avons décrit les étapes générales de tout processus d'apprentissage. Ensuite, nous avons passé en revue les caractéristiques principales des algorithmes d'apprentissage automatique supervisé et des méthodes de sélection de variables les plus généralement utilisés. Nous avons finalement décrit les façons d'évaluer la qualité des modèles prédictifs.

Ontologie

T. R. Gruber [Gru93] a défini une ontologie comme "une spécification formelle, explicite d'une conceptualisation partagée". Une ontologie délimite un vocabulaire commun que les utilisateurs peuvent partager dans un domaine de connaissance. Les ontologies sont devenues très importantes dans la mouvance globale du Web sémantique. Elles sont utilisées comme des modèles conceptuels pour l'intégration de données ou pour représenter directement de l'information dans des champs disciplinaires très variés. Les ontologies sont disponibles sur le Web pour une multitude de machines afin de favoriser les échanges et partager la connaissance. Aussi, beaucoup de raisons devraient motiver le développement d'ontologie ; par exemple, pour :

- partager la compréhension commune de la structure de l'information parmi les personnes ou les agents logiciels,
- permettre la réutilisation d'un domaine de connaissance,
- rendre explicite les hypothèses d'un domaine,

- séparer la connaissance du domaine de la connaissance opérationnelle,
- analyser le domaine de connaissance.

Le potentiel des ontologies, à savoir fournir des domaines avec un vocabulaire non ambigu, a attiré l'intérêt de beaucoup d'experts de domaine provenant de champs divers tels que la médecine, la bio-informatique, la géographie, etc. De gros efforts ont été faits pour modéliser et maintenir les ontologies. Un exemple notable de ce processus est le thésaurus *NCI-t* de l'Institut National du Cancer[4]. NCI-t a été mis à jour mensuellement depuis 2003 et beaucoup de versions de cette ontologie sont téléchargeables gratuitement. D'autres ontologies médicales d'importance pourraient aussi être mentionnées : *Snomed Ct*[5], le projet *Open Galen* [6] et *FMA*[7].

Les ontologies sont typiquement encodées dans un langage de représentation des connaissances. Une classe importante des langages d'ontologie est fondée sur la famille des logiques de description (DLs) [BCM+10], qui sont généralement des fragments décidables de la logique du premier ordre (FOL). Par conséquent, une ontologie peut être vue comme un ensemble de formules FOL, dites *axiomes*. C'est donc une théorie de la logique. Un exemple notable de langages d'ontologie fondé sur les DLs est le langage OWL (*i.e.* *Web Ontology Language* [HPSV03], un standard du W3C qui a été utilisé à la fois par le monde académique et le monde industriel depuis que sa première version est devenue une recommandation du W3C en 2004.

Raisonneur

Un aspect central de la représentation des connaissance basée logique est la capacité à employer (efficacement) des systèmes de déduction automatisés. En DLs, ces systèmes, appelés raisonneurs, implémentent des procédures de décision (par exemple, l'*algorithme Tableau* [BS01], *Hyper-Tableau* [MSH09] et *Consequence-Based* [SKH11]). Ils déduisent des conséquences logiques à partir d'un ensemble de faits ou d'axiomes explicitement affirmés et fournissent typiquement un support automatisé pour les tâches de raisonnement [BCM+03].

Les raisonneurs DL modernes supportent un nombre de services logiques primaires tels que la cohérence, la classification, le contrôle d'instance et le contrôle de l'implication logique, et des service secondaires tels que l'explication pour l'implication logique et l'incohérence, l'extraction (sémantique) de module et les accès aux données basées ontologies (OBDA).

[4]`https://wiki.nci.nih.gov/display/VKC/NCI+Thesaurus+Terminology`
[5]Plus de détails sur *Snomed Ct* sur le site `http://www.ihtsdo.org/`
[6]voir `http://www.opengalen.org/`
[7]voir `http://sig.biostr.washington.edu/projects/fm/AboutFM.html`

Propriété **Catégories**	**FacT++**	**JFacT**	**Pellet**	**TrOWL**	**ELK**	**HermiT**	**MoRE**	**Konclude**
Langage de programmation	C++	Java	Java	C++	Java	Java	Java	C++
Interface de programmation	OWL+Jena	OWL	OWL	OWL+Jena	OWL	OWL	OWL	OWL+Jena
Éditeurs supportés	PG	PG+NT	PG+NT	PG	PG	PG+NT	PG	∅
Open source	Oui	Oui	Oui	Non	Oui	Oui	Oui	Oui
Vérification ABox	Oui	Oui	Oui	Oui	Non	Oui	Non	Oui
Support des types de données	limité	limité	Oui	limité	Non	Oui	Oui	Oui
Justification	Non	Non	Oui	Non	Non	Non	Non	Non
Support des règles	Non	Non	Oui	Non	Oui	Oui	Non	Oui
Profil natif	DL	DL	DL,EL	Dl,EL	EL	DL	DL,EL	DL
Expressivité	$\mathcal{SROIQ}$	$\mathcal{SROIQ}$	$\mathcal{SROIQ}$	$\mathcal{SROIQ}$	$\mathcal{EL}+$	$\mathcal{SROIQ}$	$\mathcal{SROIQ}$	$\mathcal{SROIQV}$
Méthodologie	MC (T)	MC (T)	MC (T)	CB	CB	MC (HT)	HM (MOD+MC)	HM (CB+MC)
Correction/ Complétude	SC	SC	SC	SC/ (EL+QL)	SC/ (EL)	SC	SC	SC

Table 1: Comparaison des attributs fonctionnels des raisonneurs DL.

Un raisonneur est un programme qui infère des conséquences logiques à partir d'un ensemble de faits ou d'axiomes explicitement avérés et fournit typiquement un support automatisé pour les tâches de raisonnement telles que la classification , le débogage et le requêtage. Tout raisonneur pourrait être distingué selon un ensemble important d'attributs fonctionnels tels que l'algorithme d'inférence, la logique supportée, le degré de complétude du raisonnement, le langage d'implémentation, etc. Dans cette section, nous essaierons d'établir les caractéristiques généralement exhibées des raisonneurs DL modernes. Plutôt que de comparer les performances des raisonneurs, nous discuterons des caractéristiques fonctionnelles pertinentes de ces implémentations et essaierons de les classer dans des catégories. En accord avec l'étude la plus récente sur les raisonneurs d'ontologie établie par Matentzoglu et al. [MLH+15], il existe pratiquement 35 raisonneurs d'ontologies connus dont la plupart ont été développés entre 2010 and 2015. Ces réalisations pourraient être catégorisées selon des critères variés. La Table 1.4 synthétise les caractéristiques fonctionnelles détaillées précédemment pour ces raisonneurs. La comparaison a été mise en place selon les critères de classification introduits. Dans ce tableau, (OWL) représente OWL API et (J) Jena API. Par (PG), nous faisons référence à l'éditeur Protégé et par (NT) à Neon-Toolkit.

A l'issu de cet examen, il parait évident que les raisonneurs d'ontologies varient de façon significative au regard des tous les attributs que nous avons examinés. Cette richesse en terme d'approche de raisonnement, de capacité et de couverture fonctionnel rend difficile la tâche de sélection d'un raisonneur pour une application basée sur les ontologies. En outre, on ignore encore l'impact de ces attributs fonctionnels sur les performances empiriques des raisonneurs. Ainsi, il est certes nécessaire d'établir des évaluations empiriques critiques des performances de ces raisonneurs afin de pouvoir juger avec certitude de leurs robustesse, spécialement à l'égard des ontologies issues du monde réel.

Apprentissage automatique

La modélisation prédictive [HTF01] est le nom donné à une collection de techniques mathématiques et statistiques ayant comme but commun de trouver des patrons cachés dans les données et de développer des fonctions capables de prévoir les tendances et les résultats futurs avec un degré acceptable de fiabilité. En d'autres mots, la modélisation prédictive est un concept général d'analyse des données réelles dans le but de construire des modèles qui sont capables de faire des prédictions sur un domaine d'intérêt. Typiquement, un tel modèle inclut *des algorithmes d'apprentissage automatique*, dit *machine learning* (ML).

La présentation générale du processus de ML est donnée ci-dessous (plus de détails sont fournis dans [Abb14]) :

Collection de données : les données liées au domaine d'étude sont extraites. Elles seront dans un format brut et non structuré. D'où, des mesures de pré-traitement doivent être adoptées.

Pré-traitement de données : l'ensemble de données (dit *dataset*) initial est soumis à un pré-traitement. Le pré-traitement est exécuté pour enlever les données non pertinentes et/ou redondantes. Les valeurs manquantes sont remplacées par normalisation et transformation.

Développement de modèle : les données pré-traitées sont soumises à différents algorithmes ML pour le développement de modèles. Ces modèles sont construits à partir de techniques de classification ou bien de clustering ou encore à partir des règles d'association.

Extraction de connaissance : les modèles sont évalués pour représenter la connaissance capturée.

En pratique, la tâche de modélisation prédictive décrite précédemment a une importance pour beaucoup de domaines, de la médecine aux sciences sociales en passant par divers champs de l'ingénierie.

En se référant à [Kot07], nous introduisons quelques concepts de base du *machine learning* (ML). Explicitement, chaque cas dans tout dataset utilisé par des algorithmes ML est représenté en utilisant le même ensemble de caractéristiques. Puis, chaque cas est un vecteur M-dimensionnel

Cas	**Caractéristique1**	**Caractéristique2**	**...**	**Caractéristique$M$**	**Classe**
1	XX	XX	...	XX	bon
2	XX	XX	...	XX	mauvais
...					
N	XX	XX	...	XX	bon

Table 2: Exemple de dataset pour l'apprentissage supervisé [Kot07]

$x^{(i)} = [{x_1}^{(i)}, {x_2}^{(i)}, \ldots, {x_M}^{(i)}]$ appelé un *vecteur de caractéristiques*, où i réfère le i-ème cas du dataset, $i \in [1 \cdots N]$ avec N le nombre total de cas et M le nombre total de caractéristiques. Nous utiliserons aussi $\mathcal{X}$ pour noter l'ensemble de caractéristiques contenant N vecteurs de caractéristiques, sinon l'espace des valeurs d'entrée. Les caractéristiques peuvent être continues, catégorielles ou binaires. Dans ce mémoire, nous nous sommes intéressés spécialement à l'apprentissage supervisé. Le vecteur de tous les labels est noté $\mathcal{Y}$, où y_i est le label du i-ème cas. Ainsi, le dataset pourrait être noté par $\mathbb{D} = [\mathcal{X}_{N,M} \mid \mathcal{Y}^T]$. Un exemple d'un dataset pour un apprentissage supervisé est donné dans la Table 1.5. Un problème supervisé est dit de *classification* quand les labels sont qualitatifs (*e.g.* bon/mauvais) et de *régression* dans le cas de valeurs quantitatives (*e.g.* valeur de température). Ici, il est important de noter la différence entre la tâche de "classification" en ML et la "classification" décrite précédemment dans le contexte du raisonnement. Pour éviter toute ambiguïté, nous nous référerons à cette dernière par la dénomination "classification DL" et à la première par "classification ML". Quand le vecteur de label, $\mathcal{Y}$, contient seulement deux labels discrets, alors le problème de classification ML est appelé *classification ML binaire*, sinon c'est un problème de *classification ML multi-classes*.

Soit le dataset d'entraînement $\mathbb{D}$, une tâche d'apprentissage supervisée est le processus de déduction d'une fonction souvent mentionnée comme une hypothèse ou un modèle, $\mathcal{F} : \mathcal{X} \to \mathcal{Y}$, qui prédit le "*meilleur*" label y à partir du vecteur de caractéristiques x pour toute paire (x, y) tirée de $\mathbb{D}$. Les fonctions prédictives peuvent être une distribution de probabilités, un réseau neuronal, un arbre de décision, etc. Nous avons décrit, de plus, différentes catégories d'algorithmes d'apprentissage supervisé dans le mémoire. Après avoir construit le modèle prédictif, quand un nouveau cas apparaît, x', qui n'appartient pas au dataset $\mathbb{D}$ mais ayant le même type et décrit avec le même type de caractéristiques que certains cas dans $\mathbb{D}$, alors la tâche est de prédire son label exact en utilisant le modèle préalablement construit $\widetilde{y} = \mathcal{F}(x')$.

Sur les bases de ce court examen, nous admettrons que le choix des techniques d'apprentissage appropriées est un problème réel. En effet, il n'existe pas une étude comparative approfondie de ces techniques qui a été dédié au problématique de prédiction des performances d'algorithmes et en particulier les performances des raisonneurs d'ontologies. Ayant présenté une pléthore riche et importante de réalisations existantes, nous avons l'intention d'examiner empiriquement un certain nombre de ces réalisation afin de pouvoir juger avec certitude de leur adéquation à notre contexte d'étude.

0.3 Résumé du chapitre 2 : Travaux connexes

Le but de ce chapitre est de présenter le contexte général de ce mémoire et de placer notre travail dans l'état de l'art actuel. Ainsi, ce chapitre inclut à la fois une vue générale de la littérature des domaines concernés et une introduction à de nouvelles idées, présentées de façon entrelacée.

Notre travail pose la question très pertinente pour un utilisateur final :

"*Est-il possible de raisonner sur une ontologie écrite avec une logique de description particulière, en utilisant les meilleurs des raisonneurs disponibles ?*"

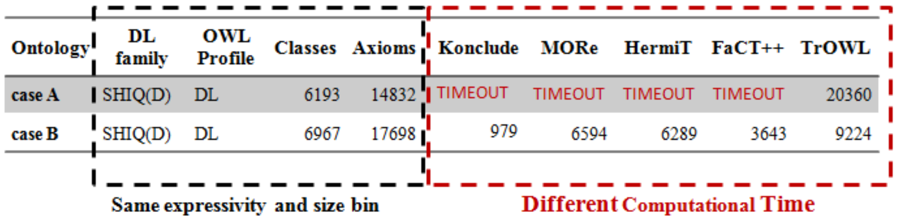
Pour répondre à cette question, nous commencerons par une discussion sur cette faisabilité d'un point de vue théorique en rappelant les classes principales de complexité computationnelle du langage d'ontologie, *i.e.* OWL, et ses DLs sous-jacentes. Puis, nous étudierons la faisabilité d'un point de vue empirique. Par conséquent, nous décrirons la complexité pratique des algorithmes de raisonnement. Nous esquisserons quelques challenges concernant à la fois les concepteurs et les utilisateurs des raisonneurs DL modernes. Plus spécifiquement, nous éclaircirons les phénomènes de variabilité de performance des raisonneurs, puisque la plupart des problèmes concernent la compilation lors de l'utilisation pratique des raisonneurs. Notre but est d'examiner les efforts de recherche qui ont abordé cette question et de discuter de l'utilité de leurs propositions. Aussi, nous mettrons en perspective des méthodes et des outils qui ont essayé de détecter et d'analyser ces phénomènes. Puis, nous aborderons cette question dans une plus large perspective en examinant des solutions proposées dans des champs différents, tels que le machine learning (ML) et l'intelligence artificielle (IA). Plus spécifiquement, nous examinerons les avancées s'adressant aux problèmes liés à la difficulté empirique et à la sélection d'algorithme. En conséquence, les approches dans le méta-apprentissage et la construction de portefeuille d'algorithmes seront passées en revue. Tout ces efforts d'investigation sont mis en avant dans le but de déduire les leçons qui pourraient être tirées de ces domaines et, point essentiel, ceux transférables à notre contexte d'étude, *i.e.* le raisonnement d'ontologie fondé sur les DLs.

Phénomène de variabilité des raisonneurs et solutions existantes

Lors d'une analyse théorique, il n'y a pas besoin d'introduire une procédure efficace qui résolve un problème de déduction. Les classes de complexité sont établies juste en analysant la description des algorithmes et les caractéristiques génériques des cas d'entrée sans exécution réelle. *A contrario*, lors d'une analyse pratique, nous déterminons la complexité temporelle et la complexité spatiale d'un algorithme particulier qui est proposé pour implémenter un problème de déduction à résoudre. En fait, la complexité théorique d'un langage est indépendant de l'algorithme qui le décide et présume que l'algorithme sait quelle branche survivra dans le cas du non déterminisme. Par conséquent, tous les résultats analytiques fournis restent des résultats théoriques et peuvent facilement être réfutés par une analyse expérimentale. Pour cette raison, nous nous sommes tournés vers une analyse empirique plutôt qu'analytique.

Différents travaux [MRW11, Abb12] ont établi qu'en général le raisonnement est faisable en pratique et est beaucoup moins complexe que la complexité théorique prévue. De façon assez intéressante, même avec les fragments assez expressifs d'OWL 2, des performances acceptables de raisonnement pourraient être atteintes. Ceci serait dû aux efforts considérables, en recherche et en ingénierie, qui ont été consacrés à la conception d'algorithmes de raisonnement sophistiqués avec des implémentations hautement optimisées. Dans le Chapitre 1, nous avons effectué un tour d'horizon des raisonnements examinant avec attention ces efforts. Nous avons décrit les raisonneurs les plus connus et comparé leurs attributs fonctionnels. Néanmoins, les auteurs respectifs de [WLL+07, GPS12] ont souligné que, souvent, en pratique, les raisonneurs tendent à montrer des comportements *imprévisibles* sur des ontologies réelles et que leurs performances varient énormément selon les ontologies traitées, même quand la taille ou l'expressivité de ces dernières sont maintenues constantes. Nous avons exposé deux aspects principaux qui devraient décrire le *phénomène de variabilité des performances de raisonneur* : (i) la variabilité du temps d'exécution des raisonneurs en fonction des ontologies ; (ii) la disparité des déductions calculées par différents raisonneurs sur une même ontologie.

Concernant la variabilité d'exécution des raisonneurs, Gonçalves et al. [GPS12] ont exposé trois situations particulières qu'un utilisateur pourrait rencontrer lors d'une tentative de raisonnement sur une ontologie : (i) pour une ontologie de cas de test, changer de raisonneur peut dégrader le temps de raisonnement de quelques secondes à une non terminaison ; (ii) les ontologies ayant la même taille et la même expressivité dépenseront différentes plages de temps de calcul pour le même raisonneur ; (iii) un changement minimal sur une ontologie OWL augmentera ou probablement diminuera le temps de raisonnement sur un raisonneur. Lorsqu'une de ces situations survient, souvent, aucun retour d'information n'est obtenue par l'utilisateur. Habituellement, ce dernier continuera à changer de raisonneurs jusqu'à ce qu'il en trouve un qui lui convienne. D'autres utilisateurs essaieront de



Ontology	DL family	OWL Profile	Classes	Axioms	Konclude	MORe	HermiT	FaCT++	TrOWL
case A	SHIQ(D)	DL	6193	14832	TIMEOUT	TIMEOUT	TIMEOUT	TIMEOUT	20360
case B	SHIQ(D)	DL	6967	17698	979	6594	6289	3643	9224

Same expressivity and size bin

Different Computational Time

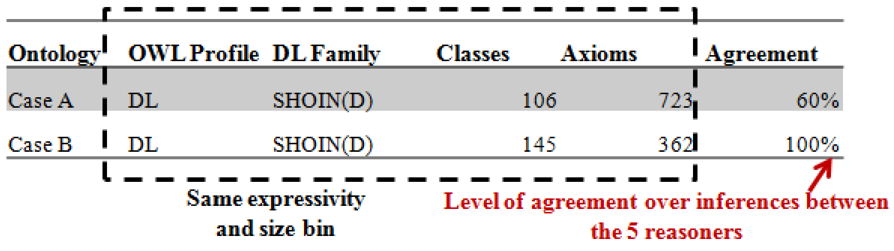
Figure 1: Exemple de la variabilité du temps d'exécution de raisonneurs, extrait des résultats de classification ORE 2014 [BGJ+14]. Le temps est en millisecondes.

modifier leur ontologie en espérant une amélioration, mais courant le risque de rendre la situation pire que précédemment. Globalement, il est accepté qu'il n'y ait pas de système de raisonnement qui puisse surpasser les autres pour toutes les ontologies. Ceci pourrait être observé en examinant les résultats émis lors des campagnes d'évaluation annuelle de raisonneurs, telles que l'"*ORE Evaluation Workshop*" [GBJR+13, BGG+13, BGJ+14]. La Figure 2.2 décrit deux cas de test que nous avons extraits des résultats publics disponibles du challenge de classification DL d'ORE 2014[8].

Dans chaque cas, une ontologie a été classée par différents raisonneurs. Le temps de calcul était enregistré sachant que le temps limité était fixé à 3 minutes. Lors de la consultation de ces résultats, nous avons remarqué que deux ontologies ont le même profil OWL, appartiennent à la même famille DL et leur tailles sont dans la même classe ; cependant, les raisonneurs examinés ont montré des comportements assez distincts sur ces entrées. D'un point de vue théorique, nous nous attendions à ce que le raisonnement sur ces ontologies ne se termine pas dans un temps raisonnable, puisque la complexité de la classe de $\mathcal{SHOIN}$ est N-ExpTime. Cependant, les résultats ont montré le contraire, particulièrement dans le second cas. Néanmoins, le temps d'exécution de ces raisonneurs est assez variable pour la même ontologie, *e.g.* dans le second cas, une différence de temps d'exécution jusqu'à un ordre de grandeur de 10 entre Konclude et TrOWL ; et, à l'analyse des deux cas, *e.g.* TrOWL a été le seul raisonneur à terminer la classification dans le premier cas mais a été le plus lent dans le second cas, sachant que les deux ontologies sont similaires en regard de leur taille et de leur degré d'expressivité.

Concernant la dispersion des résultats des raisonneurs, beaucoup moins de travaux ont abordé ce phénomène puisque les bases formelles des procédures au coeur du raisonneur sont bien comprises et que les concepteurs fournissent souvent une preuve théorique de la correction et de la complétude de leurs algorithmes de raisonnement. Cependant, Gardiner et al. [GTH06] et, plus récemment, Lee et al. [LMPS15] ont identifié que les raisonneurs pouvaient délivrer des déductions ou des réponses à des requêtes assez distinctes à partir d'une ontologie. Ils ont mis en évidence que ce problème pourrait

[8]résultats ORE 2014 disponibles à `https://zenodo.org/record/11142/`



Ontology	OWL Profile	DL Family	Classes	Axioms	Agreement
Case A	DL	SHOIN(D)	106	723	60%
Case B	DL	SHOIN(D)	145	362	100%

Same expressivity and size bin
Level of agreement over inferences between the 5 reasoners

Figure 2: Exemple sur la diversité de déductions calculées de raisonneur, fondé sur le challenge de classification d'ontologie ORE 2014 [BGJ+14].

entraver le développement et menacer sérieusement l'interopérabilité du Web sémantique. De plus, les raisonneurs ayant calculé des déductions pendant les évaluations ORE sont comparés aux autres, dans le but d'identifier les cas de divergences et afin de décider quels résultats de raisonneur sont probablement les plus corrects. La Figure 2.3 décrit un autre extrait des résultats de classification d'ORE 2014 DL. Ici, nous avons vérifié le degré d'accord entre les cinq raisonneurs que nous avions précédemment listés dans la Figure 2.2. Nous aimerions faire remarquer que le degré d'accord était différent sur deux ontologies appartenant à la même taille et au même niveau d'expressivité. En effet, dans le premier cas, seulement 3 des 5 raisonneurs ont maintenu les mêmes axiomes ; cependant, dans le second cas, tous ont obtenu les mêmes résultats.

Gardiner et al. ont expliqué que la divergence peut être due au mécanisme d'approximation que certains concepteurs de raisonneur utiliseraient pour manipuler des expressivités particulières et qui sont difficiles à manipuler par des composants DL. Cependant, les concepteurs de raisonneur souvent ne communiquent pas beaucoup de détails sur ces composants et leurs techniques d'approximation. Lee et al. ont aussi souligné ce point. Ils ont mis en évidence que peu de détails d'optimisation de raisonnement et d'astuces d'ingénierie sont fournis ; et, habituellement, ces astuces sont seulement contrôlées par les développeurs. Ayant acté cela, nous aimerions admettre que la fiabilité des implémentations de raisonneurs est encore en cours d'investigation.

Selon cette étude, nous pouvons affirmer que, bien que le raisonnement en pratique est bien moins complexe que la complexité computationnelle théorique établie, les comportements empiriques des raisonneurs sont encore difficilement prévisibles *a priori* dus à une variabilité inattendue sur les ontologies. Par conséquent, de nouveaux challenges émergent, tels que *comment est-il possible d'anticiper cette variabilité et comment en expliquer les causes.* Globalement, nous pouvons noter que les ontologies appartenant à la même classe de complexité théorique ne sont pas toutes équitablement difficiles. Manifestement, il existe certaines caractéristiques d'ontologies autres que la taille ou l'expressivité qui vont probablement avoir un impact sur les performances des raisonneurs. Par conséquent, une implémentation de raisonneur qui exploite certaines caractéristiques particulières du langage OWL peut s'exécuter sur la classe correspondante d'ontologies mieux que le scénario du pire des cas. En effet, plusieurs auteurs [GTH06, GPS12, LMPS15] ont expliqué la variabilité des performances de raisonneur par le fait que les systèmes de raisonnement modernes ont une architecture de conception complexe, et tendent à implémenter un large spectre de techniques d'optimisation qui peuvent interagir et s'affecter les unes les autres lors du traitement d'ontologies particulières. Par conséquent, un raisonneur pourrait être optimisé pour certaines ontologies mais non pour toutes. Sur ce constat, nous admettrons qu'une meilleure compréhension des facteurs de complexité des ontologies générant des difficultés pour les raisonneurs est d'un besoin impérieux. Selon nous, se focaliser sur *ce qui rend un raisonnement difficile en pratique* peut guider le processus de modélisation des ontologies pour éviter les facteurs de dégradation de performance du raisonnement. En outre, les ontologies existantes peuvent être révisées vers un raisonnement efficace en détectant, et même en réparant, leurs composants critiques. D'autre part, quand une telle connaissance est rendue disponible, la sélection automatique d'un raisonneur adéquat pour une ontologie donnée en entrée pourrait être fait plus facilement et sans besoin d'expérimentations et d'essais excessifs. Dans la suite de ce chapitre, nous étudierons et discuterons des solutions existantes qui concernent la complexité pratique des raisonneurs DL et fournissent une solution pour la gérer.

Sélection automatique des algorithmes

La Table 2.2 synthétise les attributs fonctionnels de base et l'utilisation pratique des approches décrites. Selon cette étude, il semble clair qu'en employant des méthodes statistiques et des méthodes de ML supervisé plus spécifiques, nous pourrions tirer avantage d'une compréhension plus nuancée des comportements empiriques d'algorithme plutôt que des analyses analytiques dans le pire des cas. Grâce à cette approche, il a été montré que les performances des algorithmes provenant de champs différents (ceux du ML ou ceux de l'AI) pourraient être décrites avec un haut degré de fiabilité fondée sur la facilité de traitements des caractéristiques des cas de problème. Aussi, divers problèmes pratiques algorithmiques tels que la sélection automatique d'algorithme, la recommandation ou la paramétrisation optimale ont été résolus efficacement. Ceci rend l'approche basée ML pour une analyse et une sélection d'algorithme importante même pour les informaticiens qui sont théoriquement enclin à préférer des preuves à des découvertes expérimentales.

Toutes ces études renforcent notre conviction que l'utilisation des approches basées ML pour la prédiction des performances et de sélection d'algorithme devraient apporter une réponse à nos questions de recherche. Quelques leçons pourraient être tirées, de cet état de l'art, selon nous, qui sont d'une importance majeure pour arriver à une prédiction et une sélection de raisonneurs réalistes.

- un des challenges pratiques les plus importants dans n'importe quelles expérimentations al-

Propos	**Critères**	**Techniques ML**	**Usage practique**
Méta-apprentissage	- Exécution - Qualité de prédiction - Critères combinés	- Classification - Régression - Apprentissage de préférence	- Recommandation d'Algorithme - Paramètre de réglage d'algorithme. - Apprentissage "Ensemble". - Sélection dynamique de biais.
Algorithme portfolio	- Exécution	- Classification - Régression - Apprentissage de préférence	- Construction algorithme portfolio. - Paramètre de réglage d'algorithme. - Génération des bancs de test des cas difficiles. - Identifier les caractéristiques clé de la difficulté des cas de problème.

Table 3: Résumé des attributs des approches d'analyse et de sélections d'algorithme.

gorithmiques est de rassembler les cas de test exigés. Les meilleurs cas de test sont probablement ceux du monde réel qui sont extraits des applications réelles. C'est aussi le cas si l'on veut tester les performances des raisonneurs Une population représentative des ontologies devrait être sélectionnée avec attention pour éviter qu'un raisonneur, sachant traiter seulement certaines ontologies et donc optimisé pour elles, est de bons résultats ;

- la sélection des caractéristiques d'ontologie est liée fortement à la connaissance du domaine. Notre tour d'horizon des travaux existants a montré qu'il n'y a pas d'accord sur les origines de la difficulté des ontologies par rapport aux tâches de raisonnement et qu'il n'y a aucune définition de la notion de caractéristique d'une "*bonne*" ontologie. Globalement, une caractéristique devrait être facile à traiter (habituellement c'est négligeable comparée au temps de raisonnement) et plus pertinente à la performance d'un raisonneur particulier ou une classe d'ontologies particulière ;
- le temps d'exécution et la qualité de la solution sont les deux mesures les plus connues de performance d'algorithme. Dans notre cas, nous essaierons de prédire le temps d'exécution du raisonnement et la justesse des résultats engendrés. Cependant, des contraintes expérimentales devraient être fixées pour assurer la reproductibilité des résultats. De plus, une tâche de raisonnement particulier, *e.g.* la classification d'ontologies ou le contrôle de cohérence, devrait être fixée comme le sujet principal de la prédiction de la performance ;
- un autre problème concernant la fiabilité des données empiriques pour la prédiction est l'équilibre entre le temps et la qualité de la solution. Jusqu'ici, la façon la plus efficace pour comparer

différents algorithmes heuristiques ou de décision est de permettre à tous les algorithmes de consommer la même quantité de ressources et de traiter le même ensemble de cas de problèmes ;

- il n'y a pas de *meilleur* algorithme de ML supervisé pour la prédiction. Aussi, différents algorithmes d'apprentissage devraient être explorés pour identifier celui qui serait capable de faire l'entraînement des modèles avec une haute précision engendrant des données analysables sur les performances des raisonneurs ;
- en général, tout raisonneur peut être utilisé comme un candidat pour une tâche de sélection ou de recommandation. Mais, en pratique, nous pouvons utiliser la connaissance du domaine pour éliminer les raisonneurs qui sont évidemment inférieurs aux autres ;
- le choix de la technique de sélection de raisonneurs est fortement dépendante du but final et du contexte général de l'application. Certaines techniques sont sensibles au temps, principalement pour l'algorithme de construction de portfolio. D'autres sont beaucoup plus informatifs et devraient être souples quant aux conseils à donner aux utilisateurs finaux sur la disponibilité des raisonneurs. Par conséquent, ce point devrait être examiné lors de la conception de notre solution.

0.4 Résumé du chapitre 3 : Profilage des ontologies

Le but principal de cette thèse est de faire face à la variabilité des performances empiriques des raisonneurs d'ontologies par une application rigoureuse d'une approche d'apprentissage supervisé. Cette dernière nécessite la mise en correspondance entre un ensemble de caractéristiques décrivant l'entrée du raisonneur, *i.e.* l'ontologie, et la variable de sortie représentant la performance du raisonneur à prédire. De toute évidence, choisir de bonnes caractéristiques est crucial pour la construction de bons modèles prédictifs. Dans le chapitre précédent, nous avons souligné le manque de méthodologie pour automatiser la construction de bons ensembles de caractéristiques pour décrire la complexité computationnelle des ontologies OWL. En effet, les chercheurs doivent utiliser diverses connaissances du domaine pour identifier les propriétés susceptibles de fournir des informations utiles quant à la complexité du modèle conceptuel de l'ontologie à l'égard des traitements du raisonnement.

Compte-tenu des constatations faites dans ce chapitre, nous essaierons de déterminer les principaux descripteurs de la complexité empirique d'une ontologie OWL. Pour ce faire, nous avons conduit une large étude sur les travaux existants proposant des métriques pour évaluer la qualité de conception d'une ontologies [GCCL06, TAM$^+$05, LNJ$^+$10, ZLT10] ou en abordant les origines de la complexité du raisonnement en logiques de description [Don03, THPS07, BCM$^+$10]. Notre étude s'est portée sur la richesse de l'expressivité de l'ontologie du point de vue de la représentation des connaissances.

Nos investigations nous ont amené à proposer une nouvelle collection comprenant 116 descripteurs d'ontologie. Ces derniers sont, dans leur forme de base, les statistiques qualitatives et quantitatives décrivant un large éventail d'attributs structurels et syntaxiques d'une ontologie modélisée avec le langage OWL.

Dans la suite de ce chapitre, nous introduisons notre suite de descripteurs d'ontologie. Puis, nous décrivons le corpus d'ontologie choisi pour les différentes expérimentations planifiées dans le cadre de cette thèse. Enfin, nous soumettons à la discussion du coût de calcul des principaux sous ensembles de descripteurs, afin de mettre le point sur ceux qui sont le plus efficacement calculables.

Caractéristiques des ontologies.

Comme énoncé précédemment, il n'existe pas de méthodologie pour construire de "bons" ensembles de caractéristiques d'ontologie. Pour répondre à nos problématiques, nous avons réutilisé quelques caractéristiques d'ontologie proposées précédemment et en avons définies de nouvelles. Principalement, nous avons renoncé à celles qui sont calculées sur des transpositions spécifiques de graphe d'ontologie OWL, car il n'y a pas d'accord sur la façon dont ces ontologies devraient être traduites en graphe [KKL15]. Nous avons scindé les caractéristiques d'ontologie en quatre catégories à savoir : (i) description de la taille, (ii) description de l'expressivité, (iii) caractéristiques structurelles et (iv) caractéristiques syntaxiques. La Figure 3 liste les principales[9].

Description de la taille de l'ontologie

Le propos de cette catégorie est de spécifier la taille de l'ontologie en considérant à la fois le nombre de ses termes et de ses axiomes. Donc, nous avons conçu cinq caractéristiques, chacune enregistrant les noms d'une entité particulière OWL. De plus, nous avons calculé le nombre d'axiomes (**SA**) et plus particulièrement les logiques (**SLA**).

Description de l'expressivité de l'ontologie

Nous avons retenu deux caractéristiques principales pour identifier l'expressivité du langage de l'ontologie , nommément le profil OWL (**OPR**) et le nom de la famille DL (**DFN**).

[9]pour plus de détails, voir le mémoire

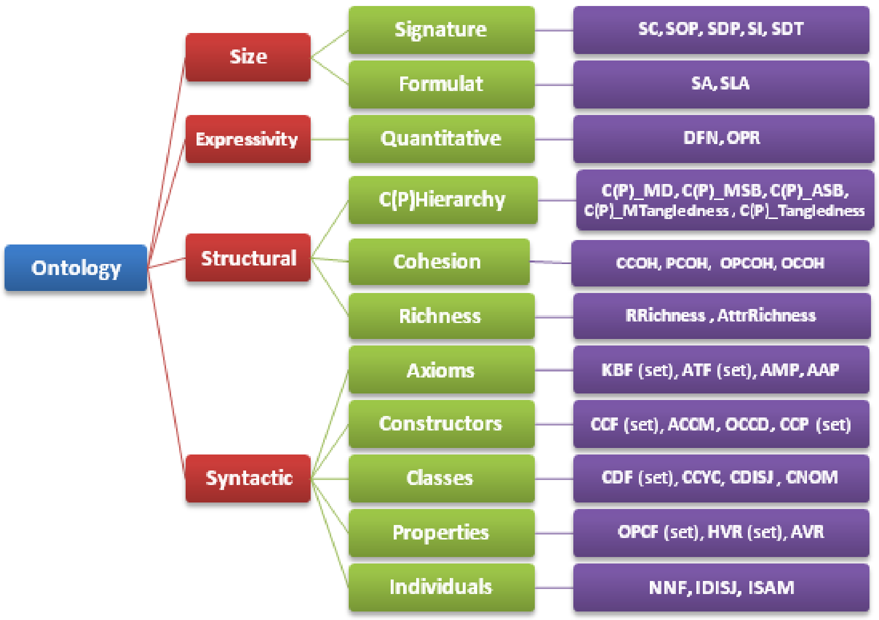


Figure 3: Catalogue des caractéristiques d'ontologie.

Description structurelle de l'ontologie

Nous avons porté une attention spéciale à caractériser la structure taxonomique d'une ontologie, *i.e.* sa hiérarchie d'héritage. Cette dernière schématise la structure arborescente des relations de subsumption entre les classes $A \sqsubseteq B$ et entre les relations $R \sqsubseteq S$. Dans cette catégorie, nous avons rassemblé différentes caractéristiques qui ont été définies dans la littérature pour décrire les hiérarchies d'ontologie. Ce sont, tout simplement, des métriques largement utilisées par la communauté d'évaluation de la qualité d'ontologie [GCCL06, TAM$^+$05, LNJ$^+$10]. Les sous-catégories suivantes décrivent l'essence des caractéristiques retenues.

Caractéristiques hiérarchiques des classes et des propriétés Cinq caractéristiques ont été spécifiées pour souligner la conception de la hiérarchie de classes (**CHierarchy**). Elles considèrent la profondeur de l'arbre et la distribution des sous-classes aussi bien que des super-classes. Ces caractéristiques ont été aussi utilisées pour spécifier la conception de la hiérarchie des propriétés (**PHierarchy**).

Caractéristiques de cohésion La littérature fournit pléthore de métriques pour concevoir la *cohésion* de l'ontologie et, d'autre part, le degré de parenté de ses entités. Nous avons retenu celles introduites par [FW10]. Grossièrement, la cohésion de l'ontologie (**OCOH**) est une agrégation pondérée de la cohésion de classes (**CCOH**), la cohésion de propriétés (**PCOH**) et la cohésion de propriétés d'objet (**OPCOH**).

Caractéristiques de richesse de schéma Nous avons enrichi la catégorie structurelle de l'ontologie par deux propriétés supplémentaires proposées par [TAM+05] : la richesse de relations de schéma (**RRichness**) et la richesse d'attributs de schéma (**AttrRichness**). Ces caractéristiques sont bien connues par la communauté d'évaluation d'ontologies comme une partie de l'outil *ontoQA*.

Caractéristiques syntaxiques d'ontologie

Lors de la collecte des caractéristiques de cette catégorie, nous avons conduit une étude sur les algorithmes de raisonnement [BS01, MSH09]. Notre propos était de quantifier certaines des connaissances générales théoriques sur les sources de complexité DL. Ainsi, nous avons rassemblé les caractéristiques pertinentes qui ont inspirées l'implémentation des techniques bien connues d'optimisation de raisonnement [HPSV03, TH06]. Les caractéristiques de ce groupe sont divisés en six sous-catégories, chacune spécifique à un composant syntaxique particulier d'une ontologie. Cette organisation a été inspirée de celle introduite par [KLK12].

Niveau d'axiomes Les raisonneurs traitent différemment chaque type d'axiome avec différents coûts computationnels [BCM+03]. Nous avons introduit deux ensembles de caractéristiques, (**KBF**) et (**ATF**), dans le but de spécifier les différents types d'axiomes OWL aussi bien que leurs pertinences respectives dans l'ontologie. De plus, nous avons calculé la profondeur maximale et moyennée d'analyse syntaxique des axiomes (**AMP**, **AAP**).

Niveau de constructeurs Ceci concerne particulièrement les constructeurs de classes DL[10] [BCM+03]. Dans les précédents travaux sur la prédiction des raisonneurs [KLK12, KLK14], les auteurs ont simplement compté les axiomes qui impliquent potentiellement les constructeurs difficiles. Cependant, ils ont oublié qu'un constructeur pouvait être invoqué plus d'une fois dans le même axiome. Considérant cela, nous avons proposé une métrique pour calculer la fréquence d'un constructeur de classe dans une ontologie (**CCF**). De plus, nous avons défini la *densité* des constructeurs globaux (**OCCD**). Nous avons aussi introduit trois patrons de modélisation de combinaisons particulières de constructeurs. Nous croyons que ces combinaisons, chaque fois qu'elles sont utilisées dans un axiome, devraient augmenter le coût de déduction computationnelle.

[10]Par constructeur de classe DL, nous faisons référence à la conjonction ($\sqcup$), la disjonction ($\sqcap$), la négation ($\neg$), la quantification ($\exists$, $\forall$), etc.

Niveau de classes Les classes dans une ontologie pourraient être nommées, des expressions simples ou complexes. Elles pourraient être cycliques ou disjointes. Dans cette sous-catégorie, nous avons introduit des métriques qui caractérisent l'utilisation de chacune de ces méthodes dans la partie *TBox* de l'ontologie.

Niveau de propriétés Nous nous sommes intéressés à la capture de deux aspects de description syntaxique de propriétés d'ontologie. Tout d'abord, nous avons essayé de souligner la pertinence des caractéristiques spécifiques de propriété d'objet, telles que la transitivité, la symétrie, la réflexivité, etc. Puis, nous avons introduit une métrique, (**OPCF**), qui mesure leurs fréquences respectives dans l'ontologie. Puis, nous avons proposé deux métriques supplémentaires (**HCV**, **AVC**) visant à examiner l'impact de l'utilisation de hautes valeurs dans les restrictions de cardinalités.

Niveau d'individus Nous spécifions certaines des caractéristiques intéressantes d'individus nommés qui devraient être déclarés dans l'ontologie. Nous avons examiné le ratio des nominaux dans la *TBox* (**NNF**) et calculé le nombre d'individus déclarés comme équivalents (**ISAM**) ou disjoints (**IDISJ**).

Au cour de la modélisation de ces descripteurs, nous avons parti du principe que, en utilisant les caractéristiques décrites ci-dessous, les ingénieurs d'ontologie pourraient obtenir une meilleure compréhension de la complexité globale de leurs ontologies. Par exemple, ils pourraient caractériser sa taille, son degré d'expressivité et quel degré de régularité ou d'irrégularité potentielle a son arbre d'héritage de classes et/ou de propriétés. De plus, ils pourraient examiner la richesse de l'ontologie en terme de types d'axiome ou bien le degré d'utilisation des constructeurs pour exprimer des classes ou les propriétés complexes. Il serait aussi possible d'informer l'ingénieur d'ontologie sur la présence de certaines sources de complexité bien connues en logique de description dans le modèle conceptuel de son ontologie.

Corpus local d'ontologie

Nous avons choisi 2500 ontologies provenant du corpus de la campagne d'évaluation des raisonneurs d'ontologies, ORE [MBP13]. Cette sélection représente notre échantillon d'étude d'ontologies OWL, notée par $\mathcal{C}(O) = \{O_1, \ldots, O_n\}$. Cet ensemble sera utilisé dans les diverses expérimentations planifiées tout au long de cette thèse. Notre échantillon couvre 167 différentes familles de logique de description et les diverses profils d'OWL 2, dont OWL DL, EL, RL et QL. Les ontologies sont organisées en classes selon leur taille[11]. Nous pouvons distinguer entre les petites $(0, 100]$, moyennes $(1000, 10000]$,

[11]La taille est calculée en terme de nombre d'axiomes logiques

grandes $(10000, 100000]$ et très grandes > 100000 ontologies. Il est à noter que la plus grande ontologie comporte 3 182 261 axiomes logiques.

Coût de calcul des descripteurs

Afin d'évaluer leur coût, nous avons mémorisé le temps de calcul des valeurs des différents descripteurs pour chacune de nos ontologies. Ensuite, le temps accumulé de calcul de certain ensemble de descripteurs a été comparé au temps de chargement des ontologies. Comme résultat, nous avons identifié qu'au moyenne 8 parmi nos 116 descripteurs sont le plus exigeants en temps de calcul. Plus précisément, il s'agit des descripteurs en relation avec le calcul du nombre des classes cycliques, l'identification des patrons de couplage des constructeurs et le calcul du profondeur de hiérarchie de classe. De manière général, nous pouvons considéré que notre bilan est positif. La majorité de nos descripteurs peuvent être mesurés au bout d'un temps qui ne dépasse par celui nécessaire au chargement de l'ontologie. Ceci nous indique que nous pouvons les déployer dans un système de prédiction autour des raisonneurs sans craindre une charge supplémentaire qui risque de ralentir la réponse du système.

0.5 Résumé du chapitre 4 : Modélisation prédictive de la robustesse des raisonneurs

Le but principal de cette thèse est de fournir une assistance pour les utilisateurs afin de faire face aux phénomènes de variabilité de performances des raisonneurs d'ontologie. Les comportements empiriques de ces systèmes sont étudiés et analysés par recourt à des techniques d'exploration intelligente des données décrivant l'historique des performances des raisonneurs. En particulier, nous avons opté pour une approche fondée sur l'apprentissage supervisé (SML) afin de pouvoir capter les relations de corrélation entre la complexité des modèles conceptuels des ontologies et les performances des raisonneurs. La finalité de ce travail est d'apprendre des modèles génériques de prédiction capable d'anticiper les comportements futurs des raisonneurs sur n'importe quelle ontologie. Plus spécifiquement, nous avons mis en place une méthodologie générique d'apprentissage capable de traiter différents problèmes de prédiction autour des raisonneur. Grâce à notre solution, nous avons pu construire un modèle de prédiction de la *difficulté empirique* d'une ontologie à l'égard d'un groupe homogène de raisonneurs. En outre, nous avons conduit une analyse fine des raisonneurs, pris individuellement, dans le but de créer, pour chacun d'eux, un *profil prédictif de robustesse* (*PRP*). Cette méthodologie a été conçu toute en respectant un ensemble de contraintes fonctionnelles et non fonctionnelles relatives à cadre spécifique d'utilisation des raisonneur. Dans cette thèse, nous avons choisi la classification e*n ligne* des ontologies OWL comme le scénario principal pour l'évaluation et l'étude de performances des raisonneurs. Un

Framework générique d'apprentissage employant un riche ensemble de techniques SML a été mis en place. Les détails de description de ce dernier aussi bien que son évaluation empirique sera présentée dans ce chapitre. Suite

Nous avons organisé ce chapitre comme suit. Nous avons commencé par spécifier les problèmes prédictifs examinés dans cette thèse. Ensuite, nous avons décrit la conception de notre méthodologie générique d'apprentissage. Pour cela, nous avons discuté les diverses techniques d'apprentissage supervisé que nous avons employées. Ensuite, nous avons illustrés les résultats d'évaluation empirique des performances des raisonneurs dans le cadre de classification d'ontologie OWL. A l'issue de ces évaluations, des bases de données d'apprentissage ont été produites, ensuite fournies à notre Framework dans le but de construire les modèles prédictifs sur la "difficulté" des ontologies et sir la robustesse des raisonneurs. Les détails d'évaluation de ce cas d'étude sont fournis dans le mémoire. Enfin, les divers modèles que nous avons produits ont été examinés dans le but de découvrir les descripteurs clés des ontologies, ceux qui ont été identifié comme les plus corrélés aux comportements des raisonneurs.

Définition de la robustesse

V. Mikolàšek [Mik09] ont défini la robustesse dans le champ logiciel comme *la capacité d'un système à délivrer ses fonctions et à éviter les échecs de service lors de certaines conditions inattendues*. Gonçalves et al. [GMPS13] se sont fiés à cette définition dans le but de conduire une étude empirique sur la robustesse des raisonneurs DL. Les auteurs ont souligné la nécessité de spécifier des contraintes particulières computationnelles avant d'évaluer les raisonneurs. Ces contraintes devraient décrire un scénario d'usage spécifique de raisonneur, sous lequel un raisonneur peut être considéré comme robuste ou non. Nous avons étendu les recommandations de Gonçalves et al. en concevant l'ensemble **RJC** comme suit :

Ensemble RJC RJC repose sur les contraintes de jugement de la robustesse d'un raisonneur. C'est un ensemble fini de variables qui décrivent des exigences fonctionnelles (FC) et non fonctionnelles (NFC) relatives à un scénario particulier de raisonnement. RJC inclut à la fois FC=$\{$*limite-temps*, *limite-mémoire*, *variété-ontologie*$\}$ et NFC=$\{$*tâche-raisonnement*, *état-succès*, *état-échec*$\}$, *i.e.* $RJC = FC \cup NFC$.

Dans notre étude, nous avons choisi d'examiner un scénario de classification en ligne des ontologies OWL. Dans ce contexte, la durée du traitement est susceptible d'être très réduite, généralement limitée à quelques minutes alors que l'utilisation de la mémoire pourrait être moins contrainte puisque les raisonneurs devraient être exécutés sur des serveurs puissants. N'importe quel type d'ontologie devrait être supporté par les raisonneurs. D'autre part, nous avons décidé de vérifier la tâche de classification

DL dont nous avons montré l'importance précédemment. Ceci conduit aux définitions suivantes des états de succès et d'échec :

- le *succès* est reporté quand le raisonneur termine la tâche de classification d'ontologies dans le temps-limite imposé et délivre des résultats corrects ;
- l'*échec* signifie soit un raisonneur qui s'arrête lors du traitement de l'ontologie, soit qui dépasse le temps-limite, soit qui, éventuellement, délivre des résultats inattendus.

Compte-tenu de cela, nous définissons la robustesse d'un raisonneur dans le respect d'un ensemble RJC particulier comme suit :

Robustesse de Raisonneur La robustesse d'un raisonneur montre sa capacité à terminer une tâche de raisonnement avec succès sous des contraintes spécifiques de ressources computationnelles pour toute ontologie donnée.

Nous croyons que notre définition de robustesse devrait couvrir les qualités désirées à la fois d'efficacité et de justesse. Ainsi, en retrouvant cette propriété, nous devrions fournir des moyens de comprendre et de diminuer les phénomènes de variabilité des performances de raisonneur précédemment explicités. La définition ci-dessus peut être, de plus, étendue pour spécifier le *raisonneur le plus robuste* selon un corpus d'ontologie et un groupe particulier de raisonneurs candidats.

Raisonneur le plus robuste Soit un ensemble fixé de RJC, le raisonneur le plus robuste pour un corpus d'ontologies est celui satisfaisant l'exigence de succès pour le plus grand nombre d'ontologies tout en maintenant le temps de traitement le plus court.

Considérant les définitions précédentes, nous pouvons maintenant établir le concept de difficulté globale empirique d'ontologie (OOH)[12] comme suit :

Difficulté globale d'ontologie Soit un ensemble fixé de RJC, une ontologie est dite empiriquement difficile à l'égard d'un groupe homogène de raisonneurs lorsqu'au moins un de ces raisonneurs ne réussit pas à la traiter avec succès.

Un groupe de raisonneurs est dit homogène lorsqu'il partage un ou plusieurs attributs fonctionnels. Dans cet thèse, nous avons choisi de catégoriser les raisonneurs selon leur méthodologie de raisonnement. Par example, nous avons distingué entre les raisonneurs qui implémentent une approche basée sur l'algorithme du Tableau [BS01] comme Pellet [SPG$^+$07] ou bien Hermit [?], de ceux qui adoptent une approche de raisonnement par conséquence [SKH11], tel que Konclude [LSN10].

[12] *i.e.* Ontology Overall Hardness

Définition des problèmes d'apprentissage

Partant des définitions ci-dessous, notre objectif est de construire des modèles de prédictions du niveau de difficulté empirique des ontologie et de niveau de robustesse empirique des raisonneurs.

Prédiction de la difficulté des ontologies Il s'agit de construire un modèle de classification bi-classe qui permet d'attribuer à une ontologie décrite par un ensemble de descripteurs, une de ceux deux labels : "Safe" ou bien "Hard". Ce modèle est appelé M_{OOH}. Lorsqu'une ontologie est étiquetée par "Safe", cela signifie que tous les raisonneurs du groupe en cours d'étude, peuvent l'inférer au bout d'un délai fixé et que les résultats issus de ce traitement sont corrects. Dans le cas d'une classification en tant que "Hard", le modèle anticipe qu'au moins un des raisonneurs ne réussirait pas son traitement. Le vecteur qui contient les valeurs calculées des descripteurs de l'ontologie est noté $X(O)$. Ce vecteur appartient au dimension de tous les descripteurs noté par $\mathcal{F}^s$.

$$M_{OOH} : X(O) \in \mathcal{F}^s \mapsto \hat{l} = "Safe", "Hard" \tag{1}$$

Prédiction de la robustesse des ontologies Dans cette thèse, nous nous sommes focalisé sur deux aspects qui caractérisent la robustesse empirique d'un raisonneur. Il s'agit de son efficacité et de l'exactitude de ses résultats. Partant des définitions ci-dessous et en particulier la spécification des cas de "succès" et d'"échec", nous avons proposé de construire deux modèles de prédiction, afin d'anticiper les qualités d'un raisonneur que nous avons précisé. Le premier, noté par $M_{S|R}$, est un modèle de classification multi-classes. Il permet de prédire l'état de terminaison de la tâche du raisonnement réalisé par le raisonneur R sur une ontologie O donnée en entré. Plus précisément, il s'agit de discriminer entre 4 états : *i)* "succès", qui indique une terminaison au bout d'un délai fixé à l'avance et que des résultats sont correctes; *ii)* "inattendu", qui indique aussi une terminaison à temps mais avec des résultats inattendus; *iii)* "timeout", lorsque le délai est dépassé; et finalement *iv)* "arrêt", lorsque les traitements sont interrompus suite à une erreur d'exécution. Ces états représentent les labels possibles que le modèle de prédiction $M_{S|R}$ peut attribuer à une ontologie décrite par un vecteur de descripteurs $X(O)$. Le deuxième modèle que nous avons proposé de développer, est noté par $M_{T|R}$ est un modèle d'estimation du temps d'exécution d'un raisonneur lorsque le succès de sa tâche de raisonnement est déjà prédite. Il s'agit de construire un modèle de régression est associe au vecteur des descripteurs de l'ontologie, une valeur continue qui représente le temps estimé pour le calcul. L'association entre ces deux modèles, i.e $M_{S|R}$ et $M_{T|R}$, destinés à prédire l'exactitude et l'efficacité d'un raisonneur, correspond à notre concept de profil prédictif de robustesse d'un raisonneur. Ce profil est noté par PRP_R. Dans ce mémoire, nous avons décrit ce concept avec plus de précision.

Méthodologie d'apprentissage

Rappelons que les performances des algorithmes d'apprentissage supervisé sont très sensibles à la qualité des variables qui décrivent les instances d'une base d'apprentissage. Dans notre cas, il s'agit des descripteurs d'ontologies. Ces descripteurs peuvent être inter-corrélés, redondants ou encore non pertinents. Autrement dit, ils ne contribuent pas à l'anticipation des futures comportements des raisonneurs et risquent même de détériorer détériorer la précision des modèles prédictifs. Une approche simple pour filtrer ces descripteurs est d'opter pour des méthodes de sélection de variable [CS14]. Ces méthodes sont utilisées dans le but d'ôter les descripteurs redondantes et peu corrélés aux performances observées des raisonneurs. Un nombre considérable de ce type de méthodes est disponible dans la littérature. Nous avons choisi les plus connues et nous les avons regroupé dans un ensemble spécifique que nous avons noté $\mathcal{FS}$. L'application de ces méthodes impliquera la création de nouvelles variantes de base de données d'apprentissage, appelées les bases *pré-traitées*. Ces derniers sont caractérisés par un éventuel sous-ensemble réduit de descripteurs d'ontologie. De la même façon, nous avons étudié plusieurs types d'algorithmes d'apprentissage supervisé, que nous avons regroupé dans l'ensemble fini noté par $\mathcal{SA}$. Il est intéressant de souligner que l'ensemble $\mathcal{FS}$ contient l'élément $\varnothing$, qui signifie qu'aucun élagage des descripteurs n'est appliqué, autrement dit la base est maintenue dans son état brute. Cette dernière est noté par $RAWD$. Le choix de cette configuration nous a permis de comparer entre la qualité des modèles prédictifs issues de $RAWD$ et ceux construits à partir des bases pré-traitées. Étant donné une problème P qui nécessite la construction d'une modèle prédictif, les étapes d'apprentissage seront répétées pour chaque variante de base d'apprentissage de ce problème, en utilisant à chaque fois un algorithme d'apprentissage différent. A l'issue de cette phase un ensemble de modèles candidats sont appris. Ainsi, la tâche suivante consiste à déterminer le *"meilleur"* modèle parmi ces candidats. Il s'agit de choisir le modèle ayant la meilleure qualité de prévisions et le plus faible nombre de descripteurs d'ontologies. Afin de réaliser cette sélection, les modèles établis sont évalués par rapport à un groupe de mesures de qualité de prévision, noté $\mathcal{AM}$. La validation croisée stratifié est la technique principale que nous avons appliqué lors des évaluations. Nous avons proposé un ensemble de scores qui synthétisent la qualité d'un modèle de prédiction. Ces scores varient selon la nature du problème d'apprentissage. En se référant aux valeurs de ces scores, les modèles candidats sont classés et le *"meilleur"* est retenu comme modèle final à sauvegarder pour un futur usage.

Les différents étapes qui constituent notre méthodologie d'apprentissage sont mis en place dans le but d'identifier la combinaison la plus efficace (algorithme d'apprentissage, méthode de sélection de variable), capable de nous fournir les meilleurs prévisions possibles, étant donnée, les données mises à notre disposition. Dans notre cadre d'apprentissage, toutes les techniques SML déployées sont rassemblées dans une base de techniques SML, notée $\mathcal{TB}_{sml}$. *Grosso modo*, $\mathcal{TB}_{sml}$ correspond au triplet $< \mathcal{FS}, \mathcal{SA}, \mathcal{AM} >$. Les éléments de $\mathcal{TB}_{sml}$ sont organisés de façon que cela indique s'ils devraient être appliqués à la classification ML ou aux problèmes de régression.

```
Algorithm 1: Cadre d'apprentissage supervisé
  Entrée: $\mathcal{C}(O) = \{O_1, \dots, O_n\}$ l'ensemble des ontologies d'entraînement, $\mathcal{L}_P = [l(O_1), \dots, l(O_n)]$
          le vecteur de labels relatif au problème $P$.
  Sortie : $M_P^{Best}$ meilleur modèle appris selon le problème $P$.
1  $[X^{(O_1)}, \dots, X^{(O_N)}] \leftarrow computeTheFeatureVectors(\mathcal{S}(O))$;
2  $\mathcal{D}_P \leftarrow buildTheRawDataSet([X^{(O_1)}, \dots, X^{(O_N)}], \mathcal{L}_P)$ ;
3  $\mathcal{TB}_{sml} \leftarrow getSMLTechniquesBase(\mathcal{L}_P)$ ;
4  foreach $F \in \mathcal{FS}$ do
5      $\mathcal{D}_P^F \leftarrow buildTheFeaturedDataSet(F, \mathcal{D}_P)$;
6      foreach $A \in \mathcal{SA}$ do
7          $< M^{(F,A)}, \pi^{(F,A)} > \leftarrow trainAndAssess(A, \mathcal{D}_P^F, \mathcal{AM})$ ;
8          $\mathcal{M}^F \leftarrow \mathcal{M}^F \cup \{< M^{(F,A)}, \pi^{(F,A)} >\}$ ;
9      end
10     $< M^{(F,best)}, \pi^{(F,best)} > \leftarrow bestModelFromADataSet(\mathcal{M}^F)$ ;
11     $\mathcal{M}^{Best} \leftarrow \mathcal{M}^{Best} \cup \{< M^{(F,best)}, \pi^{(F,best)} >\}$ ;
12 end
13 $M_P^{Best} \leftarrow bestModelAccrossDataSets(\mathcal{M}^{Best})$ ;
14 return $M_P^{Best}$;
```

L'algorithme 2 illustre les étapes principales du processus de production d'un modèle prédictif selon la nature du problème d'apprentissage, i.e. P, qui a été précisé en entrée.

Description des techniques d'apprentissage

Nous avons choisi les algorithmes les plus représentatifs des principales catégories des approches d'apprentissage supervisé (*cf.* Chapitre 1). Ainsi, nous avons assemblé, ensuite organisé le groupe $\mathcal{SA}$ des algorithme SML selon la nature du problème d'apprentissage. La Table 4.1 liste et décrit brièvement les divers algorithmes déployés dans l'ensemble $\mathcal{SA}$. Il est important de noter que toutes les implémentations utilisées dans nos évaluation sont appliquées avec les paramètres par défaut de Weka Toolkit [HFH+09]. Plus de détails et de descriptions sur ces algorithmes peuvent être trouvés dans [Abb14]. Outre les algorithmes d'apprentissage, nous avons choisi 3 différentes méthodes de sélection de variable. Ces méthodes ont été décrites et comparées dans [AT16], à savoir : *1)* la méthode

Algorithme	Catégorie	Tâche ML	Description
Random Forest (**RF**)	Logique	Classification+ Régression	Une combinaison de prophètes d'arbre de décision C4.5.
Simple Logistique (**SL**)	Logistique	Classification	Une régression logistique linéaire, utilisant *LogitBoost* avec une fonction de régression simple.
Perceptron multi-couches (**MP**)	ANN	Classification	Un algorithme de propagation arrière pour construire un modèle de réseau de neurones artificiel pour classer les cas.
Optimisation minimale séquentielle (**SMO**)	SVM	Classification+ Régression	Une implémentation de Machine de Vecteur de Support avec un noyau polynomial.
SVM L2-régularisé (**L2SVM**)	SVM	Classification+ Régression	Une autre implémentation de SVM avec un noyau linéaire.
K-Plus proche-Voisin (**IBk**)	Lazy	Classification+ Régression	Un algorithme basé cas s'exécutant avec une distance euclidienne normalisée.
Estimateurs moyennés N-Dépendance (**A1DE**)	Bayes naïf	Classification	Un algorithme de pondération sur un petit espace de modèles alternatifs "Bayes naïves".
Régression linéaire (**LR**)	Linéaire	Régression	Un algorithme d'usage courant qui essaie d'adapter les données à une courbe linéaire.

Table 4: Description des algorithmes d'apprentissage supervisé.

RELIF de filtrage de variables, noté par RLF; 2) la méthode d'élimination de corrélation des sous groupes de variable, noté par CSF; 3) la méthode de discrétisation supervisée proposée par Fayyad & Irani's et connu par MDL. Nous avons enrichi cet ensemble par deux méthodes supplémentaires que nous avons introduit. Il s'agit de MDL_RLF et MDL_CSF. Comme le nom l'indique, ces méthodes sont une combinaison des méthodes élémentaires RLF, CSF et MDL. En outre, nous avons choisi d'intégrer les bases d'apprentissage brutes dans notre processus. Ces bases comportent la totalité de nos descripteurs d'ontologie et elles sont notées par $RAWD$. Les différents méthodes de sélection de variable que nous avons listés constituent notre ensemble $\mathcal{FS}$.

Synthèse des résultats des évaluations des raisonneurs

Nous avons établi la condition de limite de temps à 3 minutes pour classer une ontologie par un raisonneur. Ce temps-limite serré est compatible avec le scénario choisi, *i.e.* la classification DL en

Raisonneur	**#Succès**	**#Échec**		
		Inattendu	**Hors temps**	**Arrêt**
Konclude	2408	44	2	6
MORe	2182	71	241	6
HermiT	2167	3	312	18
TrOWL	2149	257	74	20
FaCT++	1968	18	419	95
JFact	1679	41	636	144

Table 5: Synthèse des résultats d'évaluation des raisonneurs.

ligne. La Table 4.2 synthétise les résultats du challenge de classification d'ontologie effectué[13]. Elle illustre pour chaque raisonneur le nombre d'ontologies classées avec succès ; sinon le nombre d'ontologies traitées à temps avec des déductions correctes. Le nombres de cas où le raisonneur est marqué avec un échec est aussi reporté dans la Table 4.2 et séparé selon la nature de l'échec. Les raisonneurs sont listés dans la Table 4.2 dans l'ordre décroissant de leur taux de succès ; le raisonneur le plus robuste sur le corpus d'ontologies est celui en haut de la liste (Définition 23). De plus, la Figure 4.1 décrit le temps d'exécution moyen passé par chaque raisonneur sur les ontologies classées correctement, i.e. les cas de succès. Nous pouvons facilement remarquer que Konclude est le raisonneur le plus robuste ; cependant, ce système échoue à délivrer des résultats corrects pour 44 ontologies. D'autre part, TrOWL est le raisonneur le moins fiable avec 257 cas de résultats inattendus ; ceci serait dû à son incomplétude pour certains profils OWL. Néanmoins, TrOWL a le second temps d'exécution moyen le plus court pour des cas corrects. Nous devrions aussi remarquer qu'en dépit de la capacité d'HermiT à délivrer des résultats corrects, i.e. seulement 3 cas inattendus, son temps d'exécution moyen sur les cas corrects est très haut. Par conséquent, il pourrait être difficile de déployer ce raisonneur dans des applications en ligne. Tout ceci devrait souligner la difficulté à choisir un raisonneur absolument approprié pour toutes les ontologies. Évidemment, un raisonneur devrait être choisi selon ses performances pour une ontologie particulière donnée en entrée. Cependant, une telle tâche a un coût en terme de temps et d'effort puisqu'il exige beaucoup d'expérimentations. C'est pourquoi, l'apprentissage pour prédire automatiquement les performances d'un raisonneur semble une solution prometteuse. A partir des résultats décrits ci-dessus concernant les évaluations des raisonneurs, les bases d'apprentissage du profil prédictif de robustesse de chaque raisonneur, ainsi que la base d'apprentissage du modèle de prédiction du niveau de difficulté d'un ontologie seront construits.

[13]Les résultats de notre évaluation sont disponibles à `https://github.com/Alaya2016/OntoClassification-Results2016/`.

Synthèse des résultats de prédiction de la difficulté des ontologies

Le groupe de raisonneurs que nous avons choisi d'étudier dans cette thèse comporte les raisonneurs Hermit, MoRE, FaCT++ et JFact. Il s'agit d'une groupe homogène vu qu'ils implémentent des approches de raisonnement inspirées de l'algorithme du Tableau [BS01] et partagent diverses techniques d'optimisation [Hor03]. En partant des résultats d'évaluation des raisonneurs décrites ci-dessous, nous avons étiqueté chacune de nos ontologies par l'une des deux labels "Hard" ou bien "Safe". Rappelons, qu'une ontologie est dites difficiles ("hard"), lorsqu'au moins un raisonneur du groupe échoue à l'inférer tout en respectant nos contraintes de jugement de robustesses. Au total, nous avons identifié 958 ontologies difficiles, soit 38.32% de l'ensemble de départ. Après calcul des descripteurs de chaque ontologie, une dataset pour l'apprentissage du modèle de prédiction du niveau de difficulté empirique d'une ontologie a été construit en appliquant notre méthodologie générique d'apprentissage. Nous avons comparer la qualité de prédiction d'un ensemble des modèles candidats, générés par combinaison d'un ensemble d'algorithmes d'apprentissage et de méthodes de sélection de variable. Figure 4.3 est une synthèse des résultats de cette comparaison. Il est à noter que les modèles générés sont d'une bonne qualité en terme de F1TP. Le meilleur modèle est celui construit en utilisant l'algorithme d'apprentissage **L2SVM**, après application de la méthode de sélection de variable **RLF_MDL**. Cette dernière a identifié 51 descripteurs d'ontologie comme étant les plus pertinents pour l'apprentissage. Le F1-score de ce modèle de prédiction est égal à 0.947 alors que son $F1TP$ score est égal à 844.182. Ces valeurs sont remarquables et démontre que nous pouvons prédire la difficulté empirique d'une ontologie avec une bonne marge de précision.

Synthèse des résultats de construction des profils prédictifs de robustesse

En appliquant notre méthodologie d'apprentissage, nous avons aussi réussi à établir des modèles de prédiction de l'état de terminaison d'une tâche de raisonnement et de son temps de calcul. Ces modèles ont été construits pour chaque raisonneur de notre étude. Tableau 4.4 résume les résultats de notre analyse empirique des algorithme d'apprentissage et de méthode de sélection de variables. Le tableau décrit la qualité des modèles de prédiction, identifiés comme les "meilleurs" en terme des scores d'évaluation des performances que nous avons adopté. Nos résultats démontrent que l'exactitude ainsi que la robustesse des raisonneurs peuvent être prédites avec grande précision. Ainsi, nous avons composé le profil de robustesse prédictif pour chaque raisonneur examiné. A notre connaissance, nous sommes les premiers à tenter de découvrir plus d'un critère possible d'évaluation de raisonneur. Nous croyons que nos profils assemblés de raisonneurs et les modèles de difficulté globale d'ontologie sont les premières étapes pour construire une base de connaissance concise unissant les données relatives

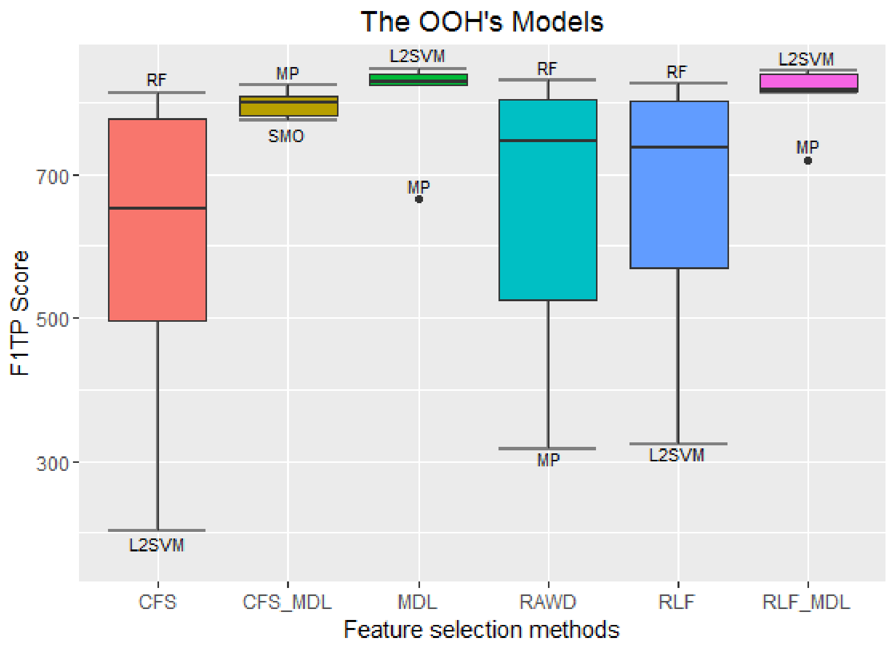


Figure 4: Comparison des scores F1TP des différents modèles candidats pour la préduction de la difficulté d'une ontologie.

aux raisonneurs DL actuels existants et les ontologies OWL. Nous croyons que ce type de connaissance devrait fournir une assistance pour les concepteurs de raisonneur et d'ontologie pour un meilleur examen de leurs réalisations et, cela, de façon plus facile. De plus, l'amélioration de notre cadre pourrait être envisagée afin d'obtenir de meilleurs modèles prédictifs en moins de temps d'apprentissage. Par exemple, nous aimerions étudier plus de techniques récentes de discrétisation. Il serait aussi intéressant d'examiner plus de solutions de pointe pour manipuler le problème des datasets déséquilibrés, tels que le sur/sous échantillonnage de cas ou l'apprentissage "sensible coût". D'autre part, nous supposons que la réduction du temps global d'apprentissage exigerait plus de métaconnaissance sur les algorithmes d'apprentissage. En effet, nous pouvons considérer que notre recherche pour identifier les techniques les plus appropriées selon nos datasets d'apprentissage est une sorte de problème d'algorithme de sélection. D'où, examiner les solutions de méta-apprentissage [BGCSV08] est certainement une possibilité attirante. Cependant, elle ne devrait pas être surestimée puisque nous ne sommes pas sûrs que les techniques existantes conviendraient à nos diverses et particulières exigences et sans aucun coût de calcul supplémentaire.

Synthèse générale

Les questions de recherche dont nous avons discuté dans ce chapitre sont à la croisée de deux champs : le Web sémantique et l'apprentissage machine. Notre propos était d'utiliser des techniques

	Success models $M^{Best}_{(S\|R)}$					Time models $M^{Best}_{(T\|R)}$					
Reasoner	**FS**	**SA**	**MKS**	**F1-Score**	**#F**	**FS**	**SA**	R^2	**RMSE**	**RRS**	**#F**
Konclude	RLF_MDL	L2SVM	0.653	0.974	53	CFS	RF	0.987	0.334	0.004	20
MORe	RLF_MDL	MP	0.827	0.960	65	CFS	RF	0.963	0.251	0.009	28
HermiT	RLF_MDL	SL	0.889	0.972	67	CFS	RF	0.968	0.383	0.012	16
TrOWL	RLF_MDL	RF	0.787	0.951	55	CFS	RF	0.975	0.291	0.007	16
FaCT++	RLF_MDL	L2SVM	0.881	0.956	63	RLF	RF	0.944	0.455	0.026	22
JFact	RLF_MDL	RF	0.904	0.952	60	CFS	RF	0.971	0.386	0.011	13

Table 6: Assessment Summary of the Best Trained Predictive Models

d'apprentissage machine supervisé pour être capable de détecter la variabilité des performances empiriques des raisonneurs DL. Nous avons porté notre attention sur deux problèmes d'apprentissage à savoir, d'une part, la prédiction de la difficulté globale des ontologies OWL au regard d'un groupe de raisonneurs et, d'autre part, l'anticipation de la robustesse d'un raisonneur unique lors de déduction d'ontologies sous certaines contraintes d'usage en ligne. Pour atteindre cet objectif, nous avons établi un cadre générique d'apprentissage supervisé qui intègre les caractéristiques des ontologies que nous avons proposées dans le chapitre précédant. Dans la suite de ce mémoire, nous introduirons deux champs d'application où les modèles prédictifs établis pourraient être utilisés.

0.6 Résumé du chapitre 5 : Système générique d'assistance au raisonnement autour d'ontologies

Nous rappelons que le but principal de cette thèse est d'apporter des solutions pour faire face au phénomène de variabilité des performances de raisonneur et, par conséquent, faire progresser l'état de l'art dans l'analyse empirique des raisonneurs d'ontologies. La finalité de ce travail est de fournir des directives simples et efficacement calculables destinées aux novices tout comme aux experts afin d'anticiper, d'une part, la *difficulté* empirique des ontologies OWL, et, d'autre part, la *robustesse* empirique des raisonneurs. Ces directives sont principalement un ensemble de prédictions sur les comportements attendus d'un raisonneur ou bien un classement d'un groupe de raisonneurs en fonction des caractéristiques d'une ontologie donnée en entrée.

Pour obtenir ceci, nous avons modélisé un système intelligent pour fournir une assistance à l'utilisateur pour établir le choix d'un raisonneur. Nous avons appelé ce système **ADSOR**, pour "*Advising System Over the Ontology Reasoners*". ADSOR fournit deux types d'assistance à un util-

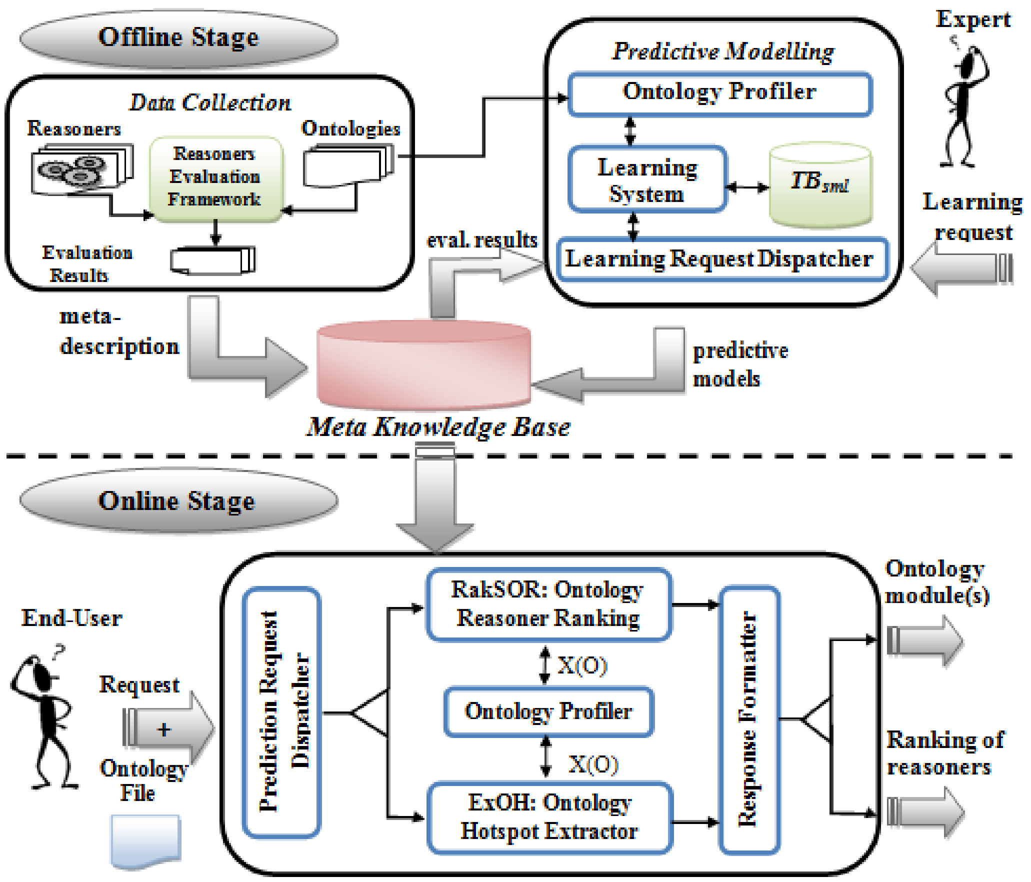


Figure 5: Architecture générale du système ADSOR.

isateur final : *(i)* il suggère un classement de raisonneurs selon leurs performances prédites par rapport à l'ontologie d'entrée. L'ordonnancement, dit souvent le *"ranking"*, est fourni dans le but d'aider l'utilisateur à choisir le raisonneur le plus adéquat pour son application ; et *(ii)* il permet d'identifier et d'extraire les sous-parties les plus complexes d'une ontologie empiriquement difficile. Ces extraits sont des points de départ fournis à l'utilisateur afin de lui permettre d'établir des analyses plus approfondies de son ontologie et potentiellement d'entrainer la révision de son modèle conceptuel.

Description générale du système ADSOR

Une description détaillée du composant principal d'ADSOR, ainsi qu'une analyse expérimentale rigoureuse de leurs performances respectives sont fournies dans ce mémoire. Nous commençons à décrire l'architecture conceptuelle globale d'ADSOR (*cf.* Figure 5.2).

Étape hors ligne : mode d'acquisition des connaissances. Le but principal de cette étape est d'entraîner les modèles prédictifs sur divers problèmes autour du raisonnement d'ontologie. Les modèles sont acquis par une exploration intelligente de données décrivant des performances empiriques sur un grand ensemble d'ontologies. Par conséquent, collecter ces données empiriques est la première étape dans l'acquisition des connaissances. ADSOR utilise le cadre d'évaluation d'ORE [GBJR+13] qui est disponible au public pour extraire des évaluations régulières sur les raisonneurs DL de pointe ; et, ainsi, ADSOR délivre les données nécessaires sur lesquelles nous fondons le processus d'apprentissage. Ce dernier est déclenché en suivant un utilisateur expert et est terminé par notre *système d'apprentissage*. Dans le Chapitre 4, nous avons introduit quelques caractéristiques fonctionnelles de ce système. Nous fournissons ici plus de détails. Un autre composant également important est le *profileur d'ontologie* que nous avons présenté dans le Chapitre 3. Il caractérise la complexité de conception des ontologies étudiées en utilisant une riche collection des caractéristiques efficacement calculables. Certains de ces modèles entraînés sont produits par l'étape d'apprentissage décrit précédemment dans le Chapitre 4 tels que les profils de robustesse prédictive des raisonneurs et les ontologies sur les modèles de difficultés. Selon les auteurs respectifs [SM09, Van11], en combinant les caractéristiques avec les métriques de performance sur un grand nombre d'ontologies déduites par différents raisonneurs, cela crée un ensemble compréhensible de métadonnées sur les performances empiriques de raisonneur. Ces métadonnées associées avec les modèles prédictifs entraînés représente la connaissance de fond sur les raisonneurs DL impliqués. L'expertise apprise est stockée dans un format interprétable par les machines dans une base de données spécifiques, appelée communément *Base de métaconnaissances*. Par conséquent, elle pourrait être extraite automatiquement et appliquée sur de nouveaux problèmes.

Étape en ligne : mode d'assistance. ADSOR propose deux types de conseils aux utilisateurs d'ontologies. Ces conseils sont fondés sur la connaissance acquise sur la robustesse empirique des raisonneurs et le niveau de difficulté empirique des ontologies OWL. Les conseils sont demandés par un utilisateur final sur une ontologie donnée en entrée. Il s'agit de lui proposer un classement des raisonneurs selon les performances prédites étant donnée l'ontologie d'entré. Ce classement est une sorte de recommandation fournie à l'utilisateur dans le but de l'aider à sélectionner le raisonneur le plus robuste capable de traiter son ontologie efficacement et avec exactitude. Le deuxième conseil est offert dans le cas où l'ontologie est prédite comme état empiriquement difficile. Dans ce cas, un ensemble de modules logiques sont extraits de cette ontologie. Ces modules sont susceptibles d'être les sous parties de l'ontologie responsable de son niveau de difficulté. Dans la suite, nous décrivons brièvement les composants principales d'ADSOR.

RakSOR: Ordonnancement des raisonneurs d'ontologie par apprentissage supervisé

Ce composant conseille les utilisateurs potentiels au regard des *meilleurs* raisonneurs disponibles en tenant en considération la nature et les caractéristiques de conception de son ontologie. Les conseils fournis sont dans leur forme basique, un classement des raisonneurs dans un ordre décroissant de préférence, *i.e* le plus préféré possède le rang le plus faible.

Fournir des explications au regard du classement ou des résultats d'extraction est une des exigences principales qui avait guidé le choix de nos techniques de conseils. Dans [Van11], J. Vanschoren a mis en lumière qu'un système d'assistance à l'utilisateur devrait fournir la logique des conseils proposés. D'un autre côté, il est important d'expliquer à l'utilisateur les composants de base qui ont amené à de tels résultats et de lui offrir autant de détails que possible. Un tel type d'information devrait lui permettre d'accepter l'aide fournie ou peut-être la rejeter et se tourner vers des propositions alternatives. Dans le mémoire, nous décrivons plus en détails la conception de premier niveau des composants de classement et d'extraction.

La robustesse d'un raisonneur est le critère principal de classement, ceci implique l'examen de l'exactitude des résultats et de l'efficacité des traitements. Pour intégrer ces propositions dans une application, nous avons formulé un nouveau cadre d'ordonnancement par apprentissage supervisé des raisonneurs d'ontologie. Nous avons étudié deux sortes de stratégies de classement. Une première méthode de classement fondée sur une prédiction d'un unique label par raisonneur candidat. Elle tire avantage des profils de raisonneurs que nous avons construit. Après avoir prédit les performances de chaque raisonneur sur une ontologie d'entrée, un algorithme les classe selon les règles de préférence que nous avons préciser. La deuxième méthode est fondée sur des technique de classification et de régression multi-labels. Il s'agit de construire un seul modèle de prédiction capable de fournir le classement complet des raisonneurs en tenant en compte les descripteurs de l'ontologie d'entré. Figure ?? décrit les étapes majeures du processus d'apprentissage de chacune de nos stratégies.

Relation d'ordre entre raisonneurs Les deux stratégies d'apprentissage d'ordonnancement respecte un ensemble de règles de classement des raisonneurs. Ces règles traduisent nos préférences quant à l'adéquation d'un raisonneur pour une ontologie tout en tenant en compte les deux critères d'efficacité et d'exactitude. Intuitivement, en règle générale, le raisonneur avec le plus rapide succès est préféré aux autres. Sauf dans certain cas, deux raisonneurs peuvent échouer à traiter une ontologie pour la même cause, par example, un état de Timeout. Ainsi, il serait impossible d'établir un ordre de préférence entre ceux deux raisonneurs. Alors, ils sont dits incomparables. Partant de ces constats, nous pouvons affirmer que dans notre contexte, un recourt à une modélisation fondée sur la notion d'ordre partiel [BCLS06] s'impose. Plus spécifiquement, nous avons choisi de se référer à un

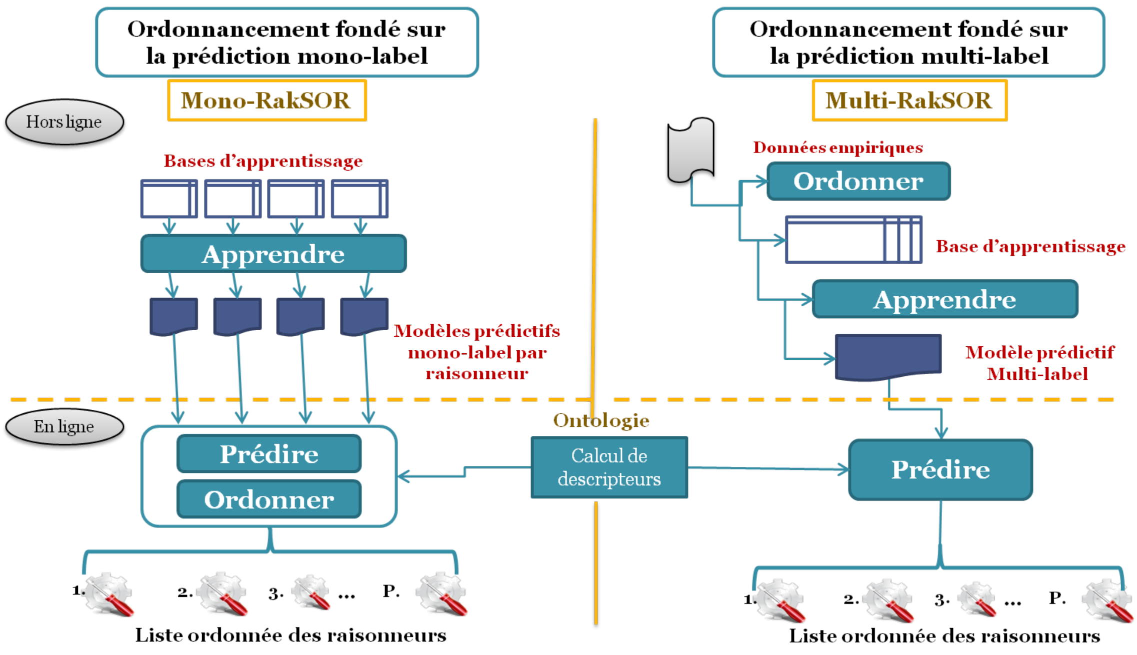

Ordonnancement fondé sur la prédiction mono-label
Mono-RakSOR
Hors ligne
Bases d'apprentissage
Apprendre
Modèles prédictifs mono-label par raisonneur
En ligne
Prédire
Ordonner
1. 2. 3. ... P.
Liste ordonnée des raisonneurs
Ontologie
Calcul de descripteurs
Ordonnancement fondé sur la prédiction multi-label
Multi-RakSOR
Données empiriques
Ordonner
Base d'apprentissage
Apprendre
Modèle prédictif Multi-label
Prédire
1. 2. 3. ... P.
Liste ordonnée des raisonneurs

Figure 6: Mise en correspondance entre les deux approches d'ordonnancement des raisonneurs par apprentissage.

cas particulier d'ordre partiel, dit ordre par paquet ("Bucket Order") [FKM+06]. Selon cet ordre, les alternatives au classement sont regroupé en paquets distincts. Les alternatives dans le même paquet ont le même rang sachant que les paquets sont classés dans un ordre linéaire strict. En considérant nos critères de classement, les paquets correspondent aux différents états de terminaison possibles d'une tâche de raisonnement. Nous avons auparavant spécifier 4 états, par la suite 4 paquets que nous avons ordonné comme suit: "Succès" $\prec$ "Inattendu" $\prec$ "Timeout" $\prec$ "Arrêt", tel que $\prec$ est la relation qui correspond à la notion "*est préféré à*". Donc, les raisonneurs sont d'abord répartis sur les différents paquets, ensuite ceux dans le paquet de succès subisse un deuxième classement qui tien en compte leurs temps d'exécution. Ainsi, le premier paquet est éclaté en plusieurs paquets à élément unique, triés dans un ordre croissant. Figure 7 explique le mécanisme d'ordre par paquet que nous avons proposé. Un algorithme a été conçu afin de calculé cet ordre avec efficacité. Plus de descriptions sont établis dans ce mémoire.

ExOH: Extraction des modules difficiles d'une ontologie

Au delà de la sélection de raisonneur, nous avons proposé un autre type de conseil pour les utilisateurs finaux cherchant à gagner en compréhension de leurs ontologies. En effet, quand une ontologie est reconnue comme étant difficile à gérer, un utilisateur cherchera à détecter les causes de cette difficulté. Dans ce cas, nous suggérons de lui fournir la possibilité d'extraire les sous-parties de son ontologie qui

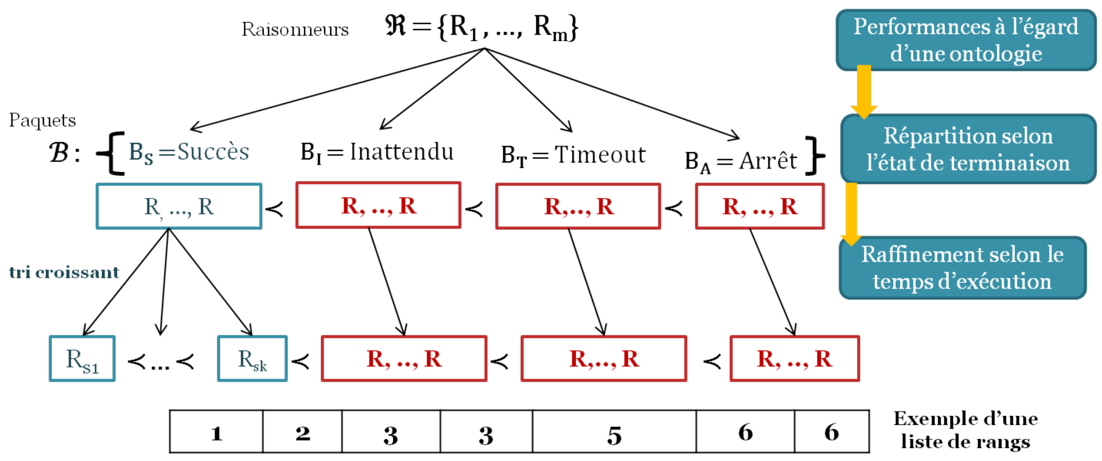

Raisonneurs
$\mathfrak{R} = \{R_1, ..., R_m\}$
Performances à l'égard d'une ontologie
Paquets
$\mathcal{B}$: B_S = Succès B_I = Inattendu B_T = Timeout B_A = Arrêt
Répartition selon l'état de terminaison
Raffinement selon le temps d'exécution
tri croissant
R_{s1} $\prec ... \prec$ R_{sk}
1 2 3 3 5 6 6
Exemple d'une liste de rangs

Figure 7: La relation d'ordre par paquet appliquée aux raisonneurs d'ontologie.

sont potentiellement les plus exigeantes en terme de coût de raisonnement. Dans la littérature, ces sous-parties ont été appelées des *hotspots* d'ontologie [GPS12]. Les hotspots sont des goulots d'étranglement de performance lors du raisonnement, et ils représentent des opportunités de remaniement pour les concepteurs d'ontologie. Aussi, une identification efficace de ces composants est hautement nécessaire. Nous avons revisité les approches respectives de Gonçalves et al. [GPS12] et Kang et al. [KLK14] d'extraction de hotspots. Nous avons suggéré une nouvelle approche qui repose sur la représentation de graphe de dépendance des axiomes d'une ontologie, dit graphe d'atomes [DPSS11]. Un atome est un groupe minimal d'axiomes qui co-occurrent ensemble dans les différents modules ontologique de localité syntaxique [GHKS08], dont l'extraction est possible à partir d'une ontologie. Toujours d'après Grau et al. [GHKS08], le nombre de ces modules peut être exponentiel par rapport à la taille de l'ontologie. Ainsi, le graphe d'atome peut être considérer comme une représentation compacte de tous les modules logiques de localité d'une ontologie. Partant de cette propriété, nous avons choisi de calculer la décomposition atomique de l'ontologie. Ensuite, les modules logiques sont construits à partir des signatures des atomes, identifié comme les plus *pertinent*. Nous avons proposé deux stratégie de parcours du graphe d'atome dans le but d'identifier les atomes candidats pour l'extraction. Fur et à mesure de l'extraction, l'identification des modules complexes est appliquée par recourt à notre modèle de prédiction du niveau de difficulté d'une ontologie. Figure ?? décrit les principales étapes qui composent le processus d'extraction et d'identification des modules ontologiques difficiles.

Dans nos évaluations, nous avons opté pour à une nouvelle méthode d'extraction des modules logiques de localité syntaxique et de décomposition atomique d'une ontologie. Cette approche a été introduite par Martín-Recuerda and Walther [MRW14]. Elle a outrepassé les méthodes existante en terme d'efficacité. Son point fort, c'est l'utilisation des hypergraphes [GLPN93] pour modéliser la dépendances entre les axiomes d'une ontologie. Les premiers résultats expérimentaux que nous avons

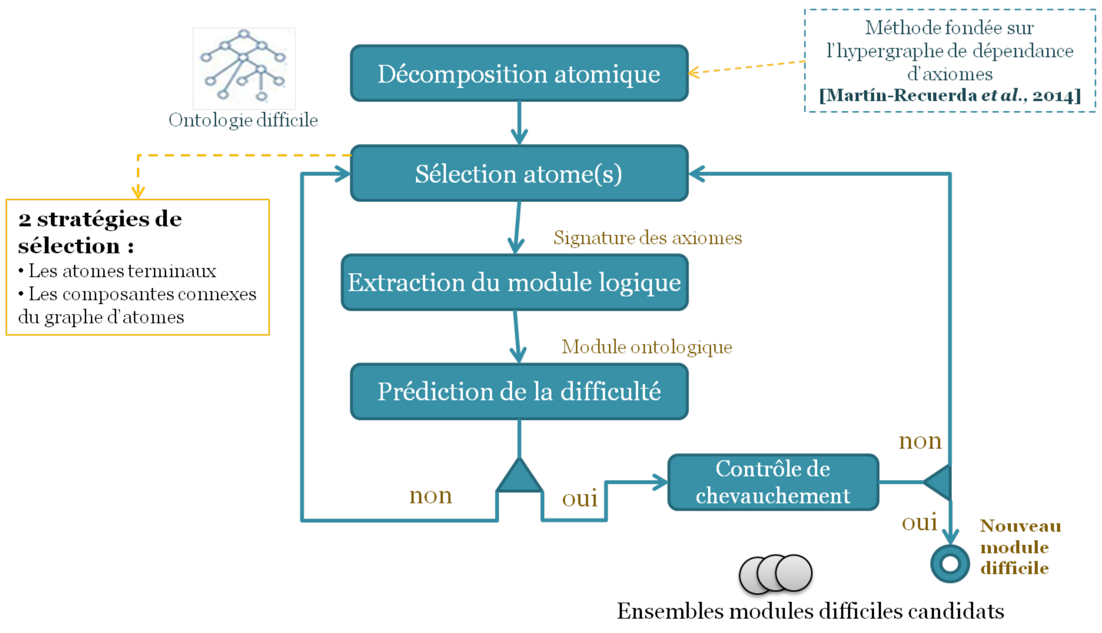

Ontologie difficile
Décomposition atomique
Méthode fondée sur l'hypergraphe de dépendance d'axiomes [Martín-Recuerda et al., 2014]
Sélection atome(s)
2 stratégies de sélection :
• Les atomes terminaux
• Les composantes connexes du graphe d'atomes
Signature des axiomes
Extraction du module logique
Module ontologique
Prédiction de la difficulté
non
oui
Contrôle de chevauchement
non
oui
Nouveau module difficile
Ensembles modules difficiles candidats

Figure 8: Description générale de l'approche ExOH.

décrit dans ce mémoire sont encourageants, témoignant du potentiel de notre approche.

0.7 Conclusion

Dans cette section, nous résumerons les principaux résultats de cette thèse. Tout d'abord, nous allons rappeler notre problématique de recherche, ensuite nous allons décrire les traits principaux de la démarche de résolution, que nous avons détaillée lors des chapitres précédents. Finalement, nous allons discuté la suite logique de nos avancées par le biais d'un certain nombres de nos perspectives de recherche.

Principales contributions

Cette dernière décennie, plusieurs algorithmes de raisonnement sémantique et de raisonneurs ont été proposés pour restreindre la complexité du calcul des tâches de déduction pour des langages expressifs d'ontologie tels que OWL 2 DL. De gros efforts de recherche ont été dédiés à la conception d'algorithmes efficaces de raisonnement avec des techniques d'optimisation hautement sophistiquées. Néanmoins, des compétitions annuelles de raisonneurs ont révélé qu'il n'y a pas de moteur de raisonnement qui puisse être performant sur toutes les ontologies données en entrée. *Grosso modo*, les performances d'un raisonneur sont largement dépendantes du succès ou de l'échec de ses astuces d'optimisation. Un raisonneur pourrait donc être optimisé pour certaines ontologies mais non pour d'autres. En conséquence, un phénomène de variabilité des performances empiriques de raisonneur est souvent observé en pratique. De plus, il y a un réel manque de connaissances et d'outils d'assistance devant aider à pointer les caractéristiques d'ontologie qui sont des facteurs de dégradation des performances des raisonneurs. Aussi, des règles de décision pré-établies par des experts pour tester l'adéquation d'un raisonneur pour une application basée ontologie ne sont pas efficaces en pratique, hormis pour les cas triviaux. Toutes ces découvertes ont donné naissance à de nouveaux défis dans ce champ du raisonnement. En effet, un utilisateur doit faire face aux problèmes suivants:

"Comment obtenir un aperçu de la difficulté empirique d'une ontologie? Comment établir un choix intelligent d'un raisonneur pour une certaine ontologie, de façon à aboutir à des meilleures performances que celle achevées par un raisonneur sélectionné arbitrairement ?"

Cette thèse a traité ces questions et a essayé de fournir des réponses pertinentes, faciles à comprendre pour un utilisateur final. Les solutions proposées dépendent d'une analyse empirique algorithmique et utilisent des techniques d'apprentissage supervisé. Un des principaux avantages de cette analyse empirique est qu'elle permet d'étudier à quel point la performance algorithmique dépend statistiquement des caractéristiques du problème. Par conséquent, les caractéristiques clés d'ontologie pourraient être identifiées en essayant de traquer les performances des raisonneurs. Les dépendances ont été modélisées comme des fonctions prédictives capables d'anticiper les performances d'un raisonneur pour une ontologie donnée. Ces ontologies devraient être décrites par un ensemble de caractéristiques. Un autre avantage de ce type d'analyse est qu'elle permet de comparer et de trier automatiquement différents raisonneurs selon leurs performances attendues sur la même ontologie. En effet, les futurs comportements empiriques de raisonneurs pourraient être appris par une exploration intelligente de leurs exécutions passées. Par conséquent, aucune expérimentation supplémentaire ne serait nécessaire pour décider quel raisonneur choisir.

Cette recherche a induit plusieurs contributions dans le champs du raisonnement avec des ontologies. Indiquons succinctement nos contributions dans les champs disciplinaires distincts mais complémentaires suivants :

(i) Nous avons proposé un ensemble riche de caractéristiques, complet et efficacement calculable, pour modéliser la complexité de conception des ontologies OWL. Nos caractéristiques ont été classées en quatre catégories couvrant un large spectre de propriétés syntaxiques, structurelles et sémantiques des composants d'une ontologie.

(ii) Nous avons proposé de régler la variabilité des performances des raisonneurs en les prédisant automatiquement pour toute ontologie donnée en entrée. Nous avons étudié aussi la robustesse des raisonneurs. Par robustesse, nous signifions la capacité d'un raisonneur à terminer une tâche de raisonnement de façon correcte et en un temps limite fixé. Nous avons fait valoir que la robustesse devrait être déclarée sous certaines contraintes computationnelles et fonctionnelles. C'est pourquoi nous avons choisi d'examiner un scénario d'usage en ligne où la classification DL d'ontologie est la tâche principale requise.

(iii) Nous avons appliqué certains concepts d'apprentissage supervisé au problème de la prédiction empirique de la robustesse de raisonneur. Nous avons proposé une méthodologie d'apprentissage qui utilise notre suite de caractéristiques d'ontologie et qui dépend d'un ensemble riche de techniques avancées de l'apprentissage machine supervisé. Grâce à cette proposition, nous avons modélisé un profil de robustesse de raisonneur en matière de modèles prédictifs capable d'anticiper sa justesse et son efficacité quelle que soit l'ontologie à manipuler.

(iv) En utilisant la même méthodologie, nous avons appris à prédire les ontologies difficiles empiriquement. Rappelons que nous parlons de "difficile" lorsque ces dernières sont difficiles à gérer par un ensemble de raisonneurs.

(v) L'efficacité des systèmes d'apprentissage proposés a été examinée au travers de très nombreuses expérimentations. Nous avons étudié six raisonneurs bien connus de la communauté et les avons testés avec 2500 ontologies. Nous avons comparé divers algorithmes d'apprentissage supervisé et techniques de sélection de caractéristiques dans le but d'entraîner les meilleurs modèles prédictifs de raisonneur. Nous avons montré que tous nos modèles surpassaient les solutions de pointe hautement performantes actuelles.

(vi) Les modèles entraînés ont été comparés et étudiés dans le but de découvrir les caractéristiques d'ontologie les plus importantes. Certaines ont été décelées comme obligatoires pour assurer une bonne qualité de prédictions. Autrement dit, elles sont fortement corrélées aux performances des raisonneurs examinés. Nous pensons qu'une telle connaissance améliorerait des études futures sur des méthodes de modélisation d'ontologie afin de faire en sorte que les facteurs de dégradation des performances des raisonneurs soient évités.

(vii) Nous avons fourni un nouveau cadre de travail pour soutenir les utilisateurs non experts dans la tâche de sélection de raisonneur. Le but principal de ce cadre est d'être capable de classer automatiquement un ensemble de raisonneurs candidats selon leur robustesse attendue et ce, sur n'importe quelle ontologie. Notre cadre de classement suit une approche de méta-apprentissage et opère en deux phases : une phase hors ligne pour acquérir la connaissance ; et une phase pour appliquer les règles de classement. Ce classement est calculé selon des règles de préférence qui tiennent compte à la fois de l'exactitude et de l'efficacité des raisonneurs examinés. Ce cadre produit un ordonnancement des raisonneurs qui applique une relation d'ordre par paquet. Celle-ci est un cas spécial d'une relation d'ordre partiel. A notre connaissance, nous sommes les premiers à considérer plus d'un critère dans le processus de classement des raisonneurs.

(viii) Nous avons proposé deux stratégies d'ordonnancement par apprentissage supervisé qui appliquent les règles de préférence établies. La première stratégie est fondée sur une prédiction d'un unique label. Il tire avantage des profils de raisonneurs appris. Après avoir prédit les performances de chaque raisonneur sur une ontologie d'entrée, un algorithme les classe selon les règles de préférence. La seconde stratégie est fondée sur des techniques multi-labels. Un modèle prédictif est entraîné afin d'être capable de coupler les caractéristiques d'une ontologie à un classement complet des raisonneurs. Un nouvel algorithme de classement multi-labels a été introduit. Le propos principal de cet algorithme est de produire un modèle capable d'appliquer nos règles de préférence. Tous nos résultats expérimentaux témoignent de l'efficacité de nos propositions. Nous avons réalisé un classement de grande qualité qui a surpassé ceux des solutions existantes.

(ix) Nous avons aussi examiné la possibilité de l'utilisation d'un modèle prédictif de difficulté d'ontologie pour guider l'identification des zones sensibles, dites "*hotspots*", des ontologies. Nous avons cherché à identifier ces sous-parties d'ontologie. La solution dépend de la décomposition atomique de l'ontologie et du mécanisme d'extraction de la localité. Une fois que les atomes de l'ontologie ont été calculés et transformés en modules, la difficulté de ces derniers est prédite en utilisant notre modèle de prédiction. Nos premières évaluations ont montré des résultats encourageants, qui prouvent le potentiel de notre solution.

(x) Toutes nos propositions ont été rassemblées et modélisées comme des services distincts fournis par un système, appelé ADSOR (pour "*Advisor System over Ontology Reasoners*"). Le système a été conçu pour permettre les différentes opérations d'apprentissage dans la phase hors ligne et pour interagir avec des requêtes utilisateurs lors de la phase en ligne.

Perspectives

Tout au long de notre projet de recherche, nous avons fait un certain nombre de choix pour assurer le succès de nos travaux. Cependant, certains de ces choix pourraient avoir limité la portée du travail présenté. En conséquences, un certain nombre de questions demeurent et plusieurs directions futures de recherche sont concevables. Nous présentons plusieurs possibilités mais nous soulignons que cette liste n'est en aucun cas exhaustive.

Méthode alternative de vérification de la justesse d'un raisonneur Par notre étude, nous avons mis en exergue la nécessité d'examiner la justesse des résultats de raisonnement puisque les raisonneurs devraient fournir des résultats assez distincts sur la même ontologie d'entrée. La justesse est un critère important lorsque l'on conseille un utilisateur sur la pertinence d'un raisonneur. Cependant, très peu de travaux ont parlé du problème de la vérification automatique de la justesse. Les recherches dans ce domaine commencent à peine. Dans notre projet, nous avons utilisé une méthode basique de vérification de la justesse dépendant du mécanisme de vote majoritaire. Assurément, ce n'est pas une méthode irréprochable, par conséquent, il serait intéressant d'examiner des méthodes plus avancées. Nous devrions mettre en évidence que le changement de méthodes de vérification de la justesse de raisonneur ne devrait pas affecter nos méthodologies respectives d'apprentissage et de classement. Uniquement le processus d'évaluation de raisonneur et, par conséquent, les données collectées subiront ces changements. Nous pensons que l'utilisation de méthodes plus fiables de vérification de justesse engendrerait des conseils plus significatifs et ainsi une meilleure satisfaction de l'utilisateur.

Évolution du système de classement de raisonneur Nous avons introduit les premières idées de conception de ce que pourrait être un système de recommandation de raisonneurs d'ontologie. Néanmoins, des améliorations supplémentaires devraient être faites. Nos futures propositions vont vers un système de recommandation de raisonneur plus évolué selon les trois points suivants :

1. Augmentation du nombre de scénarios d'usage de raisonneur Dans notre étude, nous nous sommes focalisés sur un scénario de classification des ontologies OWL. Nous nous sommes fixés un calendrier très serré et nous avons fourni un travail de mémoire relativement important. En perspective, nous avons l'intention de regarder de plus près des scénarios tels que l'utilisation de raisonneurs dans des dispositifs mobiles. Dans ce contexte, la consommation mémoire devrait être réduite tout en gardant le même degré d'efficacité. C'est un vrai challenge pour les raisonneurs. Par conséquent, prévoir leur robustesse sous de telles contraintes est digne d'intérêt. Nous aimerions aussi étudier le cas où plus de contraintes de relâchement sont imposées. En fait, notre but ultime est de conduire différentes campagnes d'évaluation de raisonneurs avec des machines standard, des contraintes temporelles et une certaine configuration mémoire. Une fois que ces données sont rassemblées et leurs modèles prédictifs entraînés, de nouveaux critères pourraient être proposés qui prennent en compte les contraintes computationnelles étudiées.

2. Prise en comte des préférences utilisateur dans le classement Toutes les règles de préférence que nous avons utilisé pour classer les raisonneurs sont par défaut codées dans le système. Il devrait être aussi possible d'interroger l'utilisateur sur sa propre préférence et sur les conditions de classement. Une telle information pourrait être traduite en des poids de critère ou d'autre forme de règles de

préférence. Dans ce cas, il serait digne d'intérêt d'employer une approche de prise de décisions multi-critères, telles que Outranking ou MAULT, pour classer les raisonneurs.

3. Analyse du retour de l'utilisateur Dans les systèmes de recommandation standard, les retours d'utilisateur sur les recommandations fournies sont souvent demandés et analysés pour juger de la qualité du système. Dans certains systèmes de recommandation d'élément, ces commentaires sont vus comme de la connaissance auxiliaire précieuse et sont réutilisés pour améliorer les recommandations futures. Dans notre contexte, ceci ne pourrait pas être appliqué puisque nos conseils dépendent uniquement des comportements des raisonneurs. Cependant, il serait intéressant de demander des retours à l'utilisateur comme, par exemple, demander à quel point nos conseils lui ont donné satisfaction.

Vers un système de méta-raisonnement Nous avons abordé le problème de sélection de raisonneur du point de vue de l'utilisateur final. Nous avons calculé les classements pour lui fournir des indications. Néanmoins, nous sommes convaincus que nos méthodes de classement pourraient être utilisées pour construire un méta-raisonneur. Ce dernier est fondé sur une approche d'algorithme portfolio. Pour assurer que le méta-raisonneur surpasse les moteurs existants, Le temps de dépassement dû aux étapes de prédiction devrait être réduit. Par conséquent, des étapes d'optimisation supplémentaires devraient être considérées. Une optimisation possible est de mettre un raisonneur par défaut à lancer quand l'ontologie à étudier est un cas trivial. Chainsaw [TP12] est un autre type de méta-raisonneur. Il extrait une module de l'ontologie pour chaque requête utilisateur et crée un raisonneur adaptable à ce module pour être capable de répondre à la requête. Les auteurs ont souligné qu'ils cherchent encore une stratégie pour choisir le raisonneur le mieux adapté à un module donné. A l'heure actuelle, ils travaillent avec un couple de raisonneurs de façon arbitraire avec aucune sélection automatique concrète. Nous sommes prêts à intégrer notre méthode de classement dans ce méta-raisonneur et à conduire une nouvelle étude à grande échelle qui devrait nous aider à prouver les avantages de notre méthode de classement supervisé.

Introduction

Context

When the vision of the Semantic Web was formulated by Berners-Lee et al. [BLHL01], they have given great attention to ontologies. The Semantic Web of future would represent information more meaningfully for humans and computers alike and the ontologies are the backbone of this representation. Indeed, an ontology can provide a common vocabulary, a grammar for publishing data, and can supply a semantic description of the web pages. Henceforth, ontologies are used as conceptual models for data integration in a variety of domain areas.

An ontology is typically encoded in a Knowledge Representation (KR) language. A major class of ontology languages is based on Description logics (DLs) [BCM+10], which are decidable fragments of the First Order Logic (FOL). The Web Ontology Language, OWL [HPSV03] is a notable example of this class of languages. OWL is widely used in both academic and industrial environments, since it is recommended by the World Wide Web Consortium (W3C) as a standard language for the ontology authoring. An OWL ontology may be viewed as a set of FOL formulae, called *axioms*, and it is therefore a logical theory. Hence, the vocabulary described in an OWL ontology is provided with a well-defined, unambiguous meaning. However, there is often a mismatch between the intended and the actual representation of knowledge within an ontology. Whilst the knowledge of the domain of interest is generally well-understood, or at least agreed on, it is not trivial to foresee and understand the logical consequences of adding, removing, or modifying an axiom, especially when the terminology is highly interconnected. Hence, an ontology engineer needs to employ automated deduction systems, so-called *reasoners*, to verify certain forms of consequences of their stated axioms and to make the implicitly asserted ones, explicit. Indeed, *reasoners* are for now the key components to work with ontologies. Given the crucial role and the high importance of reasoners, considerable research and engineering efforts was dedicated to propose new semantic reasoning algorithms [BS01, MSH09, SKH11] with highly optimized implementation, such as FaCT++ [TH06], Pellet [SPG+07] and HermiT [GHM+12]. Matentzoglu et al. [MLH+15] have recently scrutinized this research effort in a detailed survey.

Unfortunately, the proliferation of ontologies came with a cost. Ontologies continue to gain in importance, as well as in size and complexity at a rate that has outpaced the cognitive capabilities of human to understand and to manually handle the modelled knowledge. This cognitive complexity have lead humans to rely on reasoners to check and manage their ontologies. Whereas, reasoners are also gaining in complexity and their empirical behaviour is often hazy even to experts. In fact, it exists a real tradeoff between the logical expressivity of the ontology language and the computational complexity of the DL reasoning: the more expressive a language, the higher its computational complexity. Horrocks et al. [HKS06] have shown that the reasoning problems over ontologies written in $\mathcal{SROIQ}$, the description logic underpinning OWL 2 DL[14] are, in the worst case solvable by a non-deterministic algorithm in time that is (double) exponential with respect to the size of the input. Hence from a theoretical perspective, one would assume that reasoning with OWL ontologies is infeasible in practice. Luckily, various works including [MRW11, Abb12] have settled that in general reasoning from an empirical perspective is far less complex than the established theoretical complexity. Interestingly enough, even with fairly expressive fragments of OWL 2, acceptable reasoning performances could be achieved. This would be owing to the various sophisticated reasoning optimization techniques [Hor03, THPS07]. Nevertheless, the respective authors of [WLL+07, GPS12] have outlined that often in practice, reasoners tend to exhibit *unpredictable* behaviours when dealing with real world ontologies and their performances dramatically vary across the ontologies, even when the size or the expressivity of the latter ones are held constant. Indeed, the observed sparsity over reasoner performances have prompted the organization of several reasoner benchmarks [DCtTdK11, Abb12] and annual reasoner evaluation competitions, such as the Ontology Reasoner Evaluation Workshop, ORE [GBJR+13, BGJ+14, DGG+15]. All of them attempt to compare the advances in reasoning systems and examine their robustness facing real world ontologies. The empirical results of these evaluations are often surprising the experts as reasoner performances, may remarkably vary on different ontologies belonging to the same theoretical complexity class. Furthermore, it is well-accepted that there is no reasoning engine that can outperform in all input ontologies. From an empirical perspective, there is merely a general agreement [GTH06, WLL+07, WP07, GPS12, GMPS13, LMPS15] that the most compiling issue when dealing with reasoners is *the performance variability phenomena*. This issue have give raise to various new challenges to both the ontology and reasoner respective communities. We will be discuss some of them in what follows.

[14]The second version of OWL, OWL 2 [Gro09], is a W3C recommendation since 2009

Challenges

Despite the remarkable progress in optimizing reasoning algorithms, unpredictable behaviours of reasoner systems is often observed in practice, particularly when dealing with real world ontologies. We have delineated two main aspects that would depict the phenomena under examination: (*i*) the variability of reasoners runtime across the ontologies; (*ii*) the sparsity of the computed inferences by different reasoners over the same input ontology.

On the one hand, Gonçalves et al. [GPS12] have outlined the reasoner runtime variability by three particular situations that a user might face when attempting to reason about an ontology: (*i*) For one test case ontology, switching the reasoner can degrade reasoning time from seconds to none termination; (*ii*) Ontologies with the same size and expressivity would spend wildly different ranges of computational time on the same reasoner; (*iii*) An insignificant change to an OWL ontology would increase or probably decrease reasoning time on one reasoner. As far as one of these situations happens, often no feedback is returned to the user. Commonly, the latter one will keep shifting reasoners until finding the suitable one. Others would try to adjust their ontology hoping for an improvement, but running the risk of making the matters worse. Lee et al. [LMPS15] explained the variability of reasoner runtime by the fact that modern reasoning systems have complex design architecture and tend to implement a wide range of optimisation techniques that might interact and affect each other while processing some particular ontologies. However, it is still not well understood which of the ontology characteristics are badly affecting the reasoner optimisation tricks. By consequence, an ontology, which is a performance bottleneck for some particular reasoner, is hardly predictable a priori.

Furthermore, Gardiner et al. [GTH06] and more recently Lee et al. [LMPS15] have shed light on the *disparity* of the reasoning results derived by distinct reasoners from the same input ontology. This disagreement across the reasoners over inferences or query answers would make it hard to provide interoperability in the Semantic Web. Therefore, an empirical correctness checking method was added to the ORE evaluation framework to examine the various reasoner results. The method described in [GBJR+13] is a simple, majority voting procedure which has been put in place to resolve disagreements and decide which inferences are likely to be the most consistent ones. By referring to this method, it has been observed that there was no single reasoner, which correctly processed and outperformed on all given inputs. Even the fastest reasoner has failed to derive accurate results for some ontologies, while others less performing systems have succeed to correctly process them. Actually, while the formal basis of the core reasoner procedures are well understood, many optimizations and most of the engineering details are less explained and often reviewed only by the reasoner developer. Hence, it seems that even the reliability of modern reasoner implementations is still a matter of investigation.

Through this discussion, we would assert that the most desired qualities, i.e. result correctness and time efficiency, are not empirically guaranteed by all the reasoners and for every ontology. A reasoner could be optimized for some, but not all ontologies. These observations pinpoints the hardness of understanding reasoner empirical behaviours even for experienced and skilled reasoner designers. As previously argued by Wand and Parsia [WP07], the actual problem with DL reasoning is the lack of theory and tool support helping both ontology novice and expert users to analyse the performances of the reasoners and figure out which particular aspects in the ontology that are reasoner performances degrading factors. This lack of knowledge are making the experts' pre-established decision rules about the appropriateness of a reasoner for a particular ontology less effective in practice, except for trivial cases. One further consequence is the growing need to characterize the hard to manage ontologies and gaining insights about their particular attributes.

Based of these findings, in our thesis, we will be developing and further investigating two basic research questions, that we considerer to be the most relevant to an end user:

1. Which ontology features are likely to have impact on reasoner performances?
2. How feasible is it to reason over an ontology written in particular DL, using the best available reasoners?

Our second research question could further be detailed. Indeed, a couple of finer grained interrogations emerges from it: first, we would wonder whether is it possible to a priori predict the performances of any reasoner over any ontology, and then, how could we assist the end users in the automatic selection of the most *"appropriate"* reasoners for their ontologies?

Goals and Contributions

The main goal of this thesis is to bring solutions to cope with the reasoner performances variability phenomena and hence, advance the state of the art in the empirical analysis of DL reasoners. We believe that providing guidance over the empirical behaviours of reasoners could further improve reasoner optimizations and enhance ontology design by avoiding reasoner performances degrading factors.

To provide solutions to our first research question, we carried out an investigation about existing methods and tools that intended to identify potential correlation between reasoner empirical behaviour and particular ontological features. We tried to give users an overall view of the state of art in this field. The pioneering works were analysed and then broke down into categories. Our investigations have lead us to believe that the design complexity of an ontology is the result of complicated combination of various dimensional characteristics and can not be measured directly using a single metric. Therefore, we proposed a set of ontology features covering a broad range of structural and syntactic ontology characteristics. We claim that these features would be key indicators of the ontology hardness level against reasoning tasks.

Our second research question addresses two issues: (i) prediction of reasoner empirical behaviours and (ii) automatic selection of adequate reasoner given an input ontology. Reasoner performance prediction was also investigated by Gonçalves et al. [GPS12]. They suggested that one way to explain why a reasoner performs poorly on some ontology is to predict whether the ontology contains some computationally expensive components. Their main claim is that whenever the time required to classify an ontology by a particular reasoner is not linearly correlated to the size of its subsets, then the ontology is said to be performance-heterogeneous for this reasoner. In this case, it is likely that some ontology sub-parts, i.e. modules, are reasoner performance *"hotspots"*. However, their experiments have revealed that there is no precise co-relation between the reasoning time of a *hotspot* alone, and the reduction in reasoning time when such *hotspot* is removed. They affirmed that more investigations should be made about possible interactions between the hotspots and other ontology features. Another steam of works, mainly described in [KLK12, SSB14, KLK14], used supervised machine learning (SML) techniques [Kot07] aiming at predicting the computational time of a reasoner over any ontology. Their predictive models take advantage from a large set of pre-computed ontological metrics. The rational behind this choice is to be able to automatically learn future reasoner behaviours based on what was experienced in their past running. Reasoner performance prediction approaches are still in their beginning but showed very promising results in different area of applications. For instance, the respective authors of [KLK14, SSB14] have deployed their reasoner runtime predictive models to identify the ontology *hotspots*. Furthermore, we should highlight that the idea of learning to predict an algorithm runtime from empirical data is not new. Over almost a decade of intensive research, Brown et al. [LBHHX14] have demonstrated that the runtime of solvers over NP-Complete SAT problems can accurately be predicted based on easy to compute features and by employing SML techniques. Hutter et al. [HXHLB14] have carried out a large scale survey summarizing all of the pioneering works attempting to learn algorithm *empirical performance models* (EPMs). Motivated enough, we find that machine learning based approaches are of great interest to meet the demand of tracking down the reasoner performances variability phenomena. As a matter of fact, the learning methods are generic ones, since they could handle any reasoner or any ontology, and have a broad scope of applications. Nevertheless, the existing solutions of reasoner prediction are still limited ones, since all of them assess only of the reasoner runtime. Yet, in our opinion, a reasoner that quickly but incorrectly processes an ontology is of little use. In this thesis, we propose to advance the predictive modelling of reasoner empirical behaviours by investigating further reasoner output qualities and by identifying those ontologies which are likely to damage these qualities. More specifically, we will try to predict the correctness of reasoners as well as their efficiency. To achieve this purpose, we will establish a detailed specification of a generic learning framework capable to accomplish several predictive modelling tasks about ontology reasoning. The framework uses the proposed ontology feature suite and a large set of supervised machine learning algorithms coupled with multiple feature selection techniques. By employing this framework, we will first train a model able to predict whether an ontology is *empirically hard* with respect to a group of reasoners. Thus, performance unmanageable ontologies could be identified. Seeking more finer grained analysis, we will try to portray the robustness of individual reasoner by combining conditional predictive models of its correctness and its efficiency. This particular association materialize our concept of a reasoner *predictive robustness profile*. We will show that based only on simple easy-to-compute ontology features, highly accurate models of both the *hardness* of ontologies and the *robustness* of reasoners could be learned by an intelligent exploration of the empirical data. Finally, we will provide ways to interpret the different trained predictive models in order to uncover new, simple relationships between ontology characteristics and reasoner qualities, and thereby catalyse our understanding of the key ontology features likely to downgrade reasoner empirical robustness. Our assumption is that the more effective are the features in describing the design complexity of the ontologies, the more accurate will be the trained predictive models.

Our next interrogation is linked to the long-standing problem of algorithm selection, discussed in the first place by John Rice [Ric76] in the mid-70s. This issue has been at the crossroads of many problem fields, such as Machine Learning (ML) and Artificial Intelligence (AI). A common trait among the proposed solutions is that they employ supervised machine learning techniques, regression or ML classification[15], to build an efficient predictor of the performances of candidate algorithms for each problem instance. Hence, the most adequate one could be identified. Meta-learning [BGCSV08] and algorithm portfolio [GS01] construction are the main fields where the Rice's model of algorithm selection was put into practice. While the meta-learning approaches basically aim to recommend the best machine learning algorithm given an input dataset, the final goal of portfolio approaches is to take advantage of the complementarity of the algorithms by combining them in order to obtain improved performances in the average case. The idea of building reasoner portfolio have recently seduced Kang et al. [KKL15]. The latter ones have employed their previously trained reasoner runtime predictive models, to build R_2O_2 a kind of *meta-reasoner*, which combines existing reasoners and tries to determine the most efficient one for a given ontology. Their results were remarkable, claiming that R_2O_2 could outperform the most efficient state of art reasoner. However, no rigorous evaluation under

[15]It is important to notice that "DL classification" stands for a reasoning task aiming at inferring the subsumption relations between the classes and the properties in given ontology, however the "ML classification" denotes the process of training a predictive model from a labelled dataset.

no idiosyncratic conditions have been conducted to confirm the Kang's et al. assertions. In fact, the time overheard due to the prediction steps was not discussed. In this thesis, we chose to address the problem of ontology reasoner selection from a different perspective. We believe that to have reasoners selected as being best suited from an input ontology is a kind of recommendation that would be made for users seeking for guidance. The provided advice would be in its basic form, a ranking of reasoners, in a most preferred order. As previously highlighted by [PdSL11], providing a ranking of candidate algorithms is a more informative and flexible solution, than suggesting one single best algorithm. For instance, a user may decide to continue using her/his favourite reasoner, if its performance is slightly below the topmost one in the ranking. To achieve this purpose, we formulated a novel reasoner ranking framework, which take into account information regarding the robustness of the candidate reasoners, more specifically their correctness and efficiency level regarding an input ontology. We investigated two distinct ranking approaches: a first ranking method based on single label prediction; and then, a multi-label based method for ranking prediction. We experimentally and analytically proved the worthiness of each proposed approach and put them into comparison. To the best of our knowledge, we are the first to employ more than one criterion to rank the reasoners and we believe it is one important step in the direction of multi-criteria ranking of ontology reasoners.

Beyond reasoner selection, we proposed another kind of advising for end-users seeking to gain more insights about their ontologies. Indeed, whenever an ontology is recognized as a hard to manage one, a user will look for assistance to gain insights about the potential causes of this hardness. In this case, we suggest to provide her/him the possibility to extract those ontology sub-parts that are probably the most computationally demanding ones. As described above, these sub-parts are commonly called the ontology *hotspots* [GPS12]. Hotspots are performance bottlenecks for reasoning. An efficient identification of these components is highly desirable since they represent re-factoring opportunities for ontology designers. We revisited both the Gonçalves et al. [GPS12] and the Kang et al. [KLK14] respective approaches of hotspot extraction. We suggested a new method which relies on atomic graph representation of the ontology axioms. Our extraction takes advantage of the trained ontology overall hardness model and is guided by those ontology features recognized as key ones. Our first experimental results are encouraging ones, testifying the potential of this method.

Thesis Outline

To summarize, the main purpose of this thesis is to investigate means to cope with the reasoner performances variability phenomena and ways to gain insights into the ontology primary features likely to be reasoner performances degrading factors. Our final goal is to provide simple, efficiently computable guidelines to novice and expert users about, on the one hand the empirical *hardness* of OWL ontologies, and on the other hand, the empirical *robustness* of modern DL reasoners. The guidelines are basically a set of predictions about the expected behaviours of an individual reasoner or a ranking of group of reasoners given an input ontology. We believe that delivering such indications for novel user with little expertise in sophisticated reasoning algorithms, would be a worthwhile solution saving her/him valuable time in research and analysis. But it is also a worthwhile starting point for experts to conduct further in depth investigations about their ontologies or reasoners. To achieve this purpose, we opted for the supervised learning as the kernel theory to guide the design of our solutions.

This dissertation is structured along the way the experimental study was carried out. In more details, the subsequent chapters focus on the following subjects:

Chapter 1 lays the groundwork in terms of employed terminology and core background notions used throughout the thesis. Our research questions are at the crossroads of two fields: the Semantic Web [BLHL01] and the Machine Learning [Sam59]. That is why, we will be recalling some of their basic notions. In particular, we focus on the three key notions: the ontology, the reasoner and the supervised learning.

Chapter 2 scrutinizes our related works. We will first discuss the complexity levels of modelling OWL ontologies and ways to characterize its design. Then, we will focus our study on the reasoner performances variability phenomena. We will be putting into comparison methods and tools which tried to detect and analyse it. The final section of this chapter will be dedicated to the advances in addressing the algorithm selection problem. We will examine known approaches in meta-learning and algorithm portfolio construction.

Chapter 3 gives a detailed description of our set of proposed ontology features. The latter ones are arranged in four basic categories and cover a wide range structural and syntactic ontology characteristics. We employed our features to describe OWL ontologies selected from a well known corpus. Finally, we conducted a comparison about the computational cost of measuring our features in order to identify the most efficiently computable ones.

Chapter 4 describes our reasoner learning methodology which relies on our rich set of advanced ontology features. Owing to the benefits of this proposal, we modelled the robustness profile of a reasoner, in terms of predictive models of its correctness and efficiency. To demonstrate the worthiness of our proposals, we conducted a rigorous investigation about 6 well known reasoners and over 2500 ontologies. We put into comparison various supervised learning algorithms and feature selection techniques in order to train best reasoner predictive models. Our trained reasoner predictive models showed to be highly accurate outperforming state-of-art solutions.

Chapter 5 describes the two domain applications were the established predictive models could be integrated. Both of them are part of a generic system that we called ADSOR standing for "Advising System Over Ontology Reasoners". The main purpose of this system is to provide guidance for end-users for an easy selection of the most appropriate reasoner for their ontology based applications. Furthermore, the system help users to gain insights about the empirical hardness of their ontologies. It suggests to predict the overall hardness of an input ontology and more interestingly, it offers the possibility to extract its most computationally challenging sub-parts. The various approaches that provide the basis for the core design of this system are detailed in this chapter. The worthiness of each approach was experimentally proved and in depth analysed.

Finally, we concluded this dissertation by summarizing our main contributions and relevant results and by pointing out some open questions and future directions.

CHAPTER 1

Theoretical Background

Contents

1.1 Introduction

This chapter lays the groundwork in terms of employed terminology and core background notions used throughout the thesis. Our research questions are at the crossroads of two fields: the Semantic Web [BLHL01] and the Machine Learning [Sam59]. That is why, we will be recalling some of their basic notions. In particular, we focus on the three key notions of our thesis: the ontology, the reasoner and the supervized learning. The term "ontology", in computer science, is used to refer to a representation of knowledge as a set of terms, and relations between these terms. This thesis is focused on description logic based ontologies written in the Web Ontology Language (OWL), which is discussed in Section 1.3 along its applications and profiles. Then in Section 1.4, we formally define logical inference problems that correspond to standard reasoning services provided by DL systems. Furthermore, we will conduct a meta-data study about modern DL reasoners and put into comparison their relevant functional features. Finally in Section 1.5, we will briefly recap some basic notions of machine learning (ML) field of research while putting more attention on supervized learning (SML) concepts. We will also give an overview of the most common used SML techniques for data preprocessing and predictive model training. These techniques will be considered throughout this thesis. Besides, we depicted the relevant assessment metrics deployed in SML validation process.

1.2 Semantic Web

Today's World Wide Web gives new opportunities for discipline called Knowledge Representation (KR). Web resources needs to be better adopted for automatic processing by computer agents. That is possible by more systematic and formal description mechanisms used for defining contents of the pages. One of the most powerful feature of the World Wide Web is the possibility of interconnections between knowledge sources. Interconnections are provided by linking mechanisms on pages. In other words, instead of copying information from external sources created by other people to our page, we just provide links to such sources. More in valuable, the Uniform Resource Identifiers (URIs), which are the naming scheme for web resources, allows KR systems to improve linking between semantic documents by avoiding the ambiguities of natural language. Henceforth, new possibilities and challenges were opened for knowledge representation assumptions [VLP08], which give rise to the Semantic Web.

The Semantic Web can be treated as an extension to the current web and the successor of Web 2.0. In recent Web 2.0 architecture, there is a lot of unstructured data. Information is represented in various ways, incompatible witch each other. What Semantic Web offers is defined meaning of information, machine-readable description of contents (meta-data), simplification of information exchange, classification and inference mechanisms, data processing and integration in machine-automated manner.

"The Semantic Web is an extension of the current Web in which information is given well-defined meaning, better enabling computers and people to work in cooperation."

Tim Berners-Lee [BLHL01]

"The goal of Semantic Web research is to transform the Web from a linked document repository into a distributed knowledge base and application platform, thus allowing the vast range of available information and services to be more effectively exploited."

Ian Horrocks [Hor07]

As it was said, the Semantic Web is a web that is able to provide information about things in the way computers can understand. The human-understandable sentences which describe things, are going to be intelligently processed in the Semantic Web. Statements appropriate for dedicated language are built using syntax rules, which are defined by this language. In Semantic Web field statements are built using rules provided by specific KR languages and such statements are semantic aware. Semantic analysis of such statements allows to convert them to knowledge base queries. To search and access the Semantic Web content, tools called *Semantic Web Agents* or *Semantic Web Services* are required. It is because searching process in Semantic Web will be much different than today's searching in free text - complicated math algorithms are going to be provided to bring answers to these emerging needs.

"The Semantic Web provides a common framework that allows data to be shared and reused across application, enterprise, and community boundaries."

World Wide Web Consortium [W3C]

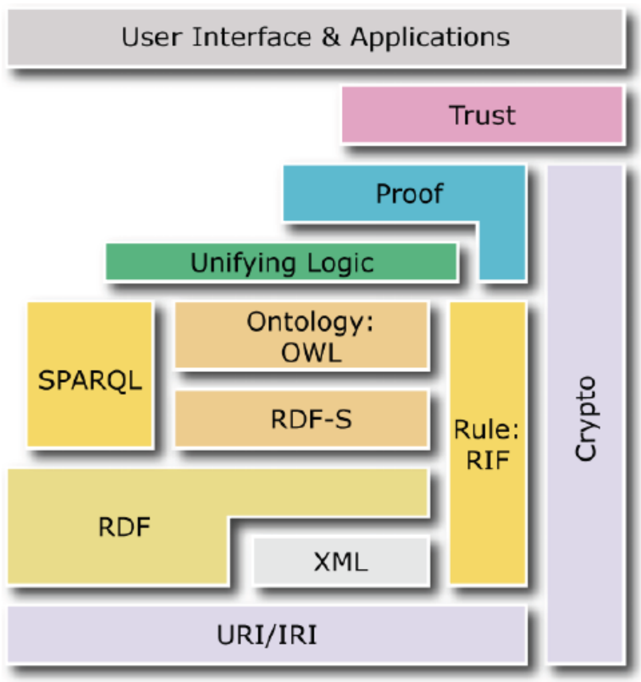


Figure 1.1: Semantic Web Stack [W3C]

The Semantic Web layered architecture recommended by the W3C (Figure 1.1) covers such technological aspects like application interfaces, security, rules and logic (SWRL, RIF), ontologies (RDFS, OWL), meta-data (RDF), serialization (XML) and resource identification (URI). This layers are briefly described below:

URI and Unicode : a uniform system of identifiers (URIs) is used to locate resources, or any other thing on the Web. Otherwise, the URI *uniquely* identifies the Web resources. Unicode is the chosen standard for computer character representation.

eXtensible Markup Language (XML) is a data format. XML plays an important role in exchanging different types of data on the Web and in electronic publishing. It is meant to be a the standard serialization format for the Semantic Web, thus information could be parsed appropriately when they are transferred between machines.

RDF : Resource Description Framework is a data model. It allows for representing semantics of information about resources on the Web in a machine-accessible way. RDF employs the URIs to identify Web resources and to describe the relationships between them, using a labelled directed graph model.

RDF-S : the Resource Description Framework Schema, is a lightweight, easy to use language for defining RDF vocabularies (i.e. ontology). RDF-S is expressed in RDF and allows to reduce the ambiguity of this latter one by defining some object-oriented concepts such as classes and properties.

Using RDF-S some basic inference rules could be applied.

Ontology and OWL : an ontology is the center-piece of the Semantic Web. It provide a uniform vocabulary with a precise meaning to describe the semantics of the data on the Web and hence, enable exchange and interoperability between the different Semantic Web agents, i.e. humans and/or machines. The Ontology Web Language (OWL) is another data modelling language used to describe RDF vocabularies. It extends RDFS and have allow for more expressive description of knowledge. OWL is based on description logics (DL) and thus allows automatic deduction and inference over the ontology knowledge.

SPARQL : for querying RDF data as well as RDFS and OWL ontologies with knowledge bases, a Simple Protocol and RDF Query Language (SPARQL) is designed. It is SQL-like language, but uses RDF triples.

Logic and Proof : all the above data modelling languages are designed to provide structured knowledge about the Web resources, which follows some logic. Hence, the consistency of this knowledge could be checked and inferred using deduction systems, called *reasoners*.

Trust : the final layer of the Semantic Web which concerns the trustworthiness of the information on the Web in order to provide an assurance of its quality.

In the remainder of this chapter, we will focus our attention on two particular layers: the ontology from a knowledge representation (KR) perspective and the logic/proof layer to get more insights about the description logic and the automatic reasoning.

1.3 Ontology

T. R. Gruber [Gru93] defined an ontology as "formal, explicit specification of a shared conceptualization". An ontology delineate a common vocabulary that users can share within a domain of knowledge. Ontologies have become increasingly important with the global move towards the Semantic Web. They have been used as conceptual models, for data integration, or to directly represent information in a variety of domain areas. Ontologies are made available on the web to a multitude of machines, to promote the exchange and sharing of knowledge. On these basis, many reasons would further motivate the development of ontologies, for instance:

- to share common understanding of the structure of information among people or software agents;
- to enable reuse of domain knowledge;
- to make domain assumptions explicit;
- to separate domain knowledge from the operational knowledge;

- to analyse domain knowledge.

The ontology potential of providing domains with an unambiguous vocabulary has attracted the interest of many domain experts from diverse areas, such as medicine, bio-informatics and geography. Strong efforts were put in modelling and maintaining ontologies. A notable example of this process is the National Cancer Institute thesaurus[1]. NCI-t has been updated monthly since 2003, and many versions of the NCI-t ontology are freely downloadable. Other important medical ontologies could also be mentioned: Snomed Ct[2], the Open Galen project[3] and the FMA ontology[4].

Ontologies are typically encoded in a Knowledge Representation language. A major class of ontology languages is based on the family of Description logics (DLs) [BCM+10], which generally are decidable fragments of the First Order Logic (FOL). Hence, an ontology may be viewed as a set of FOL formulae, called *axioms*, and it is therefore a logical theory. A notable example of an ontology language based on DLs is the Web Ontology Language OWL [HPSV03], a World Wide Web Consortium Standard which has been used both in academic and industrial environments since its first version became a W3C recommendation in 2004. In this thesis, we put our focus on studying the ontology expressivity at the KR language level and more in particular, the expressivity of the OWL language. In the remainder of this section, we first recall the basics of the Description Logics and then, we remind the main concepts of the OWL language.

1.3.1 Description Logics Foundation

Description Logics [BCM+10] are a well-investigated family of logic based knowledge representation formalisms, which can be used to represent the conceptual knowledge of an application domain in a structured and formally well-understood way. They are employed in various application domains, such as natural language processing, configuration, and databases, but their most notable success so far is the adoption of the DL-based language, OWL, as a standard ontology language for the Semantic Web [HPSV03].

Roughly speaking, Description Logics (DLs) are subsets of the first-order predication logic (FOL) [BCM+10]. A DL ontology $\mathcal{O}$ is composed of a set of asserted axioms, analogous to FOL formulae, describing relationships between terms in a domain of interest. These terms are basically concept, role, and individual names, organized respectively in three sets N_C , N_R, and N_I. The union of these sets, that is the set of all terms mentioned in $\mathcal{O}$, is called the signature of $\mathcal{O}$, and denoted $\widetilde{\mathcal{O}}$. We use the

[1]https://wiki.nci.nih.gov/display/VKC/NCI+Thesaurus+Terminology
[2]More details about Snomed Ct is available at http://www.ihtsdo.org/
[3]More details about Galen is available at http://www.opengalen.org/
[4]More details about FMA is available at http://sig.biostr.washington.edu/projects/fm/AboutFM.html

term *"signature"* and associated notation for any set of terms mentioned in an ontology or an element thereof; for example, the signature of an axiom α is denoted $\widetilde{\alpha}$, and of a concept C denoted by $\widetilde{C}$. The term "*concept*" is used for any (possibly complex) concept, while "atomic concept" is used to refer to a concept name, and similarly "atomic role" stands for a role name. Commonly, A and B are used to design named concepts, C and D are complex concept descriptions [BCM+03], R and S are role names or descriptions and a, b and x are names for individuals. Complex concept descriptions are built based on concept and role constructors, as well as names from N_C , N_R or N_I.

In DL, an ontology is basically defined as a knowledge base $\mathcal{K} = \langle \mathcal{T}, \mathcal{R}, \mathcal{A} \rangle$, where $\mathcal{T}$ denotes *TBox*, which comprises *terminological* axioms describing concepts, $\mathcal{R}$ denotes *RBox* for axioms describing roles, and $\mathcal{A}$ stands for *ABox*, which is the assertional part of knowledge base describing individuals. The semantics of ontology is defined using *interpretation* $\mathcal{I}$ that consists of a tuple $\mathcal{I} = (\triangle^{\mathcal{I}}, \cdot^{\mathcal{I}})$, where $\triangle^{I}$ is a non-empty set of instances (the domain of interpretation) and $\cdot^{\mathcal{I}}$ the interpretation *function*, which assigns a set $A^{I} \subseteq \triangle^{\mathcal{I}}$ to every atomic concept $A \in N_C$, a binary relation $R^{\mathcal{I}} \subseteq \triangle^{\mathcal{I}} \times \triangle^{\mathcal{I}}$ to every atomic role $R \in N_R$ and finally to each instance $a \in N_I$ an element of $\triangle^{\mathcal{I}}$. Table 1.3 summarizes the basic forms of axioms and their syntax and semantics. Different families of Description Logic provide

	Name	**Syntax**	**Semantics**
TBox	$\top$	Top	$\triangle^{\mathcal{I}}$
	$\bot$	Bottom	$\emptyset$
	Concept Definition	$A \equiv C$	$A^{I} = C^{I}$
	Concept Inclusion	$C \sqsubseteq D$	$A^{I} \sqsubseteq C^{I}$
RBox	Role equivalence	$R \equiv R$	$R^{I} = S^{I}$
	Role hierarchy	$R \sqsubseteq S$	$R^{I} \sqsubseteq S^{I}$
	General role inclusion	$R_1 \circ \ldots \circ R_2 \sqsubseteq S$	$R_1^{I} \circ \ldots \circ R_2^{I} \sqsubseteq S^{I}$ where $\circ$ is a binary composition of relation
	Role functionality, transitivity, symmetry, asymmetry reflexivity, irreflexivity	Func(R), Tra(R) Sym(R), Asy(R) Ref(R), Irr(R)	R^{I} is functional, transitive symmetric, asymmetric, reflexive and irreflexive
ABox	class assertion	$C(a)$	$a^{I} \in C^{I}$
	role assertion	$R(a, b)$	$< a^{I}, b^{I} > \in R^{I}$
	Negative role assertion	$\neg R(a, b)$	$< a^{I}, b^{I} > \notin R^{I}$

Table 1.1: Basic axioms of TBox, RBox and ABox

different sets of concept and property constructors, besides of axiom types. One of the simplest DLs is known as $\mathcal{AL}$ (Attributive Language). This DL supports concept conjunction ($C \sqcap D$), universal quantification ($\forall R.C$), limited existential quantification ($\exists R.\top$) and atomic negation ($\neg A$). More expressive DLs can be obtained from $\mathcal{AL}$ by adding further constructors. Each constructor is given a specific letter which is used to derive a name for any particular DL. For instance, adding full negation

($\neg C$) to $\mathcal{AL}$ produces the DL $\mathcal{ALC}$, which also contains concept disjunction ($C \sqcup D$) and full existential quantification ($\exists R.C$). However, extending the logic $\mathcal{ALC}$ with transitive roles becomes the logic $\mathcal{S}$. Then, $\mathcal{SH}$ extends $\mathcal{S}$ with role hierarchies $\mathcal{H}$. Adding nominals $\mathcal{O}$, inverse properties $\mathcal{I}$ and number restrictions $\mathcal{N}$ to $\mathcal{SH}$ produces $\mathcal{SHOIN}$. $\mathcal{SHOIQ}$ extends $\mathcal{SHOIN}$ with qualified cardinality $\mathcal{Q}$. Finally, $\mathcal{SROIQ}$ extends $\mathcal{SHOIQ}$ by adding reflexive, irreflexive, complex chains and disjoint roles definitions denoted by $\mathcal{R}$. More DL families could be obtained by further combining the concept and role constructors.

An interpretation $\mathcal{I}$ satisfies a $\mathcal{SROIQ}$ axiom α, denoted $\mathcal{I} \models \alpha$, from those axiom types mentioned above as follows:

$$\begin{array}{lll}
\mathcal{I} \models & C \sqsubseteq D & \text{if } C^{\mathcal{I}} \subseteq D^{\mathcal{I}} \\
\mathcal{I} \models & r \sqsubseteq s & \text{if } r^{\mathcal{I}} \subseteq s^{\mathcal{I}} \\
\mathcal{I} \models & r \circ s \sqsubseteq t & \text{if } r^{\mathcal{I}} \circ s^{\mathcal{I}} \subseteq t^{\mathcal{I}} \\
\mathcal{I} \models & C(a) & \text{if } a^{\mathcal{I}} \in C^{\mathcal{I}} \\
\mathcal{I} \models & r(a,b) & \text{if } \langle a^{\mathcal{I}}, b^{\mathcal{I}} \rangle \in r^{\mathcal{I}}
\end{array}$$

If an interpretation $\mathcal{I}$ satisfies every axiom in an ontology $\mathcal{O}$, $\mathcal{I}$ is called a model of $\mathcal{O}$, denoted $\mathcal{I} \models \mathcal{O}$. The restriction of an interpretation $\mathcal{I}$ to a signature Σ is denoted $\mathcal{I} \mid_{\Sigma}$. Two interpretations $\mathcal{I}$ and $\mathcal{J}$ coincide on a signature Σ (denoted $\mathcal{I} \mid_{\Sigma} = \mathcal{J} \mid_{\Sigma}$) if $\triangle^{I} = \triangle^{J}$ and $t^{\mathcal{I}} = t^{\mathcal{J}}$ for each $t \in \Sigma$.

1.3.2 Ontology Web Language (OWL)

A number of formalisms have been proposed for representing Semantic Web meta-data, in particular RDF[5], RDFS[6] and OWL (previously known as DAML+OIL) [HPSV03]. The Resource Description Framework (RDF) is a framework developed by the W3C (World Wide Web Consortium) for representing information in the World Wide Web. RDF inherits XML syntax and exploits URI to identify resources. RDF Schema (RDFS) is used to specify the vocabularies in RDF. Whereas RDF merely provides a syntax for representing assertions like "*Book A is authored by person B*", schema or ontology languages such as RDFS and OWL allow one to state properties of the terms used in such assertions, e.g. "*no person can be a text*" and thus building upon descriptions of resources and their schemas as detailed in the architectural road map for the Semantic Web (Figure 1.1). In this thesis, we will focus our study on **OWL** as it is the latest recommended language for the ontology authoring.

[5]https://www.w3.org/TR/rdf11-mt/
[6]https://www.w3.org/TR/rdf-schema/

1.3.2.1 OWL profiles

The **O**ntology **W**eb **L**anguage, OWL, is one of the most widely used ontology languages. It has excellent tool support in terms of editors and reasoning engines, thanks to its formally defined meaning. OWL lays on top of RDF and RDFS and comes with a larger vocabulary and stronger syntax. In OWL there are two kinds of axioms allowed: logical axioms and non-logical (annotation) axioms. The latter are analogous, with respect to reasoning, to comments in programming languages. The former, i.e. logical axioms, are statements that are asserted to be true under well defined semantics, which in turn allow inferring new implicit axioms, so-called *entailments*, that can be deduced as true ones based on the explicitly asserted axioms. Primarily developed for the Semantic Web, OWL became a W3C recommendation in 2004, at the time, it was based on the description logic $\mathcal{SHOIN}$(DL) [BCM+10] and called OWL 1.0. The first iteration of OWL specified the three "*species*": OWL Lite, OWL DL and OWL Full. The second iteration of OWL, the OWL 2 [Gro09], became a W3C recommendation in 2009 and it is based on the expressinve DL, $\mathcal{SROIQ}(DL)$ [HKS06]. It also incorporates several species, now called "*profiles*". The latter ones were described in details in [Krö12]. They are namely: OWL EL, OWL RL and OWL QL, while the full profile is called OWL DL. The profiles are lightweight sub-languages of OWL 2, restricting the available modelling features in order to simplify reasoning. This has made them very attractive for ontology practitioners. Commonly, OWL EL is used to model huge biomedical ontologies, OWL RL became the preferred approach for reasoning with Web data, and OWL QL provides database applications with an ontological data access layer. Worth to note that any OWL ontology can be written (and serialised) in several syntaxes: RDF/XML (the primary exchange syntax), OWL/XML, Functional, Manchester, and Turtle. In the remainder of this thesis, we will study in particular the OWL 2 ontologies and hence, by OWL we will mean OWL2.

1.3.2.2 OWL language components

Roughly speaking, a concept in DL is referred to as a class in OWL. A role in DL is a property in OWL, which could be an `Object Property` (properties for which the value is an individual) or a `Data Property` (properties for which the value is a data literal). Axioms and individuals have the same meaning in DL and OWL. Owe to snugness connection between OWL and DLs, in this thesis, we will make no distinction between ontologies (in OWL) and knowledge bases (in DL). OWL classes can be created from basic classes or properties by variety of constructors. The constructors supported by OWL, are shown in Table 1.2. Table 1.3 shows some types of OWL axioms belonging to different categories, where A and B can be named concepts, C and D are complex concept descriptions, R and S are role names or descriptions, and a, b and x are names for individuals.

OWL Constructor	**DL Syntax**	**Example**
intersectionOf	$C_1 \sqcap \cdots \sqcap C_n$	$Human \sqcap Male$
unionOf	$C_1 \sqcup \cdots \sqcup C_n$	$Doctor \sqcup Lawyer$
complementOf	$\neg C$	$\neg Male$
oneOf	$\{x_1 \cdots x_2\}$	$\{john, mary\}$
allValuesFrom	$\forall P.C$	$\forall hasChild.Doctor$
someValuesFrom	$\exists r.C$	$\exists hasChild.Lawyer$
hasValue	$\exists r.\{x\}$	$\exists citizensOf.\{USA\}$
minCardinality	$(\geq nr)$	$(\geq 2\ hasChild)$
maxCardinality	$(\leq nr)$	$(\leq 2\ hasChild)$
inverseOf	r^-	$hasChild^-\ (= childOf)$

Table 1.2: Basic OWL constructors derived from [VLP08].

	OWL Axiom	**DL Syntax**	**Example**
TBox	equivalentClass	$A \equiv C$	Man ≡ Human ⊓ Male
	subClassOf	$C \sqsubseteq D$	Human ⊑ Animal ⊓ Biped
	disjointWith	$C \equiv \neg D$	$Male \equiv \neg Female$
RBox	equivalentProperty	$R \equiv S$	cost ≡ price
	subPropertyOf	$R \sqsubseteq S$	hasDaughter ⊑ hasChild
	inverseOf	$R \equiv S^-$	hasChild ≡ hasParent$^-$
	transitiveProperty	$R^+ \sqsubseteq R$	ancestor$^+$ ⊑ ancestor
	functionalProperty	$\top \sqsubseteq \leq 1R$	⊤ ⊑ ≤ 1 hasMother
	inverseFunctionalProperty	$\top \sqsubseteq \leq 1R^-$	⊤ ⊑ ≤ 1 isMotherOf$^-$
ABox	Concept assertion	$C(a)$	Human(Peter)
	Role assertion	$R(a, b)$	hasMother(Peter, Mary)
	sameIndividualAs	$\{x_1\} \equiv \{x_2\}$	President-Kennedy ≡ J-F-K
	differentFrom	$\{x_1\} \equiv \neg\{x_2\}$	Peter ≡ ¬ John

Table 1.3: Basic OWL axioms from TBox, RBox and ABox, the knowledge base sub-parts derived from [VLP08]

In the other hand, OWL has different types of "*roles*", as well as further features which are not originally found in DLs. These are mainly:

Annotations As of OWL 2, annotations can be added to terms (referred to as *entities* in the OWL specification), axioms, and the ontology itself by means of so-called annotation properties. For instance, the annotation property `rdfs:label` is commonly applied to terms, and used to change the way OWL ontology interfaces render term names in an ontology: according to their label value or URI.

Imports In OWL it is possible to import external ontologies (for reuse, for example) via the`owl:imports` construct. Systems like reasoners that take ontologies as input will typically take into account the entire imports closure when performing reasoning tasks.

Concrete domains OWL draws on advances in augmenting DLs with concrete domains to allow describing concrete qualities of objects, such as weight, height, temperature, etc. To make use of concrete domains, OWL features so-called data properties, which assign to a class or individual a concrete value of some (data) type. The accepted "*datatypes*", as they are referred to in the OWL specification, are taken from the set of XML Schema Datatypes[7], the RDF specification[8] and the specification of plain literals[9].

1.3.3 Ontology Engineering Tools

In this subsection, we overview of some the semantic web technologies that will be considered throughout this thesis. In particular, we present the semantic APIs, ontology query languages and ontology editors.

Semantic APIs (Jena and OWL API) **Jena**[10] [McB02] is an ontology API to manage OWL ontologies and RDF data in Java applications. Jena is appropriate to manage OWL 1 Full ontologies, but support for OWL 2 is not available yet. However, it is much more used for the serialization of RDF triples and the manipulation of RDF graphs. Jena can interact with semantic reasoners to discover implicit knowledge. The latest versions of Jena are split into two packages, namely jena-fuseki (with the Jena SPARQL server), and apache-jena (with APIs, SPARQL engine, RDF database, and other tools). **OWL API**[11] [HB11] is another API particularly designed to manage the ontologies written

[7]http://www.w3.org/TR/xmlschema11-2/
[8]http://www.w3.org/TR/2004/REC-rdf-concepts-20040210/
[9]http://www.w3.org/TR/rdf-plain-literal-1/
[10]http://jena.apache.org
[11]http://owlapi.sourceforge.net

with the OWL language, and to provide a common interface to interact with DL reasoners. It can be considered as the current standard in ontology editing, as the most recent versions of most of the semantics tools and reasoners use the OWL API to load and process OWL 1 & 2 ontologies. The OWL API is able to process each of the OWL syntaxes defined in the W3C specification (functional, RDF/XML, OWL/XML, Manchester, and Turtle) and to identify the OWL 2 profiles (OWL 2 DL, OWL 2 EL, OWL 2 QL, and OWL 2 RL). The OWL API is less appropriate for the management of RDF ontologies.

Query Languages (QLs) for Semantic Web ontologies as described by [BBFS09] can be classified under two categories: RDF-based QLs and OWL-based QLs. RDF-based query QLs, such as RDQL3, SeRQL4 and the W3C recommendation SPARQL[12], are based on the notion of RDF triple patterns and their semantics is based on matching triples with RDF graphs. SPARQL is the most common used query language for RDF. It provide the possibility to express queries across diverse data sources, whether the data is stored natively as RDF or viewed as RDF via middleware. Like SQL, SPARQL selects data from the query data set by using a SELECT statement to determine which subset of the selected data is returned. Also, SPARQL uses a WHERE clause to define graph patterns to find a match for in the query data set. A graph pattern in a SPARQL WHERE clause consists of the subject, predicate and object triple to find a match for in the data. However, the triple patterns in a query do not necessarily map to well-formed OWL-DL constructs. In the following, an example of a SPARQL query:

```
PREFIX sch-ont:<http://education.data.gov.uk/def/school/>
SELECT ?name WHERE {
?school a sch-ont:School.
?school sch-ont:establishmentName ?name.
?school sch-ont:districtAdministrative
<http://statistics.data.gov.uk/id/local-authority-district/00AA>.
}
ORDER BY ?name
```

OWL-based QLs are still in their infancy compared to those for RDF. OWL-QL is a well known language for querying OWL data and is an updated version of the DAML Query. Unlike the RDF query languages, it focuses on the querying of schema rather than instance data language. Other languages exist such as the ASK queries of DIG protocol or nRQL queries of RacerPro system. They

[12]https://www.w3.org/TR/rdf-sparql-query/

have well-defined semantics based on the DL model theory. However, DIG queries are limited to atomic (TBox or RBox or ABox) queries whereas nRQL supports only conjunctive ABox queries.

Ontology Editors are tools that allow users. to visually manipulate, inspect, browse, code ontologies, support the ontology development and maintenance task. Many ontology editors are available. Protégé[13] and Neon Toolkit[14] are the most known and used ones. The first, Protégé (Stanford University School of Medicine, n.d.) is a free, open-source platform that provides a growing user community with a suite of tools to construct domain models and knowledge-base applications with ontologies. It implements a rich set of knowledge-modeling structures and action that support the creation, visualization, interrogation and manipulation of ontologies in various representation formats. On the other hand, Neon Toolkit is especially suited for heavy-weight projects like multi-modular ontologies, multilingual, ontology integration, etc. Both editors can be extended by a plug-in architecture and Java based application programming interface (API) for building knowledge-base tools and applications.

1.4 The DL Reasoner

A central aspect of logic-based knowledge representation formalisms is the ability to employ (efficient) automated deduction systems. In DLs these systems, so-called reasoners, implement decision procedures (for instance, the *Tableau algorithm* [BS01], *Hyper-Tableau* [MSH09] and *Consequence-Based* [SKH11]). They infer logical consequences from a set of explicitly asserted facts or axioms and typically provide automated support for reasoning tasks [BCM+03]. A brief description of main logical reasoning services and a discussion of the functional features of the set of modern ontology reasoners will by provided in the remainder of this section.

1.4.1 Standard Reasoning Services

Modern DL reasoners support a number of primary logical services such as consistency, classification, instance checking and entailment checking, and secondary services such as explanation for entailments and inconsistency, (semantic) module extraction and ontology based data access (OBDA). In this subsection, we recall the standard logical services provided by ontology reasoners. They are namely:

Satisfiability A concept C is satisfiable with respect to $\mathcal{O}$ if it exists at least one model $\mathcal{I}$ of $\mathcal{O}$ where $C^{\mathcal{I}} \neq \emptyset$. Otherwise, i.e. $C^{\mathcal{I}} = \emptyset$ in all models of $\mathcal{O}$, C is *unsatisfiable* (thus $\mathcal{O} \models C \sqsubseteq \bot$). The

[13]http://protege.stanford.edu
[14]http://neon-toolkit.org/

computational complexity of deciding satisfiability is N2ExpTime for the $\mathcal{SROIQ}$ DL, and PTime for $\mathcal{EL}$ as it has been shown in [HKS06] and [BBM11]. If $\mathcal{O}$ contains one or more unsatisfiable atomic concepts, it is said to be incoherent, otherwise $\mathcal{O}$ is coherent.

Consistency A ontology $\mathcal{O}$ is consistent if there exists an interpretation $\mathcal{I}$ that satisfies every axiom in $\mathcal{O}$, i.e. $\mathcal{O}$ has a model $\mathcal{I} \models \mathcal{O}$. Deciding consistency of an ontology written over a DL $\mathcal{L}$ is as computationally complex as deciding satisfiability in $\mathcal{L}$ (as discussed below).

Entailment An axiom α is entailed by an ontology $\mathcal{O}$, denoted $\mathcal{O} \models \alpha$, if in every model $\mathcal{I}$ of $\mathcal{O}$ we have $\mathcal{I} \models \alpha$. Deciding entailment over a $\mathcal{SROIQ}$ (DL) is reducible to deciding satisfiability in $\mathcal{SROIQ}$ [BCM+10]; checking if $\mathcal{O} \models A \sqsubseteq B$ can be done by checking if $A \sqcap \neg B$ is unsatisfiable with respect to $\mathcal{O}$.

Realisation Realising an ontology entails finding all atomic concepts each instance is a type of, i.e., testing entailments of the form $\mathcal{O} \models C(a)$, where a is an instance and C an atomic concept in $\widetilde{\mathcal{O}}$. In the worst case, this task requires performing $|N_C| \times |N_A|$ entailment tests.

Classification Classification is the task of computing all subsumptions between atomic concepts (and atomic roles), i.e., testing entailments of the form $\mathcal{O} \models A \sqsubseteq B$, where A, B are atomic concepts (respectively, $A \sqsubseteq B$, where r, s are atomic roles). In the worst case, this task requires performing a quadratic number of entailment tests in the order of $|\widetilde{\mathcal{O}}|$ restricted to atomic concepts (respectively, atomic roles) only. The problem of computing role hierarchies can be reduced to a concept hierarchy computation problem.

Note that, for DLs with negation, the first three problems are inter-reducible, and thus belong in the same complexity class: verifying whether an axiom $\alpha A \sqsubseteq B$ follows from $\mathcal{O}$ can be done by testing whether $A \sqcap \neg B$ is unsatisfiable (and vice versa). Similarly, consistency of an ontology $\mathcal{O}$ is reducible to satisfiability, by verifying whether $\top$ is satisfiable, i.e., whether $\mathcal{O} \models \top \sqsubseteq \bot$.

Among these tasks, *classification* is considered as the key reasoning task. It computes the full concept and role hierarchies. Explicit and implicit subsumption will be derived to help users navigating through the ontology towards mainly explanation and/or query answering respective tasks. Thus, it is supported by all modern DL reasoners and its duration is often used as a performance indicator to benchmark reasoning engines [Abb12]. From an application point of view, an ontology should be classified regularly during its development and maintenance in order to detect undesired subsumptions as soon as possible. To make this feasible, in particular for large ontologies, classification has to be carried out as swiftly as possible. However, the increasing complexity of modern ontologies is an actual hamper towards reaching such a goal. Reasoner designers are keeping optimizing their reasoning algorithms to overcome the complexity and the intractability of the expressive DLs underpinning the OWL language.

1.4.2 Landscape of modern DL reasoners

A reasoner is a program that infers logical consequences from a set of explicitly asserted facts or axioms and typically provides automated support for reasoning tasks such as classification, debugging and querying. Any reasoner could be distinguished according to an important set of functional attributes such as the inference algorithm, the supporting logic, the degree of completeness of reasoning, the implementation language and others. In this subsection, we will try to draw up commonly exhibited features of modern DL reasoners. Rather than comparing reasoner performances, we will be discussing relevant functional characteristics of these implementations and try to classify them into categories. According to the most recent survey about ontology reasoners established by Matentzoglu et al. [MLH+15], there exist almost 35 known ontology reasoners most of them were developed during the years between 2010 and 2015. These realisations could be categorized according to various criteria that we discuss in the following.

1.4.2.1 Comparison Criteria of DL reasoners

Reasoner comparison criteria described in this subsection stem from a literature review about surveys discussing modern reasoners, for instance the comparison of reasoners for large EL ontologies established by Dentler et al. [DCtTdK11], the survey work conducted by S. Abburu [Abb12] and more recently the reasoner meta-study set up by Matentzoglu et al. [MLH+15]. Our study covers most of the dimensions they described. Literally, these criteria are:

Implementation features. This category describes basic characteristics which determines the usability level of the reasoner implementation. This includes the reasoner programming language, the access interface (Jena, OWL API or command line), the type of its licence and thus its availability and finally editors that support integrating it.

Supported reasoning services. Most reasoners support the standard reasoning services related to the TBox. However, fewer of them are asserted to support ABox related tasks, such as realisation, conjunctive query answering and ABox consistency checking. Justification is a non standard reasoning service, however providing it would be valuable for users. In fact, justification would allow to check an ontology for inconsistent concepts (unsatisfiable) and generates an explanation for each reported inconsistency.

Supported levels of expressivity. It indicates the logical fragments of OWL supported by the reasoner. Besides, we will be checking reasoners against a minimal support for datatypes. Finally, the support of rule language, SWRL[15], will be another indicator helping categorising the reasoners.

Reasoning Methodology. This covers the different primary calculus underlying a reasoner core reasoning service. According to Matentzoglu et al. [MLH+15], one can distinguish between four categories of reasoning methodologies: *I) Consequence-based* (**CB**) [SKH11] which relies on adding logical consequences (entailments) to a knowledge base (KB) without the need to check possible consequences that are not entailed by the knowledge base; *II) Model construction* (**MC**) techniques are based on constructing models based on the KB and then checking its consistency. This category includes those reasoners relying on *Tableau* (**T**) [BS01] and *Hyper-tableau* (**HT**) [MSH09] algorithms; *III) Rewriting* (**RW**) is a technique used to expand a KB by rewriting the facts of the KB, to be used for a specific task, e.g. query answering, making the reasoner less reliant on the terminological aspect and more involved in the data aspect; and finally *V) Hybrid methodologies* (**HM**) which include reasoners employing combined reasoning techniques, for example consequence-based techniques with tableau-style techniques; and those reasoners making use of ontology modularity (**MOD**) in order to combine other existing and well performing reasoners.

Reasoning algorithm completeness and soundness. An algorithm is *sound* if, any time it returns an answer, that answer is true. An algorithm is *complete* if it guarantees to return a correct answer for any arbitrary input (or, if no answer exists, it guarantees to return failure). Soundness is a weak guarantee. It does not promise that the algorithm will terminate. Soundness and Completeness are related concepts, since they are the logical converse of each other. Soundness would admit that if an answer is returned that answer is true, whereas completeness says that an answer is true if it is returned. Most of the underlying reasoning methodologies have been proven to be sound. However, we can distinguish some particular categories of reasoners where completeness is not guaranteed for all OWL profiles. To make it clear, we will be denoting by **SC** reasoners ensuring soundness/completeness, **SI** for the ones sound and incomplete, **SC/Profile** means that the reasoner is sound and complete only for particular OWL profile(s). For instance, **SC/OWL EL** means sound and complete for OWL 2 EL.

1.4.2.2 Overview of DL reasoners

In this section, we present an overview 8 DL reasoners that will be considered throughout this thesis. For completeness sake, we will also reference other relevant DL reasoners in the literature. Finally, we will draw a comparison of theses engines according to afore discussed functional criteria.

[15]https://www.w3.org/Submission/SWRL/

FacT++ and JFacT [TH06] are the successors of the Fact reasoner (FAst Classification of Terminologies). They employ different architecture and a more efficient implementation. For instance, FaCT++ is written in C++, whereas JFact[16] is the Java port of FaCT++. The design of both reasoners matches very closely, except for the low level details where Java offers different ways of implementing reasoning procedures. JFact provides an improved of the datatype support. Worth noting, JFact does not include some parts of FaCT++ like the DIG interface and the command line support are absent. Both Fact++ and JFact completely support OWL 2 and implement a tableau algorithm [BS01] with several optimization techniques.

Pellet [SPG$^+$07] supports full OWL 2 and DL safe rules. It also implements an optimized tableau algorithm. It was the first reasoner fully supporting OWL 1 DL and its underpinning DL $\mathcal{SHOIN}(\mathcal{D})$ and has been extended to OWL 2 ($\mathcal{SROIQ}(\mathcal{D})$). Pellet handles the ontologies through Jena as well as OWL API interfaces and supports the explanation of bugs.

TrOWL [TPR10] is implemented in Java and supports OWL 2, offering sound and complete reasoning for OWL 2 EL and OWL 2 QL, and approximate reasoning for OWL 2 DL. Inference services include classification and conjunctive query answering. TrOWL includes an OWL 2 EL reasoner (REL) to compute the classification and an OWL 2 QL reasoner (Quill) to answer conjunctive queries. Reasoning with OWL 2 DL ontologies is achieved by means of a syntactic approximation into OWL 2 EL or a semantic approximation into OWL 2 QL, depending on the reasoning task. TrOWL can be used through several interfaces, including OWL API.

ELK [KKS11] implemented in Java, is a Consequence-Based reasoner for a subset of OWL 2 EL. ELK uses multiple cores/processors by parallelising multiple threads. It supports different reasoning tasks, which include classification, consistency checking, subsumption, and realization. The classification procedure is different from other algorithms for OWL 2 EL. For instance, it includes several optimizations such as concurrency of the inference rules. ELK can be used through several interfaces, including OWL API.

HermiT [GHM$^+$12] implements an Hyper-Tableau reasoning algorithm with several optimization techniques. It supports OWL 2 and DL safe rules. Historically, it was the first DL reasoner that was able to classify some large ontologies (such as GALEN-original) thanks to a novel and efficient classification algorithm. HermiT is implemented in Java, and is accessible through several interfaces, including the OWL API and a Protégé plug-in.

[16]`http://jfact.sourceforge.net/`

MORe [RGH12] is an OWL 2 meta-reasoner that exploits module extraction techniques to divide complex reasoning tasks into simpler ones that can be solved using different reasoners. The modules of the ontology in the OWL 2 EL profile are solved by the ELK reasoner, and the more expressive ones are handled using HermiT, JFact or Pellet. MORe currently supports classification and concept satisfiability. It is implemented in Java and accessible through the OWL API and using a Protégé plug-in.

Konclude [SLG14] is a high-performance reasoner for the Description Logic SROIQ. It is written in C++ and supports the full OWL 2 DL profile except datatypes. Konclude design is combination between Consequence-Based techniques with an optimized Tableau-like algorithm. It also uses parallel processing techniques to handle non-deterministic branches and the higher-level reasoning tasks, e.g. satisfiability and subsumption tests. Unfortunately, there is no Protégé plugin to work with Konclude, and some configurations are required to be able to use it with the OWL API.

Table 1.4 summarizes all of the above detailed functional features of the selected reasoners. The comparison is set up according to the classification criteria introduced in the Subsection 1.4.2.1. In this table, (OW) stands for the OWL API and (J) for the Jena API. By (PG), we referenced the Protégé editor and by (NT) the Neon-Toolkit.

1.5 The Predictive Modelling

Predictive modelling [HTF01] is a name given to a collection of mathematical and statistical techniques having in common the goal of finding hidden patterns in data and developing functions capable with an acceptable level of reliability to forecast future outcomes and trends. In other words, predictive modelling is the general concept of analysing current data in order to build models that are capable of making future predictions about a domain of interest. Typically, such a model includes *machine learning algorithms*.

Arthur Samuel in 1959 [Sam59] defined *machine learning* (ML) as the field of study that gives computers the ability to learn without being explicitly programmed. The learning that is being done is always based on some kinds of observations or data, commonly called a dataset. In general, a machine learning is about learning to do better in the future based on what was experienced in the past. A classic setting is the supervised learning, where the ML algorithm has access to a set of training examples along with their labels and must learn a model that is able to accurately predict the label of future (unseen) examples. On the other hand, an unsupervised learning algorithm has no access to the labels of the training data. Clustering is a classic example of unsupervised ML task, aiming at

Feature Category	**FacT++**	**JFacT**	**Pellet**	**TrOWL**	**ELK**	**HermiT**	**MoRE**	**Konclude**
Prog. Language	C++	Java	Java	C++	Java	Java	Java	C++
Prog. Interface	OWL+Jena	OWL	OWL	OWL+Jena	OWL	OWL	OWL	OWL+Jena
Editors Support	PG	PG+NT	PG+NT	PG	PG	PG+NT	PG	∅
Open Source	Yes	Yes	Yes	No	Yes	Yes	Yes	Yes
ABox checking	Yes	Yes	Yes	Yes	No	Yes	No	Yes
Datatype Support	limited	limited	Yes	limited	No	Yes	Yes	Yes
Justification	No	No	Yes	No	No	No	No	No
Rule Support	No	No	Yes	No	Yes	Yes	No	Yes
Native Profile	DL	DL	DL,EL	Dl,EL	EL	DL	DL,EL	DL
Expressivity	$\mathcal{SROIQ}$	$\mathcal{SROIQ}$	$\mathcal{SROIQ}$	$\mathcal{SROIQ}$	$\mathcal{EL}+$	$\mathcal{SROIQ}$	$\mathcal{SROIQ}$	$\mathcal{SROIQV}$
Methodology	MC (T)	MC (T)	MC (T)	CB	CB	MC (HT)	HM (MOD+MC)	HM (CB+MC)
Soundness/ Completeness	SC	SC	SC	SC/ (EL+QL)	SC/ (EL)	SC	SC	SC

Table 1.4: Comparing the DL reasoner functional attributes.

assigning data into similar groups. The generalization ability of the learned model (i.e., its performance on unseen examples) can sometimes be guaranteed using arguments from statistical learning theory.

The general layout for machine learning process is described below, more details are given in [Abb14]:

Data collection: the data related to domain of concern is extracted. The data will be raw and unstructured format. Hence pre-processing measures must be adopted.

Data pre-processing: the initial dataset is subjected for pre-processing. Pre-processing is performed to remove the outliers and redundant data. The missing values are replaced by normalization and transformation.

Development of models: the pre-processed data is subjected to different machine learning algorithms for development of models. The models are constructed based on classification, clustering, pattern recognition and association rules.

Knowledge Extraction: the models are evaluated to represent the knowledge captured.

Instance	**Feature1**	**Feature2**	**...**	**Feature$M$**	**Class**
1	XX	XX	...	XX	good
2	XX	XX	...	XX	bad
...					
N	XX	XX	...	XX	good

Table 1.5: Dataset example for supervised learning [Kot07]

The task of predictive modelling described above is of practical importance to many domains, ranging from medicine and social sciences to various areas of engineering.

1.5.1 Problem formulation for the supervised learning

By referring to [Kot07], we will introduce some basic concepts of machine learning (ML). Explicitly, every instance in any dataset used by machine learning algorithms is represented using the same set of features. Then, each instance is an M-dimensional vector $x^{(i)} = [{x_1}^{(i)}, {x_2}^{(i)}, \ldots, {x_M}^{(i)}]$ called a *feature vector*, where i referred to the i-th instance in the dataset, $i \in [1 \cdots N]$ with N the total number of instances and M the total number of features. We would also use $\mathcal{X}$ to denote the feature set containing N feature vectors, otherwise the space of input values. The features may be continuous, categorical or binary. In this thesis, we are particularly interested in supervised learning. The vector of all labels is denoted $\mathcal{Y}$, where y_i is the label of the i-th instance. Thus, the dataset could be denoted by $\mathbb{D} = [\mathcal{X}_{N,M} \mid \mathcal{Y}^T]$. An example of a dataset for a supervised learning is given in the Table 1.5. A supervised problem is called *classification* when the labels are qualitative ones (e.g. bad/good) and *regression* in the case of quantitative values (e.g. temperature value). Here, it is important to notice the difference between the "classification" task in machine learning and the afore described "classification" in ontology reasoning (Section 1.4.1). To avoid any ambiguity, in the rest of this thesis, we will refer to latter one as "DL classification" and the former one as "ML classification". When, the label vector, $\mathcal{Y}$, contains only two discrete labels, then the ML classification problem is called a *binary ML classification*, otherwise it is a *multi-class ML classification* problem.

Given the training dataset $\mathbb{D}$, a supervised learning task is the process of inferring a function often referred to as a hypothesis or a model, $\mathcal{F} : \mathcal{X} \rightarrow \mathcal{Y}$, which "*best*" predicts the label y from the feature vector x for any pair (x, y) drawn from $\mathbb{D}$. The predictive functions would be a distribution of probabilities, a neural network, a decision tree, etc. We will further describe different categories of supervised learning algorithms in Section 1.5.2. After constructing the predictive model, when a new instance appears, x', that does not belong to the dataset $\mathbb{D}$ but having the same type and described with the same kind of features as the ones in $\mathbb{D}$, then the task is to predict its exact label using the trained model $\widetilde{y} = \mathcal{F}(x')$.

1.5.2 Supervised machine learning Algorithms

Following [Kot07], the supervised learning algorithms can be divided or grouped as logical based algorithms such as decision trees [Qui93], Artificial Neural Network (ANN) based techniques such as multi-layered perceptron [RHW86], Statistical learning algorithms such as Bayesian Networks, Instance based learning algorithms [AKA91] and Support vector machines [Pla99]. In the following, we recall basic notions of each category:

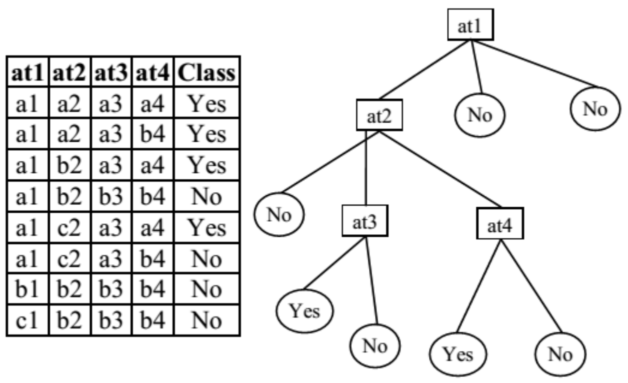
1.5.2.1 Logic-based algorithms

The logic (symbolic) based category of learning algorithms is characterized by two main groups of methods: decision trees and rule-based classifiers.

Decision Trees (DT) are trees that classify instances by sorting them based on feature values, where each node in a decision tree represents a feature in an instance to be classified, and each branch represents a value that the node can assume. Instances are classified starting at the root node and sorted based on their feature values. The feature that best divides the training data would be the root node of the tree. There are different methods to extract the features that best divides the training data such as information gain and gini index. C4.5 algorithm proposed by J. R. Quinlan [Qui93] is one of the most popular and the efficient method in decision tree-based approach. The algorithm creates a tree model by using values of only one attribute at a time. According to [Kot07], the decision tree induction, which was initially designed to solve classification problems, has been extended to deal with single or multi-dimensional regression. The major benefits of decision trees are *i)* produce intensive results, *ii)* easy to understand, *iii)* and holds well-organized knowledge structure. It is also possible that decision trees can be translated into a set of rules by creating a separate rule for each path from the root to a leaf in the tree. However, rules can also be directly induced from training data using a variety of rule-based algorithms.

1.5.2.2 Bayesian-based algorithms

The Naive Bayes machine learning method [Ris01] is based on the so-called Bayesian theorem. This method models the class posterior with the assumption that the attributes are conditionally independent in each class. The Naive Bayes classifier produces probability estimates rather than predictions.



at1	at2	at3	at4	Class
a1	a2	a3	a4	Yes
a1	a2	a3	b4	Yes
a1	b2	a3	a4	Yes
a1	b2	b3	b4	No
a1	c2	a3	a4	Yes
a1	c2	a3	b4	No
b1	b2	b3	b4	No
c1	b2	b3	b4	No

Figure 1.2: A sample training set (left) and its corresponding Decision Tree (right)

The probability estimate is the conditional probability distribution of the values of the class attribute based on the values of other attributes:

$$p(x|y = k) = \prod_{i=1}^{attr} p(x^i|y = k) \tag{1.1}$$

In this way Naive Bayes classifier is just an alternative way of representing a conditional probability distribution and can only represent simple distributions. But Bayesian Network is a theoretically well-founded way of representing probability distributions concisely and comprehensibly in a graphical manner. The structure of Bayesian Network (BN) is represented by a directed acyclic graph (DAG), which is a network of nodes; represents attributes, connected by directed edges, expresses dependencies between attributes, in such a way that there are no cycles.

1.5.2.3 Artificial Neural Network based algorithms

They are learning algorithms inspired from the neurons in human brain as firstly described [MP88]. The learning network comprises of interconnected neurons as a function of input data. Based on the synapse received from input data, weights are generated to compute the output function. The networks can either be feed forward or feed-back in nature depending upon the directed path of the output function. The error in input function is minimized by subjecting the network for back-propagation which optimizes the error rate. The network may also be computed from several layers of input called as multilayer perceptron proposed by [RHW86]. Neural networks have immense applications in pattern recognition, speech recognition and financial modelling. Neural networks can actually perform a number of regression and/or classification tasks at once, although commonly each network performs only one.

1.5.2.4 Linear and logistic-based regression algorithms

It is a category of linear learning algorithms employed for both classification and regression tasks. They are characterized by linear function that assigns a score to each possible category k by combining the feature vector of an instance with a vector of weights, using a dot product. The predicted category is the one with the highest score. This type of score function is known as a linear predictor function and has the following general form:

$$score(X_i, k) = \beta_k \times X_i,$$

where X_i is the feature vector for instance i, β_k is the vector of weights corresponding to category k, and $score(X_i, k)$ is the score associated with assigning instance i to category k. Examples of such algorithms are Logistic regression, Multinomial logistic regression, Probit regression and Linear discriminant analysis. Further details about these algorithms is provided by [Fil08]. In this thesis, we are particularly interested in Logistic regression algorithm introduced by [FHT98]. It is an approach to learning functions of the form $f : X \to Y$, or $P(Y|X)$ in the case where Y is discrete-valued, and $X = \langle X_1 \ldots X_n \rangle$ is any vector containing discrete or continuous variables. Logistic Regression assumes a parametric form for the distribution $P(Y|X)$, then directly estimates its parameters from the training data. One highly convenient property of this form for $P(Y|X)$ is that it leads to a simple linear expression for classification.

1.5.2.5 Lazy algorithms

They are instance-based SML algorithms. The k-nearest neighbour [AKA91] is one of the most known algorithms of this category. It is recognized as a simple machine learning algorithm capable to handle ML regression and classification tasks. The k-nearest neighbour algorithm requires no model to be fit. Given an observation x, the classifier finds the k-nearest samples in the training set according to a distance measured and takes a majority vote among the classes of these samples. This is equivalent to estimating the posterior probabilities $p(y|x)$ by the proportions of the classes in the neighbourhood. The best choice of k depends upon the data. Generally, larger values of k reduce the effect of noise on the classification, but make boundaries between classes less distinct. The parameter k is determined with cross-validation between 1 and $\sqrt{n}$.

1.5.2.6 Support Vector Machine

Support Vector Machine method is based on the idea of learning a linear hyperplane from the training set that separates positive samples from negative samples. Basically, they are supervised learning algorithms based on non-probabilistic classification of dataset into categories in high dimensional space.

The algorithm was first proposed by Cortes and Vapnik [CV95]. The training dataset is assumed as a p-dimensional vector which is to be classified using (p-1) dimensional hyperplane. The largest separation achieved between data points is considered optimal. Support Vector Machines (SVMs) have been used for classification, regression and outliers detection. There are number of benefits for using SVM such as: *i)* it is effective is high dimensional space, *ii)* uses a subset of training points in the decision function (called support vectors), hence it is also memory efficient, *iii)* it is versatile because holds different kernel functions can be specified for the decision function. Common kernels are provided like the linear, the polynomial, the radial basis function and the sigmoid but it is also possible to specify custom kernels. More detailed discussion about kernels for SVM methods could be found in [Pla99].

1.5.2.7 Ensemble methods: Bagging, Boosting and Stacking

Strictly speaking, this is not a machine learning algorithm, and it is more like a kind of method or a general strategy of optimization. Ensemble methods usually combine a number of simple weak machine learning algorithm to make more reliable decisions. Bagging and Boosting empirically studied in [MO97] are two widely known ensemble learning approaches. The first approach is based on an idea of making various samples of the training set. A classifier is generated for each of these training set samples by a selected machine learning algorithm. In this way, for k variations of the training set, we get k particular classifiers. The result will be given as a combination of individual particular classifiers. On the other hand, boosting performs experiments over training sets as well. This method works with weights of training examples. Higher weights are assigned to incorrectly classified examples. That means, that the importance of these examples is emphasized. After the weights of training sets are updated, a new (particular) classifier is generated. A final classifier is calculated as a combination of particular classifiers. In both bagging and boosting, the multiple models being combined are all the same type. Stacking [MO97] is able to combine models built by different learning algorithms. For example, we can combine a decision tree, a naive Bayes model, and a Bayesian network model trained from the same data, in order to form a new ML classifier. Stacking relies a *meta-learner*[17] to predict which classifier is the most reliable one and to decide how best to combine the predictions of all the base classifiers.

Random Forest proposed by L. Breiman [Bre01] is one of the well known learning algorithms based on ensemble strategies. Breiman has combined two main approaches of ensemble learning, i.e. the sub-sampling of the examples like in bagging and the sub-sampling of the features like in feature selection methods. More in details, a Random Forest is an ensemble of decision trees constructed for building the predictive model. It can be used for both classification and regression tasks.

[17]We will describe the meta-learner concept more in details in Chapter 2.

1.5.3 Preprocessing and feature selection

The success of the supervised learning process is heavily dependent on the quality of the data provided for the training. A learner has only the input to learn from. If the data is inadequate or irrelevant then the concept descriptions will reflect this and misclassification will result when they are applied to new data.

Feature Selection Feature selection and dimensionality reduction methods [AT16] are procedures of finding a set of most compact and informative original features for the purpose of better prediction of the output of the learning algorithm. For the classification problem, feature selection aims to select subset of highly discriminant features. In other words, it selects features that are capable of discriminating samples that belong to different classes. While having more features endows a learning algorithm with a greater discriminating power, performance degradation often sets in when many irrelevant or redundant features are included. The inclusion of irrelevant and redundant features also increases the computational complexity of the learning algorithm. Besides, it is also known that feature selection can potentially benefit data visualization and data understanding, data storage reduction and the easy deployment of the learning algorithm. Consequently, feature selection has been an area of much research effort in various learning tasks. Plenty of feature selection methods are available in literature, making to choose a little bit confusion.

As previously explained in [AT16], feature selection methods can be categorized into supervised, unsupervised and semi-supervised ones according to whether the training set is labelled or not. Supervised feature selection methods can further be assorted into filter models, wrapper models and embedded models. The filter model relies on measures of the general characteristics of the training data such as distance, consistency, dependency, information, and correlation. *Relief*, *Fisher score* and *Information Gain* based methods are among the most representative algorithms of this category. The wrapper model uses the predictive accuracy of a predetermined learning algorithm to determine the quality of selected features. These methods are prohibitively expensive to run for data with a large number of features. Due to these shortcomings in each model, the embedded model was proposed to bridge the gap between the filter and wrapper models. First, it incorporates the statistical criteria, as filter model does, to select several candidate features subsets with a given cardinality. Second, it chooses the subset with the highest classification accuracy. Thus, the embedded model usually achieves both comparable accuracy to the wrapper and comparable efficiency to the filter model. The embedded model performs feature selection in the training time.

Discretization. Most classification tasks in machine learning involve learning to distinguish among nominal class values, but may involve features that are ordinal or continuous as well as nominal. While many machine learning algorithms have been developed to deal with mixed data of this kind, recent research shows that common machine learning algorithms such as instance based learners and naive Bayes benefit from treating all features in a uniform fashion. One of the most common methods of accomplishing this, is the *discretization* method. Discretization [GLS+13] stands for the process of transforming continuous valued attributes to nominal ones. This process could be supervised or unsupervised one, global or local and static or dynamic. We are more interested in the supervised discretization methods. The latter ones transform the continuous feature values into nominal ones by considering the label class distribution of the target variable. Authors of [GLS+13] have previously highlighted the good potential of these methods in enhancing the feature selection process. In practice, one of the best general techniques for supervised discretization is the entropy-based method with the MDL stopping rules proposed by Fayyad and Irani [FI93]. It uses a minimum entropy heuristic to discretize numeric features. The algorithm relies on the class entropy of candidate partitions to select a cut point for discretization. The method can then be applied recursively to the two intervals of the previous split until some stopping conditions are satisfied.

1.5.4 Predictive models assessment measures

Before any predictive model can be used with confidence, adequate validation or assessment of the magnitude of the errors that may result from their use should be performed. Model validation, in its simplest form, is a comparison between simulated and observed values. In research or experimentation, to measure the performance achieved by a trained model, a test set consisting of examples with known labels is needed. The predictive model is constructed based on the training set, and then applied to the test set, in order to measure the performance of the model by comparing the predicted labels with the true labels (which were not available to the training algorithm). These performances could be assessed with different metrics according to the nature of the learning task under examination, i.e. ML classification or ML regression. In our experiments, we often had a fixed database of labelled examples available. Thus, we were faced with a dilemma: we would like to use all the examples for training, but we would also like to use many examples as an independent test set. That is why, we deployed a cross-validation procedure for overcoming this dilemma. The different assessment measurements as well as the cross validation procedure are described in the following:

		Predicted Labels		
		Positive	**Negative**	Total
Actual Labels	**Positive**	True Positive (**TP**)	False Negative (**FN**)	$P = TP + FN$
	Negative	False Positive (**FP**)	True Negative (**TN**)	$N = FP + TN$

Table 1.6: Confusion matrix of a binary ML classification task

1.5.4.1 Assessment measures of ML classification models

In machine learning, the performance of a predictive model for ML classification task is basically evaluated by a *confusion matrix* (also called contingency table) [KP98]. The confusion matrix is a square matrix, $|Y| \times |Y|$, where $|Y|$ is the number of labels to be predicted. It shows how the predictions are made by the model. The rows correspond to the known labels of the data. The columns correspond to the predictions made by the model. Each cell (i, j) of the matrix contains the number of instances from the dataset that actually have the label i and were predicted as with label j. A perfect prediction model would have its confusion matrix with all elements on the diagonal. In case of a common binary (positive/negative) ML classification problem, confusion matrix represents the relationship between actual class outcome and predicted class outcome based on four estimates measures given in Table 5.3. The acronym TP, FN, FP, and TN of the confusion matrix cells refers to the following:

- **TP** = true positive, the number of positive cases that are correctly identified as positive;
- **FN** = false negative, the number of positive cases that are misclassified as negative cases;
- **FP** = false positive, the number of negative cases that are incorrectly identified as positive;
- **TN** = true negative, the number of negative cases that are correctly identified as negative.

It is worth noting that confusion matrix could be easy generalized to the problem of multi-class ML classification. A variety of assessment measures of ML classification algorithms are defined based on the confusion matrix. We recall the common ones computed for a binary ML classification problem:

Accuracy and Error rate Accuracy, the most common metric for ML classifier evaluation, assesses the overall effectiveness of the algorithm by estimating the probability of the true value of the class label. On the other hand, the error rate is an estimation of misclassification probability according to model prediction.

$$\textbf{Accuracy} = \frac{TP + TN}{P + N} \quad , \quad \textbf{Error-rate} = 1 - Accuracy = \frac{FP + FN}{P + N} \tag{1.2}$$

Precision, Recall , and F1-Score These are common measures of model effectiveness. *Precision* is a measure of how many errors we make in predicting instances as being of some label l. On the other hand, *Recall* assesses how good we are in not leaving out instances that should have been predicted with the label l.

$$\textbf{Precision} = \frac{TP}{TP + FP} \quad , \quad \textbf{Recall} = \frac{TP}{TP + FN} \tag{1.3}$$

In a binary problem, a perfect model will capture all positive examples (Recall = 1), and score as only the examples that are in fact (Precision = 1); from an analytical point of view, it is desirable to increase recall without sacrificing accuracy. That is why, both measures are misleading when considered separated. Usually, we rather employ the *F1-Score* (**F1**), also known as the F-Measure, to assess the model. This score combines both *Precision* and *Recall* into a single value. The F1 values vary within the unit interval. High values close to 1 would indicate a good predictive model.

$$\textbf{F1-Score} = \frac{2 \times Recall \times Precision}{Recall + Precision} \tag{1.4}$$

ROC and AUC Sometimes we would like to compare several competing models rather than estimate their performance independently. In order to do that, we use a technique developed in the 1950s for analysis of noisy signals: the Receiver Operating Characteristic (ROC) Curve. A ROC curve characterizes the relation between the benefits, which stand for the correct detection rate or the true positive rate and cost which stand for false detection or the false positive rate. The true positive rate is the same as the recall and the false positive rate is equal to the ratio between the number of false positive and the total number of negative instances. A ROC curve gives indication if a trained model is superior to another model over a wide range of operating points. On the other hand, the area under the curve (AUC) computed based on the ROC curve is often used to summarize the performance of the predictive model into a single metric. The larger the AUC is, the better are the model performances.

Imbalanced data problem and assessment measures In case of multi-class ML classification where the size of the label vector is superior to 2, we are very often concerned with the issue of the *imbalanced dataset*. In the latter one, a class stands for the collection of observations sharing the same label. According to Bao-Gang et al. [HD14], a dataset is considered imbalanced if at least one of the classes (minority class(es)) contains much smaller number of samples than the remaining classes (majority classes). In this case, supervised machine learning algorithms can have good accuracy on the majority class but very poor accuracy on the minority class(es). Besides, usual model quality metrics such as Precision, Recall, F1-Score and even AUC could be misleading. Therefore, we retained from [HD14], two further assessment measures known for their resistance to the problem of data imbalance. Namely, this is the ***Kappa coefficient*** (κ) and the ***Matthews correlation coefficient*** (**MCC**).

Kappa coefficient (κ): also known as the Cohen's kappa. It measures the agreement of prediction with the true class. The value of *Kappa* lies between -1 and 1, where 0 represents agreement

Kappa value	**Level of Agreement**
0	Absence
$(0, 0.20]$	Minimal
$(0.20, 0.40]$	Weak
$(0.40, 0.060]$	Moderate
$(0.60, 0.80]$	Strong
$(0.80, 1]$	Almost perfect

Table 1.7: Interpretation of Kappa values

due to chance. The value 1 represents a complete agreement between both values. In rare cases, *Kappa* can be negative. Formally, Kappa is computed as follows:

$$Kappa = \frac{P_o - P_c}{1 - P_c} \tag{1.5}$$

where P_o is the observed proportion of the agreement, otherwise the probability of success in the classification and the P_c is the proportion or the probability of success expected by chance. Table 1.7 illustrates in more details how we would interpret the Kappa values.

Matthews correlation coefficient (**MCC**): it is performance measure barely influenced by imbalanced test sets since it considers mutually accuracies and error rates on all labels. Therefore, it involves all values of the confusion matrix. MCC values range from 1 for a perfect prediction to -1 for the worst possible prediction. A value of MCC close to 0 indicates a model that performs randomly. Formally, MCC is computed as follows based on the confusion matrix in case of binary classification problem. This formula is also generalised in case of multi-class classification problem:

$$MCC = \frac{TP \times TN - FP \times FN}{\sqrt{(TP + FP)(TP + FN)(TN + FP)(TN + FN)}} \tag{1.6}$$

1.5.4.2 Assessment measures of ML regression models

In regression models, the target class to be predicted has continuous values making the estimation process much harder. The famous Bayesian statistician George Box, said: *"All Models are wrong, but some are useful"*. The goal of analysing the results of a linear or nonlinear regression model is to estimate how "*closely enough*" for the model to be useful in practice, otherwise to measure the distance between the estimated output from the actual output. The evaluation process has four common components: to determine how well the model fits the data (goodness of fit), if the model fits the data better than an alternative model and if the equation fits this data differently for a different set of data.

There is a rich plethora of criteria by which they can be evaluated and compared the regression models. We recall the most commonly used ones. More details about these metrics are available in [Abb14].

RMSE The Root Mean Square Error (also called the root mean square deviation, RMSD) is a frequently used measure of the difference between values predicted by a model and the values actually observed from the environment that is being modelled. These individual differences are also called residuals, and the RMSE serves to aggregate them into a single measure showing the predictive power of the model. Let X_{obs} is the actual observed values, X_{pred} is the predicted values at time/place i and n the size of the sample, then RMSE is computed as follows:

$$RMSE = \sqrt{\frac{\sum_{i=1}^{n}(X_{obs,i} - X_{pred,i})^2}{n}} \tag{1.7}$$

Since the errors are squared before they are averaged, the RMSE gives a relatively high weight to large errors. This means the RMSE is most useful when large errors are particularly undesirable. The calculated RMSE values will have units, and RMSE for phosphorus concentrations can for this reason not be directly compared to RMSE values for chlorophyll concentrations, etc. RMSE values can be used to compare the individual model performance to that of other predictive models. It values range from 0 to $+\infty$ and have a negatively-oriented score, therefore lower values are better.

MAE Mean Absolute Error measures the average magnitude of the errors in a set of predicted values, without considering their direction. MAE has the same unit as the original data, and it can only be compared between models whose errors are measured in the same units. It is usually similar in magnitude to RMSE, but slightly smaller. Let X_{obs} is the actual observed values, X_{pred} is the predicted values at time/place i and n the size of the sample, then MAE is computed as follows:

$$MAE = \frac{\sum_{i=1}^{n}|X_{obs,i} - X_{pred,i}|}{n} \tag{1.8}$$

Like the RMSE, MAE can range from 0 to $+\infty$. It is a negatively-oriented score, therefore lower values are better.

R-Squared (R^2) The coefficient of determination (R^2) summarizes the explanatory power of the regression model and is computed from the sums-of-squares terms. Formally, R^2 is measured as follows:

$$R^2 = 1 - \frac{SSR}{SST} \tag{1.9}$$

where SSR is the sum of square of the residuals (the value that is minimized by the regression procedure) and $SST = \sum(X_{obs} - X_{pred})^2$ (the sum of the squared differences between the data points and the average of the data points). It can be seen that if the value for SSR is minimized, the value for R^2 will approach 1. Thus, the closer R^2 is to 1, the better the model is at fitting the data. We should highlight that R^2 has no unit and orders from 0 to 1 while the model has healthy predictive ability when it is near to 1 and is not analysing whatever when it is near to 0.

1.5.4.3 Cross Validation

Cross-validation is a widespread strategy to evaluate the robustness of the classifier because of its simplicity and its (apparent) universality. K-fold cross validation is typically used to provide an estimate for the mean performance of an algorithm on a held-out test set.

In ten fold cross validation, the training set is equally divided into 10 different subsets. Nine out of ten of the training subsets are used to train the learner and the tenth subset is used as the test set. The procedure is repeated ten times, with a different subset being used as the test set. The splitting of dataset during the cross validation is usually associated to stratification. The latter one ensures that the class distribution from the whole dataset is preserved in the training and test sets. Stratification has been shown to help reduce the variance of the estimated accuracy especially for datasets with many classes. The output of any k-fold cross-validation is a confusion matrix based on using each labelled example as a test example exactly once. Worth noting that cross-validation does not produce any single final classifier, and the confusion matrix it provides is not the performance of any specific single classifier. Instead, this matrix is an estimate of the average performance of a classifier learned from a training set of size $(k-1)n/k$ where n is the size of dataset. The common procedure is to create a final classifier by training on all of dataset, and then to use the confusion matrix obtained from cross-validation as an informal estimate of the performance of this classifier.

1.5.5 Machine learning working environement

Over the last decades many general-purpose machine learning frameworks and libraries emerged from both academia and industry. A detailed comparison of these realisation was established by D. Pop and G. Iuhasz [PI11]. Throughout this thesis, we will be mainly working with the Weka toolkit [HFH$^+$09]. In particular, we have used the Java API of Weka[18] as base tool to implement our automatic learning prototypes. The rational behind this choice is twofold: *i)* Weka is a rich toolkit covering a huge plethora of ML and data mining algorithms; *ii)* its API is written in Java which is the same development language as the OWL API, which makes it possible to integrate the both APIs in one single system. In that system, we are intending to integrate both Semantic Web and Machine learning technologies

[18]The Weka API is available for download at `www.cs.waikato.ac.nz/ml/weka/downloading.html`

for a better collaboration over our ontology reasoning learning problems. In the following, we give a brief description of the Weka toolkit.

WEKA [HFH$^+$09] is an assembly of tools for machine learning applications. The main strength of WEKA is that the toolkit is freely available under GNU General Public Licence, very portable because it used Java Programming Language that can run in almost all modern platform, contains a large number of data preprocessing and modelling technique, and easy to use by beginner thanks to its interactive graphical user interface. Weka supports several standard data mining tasks, more specifically, data preprocessing, clustering, classification, regression, visualization, and feature selection. All of Weka techniques are predicated on the assumption that the data is available as a single flat file or relation, where each data point is described by a fixed number of attributes (normally, numeric or nominal attributes, but some other attribute types are also supported). Weka provides access to SQL databases using Java Database Connectivity and can process the result returned by a database query. The WEKA tool incorporates the four applications within it: *1)* Weka Explorer, *2)* Weka Experiment, *3)* Weka Knowledge Flow and *4)* Simple CLI. Weka Explorer incorporates the following features:

- **Preprocess**: it is used to process the input data. For this purpose the filters are used that can transform the data from one form to another form. Basically two types of filters are used i.e. supervised and unsupervised.
- **Classify**: its corresponding tab is used for the classification purpose. A large number of classifiers are used in weka such as bayes, function, rule, tree and meta etc. Four type of test option are mentioned within it.
- **Cluster**: it is used for the clustering of the data.
- **Associate**: establish the association rules for the data.
- **Select attributes**: it is used to select the most relevant attributes in the data.
- **Visualize**: view an interactive 2D plot of the data.

1.6 Conclusion

In this chapter, we review concepts from the related areas to our research questions. These concepts are being introduced partly to make the thesis more self-contained and, more importantly, to set up the formal notations that will be used throughout this thesis.

We first presented the basic notions of the Semantic Web; then we put our focus on studying the ontologies from a knowledge representation perspective. Hence, we recalled the main features of the ontology authoring language: OWL. In particular, we studied the description logics (DL) from which

OWL has been derived. DL are particularly well suited for the representation of ontologies and they are relevant for the Semantic Web as they have precise semantics and allow for automatic reasoning. DL offer rich expressive features and different levels of logics. However, a trade-off between the DL expressivity and the complexity of reasoning should be provided. Indeed, various reasoning engines were introduced in the literature to process the ontologies in the most reliable and efficient way. We put into comparison their main functional attributes. Hence for now, we have a general idea of the coverage and directions of current reasoner development trends. We would assert that modern DL reasoners vary significantly with regard to all the examined attributes. Therefore, a critical empirical evaluation of their performances over real world ontologies is needed before selecting a reasoner for a real-life applications.

In the second part of this chapter, we reviewed the basic notions of the machine learning field. We remind that we intend to use the supervized learning approach to cope with reasoner performance variability phenomena. To meet this requirement, we recalled the common formal notations. Then we depicted the general steps in any learning process. Later on, we reviewed the main features of the most commonly used supervized machine learning algorithms. We finally outlined the ways to assess the quality of the trained predictive models. On the basis of this short review, we would admit that choosing the appropriate learning techniques is a real issue given the rich and large plethora of existing realisations. We will rely on some background knowledge in order to select the most powerful learning techniques and then empirically examine how well they would fit to our study context.

CHAPTER 2

Related Works

Contents

2.1 Introduction

The aim of this chapter is to portray the general context of this dissertation and to position our work in the current state of the art. Thus, this chapter includes both a literature overview, and the introduction of novel ideas, presented in an intertwined manner.

Our work asks the question most relevant to an end user: *"How feasible is it to reason over an ontology written in particular DL, using the best available reasoners?"*. To bring response to this interrogation, we will start discussing this feasibility from a theoretical perspective by recalling the main computational complexity classes of the ontology language, i.e. OWL, and its underpinning DLs. Then, we will study the feasibility from an empirical perspective. Hence, we will outline the practical complexity of the reasoning algorithms. We will depict some of the challenges facing both the designers and the users of modern DL reasoners. More specifically, we will shed light on the reasoner performance variability phenomena, since it is the most compelling issue in the practical use of reasoners. Our purpose is to scrutinize the research efforts which have addressed this issue and discuss the usefulness of their proposals. Hence, we will put into comparison methods and tools which tried to detect and analyse this phenomena. Then, we will tackle the issue from a broader perspective by investigating solutions proposed in different problem fields, such as machine learning (ML) and artificial intelligence (AI). More specifically, we will examine the advances in addressing the empirical hardness and the algorithm selection problems. Hence, approaches in meta-learning and algorithm portfolio construction will be reviewed. All of this investigation effort is put forward in order to depict the lessons that could be learned from these fields and highlight the ones transferable to our study context, i.e. DL based ontology reasoning.

2.2 Complexity of Ontology Reasoning

The complexity of ontology reasoning refers to two different concepts in the literature:*i)* complexity of the language which also may be referred to as theoretical complexity, and *ii)* complexity of the algorithm which is also referred to as practical complexity. In this section, we will discuss each of these concepts after recalling some basic notions about the theory of computational complexity.

2.2.1 Preliminaries on computational complexity

We assume that the reader is familiar with the *theory of computational complexity*, as defined in standard textbooks such as [GJ90, Pap03]. To avoid any ambiguity, we will recall some basic notions of this theory, which we will use throughout this thesis.

Informally, a *problem* is a general question to be answered, usually having several parameters. Thus, an *instance* of a problem is obtained by specifying particular values for all of its parameters. Standard complexity theory deals with decision problems, i.e. problems that admit a yes/no answer. In computer science, to solve a problem means to find an *algorithm* that solves all instances of the problem (assuming enough time, space, and other resources are provided). We must specify that an algorithm is a general, mechanical procedures that can be followed step-by-step to solve a problem. When there is only one possible action each step, then the algorithm is said to be *deterministic*, otherwise, when there is a finite many actions in each step, then the algorithm is called *non-deterministic*. The complexity (difficulty) of an algorithm is measured in terms of the amount of resources (time, space) required to solve the problem. On the other hand, the intrinsic complexity of a computational problem is measured by the worst-case complexity of the *best* algorithm that has been found to solve it so far. Hence, a problem is considered as *tractable* if there exists a polynomial algorithm to solve it. In contrast, a problem is *intractable* if it is so hard that no polynomial algorithm can possibly solve it. We can also distinguish some problem *complexity classes* as follows:

- **LogSpace** and **NLogSpace**: set of problems solvable in logarithmic space by a non-deterministic algorithm.
- **PTime**: set of problems solvable in polynomial time by a deterministic algorithms. These problems are considered tractable, i.e., solvable for large inputs.
- **NP**: set of problems solvable in polynomial time by a non-deterministic algorithm. These problems are believed intractable, i.e., unsolvable for large inputs. The best known algorithms actually require exponential time.
- **PSpace**: set of problems solvable in polynomial space by a deterministic algorithm. Polynomial space is "*not really easy*", since these problems may require exponential time.
- **ExpTime** set of problems solvable in exponential time by a deterministic algorithm. This is the first provably intractable complexity class.
- **NExpTime**: set of problems solvable in exponential time by a non-deterministic algorithm. N2ExpTime are just the problems that are solvable in double exponential time with respect to size of the input and by a non-deterministic algorithm. These problems are considered to be intractable and very hard.

We can also depict the general known relationships between the above described complexity classes as follows:

LogSpace $\subseteq$ **NLogSpace** $\subseteq$ **PTime** $\subseteq$ **NP** $\subseteq$ **PSpace** $\subseteq$ **ExpTime** $\subseteq$ **NExpTime** $\subseteq$ **N2ExpTime**

Given all these notions, the analytic analysis of algorithms could be established to study from a theoretical perspective the amount of resources that an algorithm would require to solve the computational problems, known that these problems are classified according to their intrinsic computing difficulty.

2.2.2 Theoretical complexity: complexity of the ontology language

In our context of research, ontologies would be regarded as instances of inference problems[1], and reasoners as the algorithms proposed to solve these instances. Extensive research has been carried out, aiming to establish the complexity class of reasoning over ontologies encoded in a certain DL. Actually, the DL languages vary in the complexity class they belong to, from low expressive languages whose standard reasoning tasks can be performed in polynomial time, such as the $\mathcal{AL}$ [BBM11], to highly expressive DLs, such as $\mathcal{SROIQ}$, whose the complexity class at the worst case is N2ExpTime [HKS06]. We could track more in details the complexity of the ontology language by looking into the different OWL profiles (c.f. Chapter 1, Subsection 1.3.2) and their underpinning DLs.

In the first iteration of OWL, three sub-languages were designed:

- **OWL 1 DL**: is based on the $\mathcal{SHOIN}$ (DL), whose key inference services have an NExpTime worst case complexity.
- **OWL Lite**: is based on the $\mathcal{SHIF}$ (DL), which benefits from *"more tractable"* inference services, the worst case complexity is being ExpTime.
- **OWL Full**: is a syntactic and semantic extension of RDF and RDF Schema (RDFS) and a superset of OWL DL for which inference services are *undecidable*. An ontology is in OWL Full if certain constraints usually imposed on axioms in order to retain the decidability property in OWL DL, are not met.

The second iteration of OWL, the OWL 2 [Gro09], also incorporates several profiles with distinct classes of complexity:

- **OWL 2 DL**: based on the $\mathcal{SROIQ}$(DL), with a worst case complexity of N2ExpTime for key inference services [HKS06].
- **OWL 2 EL**: based on the $\mathcal{EL}++$ (DL), for which the key reasoning tasks have a PTime worst case [BBM11].
- **OWL 2 RL**: a subset of OWL 2 DL and an extension of RDFS, the RL profile is motivated by

[1]An inference problem could be reduced to a decidable one, for instance DL classification is combination of a consecutive checking of concept satisfiability.

Description Logic Programs (DLP), and designed to allow for scalable reasoning based on rules. The standard reasoning in OWL 2 RL are PTIME-COMPLETE.

- **OWL 2 QL**: based on the $DL - Lite$ family of logics, in particular $DL - Lite_R$, it is meant to provide efficient query answering with a worst case complexity of NLOGSPACE and even LOGSPACE with respect to the size of the ABox.

It is obvious that the standard reasoning with ontologies written in EL, QL or RL is tractable and could be achieved in polynomial time. These restricted profiles remarkably simplify the reasoning complexity with OWL 2 ontologies and allow for scalable and fast implementations. However, we should note that any union of any two of the profiles is no longer light-weight, i.e. QL+RL, QL+EL, RL+EL are all EXPTIME-HARD [Krö12]. Figure 2.4 summarizes in an ascendant order the complexity classes of the main DLs and their corresponding OWL versions.

All of these inspection highlight the existing trade-off between the expressivity of the ontology language and the complexity of the standard reasoning services: the more expressive the ontology is, the more complex would be the reasoning. In overall, we would admit that from a theoretical perspective and with respect to the worst-case analysis, the ontology reasoning is highly complex for expressive DLs. Hence, we would wonder if in practice, it is possible to efficiently infer ontologies written in these DLs? and which would be the impact of the non-deterministic nature of DL based languages on the performances of the reasoning engines?

2.2.3 Empirical complexity: complexity of the reasoning engines

In a theoretical analysis, there is no need to introduce an effective procedure which solves an inference problem. The complexity classes are established by just analysing the description of the algorithms

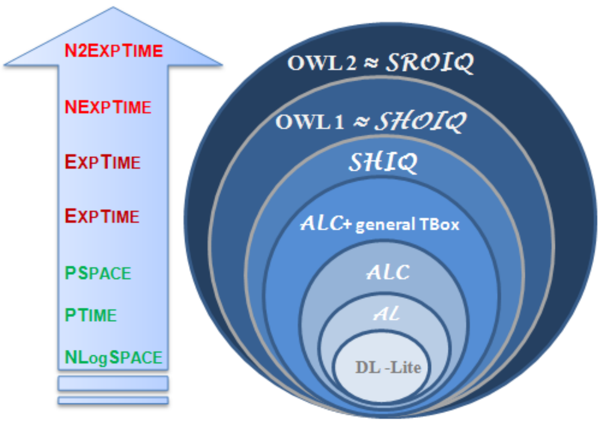
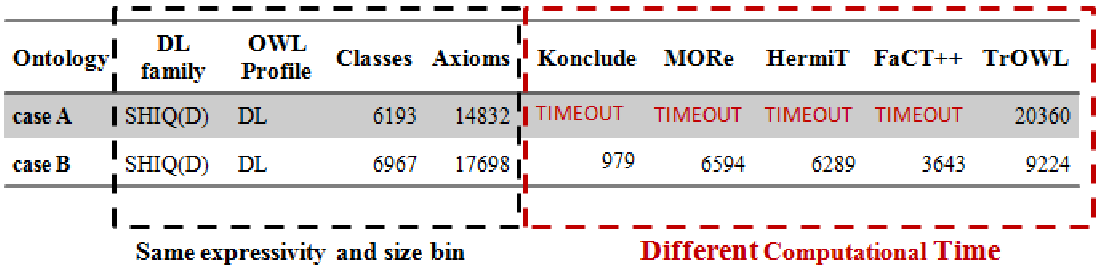


Figure 2.1: Summary of the theoretical complexity of some DL languages

and the generic features of the input instances without any actual running. In contrast, in a practical analysis, we determine the time and space complexity of a particular algorithm proposed to implement a proposed inference task. In fact, the theoretical complexity of a language is independent from the algorithm which decides it and assumes that the algorithm knows which branch will survive in the case of non-determinism. Hence, all the provided analytical results remains theoretical ones and can easily be disproved by experimental analysis. For this reason, we turned to the empirical, rather than the analytic, analysis. Indeed, various works [MRW11, Abb12] have settled that, in general, reasoning is feasible in practice and is far less complex than the established theoretical complexity. Interestingly enough, even with fairly expressive fragments of OWL 2, acceptable reasoning performances could be achieved. This would be owing to the considerable research and engineering efforts that have been devoted to design sophisticated reasoning algorithms with highly optimized implementations. In Chapter 1, we carried out a review scrutinizing these efforts. We described the most known reasoners and put into comparison their functional attributes. Nevertheless, the respective authors of [WLL+07, GPS12] have outlined that often in practice, reasoners tend to exhibit *unpredictable* behaviours when dealing with real world ontologies and their performances dramatically vary across the ontologies, even when the size or the expressivity of the latter ones are held constant. We have delineated two main aspects that would depict the *variability phenomenon of reasoner performances*: (*i*) the variability of reasoners runtime across the ontologies; (*ii*) the sparsity of reasoners computed inferences over the same input ontology. In the following, each of these issues are discussed more in details.

a. the variability of reasoners runtimes Gonçalves et al. [GPS12] have outlined three particular situations that a user could face when attempting to reason about an ontology: (*i*) for one test case ontology, switching the reasoner can degrade reasoning time from seconds to none termination; (*ii*) ontologies with the same size and expressivity would spend wildly different ranges of computational time on the same reasoner; (*iii*) a minimal change to an OWL ontology would increase or probably decrease reasoning time on one reasoner. As far as one of these situations happens, often no feedback is returned back to the user. Commonly, the latter one will keep shifting reasoners until finding the suitable one. Others would try to adjust their ontology hoping for an improvement, but running the risk of making the matters worse. In overall, it is well-accepted that there is no reasoning system that can outperform the other ones for all input ontologies. This could be observed by examining the reported results from the annual reasoners evaluation challenges, such as the ORE Evaluation Workshop [GBJR+13, BGJ+14, DGG+15]. Figure 2.2 depicts two test cases, that we have extracted from the publicly available results of the ORE 2014 DL classification challenge[2]. In each case, an ontology has been classified by various reasoners. The processing time was recorded, known that

[2]The ORE 2014 results are available at https://zenodo.org/record/11142/



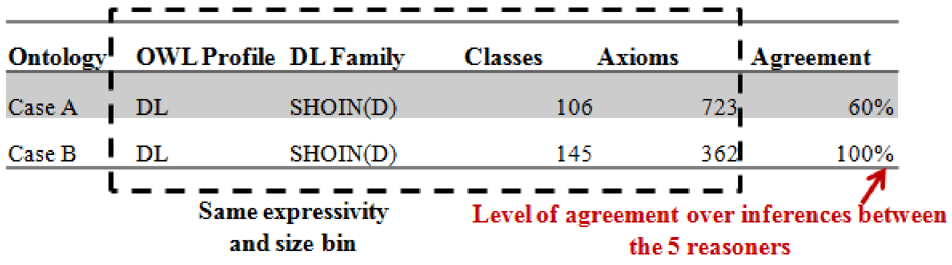
Ontology	DL family	OWL Profile	Classes	Axioms	Konclude	MORe	HermiT	FaCT++	TrOWL
case A	SHIQ(D)	DL	6193	14832	TIMEOUT	TIMEOUT	TIMEOUT	TIMEOUT	20360
case B	SHIQ(D)	DL	6967	17698	979	6594	6289	3643	9224

Same expressivity and size bin
Different Computational Time

Figure 2.2: Motivating example about the variability of reasoner runtime, extracted from the ORE 2014 ontology classification results [BGJ+14]. The time is in milliseconds.

the time limit is fixed to 3 minutes. When looking to these results, we would remark that the two ontologies have the same OWL profile, belong to the same DL family and their sizes are in the same bin, however the examined reasoners exhibited quiet distinct behaviours on these inputs. From a theoretical perspective, we would expect that the reasoning over these ontologies will not terminate in reasonable time, since the class complexity of $\mathcal{SHOIN}$ is N-ExpTime. However, the results proved the opposite particularly in the second case. Nevertheless, the runtime of these reasoners is quite variable for the same input ontology, e.g. in the second case a scatter up to 10 order of magnitude between the Konclude and the TrOWL respective runtime. We can also remark that, in the two test cases, TrOWL was the only reasoner to terminate the classification task within the time limit. This would suggest that this reasoner is the most performing one across the examined list. Nevertheless, TrOWL was the slowest one in the second test case, known that the two ontologies are similar ones. Hence, our first suggestion is refuted.

b. The sparsity of reasoners results Far less works have discussed this phenomena. This is because the formal basis of the core reasoner procedures are well understood and designers often provide theoretically proof of the soundness and the completeness of their reasoning algorithms. However, Gardiner et al[GTH06] and more recently Lee et al. [LMPS15] have pinpointed that reasoners could deliver quite distinct inferences or query answers, from one input ontology. They highlighted that this issue could hamper the development and seriously threaten the interoperability on the Semantic Web. Therefore, the reasoners computed inferences during the ORE evaluations were compared to each other, in order to identify the cases of disagreement and decide which reasoner results are likely to be the correct ones. Figure 2.3 depicts another snippet from the ORE 2014 DL classification results. Here, we checked the agreement level between the 5 reasoners that we previously listed in Figure 2.2. By agreement, we mean that the reasoners have delivered the same classification results when processing the test ontology. By inspecting Figure 2.3, we would remark that the level of agreement was different across the two ontologies, despite the fact that they belong to the same size and expressivity bins.



Ontology	OWL Profile	DL Family	Classes	Axioms	Agreement
Case A	DL	SHOIN(D)	106	723	60%
Case B	DL	SHOIN(D)	145	362	100%

Same expressivity and size bin
Level of agreement over inferences between the 5 reasoners

Figure 2.3: Motivating example about the sparsity of reasoner computed inferences, based on the ORE 2014 ontology classification challenge [BGJ+14].

Indeed in the first case, only 3 out of the 5 reasoners derived the same inferences, however in the second case all of the reasoners have computed the same results. Gardiner et al. [GTH06] argued that the disagreement may be due to the approximation mechanism that some reasoner designers would use to handle particular hard to handle DL constructors. Actually, reasoner designers often do not communicate further details about their approximation techniques to deal with such constructors. Lee et al. [LMPS15] also outlined this point. They highlighted that few details about many of the reasoning optimisations and engineering tricks are provided, and in common case, these tricks are only inspected by the developers. Given these findings, we would admit that the reliability of reasoner implementations is still a matter of investigations.

2.2.4 Discussion

Based this review, we can assert that although reasoning in practice is far less complex than the established theoretical computational complexity, reasoner empirical behaviours are still hardly predictable a priori due to the unexpected variability across the ontologies. Hence, new compiling challenges are raised, such as how would it be possible to anticipate this variability and explain its causes. In overall, we can notice that ontologies belonging to the same theoretical complexity class are not all equally hard. Obviously, it exists some ontology characteristics other than just size and expressivity which are likely to have an impact on reasoner performances. Hence, a reasoner implementation that exploits some particular features of the OWL language can perform on the corresponding class of ontologies better than the worst-case scenario. Indeed, several authors [GTH06, GPS12, LMPS15] explained the variability of reasoner performances by the fact that modern reasoning systems have complex design architecture and tend to implement a wide range of optimisation techniques that might interact and affect each other while processing some particular ontologies. Hence, a reasoner could be optimised for some, but not all the ontologies. On these basis, we would admit that a better understanding of ontology complexity factors that may trigger difficulties to reasoners is of a compelling need.

In our opinion, pointing out *what makes reasoning hard in practice* can guide the modelling process of ontologies as to avoid the reasoning performance degrading factors. Furthermore, existing ontologies may be revised towards efficient reasoning by detecting, and even repairing their critical components. On the other hand, when such knowledge is made available, the automatic selection of adequate reasoner of an input ontology could be achieved more easily and without any need for excessive trials and experimentations. In remainder of this chapter, we will review and discuss the existing solutions which have addressed the practical complexity of DL reasoners and provided solution to manage it.

2.3 Reasoner performance analysis: landscape of existing works

In this section, we tried to draw out the landscape of the state of art works, which addressed the issues related to reasoner performance variability phenomena and attempted to characterise the ontology hardness key features. These works are categorized according to the scope of their investigations and the nature of their solutions. We distinguished between two main categories: works that empirically evaluate the reasoner performances; and works that try to predict these performances. Below is a general description of these works and a consolidation of some of our comments and remarks.

2.3.1 Evaluation and profiling of reasoner empirical performances

We start our discussion by reviewing state-of-art works that assess the empirical performances of DL reasoners. We identified three groups of works, described in what follows:

A. Empirical evaluation and comparison Reasoner benchmarks [DCtTdK11, Abb12] and competitions [GBJR+13, BGJ+14, DGG+15] are annually held to compare the latest innovations in the field of semantic reasoning. The performances of ontology reasoners against well selected ontologies are evaluated mainly considering the computational runtime. The final purpose of these studies is to compare the reasoners in order to identify at average the most performing ones. Reasoners unpredictable behaviours and performance variability is often reported during these events, but no efforts were deployed to try to explain them. Commonly, the performances are recorded and made available to practitioners as a starting point for further analysis.

B. Reasoner profiling tools The variability of reasoner performances has motivated the development of various reasoner efficiency analysis tools like *Tweezers* [WP07], *Pellint* [LS08] and the *OCT tool* [Cha10]. They employ software profiling techniques to help inspecting the reasoner processing mode and to identify its potential pitfalls. To fulfil this task, *Tweezers* and *OCT tool* report particular performance statistics like the time required to accomplish a particular concept satisfiability test and the memory greediness. *Tweezers* provides further statistics such as the size of the completion graph constructed by a tableau based algorithm [BS01] and the number of iteration exhibited until finding a clash [Hor03]. On the other hand, *Pellint* inspects the ontology design without running the reasoner. This tool reports and even repairs some ontology modelling pitfalls, so-called ontology anti-patterns [RZ13]. The latter ones are suspected to brook down the reasoner efficiency and prevent it from scaling up. Unfortunately, both *Tweezers* and *Pellint* were designed with respect to one particular reasoner, i.e. Pellet [SPG+07]. Furthermore, the *OCT tool* was implemented to particularly check how reasoners could handle one particular ontology, i.e. the *GlycO* ontology[3]. This tool was not applied to other ontologies. Given these findings, it is obvious that the usability of the above described tools is limited. Generalizing their results could not be significant, since each of these tools was designed to handle a specific context (particular reasoner or ontology). Besides, the reasoning methodologies, programming languages and implementation details vary from one reasoner to another (c.f Chapter 1, Subsection 1.4.2), which makes it difficult to design a generic profiling tool that could analyse all the available reasoners and capture all of their pitfalls.

C. Reasoner correctness checking methods Another aspect of reasoner performance evaluation is to check the correctness of its results. A manual inspection of the computed inferences or query answers would be relatively easy for small ontologies, but usually infeasible for real world ones. On the other hand, there is no analytical assessment method of result correctness since both the completeness and the soundness of reasoning algorithms are often formally proven even before the reasoner is made available. Gardiner et al. [GTH06] claimed that the most straightforward way to automatically determine the correctness is by comparing answers derived by different reasoners for the same ontology. Consistency across multiple answers would imply a high probability of correctness, since the reasoners have been designed and implemented independently and using different algorithms. Accordingly, they proposed an automated reasoner empirical correctness checking method where reasoners outputs were assessed by a majority vote, i.e. the result returned by the most reasoners was considered as correct. This methods is part of the ORE evaluation framework [GBJR+13]. A more elaborated reasoner correctness checking method with a logical foundation was introduced by Lee et al. [LMPS15]. They also compare the various derived results by different reasoners for the same ontology in order to detect the disagreement. Then, they identify the axioms behind this disagreement and analyse them by computing their justifications. Their purpose is to check whether some implementation bug is causing the disagreement. The method partially relies on expert judgement when it is hard to decide which of the reasoners is the correct one. Although, authors admitted that even if their solution is more reliable

[3] `http://cobweb.cs.uga.edu/~jam/glyco/GlycO_v0_95.owl`

in assessing the reasoner correctness, they are still unable to recognize the source of each error in every reasoner evaluation. In the both solutions, the correctness of a reasoner given an input ontology is not anticipated or checked independently from the other reasoners. This is because, it is still not well understood how to decide that a computed class hierarchy or query answer is incorrect. Furthermore, it is yet not clear which of the reasoner optimisation or approximation techniques are badly affected by particular characteristics of the OWL ontologies.

2.3.2 Prediction of reasoner empirical performances

All the related works within this second category try to predict the reasoner performances to avoid the excessive cost of the empirical evaluations. However, they chose to employ quite distinct prediction approaches. We identified two main groups of these approaches, each of them is discussed in the upcoming paragraphs.

A. Heterogeneity based prediction Gonçalves et al. [GPS12] argued that it will be cost-efficient to predict whether an ontology is a reasoner performance bottleneck before performing any potentially expensive search for the ontology pitfalls that might not exist at the first place. They suggested that there are ontologies which are *performance homogeneous* and others *performance heterogeneous* ones. Their main claim is that whenever the measured runtime of some reasoner over reduced subsets of an ontology is proportional to the reported runtime over the whole input, then the ontology is *performance homogeneous* for that reasoner; otherwise it said to be *performance heterogeneous*. Gonçalves et al. asserted that reasoners would perform poorly on *performance heterogeneous* ontologies. They assumed that in the latter ontologies, it exist some sub-parts, i.e. modules, that are hard to manage by the reasoner. They called them the ontology *"hotspots"*. Authors explained that these hotspots are due to some combined effects of particular entanglements between axioms in the ontology. They also proposed a method to extract these hotspots, starting from those classes which are usually more time-consuming with respect to satisfiability test (SAT). However, their experiments have revealed that there is no precise co-relation between the reasoning time of a *hotspot* alone, and the reduction in reasoning time when such *hotspot* is removed. They affirmed that more investigations should be made about possible interactions between the hotspots and other ontology features.

B. Machine learning based prediction Another steam of works, basically described in [KLK12, SSB14, KLK14], used supervised machine learning (SML) techniques [Kot07] aiming at a priori predicting the computational time of a reasoner for a given ontology. Their predictive models take advantage from a large set of pre-computed ontological metrics. Kang et al. [KLK12] were the first to apply machine learning techniques to predict the ontology classification computational time carried by a specific reasoner. 27 metrics were computed for each ontology. These metrics were previously proposed by a work stressing on ontology design complexity [ZLT10]. The labels to be predicted were time intervals predefined by the authors. They carried out a comparison of the most efficient learning algorithms, then they selected Random Forest as the base learner. Runtime predictive models were trained for several state of art reasoners and showed good accuracy. Furthermore, they proposed an algorithm to compute the impact factors of the ontology metrics according to their effectiveness in the reasoner prediction process. Kang et al. have improved their approach, in a more recent work [KLK14]. They replaced time interval labels by concrete values of reasoner classification runtime and proposed additional metrics. Hence, they trained regression models using Random Forest. They also demonstrated the strengths of their predictive models by applying them to the problem of ontology *Hotspots* extraction. On the other hand, Sazonau et al. [SSB14] claimed that Kang's metrics based on graph translation of OWL ontologies are not effective. Thus, they proposed a different set of metrics and used Principal Component analysis (PCA) to reduce the dimensionality of the ontology feature vectors. Sazonau et al. have basically proposed a *local* prediction approach. The ontology corpus needed for the training is replaced by a set of ontology modules extracted from one single input ontology. Hence, a predictive model is built in for each couple (ontology, reasoner). The authors argued that their local approach is more suitable to anticipate the computational time of a reasoner over an ontology. However, we would wonder how practical is this approach given the rapid growth in the number of ontologies and reasoners. Lately, Kang et al. [KKL15] have investigated a further application of their reasoner runtime prediction models. They proposed to build a kind of meta-reasoner which assembles a set of state-of-art reasoners and automatically selects the most efficient one to handle an input ontology. The reported results are promising ones, however no intrinsic assessment have been carried out to prove the efficiency of this meta-reasoner. In particular, the computational cost of the prediction steps were not reported. Therefore, it is difficult to admit that a meta-reasoner which applies various prediction operations to choose a suitable reasoner then runs this selected one, could be faster than any single reasoner. Table 2.1 summarizes the main features of the afore described works.

2.3.3 Discussion

Based on this review of state, we assert that in overall there is a real lack of theory and tool support for both ontology novice and expert users helping analysing reasoner behaviours against case study ontologies. Indeed, the existing reasoner profiling tools are still in their first steps, needing more improvements to be able to handle all the available reasoners. On the other hand, we agree with Gonçalves et al. [GPS12], that ontologies making the reasoning particularly arduous, should be predicted be-

Work	**Target Class**	**Ontology Features**	**Dimensionality Reduction**	**Practical use of models**
[KLK12]	Time Bins (discrete)	27 metrics (Graph based)	6 feature selection Methods	ontology feature impact factor
[KLK14]	Runtime (continuous)	91 features (Graph and OWL based)	Inter-correlation removal	ontology hotspot extraction
[SSB14]	Time Bins (discrete)	57 features OWL based	Principal Component analysis	ontology hardness prediction
[KKL15]	Runtime (continuous)	91 features (Graph and OWL based)	Inter-correlation removal	meta-reasoner

Table 2.1: Summary of previous works about predicting ontology reasoners

fore investing any substantial resources in reasoner pitfall inspections. However, we do not approve their prediction method. We think that it would be more reasonable to rely on well-founded machine learning prediction solutions. We have outlined that the latter ones have witnessed their utility, when applied to reasoner runtime prediction. Indeed, we find that machine learning based approaches are of great interest to meet the demand of tracking down the reasoner performances variability phenomena. As a matter of fact, the learning methods are generic ones, they could handle any reasoner or any ontology. Besides, the predictive models have a broad scope of applications, previously enumerated in Table 2.1. These applications meet the goals that we fixed for this thesis. In particular, using machine learning techniques, we can:

- effectively predict the computing time of any reasoner over any ontology and thus, we can anticipate the variability of reasoner performances and pinpoint the ontologies that cause it;
- select the most efficient reasoner for any given ontology without being required to have expertise in DL reasoning or to carry costly evaluations each time we need to choose a suitable reasoner;
- ontology features which may have impact on reasoner performance could be pointed out.

Nevertheless, the existing ML based solutions for reasoner performance prediction should further be extended. More reasoner qualities must be examined, in particular the correctness of the reasoner. Yet, a reasoner that quickly but incorrectly processes an ontology is of little use. Moreover, we think that the learning could be engaged from a different perspective. Actually, it would be worthwhile to be able to predict how hard is an ontology instead of predicting how performing is a reasoner. Based on our study, we also noted that the potential success of the ML based solutions for reasoner prediction is closely depending on the quality of the employed ontology features. Unfortunately, our review of state of art confirmed that there is no known, automatic way of constructing suitable ontology feature sets. Hence, we believe that existing ontology features, those employed in the above discussed machine learning methods, must be revisited and extended. Integrating properties from distinct domain knowledge that appear likely to provide useful information about the design complexity of an ontology is one way to improve the existing feature sets. Finally, we should highlight that none of the above prediction methods was made available for use. The described techniques and the reported prediction results are still experimental ones. Authors of the ML based solutions for reasoner performance prediction didn't suggest any tool or discussed how their results would be put into practice to help experienced or novel ontology/reasoner users.

Motivated enough, we are convinced that machine learning based approaches could bring response to our research questions. However, existing reasoner prediction works need to be revisited and extended to meet all of our requirements. Though, we must admit that all of the discussed works are very recent ones and mainly carried out by one research group, i.e. Kang et al. [KLK12, KLK14, KKL15]. Hence, one may argue that there is not enough proof that machine learning solutions could be effective to cope with the issues derived from the reasoner performance variability phenomenon. For this reason, we decided to explore further research areas where algorithm performance prediction have known a success. In our opinion, this would provide us with a more in depth understanding of these techniques and open to us new perspectives for potential improvements of reasoner prediction solutions. In remainder of this section, we will scrutinize the learned lessons about the automatic performance prediction and selection of algorithms.

2.4 Algorithm empirical analysis

Over the years, a myriad of algorithms and heuristics have been developed to solve various problems and tasks. As a consequence, it becomes important for researchers and practitioners to discover and implement mechanisms that may determine which algorithm perform best on which problem instance. The No Free Lunch (NFL) Theorem established by Wolpert and Macready [WM97] states that it is impossible to build an algorithm that performs optimally in all cases. Different algorithms often perform better on different classes of problem instances. This issue has been pivotal in many domains, such as Machine Learning (ML) and Artificial Intelligence (AI), and known as the *algorithm prediction and selection problem.*

2.4.1 Empirical hardness and selection problems

Recently there has been a growth interest in experimental analysis of algorithms [McG12]. This is because more and more researchers have realized that theoretical results alone cannot fully characterize the algorithm performances and in particular, its empirical behaviours. This is specifically the case with randomized, non-deterministic and heuristic algorithms which are too complex for a detailed mathematical analysis [GJ90]. Yet, there exist a solid science of experimental algorithmics. Generally speaking, any algorithmic experiments involve running some algorithms on some problem instances, collecting the data, and analysing the results. Besides, any experiments have in advance some hypotheses or questions to test and ask. Indeed, the respective authors of [LBHHX14, HXHLB14] have concluded over a decade of research about the empirical hardness of NP-complete problems. They asserted that one of the main advantages of experimental algorithmic analysis is that it allows to investigate how the algorithm performances statistically depends on problem characteristics. This was the central hypothesis behind automatic algorithm selection, as it was originally designed by J. Rice [Ric76] in mid-1970s. He presented a formal abstract model made up from four essential components. The abstract model is reproduced in Figure 2.4. We would distinguish the following components:

1. the problem space $\mathcal{P}$ represents the set of instances of a problem class;
2. the feature space $\mathcal{F}$ contains measurable characteristics of the instances generated by a computational feature extraction process applied to $\mathcal{P}$;
3. the algorithm space $\mathcal{A}$ is the set of all considered algorithms for tackling the problem;
4. the performance space $\mathcal{Y}$ represents the mapping of each algorithm to a set of performance metrics.

Formally, given a problem instance $x \in \mathcal{P}$, with features $f(x) \in \mathcal{F}$, find the selection mapping $S(f(x))$ into algorithm space $\mathcal{A}$, such that the selected algorithm $\alpha \in \mathcal{A}$ maximizes the performance mapping $y(\alpha(x)) \in \mathcal{Y}$. Of course, the choice of features depends very much on the problem domain as well as the chosen algorithms. Features must be selected, so that the varying complexities of the problem instances are exposed, any known structural properties of the problems are captured, and any known advantages and limitations of the different algorithms are covered. Clearly, it is very challenging and complex matter to design suitable features for a given problem domain, and this has likely been a significant obstacle to the widespread application of the Rice's model. Rice has also outlined the hardness of approximating the selection mapping function S. He discussed several methods, including linear, piece-wise linear, and polynomial approximations. He concluded, however, that "*the determination of the proper non-linear form is still somewhat of an art and there is no algorithm for making the choice*" [Ric76]. In fact, the choice of the best algorithm for selecting the mapping function S is, ironically, another algorithm selection problem.

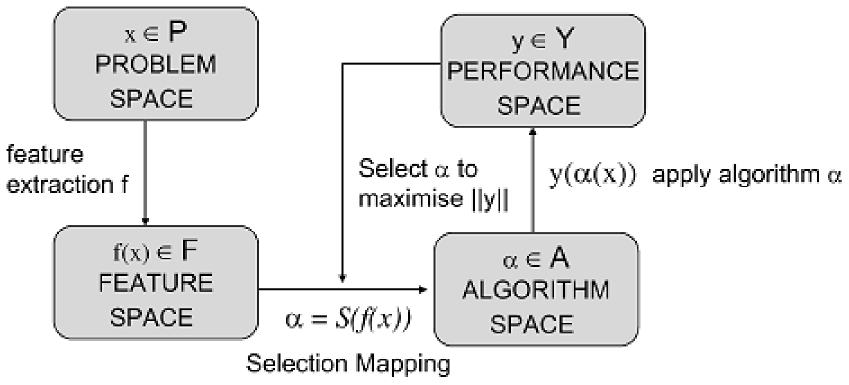


Figure 2.4: Schematic diagram of the [Ric76]'s algorithm selection problem. The objective is to determine the selection mapping $\mathcal{S}$ that returns the best algorithm α

2.4.2 ML based approaches for algorithm analysis and selection

In the last decades and in different areas such as ML and AI, the Rice's model was put into practice by employing supervised machine learning methods as the mapping function, S, between the features of problem instances and the performance of an algorithm. Hence, by using the S function of each candidate algorithm, the performances of these ones could be predicted on unseen instances allowing best algorithm selection. This approach was undertook in different ways and to achieve other purposes than automatic algorithm selection. In the following, we will review the pioneering works which have opted for this approach. Then, we will discuss the lessons that could be learned from them and highlight the ones transferable to our class of problems, i.e. DL based ontology reasoning.

2.4.2.1 The meta-learning approach

In the last 20 years, machine learning research was faced with an increasing number of available learning algorithms including a multitude of parametrisation, preprocessing and post-processing approaches [Abb14]. In Chapter 1, we have already reviewed some of the ML classification algorithms and listed their most known categories. Of course, these algorithms are not equally performing in terms of prediction accuracy and confidence level across the training datasets. Hence, the selection of the most adequate one for a new problem is a critical and difficult task to achieve in particular for non-expert user. Meta-learning [BGCSV08] is a specialized form of machine learning that is able to learn about the learning process itself to increase the quality of the predictive results. Meta Learning has emerged from the need to support data mining automation in issues related to algorithm and parameter selection. Recently, Lemke et al. [LBG15] have conducted a survey about the current trends in the meta-learning research field. The authors assert that meta-learning can provide an invaluable help

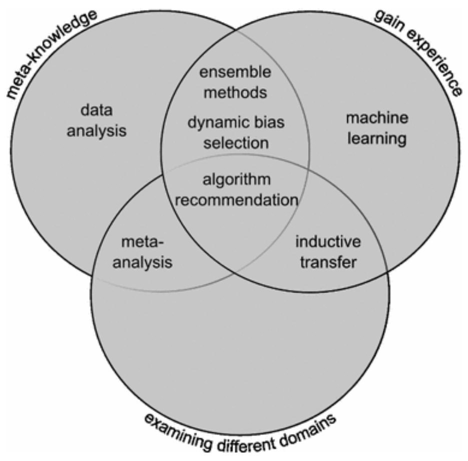


Figure 2.5: Notions and components of a Meta-learning system according to Lemke et al. [LBG15]

avoiding extensive trial and error procedures for algorithm selection, and brute force searches for suitable parametrisation. The performances of a learning algorithm could stand for the prediction quality of its derived model, or the runtime required to build this model or even a combination of both of them. Roughly speaking in meta-learning, the Rice's mapping function is acquired from meta-examples that store: *i)* the features describing the datasets (problems) and *ii)* performance information obtained by executing candidate learning algorithms on datasets. After generation of meta-examples, a meta-learner (a particular supervised learning algorithm) is applied to acquire knowledge that relates performance of candidate algorithms to the features of the datasets (problems). Meta-learning can be employed in a variety of settings. Figure 2.5 depicts most of these settings and the basic traits of a meta-learning system as it was seen by Lemke et al. [LBG15]. These authors suggested that ensemble learning methods and combinations of base-learners such as bagging, boosting and stacking, which we have briefly describe in Chapter 1, Subsection 1.5.2.7, are kinds of meta-learning. Another interesting application is the selection of the optimal parameter values of an algorithm. In this special case, versions of the same algorithm with different parameter settings are considered as different candidate algorithms and hence, the meta-learner will try to predict which version is the most efficient with respect to the input dataset. However from an end-user perspective, algorithm recommendation is one of the most interesting application of meta-learning, with the final goal of providing advise about the utmost promising learning algorithms to handle a user dataset. Recommendations are suggested in the form of algorithm ranking. The latter one could be acquired by sorting the predicted performances of individual algorithms or directly computed using an established ranking model training by preference learning algorithms [SP13].

2.4.2.2 The algorithm portfolios approach

NP-Complet problems [Coo71] such as propositional satisfiability (SAT), travelling salesperson (TSP), mixed integer programming (MIP) and combinatorial problems (CP) are at the core of the artificial intelligence. The AI community has achieved great success in designing high-performance algorithms capable to solve these hard problems, despite their worst-case complexity. However, it was observed that in practice these algorithms often exhibit extreme runtime variation across problem instances, even when the problem size is held constant. Gomes et al. [GSCK00] have established that one solver could perform better at tackling some problem instances from a given class, but is substantially worse on other instances. They have also admitted that there is little theoretical explanation of this variation. Hence, practitioners are faced with a challenging algorithm selection problem.

Measuring different algorithms' performances on a given problem distribution and then using only the algorithm that had the lowest average runtime has been a popular answer to this issue. This approach is commonly known as the "*winner-take-all*". However, it has resulted in the neglect of many interesting algorithms that, while uncompetitive on average, offer excellent performance on particular classes of problem instances. Later on, it has been shown that we can exploit the complementarity of the solvers by combining them into *an algorithm portfolio* [GS01]. This latter term was used by C. P Gomes and B. Selman [GS01] to describe a strategy for running a set of algorithms in parallel. Algorithm portfolio has since also been employed to describe any strategy that combines multiple, independent algorithms to solve a single problem instance. Such portfolio may include methods that pick a single solver for each problem instance [OHH+08, XHHLB08], methods that makes online decisions to switch between solvers [SM07], and methods that execute multiple solvers independently per instance, either in parallel, sequentially, or partly sequential/parallel [GS01]. The main purpose is that the portfolio should outperform all the single algorithms it is made from. A prime example of algorithm portfolio for SAT problems is SATZilla [XHHLB08]. This system has won several annual SAT competitions by employing empirical hardness predictive models [LBHHX14] to be able to decide the most efficient solver to apply. Indeed, a common trait among the proposed portfolio strategies is that they employ ML regression or classification methods to build an efficient predictor of a solver performances. More recently, Oentaryo et al. [OHL15] claimed that, while showing successes to some extent, the contemporary regression- or classification-based algorithm selection methods are not designed to directly capture the notion of *preferability* among different solvers for a given problem instance. Therefore, they suggested to pose the algorithm selection problem as a ranking issue. Their main purpose is to predict the appropriate (unique) top K best ordering of solvers for a given problem instance, while employing a unique predictive model, called the *ranker*. To realize this, Oentaryo et al. proposed a Ranking-based Algorithm Selection (RAS) methodology, following preference learning

Purpose	**Criteria**	**ML Techniques**	**Practical use**
Meta-learning	- Runtime - Prediction quality - Combined criteria	- Classification - Regression - Preference learning	- Algorithm recommendation - Algorithm parameter tuning. - Ensemble learning. - Dynamic bias selection.
Algorithm portfolio	- Runtime	- Classification - Regression - Preference learning	- Algorithm portfolio construction. - Algorithm parameter tuning. - Generation of benchmarks of hard instances. - Identify the hardness key features of the problem instances.

Table 2.2: Attribute summary of the machine learning based approaches for algorithm analysis and selection.

principals [FH10].

2.4.3 Discussion

Table 2.2 summarizes the basic functional attributes and the practical use of the above described approaches. Through this review, it seems clearly that by employing statistical methods and more specifically supervised machine learning methods, we could benefit from a more nuanced understanding of algorithm empirical behaviours than by analytic, worst-case analysis. This confirms our previous conclusions when studying the ontology reasoners. We argued that the empirical analysis of reasoners is more informative that the theoretical one and we particularly highlighted the valuable benefit of applying supervised machine learning to the problem of reasoner performance analysis. Indeed, thanks to the ML solutions, it has been shown that the performances of algorithms from different nature (ML or AI ones) could be predicted at a high degree of confidence based on easy to compute features of problem instances. Hence, various practical algorithmic problems such as automatic algorithm selection or recommendation or optimal parameter tuning have efficiently been solved. This makes the machine learning based approach for algorithm analysis and selection important even for theoretically inclined computer scientists who prefer proofs to experimental findings.

All of these inspections strengthens our conviction that employing machine learning based approaches for algorithm performances prediction and selection would bring response to our research questions. Some lessons could be drawn from this literature review, that in our opinion, are of major importance to achieve a successful prediction and selection of ontology reasoners.

- One of the biggest practical challenges in any algorithmic experiments is assembling the required test instances. The best test instances are probably those real world instances that are taken from real world applications. This is also the case when assessing the performances of reasoners. A representative population of ontologies should be carefully selected, to avoid that a reasoner would process only ontologies, it is particularly optimized for;
- The selection of ontology features relies heavily on domain knowledge. Our review of related works showed that there is no agreement about the sources of the ontology hardness against the reasoning tasks and no common definition of a good ontology feature. In overall, a feature should be easy to compute (usually negligible compared to the reasoning time) and most relevant to the performance of a particular reasoner or a particular class of ontologies;
- Running time and solution quality are two most common measures of algorithm performance. In our case, we will try to predict the reasoning runtime and the correctness of the derived results. However, some experimental constraints should be fixed to ensure the to ensure reproducibility of the results. Moreover, a particular reasoning task, e.g. ontology classification or consistency checking should be fixed as the main subject of performance prediction;
- There is no *best* supervised machine learning algorithm for the prediction purpose. Hence, different learning algorithms should be investigated, to identify the one capable to train highly accurate models given the available data about the reasoner performances.
- In general, any reasoner can be used as a candidate for a selection or recommendation task. But in practice, we can use domain knowledge to wipe out reasoners that are obviously inferior to other ones;
- The choice of the reasoner selection techniques is closely depending on the final purpose and the general application context. Some techniques are time-sensitive ones, which make them appropriate to be applied in algorithm portfolios. Others are more informative and would be suitable for end-user advising system.

2.5 Discussion Summary

We recall that we are looking to provide solutions to end-users to gain a better understanding of reasoner empirical behaviours, and ontology hardness factors. Thanks to this review, we draw some important conclusions that we can list as follows:

- Although the analytical analysis methods provide the basis to characterize the complexity of ontologies and the amount of resources needed by the reasoners, it is of little use to meet the demand of tracking down the empirical behaviours of reasoners and to automatically select the best available one for any given ontology. This is because the theoretical worst case complexity does not necessarily unveil reasoner real-world performances.

- The most challenging problem when dealing with reasoners is the variability of their performances across the ontologies. This variability could affect the reasoner efficiency as well as the correctness of its results. This phenomena is largely due to the uncertainty over reasoner implementation factors and the potential interaction between optimisation techniques and some particular ontology features.
- There is a real lack of theory and tool support helping both ontology novice and expert users to figure out which particular aspects in the ontology that are damaging the reasoners performances. In overall, it is obvious that performance analysis of reasoners against ontologies is still a fastidious, time consuming task mainly based on manual checks.
- Machine learning-based techniques applied to empirical data describing reasoner past running can be used to accurately predict its runtime over any ontology. Thus, it would make it possible to cope with the reasoner performances variability phenomena. The techniques can also help to build systems to automatically select the most suitable reasoner according to the characteristics of the input ontology. Given its various benefits, we chose supervised machine learning as the kernel theory to guide the design of our solutions.
- Machine learning approaches of algorithm performance prediction and selection have known a great success when applied to hard problems in different fields such as AI and ML. They can track different performance criterion and can help gaining insights over the instance key hardness features. Algorithm predictive models are employable in various practical applications. Hence, different lessons could be learnt from these works, most of them are transferable to our research context.

Given these findings, we can set up the main stages that we will follow to achieve the goals of this thesis:

1. We will start our research work by studying the ontology attributes that we can employ as input features to train predictive models of reasoner performances. Existing features will be examined, revisited and potentially extended.
2. We are willing to advance state-of-art works in reasoner performance prediction. Our purpose is not only to predict the reasoner runtime, but also its correctness. We are also planning to introduce a new prediction task which aims to adjudge whether a given ontology is hard to manage by reasoners. To achieve these goals, we will introduce a generic learning framework that can train distinct predictive models related to the ontology reasoning context.
3. We will approach the issue of the automatic selection of ontology reasoners. In contrast to the

related work where a reasoner portfolio was proposed [KKL15], we are willing to employ meta-learning to introduce an automatic ranking method of reasoners. In our opinion, recommending a set of reasoners ranked according to their expected performances over a given ontology is a more informative solution for users seeking assistance.

4. We will try to help an end user to gain insights about the hardness factors in its ontology: first, by highlighting the key ontology features likely to be reasoner performance degrading factors; then, by helping him extracting the ontology subsets that are potentially the most compelling ones for the reasoning tasks.

As previously highlighted in the introduction of this thesis, our solutions are meant to be useful for end-users whether they are experienced or not, regarding the ontology reasoning. Therefore, one important goal of this thesis is to provide an interactive tool which allow users to benefit from our proposals.

2.6 Conclusion

In this chapter, we have reviewed researches in various areas that are closely related to our research issues. We recall that we are looking to provide solutions to end-users to gain a better understanding of reasoner empirical behaviours, and more specifically, to gain insights about its ontology hardness factors. We showed that formal complexity analysis of reasoner performances seems hopeless in bringing solutions to our research interrogation. For this reason, we turned to empirical, rather than analytic, analysis. Based on our review, we argued that from an empirical perspective, the most arduous problem with DL reasoners is the *performance variability phenomenon*. We examined related works which has introduced solutions to cope with this challenging issue. We came to the conclusion that machine learning and data mining techniques can help us gain more insights and knowledge on algorithm's empirical behaviour which analytical methods can not provide. Thus, we chose these techniques as the kernel theory to guide the design of our solutions. At the end of this chapter, we set up the key elements in this thesis roadmap and the contribution we are willing to achieve.

In the next chapter, we will start establishing our machine learning based approach for reasoner performance prediction, by introducing a novel collection of ontology features.

CHAPTER 3

Ontology Profiling

Contents

3.1 Introduction

The main goal of this thesis is to track down to variability of DL reasoner empirical performances through a rigorous application of the supervized learning approach. The latter one entails learning a mapping between a set of features describing the reasoner input, i.e. the ontology, and an output variable representing the reasoner performance to be predicted. Obviously, choosing good features is crucial to the construction of good predictive models. In chapter 2, we highlighted the lack of automatic way of constructing good feature sets to outline the computational complexity of OWL ontologies. Indeed, researchers must use domain knowledge to identify properties of the instance that appear likely to provide useful information.

Given these findings in this chapter, we will be looking to determine the primary features that would reveal the design complexity of OWL ontologies regarding the reasoning tasks. To accomplish this purpose, we conducted a large investigation about existing works proposing metrics to evaluate the design quality of an ontology [GCCL06, TAM+05, LNJ+10, ZLT10] or discussing the sources of reasoning complexity in Description Logics [Don03, THPS07, BCM+10]. Our focus was on the ontology expressivity richness from a knowledge representation perspective. Our investigations have lead us to propose a novel collection comprising 116 ontology features. The latter ones are in their basic form, qualitative and quantitative statistics outlining a broad range of structural and syntactic ontology design attributes.

In the remainder of this chapter, we introduce our ontology feature suite. Then, we describe the ontology corpus selected for the different scheduled experiments throughout this thesis. Finally, we put into discuss the computational cost of the main sets of features, thus the most efficiently computable ones could be outlined.

3.2 Modelling the Ontology Design Complexity

The first step in our research work aims to identify domain independent relevant features, likely to outline the *empirical hardness* of an ontology against the reasoning tasks. To achieve this purpose, we are planning to profile the design complexity of an ontology using easy to understand and efficiently computable metrics. Our focus will be on OWL ontologies. Therefore, the modelling of our features is mainly guided by the specification of the OWL language, previously provided in Chapter 1, Section 1.3. More specifically, we established a meta-model of this language in order to ease the understanding of its internal structure and to highlight the primary components of its grammar. This meta-model is depicted in Figure 3.1. Throughout this figure, we can distinguish the main entities of the OWL

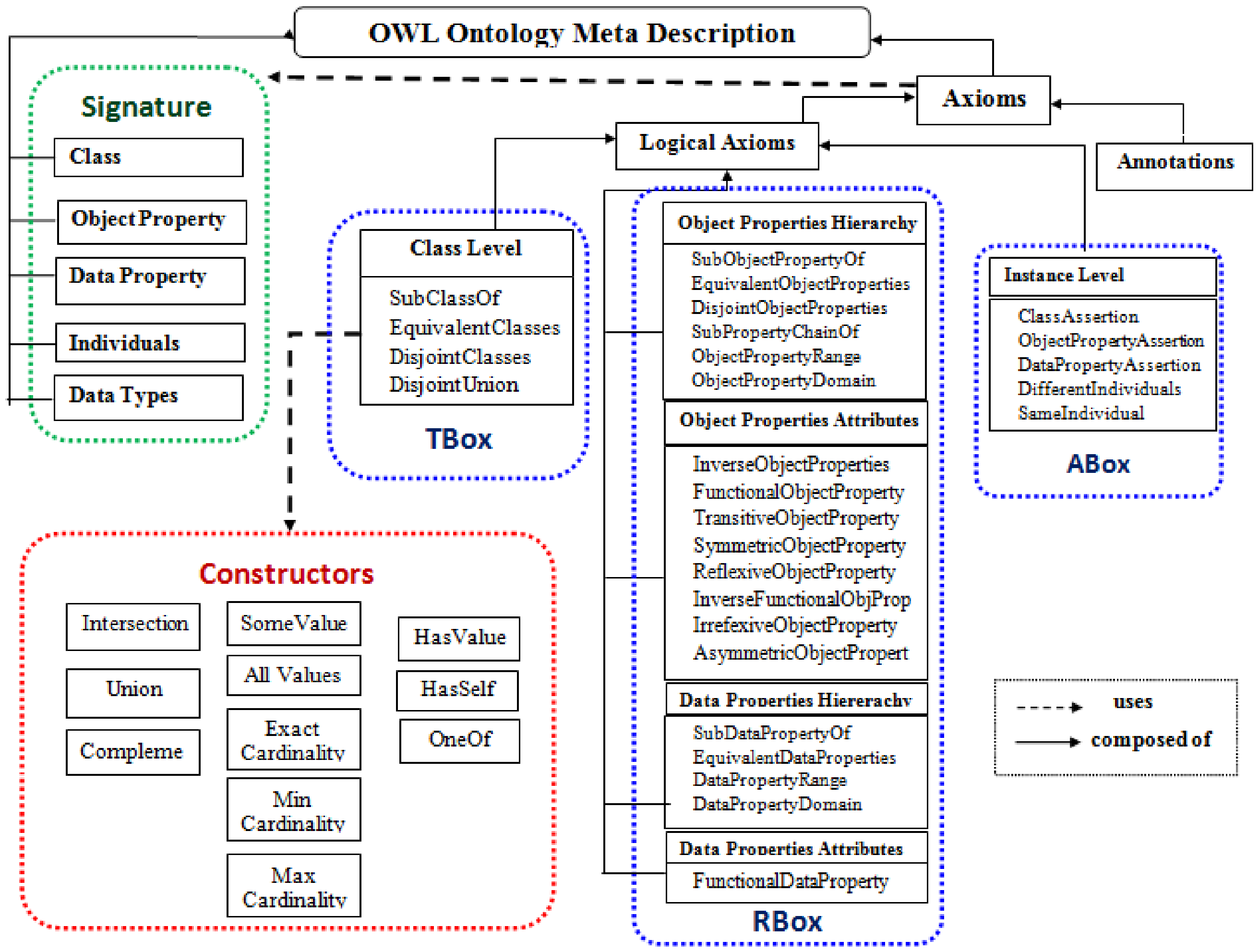


Figure 3.1: Meta Description of The OWL Language

language and their interdependencies. We can also identify some concepts of the Description Logics terminology highlighted in dashed boxes, together with their corresponding OWL entities.

We designed 116 ontology features that cover different structural and syntactic attributes of an OWL ontology. We should highlight that some of them are already widely recognized ontology quality evaluation metrics. In overall, we organized our ontology features suite into 4 main categories described in the following:

1. **Ontology size description** is a group of metrics characterizing the size of the ontology considering both the amount of its terms and axioms.
2. **Ontology expressivity description** is based on two main features, namely the OWL profile[1] and the DL family name.

[1]For further details about OWL 2 profiles, the reader is kindly referred to http://www.w3.org/TR/owl2-profiles/.

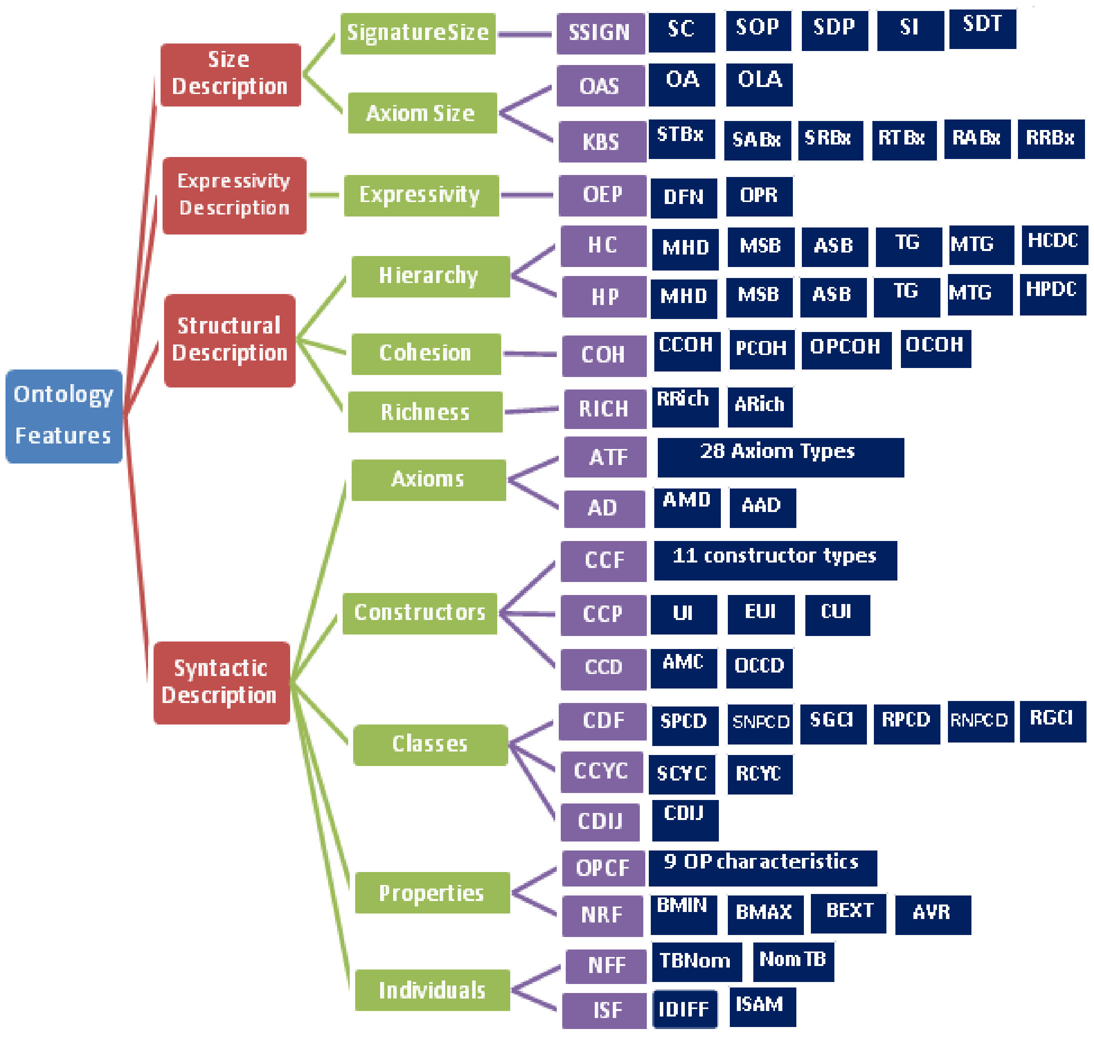


Figure 3.2: The Taxonomy of the Ontology Feature Suite

3. **Ontology structural description** gathers various metrics commonly used by the ontology quality evaluation community [LNJ+10] to characterize the taxonomic structure of an ontology.
4. **Ontology syntactic description** is a set of metrics that we proposed in order to quantify some of the general theoretical knowledge about DL complexity sources [BCM+10].

We further refined these categories by splitting them up into more explicit sub-categories. One or more set of features are within each sub-category. The full taxonomy of the proposed features is depicted in Figure 3.3, where the leaf boxes (coloured in dark blue) stand for the final features. The direct ancestors of these boxes represent the different feature sets (purple color), which in turn belong to the sub-categories (green color). It is important to note that we avoided the translation of the OWL ontology into any specific graph representation, except for its initially defined hierarchies of named classed or named properties. The latter ones have native tree-like structure. The rational behind this choice is twofold: i) as previously asserted by Sazonau et al. [SSB14], there is no general agreement about the way an OWL ontology should be translated into graph; ii) the translation operation is time and memory consuming one, in particular for large scale ontologies. As consequence, we discarded state-of-art ontology evaluation metrics which are graph based ones and we reformulated others so that it could be directly computed from the OWL representation of the ontology.

In the remainder of this chapter, we will provide a formal description of each proposed ontology feature and expose the rational behind its proposal. The upcoming descriptions were partly introduced in our previous work [AYL15b].

3.2.1 Ontology Size Description

Features in this category are intended to characterize the size of the ontology.

Definition 1 (Class Size) *Given an OWL ontology signature $\widetilde{\mathcal{O}} = \langle N_C, N_R, N_I \rangle$, the class size, denoted by* ***SC****, stands the number of names nc_i, which uniquely identify classes in the ontology, $nc_i \in N_C$ and $SC{=}|N_C|$.*

Analogously to Definition 1, we propose to compute the number of user-defined object and data property names, respectively denoted by **SOP** and and **SDP**. Similarly, we suggest to record **SI**, which stands for the size of the ontology in terms of named individuals. Finally, **SDT** represents the number of data types[2]. The set of features made up from SC, SOP, SDP, SI and SDT is denoted by **SSIGN** which depicts the size of the ontology signature.

Rationale Using the above features, the amount of terms defined in a given ontology is quantified. It is worth noting that not all of the modern DL reasoners have support for data types. An increasing number of the latter ones would compromise their performances and even causes their crash while processing the ontology.

Definition 2 (Axiom Size) *The ontology size in terms of axioms is characterized by two values: the number of OWL axioms qualified as logical ones, designed by* ***OLA****; and the total number of axioms, denoted by* ***OA****. Hence, we will be denoting by* ***OAS*** *the set made up from OLA and OA.*

Definition 3 (KB Size) *The size of the knowledge base, denoted by* KBS, *is a sub-set of features intended to characterize the amount of axioms in each knowledge base sub-part. Thus, KBS includes* ***STBx****,* ***SRBx*** *and* ***SABx****, which stand respectively to the number of axioms belonging either to TBox,*

[2]These are RDF literals or simple types defined in accordance with XML Schema datatypes.

RBox or ABox. Furthermore, KBS involves ***RTBx***, ***RRBx*** *and* ***RABx***, *as the respective size ratio of each KB sub-part w.r.t. the total number of logical axioms (OLA).*

Rationale Commonly known, reasoners only deal with axioms belonging to the KB's sub-parts *TBox*, *RBox* or *ABox*. Annotations are simply ignored when processing an ontology for a reasoning task. Furthermore, it is well accepted that some reasoners are particularly optimised to operate on the concept level (TBox, RBox), while others are more efficient in managing the ontology assertional sub-part (ABox). Therefore, it is important to capture the knowledge size of each of these KB components in order to anticipate the computational cost of the reasoning services.

3.2.2 Ontology Expressivity Description

The term *expressivity* could be confusing in the ontology field. In fact, it has different meaning depending on the context of its use [OAC+07]. For instance, expressivity in the field of ontology quality assessment is used to describe the ontology knowledge richness `w.r.t.` the domain under conceptualization. However, expressivity at the knowledge representation (KR) field is often used to characterize the grammar richness of the KR language according to which the ontology has been formulated. In this subsection, we put our focus on the second definition of the ontology expressivity. We retained two main qualitative features commonly used to represent the expressivity of the ontology authoring language.

Definition 4 (Ontology Expressivity Profile (*OEP*)) *is a set which includes the OWL profile name of the ontology, denoted by* ***OPR*** *and the Description Logic family name underpinning the ontology language grammar, denoted by* ***DFN***.

There are four formal OWL 2 profiles[3] namely DL, EL, QL and RL. However, in some particular cases, the language constructs used in the ontology may violate rules of each of these profiles. In this case, we denote by PNAN the profile of the ontology. On the other hand, we tag by PFULL the ontology that matches all the OWL profiles.

Rationale OWL 2 and more precisely OWL 2 DL is a highly expressive language. Key reasoning tasks like consistency checking have an extremely high worst case complexity: N2ExpTime-Complete [HKS06], i.e., *intractable*. However, this complexity could be tractable with less expressive fragments of OWL 2, i.e. EL, QL and RL profiles. Nevertheless, not all the reasoners are intended to support

[3]For further details about OWL 2 profiles, the reader is kindly referred to http://www.w3.org/TR/owl2-profiles/.

the entirety of OWL 2 DL. Some of them, such as ELK[4], are very efficient in handling the OWL 2 EL profile and less performing with OWL 2 DL profile. Therefore, the OWL profile name (OPR) is another important feature that would allow us to discriminate the ability of the reasoners. On the other hand, the DL family name (DFN) is a more explicit denomination of the ontology DL constructs set. For instance, an ontology could be $\mathcal{AL}$ or $\mathcal{SH}$ or $\mathcal{SHOIN}$, etc. Typically, the theoretical computational complexity of the reasoning tasks is defined with respect to the DL category (DFN) of the ontology language (c.f. Chapter 2, Section 2.2) .

3.2.3 Ontology Structural Description

Particular attention was paid to the characterization of the taxonomic structure of an ontology, also known as the inheritance hierarchy. The latter has a tree like shape and is built from subsumption relationships between named classes $A \sqsubseteq B$ or named properties $R \sqsubseteq S$. We remind that when classifying an ontology, the reasoner intends to infer implicit subsumption from the explicitly defined ones. Hence, a reasoner will try to scan most of the explicit subsumption relations in an efficient way. Therefore, the more the inheritance hierarchy is complex, the more computational time would be required. In this category, we gathered various features that have been previously introduced in literature by the ontology quality evaluation community [GCCL06, TAM+05, LNJ+10] and introduced new ones. The following three subcategories describe the essence of the retained features.

3.2.3.1 Hierarchical features

We build both concept hierarchy denoted by **HC** and property hierarchy denoted by **HP**. Then for each hierarchy, we measured the following features:

Definition 5 (Inheritance depth) *The maximal depth of the class hierarchy (**HC_MHD**) stands for the longest path between a named class C and the root of the class inheritance hierarchy.*

Worth noting, we consider a unique root corresponding to `owl:Thing` in any class hierarchy. If more than one already exist in the ontology, a fake root will be added joining the actual ones. Following Definition 5, we propose to compute the maximal depth of the property hierarchy (**HP_MHD**).

Rationale The inheritance depth (**HC_MHD**) was identified by LePendu et al. [LNJ+10] as a possible reasoning complexity source. It was also proposed by Zhang et al. [ZLT10] as an ontology design complexity metric and called *DIT*. The rationale is that great values of HC_MHD would indicate

[4]The reasoner is available at `https://www.cs.ox.ac.uk/isg/tools/ELK/`

more computational cost will be required to process classes at the bottom of the inheritance tree. This is because such classes will be affected by changes in any of their ancestors.

Definition 6 (Direct descendant) *It is characterized by two values: the maximal number of direct subclasses, denoted by **HC_MSB**; and the average number of direct subclasses, denoted by **HC_ASB**.*

Similarly, we propose to compute **HP_MSB** and **HP_ASB**, respectively the maximal and the average sub-properties in the inheritance property hierarchy.

Rationale The value of the direct descendant **HC_MSB** represents the utmost case where a given named class C is used by other classes. A change to C will induce consequential impact to all the dependent classes. **HC_ASB** was also used by Tartir et al. [TAM+05] and called the *Inheritance Richness* of the ontology. The authors claimed that low values of this feature would indicate that the ontology is a vertical one with high degree of knowledge specification. However, lower values would describe a shallow horizontal ontology having less detailed knowledge.

Definition 7 (Direct ancestors) *It is known as the* tangledness *of the class hierarchy and matches the number of named classes having more than one direct ancestor. By **HC_TG** we will be denoting this number. Furthermore, by **HC_MTG** we will be denoting the maximal reported number of direct ancestors among the HC classes.*

Analogously, we suggest to characterize the tangledness of the property hierarchy by respectively measuring **HP_TG** and **HP_MTG**.

Rationale The *tangledness* concept was introduced by Gangemi et al. [GCCL06] in order to outline the impact of working with the multiple inheritance design pattern when modelling the class hierarchy in an ontology. Tangledness was also called *tree impurity* by Zhang et al. [ZLT10]. They tried to outline the impact of deviating from the tree-like structure of the HC design.

Seeking a closer inspection of the class inheritance design structure, we suggest to compute the Freeman's graph centralisation measure based on node degree [Fre78]. The latter one is a measure of common use in graph and semantic network analysis. We recall that the degree of a graph node is equal to number of edges incident to this node. In the case of class hierarchy, a node could be viewed as a named class and its degree is equal to the sum of its direct ancestors and direct descendants. According to this specification and the Freeman's definition, we can suggest to compute the *class hierarchy degree centralisation* as follows:

Definition 8 (HC Degree Centralisation) *Let C_d be the function computing the degree centrality of any class in HC such as $\forall c_i \in HC$, $C_d(c_i) = degree(c_i)$ and let c^* be the class having the highest degree centrality in HC, thus the* class hierarchy degree centralisation *denoted by $HCDC$ is equal to:*

$$HCDC = \frac{\sum_{i=1}^{N}(C_d(c^*) - C_d(c_i))}{(N-1)(N-2)} \tag{3.1}$$

where N is the number of classes in the inheritance hierarchy, i.e. $N = |HC|$.

The HCDC index ranges between 0 and 1. In a similar way, we suggest to compute the *property hierarchy degree centralisation*, that we denote by $HPDC$

Rationale HCDC is intended to show at what extent is there a small number of highly central classes in HC. This index reaches its maximum value of 1 when one single class is semantically connected with all of the remaining $(N-1)$ classes, while the other classes are connected only with this central class. On that ground, we believe that in highly centralized HC, reasoning procedures affecting central classes, will have impact on larger number of classes than expected to be. Thus, these procedures would require much more computing time.

3.2.3.2 Cohesion features

The literature provides a plethora of various metrics to design the *cohesion* in the ontology. These metrics helps to determine the degree of *relatedness* between the ontology classes or properties. To apprehend this feature, we opted for the cohesion metrics introduced by Faezeh et al. [FW10]. The latter authors have proposed four measures of cohesion, covering different levels of knowledge in the ontology. For the seek of brevity, we did not report the different formulas of these metrics, yet a short description of each of them is given in the following:

- the class and property cohesion, respectively designed by **CCOH** and **PCOH**. These are fractions computed based on the number of direct and indirect hierarchical links joining one node, i.e. named class or property, and all of its descendants.
- the object property cohesion denoted by **OPCOH**. This feature is computed using the number of classes which have been associated through the particular object property domain and range.
- the ontology cohesion denoted by **OCOH**. It is simply a weighted aggregation of the previously defined cohesion metrics, i.e. **CCOH**, **PCOH** and the **OPCOH**.

3.2.3.3 Schema richness features

We further improved the ontology structural category by two additional features. The latter are well known for the ontology evaluation community, since they are part of the *OntoQA* tool [TAM+05].

- the ontology *relationship richness* (**RRich**). This feature outlines the diversity of relations in ontology. Formally, it is the ratio of the number of relations between classes that are not hierarchical w.r.t. total number of different types of the ontology relationships. We slightly modified the initial formula of this metric described in [TAM+05], in order to be able to compute it without any translation of the ontology into a graph.
- the ontology *attribute richness* (**ARich**). It is defined as the average number of attributes per class. To compute this metric, we considered a named OWL data property as an attribute relative to the named classes specified in its `rdfs:domain`.

3.2.4 Ontology Syntactic Description

Features within this category are intended to quantify some of the general theoretical knowledge about the complexity sources in description logics [Don03]. To accomplish this purpose, we conducted an investigation about main reasoning algorithms [BS01, MSH09] and the most known reasoning optimization techniques [Hor03, THPS07]. According to the respective authors of [HB91, Don03], we would distinguish three basic sources of inefficiency in reasoning with Description Logics. Namely, these are *(i)* the high degree of non-determinism introduced by the use of GCIs and more specifically disjunction ($\sqcup$) constructor, or negation associated to conjunction ($\sqcap$) *(ii)* the construction of large models due to the existential restriction ($\exists$), or *(iii)* interactions from different natures between these constructors. More in details, the authors suggested that the worst case might occurs when nominals interact with inverse roles and quantified number restriction, in particular, the max cardinality restriction. Actually, there is a general agreement that each of the above mentioned class constructor, alone, is challenging to reason with and needs special optimization techniques.

All of the above explanations are general ones, lacking precise definitions of the nature of the interactions, or the specification of boundaries to be respected when using the hard DLs constructors in the ontology knowledge modelling. Therefore, we will be looking to translate these inspections into quantitative metrics that could be computed and empirically analysed. We proposed 6 sub-categories of features, each covering specific aspects of some of the OWL entities. Detailed description of these features are provided in the upcoming subsections.

3.2.4.1 Axiom level features

Features within this subcategory attempt to characterize the relevance of particular OWL axioms.

Definition 9 (Axiom type frequencies) *It is a set, denoted by* ***ATF****, where each element is equal to the ratio between the occurrences of one particular type of OWL axioms and the total number of the logical axioms in the ontology.*

Rationale Values within the ATF set capture statistical information about every OWL axiom type[5] in order to examine whether the high frequency of some axiom types would increase the computational cost of the reasoning.

Definition 10 (Axiom nesting) *The parsing depth of an axiom is equal to the number of levels of nested expressions constructing this axiom. The axiom nesting of an ontology is characterized using two values: the highest and the average value of parsing depth of all of its logical axioms, respectively designed by* ***AMD*** *and* ***AAD****.*

Rationale **AMD** and **AAD** are features of common use, adopted also by Sazonau et al [KLK14]. The nesting nature of axioms is likely to increase the probability of entanglements across the axioms and thus it would raise the ontology reasoning computational cost. **AAD** gives insights about the worst case, where an axiom is nested. However, **AMD** would indicate at average how the axioms in the ontology are nested. As an example of a nesting axiom, we can take the next expression which has already been listed in [BCM+10]: $C \equiv Person \sqcap (\leq 1 hasChild \sqcup (\geq 3 hasChild \sqcap \exists hasChild.Female))$

3.2.4.2 Constructor level features

In previous reasoner prediction works [KLK12, KLK14], authors simply counted axioms that involve potentially complex class constructors. However, they missed that one constructor could be involved more than once in the same axiom. Moreover, we believe that we can categorize the ontologies according to how often constructs are employed to express complex classes and by characterizing the concept of *ontology construct density*. In order to achieve this purpose, we suggest three features that we describe in the following:

Definition 11 (Class Constructor Frequency) *Denoted by* ***CCF****, it is a set of 22 values, where*

[5] ATF is computed for each of the 28 OWL logical axiom specified in http://www.w3.org/TR/owl2-syntax/

each element stands for the frequency of a specific class constructor[6]*, by considering its occurrences in a single axiom than across all the ontology axioms.*

Formally, given a constructor cc_i belonging to the set of all DL constructors ($cc_i \in CC$), the **Frequency**(cc_i) stands for the ratio of the cc_i total occurrences in each *TBox* axiom, i.e. $\forall A_j^T \in \mathcal{T}$, to the sum of all constructors occurrences. The $Frequency(cc_i)$ value ranges within the unit interval.

$$Frequency(cc_i) = \frac{\sum_{j=1}^{|\mathcal{T}|} Count(cc_i, A_j^T)}{\sum_{i=1}^{|CC|} \sum_{j=1}^{|\mathcal{T}|} Count(cc_i, A_{tx_j})} \tag{3.2}$$

Rationale The expressiveness of an ontology language is often defined in terms of the class and property constructors admitted in this language. In the remainder about DL basic notions (Chapter 1, Section1.3.1), we explained that different combinations of constructors can give rise to DL languages with different computational properties. For instance, in Tableau algorithm [BS01], a different expansion rule corresponds to each kind of constructs with a different computational cost. Therefore, using the CCF feature we could outline how often we are using any of the class constructors and hence, the ones having the highest impact on the reasoning time could be identified.

Furthermore, we proposed to characterize the ontologies that are *dense* in terms of class constructors, otherwise, having their number of employed constructors close to the maximal possible number.

Definition 12 (Ontology Class Constructors Density) *Denoted by* ***OCCD****, it stands for the ratio of the sum of occurrences of all the existing constructors, to the maximal possible number of constructors in the ontology. The latter is defined as the multiplication result of the total number of* TBox *axioms* $|\mathcal{T}|$*, by the maximal counted number of constructors in a single* TBox *axiom.*

Let **AMC** be the maximal possible number of constructors in the ontology, then the ontology class constructors density (OCCD) is computed as follows:

$$AMC = max(\sum_{j=1}^{|CC|} Count(cc_j, A_j^T), \forall A_j^T \in \mathcal{T})$$

$$OCCD = \frac{\sum_{i=1}^{|\mathcal{T}|} \sum_{j=1}^{|CC|} Count(cc_j, A_j^T)}{|\mathcal{T}| \times AMC}. \tag{3.3}$$

[6]Here we refer to all of the available class constructors in OWL 2 DL such as conjunction (∪), disjunction (∩), negation (¬), quantification (∃, ∀), etc.

Rationale The value of OCCD index ranges within the unit interval $[0, 1]$. OCCD reaches its maximum when all TBox axioms have the same number of constructors whatever are their types. We believe that dense ontologies in terms of class constructors are likely to have larger number of entanglements amongst the axioms which would increase its reasoning time. Both OCCD and AMC will be computed for every ontology.

More invaluable, we suggest to examine the interactions between constructors within single axioms. This would be achieved by tracking down some constructor combinations commonly used to model complex descriptions of classes. We called these combinations *the constructors coupling patterns* and denoted them by *CCP*. The set of patterns within CCP are described in the following. We will be recording the occurrences of each pattern.

Definition 13 (Constructors Coupling Patterns) *Denoted by* ***CCP****, it is a set of three modelling patterns characterizing particular interaction of class constructors within single axioms. Namely, the patterns are IU, EUvI and CUvI defined as follows:*

- ***IU*** (Intersection, union Pattern)*: is reported when a conjunction (`owl:intersectionOf`) of class expressions in an axiom is part of a disjunction (`owl:unionOf`) of class expressions in the same axiom, and vice versa. This pattern can manifest in one of the following forms:*

$$\sqcap \quad (\ldots, \sqcup(C_1, C_2, \ldots), \ldots) \tag{3.4}$$

$$\sqcup \quad (\ldots, \sqcap(C_1, C_2, \ldots), \ldots) \tag{3.5}$$

- ***EUvI*** (Existential, universal having intersection Pattern)*: is defined by a conjunction of class expressions, that concurrently contains an existential restriction and a universal restriction associated to the same role r. This pattern can be manifested by one of the following forms:*

$$\sqcap \quad (\ldots, \exists r.C, \forall r.D, \ldots) \tag{3.6}$$

$$C_1 \sqsubseteq \exists r.C \quad \textit{and} \quad C_1 \sqsubseteq \forall r.D \tag{3.7}$$

- ***CUvI*** (Cardinality, universal having intersection Pattern)*: is a particular case of the EUvI pattern, where existential restriction is replaced by some cardinality restriction form ($\leq nr.C$, $\geq nr.C$, $= nr.C$).*

Rationale The motivation behind this proposition comes from lectures about ontology anti-patterns [RZ13]. The anti-patterns were put forward to highlight the bad practices in ontology modelling. In our case, we focused on those anti-patterns that would increase the computational cost of reasoning when Tableau-like [BS01] algorithms are employed. Actually, while checking a satisfiability of a concept by the later algorithm, expansion rules are recursively applied in order to build a completion graph, called the *model*. Each class constructor has its own expansion rule. Applied in a specific order, the rules may lead to a sharp increase in the size of the Tableau algorithm completion graph, and hence the reasoning cost. The above specified patterns are intended to describe some fragments of axioms where the order and the interaction between constructs would be hard to handle.

3.2.4.3 Class level features

Classes in the ontology could be named or specified via complex expressions. In this subcategory, we will highlight different methods to define classes and track their impact in the ontology *TBox* part. According to [Hor03], concept definition axioms are primitive ones **PCD** of the form $A \sqsubseteq D$, or non primitive ones **NPCD** of the form $A \equiv D$, where A is an atomic concept name. However, real-world KBs commonly contain general definition axioms **GCI**. These axioms are of the form $C \sqsubseteq D$ or $C \equiv D$, where both C and D are complex class descriptions involving class constructors. **GCI** axioms are known to be costly to reason with, due to the high degree of non-determinism that they introduce. Thus, optimization techniques, mainly *Absorption*, is commonly used to reduce the number of **GCI** in the ontology. *Absorption* is widely implemented in DL reasoners, and designers are often looking to increase its optimization power by employing different approaches. Motivated enough, we propose to compute the following values:

Definition 14 (Class Definition Features) *It is a set of values, denoted by* ***CDF****, which intend to characterize the dominant form of class definition in the ontology. CDF includes SPCD, SNPCD and SGCI, which stand respectively to the number of axiom being either PCD, NPCD or GCI. Similarly, CDF involves RPCD, RNPCD and RGCI, which respectively denote the ratio of SPCD, SNPCD and SGCI, w.r.t the total number of logical axioms in the ontology.*

In DL, a cyclic definition axiom is the one that references the same (or equivalent) classes (or properties) on both sides of the subsumption relation (i.e. $\exists R.C \sqsubseteq C$, or $R \circ R \sqsubseteq R$). Such an axiom may be explicit of inferred by a reasoner. We only computed the explicit ones by implementing the method described by Baader et al. [BBM11].

Definition 15 (Cyclic Class Features) *It is a set, denoted by* ***CCYC****, which involves two values: the number of cyclic class definitions, designed by* ***SCYC*** *and the ratio of these definitions w.r.t. total number of named classes, designed by* ***RCYC*** *.*

Rationale A naïve application of the Tableau rules [BS01] does not terminate if the TBox contains cycles. To ensure termination is such cases, various reasoning optimization techniques were proposed such as the *blocking* technique [Hor03, THPS07]. The CCYC features were suggested in order to characterize to which extent applying these techniques is required.

Definition 16 (Class Disjointness Feature) *This is the ratio of named classes declared as disjoin ones, w.r.t. the total number of classes. By **CDIJ**, we will be denoting this feature.*

Rationale Our motivation to compute this feature is based on stated observations by Wang et al. [WP07]. The latter ones highlighted that class disjointness can greatly reduce or eventually increase the reasoning computational time. This would depend on the "*right amount*" of disjointness statements to be put in an ontology. However, we still ignore how to set out this value. The class disjointness is also examined by the *Pellint* tool [LS08], and reported as a bottleneck when the number of statements exceeds some predefined threshold.

3.2.4.4 Property level features

In this subsection, we propose some features relative to the ontology object properties (OP). In OWL, an OP could be defined as *transitive*, *reflexive*, *functional*, etc., using a single axiom from the RBox. For instance, when specifying an object property as transitive, we only need to use the (`owl:TransitiveObjectProperty`) axiom. However, this object property could be repeatedly involved in many other *TBox* axioms and even more than once in one axiom. Consequently, reasoning techniques dealing with transitivity would be applied as far as this transitive object property is used. Given this finding, we propose to outline the frequency of each particular object property characteristic.

Definition 17 *The* object property characteristic frequency ***(OPCF)*** *is a set of 9 values, each of them is recording the frequency of particular object property characteristic.*

Let OPC be the set of all object property characteristics. The frequency of every element in OPC is computed within two stages. First, we start by collecting all the named object properties defined with the particular characteristic $C_i^{op} \in OPC$. We designed the latter set as $S(C_i^{op}) = \{OP_j, j \geq 0\}$. Then, we sum up the occurrences of each $OP_j \in S(C_i^{op})$ and denoted it $OPCO(C_i^{op})$. Thus, the frequency of C_i^{op} is equal to the fraction of $OPCO(C_i^{op})$ by the sum of $OPCO$ values computed for every characteristic $C_i^{op} \in OPC$.

$$OPCO(C_i^{OP}) = \sum_{j=1}^{|S(C_i^{op})|} \sum_{k=1}^{|\mathcal{T}|} Count(OP_j, A_{tx_k}) \tag{3.8}$$

$$Frequency(C_i^{OP}) = \frac{OPCO(C_i^{OP})}{\sum_{j=1}^{|OPC|} OPCO(C_j^{OP})} \tag{3.9}$$

The $Frequency(C_i^{OP})$ is ranging in the unit interval $[0, 1]$.

Rationale The respective authors of [Hor03, THPS07] have emphasized on the hardness of managing some of object property characteristics, since they damage the effectiveness of reasoning optimization techniques[7]. By computing the $OPCF$ set of values, we would be able to identify the thresholds from which the presence of some OP characteristics would limit the reasoner performances. Furthermore, it would allow to flag out which of the characteristic have the highest impact on the reasoning procedure.

Furthermore, we suggest to inspect the effect of using high values when defining object properties number restrictions. Otherwise, we are interested in the upper bound of values used in the qualified number restrictions: at-least, at-most and exact. For instance, taking a restriction of the form $\geq nR.C$, we will be recording $max(n)$ of all restrictions having the same form. Accordingly, we define the following sub feature set:

Definition 18 (Number Restriction Features) *Denoted by* ***NRF****, it is a set composed of the biggest cardinality values employed in qualified number restrictions, i.e.* ***BMAX****,* ***BMIN****,* ***BEXT****, and the average value of all the numbers occurring amongst the different types of cardinality restrictions, denoted by (****AVC****).*

Rationale The qualified number restrictions are expressive DL constructors which equips the language with the ability of counting. However, they can dramatically increase the complexity of the standard reasoning algorithms. In [HB91], authors admitted that the complexity of reasoning is evidently a function of the numbers occurring in the qualified number restrictions, i.e. m, n in $(\leq mR.C)$, $(\geq nR.C)$ or $(= nR.C)$. By increasing m or/and n, the size of the completion graph and the number of non-deterministic choices computed by a Tableau-like algorithm would grows tremendously. The NRF feature is designed in order to characterize the impact of this increase.

3.2.4.5 Individual level features

In this subcategory, we specify some of the interesting characteristics of named individuals that would be declared in the ontology. When recalling the basic notion of DLs (Chapter 1, Subsection 1.3.1), we mentioned that named individuals used in *TBox* axioms to define new classes are called nominals.

[7]For instance, *Internalisation* and *Caching* are less effective at the presence of inverse properties in the ontology

Definition 19 (Nominal Frequency Features) *Denoted by* ***NFF****, it is a set of two scores. The first score records the ratio of nominals w.r.t. the individuals size (SI) and denoted by* ***NomTB****. However, the second score indicates the ratio of axioms having nominals w.r.t. the* TBox *size and it is designed by* ***TBNom****.*

Rationale Modelling with nominals would come at a price. As previously pinpointed by Tsarkov et al [THPS07], nominals break the advantageous tree model property of TBoxes and tableau algorithms can no longer decide a TBox consistency test by constructing a tree-like model. Thus, more time-consuming reasoning procedures are deployed to handle the nominal semantics. However, we ignore from which threshold using the nominals would damage the reasoner performances. Features within the NFF set are designed to capture this impact, so that the exact thresholds would be figured out.

Analogously to the above defined CDIJ feature, we propose to examine the effect of missing or probably over specifying the similarities/dissimilarity across the named individuals.

Definition 20 (Individual Similarity Features) *Denoted by* ***ISF****, it is a two feature set composed of* ***IDIFF****, the ratio of named individuals specified as different ones, and* ***ISAM****, the ratio of named individuals declared as equal ones (`owl:sameAS`).*

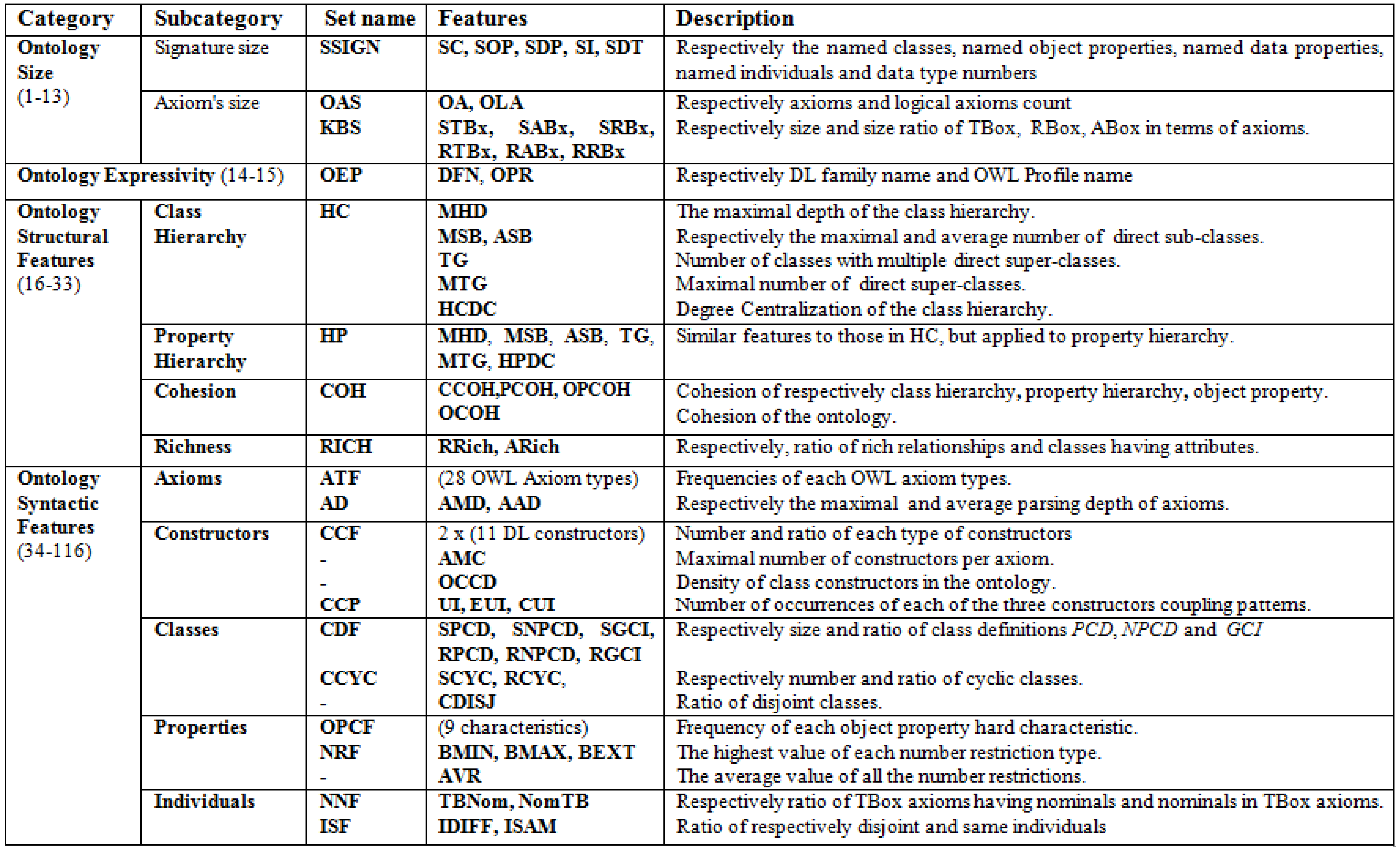
3.3 Computational Cost of the Ontology Features

To facilitate the automated computation of the proposed features, we have developed a Java metric tool, we called the *ontology profiler*. The latter one is mainly based on the OWL API (see Chapter 1, Subsection 1.3.3). It traverses each ontology, collects and stores the calculated metrics. Worth to note, we also wrote SPARQL queries in order to compute the constructors coupling patterns (CCP) (Subsection 3.2.4.2). Hence, we employed the Jena API in order to query the ontologies looking for these patterns.

In the remainder of this section, we describe the ontology corpus we will be using throughout this thesis, some ontologies will be profiled using our tool, then we discuss to computational cost of each proposed ontology feature in order to identify the most efficiently computable ones.

3.3.1 Description of the ontology corpus

We remind that the main goal of this thesis is to learn to predict DL reasoner empirical behaviours in order to provide assistance to novice and expert users when looking for guidance. Our predictive



Category	Subcategory	Set name	Features	Description
Ontology Size (1-13)	Signature size	**SSIGN**	**SC, SOP, SDP, SI, SDT**	Respectively the named classes, named object properties, named data properties, named individuals and data type numbers
	Axiom's size	**OAS**	**OA, OLA**	Respectively axioms and logical axioms count
		KBS	**STBx, SABx, SRBx, RTBx, RABx, RRBx**	Respectively size and size ratio of TBox, RBox, ABox in terms of axioms.
Ontology Expressivity (14-15)		**OEP**	**DFN, OPR**	Respectively DL family name and OWL Profile name
Ontology Structural Features (16-33)	**Class Hierarchy**	**HC**	**MHD**	The maximal depth of the class hierarchy.
			MSB, ASB	Respectively the maximal and average number of direct sub-classes.
			TG	Number of classes with multiple direct super-classes.
			MTG	Maximal number of direct super-classes.
			HCDC	Degree Centralization of the class hierarchy.
	Property Hierarchy	**HP**	**MHD, MSB, ASB, TG, MTG, HPDC**	Similar features to those in HC, but applied to property hierarchy.
	Cohesion	**COH**	**CCOH,PCOH, OPCOH**	Cohesion of respectively class hierarchy, property hierarchy, object property.
			OCOH	Cohesion of the ontology.
	Richness	**RICH**	**RRich, ARich**	Respectively, ratio of rich relationships and classes having attributes.
Ontology Syntactic Features (34-116)	**Axioms**	**ATF**	(28 OWL Axiom types)	Frequencies of each OWL axiom types.
		AD	**AMD, AAD**	Respectively the maximal and average parsing depth of axioms.
	Constructors	**CCF**	2 x (11 DL constructors)	Number and ratio of each type of constructors
		-	**AMC**	Maximal number of constructors per axiom.
		-	**OCCD**	Density of class constructors in the ontology.
		CCP	**UI, EUI, CUI**	Number of occurrences of each of the three constructors coupling patterns.
	Classes	**CDF**	**SPCD, SNPCD, SGCI, RPCD, RNPCD, RGCI**	Respectively size and ratio of class definitions *PCD*, *NPCD* and *GCI*
		CCYC	**SCYC, RCYC,**	Respectively number and ratio of cyclic classes.
		-	**CDISJ**	Ratio of disjoint classes.
	Properties	**OPCF**	(9 characteristics)	Frequency of each object property hard characteristic.
		NRF	**BMIN, BMAX, BEXT**	The highest value of each number restriction type.
		-	**AVR**	The average value of all the number restrictions.
	Individuals	**NNF**	**TBNom, NomTB**	Respectively ratio of TBox axioms having nominals and nominals in TBox axioms.
		ISF	**IDIFF, ISAM**	Ratio of respectively disjoint and same individuals

Figure 3.3: Description summary of the introduced ontology feature suite

approach is mainly based on modelling the empirical hardness of ontologies using efficiently computable features. Therefore, to ensure the good quality of the predictive results, special attention should be paid to the identification and selection of an ontology population, suitably large and representative of DL problems that are likely to occur in real word Semantic Web applications. Thus, ontologies would be treated as random factors when conducting the learning process. As previously highlighted by Sazonau et al. [SSB14] predictions trained from a biased ontology corpus, towards easy to compute ontologies would cause misleading generalizations. Hence, our primary requirement at this stage of the study is to assemble an ontology corpus, which is highly diversified in terms of ontology characteristics, in order to reduce the probability for a given reasoner to encounter only problems it is particularly optimised for. To accomplish this purpose, we chose to work with ORE Corpora [MBP13] that we describe in the following:

ORE corpora The Ontology Reasoner Evaluation Workshop, ORE [GBJR$^+$13, BGJ$^+$14, DGG$^+$15], is an annually organized event. The latter one includes a competition in which OWL reasoners are faced with different reasoning tasks: classification, consistency checking, and realization over OWL EL and OWL DL ontologies. The tasks are performed on several large corpora of real-life OWL ontologies, obtained from the web, as well as user-submitted ontologies which were found to be challenging for reasoners. In overall, the ORE 2014 ontology corpora contains 16,555 unique ontologies[8]. The set comprises:

- the MOWLCorp[9] (Manchester OWL Corpus) which was obtained through a Web Crawl and user submitted ontologies;
- the Oxford Ontology Library[10];
- a BioPortal[11] Snapshot (June 2014);
- user submitted ontologies such as BioKB, DMOP, FHKB, USDA, DPC, genomic-CDS, City-Bench, and so on (detail in [MBP13]).

Local ontology corpus We chose 2500 ontologies from the ORE corpora. This selection stands for our local sampling of OWL ontologies, denoted by $\mathcal{C}(O) = \{O_1, \ldots, O_n\}$. Within the latter set, almost 167 distinct DL families are covered, ontologies fall into various OWL 2 profiles and were binned according to their sizes[12]. We can distinguish between small $(0, 100]$, medium $(1000, 10000]$, large

[8]The full set of ORE ontologies is available at http://zenodo.org/record/10791

[9]The MOWLCorp is available at http://mowlrepo.cs.manchester.ac.uk/datasets/mowlcorp/

[10]The library is available at http://www.cs.ox.ac.uk/isg/ontologies/

[11]BioPortal is accessible via https://bioportal.bioontology.org/

[12]The size is computed in terms of the number of the logical axioms

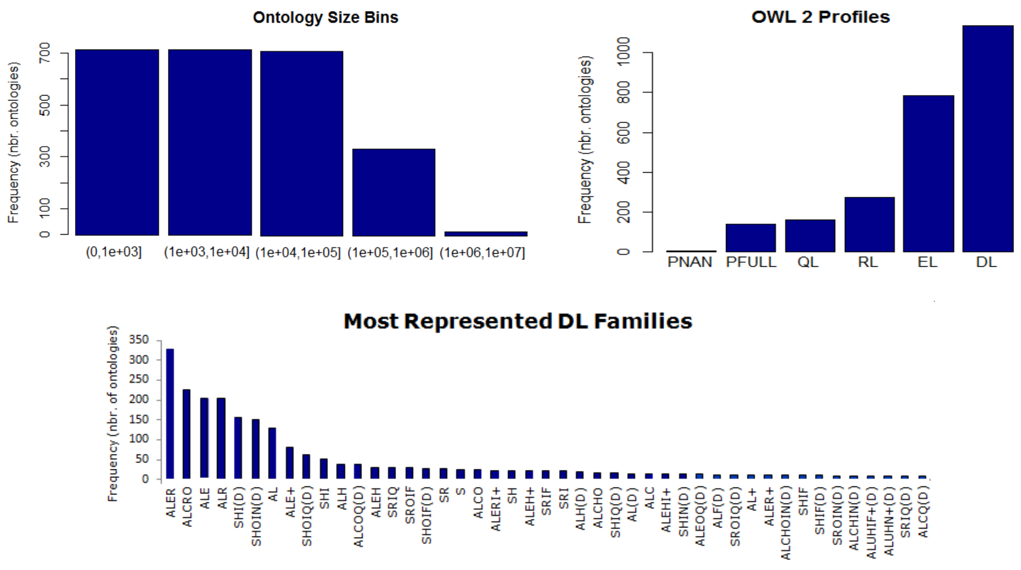


Figure 3.4: Ontology distribution over size bins, OWL 2 profiles and the most represented DL families.

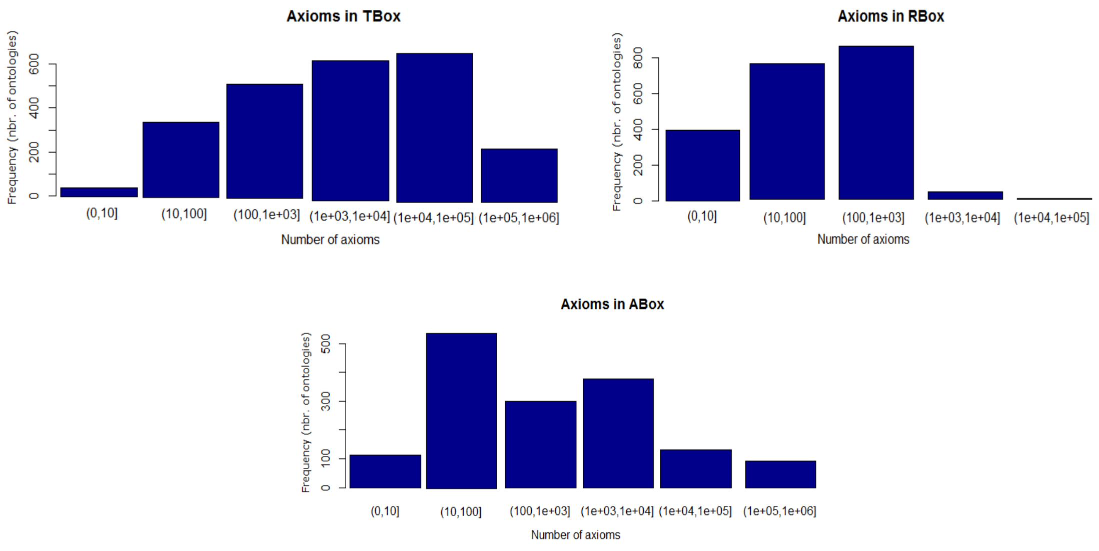


Figure 3.5: Ontology distribution over axiom bins within TBox, RBox and ABox

$(10000, 100000]$ and very large > 100000 ontologies, knowing that the largest ontology in the corpus includes 3 182 261 logical axioms. Figure 3.4 depicts the ontology distribution over the size bins, then over OWL 2 profiles[13]. It also illustrates the number of ontologies within the most represented DL families, i.e. the ones characterizing more than 7 ontologies in the corpus. Figure 3.5 gives a more detailed description of ontologies, where the number of axioms falling in TBox, RBox and ABox are depicted. Throughout these figures, we would observe that our ontology sampling is representative of the different ranges of OWL ontologies and it is richly diversified in size and expressivity.

3.3.2 Discussing the computational cost

Throughout this chapter, we described our ontology feature. Figure 3.3 summarizes this work and highlights the general taxonomy according to which the 116 proposed features are organized. Hereafter, we assume that it is important to get an idea about the computational cost to get values of each of these features. We remind that our final purpose is to employ them in prediction system about DL reasoners. More specifically, given an input ontology, its feature values must be computed in the first place, then provided to the system in order to get prediction about the performances likely to be exhibited by a reasoner when attempting to process this ontology. Hence, we expect that the time required to compute the features is shorter than the one a reasoner would take to process the ontology. At the current stage of the study, we did not yet specified the reasoner(s) to examine, neither determined which are the performance to anticipate. Therefore, we will adjudge a feature to be *computationally expensive*, whenever the time required to get its value exceeds the one spent to load the ontology. Nevertheless, we are attempting to study a whole ontology corpus rather than one single ontology. We assume that feature value computational time would significantly vary across the corpus members, since the selected ontologies belong to quite distinct size bins. Thus, we decided that all the comparison will be carried out by considering the variation of the runtime achieved across all the ontologies.

To meet this requirement, we stated by partitioning our feature suite into sets, $S_1, \ldots, S_l$, made up from computationally related features. Actually, we tried to arrange the features according to our assumption about their levels of computational complexity. Table 3.1 describes the built in sets. Later on, for each ontology in the above described corpus, we recorded the time spent to load it into memory. Then, we measured its feature values and saved the time consumed over each feature set. All the reported times are summarized in Figure 3.6 using a multiple box-and-whiskers plot. In this figure, the y-axis corresponds to the logarithmic of the recorded computing time, initially measured in milliseconds. Box-and-whiskers plot provide an excellent visual summary of the variation in the reported runtimes through 5 statistical measures: minimum runtime value (MIN), lower quartile (Q1),

[13]Here, the profiles description follows the specification we made in Subsection 3.2.2

Name	Set members (the features)
S1	The ontology expressivity features (DFN, OPR).
S2	The ontology size features (SSIGN, OA, OLA, KBS).
S3	The class hierarchy (HC) features depending on *Breadth-First* traversal of the class inheritance tree, i.e. the depth of the class hierarchy and its cohesion, respectively HC_MHD and HC_COH, in addition to the cohesion of the ontology O_COH.
S4	All the HC structural features except those in S3.
S5	The property hierarchy (HP) features depending on *Breadth-First* traversal of the property inheritance tree, i.e. the depth of the property hierarchy and its cohesion, respectively HP_MHD and HP_COH.
S6	All the HP structural features except those in S5.
S7	The RCYC and SCYC features since uncovering the cycles within class definitions requires a structural transformation of the axioms.
S8	The CCP feature as patterns are computed based on SPARQL queries.
S9	All of the syntactic features except those in S8 and S7.

Table 3.1: The ontology feature distribution over the built in sets.

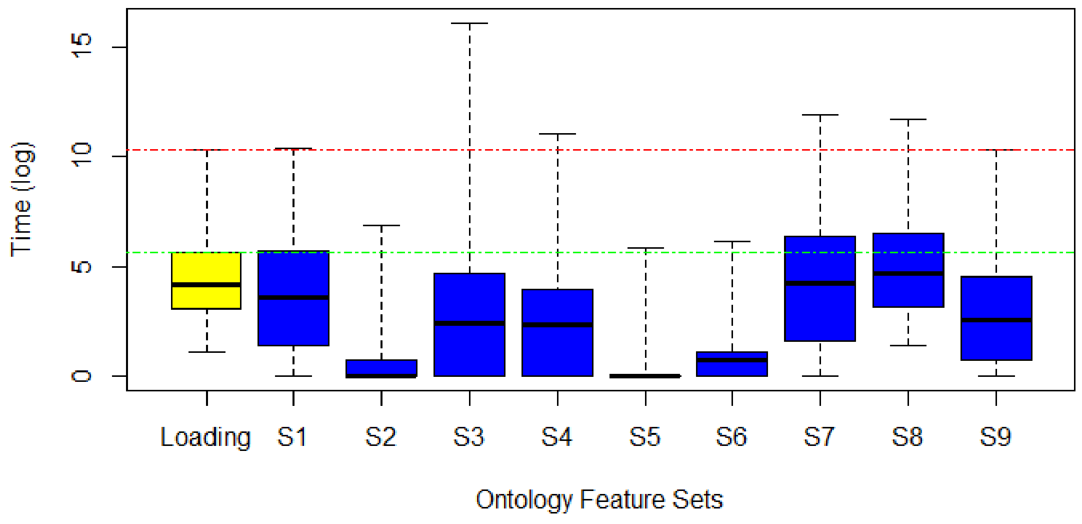


Figure 3.6: Comparing the ontology loading time to the computational time of the ontology feature sets recorded across the selected corpus.

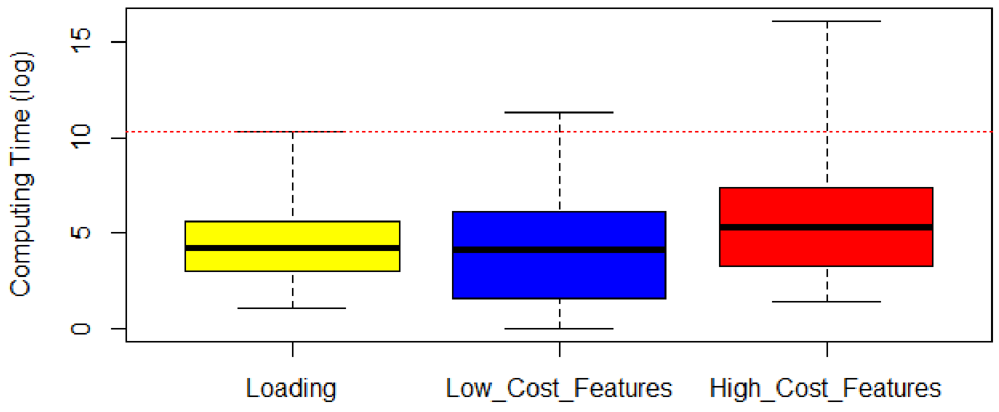


Figure 3.7: Outlining the efficiently computable ontology features

median, upper quartile (Q3), maximum time value (MAX). The ends of the vertical line, called the *whiskers*, indicate the MIN and MAX reported runtime across all the ontologies, whereas the line inside the box indicates the mean runtime value. The divergence in the loading time amongst the ontologies is highlighted in the first yellow box. Each of the remaining boxes describes the time variation of a specific feature set. The red dashed horizontal line outlines the maximal reported loading time, however the green dashed line corresponds to the third quartile. Feature sets having their upper whisker significantly above this line are considered to be *computationally expensive* ones. On these basis, we would remark that S3, S7 and S8 are at the worst case the most time consuming feature sets. We would also notice that the upper whisker corresponding to the S4 box is slightly above the red line, however its third quartile is below the green line. Therefore, we did not tag this feature set as a computationally expensive one. The remaining sets stands for the computational efficient features comparing to the loading time. More precisely, we would observe that the structural features characterizing the property hierarchy, within the S5 and S6 sets, have the lowest computing cost. Typically, a property inheritance hierarchy is far less developed than the class inheritance hierarchy. This is a common ontology modelling practice. Hence, HP feature values are often measured within a short cut-off time.

Figure 4.2 summarizes our inspections. We mainly distinguished between two groups of features according to their empirically provided computational cost, with respect to the overall ontologies loading times: the low cost features and the high cost ones. Luckily, the values of the major part of our feature suite, 108 out of the 116 features, could be computed within a time which is reasonably acceptable comparing to the one required for the loading. This observation is available for any ontology within the selected corpus. We should highlight that our ontology profiling tool could be further optimized in order to reduce the computational cost of the S4, S7 and S8 features. For instance, we would considerer approximating the depth of the HC inheritance tree, using the Knuth's estimate of search tree size [Knu74], rather than measuring its exact value with BFS. We would also look for different formulation to measure the hierarchy cohesion, to avoid the traversal search of HC. Besides, the SPARQL queries we used to detect the patterns could be replaced by OWL API based methods. However, at the current moment of the study, we are not sure whether the computationally expensive features are important or not for the prediction of the reasoner performances. Hence, any further optimization could be postponed to a later stage.

3.4 Conclusion

In this chapter, we introduced a novel ontology feature suite made up from qualitative and quantitative easy to compute metrics, capable to unveil the internal design complexity of OWL ontologies. We organized our features within four categories covering a broad range of structural and syntactic ontology design attributes. We assume that using these features, ontology engineers could achieve a better understanding of the overall complexity of their modelled ontologies. For instance, they could characterize its size, expressivity level and how regular or potentially irregular its class and property inheritance tree. Furthermore, they could inspect how rich the ontology is in terms of axioms types, how often are used the DL constructors to express more complex classes and whether some commonly known DL complexity sources appears in its vocabulary. Later on, we described the ontology corpus that we will consider throughout this thesis. The computing time of the feature values was inspected and compared to the loading time of the ontology under examination. This comparison was carried over each ontology within the selected corpus. As result, we identified the set of the most efficiently computable features and highlighted the potentially time consuming ones. In the remainder of the thesis, we will describe how these features are applied to the development of a predictive learning system about ontology reasoners.

CHAPTER 4

Learning the Empirical Robustness of Ontology Reasoners

Contents

4.1 Introduction

The main purpose of this thesis is to provide support for users looking to cope with the DL reasoner performances variability phenomena. The empirical behaviours of these systems will be tracked down by intelligent exploration of data describing their past running. Indeed, we will be employing supervized machine learning (SML) techniques to uncover the relationships between the ontology design complexity and the reasoner performances. Thus, generic models could be trained to predict the future reasoner behaviours over any input ontology. More specifically, we are aiming to learn a predictive model able to anticipate whether a given ontology is *empirically hard* regarding a group of reasoners. Then, we will conduct a finer-grained analysis of individual reasoners in order to assemble for each of them, a *predictive robustness profile* (*PRP*). All of these inspections will be carried out under a specific usage scenario. To help accomplish these purposes, we designed a generic learning framework employing a rich set of SML techniques. Description details of this framework as well as its empirical evaluation will be depicted throughout this chapter. Owing to this proposal, we succeeded to prove the utility of our proposed ontology feature suite, in predicting the robustness of 6 state-of-art DL reasoners.

We organized this chapter as follows: Section 4.2 specifies the examined ontology reasoning predictive problems and outlines the design of our generic learning framework. The various supervized learning techniques employed in this framework are depicted in Section 4.3. The DL reasoners and their ontology classification results are detailed in Section 4.4. This data collection is provided to our learning framework in order to train a first predictive model about the hardness of ontologies. Assessment details of this study case are provided in Section 4.5. Then, in Section 4.6, the built in reasoner predictive robustness profiles are depicted and analytically discussed. In Section 4.7, the various trained predictive models are put into examination in order to unveil the key ontology features. Finally, our concluding remarks are sketched in Section 5.7.

4.2 Predictive Modelling of the Ontology Reasoning

We start this chapter by clarifying the concepts of ontology *hardness* and reasoner *robustness*, then we will set up a formal definition of the reasoner predictive robustness profile, *PRP*. Later on, a description of our generic learning framework will be provided.

4.2.1 Robustness and Hardness judgement constraints

V. Mikolàšek [Mik09] defined the robustness in the software field as *the capability of the system to deliver its functions and to avoid service failures under unforeseen conditions.* Gonçalves et al. [GMPS13] have relied on this definition in order to conduct an empirical study about the robustness of DL reasoners. The authors have underlined the need to specify some particular computational constraints before assessing the reasoners. These constraints would describe a specific reasoner usage scenario, under which a reasoner may be considered either as robust or non-robust. We extended the Gonçalves et al. recommendations by designing the **RJC** set as follows:

Definition 21 (RJC set) *RJC stands for the reasoner robustness judgement constraints. It is a finite set of variables representing functional (FC) and non-functional (NFC) requirements relative to a particular DL reasoning scenario. RJC includes both FC={*time-limit*,* memory-limit*,* ontology-range*} and NFC={*reasoning-task*,* success-state*,* failure-state*}, i.e.* $RJC = FC \cup NFC$*.*

In our study, we chose to examine an online reasoning scenario. Under this context, the provided time frames are likely to be tight ones, commonly restricted to few minutes, while the memory usage might be less constrained since reasoners would be running on powerful server machines. Any range of ontologies should be supported by the reasoners. On the other hand, we decided to inspect the DL classification task given its earlier discussed importance (see Chapter 1, Subsection 1.4.1). This yields to the following definitions of success and failure states: the *success* is reported when the reasoner terminates the ontology classification task within the fixed time-limit and delivers correct results; the *failure* means either a reasoner crashes when processing the ontology, or exceeds the time limit or eventually delivers unexpected results. On these basis, we would define the robustness of a single reasoner with respect to a particular RJC set as follows:

Definition 22 (Reasoner Robustness) *The robustness of a reasoner stands for its ability to successfully terminate a reasoning task under specific computational resource constraints, for any given ontology.*

We believe that our definition of reasoner robustness would cover both of the desired efficiency and the correctness qualities. Thus by tracking down this property, we would provide ways to understand and to cope with the reasoner performances variability phenomena, previously inspected. Definition 22 can be further extended to specify the *most robust reasoner* with respect to an ontology corpus and a particular group of candidate reasoners.

Definition 23 (Most Robust Reasoner) *Given a fixed set of RJC, the most robust reasoner over an ontology corpus is the one satisfying the success requirement for the highest number of ontologies while maintaining the shortest computational time.*

Considering the previous definitions, we can now settle the concept of an ontology overall empirical hardness (OOH) as follows:

Definition 24 (Ontology Overall Hardness) *Given a fixed set of RJC, a hard ontology regarding a homogeneous group of reasoners is the one causing the failure of at least one of them.*

Obviously, a rule must be set to be able to adjudge a group of reasoners as a homogeneous one. In our study, we chose the reasoning methodology as the main criterion to construct the reasoner groups.

4.2.2 Ontology and Reasoner profiles from a predictive perspective

Our first learning problem is about predicting an ontology overall hardness (OOH) with respect to its feature description. We recall that the features are metrics characterizing the ontology design complexity, and belong to some feature space $\mathcal{F}^d$, where d stands for the space dimension. Thus, any ontology O would be represented by a d-dimensional vector $X^{(O)} = [x_1, x_2, \ldots, x_d]$ called a *feature vector*, where x_i with $i \in [1, d]$ is a feature value. We design the learning of OOH as a binary ML classification task [KZP06] where exactly two labels are to be predicted. Given a set of observations about a collection of ontologies provided with their reasoning results, the OOH predictive model, denoted by $M_{(OOH)}$, would be trained to categorize any ontology either as "Hard" or "Safe". Formally, M_{OOH} maps the ontology feature vector, i.e. $X(O)$, into one label value, i.e. $\hat{l}$, of the target variable $\{Hard, Safe\}$. It is worth noting that the term "*Safe*" would only imply that, with a certain degree of confidence, all the inspected reasoners are able to successfully achieve the reasoning task over the examined ontology while operating under specified RJC constraints.

In our second learning problem, we chose to investigate means to cope with the variability of reasoner empirical robustness across the ontologies. Given a suitably large and diversified population of ontologies and a fixed set of RJC, we assume that the output quality of a reasoner is closely depending on the quality of the input, i.e. the ontology. Such quality would be depicted using our proposed ontology features suite and formally represented via a feature vector, $X(O) \in \mathcal{F}^s$. On the other hand, the quality measure of the output may be any characteristic of the result that we find significant in the description of the reasoner robustness. We design by $\mathcal{Q}^s$, the space of reasoner output quality attributes, where s stands for the space dimension. Henceforth, our problem is about characterizing the correlation between $\mathcal{F}^d$ and $\mathcal{Q}^s$. Our main claim is that such correlation could be materialized using a reasoner robustness profile. The latter one is a composition of predictive models, such that every model maps the quality of the input to an expected output quality attribute. We call this composition, the *predictive robustness profile* (*PRP*) of an individual reasoner. A generic formal definition of PRP is provided as follows:

Definition 25 (Predictive robustness profile) *Given a finite set of RJC, an ontology quality features space $\mathcal{F}^d$ and a reasoner output quality attributes space $\mathcal{Q}^s$, the predictive robustness profile of a particular reasoner R, denoted by PRP_R, is a s-dimensional vector, such that each of its elements is a predictive model relative to a particular output quality attribute, i.e. $PRP_R = (M_{(Q_1|R)}, \ldots, M_{(Q_s|R)})$, where $\forall i \in [1, s]$, $M_{(Q_i|R)}$ stands for the learned model, able to map the feature vector of an ontology O, denoted by $X(O) \in \mathcal{F}^d$, to one particular value of the quality output of a reasoner R according to the attribute Q_i.*

$$PRP_R^{RJC)} : \mathcal{F}^d \mapsto \mathcal{Q}^s \tag{4.1}$$

With respect to our above described reasoning scenario, we selected two main reasoner output quality attributes. The first one stands for the nature of the termination state, denoted by S. Whereas, the second attribute, designed by T, outlines the exact time spent by the reasoner to get this state. The T attribute would allow for reasoner efficiency inspection and comparison versus other systems. Hereafter, we will need to train the models corresponding to each attribute from our output quality space $Q^s = \{S, T\}$. The main purpose of predicting of a reasoner termination state is to have a model capable to discriminate between the success state and the different forms of a reasoner failure for any unforeseen ontology. Hence, we could list four possible labels: "*Success*", "*Unexpected*", "*Timeout*" and "*Halt*". According to above specification of RJC constraints, the *Unexpected* state is notified when the reasoning task is achieved in time, however the delivered inferences are adjudged as unexpected with respect to the employed reasoner correctness checking method. *Timeout* stands for the case where the task does not terminate within the time limit and *Halt* indicates a sudden stop of the reasoner owing to some execution error. Considering this description, the learning task represents a multi-class ML problem [Kot07]. A class stands for those ontologies sharing the same label. We denote by $M_{(S|R)}$ such a model. The second quality attribute would rather require the training of a regression model since the target variable is a quantitative value, i.e the runtime. We denote by $M_{(T|R)}$, the underlying model. Intuitively, one would be interested in checking the reasoner runtime only in case of reasoner success, which entails the correctness of the results. Therefore, $M_{(T|R)}$ is conditionally depending on $M_{(S|R)}$. Literally, $M_{(T|R)}$ would estimate the reasoner computational time only when the success is predicted.

To train the above described predictive models, we need to provide empirical data describing the past running of a reasoner over a rich collection of ontologies. This data would indicate the reasoner runtime and termination state. One major obstacle we faced when looking to achieve this purpose was about how automatically check the correctness of a reasoner results. In chapter 2, we already reviewed the state-of-art solutions, and we highlighted that no existing method can handle a reasoner on its own and adjudge the correctness of its results without any comparison with other reasoners. In this thesis, we chose to work with Gardiner et al. [GTH06] reasoner correctness checking method. This is because it is a fully automated solution, i.e. no expert or any human intervention is required. The reasoner output are checked for correctness by a majority vote, i.e. the result returned by the most reasoners are considered to be correct. Certainly, this is not a faultless method. Improving it would be advantageous for our study, however this issue is out of the scope of this thesis.

4.2.3 Description of the learning framework

All of the previously specified learning problems are supervised ML tasks. $M_{(OOH)}$ and $M_{(S|R)}$ are ML classification models, while $M_{(T|R)}$ is a regression model. Nevertheless, these models have a common design structure. Actually, all of them try to map some feature vector describing an ontology, to some label value from the target variable. Obviously, this target is close to the nature of the prediction problem. Despite this difference, we can design a generic learning framework capable to accomplish all the needed training steps for each of the designed models. Description details of this framework is provided in the following. We abbreviated by P, any learning problem about ontology reasoning. In our context, P would be either (OOH), or $(S|R)$ or $(T|R)$. It is important to note that this framework is a more improved and basically generalized version of the learning methodology, we first established in [AYL15a].

In any standard machine learning process, every instance in a training dataset is represented using the same set of features. In our case, the instances are ontologies belonging to some corpus $\mathcal{C}(\mathcal{O}) = \{O_1, \ldots, O_N\}$, with N denoting the size of the corpus $n =\mid \mathcal{C}(\mathcal{O}) \mid$. Every ontology $O_i \in \mathcal{C}(\mathcal{O})$ is represented by a d-dimensional feature vector $X^{(O_i)} \in \mathcal{F}^s$. The ontologies are provided with a known label. The vector of all ontologies labels is specific to one particular learning problem P and denoted by $\mathcal{L}_P = [l(O_1), l(O_2), \ldots, l(O_n)]$, where $l(O_i)$ is the label of the i-th ontology. P belongs to our problem space $\mathcal{P} = \{OOH, (S|R), (T|R)\}$. By gathering the pairs (feature vectors, labels), a dataset is built in and designed by $\mathcal{D}_P = \{(X^{(O_i)}, l_R(O_i)\}$, $i \in [1, N]$. Afterwards, the $\mathcal{D}_P$ is provided to a supervised machine learning (SML) algorithm in order to train its data, uncover the hidden patterns and establish the predictive model about the examined problem, i.e. the M_P model. The latter one can have different forms: mathematical function, a graph or a probability distribution, etc. The form of the model depends on the employed supervised ML algorithm. When a new ontology, which does not belong to the dataset, is introduced, then the task is to compute its features and then predict its exact label $\hat{l}_P(X^{(O)})$, using the already trained $\mathcal{M}_P$.

$$\mathcal{M}_P \ : X \in \mathcal{F}^d \mapsto \hat{l}_P(X) \in \mathcal{L}_P \tag{4.2}$$

We remind that we are aiming to uncover which of the ontology features have the highest impact on reasoner empirical robustness. A simple approach is to force models to contain fewer uncorrelated features. Indeed, the performances of the SML algorithms are very sensitive to the quality of the employed features. Uninformative or highly inter-correlated ones could deteriorate the accuracy of the learned models. Therefore, feature selection methods [CS14] are employed in order to remove irrelevant and redundant features. A rich plethora of this kind of methods is available in the dedicated literature. We selected the most known ones, and we put them in a specific set, that we denoted by $\mathcal{FS}$. The application of those methods will involve the creation of new dataset variants, called the *featured* datasets. The latter ones are characterized by an eventually reduced subset of features. Similarly, we will investigate several kinds of SML algorithms, gathered in a finite set denoted by $\mathcal{SA}$. Worth noting, the $\mathcal{FS}$ set contains the $\varnothing$ element, which means no feature pruning to apply. This configuration allows us to maintain the initial dataset, denoted by $RAWD$, and thus compare between models trained from $RAWD$ and those trained from the featured datasets. The training steps will be repeated as far as the number of the computed datasets for a given problem and the number of available SML algorithms. Then, the established models will be evaluated against a bunch of assessment measures, denoted by $\mathcal{AM}$. Furthermore, we will provide methods to compare the predictive models, in order to uncover their *"best"* one regarding the employed assessment measures.

We put forward these various inspections in order to identify the most effective combination (learning algorithm, feature selection method), which yields to the most accurate model given the available data. In our learning framework, all of deployed SML techniques are gathered in one SML techniques base, denoted by $\mathcal{TB}_{sml}$. Roughly speaking, $\mathcal{TB}_{sml}$ stands for the triplet $< \mathcal{FS}, \mathcal{SA}, \mathcal{AM} >$. Elements within $\mathcal{TB}_{sml}$ are arranged in a way that indicates whether they would be applied to ML classification or regression problems.

Algorithm 2 illustrates the previously described main steps in the training process given a specific learning problem P. It starts by building the dataset according to the input data which include computing the ontology features, clearing and normalizing the different values. Then, by referring to the provided labels in the vector $\mathcal{L}_P$, the problem nature is identified and accordingly the SML techniques base $\mathcal{TB}_{sml}$ is loaded (Algorithm 2, Line 3). In each nested iteration, $M^{(F,A)}$ stands for the predictive model trained by a particular SML algorithm $A \in \mathcal{SA}$ from a featured dataset computed

Algorithm 2: The Surpervized Learning Framework

```
Input  : $\mathcal{C}(O) = \{O_1, \ldots, O_n\}$ the set of the training ontologies, $\mathcal{L}_P = [l(O_1), \ldots, l(O_n)]$ the
         labels vector relative to the problem $P$.
Output: $M_P^{Best}$ best learned model given the problem $P$.
1  $[X^{(O_1)}, \ldots, X^{(O_N)}] \leftarrow computeTheFeatureVectors(\mathcal{S}(O))$;
2  $\mathcal{D}_P \leftarrow buildTheRawDataSet([X^{(O_1)}, \ldots, X^{(O_N)}], \mathcal{L}_P)$ ;
3  $\mathcal{TB}_{sml} \leftarrow getSMLTechniquesBase(\mathcal{L}_P)$ ;
4  foreach $F \in \mathcal{FS}$ do
5      $\mathcal{D}_P^F \leftarrow buildTheFeaturedDataSet(F, \mathcal{D}_P)$;
6      foreach $A \in \mathcal{SA}$ do
7          $< M^{(F,A)}, \pi^{(F,A)} > \leftarrow trainAndAssess(A, \mathcal{D}_P^F, \mathcal{AM})$ ;
8          $\mathcal{M}^F \leftarrow \mathcal{M}^F \cup \{< M^{(F,A)}, \pi^{(F,A)} >\}$ ;
9      end
10     $< M^{(F,best)}, \pi^{(F,best)} > \leftarrow bestModelFromADataSet(\mathcal{M}^F)$ ;
11     $\mathcal{M}^{Best} \leftarrow \mathcal{M}^{Best} \cup \{< M^{(F,best)}, \pi^{(F,best)} >\}$ ;
12 end
13 $M_P^{Best} \leftarrow bestModelAccrossDataSets(\mathcal{M}^{Best})$ ;
14 return $M_P^{Best}$;
```

using a feature selection method $F \in \mathcal{FS}$. Within `trainAndAssess` function (Algorithm 2, Line 7), we utilize the standard stratified 10-fold *cross-validation* technique (see Chapter 1, Subsection 1.5.4.3) for the evaluation of the trained models. For the sake of simplicity, we only retain the average values of the computed assessment measures over all the cross-validation iterations. Then, we calculate an overall quality assessment score for each trained $M^{(F,A)}$ model, that denote by $\pi^{(F,A)}$. Based on this score, the *"best"* model given a specific dataset variant is selected, then the *"best"* one across all the datasets will be decided. Worth to note, the comparison of models across the datasets (Algorithm 2, Line 13) takes into account the number of the employed ontology features as well as the value of the quality score. More precisely, the model having the lowest features dimension while approximating the highest computed value of the $\pi^{(F,A)}$ score, will be picked as the final M_P^{Best}. In Section 4.3, we will provide a nominative description of the $\mathcal{TB}_{sml}$ elements and more details about the selection modes.

Obviously, the above described learning framework is a time consuming one. This would be due to the high complexity of the employed learning techniques and the cost of the iterative cross-validation process. Fortunately, this kind of study is often established offline, with enough computational resources. As a result, a kind of meta-knowledge base characterizing the examined problems in terms of predictive models would be built in. On the other hand, the predictions using priorly learned models and can efficiently be computed online.

4.3 Learning Methods and Materials

All of the learning steps described in Algorithm 2 are written with the Java language in one single prototype. Our techniques base $\mathcal{TB}_{sml}$ is basically composed from SML techniques provided within the widely accepted *Weka* [HFH+09] tool-kit (cf. Chapter 1, Subsection 1.5.5). More specifically, we made use of the Weka Java API[1] in order to automate the learning process. In this section, we provide a brief description of the employed SML techniques respectively arranged within $\mathcal{FS}$, $\mathcal{SA}$ and $\mathcal{AM}$ sets. For a more comprehensive description, please refer to Chapter 1, Section 1.5.

4.3.1 Feature selection methods ($\mathcal{FS}$)

The *feature selection* is one important step in dataset preprocessing for training purpose. It aims at selecting the most relevant features, by weeding out the useless ones. In previous works, Kang et al. [KLK12] have compared 6 feature selection methods. Whereas, Sazonau et al. [SSB14] have deeply investigated the inter-correlation between ontology features, using the *Principal Component Analysis* (**PCA**). In our study, we are more interested in tracking out the subset of features, which correlate the most with the performances of a given reasoner. We therefore advocate including all features that show some promise of being predictive, and relying on feature selection methods to identify the most useful ones. To achieve this task, we chose two well known feature subset selection methods, described and compared in [AT16]: first, the *Relief* method (**RLF**), an instance-based attribute ranking algorithm[2]; then, the *Correlation-based feature subset selection* method (**CFS**), which looks for subsets of features that are highly correlated with the target variable while having low inter-correlations. Both methods could be applied to ML classification or regression problems. Furthermore, we chose to investigate the utility of employing *supervised discretization* [GLS+13] as a feature selection method for ML classification problems. Basically, this technique stands for the transformation task of continuous data into discrete bins while considering the distribution of the target labels. When no bins are identified for a given feature, this latter one is considered useless and could be safely removed from the dataset. As an application method, we opted for the well known Fayyad & Irani's *supervised* discretization technique (**MDL**) [GLS+13]. Furthermore, some observations we made throughout our previous experiments [AYL15a] have motivated us to propose two additional selection methods, namely

[1]The Weka API is available at `www.cs.waikato.ac.nz/ml/weka/downloading.html`

[2]*RLF* will be used with a ranking threshold equal to 0.01.

RLF_MDL and **CFS_MDL**. Actually, we noted that MDL improves the predictive power of some SML algorithms, however it does not significantly reduce the dimensionality of our features space. On the other hand, RLF and CFS produces more compact feature vectors, but with little or even no increase in the models accuracy. Therefore, we decided to propose a further feature selection procedure operating within two steps: first, a base $\mathcal{FS}$ method, i.e. **RLF** or **CFS** is employed, then **MDL** is applied over the achieved results. In overall, the $\mathcal{FS}$ set is made up from $\{RAWD(\varnothing)$, **MDL**, **CFS**, **RLF**, **RLF_MDL**, **CFS_MDL** $\}$.

4.3.2 Supervised machine learning algorithms ($\mathcal{SA}$).

We chose representative algorithms from distinct categories of the supervised learning approaches (cf. Chapter 1, Section 1.5.2). Then, we arranged the $\mathcal{SA}$ set according to the nature the learning problem. Table 4.1 lists the various algorithms deployed within the $\mathcal{SA}$ set and gives a brief description of their main features. It is important to note that all the used implementations were applied with the default parameters of the Weka Toolkit. Further details and description about these algorithms can be found in [Abb14].

4.3.3 Assessment measures ($\mathcal{AM}$)

Evaluating the quality of a model is one of the core stages in machine learning. It indicates how successful are the predictions made by a trained model. In Subsection 1.5.4 of Chapter 1, we listed various assessment measures and distinguished between those evaluating the accuracy of ML classification models and those checking the goodness of fit of the regression models. In the following, we briefly recall the measures that will be considering in our learning framework.

Assessing binary ML models In a binary ML classification scenario, the target variable has only two possible outcomes, such as {Hard, Safe} in the case of OOH prediction problem. Usually, we refer to these class of labels as positive or negative. The positive and negative instances that a model predicts correctly are called respectively true positives (TP) and true negatives (TN). Similarly, the incorrectly classified instances are called false positives (FP) and false negatives (FN). These four values compose the *confusion matrix* [Abb14, Kot07] of the model. A rich variety of model quality scores could be derived from this matrix. Our focus was put on the *F1-Score* (**F1**) to summarize the quality of a model. This measure combines both *Precision* and *Recall* into a single value. The *F1* score ranges from 0 to 1 and it is a positively-oriented score, i.e. the higher is the *F1*, the more accurate the model is.

Algorithm	Category	ML Task	Description
Random Forest (**RF**)	Logic	Classification+ Regression	A combination of C4.5 decision tree predictors.
Simple Logistic (**SL**)	Logistic	Classification	A linear logistic regression algorithm, using *LogitBoost* with simple regression function.
Multilayer Perceptron (**MP**)	ANN	Classification	A back propagation algorithm to build an Artificial Neural Network model to classify the instances.
Sequential minimal optimization (**SMO**)	SVM	Classification+ Regression	An implementation of the Support Vector Machine with polynomial kernel.
L2-regularized SVM (**L2SVM**)	SVM	Classification+ Regression	Another implementation of SVM with a linear kernel.
K-Nearest-Neighbour (**IBk**)	Lazy	Classification+ Regression	An instance-based algorithm running with a normalized euclidean distance.
Averaged N-Dependence Estimators (**A1DE**)	Naive-Bayes	Classification	An algorithm averaging over a small space of alternative naive-Bayes-like models.
Linear regression (**LR**)	Linear	Regression	An algorithm of common use which tries to fit the data to a linear curve.

Table 4.1: Description of the supervised learning algorithms.

Assessing ML multi-class models In case of ML multi-class problem, we are very often concerned with the *imbalanced dataset* issue. According to Bao-Gang et al. [HD14], a dataset is considered imbalanced if at least one of the classes (minority class(-es)) contains much smaller number of samples than the remaining classes (majority classes). In previous works [KLK12, SSB14], the accuracy standing for the fraction of the correct predicted labels out of the total number of the dataset samples, was adopted as main evaluation metric for assessing the multi-class models. However, accuracy would be misleading in the case of imbalanced datasets as it places more weight on the majority class(es) than the minority one(s). Consequently, high accuracy rates would be reported, even if the predictive model is not necessarily a good one.

Therefore, based on the comparative study conducted by Bao-Gang et al. [HD14], we chose to measure the ***Cohen's Kappa Coefficient*** (**Kappa**) and the ***Matthews Correlation Coefficient*** (**MCC**). Both of them try to quantify the degree of agreement between the predicted and the real labels by considering all values composing the generalized confusion matrix. We already provided the formal definition of these scores in Chapter 1, Subsection 1.5.4. *Kappa* and the *MCC* values vary between −1 and 1, where 0 represents agreement due to chance. In rare cases, negative values of MCC or Kappa are reported, whereas values relying between 0.6 and 0.8 indicate a good predictive model.

Assessing ML regression problems The predictive accuracy of the trained runtime models will be inspected using two well established indices, the *Root Mean Squared Error* (**RMSE**) and the *Coefficients of determination* (R^2) also known as the R-square. Both measures are described in [Abb14]. RMSE squares, then aggregates the difference between values predicted by the model and the real observed values. Thus, the computed RMSE score will have the same unit as the target variable, i.e. the time unit in our case. RMSE values range from 0 up to $+\infty$ and it is a negatively-oriented score, so lower values are better. On the other hand, R^2 is a coefficient ranging from 0.0 to 1.0 with no units. A model has healthy predictive ability when its R^2 is near to 1.

4.3.4 Best predictive model selection modes

Given a specific predictive problem and a variety of learning techniques available within the $\mathcal{TB}_{sml}$ base, our framework compares the trained predictive models and automatically adjudges the *best* one on the basis of some decision rules stored in the system. By *"best"*, we are denoting the model producing prediction with the highest degree of confidence, while employing the fewest possible number of ontology features. The ranking of models is decided according to their quality score, i.e. the π score (Algorithm 2, Line 10). The computing mode of the latter one is close to the nature of the learning problem. Hence, we suggest three problem oriented specialization of the π score: $F1TP$, MKS and RRS. These scores are respectively proposed to check the quality of binary ML models, multi-class ML models and regression models. The computing details are described in the following:

Selection of Best Binary ML classification model Commonly in binary ML problem, we are more likely to focus on either one of the label classes, i.e. the one considered particularly interesting for the learning. We refer to this latter one as the positive class. Hence, we would be looking for the model maximizing the TP value (true positive). However, this does not mean that quality prediction of the negative class is unimportant. In our opinion, a good ML binary model is the one showing an optimal trade-off between the TP value and the F1-Score value. We have expressed this preference rule in the form of a model quality score, that we denote by ***F1TP***. The latter one combines the above discussed measures as follows:

$$F1TP(M_P^{(F,A)}) = F1(M_P^{(F,A)}) \times TP(M_P^{(F,A)}) \tag{4.3}$$

where $M_P^{(F,A)}$ is the binary model trained from the dataset F using the SML algorithm A. We recall that $F1$ is a coefficient positively varying within the unit interval $[0, 1]$ and TP is an integer which takes values in $[0, pos]$, where *pos* stands for the number of the dataset instances having the positive label. Hence, the F1TP score is a float positively varying within $[0, pos]$. A perfect binary ML model will have its $F1TP$ equal to *pos*. Henceforth, the *best* binary ML model derived from F is the one *maximizing* the *F1TP* score.

Selection of Best Multi-class ML classification model We suggest to compute the *Matthews Kappa Score*, denoted by ***MKS***. As the acronym would suggest, *MKS* is a weighted aggregation of the *MCC* and the *Kappa* values, which would summarize how the model is resisting to the imbalance of the dataset:

$$MKS(M_P^{(F,A)}) = \frac{\alpha * MCC(M_P^{(F,A)}) + \beta * Kappa(M_P^{(F,A)})}{\alpha + \beta} \tag{4.4}$$

such that $(\alpha + \beta) = 1$ and $M_P^{(F,A)}$ is the model trained from the dataset F using the SML algorithm A. Our aggregation is coherent, since both *MCC* and *Kappa* are positively oriented scores ranging in $[-1, 1]$. Hereafter, the *best* multi-class ML model derived from F is the one *maximizing* the *MKS* score.

Selection of Best ML regression model A good regression model will have a low RMSE value and a high R^2 close to 1. Accordingly, we suggest to summarize the quality of a regression model by combining the above measures, into a single value that we design by *RRS*. The latter ones is computed as:

$$RRS(M_P^{(F,A)}) = (1 - R^2(M_P^{(F,A)})) \times RMSE(M_P^{(F,A)}) \tag{4.5}$$

where $M_P^{(F,A)}$ is the model trained from the dataset F using the SML algorithm A. We remind that R^2 is a coefficient positively varying within $[0, 1]$. Thus, (1-R^2) makes it possible to increase RMSE values when the R^2 decreases. On this basis, we can assert that *RRS* is a negatively oriented score and have the same unit as the RMSE score. Hereafter, the *best* regression model derived from F is the one *minimizing* the *RRS* score.

4.4 Data Collection for the Learning Process

To bring response to the above specified learning problems, we need to provide enough data describing previous running of reasoners over a selection of ontologies. Definitely, these running should be conducted under the same robustness judgement constraints (RJC). We remind that we have chosen to examine the online classification of OWL ontologies. Seeking for result reliability, we have chosen to work with the same evaluation tools employed in the latest ORE competitions [BGJ+14, DGG+15]. This includes the evaluation Framework[3] [GBJR+13] and the ontology corpora[4] [MBP13]. The motivation behind our choice is multi-fold: first, the ORE event is widely recognized by the Semantic Web community; the ontology corpora is very large, e.g. includes 16k ontologies, and well diversified throughout easy and hard cases; and finally the Framework checks both the efficiency and the correctness of reasoners.

In Chapter 2, Subsection 3.3.1, we depicted the main characteristic of the ontology corpus that we will use through this thesis. We recall that this corpus is composed from 2500 distinct ontologies, selected from the ORE corpora. Hence, using the ORE evaluation framework, we carried out a new ontology classification challenge gathering 6 DL reasoners. The latter ones were put into comparison when attempting to classify the 2500 ontologies. In the remaining of this section, we describe the examined reasoners and illustrate the evaluation results.

Reasoners. We selected a representative subset of popular reasoners to carry out our predictive learning study. Reasoner predictive robustness profiles will be constructed for each one in this selection. More precisely, we picked up the 6 best ranked reasoners[5] in both the DL and EL classification challenges during the latest ORE competitions series, i.e. 2014 and 2015. They are namely ***Konclude*** [SLG14], ***MORe*** [RGH12], ***HermiT*** [GHM+12], ***TrOWL*** [TPR10], ***FaCT++*** [TH06] and ***JFact***. We excluded ***ELK*** [KKS11] despite its good results and high rank, as far as it does not support the classification of ontologies within the OWL 2 DL profile. Besides, we were enable to include the *Pellet* [SPG+07] reasoner in our study, as the needed plugin to integrate it into the ORE Framework was not made available. We have previously characterized the functional attributes of the selected reasoners (cf. Chapter 1, Section 1.4) and discussed their main features. We would assert that this selection is a very diversified one covering different reasoning methodologies, programming languages and expressivity levels.

Result summary of reasoner evaluations. We run the ORE Framework in the sequential mode on a machine equipped with an Intel Core I7, CPU running at 3.4GHz and having 32GB RAM, where 12GB were made available for each reasoner. We set the condition of 3 minutes time limit to classify an ontology by a reasoner. This tight schedule would be consistent with the chosen scenario, i.e. the online DL classification of an ontology. Table 4.2 summarizes the results of the carried out ontology

[3]The ORE Framework is available at https://github.com/andreas-steigmiller/ore-2014-competition-framework/

[4]The full corpus is available at `http://zenodo.org/record/10791`

[5]All ORE'2014 reasoners are available at `https://zenodo.org/record/11145/`

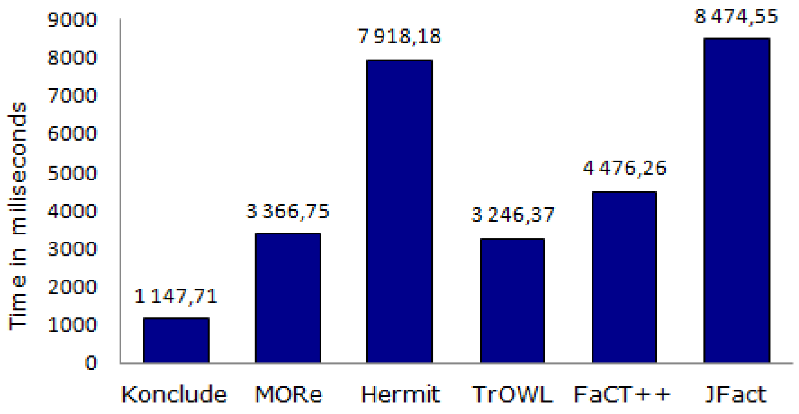


Figure 4.1: Comparison of reasoner average runtime over the correctly classified ontologies.

Reasoner	**#Success**	**#Failure**		
		Unexpected	**Timeout**	**Halt**
Konclude	2408	44	2	6
MORe	2182	71	241	6
HermiT	2167	3	312	18
TrOWL	2149	257	74	20
FaCT++	1968	18	419	95
JFact	1679	41	636	144

Table 4.2: Summary of reasoner evaluation results.

classification challenge[6]. It illustrates for each reasoner the number of ontologies classified with success; otherwise processed in time with correct derived inferences. The number of cases where the reasoner is flagged with failure are also reported in Table 4.2 and split up according to the nature of the failure. The reasoners are listed in Table 4.2 in decreasing order by their success rate; otherwise the most robust reasoner over the ontology corpus is the one at the top of the list (Definition 23). Furthermore, Figure 4.1 depicts the average runtime exhibited by each reasoner over the correctly classified ontologies, i.e. the success cases. We can easily notice that Konclude is the most robust reasoner, however this system failed to deliver correct results for 44 ontologies. On the other hand, TrOWL is the least reliable reasoner with 257 cases of unexpected results, this would be due to its incompleteness for some OWL profiles. Nevertheless, TrOWL has the second shortest average runtime over the correct cases. We would also remark that despite the ability of HermiT to deliver correct results, i.e. only 3 unexpected cases, its average runtime over correct cases is very high. Hence, it might be difficult to deploy this reasoner in online applications. All of these facts would outline the hardship to decide an absolute

[6]Results of our evaluations are available at `https://github.com/Alaya2016/OntoClassification-Results2016/`.

appropriate reasoner for all the ontologies. Obviously, a reasoner should be chosen according to its performances for a particular input ontology. However, such a task is a time and effort consuming one since it requires a lot of experimentation. That is why, learning to automatically predict the performances of a reasoner stands as a promising solution. Based on the above described reasoner evaluation results, training datasets will be built for each of the previously specified problems.

4.5 Predicting the Ontology Overall Hardness

Our first learning problem, P, is about training a model able to predict the overall empirical hardness of any given ontology. Specification of this problem has been formulated in Subsection 4.2.2. In the following, we will detail the training steps and illustrate the results of the conducted evaluations.

4.5.1 Training the OOH model

We started this study by compiling the label vector, $\mathcal{L}_{(OOH)}$, based on the DL classification results described in Section 4.4. $\mathcal{L}_{(OOH)}$ indicates for each test case ontology, i.e. $O \in \mathcal{C}(\mathcal{O})$, whether it is "Hard" or "Safe" with respect to 4 selected reasoners, namely: HermiT, MORe, FacT++ and JFacT. All of these reasoners implement Tableau-based algorithms [BS01] (c.f. Chapter 1, Section 1.4) and hence, could be considered as homogeneous group. We remind that an ontology is labelled as "Hard", whenever one of the studied reasoners fails to correctly classify it within the fixed time limit. Given this rule, we counted 958 hard ontologies out of the $\mathcal{C}(\mathcal{O})$ corpus, i.e. 38.32% of the tested ontologies. Figure 4.2 outlines the distribution of ontologies over the size and the hardness bins. We would remark that even for relatively small ontologies ($\leq$ 1000 logical axiom), hard ontologies could be reported. This would sketch out that the size is not the only cause of the ontology hardness. Whereas, the hardness is absolutely more frequent for large ontologies ($\geq$ 10 000 logical axiom). Henceforth, our task is to transform these empirical observations, into a generic model, thus that computationally troublesome ontologies could be figured out without any further experimentations. To reach this purpose, the $\mathcal{L}_{(OOH)}$ vector and the $\mathcal{C}(\mathcal{O})$ corpus are provided to our learning framework (Subsection 4.2.3) in order to train the $M_{(OOH)}$ model.

4.5.2 Assessing the OOH model

In this subsection, we discuss steps carried out by our learning framework towards identifying the optimal $M_{(OOH)}$ model[7], that could be derived from the available data and using the above described

[7]Here by optimal, we mean the one having the highest F1TP score.

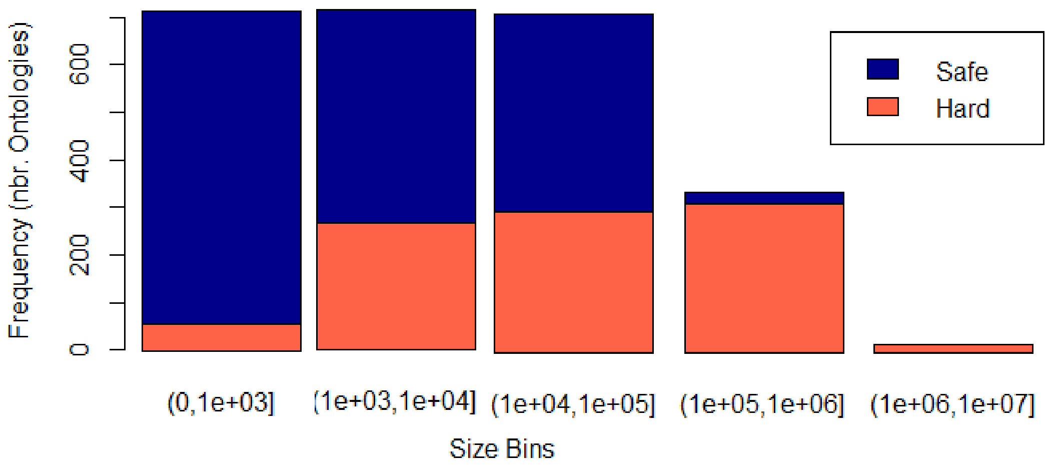


Figure 4.2: Distributions of ontologies over size and hardness bins

SML techniques (Section 4.3). To the best of our knowledge, we are the first to conduct such kind of study, thus it is not possible to make any comparison with existing works. We put special attention in inspecting how successful is the model in predicting the *hard* cases. Thus, we considered the latter class of labels as the positive one. Consequently, the TP value stands for the number of ontologies correctly predicted as hard ones. On these basis, we will be computing the $F1TP$ score (Subsection 4.3.4) of any $M_{(OOH)}$ model trained by our framework. In fact, by shifting the feature selection methods, i.e. $\mathcal{FS}$=$\{RAWD(\varnothing)$, **MDL**, **CFS**, **RLF**, **RLF_MDL**, **CFS_MDL**$\}$ and the SML algorithms, i.e. $\mathcal{SA}$=$\{$RF, SL, MP, SMO, L2SVM, IBK, A1DE$\}$, the framework builds a set of candidate models $M_{(OOH)}$, then decides the best one in terms of $F1TP$ score. Figure 4.3 depicts the variation of this score amongst the candidate models using *boxplots*. The latter ones provide a good visual summary of the computed values. Each box corresponds to the sorted $F1TP$ values computed for every model being trained from a particular dataset variant. Hence, it shows the spread of the estimated accuracies for the different employed SML algorithms and how they are related to the employed feature selection method. The ends of the vertical lines, called the "*whiskers*", indicate the minimum and the maximum reported $F1TP$ values. The name of the SML algorithm whose the corresponding model has reached these ends is also displayed in this figure. Thus, by looking to the upper whiskers, we can characterize the best trained model from each dataset, i.e the $M_{(OOH)}^{(F,Best}$ (Algorithm 2, Line 10). Before further discussing Figure 4.3, we should highlight that the number ontology features varies across the OOH datasets. In fact, initially the RAWD dataset contains 116 features. This number was reduced to 53 using **RLF**, only 18 in the case of both **CFS** and **CFS_MDL**, 51 with **RLF_MDL** and finally by applying **MDL**, we got the highest feature dimension, i.e. 103. Further important observations can be made from Figure 4.3. First, we would remark that the height of the boxes, which represent the disparity in $F1TP$ scores amongst the trained models, is remarkably scaled down in cases where

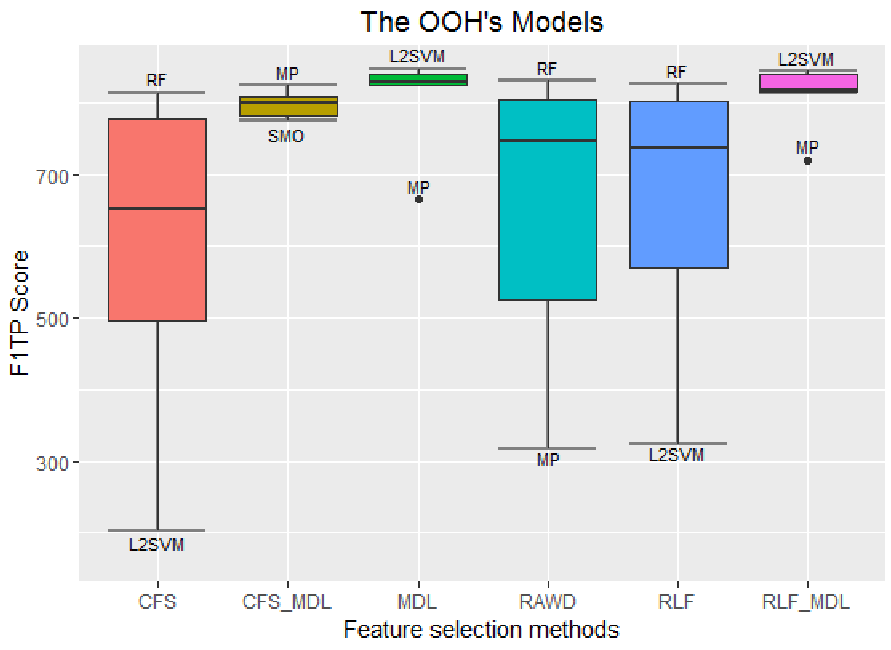


Figure 4.3: Comparison in terms of the F1TP score of the trained $M_{(OOH)}$ models.

the discretization (**MDL**) is employed. Hence, we would admit that preprocessing the data with **MDL** would enhance the predictive performances of some SML algorithms. For instance, we shall notice that the linear SVM (L2SVM) models have showed poor accuracies when trained from datasets featured by **CFS** or **RLF**, i.e. **L2SVM** is at the lower whiskers. However, models derived by this algorithm have outperformed all the other ones when **MDL** and **RLF_MDL** is employed. It is also interesting to note that the quality of the Random Forest (**RF**) models seems to be the less sensitive to the employed feature selection method. In fact, **RF** have exhibit good predictive performances even when the training is made from the raw data (RAWD). This would be owing to its internal feature selection mechanism. Furthermore, we can assert that maintaining too few features in the dataset it is just not unconditionally advantageous for the learning. This could be checked out by examining the assessment of models trained from the **CFS** dataset, which has the lowest number of ontology features. Actually, these models are at average the least performing ones in terms of $F1TP$. On the other hand, the topmost accurate models in terms of $F1TP$ score are the ones derived by **L2SVM** from the **MDL** and the **RLF_MDL** dataset variants. The average $F1$ scores achieved across the 10-fold cross validation of respectively the $M_{(OOH)}^{(MDL,Best)}$ and $M_{(OOH)}^{(RLF\_MDL,Best)}$ models are equal to 0.946 and 0.947. Besides, their respective $F1TP$ scores are about 847.122 and 844.182. These values indicates that the trained models are highly accurate, with almost 95% correctly predict cases. More specifically, the **MDL** based model is slightly better in predicting the *hard* ontologies. However, this model employs about twice the number of features that exists in the **RLF_MDL**-based model. On the basis of the previously set up selection rules (Subsection 4.3.4), our learning framework would output the $M_{(OOH)}^{(RLF\_MDL,Best)}$ model trained by **L2SVM**, as the final best model over the (OOH) problem. Hereafter, by referring to this model and without any further extensive experimentations, any user would be able to inspect whether its ontology is manageable in online applications by the examined reasoners.

4.6 Learning the Reasoners Predictive Robustness Profiles

As earlier specified, the primary purpose of this stage of study is to build meta-knowledge about a set of candidate reasoners. These are Konclude, MORe, HermiT, TrOWL, FacT++ and JFact. The meta-knowledge is coded in the form of reasoner predictive robustness profiles (PRPs) (Definition 25). The latter ones will be acquired from data describing the reasoners empirical behaviours (Section 4.4) and using our learning framework (Subsection 4.2.3). Roughly speaking, a reasoner profile is made up of two predictive models: a multi-class model predicting the reasoner success, i.e. $M_{(S|R)}$, and a regression model estimating the runtime required by the reasoner to achieve this success, i.e. $M_{(T|R)}$. In the following, we will describe the various experiments that we put forward towards identifying the most accurate models to include in the reasoners predictive robustness profiles[8].

4.6.1 Training the reasoners' success models

Hereby, the learning problem P stands for $(S|R)$, i.e. learning to predict the termination state of reasoner R over any input ontology O. According to the specification made in Subsection 4.2.2, this means learning a multi-class predictive model, $M_{(S|R)}$, capable to discriminate between cases where a particular reasoner R would succeed to achieve a DL classification task and situations where R would fail. The different forms of failure were depicted in Section 4.4. Based on the evaluation results of the examined reasoners summarized on Table 4.2, a label vector $\mathcal{L}_{(S|R)}$ is assembled and provided to our learning framework together with the ontology corpus. These steps are repeated as far as the number of the examined reasoners.

We would like to highlight that the reasoners datasets subject of this stage of the study are imbalanced ones. This would be attested by inspecting Table 4.2. For instance when checking the Konclude results, we can easily notice the large difference between the number of success cases, otherwise the ontologies that will be labelled with *"Success"* (the majority class) and those that will get one of the failure labels (the minority classes). Its obvious that predicting the minor cases is much more

[8]Evaluation results including reasoner datasets, the assessment values and the learned models are available at `https://github.com/Alaya2016/Learning-Results2016/`

Reasoner	**RLF**	**CFS**	**MDL**	**RLF_MDL**	**CFS_MDL**
Konclude	68	11	70	53	11
MORe	72	14	97	65	14
HermiT	70	14	95	67	14
TrOWL	55	12	98	55	11
FaCT++	63	12	98	63	11
JFact	60	20	105	60	20

Table 4.3: Result summary of ontology features selection for all the reasoners datasets

interesting for both ontology and reasoner designers, since they would pinpoint a bottleneck situation. However, the learning from imbalanced datasets is considered as one of the most challenging issue in machine learning [HD14]. This aspect will be taken in considered by our framework, thus the *best* trained $M^{Best}_{(S|R)}$ model will be adjudged according to the **MKS** score (Subsection 4.3.4).

4.6.2 Assessing the reasoners success models

Our learning framework have trained various candidate $M_{(S|R)}$ models for every examined reasoner. We start the result analysis by looking into the dimensionality reduction achieved over each reasoners datasets using the feature selection methods available within $\mathcal{FS}$={CFS, CFS_MDL, MDL, RLF, RLF_MDL, RAWD}. Table 4.3 summarizes the results. Similar remarks to those reported in the previous case study could be drawn from this table. For all the reasoners datasets, we would observe that **CFS** minimizes the ontology feature dimension in the most noticeable way, **MDL** keeps the dimension close to its initial size and **RLF** maintains almost the half of the features. **MDL** associated to **CFS** does not decrease the number of features any further; however when put together with **RLF** some more fewer features are removed. For the 6 reasoners, the MKS score distributions over the trained models from each dataset variant are depicted in Figure 4.4 using *boxplots*. Given a reasoner R and one dataset variant, the box shows the amount of variation in the MKS score amongst the predictive models being trained from this dataset using different SML algorithms. Like in Subsection 4.5.2, we can characterize the best trained model from each reasoner dataset variant by looking to the top of the whiskers. Observations drawn from Figure 4.4 about the predictive performances of the employed SML algorithms are in accordance with those reported in Subsection 4.5.2. In overall, we would notice that when discretization (MDL) is employed the difference in MKS score between the candidate models becomes significantly lower. Once again, the potential of this method in enhancing the prediction quality of the SML algorithms is asserted. Except for the Konclude's datasets, it is

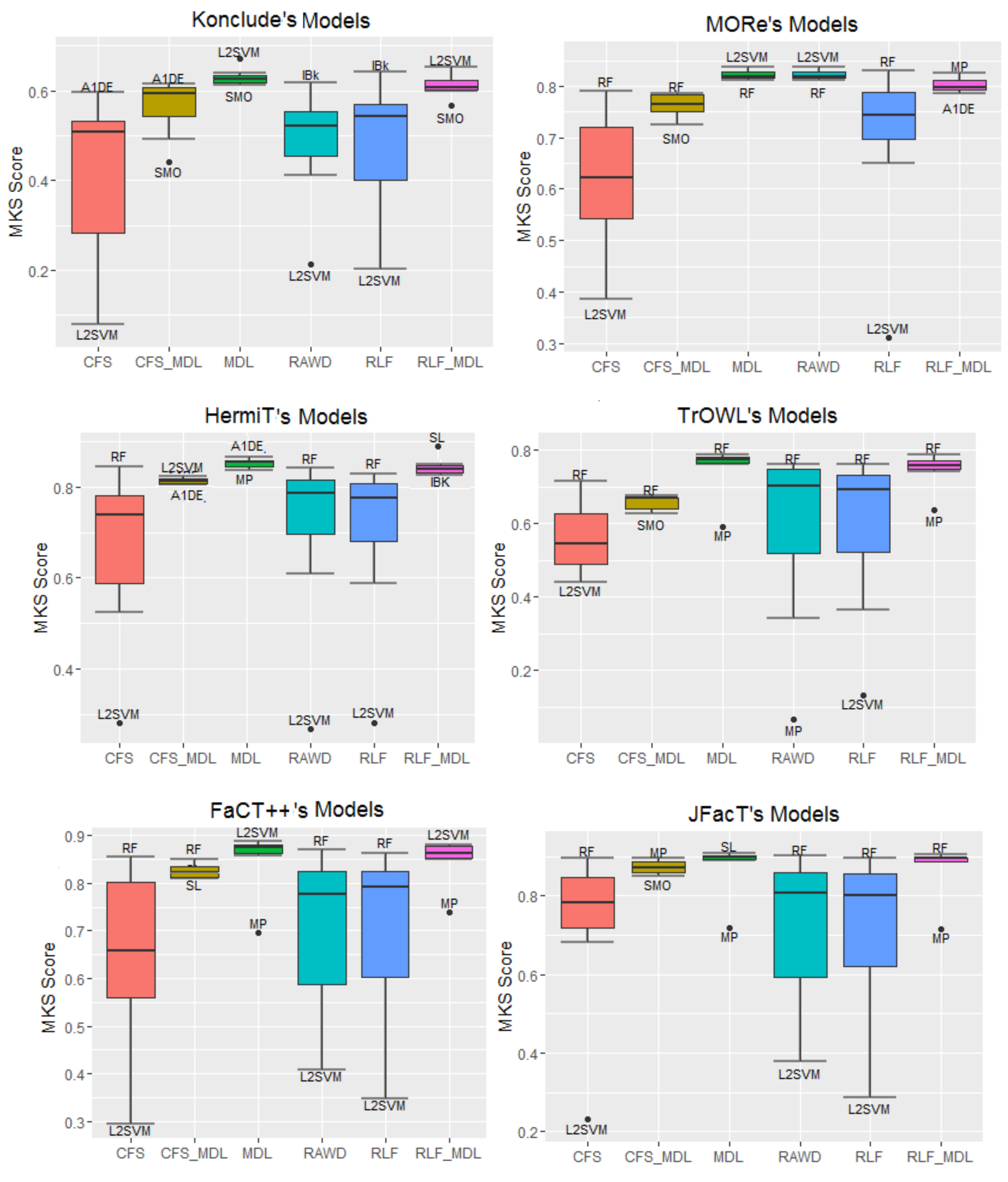


Figure 4.4: The MKS distribution over SML algorithms and across the dataset variants for all the reasoners.

Reasoner	FS	SA	MKS	F1-Score	#F
Konclude	RLF_MDL	L2SVM	0.653	0.974	53
MORe	RLF_MDL	MP	0.827	0.960	65
HermiT	RLF_MDL	SL	0.889	0.972	67
TrOWL	RLF_MDL	RF	0.787	0.951	55
FaCT++	RLF_MDL	L2SVM	0.881	0.956	63
JFact	RLF_MDL	RF	0.904	0.952	60

Table 4.4: Assessment summary of the final $M^{Best}_{(S|R)}$ models

remarkable that, in the most cases, the best models in terms of the MKS score are the ones trained by the Random Forest (RF). As previously highlighted, RF is the most stable SML algorithm which has yielded to good predictive models in all the study cases. Across the dataset variants and for all the reasoners, the scores of the models at the upper whiskers are fairly close to each others, even when the training is made from the raw dataset. This would witness that regardless of the fact that the dataset is featured or not, our initial ontology feature suite is rich enough and well representative of the ontologies' computational complexity which makes it possible to train highly accurate predictive models. However for all the reasoners, the topmost accurate models are the ones trained from MDL and MDL_RLF respective datasets. Similar to the OOH problem, the framework will prefer the $M^{Best}_{(S|)}$ trained from the MDL_RLF based dataset as its feature dimension is lower than the MDL based one. Table 4.4 illustrates the main attributes of the finally chosen $M^{Best}_{(S|R)}$ models. For every reasoner, the feature selection method (FS) and the learning algorithm (SA) which have induced the model are outlined. Table 4.4 also reports the model assessment values in terms of MKS and F1-Score averaged over 10 fold cross validation. The last column of table (#F) recalls the number of features within the corresponding reasoner dataset. A number of important observations can be made from this Table 4.4. In overall, we would remark that the learned reasoners best predictive models showed to be highly accurate, achieving in all cases, and over to $0.95F1$ value. In addition, they are well resisting to the problem of minor classes, since in all cases except for Konclude, the respective MKS scores were over 0.78. Konclude initial dataset is the most imbalanced one with little number of ontologies in minor classes. Under this constraint, an MKS about 0.65 is an acceptable value witnessing that the failure of Konclude could fairly be predicted. On the other hand, the MKS of the JFact best predictive model is above 0.90, indicating almost a *perfect* predictive model. All of these findings shed light on the high generalizability of the learned model as well as its effectiveness in predicting reasoner state of termination over online ontology classification tasks. It would also be observed that the size of feature vectors of the $M^{Best}_{(S|R)}$ models varies from 53 in the case of Konclude going to 67 in the case of HermiT. This is quite the half of the initially computed features. Such observation would further assert our assumption that there is some particular subset of the ontology features that are more correlated to the reasoner performances that the other ones. Besides, these key ontology features are close to the reasoner under study. Furthermore, we can notice that different learning algorithms are in the final selection. This would further highlight that the nature of the correlation between the ontology feature and the reasoner termination state is close to this particular reasoner. Indeed, **L2SVM** and **SL** would model the correlation using linear functions, however **MP** and **RF** employ sophisticated graph-based representation of the dependency. Hence, we would admit that the termination state of Konclude, Hermit and Fact++ could be explained and anticipated using easy to understand linear models which require relatively manageable number of ontology features. On the other hand, more complex models are required to predict MORe, TrOWL and JFacT.

4.6.3 Training the reasoners runtime models

In this subsection, the learning problem P is reduced to $(T|R)$. According to the specification provided in Subsection 4.2.2, the reasoner runtime regression model $M_{(T|R)}$ is conditionally related to $M_{(S|R)}$ model. Runtime will be estimated only when success of R over the input ontology is predicted. Given this requirement, for each reasoner, we removed from $\mathcal{C}(O)$ those ontologies which have induced its failure. Henceforth, let $\mathcal{S}_{(T|R)}(O) \subset \mathcal{C}(O)$ be the ontology training set specific to the reasoner R and the $(T|R)$ learning problem. Every ontology within $\mathcal{S}(O)$ will be labelled with the logarithm of the R's runtime. The log transformation was first proposed by Lin et al. [XHHLB08], whom found it to be of paramount importance to improve the overall accuracy of regression models. Thus, the label vector given a reasoner R would stand for $\mathcal{L}_{(T|R)} = [log(t(O_1)), \ldots, log(t(O_s))]$, where $t(O_i)$ represents the processing time spent by R to classify O_i. Hence, we will be learning to approximate, $\widehat{t_i}$, the logarithm of R' runtime. For each investigated reasoner its corresponding $\mathcal{L}_{(T|R)}$ vector and $\mathcal{S}_{(T|R)}(O)$ corpus will be provided to our learning framework, thus the best $M_{(T|R)}$ model could be established.

4.6.4 Assessing the reasoners' runtime models

Table 4.5 illustrated for every reasoner, the results of the first stage selection of its runtime predictive model; otherwise, its most accurate model derived from a featured dataset. More specifically, given a reasoner R and a feature selection method, Table 4.5 shows the name of the **SML** algorithm (column $\mathcal{SA}$) which have derived the predictive model having the lowest RRS score. The reported R^2 and $RMSE$ values are the average across the 10 folds cross-validation. The column #F shows the number

Reasoner	RAWD (∅)					CFS					RLF				
	$\mathcal{SA}$	R^2	RMSE	RRS	#F	$\mathcal{SA}$	R^2	RMSE	RRS	#F	$\mathcal{SA}$	R^2	RMSE	RRS	#F
Konclude	RF	0.984	0.388	0.006	116	RF	0.987	0.334	0.004	20	RF	0.983	0.385	0.006	27
MORe	RF	0.962	0.218	0.010	116	RF	0.963	0.251	0.009	28	RF	0.959	0.267	0.011	30
HermiT	RF	**0.969**	**0.382**	**0.012**	116	RF	0.968	0.383	0.012	16	RF	0.967	0.389	0.013	36
TrOWL	RF	0.972	0.316	0.009	116	RF	0.975	0.291	0.007	16	RF	0.972	0.312	0.009	29
FaCT++	RF	0.944	0.460	0.026	116	RF	0.944	0.455	0.026	22	RF	0.944	0.457	0.025	31
JFact	RF	0.967	0.417	0.014	116	RF	0.971	0.386	0.011	13	RF	0.968	0.412	0.013	36

Table 4.5: Comparison and selection of reasoner *best* trained runtime models across the different datasets

of the ontology features within the dataset. Based on these assessments, the second selection stage is conducted to outline the final reasoners best predictive models across all the datasets, i.e. $M^{Best}_{(T|R)}$. In Table 4.5, the latter ones are outlined in the gray color.

Important observations could be drawn from Table 4.5. First, we would highlight that the assessment values reported in Table 4.5 are very remarkable ones. Indeed, all of the indicated R^2 values are ranging over 0.944 and going up to 0.987. We should point out that the R^2 values on the regression models of the meta-reasoner R_2O_2 [KKL15] are below our values and vary from 0.73 up to 0.91. This would obviously provide an insight into how well our regression models for the 6 reasoners are fitting the data. This observation would further be confirmed by analysing the RMSE values. We recall that the smaller the RMSE value is, the more accurate the prediction model is. In our case, the RMSE was low on all the $M^{Best}_{(T|R)}$ models. More in details, the lowest reported RMSE is about 0.251 in the case of MORe, which means an average a mis-prediction of a factor of $10^{0.251} < 2$, whereas the largest RMSE is 0.455 in the case of FaCT++, i.e. a factor of $10^{0.455} < 3$. Worth of cite, the RMSE values on the regression models deployed in the portfolio-based algorithm selector SATzilla[9] [XHHLB08] were ranging in $[0.49, 1.01]$, and this was sufficient to enable SATzilla to win various medals in SAT competitions. Our RMSE values are far below those of SATzilla, and this is a valuable proof of the effectiveness of our runtime regression models. Throughout, it must be noted that for now our goal is not to outperform neither R_2O_2 nor SATZilla, but only to demonstrate that our reasoner runtime regression models could reach levels of confidence which are sufficiently satisfactory. The above inspections assert that the trained runtime predictive models can be used to accurately predict a given reasoner online classification time for a wide spectrum of OWL ontologies. Other valuable remarks about the

[9]SATZilla automatically selects the most efficient SAT solver by predicting its runtime for a given problem instance, using regression models.

SML techniques could be derived from Table 4.5. For all the reasoners and across all the employed feature selection methods, we can notice that Random Forest (RF) have dominated all of the evaluated regression algorithms, i.e Logistic Regression (LR), IBK and the SVM's (SMO and L2SVM). **RF** associated to **CFS** have delivered the reasoners topmost accurate runtime regression models, $M^{Best}_{(T|R)}$ for 5 out of the 6 reasoners. Only for FaCT++, the best model is the one trained by **RF** from the **RLF** based dataset. Furthermore, we would observe that in the case of HermiT, the best models trained respectively from **RAWD** and **CFS** based datasets have equal *RRS* scores. Here, the framework have prioritized the **CFS** based model since it employs only 16 out of the 116 ontology features. Indeed, we would remark that for all the reasoners, it exists a very tight difference, i.e. only in the second or the third decimal place, in the assessment values computed for the models across the dataset variants. This would suggest that the reduction of the ontology features space does not remarkably improve the accuracy of the regression models. However, we would recall that RF always prunes the feature dimension, regardless of whether the initial dataset is featured or not. Nevertheless, the little accuracy boosting obtained thanks to RLF and CFS methods is certainly worthwhile. Besides, when looking to the number of features employed in the final $M^{Best}_{(T|R)}$ models, we would remark the considerable reduction in the features space produced by these methods. Actually, at most only 31 features out of the 116 initial ones were held for the learning. The number of features required to train reasoners runtime regression models are far less than those needed for $M_{(OOH)}$ or $M_{(S|R)}$ models. Henceforth, thanks to such compact dimensions, we would gain more useful insights about those core ontology features imperatively required for the prediction of runtime.

Furthermore with a small number of ontology features to be measured, we assume that the time required to compute the prediction a particular reasoner over any ontology would also be reduced. However, this should be inspected more in details by examining the computational cost of each employed ontology feature. Figure 4.5 shows a scatter-plot of predicted log_{10} runtime vs. actual log_{10} runtime computed for each reasoner using its Random Forest $M_{(T|R)}$ model. We can see from these figures that the time required to classify most ontologies are predicted very accurately, and few cases are dramatically mis-predicted. In our opinion, it would be interesting to examine the mis-predicted cases across the reasoners. They might be the same ontologies, so by looking deeper into their characteristics, it would possible to gain more insights to their particular characteristics that we potentially did not cover by our feature suite. Such inspection would make it possible to incrementally improve our work. This is one of our future improvement plans.

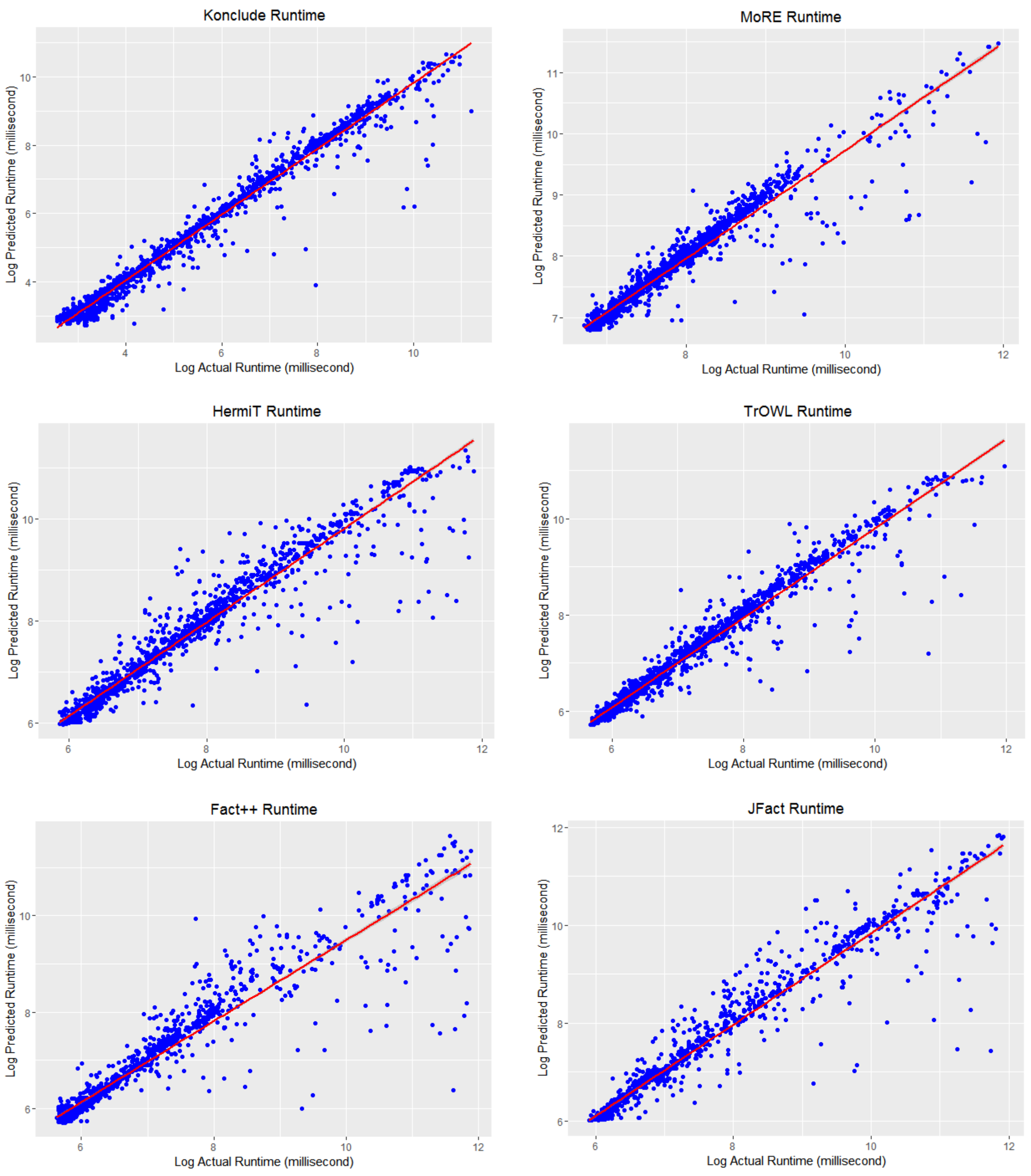


Figure 4.5: Scatterplots for Actual vs. Predicted log_{10} runtime of each examined reasoner.

$\cap$ Features	M_{OOH}	$\{M_{(S\mid R_i)}\}$	$\{M_{(T\mid R_i)}\}$
M_{OOH}	51	36	2
$\{M_{(S\mid R_i)}\}$	36	39	2
$\{M_{(T\mid R_i)}\}$	2	2	2

Table 4.6: The number of shared features across the different trained models.

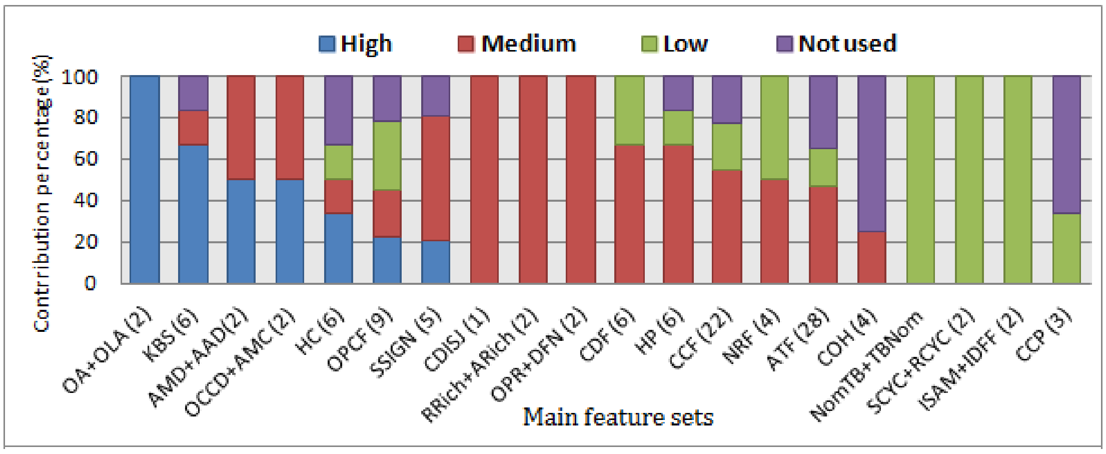
4.7 The Key Ontology Features For the Predictive Modelling

Our main idea to identify the key ontology features is to consider simultaneously multiple prediction problems about ontology reasoning and then examine their shared features. At the intuitive level, when we observe that particular features are identified as relevant predictors for different problems, we shall suggest that these features form a core group of good indicators of the ontology hardness against the reasoning tasks. Thus by focusing on these features the reasoner performances variability phenomena could further be tracked down. As a result of our learning process, 13 predictive models were approved and stored as meta-knowledge about the empirical robustness of the 6 examined reasoners for online ontology classification scenario. All of these models were trained from featured datasets. Within the latter ones, the ontology feature dimension was reduced to keep only the most relevant features and the least inter-correlated ones. In the previous Sections 4.5 and 4.6, we reported the number of the identified features for each approved model and highlighted that this value is close to the problem under examination. In the remainder of this section we will examine the distributions of these features and discuss some of them.

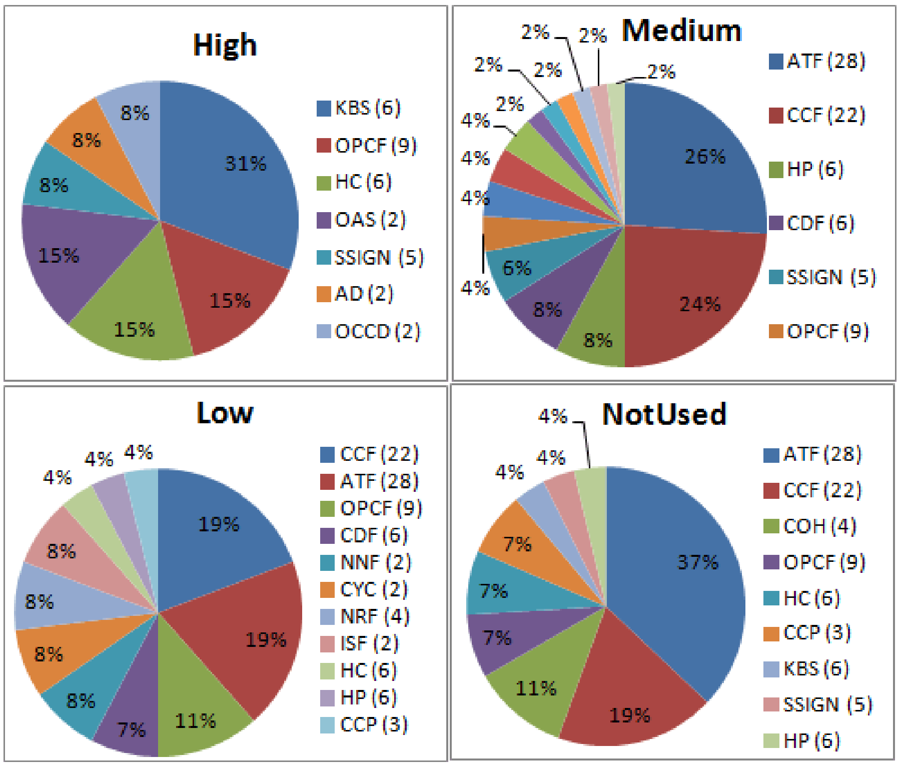
4.7.1 Ontology feature distribution across the models

We started our investigations by collecting some statistics in order to picture how the features are distributed over the inspected problems. In overall, 89 distinct features were used at least once in the approved models. Henceforth, 28 elements could be deleted from our feature space as they are found to be irrelevant for the examined problems. On the other hand, only 2 features were shared by all the 13 trained models. These are namely: the number of axioms within the ontology, i.e. the **OA** feature, and the number of the named classes, i.e. the **SC** feature. Hence, we would assert that for online scenario, the first indicator of the hardness of the ontology is the amount of knowledge it contains, otherwise its *size*. Seeking for further details, we computed the number of the shared features across specific combination of models. Table 4.6 summarized these inspections. $\{M_{(S|R_i)}\}$ and $\{M_{(T|R_i)}\}$ respectively stands for the set of all the trained models about reasoner termination state and runtime. We would observe that it exists 36 common indicators amongst models predicting the ontology hardness and the ones predicting the success of the 6 examined reasoners. However, these indicators are quite distinct from those employed to anticipate the reasoner efficiency. Actually, only two features are in common, those previously depicted, the SC and the OA. The latter ones are also the uniquely shared features across the different $\{M_{(T|R_i}\}$ models. Indeed, the examined reasoners follow different reasoning methodologies and implement different optimization techniques to overcome particular ontology complexity sources and thus, indicators of their efficiency would also vary. Nevertheless, thanks to our learned models, these indicators are no more unknown. One can identify them by looking into the local features, those employed in each reasoner $M_{(T|R)}$ predictive model. We remind that the local features were leveraged by feature selection methods as the most correlated ones with the runtime of the corresponding reasoner. In total, we listed 63 distinct features within the $M_{(T|R_i}\}$ models.

To gain further insights, we designed a scale of feature frequency across the final models. Let o be the number of occurrences of the feature f in the set of the 13 models. f has *high* frequency if $o \geq 9$, *medium* one if $8 \leq o \leq 5$, *low* in case $4 \leq o \leq 1$ and otherwise it is not used. In overall, we identified 13 highly frequent features, 50 were medium ones and 26 had low frequencies. As we can not list the frequencies of each of the 89 deployed features, we choose to report the distribution of the main feature groups, over the frequency bins. The groups are the ones previously introduced in Chapter 3, Section 3.2. Figure 4.6 illustrates the size of each group jointly with its name, and depicts in different colors the percentage of its elements that are part of each frequency bin. Then, Figure 4.7 enumerates the most frequent features together with the number of models were they has been deployed with respect to the problem nature. Our focus is put on this latter figure. We would observe that features characterizing the overall size of the ontology in terms of axioms, i.e. OA and OLA, then in terms of named classes SC are at the top of the frequent indicators. Indeed, they would pinpoint whether a reasoner would scales well or poorly regarding the growing size of the ontology. Equally important, we would remark that the number of classes with multiple direct ancestors, that we called the tangledness (HC_TG) of the class inheritance is a relevant feature for 12 out the 13 models. We should highlight that Kang et al. [KLK12] have also found that tangledness is a feature with high impact factor on reasoner runtime prediction. HC_MSB is another highly frequent structural feature indicating the maximal number of direct descendants in the class inheritance. This confirms the assumption we previously made in Subsection 3.2.3, which states that HC_MSB suggests the presence of highly central class and any change to this one will induce consequential impact to all the dependent classes which would alter the reasoner performances. The AAD feature records the average nesting of the TBox axioms. Sazonau et al. [SSB14] have already argued that this feature is a good indicator of the axiom entanglements. The SABx, STBx, RRBx and RABx are part of the KBS set and attended to characterize the size of



(a) Distribution of feature sets over the frequency bins



(b) Composition of the frequency bins in terms of feature sets

Figure 4.6: Discreption summary of the recorded frequency of each feature set by considering all the trained models.

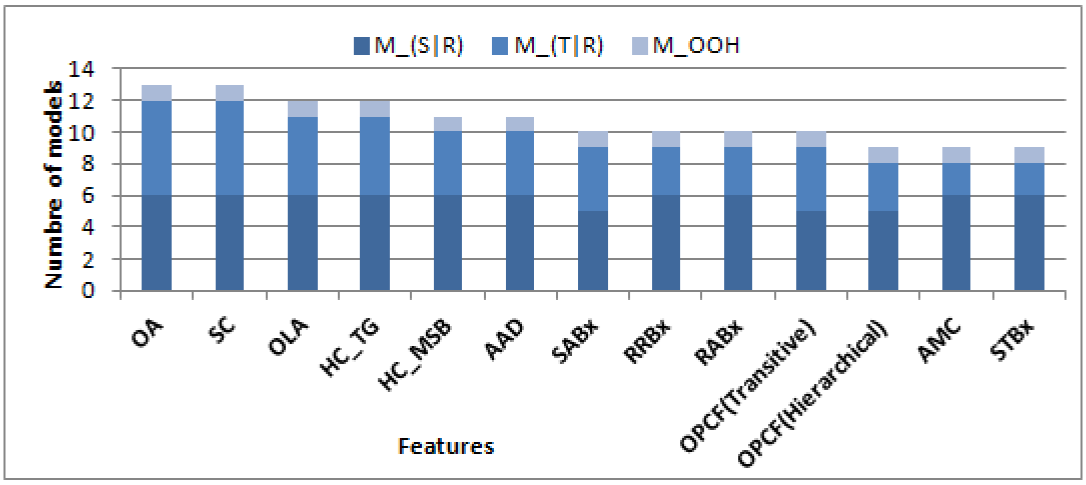


Figure 4.7: The set of the most frequent features distribution over the trained models.

the KB sub-parts. We would observe that both the size of the assertional part (SABx) and the ratio of its axioms are important features. This would further outline the general assumption that reasoner performances are sensitive to the ABox size. The remaining features are syntactic ones. AMC records the maximal number of the class constructors in one axiom. We should outline that the class constructor density, OCCD, is also a frequent indicator which has been used in 8 out of the 13 models. Finally, the OPCF of two particular object property characteristic, Transitive and Hierarchical are part of these highly used features. All of these findings agree with subsequent observations established in previous works about the significance of these particular features (cf. Section 3.2). However, we must stress that we are not concluding that the low frequent features are unimportant for the reasoners. Quite the contrary, these features could be highly specific to a particular reasoner and thus a strong indicator of its empirical behaviour.

Further inspections could be drawn from the reported results. We just provided one first step towards understanding the empirical behaviours of reasoners and ways to track down their robustness based on particular ontology features. Indeed, we would recommend the set of the highly frequent features as a starting point to conduct further improvement in ontology modelling as well as reasoning optimization. Nevertheless, it is important to point out that each reasoner has its particular ontology key features, likely to have the strongest impact on its performances. The latter ones could be retrieved from the reasoner predictive robustness profiles which henceforth, will be stored in a dedicated reasoner meta-knowledge base.

Feature	**Frequency**	**List of models**
HC_MHD	8	$M_{(OOH)}$, $M_{(S\|R)}$\{Konclude, MORe, HermiT, TrOWL, FacT++, JFacT\}, $M_{(T\|R)}$\{Konclude\}
HC_COH	7	$M_{(S\|R)}$\{MORe, HermiT\}, $M_{(T\|R)}$\{Konclude, MORe, HermiT, TrOWL, JFacT\}
Onto_COH	0	-
RCYC	4	$M_{(OOH)}$, $M_{(S\|R)}$\{MORe\}, $M_{(T\|R)}$\{MoRE, HermiT\}
SCYC	1	$M_{(T\|R)}$\{MORe\}
UI (CCP)	1	$M_{(T\|R)}$\{MORe\}
EUvI, CUvI (CCP)	0	-

Table 4.7: Distribution of computationally expensive features over the trained models.

4.7.2 Distribution of the computationally expensive features

It is worth to note, throughout this analysis, we did not distinguish between the computationally expensive features and the efficiently computable ones. This is because our study would be conducted offline with sufficient computational resources in order to acquire enough knowledge about reasoner empirical behaviours, in the the most reliable way. Hence, we are still not constrained by time and mostly, we did not yet designed the time-sensitive system where the trained predictive models could be employed. When such requirement is imposed, we can examine the trained models and check whether is possible to remove or to set as missing values the computationally expensive features, without any lost in the model accuracy. To get a precise idea whether such a scenario might take place, we listed the models where the computationally expensive features are involved. Table 4.7 summarizes the results. In overall, 5 out the 8 expensive features were employed in the final predictive models with different degree of frequency. We would remark that the depth of class inheritance (HC_MDH) is shared by all the models predicting the success of the examined reasoners and relevant to predict the runtime of Konclude. On the other hand, the cohesion of the class hierarchy seems to be important feature to runtime prediction of 5 out the 6 examined reasoners. Besides, the ratio of cyclic class is correlated to MORe and HermiT runtime.

We also examined the omission cost of the expensive features. More specifically, for each learning problem, we removed from the initial dataset (RAWD) those features known to be expensive ones, then we trained a new predictive model using the previously approved best combination (feature selection method, supervised learning algorithm). Hence, by comparing the values of the quality scores, i.e. the old versus. the new ones, we can characterize the impact of the omitting these features.

	$M_{(S\|R)}$						$M_{(T\|R)}$					
	#F		**F1**		**MKS**		**#F**		R^2		**RMSE**	
Reasoner	old	new	old	new	old	new	old	new	old	new	old	new
Konclude	53	53	0.974	**0.973**	.653	**0.644**	20	18	0.987	0.987	0.334	**0.336**
MORe	65	62	0.960	**0.958**	0.827	**0.820**	28	24	0.963	0.963	0.251	0.250
HermiT	67	66	0.972	**0.966**	0.889	**0.866**	16	14	0.968	**0.967**	0.383	**0.386**
TrOWL	55	54	0.948	0.949	0.787	0.793	16	29	0.975	**0.972**	0.291	**0.310**
FaCT++	63	62	0.952	0.955	0.881	**0.878**	31	31	0.944	0.944	0.457	0.457
JFact	60	58	0.952	**0.950**	0.904	**0.901**	13	15	0.971	0.972	0.386	0.382

Table 4.8: Examining the impact of omitting the expensive features on reasoner models.

	#F		**F1**		**TP (hard)**	
Model	old	new	old	new	old	new
$M_{(OOH)}$	51	50	0.947	**0.944**	891	**889**

Table 4.9: Examining the impact of omitting the expensive features on the quality of $M_{(OOH)}$.

Table 4.8 illustrates the results computed over the reasoner robustness models. It recalls the previously reported quality assessment values of the predictive models achieved when the full set of features is considered (e.g. designed by *old* in Table 4.8), then the *new* computed values when only the cheap features are maintained. Table 4.8 also reports the old, then the new feature dimension (e.g. the **#F** column). Similarly, Table 4.9 shows the impact of omitting the expensive features on the predictive quality of the ontology overall hardness model, $M_{(OOH)}$. In both tables, cases where the assessment values have known a decrease are highlighted in bold face. It is important to note that the RMSE is negatively oriented score, hence an increase in its values would indicate a loss in the quality of the trained model. Interestingly, we noticed a gain in the quality of some models, according to one or to all the assessment measures. These cases are outlined in the gray color. Indeed, the features are correlated in complex ways, hence there will often be other very different sets of features that would achieve similar performances or even better ones. Throughout the reported values, we would assert that, in overall, omitting the expensive features are not dramatically damaging the predictive quality of the various trained models about the success or the runtime of reasoners. In most cases, slight decrease is been noted affecting only the third or second decimal place of the assessment scores. By examining Table 4.9, we can remark the decrease in the number of the correctly predicted hard ontologies. This would suggest that the removed features, i.e. the depth of the class hierarchy HC_MHD and the ratio of cyclic classes RCYC are important indicators of the empirical hardness of the ontologies.

4.8 Conclusion

The research questions, we discussed throughout this chapter are at the crossroad of two fields: the Semantic Web and the Machine Learning. Our main purpose was to employ supervized machine learning techniques to be able to track down the variability of DL reasoner empirical performances. We put our focus on two learning problems: first, predicting the overall hardness of OWL ontologies regarding a group of reasoners; and then, anticipating a single reasoner robustness when inferring ontologies under some online usage constraints. To fulfill this goal, we established a generic supervized learning framework, which integrates the ontology features we previously proposed in Chapter 3. The framework is able to carry different predictive tasks about the ontology reasoners. Owing to this solution, we succeeded to establish highly accurate predictive models about the overall empirical hardness of ontologies, as well as the empirical correctness and efficiency of DL reasoners. Hence, we composed predictive robustness profile for each examined reasoner. To the best of our knowledge, we are the first to attempt to learn more than one reasoner evaluation criterion. We believe that our assembled reasoner profiles and ontology overall hardness models are the first steps towards building a concise knowledge base pooling data related to existing modern DL reasoners and OWL ontologies. We trust that this kind of knowledge would provide assistance for reasoner and ontology designers for better and more easier inspection of their realisations. Further improvement of our framework could be considered to get better predictive models in less training time. For instance, we would investigate more recent discretization techniques. It would also be interesting to examine more state-of-art solutions to handle the dataset imbalance issue, such as over/under sampling of instances or the cost-sensitive learning. On the other hand, we assume that reducing the overall training time would require more meta-knowledge about the learning algorithms. Indeed, we can consider that our seek to identify the most appropriate learning techniques with respect to our training datasets, is a kind of algorithm selection problem. Hence, examining the meta-learning solutions [BGCSV08] is certainly an appealing possibility. However, it should not be overestimated, since we are not sure that the existing techniques would fit to our various particular requirements and without any additional computing cost.

In the remaining of this thesis, we will introduce two area of applications were the established predictive models. First, we will be extending our learning steps by a ranking stage where reasoners could be compared based on their predicted robustness. Such ranking would made it possible to recommend the most robust reasoner for any given ontology. Then, we will introduce an extraction approach of the most computationally demanding sub-parts, from an ontology predicted to be hard.

CHAPTER 5

ADSOR: Advising System Over the Ontology Reasoners

Contents

5.1 Introduction

The main goal of this thesis is to bring solutions to cope with the reasoner performances variability phenomena and hence, advance the state of the art in the empirical analysis of DL reasoners. Our final goal is to provide simple, efficiently computable guidelines to novice and expert users about, on the one hand the empirical *hardness* of OWL ontologies, and on the other hand, the empirical *robustness* of modern DL reasoners. The guidelines are basically a set of predictions about the expected behaviours of an individual reasoner or a ranking of group of reasoners given an input ontology.

To achieve these purposes, we suggest to develop an intelligent system to provide user support when looking for guidance over ontology reasoners. We called this system, **ADSOR**, standing for "*Advising System Over the Ontology Reasoners*". ADSOR provides two types of support for an end-user: *(i)* it suggests a ranking of reasoners according to their predicted performances over the user input ontology. This ranking is provided in order to help user selecting the most adequate reasoner to their ontology based application; and *(ii)* it performs an extraction of the most challenging sub-parts from an empirically hard ontology. These sub-parts are starting points provided to the user to help him conduct more focussed analysis about its ontology hardness factors.

A detailed description of the main component of ADSOR, together with a rigorous experimental analysis of their respective performances are provided in this chapter. In Section 5.2, we start by depicting the overall conceptual architecture of ADSOR. Then in Section 5.3, we address the issue of automated reasoner selection and we introduce a novel framework of supervised ranking of DL reasoners. More specifically, we will analytically and empirically study two ranking strategies and put then into comparison. Finally in Section 5.6, we will describe our ontology *hard* modules extraction approach and its empirical evaluations.

5.2 Overall Description of the ADSOR System

In this section, we describe the general architecture of the ADSOR system and we shortly outline the prime features of its basic components. ADSOR follows a meta-learning approach (cf. Chapter 2) to provide relevant support for end-users in the field of ontology reasoning. K. A. Smith-Miles argued in [SM09] that meta-learning techniques, which were primary designed to track the behaviours of machine learning algorithms, can easily be generalized to a broad range of disciplines. The author asserted that this generalization could be achieved provided that the following four requirements are fulfilled: *(i)* supplying a large amount of problem instances of diverse complexities; *(ii)* assembling a rich set of existing algorithms with diverse behaviours; *(iii)* introducing clear metrics of algorithm performance; and *(iv)* employing effective features of instances that can be calculated and that correlate with hardness/complexity of these instances. In chapter 4, we have already satisfied these requirements and proved that we can predict at a high degree of confidence the empirical robustness of DL reasoners. By now, we are aiming to extend this preceding work to solve the problem of automatic selection of DL reasoners for online classification of OWL ontologies. Furthermore, we are determined to go beyond the selection purpose and find ways to apply the learning techniques to gain insights about the ontology hardness factors by extracting its computationally challenging sub-parts.

As it is common in the architectures of meta-learning systems [Van11], ADSOR operates within two periods: *(I)* an *offline stage* for knowledge acquisition about a set of reasoners and ontologies; and *(II)* an *online stage* which offers advising for a user seeking support to achieve a robust reasoning over an OWL ontology. Figure 5.2 depicts the main conceptual components of the ADSOR system and illustrates their organisation as well as their various interactions within each operating stage.

Offline stage: the knowledge acquisition mode. The main purpose of this stage is to train predictive models about various learning problems around the ontology reasoning. The models are acquired by an intelligent exploration of data describing reasoner empirical performances over a large set of ontologies. Hence, collecting these empirical data is the first step in the knowledge acquisition stage. ADSOR employs the publicly available ORE evaluation framework [GBJR+13] to carry on

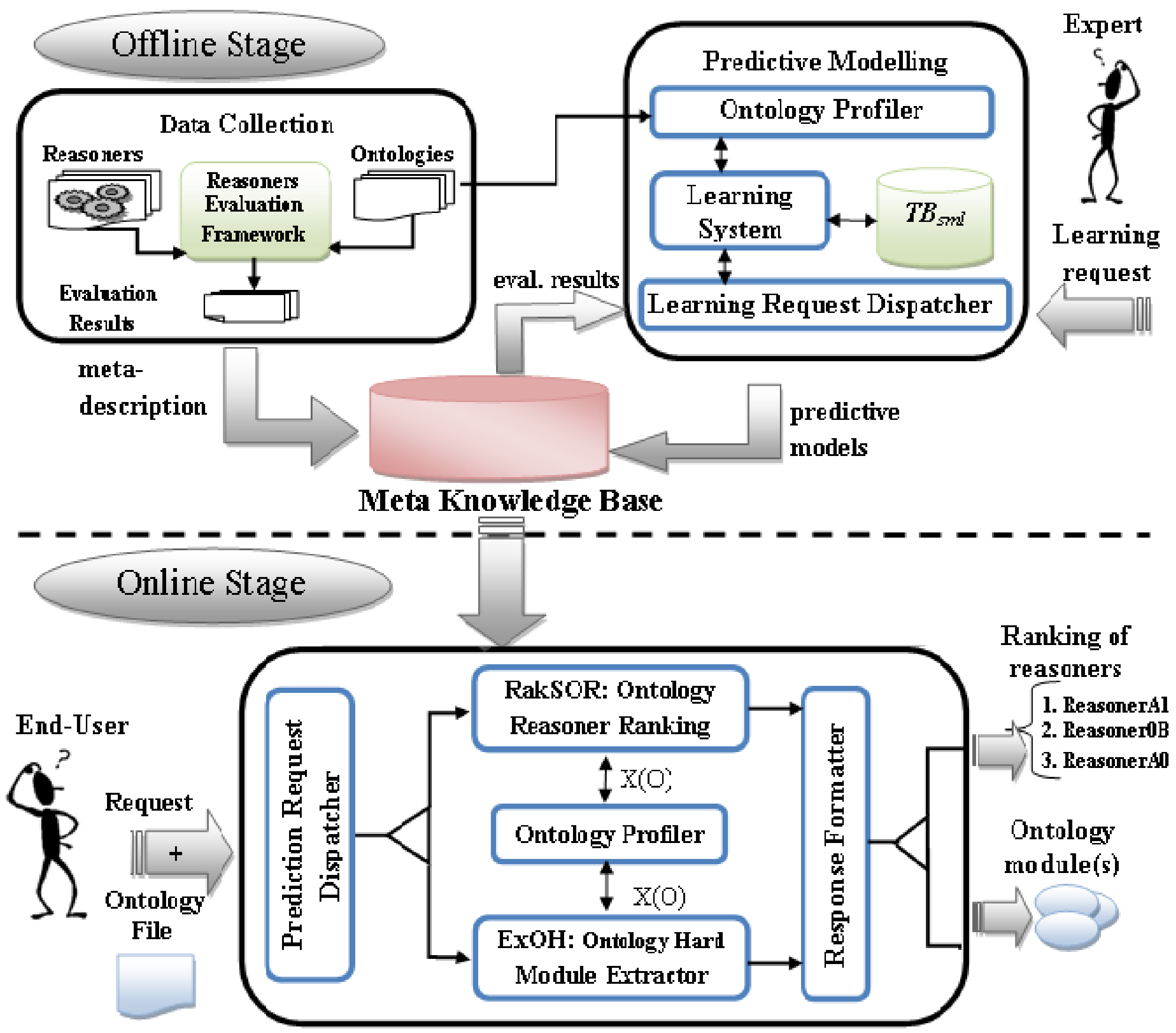


Figure 5.1: The general architecture of the ADSOR system.

regular evaluations about modern DL reasoners, and thus delivers the needed data on which to base the learning process. The latter one is triggered following the request of an expert user and achieved by our *learning system*. In Chapter 4, we have introduced some of the functional characteristics of this system. Further details about its operating mode will be provided in the remainder of this chapter. Another equally important components is the *ontology profiler*, which we have established in Chapter 3. It characterizes the design complexity of the studied ontologies using a rich collection of efficiently computable features. Some of the trained models produced by the learning stage were previously described in Chapter 4 such as the reasoner predictive robustness profiles and the ontology overall hardness models. According to the respective authors of [SM09, Van11], combining the features with the performance metrics across a large number of ontologies inferred by different reasoners creates a comprehensive set of meta-data about reasoner empirical performances. These meta-data together with the trained predictive models represent the prior (background) knowledge on the involved DL reasoners. The learned expertise is stored in machine-interpretable format within a specific database, commonly called the *Meta-Knowledge Base*.

Online stage: the advisory mode. ADSOR proposes two kinds of computed advices based on the acquired knowledge about the empirical robustness of DL reasoners and the empirical hardness of OWL ontologies. The advices are requested by an end-user about an input ontology. In the following, we briefly describe the main online components of ADSOR, together with the advices they respectively provide:

RakSOR: Ranking of Ontology Reasoners. This component advices the potential users regarding the *best* available reasoners according to the nature and the design characteristics of their ontologies. The provided advices are in their basic form, i.e. a ranking of reasoners in the most preferred order. Reasoner robustness is the main ranking criterion. To put these proposals into application, we formulated a novel supervised ranking framework of ontology reasoners. We investigated two distinct ranking strategies: a first ranking method based on single label prediction per candidate reasoner, and a multi-label based method which predicts the full reasoner ranking.

ExOH: Extraction of Ontology Hard Modules. This component helps users to gain insights about the empirical hardness of their ontologies. Indeed, whenever an ontology is predicted as a hard to manage with respect to a group of reasoners, a user will be looking for explanation about the potential causes of this hardness. In this case, we suggest to provide her/him the possibility to extract those ontology sub-parts that are potentially the most computationally demanding ones. These sub-parts are commonly called as the ontology *hotspots* [GPS12]. Our approach mainly relies on the graph representation of the ontology atoms [DPSS11]. It employs the locality based technique [GHKS08] to extract the ontology modules. The identification of the hard ones is guided by our predictive model of the ontology overall hardness.

In the remainder of this chapter, we will describe more in details the design at the base level of the ranking and the extraction respective components.

5.3 RakSOR: Ranking and Selection of Ontology Reasoners

In this section, a detailed depiction of the core design of the RakSOR component will be drawn up. We will start by setting up the issue of reasoner ranking. Then, we will specify the preference rules according to which the ranking will be computed. Later, we will separately introduce two supervised ranking strategies and empirically put them into comparison.

5.3.1 Problem formulation

Over the last decade, several ontology reasoners have been proposed to overcome the computational complexity of inference tasks on expressive ontology languages such as OWL 2 DL. Nevertheless, it is well-accepted that there is no outstanding reasoner that can outperform in all input ontologies. Thus, an algorithm selection problem [Ric76] have emerged in this field of study. Indeed, deciding the most suitable reasoner for an ontology based application is still a time and effort consuming task. In Chapter 2, we established a review on existing works, which have discussed the issue of algorithm selection in different disciplines. We have distinguished *two* main approaches: *(i)* build an algorithm portfolio and learn to select the best performing algorithm for any input problem; *(ii)* provide a ranking of the most suitable algorithms to solve an input problem and allow the user to make the selection. While the first approach is an optimisation solution to build more performing systems, the second approach is a user oriented one aiming to provide a decision support in the task of algorithm selection. In this thesis, we chose to work with the second interpretation of the algorithm selection problem. Our main purpose is to automatically rank a set of reasoners in a most-preferred order for any input ontology, and by relying on predicted performances. Afterwards, a user may decide or not to follow the advice exactly as given. Indeed, a user may choose to continue using her/his favourite reasoner, if its performance is slightly below the topmost one in the ranking. We should recall that our main goal is to provide the user with basic information about the expect behaviours of the existing reasoners regarding its ontology. Hence, we will advise him about what reasoner to choose and what to avoid. By consequence, even those reasoners expected to fail the reasoning task are included in the ranking. In our opinion, such a configuration would be advantageous to reasoner designers. Indeed, it pinpoints the unsuccessful reasoners and provides a starting material, i.e. the input ontology, to carry further investigations about the causes of this failure. This would enhance more improvements in the design of reasoners.

Given this description, we can establish some mandatory requirements to be fulfilled when designing our ranking solution. They are listed in what follows:

1. The reasoner ranking method should consider the particular features of the user input ontology.
2. The reasoner ranking method should be a supervised one based on prediction.
3. The ranking is computed over all the available reasoners whether they can handle or not the input ontology. However, the system must pinpoint the successful ones.
4. The ranking is computed with respect to the reasoner robustness judgement constraints previously

established in Chapter 4, Section 4.2. These constraints sketch an online usage scenario where DL reasoners are requested to classify OWL ontologies.

5. Reasoner robustness is the main ranking criterion. This entails that the ranking should consider both the correctness and the efficiency of the reasoners.
6. No user preferences are to be taken into account in the ranking process.

Formally, a ranking represents a *preference function* over a set of alternatives (item, objects, etc.). In our context, the alternatives are some set of DL reasoners, recognized as promising candidates. Hence, given a particular set of alternative reasoners, denoted by $\mathbf{R} = \{R_1, \ldots, R_m\}$, a ranking is defined as a vector $\sigma = (\sigma_1, \sigma_2, \ldots, \sigma_m)$, where σ_i is the rank of reasoner R_i and lower ranks are interpreted as preferable ones. Depending on the chosen preference rules, the ranking may either represents a total or a partial order [Pri02]. Roughly speaking, a total strict order is a binary, asymmetric, transitive and complete relation, denoted by $\prec$, which states that for every pair R_i, $R_j \in \mathcal{R}$ with $i \neq j$, either R_i is preferred to R_j, denoted $R_i \prec R_j$, or R_j is preferred to R_i. Besides, if R_i is preferred to R_j and R_j is preferred to R_k with $k \neq j$, then R_i must be preferred to R_k. On the other hand, a partial strict order is a generalization of the total strict order. However, it does not stick to the completeness property. More specifically, if for two alternatives R_i and R_j, neither $R_i \prec R_j$ nor $R_j \prec R_i$, then these alternatives are said to be incomparable, i.e. $R_i \prec\succ R_j$. The preference rules are closely depending on the approved ranking criteria. For instance, when the reasoners are ranked according to their runtime then it is obvious that reasoners achieving shorter computational time are the preferred ones and the ordering is a strict total one. However in our case, we chose to rank the reasoners according to a couple of criteria: the correctness and the efficiency. Thus, an interrogation would raise about how to combine these criteria and which would be the nature of the produced order?

Choosing the surprised ranking method is an equally important point in the design process of our solution. Based on the literature review scrutinized in Chapter 2, we can assert that there is two main ranking strategies which employs supervised learning techniques:

1. *Ranking based on single label prediction*: after computing the features of the input problem instance, the performances of each alternative algorithm is predicted separately using its corresponding pre-established predictive model and according to these features. Hence, a single target value (continuous variable or discrete label) is predicted per algorithm and then the total set of collected values are compared in order to produce the final ranking. This strategy has successfully been applied to rank SAT solvers for portfolio construction [XHHLB08]. It has also been employed in meta-learning systems in order to produce recommendations of machine learning algorithms. A rigorous survey of this kind of works is provided in [PdSL11].

2. *Ranking based on multi-label prediction*: the main purpose of this strategy is to train a model, called a *ranker*, capable to predict the complete ranking of the alternatives, based on the feature description of the input instance. This kind of learning is a special case of multi-target prediction, known as *label ranking* or more generally *multi-label ranking*. A large body of literature related to this field of study exists and is scrutinized in [FH10]. Label ranking has been applied to improve meta-learning [Sun14] and more recently employed in SAT solver portfolios [OHL15].

Each of the above described strategies has known a success across different disciplines. However, there is no agreement about which one is the most adequate or the most performing strategy. Adjudging them is closely depending on the problem nature and the context of the learning. Hence, we decided to investigate both of them and to compare their performances analytically and empirically.

Based on this discussion, it is obvious that to implement the RakSOR component, one has to consider two important questions: which preference rules to apply in order to take into account the information regarding the reasoner correctness and efficiency? and which supervised ranking strategy which will best satisfy our requirements? In the remainder of this section, we will consecutively bring response to each of these questions.

5.3.2 Ranking criteria and preference order

In Chapter 4, we discussed at length the ways to adjudge the robustness of DL reasoners. We shall recall that a reasoner is assessed according to the correctness of its derived results and the time, it spent to compute them. However, the correctness is not numerically quantified. A reasoner result would be correct or not and computed in time or not. Hence, we specified a first ordinal criterion which would categorize the reasoners into four groups according to their termination state. To gain more insights, we added a second criterion which track the exact time spent by a reasoner to derive *correct* results. Henceforth, a ranking of reasoners with respect to an input ontology should respect this specification.

Rich plethora of approaches and algorithms were introduced in the literature to handle multiple criteria when ranking items or objects. When the criteria are attributes describing the alternatives, then the ranking task could be reduced to a Multiple Criteria Decision Making (MCDM) problem [GEJ16]. The main goal in this field of study is to build a preference relation which takes into consideration the multiple attributes of the alternatives. Hence, the ranking are made on the basis of preferences expresses on each attribute and eventually "inter-attribute". Our reasoner ranking problem could be seen as one MCDM problem. Therefore, we studied some of the prime works in this field of research, and examined how well they could fit our requirements. The Outranking algorithms and the Utility

Theory based techniques (MAUT) are the two main MCDM approaches. A detailed survey about these works is provided in [GEJ16]. We first discarded the outranking techniques since they are heavily based on user preferences and not adapted to problems which have few ranking criteria, like in our case. The MAUT approaches mainly employ aggregation operators like the addition or the multiplication to combine the multiple criteria into a single utility function. The specification of the aggregation may rely on the user preferences. Despite, the simplification that an aggregation function would brought to our ranking problem, defining a proper function that would handle our criteria is a quite complex task since our criteria have different units. Actually, our first criterion is a discrete variable while the second one has continuous values. Besides, this latter one is conditionally depending on the first criterion.

One further possibility to remedy to the reasoner ranking problem is to apply the methodology of partial order ranking and its graphical representation (Hasse diagrams) [BCLS06]. Partial ordering is a parameter free, transparent methodology and quite adapted to ranking problems where the criteria does not exclude each other. Indeed, in our opinion, a total order of the reasoners can be misleading. This is because some alternatives could be incomparable with respect to the input ontology. For instance, reasoners predicted to fail in processing the ontology and share the same cause of failure are incomparable. More specifically, we choose to work with a special case of partial orders: the *bucket order rules* [FKM+06]. Before describing how these rules will be interpreted and applied in our study context, some basic notions will be recalled.

5.3.2.1 Preliminaries on bucket order

Following Fagin et al. [FKM+06], a *bucket order* is a special case of partial order. It could be seen as a total order over a set of buckets (groups) in such a way that objects are tied (equally ranked) within the buckets. Formally, let $\mathbf{B}$=$\{\mathcal{B}_1, \dots, \mathcal{B}_t\}$ be a set of non empty *buckets* that forms a partition of a set $\mathcal{A}$ of alternatives[1], i.e. $\bigcup_{i=1}^{t} \mathcal{B}_i = \mathcal{A}$ and $\mathcal{B}_i \cap \mathcal{B}_j = \emptyset$ if $i \neq j$, and let $\prec$ be a strict total order on $\mathcal{B}$, i.e. $\mathcal{B}_1 \prec \dots \prec \mathcal{B}_t$, we say ($\mathcal{B}$, $\prec$) forms a *bucket* order on $\mathcal{A}$. $\prec$ is a transitive, strict binary relation such that for any $x, y \in \mathcal{A}$, x precedes y denoted by $x < y$, if and only if there are i, j with $i \neq j$ such that $x \in \mathcal{B}_i$, $y \in \mathcal{B}_j$ and $\mathcal{B}_i \prec \mathcal{B}_j$. If x and y are in the same bucket, then x and y are incomparable and said to be "tied". A *totally ordered set* is a special case of a bucket ordered set in which every bucket is of size 1.

Ranking with ties. A *ranking σ with ties* over $\mathcal{A}$ is associated with a *bucket order* ($\mathbf{B}$, $\prec$) over $\mathcal{A}$ as $\sigma = \mathcal{B}_1 \prec \dots \prec \mathcal{B}_t$, where i is the rank of $x \in \mathcal{A}$, i.e. $\sigma(x) = i$, if and only if $x \in \mathcal{B}_i$. $\forall x' \in \mathcal{B}_i \wedge x \neq x'$, then x, x' are equally ranked $\sigma(x) = \sigma(x') = i$ and hence a tie. When x precedes y, i.e. $\sigma(x) < \sigma(y)$,

[1]The alternatives are the objects, or elements to be ranked.

this means a preference for x over y.

Pair ordered matrix Pandit et al. [PKK11] have further specified that given a bucket order ($\mathbf{B}$, $\prec$), a pair ordered matrix capturing the pairwise preferences over the elements of $\mathcal{A}$ could be set up. The latter one is designed by $\mathbf{C_B}$. It is a $|\mathcal{A}| \times |\mathcal{A}|$ matrix such for each $x_i, x_j \in \mathcal{A}$, $\mathbf{C_B}(i,j) = 1$ if $x_i < x_j$. Besides, $\mathbf{C_B}(i,j) = 0.5$ if x_i and x_j belongs to the same bucket and $\mathbf{C_B}(i,j) = 0$ if $x_j < x_i$. By convention, $\mathbf{C_B}(i,i) = 0.5$ and $\mathbf{C_B}(i,j)$ satisfies that $\mathbf{C_B}(i,j) + \mathbf{C_B}(j,i) = 1$.

5.3.2.2 Applying the bucket order rules to reasoner ranking problem

According to the termination state (real or predicted state) of each examined reasoner, four possible reasoner groups could be identified: "Success" (**S**), "Unexpected" (**U**), "Timeout" (**T**), "Halt" (**H**) (Chapter 4, Subsection 4.2.2). These groups are disjoint ones since a reasoning task would have a unique termination state when achieved by a particular reasoner over a given ontology. By consequence, the union of the above mentioned groups is equal to the whole set of reasoners, i.e. **R**. Therefore, we can assert that these groups form a partition of **R**. On these basis, we defines the buckets set $\{\mathcal{B}_S, \mathcal{B}_U, \mathcal{B}_T, \mathcal{B}_H\}$, which respectively matches $\{S, U, T, H\}$ groups. In addition, a precedence order relation $\prec$ over the described buckets can easily be specified. In our opinion, the most straightforward order is $\mathcal{B}_S \prec \mathcal{B}_U \prec \mathcal{B}_T \prec \mathcal{B}_H$. Intuitively, reasoners belonging to $\mathcal{B}_S$ are the most preferred ones as they can derive correct results within the fixed time limit. The reasoners of $\mathcal{B}_U$ could achieve the reasoning task within the time limit but the correctness of their results was not approved by our correctness checking method. Therefore, we think that they are much preferred than reasoners falling in the $\mathcal{B}_T$ bucket, which would not deliver any result to be assessed. Of course, the worse reasoners are in $\mathcal{B}_H$ bucket, as they are not even capable to process the ontology.

Given this specification, two particular cases would emerge. First when $|\mathcal{B}_S| = 0$, this means no reasoner has been identified as robust over the input ontology, consequently no effective reasoner could be suggested to the user. Despite this fact, reasoners within the remaining ordered buckets could still be provided to the user as an indication that their ontology might be hard to manage w.r.t the available reasoners. Furthermore, the reasoners will be listed according to the nature of their failure which would be a valuable information for users looking to process additional inspections. Another interesting case is when $|\mathcal{B}_S| > 1$. Here more than one alternative reasoner, eventually all of them, are found to be robust[2] over the input ontology and could be suggested to the user. Nevertheless, we think that the latter one would rather prefer to get informed about the most *adequate* reasoner amongst the

[2] We recall that a robust reasoner over an input ontology is the one which have achieved the classification task within the time limit and delivered correct results.

set of the *robust* ones. To achieve this purpose, we need to break the tie at the top of the ranking list by referring to our second ranking criteria: the runtime of a correctly achieved ontology classification task. On the basis of the reported (or predicted) computational time, the reasoners belonging to $\mathcal{B}_S$ bucket could be linearly ordered by a simple decreasing sort over their runtimes: the more efficient a robust reasoner is, the smaller would be its rank value. Henceforth, $\mathcal{B}_S$ will be partitioned such that every *successful* reasoner will have its own corresponding bucket. Given m alternative reasoners, of which k reasoners were found to be robust w.r.t an ontology $\mathcal{O}$, i.e $k =\mid \mathcal{B}_s \mid$, then the bucket order over this set of alternatives could be modelled as follows:

$$\overbrace{B_1 \prec \ldots \prec B_k}^{\mathcal{B}_S} \prec \mathcal{B}_U \prec \mathcal{B}_T \prec \mathcal{B}_H \tag{5.1}$$

where $|B_i| = 1$ such that $1 \leq i \leq k$ are the single element buckets corresponding to the successful[3] reasoners. By consequence, we can assert that $|\mathcal{B}_U| + |\mathcal{B}_T| + |\mathcal{B}_H| = (m - k)$. Furthermore, the rank value of reasoners within the buckets, $\mathcal{B}_U, \mathcal{B}_T, \mathcal{B}_H$, will be be updated according to the size of the $\mathcal{B}_s$. For instance, let $k =\mid \mathcal{B}_s \mid$ and $\mathcal{B}_U \neq \emptyset$, then the ranks are updated as follows: $\forall R_u \in \mathcal{B}_U, \sigma(R_u) = k+1$ and so on. To meet the aforementioned requirements, the reasoner ranking with ties according to the bucket order principals and with respect to their classification performances over an ontology $\mathcal{O}$ is defined in what follows. It is to be noted that $CT(R_i, \mathcal{O})$ denotes the computational time achieved by the reasoner R_i.

Definition 26 (Reasoners ranking with ties) *Given a set of alternative reasoners* $\mathbf{R} = \{R_1, \ldots, R_m\}$ *and an ordered bucket partition over* $\mathbf{R}$*, i.e.* $\mathbf{B} = \mathcal{B}_S \prec \mathcal{B}_U \prec \mathcal{B}_T \prec \mathcal{B}_H$*, a reasoner* R_i *precedes a reasoner* R_j*, i.e.* $\sigma(R_i) < \sigma(R_j)$*, if and only if,* $R_i \in \mathcal{B}_i$ *and* $R_j \in \mathcal{B}_j$*, such that* $\mathcal{B}_i \prec \mathcal{B}_j$ *or* $\mathcal{B}_i = \mathcal{B}_j = \mathcal{B}_S$ *and* $CT(R_i, \mathcal{O}) < CT(R_j, \mathcal{O})$*. On the other hand,* R_i *and* R_j *are tied, i.e.* $\sigma(R_i) = \sigma(R_j)$ *, if and only if,* $\mathcal{B}_i = \mathcal{B}_j \neq \mathcal{B}_S$*.*

Based of the preference rules established in Definition 26, the pair ordered matrix $\mathbf{C_B}$ over the alternative reasoners could be encoded following the description made in Subsection 5.3.2.1. We should highlight that the above preference rules could be applied to data, describing the real reasoners' performances, or the predicted ones. In the next sections, we will describe how to predict the rank of reasoners according to the established rules and for any input ontology. In the forthcoming sections, we will investigate two distinct rank prediction approaches and describe their respective experimental evaluations.

[3]Otherwise, the robust ones.

5.4 Reasoner Ranking based on single label prediction approach

In Chapter 4, we already established the concept of reasoner predictive robustness profile and specified ways to assemble it from empirical data. Roughly speaking, a profile is made up from a couple of predictive models: a multi-class model predicting the termination state of the reasoner with respect to given ontology, and a regression model estimating the runtime of this reasoning task when the correctness of the results is granted. In the remainder of this section, we will describe how these profiles would be put forward in order to compute a ranking with ties of DL reasoners by applying the bucket order principals.

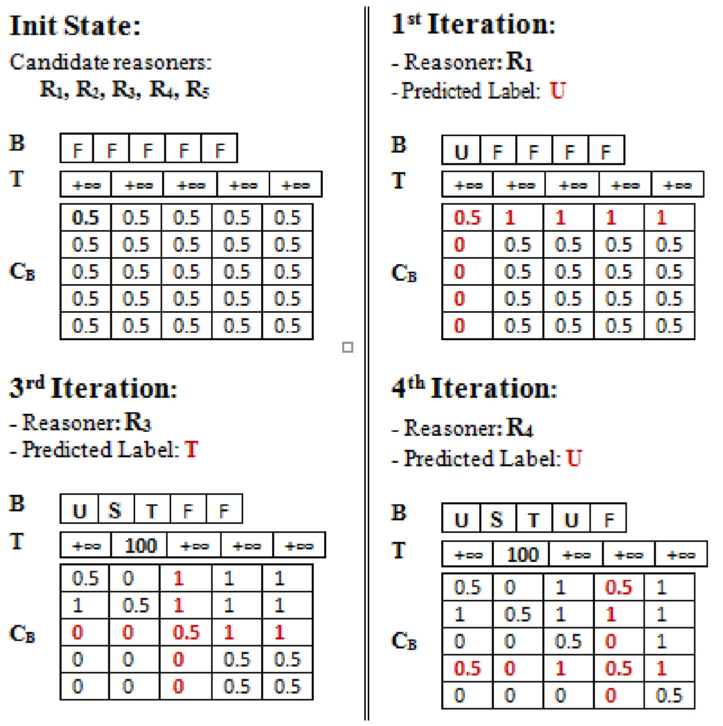
5.4.1 Design description of the ranking method

Once a user introduces her/his ontology O_u, RaKSOR, the reasoner ranking component, computes the feature vector of this ontology, $X(O_u)$, and loads the stored reasoner profiles from the meta-knowledge base. Then, the system predicts the output quality of each reasoner against this ontology. Afterwards, the alternative reasoners are ranked according to these predictions. The ranking steps are carried out by applying Algorithm 3. The main purpose of this algorithm is to partition the set of alternative reasoners into ordered buckets $\mathcal{B}_S \prec \mathcal{B}_U \prec \mathcal{B}_T \prec \mathcal{B}_H$ by considering their predicted performances and accordingly encode the *reasoner pair order* matrix $\mathbf{C_B}$ ($|\mathbf{P}| \times |\mathbf{P}|$ matrix). The algorithm outputs the ranking of reasoners together with their respective predicted performances, i.e. the termination state and potentially the exact estimated runtime. Hence, the ranking could be explained and justified. Different output formats could be provided: a simple string list or an advanced Hasse Diagram [Ros91]. To achieve this purpose, we first defined a vector, denoted by $\mathbf{B}$ and of size $|\mathbf{P}|$, recording for each reasoner the bucket it belongs to. For instance, whenever the success of a reasoner R_i in classifying the ontology O_u is predicted, then $\mathbf{B}[i] \leftarrow$ 'S'. At the beginning of the ranking process, the buckets of the reasoners are unknown. This is why, the $\mathbf{B}$ vector is initialized by a sequence of 'F' (Algorithm 3, Line 3). More specifically, 'F' stands for the *full* bucket, i.e. $\mathcal{B}_F$, which gathers all the alternatives. Hence, we can ensure that all the reasoners are initially tied within $\mathcal{B}_F$ and share the same rank. By consequence, all the pairs of the $\mathbf{C_B}$ matrix are initially set to 0.5. We should highlight that $\mathcal{B}_F$ is placed at the bottom of the ordered buckets, i.e. $\mathcal{B}_S \prec \mathcal{B}_U \prec \mathcal{B}_T \prec \mathcal{B}_H \prec \mathcal{B}_F$. Reasoners are ranked according to their success state as well as to their computational time. Hence, we defined a further vector denoted by $\mathbf{T}$. The latter one is of size $|\mathbf{P}|$ and records for each reasoner its predicted runtime. Initially, all the $\mathbf{T}$ values are equal to $+\infty$, which stands for the maximal possible computing time (Algorithm 3, Line 5). Afterwards, the $\mathbf{C_B}$ matrix, $\mathbf{B}$ and $\mathbf{T}$ vectors are progressively updated while iterating over the reasoner set and according to the computed predictions. It is important to note that only the reasoners belonging to $\mathcal{B}_S$ will have their corresponding $\mathbf{T}$ value lower than $+\infty$. As previously argued, the tie will be broken particularly for these reasoners since their robustness is predicted. Such configuration would guarantee that reasoners within the other buckets, will always be tied at the bottom of the reasoner ranking list according to the time criterion. A step by step

Algorithm 3: The Reasoner Ranking with ties

```
Input  : The user input ontology $O_u$.
Output: $Pr$ formatted representation of the predicted ranking, termination state and runtime
        of the reasoners.
1  $X^{(O_u)} \leftarrow computeTheFeatureVector(O_u)$;
2  $\mathbf{P} \leftarrow loadTheReasonerProfiles()$ ;
3  $\mathbf{B} \leftarrow [F, F, \ldots, F]$; // B is of size $m$
4  Initialize the pairs of the $\mathbf{C_B}$ matrix with 0.5 ;
5  $\mathbf{T} \leftarrow [+\infty, +\infty, \ldots, +\infty]$ ; // T is of size $m$
6  for $i \leftarrow 1$ to $|\mathbf{P}|$ do
7      $\mathbf{B}[i] \leftarrow predictTerminationState(\mathbf{P}[i], X^{(O_u)})$;
8      if $\mathbf{B}[i] =$ 'S' then // the reasoner is robust
9          $\mathbf{T}[i] \leftarrow predictTheRuntime(\mathbf{P}[i], X^{(O_u)})$;
10     end
11     for $j \leftarrow 1$ to $|\mathbf{P}|$ do
12         if $(\mathbf{B}[i] \prec \mathbf{B}[j])$ or $(\mathbf{B}[i] = \mathbf{B}[j] =$'S' and $\mathbf{T}[i] < \mathbf{T}[j])$ then
13             $\mathbf{C_B}[i][j] \leftarrow 1$ ;
14         else if $(\mathbf{B}[i] = \mathbf{B}[j] \neq$'S'$)$ then
15             $\mathbf{C_B}[i][j] \leftarrow 0.5$ ;
16         end
17         $\mathbf{C_B}[j][i] \leftarrow 1 - \mathbf{C_B}[i][j]$ ;
18     end
19 end
20 $Pr \leftarrow formatThePredictedOutputs(\mathbf{C_B}, \mathbf{B}, \mathbf{T})$;
21 return $Pr$;
```

running example of Algorithm 3 is given in Figure 5.2. Given a user ontology O_u and five alternative reasoners with their already established profiles $\mathbf{P} = [P(R_1), P(R_2), P(R_3), P(R_4), P(R_5)]$, Figure 5.2 depicts the different iterations processed in order to encode the $\mathbf{C_B}$ matrix as well as the $\mathbf{B}$ and $\mathbf{T}$ vectors. In each iteration, we show the reasoner R_i predicted label and eventually its predicted runtime.



Init State:

Candidate reasoners: R_1, R_2, R_3, R_4, R_5

B

F	F	F	F	F

T

+∞	+∞	+∞	+∞	+∞

C_B

0.5	0.5	0.5	0.5	0.5
0.5	0.5	0.5	0.5	0.5
0.5	0.5	0.5	0.5	0.5
0.5	0.5	0.5	0.5	0.5
0.5	0.5	0.5	0.5	0.5

1st Iteration:

- Reasoner: R_1
- Predicted Label: U

B

U	F	F	F	F

T

+∞	+∞	+∞	+∞	+∞

C_B

0.5	1	1	1	1
0	0.5	0.5	0.5	0.5
0	0.5	0.5	0.5	0.5
0	0.5	0.5	0.5	0.5
0	0.5	0.5	0.5	0.5

2nd Iteration:

- Reasoner: R_2
- Predicted Label: S ,
- Predicted Time: 100 s

B

U	S	F	F	F

T

+∞	100	+∞	+∞	+∞

C_B

0.5	0	1	1	1
1	0.5	1	1	1
0	0	0.5	0.5	0.5
0	0	0.5	0.5	0.5
0	0	0.5	0.5	0.5

3rd Iteration:

- Reasoner: R_3
- Predicted Label: T

B

U	S	T	F	F

T

+∞	100	+∞	+∞	+∞

C_B

0.5	0	1	1	1
1	0.5	1	1	1
0	0	0.5	1	1
0	0	0	0.5	0.5
0	0	0	0.5	0.5

4th Iteration:

- Reasoner: R_4
- Predicted Label: U

B

U	S	T	U	F

T

+∞	100	+∞	+∞	+∞

C_B

0.5	0	1	0.5	1
1	0.5	1	1	1
0	0	0.5	0	1
0.5	0	1	0.5	1
0	0	0	0	0.5

5th Iteration:

- Reasoner: R_5
- Predicted Label: S
- Predicted Time: 10s

B

U	S	T	U	S

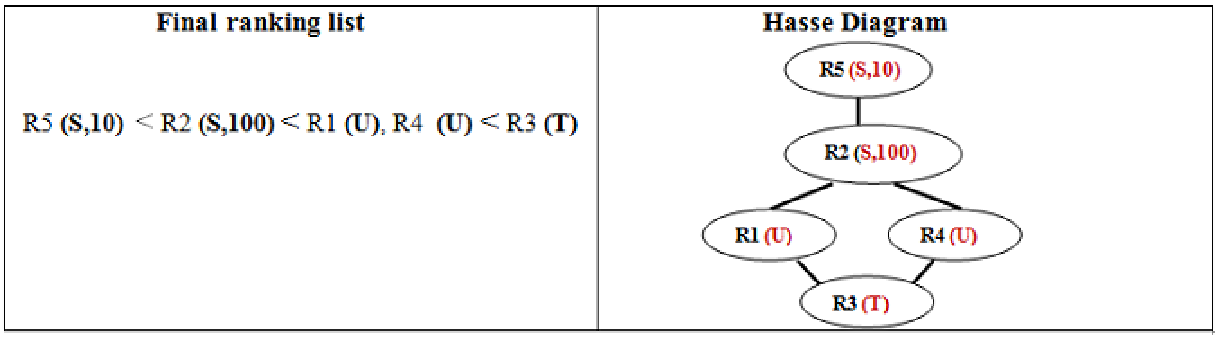
T

+∞	100	+∞	+∞	10

C_B

0.5	0	1	0.5	0
1	0.5	1	1	0
0	0	0.5	0	0
0.5	0	1	0.5	0
1	1	1	1	0.5

Figure 5.2: Running example of reasoner ranking algorithm.



Final ranking list

R5 (S,10) < R2 (S,100) < R1 (U), R4 (U) < R3 (T)

Figure 5.3: Formatted ranking according to the string and to Hass diagram respective representation.

Then, we outline the subsequent changes over the data structures, i.e. **B**, **T** and $\mathbf{C_B}$. The changes are highlighted in bold face or in red color. The computed reasoner pair order matrix could be illustrated using the well known Hasse Diagram or by a simple string representation. The final computed ranking is depicted in Figure 5.3. Given the description of the above algorithm and its resulting ranking, we would assert that our solution have satisfied all the functional requirements listed in Subsection 5.3.1 and have complied with the preference order established in Subsection 5.3.2. In the remainder of this section, we will assess the quality of the produced rankings given a large set of test ontologies.

5.4.2 Experimental setup

In this section, we describe the results of the empirical evaluation of our first reasoner ranking approach. We will begin by describing the test data. Then, we will list the assessment measures that we will employ to assess the quality of the computed rankings. Afterwards, we will report and discuss the results of the conducted experiments.

5.4.2.1 Experimental data

Our starting meta-knowledge base is mainly made up from the reasoner profiles that we have described in Chapter 4. Hence, {Konclude, MoRE, HermiT, TrOWL, FaCT++, JFact} is our set of alternative reasoners, which will try to automatically rank given any input ontology. Our ranking evaluations are based on a holdout method, where the data is separated into two sets, called the training set and the testing set. When training the reasoner predictive models, we have already used 2500 distinct ontologies (cf. Chapter 3, Subsection 3.3.1). Ontologies within this initial corpus, $\mathcal{C}(\mathcal{O})$, together with their computed feature vectors are counted as the training examples. Therefore, we selected further 500 ontologies from the original ORE corpora, to create the ranking test set, i.e. the $\mathcal{CT}(\mathcal{O})$ corpus. Ontologies within the latter one are different from those in $\mathcal{C}(\mathcal{O})$, but have similar overall distribution proportions over the size and expressivity bins. Thus, $\mathcal{CT}(\mathcal{O})$ stands for the set of the *unforeseen* ontologies to evaluate the supervised ranking method. Later by following the specification we made in Chapter 4, Subsection 4.4, we carried out new ontology classification evaluations using the six alternative reasoners and over the $\mathcal{CT}(\mathcal{O})$ ontologies. Table 5.1 summarizes the results. The reasoners are listed in a decreasing order with respect to their success rate; otherwise, the most robust reasoner over the ontology test corpus is the one at the top of the list (cf. Chapter 4, Definition 23).

These evaluations were put forward in order to record the real performances of the examined reasoners over the test ontologies. Indeed, given a test ontology, the predicted ranking of reasoners will be compared to the *ideal* one. An *ideal* ranking, also called the *target* ranking, should represent as accurately as possible the correct ordering of the reasoners given the ontology at hand. In our case, it is built by applying rules from Definition 26 on the basis of the real reasoner performances reported during the above described evaluations. Once the ideal and the predicted ranking is computed for each test ontology, the agreement between both of them is measured using known state-of-the-art metrics. Intuitively, high agreement with the ideal ranking would witness the effectiveness of our approach. In the remainder of this chapter, by *recommended ranking*, we will refer to the one computed based on the reasoner predicted performances. Besides computing the agreement with the ideal ranking, we need to compare the quality of our recommendations versus a *baseline* method. The latter one is a

Reasoner	**#S**	**#U**	**#T**	**#H**	**Runtime**		
					Min	**Max**	**Mean**
Konclude	482	8	9	1	0.04	182.75	3.37
MORe	435	19	44	2	1.09	150.68	8.03
HermiT	432	1	62	5	0.59	150.12	16.83
TrOWL	429	53	15	3	0.47	173.26	10.54
FaCT++	393	7	81	19	0.47	157.50	7.50
JFact	339	6	128	26	0.63	141.44	14.3

Table 5.1: Classification results over the test ontologies. The time is given in seconds. **#S**, **#U**, **#T** and **#H** stand respectively for the number of ontologies labelled by *Success*, *Unexpected*, *Timeout* or *Halt*.

trivial solution commonly called the *default ranking*. According to this method, the ranking remains the same regardless of the ontology under examination. The default ranking is decided on the basis of the reasoner evaluation results described in Table 5.1. This means the overall success rate of reasoners achieved over the ontologies of $CT(O)$. Thus, our default ranking starting from the best reasoner is: Konclude $\prec$ MoRE $\prec$ HermiT $\prec$ TrOWL $\prec$ FaCT++ $\prec$ JFact. A ranking method is interesting, whenever it performs significantly better than the default method. Besides being a trivial solution, the default method could be seen as an instance of the brute force approach commonly applied in practice for reasoner selection. Indeed, it is often the case that an inexperienced user selects a reasoner for her/his ontology based application, by referring to rankings published following a recent reasoner benchmark or competition. Hence, this ranking computed based on average overall performances is considered as a default valid one for any ontology and whatever is the application and the usage constraints. Obviously, this is a misleading solution since reasoner performances are closely depending on the nature of the examined ontology and varies according to the provided computational resources. In our experiments, we will show that our introduced reasoner ranking method, which consider the distinctive characteristics of the examined ontology, is more relevant and capable to deliver more effective results than the default method.

5.4.2.2 Assessment methods

In short, for each test ontology $O_t \in \mathcal{CT}(\mathcal{O})$, we do the following: *1)* build and save an *ideal* ranking of reasoners, denoted by $i\sigma(O_t) = (i\sigma_1, i\sigma_2, \ldots, i\sigma_m)$; *2)* get the sorted list of the recommended reasoners by applying our ranking with ties method, denoted by $r\sigma(O_t) = (r\sigma_1, r\sigma_2, \ldots, r\sigma_m)$; *3)* get the default list of ranking, denoted by $d\sigma = (d\sigma_1, d\sigma_2, \ldots, d\sigma_m)$ and *4)* compute the agreement between $i\sigma$ and respectively $r\sigma$ and $d\sigma$ using an appropriate assessment measure. The agreement between our recommended rankings (resp. the default ones) and the ideal rankings is qualified according to five quality measures falling into *two* categories describes in what follows.

Correlation metrics. To assess the agreement on the full computed rankings, we choose to compute two well known correlation metrics: the Kendall Tau [EM02] and the Spearman Rho [CC97]. We pay particular attention to choose only the metrics that can handle ties in ranking. First, we will be measuring the generalized *Kendall Tau* correlation coefficient for ranking with ties, denoted by τ_x and proposed by Emond and Mason [EM02]. The latter authors have noted a problem with the classical Kendall distance in case of partial ranking, i.e. the one having ties. Hence, they defined new correlation coefficient as:

$$KendallTauX = 1 - \frac{2d(R_s, R_t)}{m(m-1)} \tag{5.2}$$

where R_s, R_t are any two rankings and $d(R_s, R_t)$ is the Kemeny distance. Known that the number of alternative reasoners is m, the Kemeny distance is computed as follows:

$$d(R_s, R_t) = \frac{1}{2} \sum_{i=1}^{m} \sum_{j=1}^{m} |x_{(s)ij} - x_{(t)ij}| \tag{5.3}$$

where $x_{(s)ij}$ (resp. $x_{(t)ij}$) is defined as equal to 1 if the reasoner i is preferred to reasoner j in the ranking s, otherwise -1 and equal to 0 if the two reasoners are tied.

Furthermore, the accuracy of our recommended ranking is assessed using the *Spearman rank correlation coefficient* known as the *Spearman's rho* [CC97]. In the latter metric, tied reasoners are assigned a rank equal to the average of their positions in the ascending order of the values. For instance, a recommended ranking $r\sigma = (2, 1, 3, 3, 4)$ is transformed into $r\sigma' = (2, 1, 3.5, 3.5, 5)$. Formally, the *Spearman's rho* is defined as follows, where $r\sigma_k$ and $i\sigma_k$ are the recommended and ideal ranks of reasoner k respectively:

$$SpearmanRHO = 1 - \frac{6\sum_{k=1}^{m}(r\sigma_k - i\sigma_k)^2}{m^3 - m} \tag{5.4}$$

According to the above described correlation measures, a value of 1 represents perfect agreement, whereas -1 stands for a perfect disagreement. A correlation of 0 means that the rankings are not related.

Information retrieval (IR) based metrics [BYRN99]. By following the evaluation style of search engines, we will examine how well we are ranking the reasoners at the top of the list. Although these metrics do not take into consideration ties within the ranking, we choose to apply them since

Method	Kendall TauX	Spearman Rho	MAP@1	MAP@3	MAP@6
Recommended	0.68 (± 0.25)	0.76 (± 0.25)	0.92 (± 0.27)	0.70 (± 0.25)	0.62 (± 0.23)
Default	0.32 (± 0.23)	0.41 (± 0.27)	0.91 (± 0.29)	0.59 (± 0.19)	0.41 (± 0.14)
Gain	111.71%	87.43%	1.32%	19.73%	44.94%

Table 5.2: Evaluation summary of reasoner ranking methods over ontologies of $CT(O)$

they are easy to interpret metrics and widely recognized by the IR and the ML respective communities. In overall, for each ranking list provided for a test ontology O_t, we measure the *Precision* at position K, denoted as P@K(O_t), which stands for the percentage of correctly predicted ranking of reasoners at that position K. Then, we compute the Average of Precision at position K, denoted as AP@K (i.e. $AP@K = \frac{\sum_{i=1}^{K} P@i(O_t)}{K}$)[4]. To obtain an overall idea of precision considering the whole test set, we compute the Mean Average Precision, denoted as MAP@K (i.e. $MAP@K = \frac{\sum_{j=1}^{l} AP@K(O_j)}{M}$, where $l = |CT(O)|$). More specifically, we put our focus on three possible values of the K index. We recall that we are examining six reasoners, thus, we record for each test ontology: *(i)* the AP@1 as it reflects how well we are predicting the most performing reasoner; *(ii)* the AP@3 since the top 3 of a ranked list are the most likely to be examined by a user; and *(iii)* the AP@6 which stands for the full rank of the alternatives. Finally to sum up the evaluations, we compute MAP@1, MAP@3 and MAP@6.

5.4.3 Experimental results

Table 5.2 illustrates the mean values of correlation coefficient computed over the different rankings across the test ontologies. The table also reports the mean average precision at three different positions. Each assessment value is put together with its corresponding standard deviation. This gives a general idea about the ranking accuracy across the different test set ontologies and makes it possible to compare the overall quality of our recommended ranking to the quality of the default method.

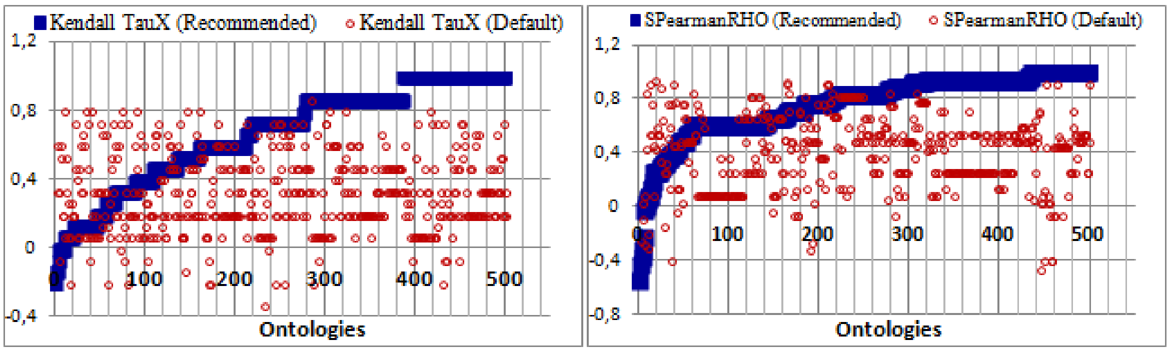
As it can be noticed, the rankings generated by our method, i.e. the recommended, are more correlated to the ideal rankings according to both Spearman's Rho and Kendall's Tau X. The difference with the values of the default ranking is significantly high, witnessing the good quality of our provided rankings. This would further be asserted by looking at the value of MAP@1. The latter one indicates that in 92% of the test cases, our method have succeeded to predict and to correctly rank the topmost performing reasoner. Our value is even better than the MAP@1 achieved by the ranking algorithms deployed in the meta-reasoner R_2O_2 [KKL15]. The authors reported a MAP@1 varying between 88.7% and 90.6%. On the other hand, the values of the mean average precision at position 3 and 6, respectively

[4]It is to be noted that when $K = 1$, P@1 is equal to AP@1.

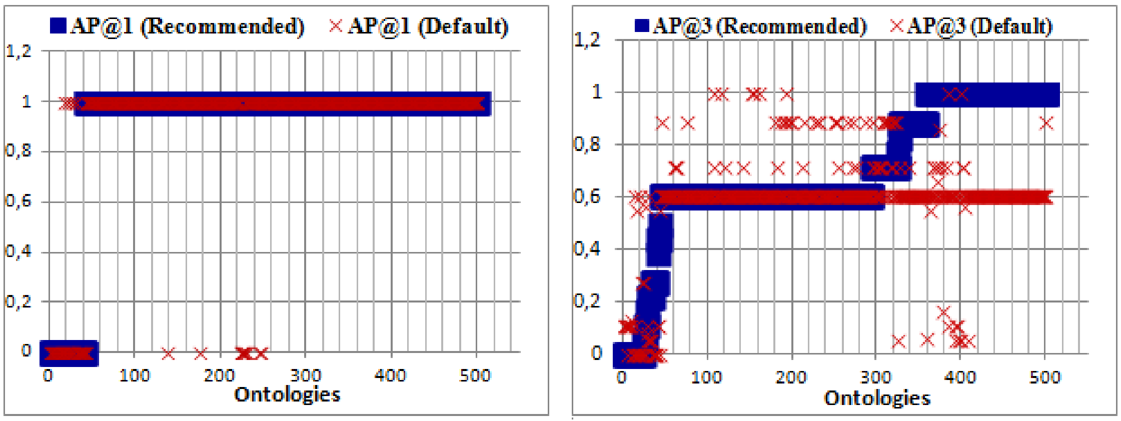
MAP@3 and MAP@6, are less good than MAP@1, revealing in some way troubles in ranking reasoners at positions higher than the first place. However, our MAP@3 and MAP@6 remain sufficiently higher than those computed for the default rankings. An overview about how much improvements our ranking with ties method can achieve over the default method is provided in the last line of Table 5.2, which reports the gain percentage according to each assessment measure. The improvement is remarkable according to Kendall Tau X, Spearman' RHO and MAP@6. However, it seems unimportant in terms of MAP@1. It is important to note that in the default ranking, the best reasoner is Konclude. The latter one is the dominant reasoner in our evaluations over the test set ontologies. Indeed, it is the best one in the ranking of 90.6% of the cases, which would explain the high value of MAP@1 achieved by the default method. Therefore, an improvement of 1.32% is not as bad as it seems to be, since the maximal possible improvement would be equal to 9.4%.

Furthermore, to ensure that the observed difference in the ranking quality is not likely to be due to chance, we applied the well known paired Student's t-test [SAC07]. The comparison between the recommended and the default method was carried out according to each aforementioned assessment measure. Hence, five distinct t-tests were computed respectively examining the difference in Kendall Tau X, Spearman's RHO, AP@1, AP@3 and AP@6. In each test case, the null hypothesis assumes that there is no difference in the mean average value of the assessment measure, while the alternative hypothesis asserts that at average the recommending method is better performing. In all cases, the achieved p-values were $< 2.2e^{-16}$, which means that the null hypothesis is rejected. These results show that, at the 95% confidence level, the performance of the proposed ranking with ties method is significantly better than the default method according to the different assessment measures. All of these findings approve our starting assumption that a ranking of reasoners based on predicted robustness is a better alternative to any manual or default selection. It is also important to highlight that in the previous work about reasoner ranking conducted by Kang et al. [KKL15], the produced rankings were not compared to any other ones even the default rankings and P@1 was the only reported assessment measure. Therefore, we would assert that our evaluations are more reliable and informative ones.

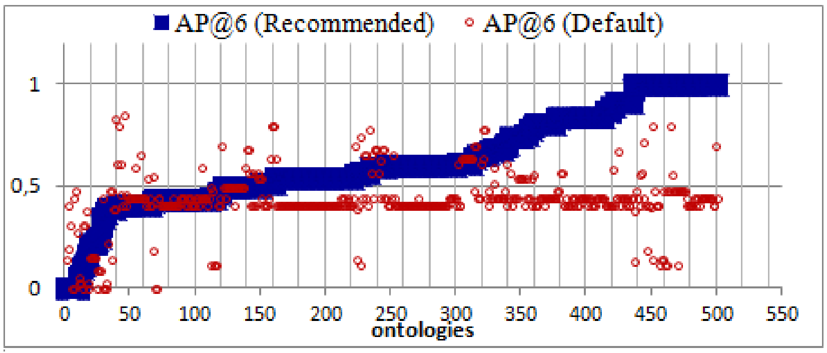
Despite the convincing good ranking accuracy achieved by our method, we were curious about its behaviour at the individual level of the testing ontologies. To get more insights, we plotted per ontology the assessment values achieved by each ranking method. Figure 5.4 scrutinizes the comparison established according to the aforementioned assessment measures and having as counterparts our recommended rankings (highlighted in blue) and the default ones (outlined in red). From the different plots illustrated in Figure 5.4, it can easily be observed that in major cases our curve is higher than the one of the default method showing better accuracy according to all the assessment measures. However for some ontologies, we did not succeed to outperform the default ranking in terms of the



(a)



(b) (c)



(d)

Figure 5.4: Per ontology comparison between the recommended rankings (in blue) and the default ones (in red)

correlation with the ideal ranking. The advantage of this kind of analysis is that it enables us to mark off the hard ontologies, which have yield to inaccuracies. Hence, our investigations would be better focussed towards more targeted improvements. Indeed by examining these ontologies, it might be possible to characterize further relevant ontology features and thus, enhance our initial feature suite. We should also assert that our ranking approach is closely depending on the quality of the predictive models included in the reasoners profiles. More specifically, the accumulated prediction errors across the reasoner models can have an important impact on the overall ranking accuracy. For instance, we observed that the misdeed of some rankings was due to the margin errors in the predicting of reasoner runtimes. Indeed, the computational time of some of the examined reasoners, in particular HermiT and MoRE, are in general very close to each other. Hence, a minor gap between the predicted and the real runtime of one of these reasoners might alter the full ranking. Furthermore, we admit that although the suggested ranking with ties method could efficiently be applied online to compute the rankings, it requires a lot of training time in order to produce the reasoner profiles.

Seeking better performances in reasoner ranking with less training efforts have motivate us to inspect different ranking strategies like the one that we will describe in the next section.

5.5 Reasoner Ranking based on multi-label prediction approach

Similarly to the conventional supervised learning, in multi-target learning [FH10, TKV10, ZZ14], each input instance is characterized by a d-dimensional feature vector $\mathbf{X}^{(i)} = (x_1^i, x_2^i, \ldots, x_d^i)$, but it is associated with a set of output labels rather than one single label. These outputs could be real-valued, binary, ordinal, categorical or even of mixed types. Let $\mathcal{X}$ be the domain of the input features, also called the predictor variables space and let $\mathcal{Y}$ be the domain of the output labels, also called the target variables space. The task of multi-target learning is to train a model, in fact a function $h : \mathcal{X} \to \mathcal{Y}$, which is capable to predict the proper output label vector $\widehat{Y} = (\widehat{y_1}, \widehat{y_2}, \ldots, \widehat{y_m})$, given the feature vector $X = (x_1, x_2, \ldots, x_d)$ of an unseen input instance. The model is learned from a dataset $\mathcal{D} = \{(X^{(1)}, Y^{(1)}), \ldots, (X^{(n)}, Y^{(n)})\}$, which assembles n training examples.

$$X = (x_1, x_2, \ldots, x_d) \overset{h(X)}{\longrightarrow} \widehat{Y} = (\widehat{y_1}, \widehat{y_2}, \ldots, \widehat{y_m}),\ X \in \mathcal{X} \textit{ and } \widehat{Y} \in \mathcal{Y} \tag{5.5}$$

It has been proven in the related literature [TKV10, ZZ14] that the multi-target models have a number of benefits over single-target models specially when the different targets are related. Indeed, they could lead to more accurate models due to their ability to exploit potential dependencies among the target variables. Besides, they are faster to learn and could lead to faster prediction times. In ranking prediction, the target variables stand for the alternatives to be sorted. Given this kind of dependency between the alternatives, the multi-target learning is said to be more suitable for ranking prediction. Motivated enough, we chose to examine how to employ this particular kind of models to meet our demand in ranking the ontology reasoners. Our solution should mostly comply with the requirements listed in Subsection 5.3.1 and respect the preference order specified in Subsection 5.3.2.

In the remainder of this section, we will first discuss the existing works, then introduce our new ranking method. Later, we will establish an empirical study to assess the quality of our solution and compare it to our previously established reasoner ranking method.

5.5.1 Multi-label learning techniques

By varying the nature of the target variables, i.e. ordinal, binary or real-value, different forms of the multi-target learning task have emerged, such as the multi-label classification (MLC), the label ranking (LR), the multi-label ranking (MLR) and the multi-target regression (MTR). Rigorous surveys scrutinizing state-of-art works falling under each of these categories are established in [FH10, TKV10, ZZ14]. We briefly recall some of their basic notions:

Multi-label classification (MLC) [ISTV10]: it is concerned with learning a model that outputs a bipartition of the output labels into relevant label set P_x and irrelevant label set N_x, where $P_x \cap N_x = \varnothing$ and $P_x \cup N_x = \mathcal{Y}$. Formally, the output space has the form $\mathcal{Y} = \{0,1\}^m$ and thus the learning function is reduced to $h : \mathcal{X} \rightarrow \{0,1\}^m$, with 0 means irrelevant and 1 otherwise. For instance, $Y = (1,1,0,0) \in \mathcal{Y}$ is the output vector, $Y = h(X)$, associated with some $X \in \mathcal{X}$. In this case, 1 indicates that the label y_i is relevant for the input instance and by consequence $y_i \in P_x$. Similarly, 0 indicates that y_j is irrelevant and thus $y_j \in N_x$.

Label Ranking (LR) [FH10]: it is concerned with learning a model that outputs an ordering of the labels according to their relevance to the input instance. Hence, the h function maps every instance $X \in \mathcal{X}$ to a total strict order, $\prec$, over the set of the output labels. A ranking over $\mathcal{Y}$ can conveniently be represented by a permutation σ of the set of indices $\{1,\ldots,m\}$, where $\sigma(i)$ stands for the position of label y_i in the ranking associated with X, i.e. its rank value. For instance given an input instance X and 5 target outputs, the LR model often called a *ranker*, produces an output vector of the form $(2,3,5,1,4)$, which stands for $(\sigma(1),\sigma(2),\sigma(3),\sigma(4))$ the ranking of the output labels.

Multi-label Ranking (MLR) [FHLMB08, ZZ14]: is a generalisation of MLC and LR. Thus, the problem of multi-label ranking involves two equally important subsequent goals. The first is to produce the bipartition of $\mathcal{Y}$ into relevant label set P_x (positives) and irrelevant label set N_x

(negatives). The second is a ranking $\prec$ over $\mathcal{Y}$. Specially, $\prec$ should be consistent, meaning that there should be no irrelevant labels ranked higher than relevant ones and vice versa. Formally, whenever $\forall y_i \in P_x$ and $\forall y_j \in N_x$, then $y_i \prec y_j$. In other words, y_i is preferred to y_j and it is ranked lower $\sigma(i) < \sigma(j)$.

Multi-target Regression (MTR) [STGV16]: it is also known as the multivariate or multi-output regression task. Techniques falling in this category aim at predicting multiple real-valued target variables instead of binary ones (the MLC case) or permutations (the LR case). Literally given an instance $X \in \mathcal{X}$, the MTR model, i.e. $h()$, maps X to the output vector $Y \in \mathcal{Y}$, such as $\mathcal{Y} \equiv \mathbb{R}^m$. Hence, all the multi-label learning forms could be seen as special cases of the multi-target regression problem. Training MTR model is a less popular task, but still arises in several interesting domains.

Multi-label based methods falling in each of the above described categories could be further categorized into two different groups: (i) problem transformation methods, which transform the multi-label learning task into one or more single-label classification or regression tasks; and (ii) algorithm adaptation methods, which extend specific learning algorithms in order to handle multi-label data directly. Further details about these groups are provided in [TKV10, ZZ14].

5.5.2 Multi-label based solutions for the algorithm selection problem

We investigated the works which have employed some multi-label learning technique to solve the algorithm selection problem. The main purpose of our analysis is to examine whether the requirements we have listed in Subsection 5.2 and the preference rules we have established in Subsection 5.3.2 could be satisfied using the existing solutions. Hence, we will focus on three main points: *(i)* whether a ranking of the alternatives is computed; *(ii)* if partial orders are considered and ties are allowed; *(iii)* how informative is the method. The investigated works fall into two major categories, both of them are discussed in the following:

MLC based solutions. Olmo et al. [ORGV15] have recently proposed a meta-learning approach based on multi-label classification. They applied their approach to a repository of educational datasets in order to recommend the most suitable ML classification algorithms to carry a learning. In this context, the instances are feature vectors describing datasets and the target variables match the examined ML algorithms. A model is trained which predicts whether an algorithm is relevant or not given an input dataset. A relevant algorithm means it is capable to train the dataset and to deliver a model having an accuracy higher than some fixed threshold. The proposed system lists the relevant algorithms as the recommended ones without any ranking or further details about their expected performances. Hence, we would assert that such approach does not meet our requirements.

LR based solutions. The label ranking models have been deployed in meta-leaning systems [SP13], in SAT solver portfolios [OHL15] and in the meta ontology reasoner, R_2O_2 [KKL15]. In these systems, the employed models produce strict total orders with high confidence levels. No further indications about the expected performances of the examined alternatives are provided. Furthermore, there is a common assumption among these works which entails that all of the alternatives are relevant ones and a ranking is an indication about what to choose at first. This is a quite different specification from what we need to fulfil. Indeed, we are interested in ranking relevant reasoners, i.e. the robust ones, as well as the irrelevant ones, those which would potentially fail to handle the ontology. This is because our main concern is to be as informative as possible (see Subsection 5.3.1). Unsuccessful reasoners could further be categorized with respect to the failure nature and then partially ranked (see Subsection 5.3.2). Learning *rankers* of partial orders is another issue in label ranking. Very few works have addressed this problem. For instance, Cheng et al. [CHWW12] have proposed to allow their LR model to *abstain* from setting a preference between two compared alternatives, if the model is not sure about what to choose. Hence, a couple of alternatives could be incomparable and by consequence a ranking is said to be a partial order. In this case, abstention is just an option, put forward to reduce the ranking errors. Furthermore, to the best of our knowledge, learning to predict bucket orders according to the rules we have established has not been investigated.

Given these findings, we would assert that the LR techniques are interesting ones: they simplify the ranking process, i.e. just one predictive model to invoke; and they are known to be highly accurate. Nevertheless, they cannot satisfy all of our requirements. Actually when a ranked list of reasoners is provided to a user, the latter one will not be able to distinguish the reasoners to avoid. Let $(2, 1, 3, 4, 4, 6)$ be the ranking of 6 reasoners according to our preference rules. Here, we can deduce that reasoners ranked as 4 and the one in the 6^{th} position are unsuccessful ones. This is because ties are only allowed among reasoners within one of the failure buckets $\{U, T, H\}$. Reasoners ranked higher than a tied alternatives are also unsuccessful ones, since the buckets are linearly sorted in an increasing order. However, we cannot decide whether the reasoner at the 3^{rd} position is a successful one or not. Indeed, any bucket can have a single element and at most 3 failure buckets might appear in a ranked list. In our example, we have already identified 2 of these buckets, hence the third bucket may exist. If this is the case, the bucket would be ranked lower than the other ones, otherwise in the 3^rd position. When, we designed our first reasoner ranking method, we paid special attention to avoid such ambiguity. Actually, our single label based method provides two equally important outputs: the rank of each reasoner and the bucket to which it belongs. Thus, the ranking could be better understood. In overall, the irrelevant reasoners are tied within at most 3 buckets and the remaining are relevant ones.

Based on these inspections, we think that the multi-label ranking (MLR) approach is the closest to meet our requirements. As a matter of fact, methods applying this approach provide a ranking as well as a bipartite partitioning of the output labels into relevant P_x and irrelevant N_x subsets. Motivated enough, we decided to examine more in details some of the methods falling in this category. We should highlight that to the best of our knowledge, there is no previous work which have attempted to solve the algorithm selection or ranking problem using the MLR techniques.

MLR based solutions. The calibrated label ranking (CLR) introduced by J. Fürnkranz et al. [FHLMB08] is the most wide spread solution of the multi-label ranking problem. The basic idea of this method is to extend the label space $\mathcal{Y}$ by introducing one additional virtual label, y_0, known as the *calibration label*. It represents the splitting point between the relevant and the irrelevant labels. Hence, every label y_i known to be relevant for an instance X, i.e. $y_i \in P_x$, should be ranked higher than the virtual label, y_0. Similarly, any irrelevant label $y_j \in N_x$ must be ranked lower than y_0.

$$\underbrace{y_1 \prec y_2 \prec \ldots \prec y_{i-1}}_{P_x \ (the\ relevant)} \prec y_0 \prec \underbrace{y_i \prec y_{i+1} \prec \ldots \prec y_m}_{N_x \ (the\ irrelevant)} \tag{5.6}$$

Following this configuration, a calibrated label ranking dataset can then be trained by any conventional label ranking (LR) algorithm over the extended label space, i.e $\mathcal{Y}' = \mathcal{Y} \cup \{y_0\}$. Authors have chosen to work with ranking by pairwise comparison, the RPC algorithm [ZZ14]. Roughly speaking, RPC transforms a dataset with m target labels into $\frac{m(m-1)}{2}$ binary datasets, one for each label pair (y_j, y_k), such that $(1 \leqslant j < k \leqslant m)$. Afterwards, a binary model is trained for each transformed dataset in order to decide where y_i is preferred to y_j. Given a new instance, all the binary models are invoked and a ranking is computed based on the obtained votes for each label. In the case of calibrated ranking, m binary datasets are added corresponding to the pairs (y_i, y_0), $\forall i \in \{1, .., m\}$. In overall, the complexity of the CLR method is quadratic with respect to the number of the required binary models, which is a relatively high complexity. On the other hand, regarding our ranking requirements, the major drawback of the above described method is the fact that no ties are allowed within the computed rankings. Indeed, a strict total order is produced over each of the two partitions P_x and N_x. Hence using the CLR method, we cannot delineate all of our buckets and we can no more allow two or more reasoners to be equally ranked. However, we can at least distinguish between the relevant and the irrelevant reasoners in a ranked list.

Inspired from the various multi-label learning approaches, we designed a novel multi-label ranking method which tries to avoid the aforementioned drawbacks and to satisfy most of our requirements.

Details are outlined in the forthcoming subsections.

5.5.3 Design description of the novel multi-label ranking method

At this stage of our research work, we are willing to employ multi-label learning techniques to predict the ranking of reasoners for any given input ontology. Known that the reasoners are ranked according to the preference rules established in Subsection 5.3.2.2, we argued that providing only the reasoner ranks might be misleading for the users. This is because the ranked list might include or be made up from reasoners expected to fail in handling the input ontology. Hence, it is important not only to predict the ranks but also to identify the reasoners respective buckets, i.e. $\mathcal{B}_S, \mathcal{B}_U, \mathcal{B}_T, \mathcal{B}_H$. Nevertheless, we choose not to outline all of the possible buckets, but only to distinguish between the successful/unsuccessful ones. Giving up some of our requirements was decided in order to achieve an effective solution of reasoner ranking using multi-label ranking (MLR) techniques.

5.5.3.1 Modelling the Extended Multi-label Ranking (EMLR) method

In our learning context, by *relevant reasoner* we mean a *successful* one with respect to the classification task of an input ontology. More specifically, we define the bipartite partition over a label set of ranked reasoners as follows:

Definition 27 (Bipartite partition over reasoner label set) *Let $\mathcal{R} = \{R_1, \dots, R_m\}$ be the set of labels matching the m alternative reasoners and let $\mathbf{B} = B_1 \prec \dots \prec B_k \prec \mathcal{B}_U \prec \mathcal{B}_T \prec \mathcal{B}_H$ be the ordered bucket partition over $\mathcal{R}$, where $|B_i| = 1$ for $\forall i \in [1, k]$, while $|\mathcal{B}_U| + |\mathcal{B}_T| + |\mathcal{B}_H| = (m - k)$. The bipartite partition of $\mathcal{R}$ is defined such that $P_x = (B_1 \cup \dots \cup B_k)$ is the subset of relevant reasoners and $N_x = (\mathcal{B}_U \cup \mathcal{B}_T \cup \mathcal{B}_H)$ is the subset of the irrelevant reasoners, known that $P_x \cup N_x = \mathcal{R}$ and $P_x \cap N_x = \emptyset$. The bipartite partition of $\mathcal{R}$ is said to be consistent with the ranking associated to the bucket order $\mathbf{B}$, whenever $\forall R_i \in P_x$ and $\forall R_j \in N_x$, then the reasoner R_i is preferred to R_j and ranked lower, i.e. $\sigma(i) < \sigma(j)$.*

Based on these specifications, our reasoner ranking problem could be approached as a multi-label ranking (MLR) problem. Unlike the CLR method which combines the ranking and the relevance prediction tasks within the same model, we choose to tackle the multi-label ranking of reasoners by transforming it into other well-established multi-label learning scenarios. Indeed, the key idea of our solution is to learn a separate multi-label model for each of the above sub-problems: *(i)* a model to predict the ranking with ties of the alternative reasoners, denoted by $h_r()$;*(ii)* a model to predict the relevance of each reasoner $h_b()$. At prediction time, the results obtained given an input ontology must be synchronized to grant their consistency. More specifically, the predicted ranking will be checked and corrected whenever a contradiction regarding the predicted relevance is being caught. A further issue to be overcome is about predicting a ranking with ties using multi-label learning techniques. We previously highlighted the lack of solutions when the ranking involves ties. Hence, we decided to address this problem from a larger perspective and tackle it using the multi-target regression (MTR) techniques. The latter ones can handle different kinds of learning problems provided that the domain of the output variable is within $\mathbb{R}^m$. Our learning context satisfies this requirement since the rank position of each reasoner takes value in $\mathbb{R}$. Hence, the ranks of all the alternative could be considered as the targets in the multi-target regression setting. The advantage of this solution is very simple, no restriction about the relation among the target variables is been made. In other words, no matter whether the target values are ordered or have ties, the MTR model will try to predict the closest values to the real ones.

We call our approach the **Extended Multi-Label Ranking** (**EMLR**) solution applied to the problem of automatic ranking of ontology reasoners. We start outlining the main characteristics of our solution by specifying the novel hypothesis space underlying its framework. Let m be the number of alternative reasoners to be ranked. Each reasoner is referred to using an output label. Hence, the EMLR model could be defined as follows:

Definition 28 (EMLR model) *The **extended multi-label ranking** model, $EMLR$, is a function mapping an input feature vector X belonging to some d-dimensional vector space $\mathcal{F}^d$, to an output matrix $\mathbf{Y}$ of size $(m \times 2)$ having elements in $\mathbb{R}$ and built over the set of output labels $\{R_1, \dots, R_m\}$. $\mathbf{Y}$ is set up such that $\forall i \in [1, m]$, $\mathbf{Y}[i, 1] = \widehat{b_i}$ is the predicted relevance of the label R_i, i.e. $\widehat{b_i} \in \{0, 1\}$, and $\mathbf{Y}[i, 2] = \widehat{\sigma(i)}$ stands for the predicted rank of the same label.*

$$h_{emlr} : \mathcal{F}^d \to M_{m\times 2}(\mathbb{R}) \tag{5.7}$$

5.5.3.2 Learning and prediction steps

In the forthcoming paragraphs, we outline the required steps to train the EMLR model and those related to the prediction stage. Still, it appears appropriate to begin by describing the main feature of our training dataset and how it could be assembled.

Dataset assembling. On the basis of Definition 28, our training dataset $\mathcal{D}$ is made up from n training examples and has the following form: $\mathcal{D} = ((X^{(1)}, Y^{(1)}), \dots, (X^{(n)}, Y^{(n)}))$, where $X^i = (x_1, \dots, x_d) \in \mathcal{F}^d$ refers to an ontology feature vector and $Y^i = ((y_{11}, \dots, y_{m1})(y_{12}, \dots, y_{m2})) \in M_{m\times 2}(\mathbb{R})$ stands for the output matrix encoding respectively the relevance and the rank of each of the m alternative reasoners. The output matrix of each example is set up using our ranking with ties algorithm (see Subsection 5.4.1) applied on real data resulting from the reasoner empirical evaluations. Besides, the reasoner relevance is computing following Definition 27.

Training time. As explained above, the extended multi-label ranking model h_{emlr} is comprised of 2 models: a multi-label classification (**MLC**) model, $h_b : \mathcal{F}^d \rightarrow \{0,1\}^m$, which predicts the relevance of the output labels, and a multi-target regression (**MTR**) model, $h_r : \mathcal{F}^d \rightarrow \mathbb{R}^m$, which predicts the ranking with ties of these labels. Each of these models is learned independently from a reduced dataset, respectively $\mathcal{D}_b$ and $\mathcal{D}_r$. The latter ones share the same input vectors, however only the required output labels are maintained. On these basis, we can distinguish two main learning subtasks which could be performed in a parallel timed manner:

Learning the bipartite partitioning model ($h_b()$): it is achieved by employing a multi-label classification (MLC) algorithm to train the reduced dataset $\mathcal{D}_b$. The latter is made up from the n input vectors, $X^{(i)}$, from the original dataset $\mathcal{D}$ such that each $X^{(i)}$ is associated to the 1^{st} column from its corresponding output matrix $\mathbf{Y}^{(i)}$. No previous work have employed any MLC algorithm to solve the problem of automatic selection of ontology reasoners. This is why we carried out an empirical evaluation of the most valuable MLC algorithms, likely to deliver accurate bipartite decomposition of our output labels. The results of our investigation will be described in Subsection 5.5.5.1. The MLC algorithm to be incorporated in our method will be decided based on the assessment results.

Learning the ranking model ($h_r()$) : it is achieved by employing a multi-label classification (MLC) algorithm to train the reduced dataset $\mathcal{D}_r$. The latter is set up in the same way as $\mathcal{D}_b$, however each $X^{(i)}$ is associated to the 2^{nd} column from its corresponding output matrix $\mathbf{Y}^{(i)}$. Similarly, the MTR algorithm to be incorporated in our method will be picked up from a set of existing ones, after assessing their ranking quality when training our dataset.

Prediction time. During this online stage, to compute the ranking for a new introduced ontology, our system operates in three steps. All of these steps are performed in a sequentially timed manner, after computing the feature values of the introduced ontology:

Step 1. Relevance prediction: the MLC model is applied on the ontology feature vector to get the predicted relevance of each output label: $(b_1, \ldots, b_m) = h_b(X(O))$.

Step 2. Ranking prediction: similarly, the MTR model is applied on the ontology feature vector to get the predicted rank of each output label, $(\sigma_1, \ldots, \sigma_m) = h_r(X(O))$.

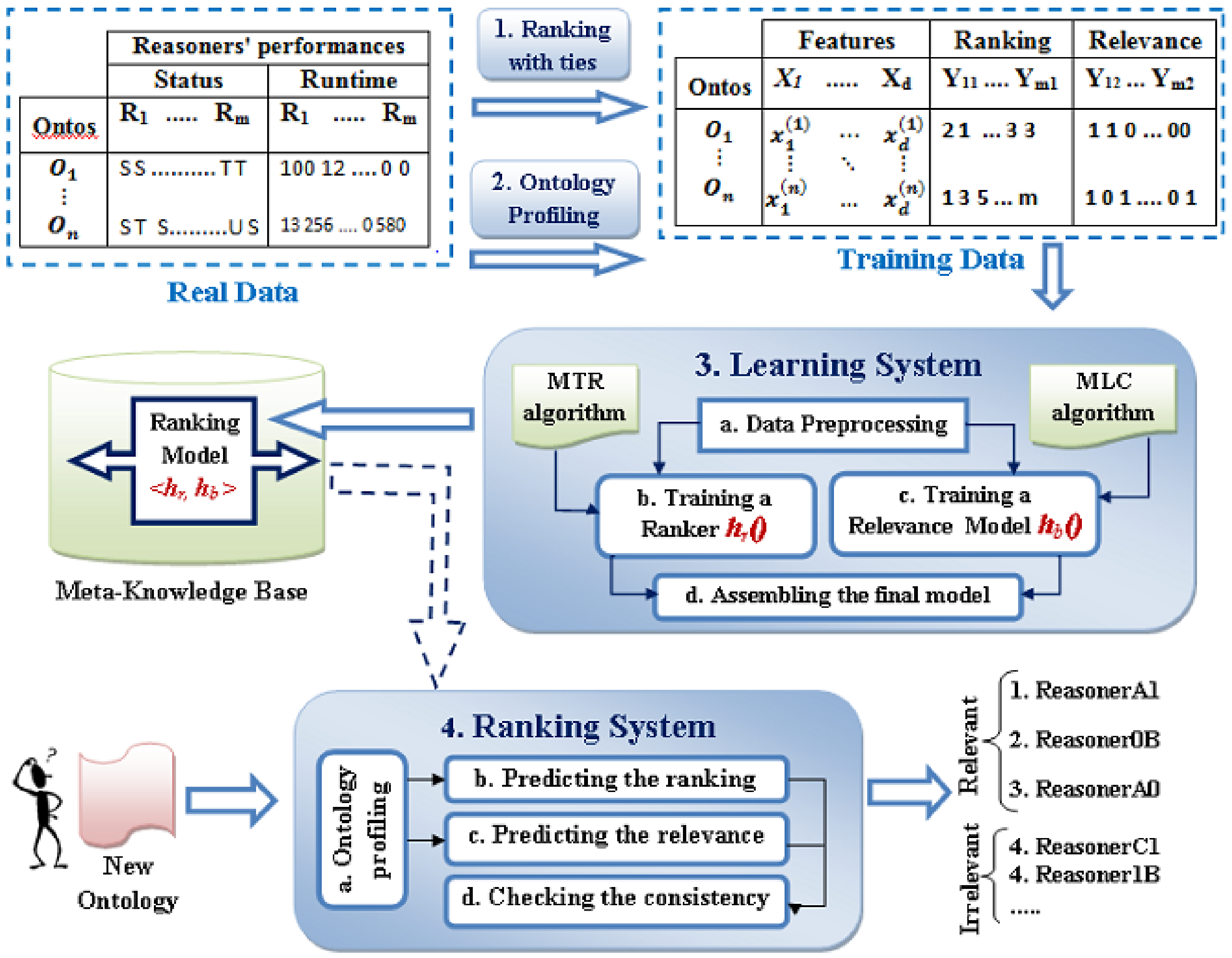


Figure 5.5: Main steps to achieve the reasoner automatic ranking according to the Extended Multi-Label Ranking approach (EMLR).

Step 3. Consistency checking at this stage, a checking method is run in order to examine the consistency of the predicted ranking and potentially to edit its values. Despite the fact that both of the materials, the bipartition and the ranking, are predicted ones and expected to show some degree of error, we choose to rely on the predicted bipartite partition as the "*baseline*" knowledge that the predicted ranking must agree with. Our assumption is that the multi-label classification models (MLC) which employ binary classification techniques are more likely to show higher performances than other types of models. Indeed, the supervised bipartite partitioning problem is a less challenging learning task than the ranking problem. Details about the operating mode of our ranking checking method are provided in the upcoming subsection.

Figure 5.5.3.2 sums up the above description. It depicts all the required training and prediction steps under the EMLR framework.

5.5.3.3 Ranking consistency checking method

Based on Definition 27, the ranking checking method must ensure that: *1)* the rank values of the relevant labels form a strict totally ordered set of natural numbers; and *2)* no relevant reasoner is ranked higher than an irrelevant one. If one of these rules is broken, then the ranking is corrected and accordingly updated. Algorithm 4 shows the steps that the function *rankingCheckingMethod()* achieves in order to satisfy these rules. The procedure takes as input the matrix $\widehat{\mathbf{Y}}$, which encodes the computed predictions. We design by $max_{\sigma}^{P_x}$ the maximal rank value of the relevant output labels. Known that the ranks take values in $\mathbb{N}^*$ and they are expected to be linearly sorted in an increasing strict order, then we can assert that $max_{\sigma}^{P_x} = |P_x|$. Through the first loop of Algorithm 4, the $max_{\sigma}^{P_x}$ value is computed and the rank values corresponding to the relevant labels are stored in the R_x array. The cells of this array which correspond to the irrelevant labels are set to 0. The resulting R_x array

Algorithm 4: The ranking consistancy checking method

```
Function rankingCheckingMethod($\widehat{\mathbf{Y}} \in M_{m\times 2}(\mathbb{R})$)
$\mathbf{R}_x \leftarrow [0, \dots, 0]$ ; // $\mathbf{R}_x$ is of size $m$
$max_{\sigma}^{P_x} \leftarrow 0$ ;
for $i \leftarrow 1$ to $|\widehat{\mathbf{Y}}|$ do
    if $\widehat{\mathbf{Y}}[i][1] = 1$ then
        $\mathbf{R}_x[i] \leftarrow \widehat{\mathbf{Y}}[i][2]$ ;
        $max_{\sigma}^{P_x} \leftarrow max_{\sigma}^{P_x} + 1$ ;
    end
end
$\mathbf{R}_x \leftarrow rankOrderTransformation(\mathbf{R}_x)$ ;
for $i \leftarrow 1$ to $|\widehat{\mathbf{Y}}|$ do
    if $\widehat{\mathbf{Y}}[i][1] <> 0$ then // Relevant label, update the rank
        $\widehat{\mathbf{Y}}[i][2] \leftarrow \mathbf{R}_x[i]$ ;
    else if $\widehat{\mathbf{Y}}[i][2] \leq max_{\sigma}^{P_x}$ then // Irrelevant label, Inconsistency case
        $\widehat{\mathbf{Y}}[i][2] \leftarrow max_{\sigma}^{P_x} + 1$ ;
    end
end
return $\widehat{\mathbf{Y}}$ ;
```

is then handled by *rankOrderTransformation()*. The main role of this function is to apply our 1^st consistency rule. The ranks in the R_x array are treated as any numerical scores. Hence, it computes the strict total order to these scores. It is built around the Freeman's algorithm [Fre70]. Ties observed in the initial R_x array[5] are arbitrary broken. For instance, let $[1, 3, 3, 3, 5, 4]$ be a predicted ranking of 6 alternatives and $[1, 0, 1, 1, 1, 0]$ their corresponding predicted relevance. In this case, the R_x array is equal to $[1, 0, 3, 3, 5, 0]$, which is obviously a non consistent[6] ranking over the P_x subset. After being edited by *rankOrderTransformation()*, the R_x array will have the following value $[1, 0, 2, 3, 4, 0]$. We should highlight that the rank transformation is computed in polynomial time with respect to the size of the output label set. Afterwards, the inconsistencies w.r.t. the second rule are caught by simply verifying whether an irrelevant label has a rank lower than $max_\sigma^{P_x}$ (see Algorithm 4, Line 14). In this case, the rank value is set to $(max_\sigma^{P_x} + 1)$, thus the irrelevant label is no more ranked lower than any relevant label. Actually, we do not have any further indication about the missed correct rank of this irrelevant label. Hence, we choose to apply an intuitive and inexpensive solution aiming at least to fix the inconsistency. As a matter of fact, in our opinion, the exact ranking of the irrelevant reasoners is less important for the user. Based on this description, we can outline 2 more inconsistencies among the irrelevant labels of the above example. Hence, by pursuing the execution of the checking method, the final edited ranking corresponding to this example will be equal to $[1, 5, 2, 3, 4, 5]$.

5.5.4 Experimental setup

In this subsection, we will describe the data and the assessment measurements that we will employ to conduct the different evaluations.

5.5.4.1 Assessment methods

To train our new multi-label ranking model, we will use the ontology corpus described in Chapter 3 and the reasoner evaluation results employed in Chapter 4 to train the robustness predictive profiles. Similarly, the assessments of the ranking quality will be conducted using the ontology test set and its corresponding reasoner evaluations employed in Subsection 5.4.2.

The quality of the multi-label ranking methods will be assessed from two different perspectives. First, we will examine the quality of the produced rankings by using some of the metrics employed in Subsection 5.4.2. More precisely, we will compute the average Kendall TauX, the MAP@1 and the MAP@6. In addition, we will examine how accurate are the deployed MLC models in distinguishing between the relevant reasoners and the irrelevant ones given any input ontology. Hence, any usual assessment metric of binary ML classification models can be used, such as the precision, recall and the

[5]The ties over values higher than 0.

[6]The ranking contains ties and is not linear.

		Predicted Labels		
		Relevant	**Irrelevant**	**Total**
Actual Labels	**Relevant**	TP=$\vert P_x \cap \widehat{P}_x\vert$	FN=$\vert P_x \cap \widehat{N}_x\vert$	$\vert P_x\vert$
	Irrelevant	FP=$\vert N_x \cap \widehat{P}_x\vert$	TN=$\vert N_x \cap \widehat{N}_x\vert$	$\vert N_x\vert$
	Total	$\vert\widehat{P}_x\vert$	$\vert\widehat{N}_x\vert$	$\vert\mathcal{Y}\vert$

Table 5.3: Confusion matrix of a bipartition task.

F1-score[7]. More in details, let $\widehat{P}_x$ and $\widehat{N}_x$ be the set of labels[8], respectively predicted as relevant and irrelevant. Thus, we can, for each individual test ontology pictured by its feature vector X, compute a two-by-two confusion matrix, C_x, of actual vs. predicted relevant and irrelevant labels. Table 5.3 illustrates such a matrix: Based of the C_x matrix of each test ontology, we will derive the F1-Score value corresponding to the computed bipartition. Then, we will averaged the F1-Score across all the testing ontologies. Besides the metrics derived from the confusion matrix, we will compute the **Hamming loss** (HM-Loss) score. The latter one is a metric of common use in the assessing of MLC models. Given a bipartition computed over an ontology X, HM-Loss computes the percentage of labels that are misclassified: $H - Loss = \frac{1}{|\mathcal{Y}|}|\widehat{P}_x \bigtriangleup P_x|$. Here, $\bigtriangleup$ stands for the symmetric difference between two sets. H-Loss will be averaged across all the test cases. A good MLC model should maximize its F1-Score value, while minimizing its HM-Loss value.

5.5.4.2 Candidate Multi-label learners

Our main concern at this stage of study is to decide which base MTR and MLC algorithms to embody in our method. It is clear that these learners are required to be as effective as possible in training our ontology reasoner dataset. Therefore, we selected some potentially powerful MTR and MLC algorithms and then compared their prediction performances. In the following, we outline the main functional features of the considered algorithms.

a) The multi-target regression algorithms All of the investigated MTR algorithms were originally introduced in [STGV16]. Specifically, we will put into comparison 3 MTR algorithms listed in the following. We should highlight that all of these algorithms apply problem transformation techniques. Thus, they require a base single label regression algorithm. In our experimentations, we will use the Random Forest[9] as the base learner.

[7]All of these metrics were previously described in Chapter 1

[8]Each label corresponds to one alternative reasoner

[9]Random Forest was described in Chapter 4 and it showed high performances when employed in reasoner runtime prediction.

- **ST**: the **S**imple **T**arget is the baseline approach of multi-target regression. According to this method, target variables can be predicted independently. Hence, the multi-target model h is composed of m elementary models h_j. Each of these models is trained separately from a reduced dataset, $\mathcal{D}_j$. Within the latter one, each instance is associated with a single value from one target variable, Y_j. Thus, the elementary model has the following form: $h_j : X \rightarrow \mathbb{R}$.

- **MTRS**: the **M**ulti-**T**arget **R**egressor **S**tacking is based on the idea of stacking the single target (ST) models. The training of MTRS is achieved within two stages: first, m ST models, $\{h_1, \ldots, h_m\}$, are trained for each target variable; then, a further set of m meta-models $\{h'_1, \ldots, h'_m\}$ are also learned. The latter models are trained from transformed datasets where the target variables and the prediction of their values are included in the feature space.

- **ERC**: the **E**nsemble of **R**egressor **C**hains is based on the idea of chaining single target (ST) models. A random chain (permutation) of the set of target variables is selected and then a separate regression model for each target is trained. However, the dataset would not be same for every target. Indeed, the feature space of the training dataset for the target Y_i will be extended by those target variables which appear in the selected chain at a position lower than i.

b) The multi-label classification algorithms A rich plethora of MLC algorithms exists in the related literature. We chose to examine some of the most commonly used ones. All the investigated algorithms are described in [ISTV10] and can produce a bipartite partition of the alternative labels. A brief description of the examined MLC algorithms is provided in what follows:

- **BR**: the **B**inary **R**elevance approach is the baseline in the multi-label classification. Like the ST approach, BR learns a separate binary classification model for each label in order to predict whether it is relevant or not. Hence, it requires a base ML single label classifier. In our experimentation, we will use the decision tree J48 algorithm as the base classifier.

- **ML-*K*NN**: the multi-label KNN is an instance based lazy approach derived from the popular k-Nearest Neighbour (kNN) algorithm. For an input instance X, the algorithm first retrieves its k nearest neighbours from the dataset according to some distance measure. Then, the maximum a posteriori principle is applied to identify the label set for this instance based on the label sets of its neighbours.

- **BP-MLL**: is a neural network approach for the multi-label classification task based on the popular back-propagation algorithm. BP-MLL extends the latter algorithm through replacing its error function with a new function defined to capture the characteristics of multi-label learning and the dependencies between the individual labels.

Algorithm	Kendall TauX	MAP@1	MAP@6
ST	0.894 (± 0.19)	0.940 (± 0.24)	0.857 (± 0.22)
MTRS	0.897 (± 0.19)	0.946 (± 0.23)	0.870 (± 0.22)
ERC	0.895 (± 0.19)	0.948 (± 0.24)	0.866 (± 0.22)

Table 5.4: Evaluation summary of MLR learning algorithms

- **RA*K*EL**: the Random K-Labelsets[10] method constructs an ensemble of label powerset (LP) classifiers. The latter ones consider each unique set of labels as one class. Hence, the problem is reduced to multi-class single label classification. However, this approach is expensive and must deal with the problem of data imbalance. RAKEL manages to take label correlations into account, while avoiding the LP's problems. Indeed, it constructs the LP ensemble where each LP classifier is trained using a different small random subset of the set of labels.

Furthermore, the quality of the final ranking computed using our extended multi-label ranking (EMLR) method will be compared to the results of our first method, which is based on single label learning. We will also assess state-of-art ranking method when training our ontology reasoner dataset. The upcoming subsection depicts the results of our different experimentations.

5.5.5 Experimental results

In the following, we start by discussing the assessment results of the investigated multi-label learning algorithms. The purpose of this investigation is to help us to select one MLC and one MTR algorithm that works well on the studied dataset. These algorithms will be incorporated in our method and used in subsequent experiments. Afterwards, the quality of the EMLR rankings as well as the achieved bipartite partitioning will be compared to those established by other different methods.

5.5.5.1 Selection results of the base learners

All of the above described MLC and MLR algorithms are already implemented within a Java library, designed for the multi-label learning. The library is called Mulan[11] [TSXVV11]. We have evaluated these algorithms using their default parameters. Results are discussed in what follows.

The reduced training dataset, $\mathcal{D}_r$, which includes 2500 ontology feature vectors associated with their respective vector of the reasoner ranking, was trained by the above described MTR algorithms.

[10] Any subset of the label space $\mathcal{Y}$ is called a *labelset*.

[11] Mulan is available for download from `http://mulan.sourceforge.net/download.html`

Measure	**BR**	**ML-*K*NN**	**BP-MLL**	**RA*K*EL**
F1-Score	0.956 (± 0.15)	0.937 (± 0.14)	0.849 (± 0.16)	0.953 (± 0.15)
HM-Loss	0.038 (± 0.10)	0.067 (± 0.13)	0.212(± 0.16)	0.039 (± 0.11)

Table 5.5: Evaluation summary of the MLC learning algorithms

Then, the quality of the learned models were checked against the testing dataset which includes 500 ontologies. Table 5.4 depicts the results of this assessment. The overall quality of the predicted rankings produced by the listed MTR models was assessed according to the Kendall TauX, the MAP@1 and the MAP@6 respective measurements. The table illustrates the average value of these measurements together with their respective standard deviation computed across all the testing samples. We can notice that the MTRS model have outperformed both ST and ERC models according to the Kendall TauX and the MAP@6 values. However, the ERC showed better performances in predicting the top of the ranked list, otherwise the most performing reasoner. Here, a choice must be made whether to pick the MTRS or the ERC algorithm. Obviously, if we were about to build a meta-reasoner like the R_2O_2 [KKL15], then we would preferred the ERC algorithm. However, our main goal is to provide insights about all of the examined reasoners. Hence, it is important for us to grant the quality of the full ranking. That is why we choose to carry on with the MTRS algorithm. It is also important to notice that all of the above reported assessment values exceed those achieved with our first reasoner ranking method, over the same testing dataset. For instance, the average Kendall TauX of our first method reported in Table 5.2, Subsection 5.4.2 is about 0.68, while it is over 0.895 using the MTR algorithms. The difference is remarkable and promising one. However, we must outline that at this stage of the study the computed rankings are still not consistent ones. They still need to be checked and eventually modified to match the predicted relevance of the reasoners. Therefore, we will later inspect whether the remarked improvement in the ranking quality is maintained.

Similar to our first set of experiments, we learned MLC models from our second reduced training dataset, which includes 2500 ontology feature vectors associated with their respective vector of reasoner relevance. Then, the quality of the learned models were checked against the testing dataset. Table 5.5 sums up the assessment results. The predictive power of the examined models was checked in terms of the F1-Score and the Hamming Loss score. Table 5.5 reports the average value together with the standard deviation of each measurement. According to these values, we can state that the BR and the RA*K*EL models can better distinguish between the relevant/irrelevant reasoners than the other models. Moreover, the BR model showed the highest F1-Score while maintaining the lowest Hamming Loss Score. Hence, we will rely on this algorithm to build in our multi-label ranking method of ontology reasoners.

To sum up, we selected the Multi-Target Regressor Stacking (**MTRS**) algorithm as our base MTR learner. It will be employed to train our ranking model, $h_r()$. On the other hand, we selected the Binary Relevance (**BR**) algorithm as our base MLC learner. Henceforth, it will be used to derive our bipartite partitioning model, $h_b()$.

5.5.5.2 Comparative study of the reasoner ranking methods

The purpose of this section is to provide an empirical evaluation of the extending multi-label ranking (**EMLR**) method when all the prediction steps are applied. Furthermore, we intend to demonstrate that the EMLR method yields to better results than those achieved by our first single label based method. In order to ease the discussing, we will denote by **SLR** the latter method. The rankings predicted by the **EMLR** and by the **SLR** respective methods are comparable ones. Indeed, the assessment results of the SLR predicted rankings over our testing dataset was already reported in Subsection 5.4.3. Nevertheless, the SLR method produces a multi-class partitioning[12] of the output label set, rather than a bipartite one. The partitions stand for the buckets $\mathcal{B}_S$, $\mathcal{B}_U$,$\mathcal{B}_T$ and $\mathcal{B}_U$. Hence to be able to set up the comparison, we rearranged the SLR predicted partitions according to the rules of Definition 27.

On the other hand, we want to prove that our method is competitive with other state-of-art multi-label ranking solutions. The closest existing work is the calibrated label ranking (**CLR**) method. According to [FHLMB08], the CLR method and more specifically, the calibrated ranking by pairwise comparison, denoted by **CRPC**, can predict both the ranking and the bipartite partitioning of the output label set. However, the only available implementation of this method, which is part of the Mulan library, is exclusively designed to produce a bipartite partitioning. Actually, the method can only process datasets having an output space of the form $\{0,1\}^m$. Hence, we will denote by $\textbf{CLR}_{mulan}$ this implementation version. The quality of our predicted bi-partitioning will be compared to those produced by CLR_{mulan} and SLR. In order to expand our study, we will compare the quality of our rankings to those produced by existing *label ranking* (LR) solutions. The ranking models of three algorithms will be assessed: (1) the nearest neighbor-based approach for ranking (**LR-KNN**); (2) the predictive clustering trees for ranking (**PCTR**); and (3) the label ranking trees (**LRT**). Details about these algorithms can be found in [SP13]. The rational behind selecting these algorithms is two-folded: first, they are the building blocks of R_2O_2 [KKL15], the ontology meta-reasoner[13], which employs the label ranking techniques; then, these algorithms could be slightly adjusted so they could produce a

[12]The predicted partitions are those stored in the **B** vector encoded by Algorithm 3.

[13]We should highlight that we did not get access to R_2O_2 executable. Then, we were enable to establish any comparison with this system.

Measure	**EMLR**	**SLR**	CLR_{mulan}
F1-Score	0.956 (± 0.15)	0.958 (± 0.13)	0.955 (± 0.15)
HM-Loss	0.038 (± 0.10)	0.040 (± 0.10)	0.038 (± 0.10)

Table 5.6: Summary of the bipartite partitioning quality achieved by the examined methods

ranking with ties. It is to be noted that these algorithms are restricted to rank the labels and do not separate them into relevant and irrelevant ones.

All the examined learning models were derived from our training dataset that we have already used in our different experimentations. The dataset was adapted to the learning process of each examined method. Then, the predictions about reasoner ranking and relevance were computed for every ontology in our testing dataset. We start the discussion by examining the quality of the predicted bipartite partitioning, then the predicted the ranking.

Bi-partitioning quality Table 5.6 illustrates the assessment results of the achieved bipartite partitioning by those methods capable to produce such a prediction. We can notice, at a glance, that the values of the assessment measures of the three methods are very close to each other. In overall, our proposed methods EMLR and SLR showed very good prediction capabilities in terms of F1-score and the Hamming Loss (HM-Loss) score. Actually, our F1-scores are even better than those achieved by the well recognized calibrated method (CLR$_{m}ulan$). The gain can seem modest, but nonetheless it witnesses the effectiveness of our proposals and proves their competitiveness comparing to existing MLC solutions. Surprisingly, when the purpose is to win F1-Score, our single label method SLR outperforms the other multi-label based method. It showed a slightly higher[14] HM-Loss comparing to EMLR and CLR$_{m}ulan$, but the value remains good. Actually, the classification according to the relevance obtained by the EMLR method corresponds to those of Binary Relevance (BR) algorithm. We recall that the latter one treats each output label independently and trains a binary to predict their respective relevance. The decision tree algorithm (J48) is the base single label learner. The design of the SLR method is very similar to the BR, since that one predictive model is trained for each output label (this is the multi-class model part of a reasoner profile). The main difference is that SLR employs different base learners, one for each output label. These learners were selected after excessive experimentation to grant that they are the most performing ones given the data under training.

Ranking quality. Table 5.7 shows the assessment results of the ranking quality achieved by the six listed methods. Several interesting observations can be drawn from our experimental results.

[14]We recall that HM-Loss should be minimized

Measure	EMLR	MTRS	SLR	LR-KNN	PCTR	LRT
Kendall-TauX	0.899 (± 0.20)	0.897(± 0.19)	0.680 (± 0.25)	0.825 (± 0.24)	0.730 (± 0.26)	0.821 (± 0.23)
MAP@1	0.954 (± 0.21)	0.946 (± 0.23)	0.920 (± 0.27)	0.918 (± 0.27)	0.860 (± 0.34)	0.908 (± 0.28)
MAP@6	0.874 (± 0.22)	0.870 (± 0.22)	0.620(± 0.23)	0.773 (± 0.28)	0.688 (± 0.29)	0.758 (± 0.27)

Table 5.7: Summary of the ranking quality achieved by the examined methods

First, we should highlight that the goal behind comparing the EMLR ranking results to those of the MTRS (previously reported in Table 5.4) is to examine the impact of the ranking correction step. We can notice that EMLR improves the overall quality of the predicted rankings comparing to MTRS. Indeed, the EMLR's Kendall TauX and MAP@6 scores are higher than those of MTRS. These seemingly small improvements are statistically highly significant ($p \ll 0.0001$) according to the paired t-test. On the other hand, the hypothesis that MTRS outperforms EMLR w.r.t. the MAP@1 score was rejected by the same statistical test. Based on these findings, we can assert that our proposed procedure, which checks and corrects the consistency of a predicted ranking, is effective and can contribute to the improvement of the quality of this ranking.

Another interesting conclusion is that our multi-label based method, EMLR, remarkably increases in the quality of reasoner ranking compared to our single label based method (SLR). The achieved gain is significant according to the paired t-test with $p \ll 0.0001$. It is around 23.88% and 28.74% respectively in Kendall TauX and in MAP@6 scores. This means that with EMLR model we are better ranking the full set of alternatives and we are more capable to predict the ties. Similarly, the improvement in predicting the top most robust reasoner is about 3.77% w.r.t. SLR and about 4.61% comparing to the default method. The improvement may be due to the fact that EMLR explores the dependency across the output label values, otherwise the rank values corresponding to the different reasoners. The SLR method does not handle such information. The performances of each reasoner are independently predicted and then ranked.

More interestingly, EMLR method have outperformed all of its counterparts, LR-KNN, PCTR and LRT, w.r.t. the all assessments measures. This result is important since these are well recognized algorithms in the field of label ranking. Besides, they are part of R_2O_2, the only existing reasoner ranking system in our related literature. The empirical results could be explained by the fact that none of the above methods is originally designed to predict a ranking which includes ties. Even that they can handle such ranking, they are not particularly optimized for. Once again, these findings show that our proposed EMLR method adds good multi-label ranking abilities to the already strong existing solutions. Hence, we would assert that it was worthwhile to investigate time and effort in exploring the multi-label techniques and in accommodating them to the problem of automatic ranking of ontology reasoners.

5.5.6 Complexity analysis

In this section, we discuss the time complexity of all methods capable to produce both the ranking and the relevance based partitioning of a set of alternative reasoners: SLR, EMLR and the original CLR [FHLMB08]. All of these methods depend on at least one base single label learner (e.g. the Random Forest, the Decision Tree, etc.). Therefore, the computational complexity of each of these methods is approximated as a function of the computational complexity of the base learner(s) it employs.

We suppose that for a dataset of n ontologies, each described by d features, the training complexity of a single label learner is $O(G_{tr}(n,d))$ and the prediction complexity is $O(G_{Pr}(n,d))$. Worth noting, the training complexity varies from one single label algorithm to another. However, for most algorithms (e.g. Random Forest), the computational cost of making predictions for n examples is much smaller than the cost of training on n other examples. We should also highlight that the same set of *computationally cheap* ontology features (cf. Chapter 3, Subsection 3.3.2) were deployed in all the assessed reasoner ranking methods. Let m be the number of the alternative reasoners to be ranked, which stands for the number of the output labels in a multi-label learning model. Then, the computational complexity of the assessed ranking methods is detailed in what follows:

SLR : the training complexity of our single label based method applied to the reasoner ranking problem (SLR) is about $O(2.m.G_{tr}(n,d))$. This is because for each reasoner 2 single label models (multi-class model, regression model) are trained and stored in the robustness profile. However, different single label learning algorithms were employed to train these models with different computational complexity. This is why, the $O(G_{tr}(n,d))$ stands for the upper bound of the training complexity achieved across all of these algorithms. Similarly, the $O(G_{Pr}(n,d))$ is their corresponding upper bound prediction complexity. We recall that at the prediction time, a reasoner regression model[15] would not be invoked unless the success of this reasoner, it has been predicted. Let t be the number of regression models called at the prediction time, obviously, t takes values in $[0,m]$. On the other hand, the m reasoner multi-class models will be used. Hence, the prediction complexity using the SLR method is about $O((m+t).G_{Pr}(n,d))$. We should highlight that the processing operations relative to the ranking computation as established in Algorithm 3 could be achieved in a polynomial time w.r.t. the size of the output labels set, i.e. the m value.

EMLR : the extended multi-label ranking method of ontology reasoners involves training of two sep-

[15]This models estimate the reasoner runtime.

arate models: the MTRS multi-target regression model and the BR multi-label classification model. The training and prediction complexities of the BR method are $O(m.G'_{tr}(n,d))$ and $O(m.G'_{Pr}(n,d))$ respectively, as it involves training and querying m independent single-label binary models. The $O(G'_{tr}(n,d))$ corresponds to the training complexity of the decision tree (J48) algorithm employed by BR as the base single label learner. On the other hand, according to [STGV16], the training and prediction complexities of the MTRS algorithm are $O(m.(G''_{tr}(n,d) + G''_{tr}(n,d+m)))$ and $O(m.(G''_{Pr}(n,d) + G''_{Pr}(n,d+m)))$ respectively. Nevertheless in most cases, including our learning dataset, the number of the output labels is much smaller than the number of the input features, i.e. $m \ll d$. Hence, the effective training and prediction complexities of MTRS become $O(m.G''_{tr}(n,d))$ and $O(m.G''_{Pr}(n,d))$ respectively. In our experimentation, the base single label learner employed by MTRS is the Random Forest. Given these analysis, we can assert that the training complexity of the EMLR method is equal to the sum of the above complexities, this means $O(2.m.G_{tr}(n,d))$ such that $O(G_{tr}(n,d))$ is the upper bound complexity of both $O(G'_{tr}(n,d))$ and $O(G''_{tr}(n,d))$. At the prediction time, EMLR applies a correction step in addition to the prediction operations. As it could be observed from Algorithm 4, all the required operations could be achieved in a polynomial time w.r.t. the size of the output labels set.

CRPC : in [ZZ14], the authors have established that at the training time the computational complexity of the calibrated label ranking method is equal to $O(m^2.G_{tr}(n,d))$, known that this method employs the ranking by pairwise comparison, i.e. the RPC algorithm. Indeed, RPC trains and stores a quadratic number of single label binary models. The $O(G_{tr}(n,d))$ is the training complexity of the base learner employed to train these models. In our experimentation, we set this learner to the decision tree (J48) algorithm. Hence, the computational complexity at the prediction time using a CRPC model is equal to $m^2.O(G_{Pr}(n,d))$.

Based on these findings, we can assert that training EMLR or SLR models require less computational cost than the CRPC model. This is also true when computing the predictions. This result is very interesting one, since we have already showed that EMLR model could achieve better prediction quality than the CRPC model. Hence, our learning method could improved the prediction capabilities of the multi-label ranking system without any additional cost. It is worth noting that the training complexity of EMLR and SLR methods is almost the same. However, the SLR method could be used to compute the predictions with slightly lower computational cost. Actually, the difference in the prediction complexity is occasional and depends on the data.

5.6 ExOH: Extracting the Empirically Hard Ontology Modules

Under our advising system ADSOR, the empirical hardness of an ontology is adjudged with respect to a set of reasoners using predictive modelling techniques (c.f Chapter 4, Definition 24). An ontology predicted not to be hard entails that all of the examined reasoners can correctly classify it within the fixed time limit. Hence, a user could safely select any of these reasoners to employ it in its application. However, when the ontology is expected to be a hard one, a user would probably look to gain further insights about the causes of this hardness. One possible assistance that our system could offer is to pinpoint the ontology subsets which are the most computationally demanding ones. These subsets are commonly called the ontology *hotspots* [GPS12]. Hotspots are ontology modules, identified as performance bottlenecks for reasoning. An efficient identification of these modules is highly desirable, since they represent re-factoring opportunities for both the ontology and the reasoner designers. We revisited both the Gonçalves et al. [GPS12] and the Kang et al. [KLK14] respective approaches of hotspot identification. We suggested a novel method to extract this particular kind of ontology modules, which takes into account the specificity of our study context. Before providing more details about our extraction approach, we recall some relevant notions in the ontology *modularization* field of research [SPS09].

5.6.1 Basic notions of ontology modularization

Ontology *modularization* has an important part of the ontology engineering field. It aims at reducing the complexity and the size of an ontology, either by breaking it down to all its constituent sub-parts (i.e. *ontology partitioning*) or by extracting only a small part of the ontology (i.e. *module extraction*). A rich plethora of ontology modularization approaches and techniques exist in the related literature, some of them were consolidated in the Stuckenschmidt et al.'s book [SPS09]. In this section, we will provide some basic notions of the ontology modularization, which are relevant to our research work. More precisely, we will explain the concepts of ontology *module* and the ontology *atomic decomposition* and we will briefly describe some of the well known ontology module extraction methods.

Ontology module and the syntactic locality extraction. In the related literature, there are many sorts of "*logically founded*" definitions of ontology modules, most of them are based on *deductive conservative* extensions [SPS09]. Roughly speaking, a *module* is a subset of an ontology that includes all the axioms required to define some subset of its knowledge. By referring to [GHKS08], an ontology module could be formally defined as follows:

Definition 29 (Module) *Let $\mathcal{O}$ be an ontology and Σ be a signature*[16]*. A subset $\mathcal{M}$ of $\mathcal{O}$ is said to be a module w.r.t. Σ if for every axiom α with the signature $\widetilde{\alpha} \subseteq \Sigma$, $\mathcal{M} \models \alpha \Leftrightarrow \mathcal{O} \models \alpha$.*

In other words, $\mathcal{M}$ preserves all the entailments formulated using only the symbols defined within a signature Σ. Extracting minimal modules is computationally expensive task. Looking to overcome this issue, Grau et al. [GHKS08] have introduced the concept of *syntactic locality* based module. This is an approximation about how a minimal module would be. An axiom is called ($\perp$-) local w.r.t. a signature Σ if replacing all symbols not in Σ with $\perp$ makes the axiom a *tautology*. Thus, we can testify that this axiom is not in relation with symbols in Σ. Otherwise, we would not run the risk of altering the meaning of the Σ terms when adding this axiom to the ontology module defining over Σ. Hence, the locality based module w.r.t. Σ is made up from axioms that are non local to Σ. The checking of axiom locality is processed without involving a reasoner. It is rather accomplished by examining the syntactic structure of an axiom and by matching it to some generic patterns. In this thesis, we restrict our attention to modules based on *syntactic locality*, because they are logically sound, work for all of OWL and often behave well in practice.

The idea of extraction syntactic locality modules was recently revisited by Martín-Recuerda and Walther [MRW14]. Authors have introduced the notion of an *axiom dependency hypergraph* (ADH), a novel graph based representation of OWL ontologies. Briefly, an hypergraph [GLPN93] is a generalization of a simple graph where edges, called *hyperedges*, can connect more than two vertices. Accordingly, the ADH of an OWL ontology is a directed hypergraph where vertices are axioms and each directed hyperedge connects one or more vertices. The latter ones are arranged in the *tail* of the hyperedge with only one vertex placed at the *head* of the hyperedge. The locality concept described above is encoded within these hyperedges. More specifically, let α be the axiom placed at the head of a directed hyperedge h_e. α is *non-local* with respect to the signature Σ made up from terms used in the minimal axiom set placed at the tail of this hyperedge h_e. Given this representation, authors proposed a novel algorithm for computing locality-based modules. Roughly speaking, a module of an ontology corresponds to a set of connected vertices in the *axiom dependency hypergraph*. Experimental results of the ADH based method have shown a significant improvement in the time needed to extract locality-based modules comparing to state-of-the-art implementations of the Grau et al.'s method.

Atomic decomposition. According to Grau et al. [GHKS08], the number of all possible modules of an ontology can be exponential w.r.t. its size, otherwise the number of its terms or axioms. Hence, the Atomic Decomposition (AD) was introduced by Del Vescovo et al. [DPSS11] in order to scrutinize the modular structure of an ontology. Indeed, the atomic decomposition gathers in a compact way all

[16]A signature stands for a set of terms.

the modules of an ontology. Roughly speaking, an ontology *atom* is a subset of an ontology, whose axioms always co-occur in modules. Formally, an atom is defined as follows:

Definition 30 (Atom) *An atom* $\mathfrak{a}$ *is a maximal set of axioms of an ontology* $\mathcal{O}$ *such that for every module* $\mathcal{M}$ *of* $\mathcal{O}$ *either* $\mathfrak{a} \cap \mathcal{M} = \mathfrak{a}$ *or* $\mathfrak{a} \cap \mathcal{M} = \emptyset$*.*

A dependency between atoms exists and mirrors the subsequent relation across the corresponding modules. This dependency is formally defined as follows:

Definition 31 (Atoms Dependency Relation) *An atom* $\mathfrak{b}$ *depends on an atom* $\mathfrak{a}$ *in an ontology* $\mathcal{O}$ *(written* $\mathfrak{a} \succcurlyeq_{\mathcal{O}} \mathfrak{b}$*) if* $\mathfrak{b}$ *occurs in every module of* $\mathcal{O}$ *containing* $\mathfrak{a}$*:* $\forall$ *module* $\mathcal{M}$*, if* $\mathfrak{a} \subseteq \mathcal{M}$*, then* $\mathfrak{b} \subseteq \mathcal{M}$*.*

Based on the above definitions, an Atomic Decomposition of an ontology $\mathcal{O}$ is a graph $G = \langle \mathbf{Atoms}^x_{\mathcal{O}}, \succcurlyeq_{\mathcal{O}} \rangle$ where $\mathbf{Atoms}^x_{\mathcal{O}}$ is the set of all atoms of $\mathcal{O}$.

Del Vescovo et al. [DPSS11] have defined a tractable algorithm for computing the atomic decomposition for the locality-based modules of Grau et al. [GHKS08]. First, the module for a signature of every axiom is built, then axioms with equivalent modules are combined into a single atom. After the set of atoms is known, their modules are explored to derive dependencies among the atoms. The runtime of this algorithm was significantly improved by Martín-Recuerda and Walther [MRW14], who have introduced the hypergraph based representation of axiom dependency. In fact, they have employed this representation to efficiently compute the locality based modules, then extended their work to compute the full atomic decomposition of the ontology. In their approach, atoms of an ontology correspond to strongly connected components of the axiom dependency hypergraph.

5.6.2 Study of ontology hotspot extraction solutions

According to Gonçalves et al. [GPS12] whenever an ontology is predicted to be performance heterogeneous (c.f. Chapter 2), then this ontology may include some small subsets of logical axioms whose interaction with the remainder is performance-degrading. They called them the ontology **Hotspots**. Authors suggested that it is possible to find them. They also highlighted that the removal of a hotspot could improves or degrades the reasoner performance. They focussed on those which the removal could decrease reasoning time for the remaining ontology. Let $|\mathcal{O}|$ denotes the size of an ontology $\mathcal{O}$ and $RT(\mathcal{O}, R)$ denotes the reasoning time for an ontology $\mathcal{O}$ using a reasoner R. An ontology *hotspot* is formally defined in [GPS12] as follows:

Definition 32 (Hotspot) *Given an ontology* $\mathcal{O}$ *and reasoner* R*, a subset* $\mathcal{M} \subsetneq \mathcal{O}$ *is a hotspot of* $\mathcal{O}$ *for reasoner* R*, if* $|\mathcal{M}| \ll |\mathcal{O}|$ *while* $RT(\mathcal{O} \setminus \mathcal{M}, R) \ll RT(\mathcal{O}, R)$*.*

The hotspot identification algorithm suggested by Gonçalves et al. [GPS12] takes as input an ontology $\mathcal{O}$ and a reasoner R. It starts by collecting the satisfiability checking time (SAT) for all atomic concepts in $\mathcal{O}$ using R. The concept C with the highest runtime is picked. Then, a signature $\Sigma \subseteq \widehat{\mathcal{O}}$ which includes all the terms co-occuring with C is collected. Afterwards, a locality based module $\mathcal{M}_\Sigma$ for this signature is extracted from $\mathcal{O}$. The module $\mathcal{M}_\Sigma$ is identified as a *Hotspot*, if $|\mathcal{M}_\Sigma| \ll |\mathcal{O}|$ while $RT(\mathcal{O} \setminus \mathcal{M}_\Sigma, R) \ll RT(\mathcal{O}, R)$. The algorithm repeats these steps with a newly selected concept, until a prescribed number of hotspots is found (e.g., 3) or a prescribed max number of concepts is reached (e.g., 1000). Recently, Kang et al. [KLK14] argued that this algorithm may be very time-consuming, as the sub-ontologies $(\mathcal{O} \setminus \mathcal{M}_\Sigma)$ need to be classified repeatedly for already large and difficult ontologies, and satisfiability needs to be checked for all classes in the ontology. Hence, they proposed to optimize it by replacing the exact reasoning time $RT(\mathcal{O} \setminus \mathcal{M}_\Sigma, R)$ by a predicted one, computed using their reasoner runtime regression model. Henceforth, the locality model is extracted for each class in the ontology, without prior checking of its satisfiability. Only the modules with a size lower than some predefined threshold are accepted as a candidate modules for the hotspot identification procedure. This procedure operates with two steps. First, the features of the module are computed then, the classification time of the reasoner R over this candidate module is predicted. Finally, the k candidates with the highest predicted classification time are returned as potential hotspots. In the empirical evaluations of Kang et al., a retrieved potential hotspot module, $\mathcal{M}_c$, is flagged to be an actual one, whenever it includes at most 10% from the original axiom set, and the actual reasoning time of its corresponding residual ontology, $\mathcal{O} \setminus \mathcal{M}_c$, is lower by 30% from that of the original ontology. Using these rules, Kang's et al. succeeded to uncover more hotspots than those identified by Gonçalves et al. for the same examined ontology, reasoner couples.

We can assert that the Kang's et al. improvements of the hotspot extraction algorithm is interesting in the sense that it proves the benefits of the predictive models when applied to ontology engineering tasks. However, we find that the major drawback of both the above described algorithms lies in the way the signatures were collected to extract the modules. Indeed, there is no proof that choosing the concepts and their co-occurring terms as staring points for module extraction may entail the identification of all of the ontology modules. Indeed, we have previously recalled that the number of the extractable locality modules is exponential w.r.t the size of the ontology. Nevertheless, the atomic decomposition of an ontology offers a condensed representation of all of these modules. Hence, we are convinced that it would be worthwhile to choose the ontology atoms as starting points to recover the modules. Furthermore, Del Vescovo et al. [DPSS11] have asserted that by definition locality modules can highly overlap with each other. Such property was not took into consideration in neither Gonçalves et al., nor Kang et al. respective algorithms. Actually, limiting the total number of extracted modules or restricting their sizes does not guarantee that these modules are not highly overlapping. All of these findings will be considered when designing our novel approach of extracting the ontology *hard* modules.

5.6.3 Design description of hard modules extraction approach

The main purpose of this section is to introduce our design ideas about a novel approach of hard modules extraction from OWL ontologies. We start by establishing the problem definition, then providing an overview of our solutions together with a justification of our modelling choices. Afterwards, we detail the description of our approach by outlining its main algorithm. It is to be noted that all the terms related to graph theory employed in this section are defined in Annexe A.

5.6.3.1 Specification of the extraction solution

Under our study context, the main goal behind trying to identify the ontology hotspots is to provide users with some guidelines whenever their ontology is predicted to be a *hard* one. Given this specific situation, it was clear that we have to review the definition of an ontology hotspot in order to adjust it to fit our requirements. Two primary modification will be put forward:

1. a hotspot will be identified with respect to a set of reasoners, rather than a single one.
2. hotspot is not just an ontology subset which requires more classification time than its complement, but the one which would alter the *robustness* of at least one reasoner from a given set.

The concept of reasoner robustness was previously introduced in Chapter 4, Definition 22. It entails that a reasoner must be efficient and correct when classifying an ontology under some robustness judgement constraints. Hence, an ontology which induces the failure (timeout, execution error or unexpected results) of a reasoner is said to be *hard*. Our main assumption is that such ontology may include one or more reduced subsets that are *hard* regarding the reasoning tasks. The formal definition of these subsets is provided in the following. In the remaining of this thesis, we called them the *hard locality modules* .

Definition 33 (Hard locality module) *Given an empirically hard ontology $\mathcal{O}$ w.r.t of reasoner set* $\mathbf{R}$*, a locality module $\mathcal{M} \subsetneq \mathcal{O}$ is said to be* hard *for* $\mathbf{R}$*, if $|\mathcal{M}| \ll |\mathcal{O}|$ and at least one reasoner from* $\mathbf{R}$ *is not robust over $\mathcal{M}$.*

We should highlight that we **are not asserting** that any ontology would have at least one hard module. In our opinion, some ontologies may be hard when handled as a *whole*. We believe that to

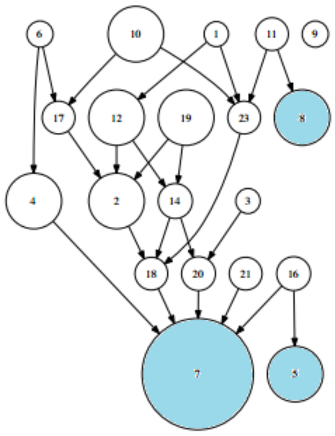


Figure 5.6: Example of the atomic decomposition graph with sinks highlighted in light blue. The graph is originally represented in [DPSS11].

achieve our purpose of extracting the hard locality modules, we should bring respond to the following couple of research questions:

1. How to extract enough modules which covers most of the ontology knowledge?
2. How to identify a hard locality module?

Our solution ideas to the above issues are described in what follows:

Locality module extraction mechanism When discussing the related works, we argued that it is more reasonable to choose the signature of the axiom atoms as the starting point to extract the locality modules. Actually, the atomic decomposition of an ontology can be viewed as a compact representation of all the modules in it. However, the number of these atoms can also be as high as the number of the logical axioms in the ontology [DPSS11]. Hence, we need to establish a selection rule of those atoms to be prioritized to achieve the module extraction task. To achieve this purpose, we examined the properties of the atomic graph (Definition 31) and put our focus in a particular one. This property asserts that if an atom $\mathfrak{b}$ depends on an atom $\mathfrak{a}$, then $\forall$ module $\mathcal{M}$, if $\mathfrak{a} \subseteq \mathcal{M}$, then $\mathfrak{b} \subseteq \mathcal{M}$. Hence, we think that it suffices to extract the module corresponding to the $\mathfrak{a}$, to get the $\mathfrak{b}$ covered. Given that the atoms' dependency relation is **transitive** [DPSS11] and all the related atoms are arranged as a directed graph, we choose to extract the locality modules *only* from the *sink* atoms in this graph. We recall that by definition a **sink** vertex in a directed graph, v, is the one having incident vertex, i.e. $indegree(v) > 0$, and one outgoing edges, i.e. $outdegree(v) = 0$. Figure 5.6 depicts an example of the atomic decomposition graph, which was originally illustrated in [DPSS11]. We highlighted with light blue the sink atoms of this graph.

This exploration strategy of the atomic decomposition graph seems to be sound, intuitive and could be efficiently processed. Whereas, there is no proof that the modules extracted based on the signatures of the sink atoms are hard ones. Hence, we proposed a back up strategy that will be applied whenever the first search round have not lead to the identification of a hard module. The hypothesis behind our second exploration strategy states that extracting modules based on the signature of an ensemble of interconnected atoms may yield to the identification of the hard modules. More specifically, an ensemble of atoms represents a *connected component* in the ontology atomic graph. Retrieving these components could easily be achieved, whenever the ontology atomic graph is a *disconnected* one. In case of a *connected* atomic graph, we will apply a standard graph decomposition algorithm [Hol92] to compute the components. Afterwards, the signature corresponding to the set of atoms within a connected component will be used for locality module extraction.

Hard locality module identification mechanism Inspired by the Kang et al. work [KLK14], we propose to employ our binary classification model for ontology overall hardness prediction[17], to adjudge whether an extracted locality module is potentially a hard one. Modules will be considered as entirely standalone ontologies. Hence, their features could be computed and submitted to the ML classification model to get the predictions. As we showed in Chapter 4, by referring only to the computationally inexpensive features highly accurate predictions about the ontology hardness could be achieved. These features are computed in a polynomial time with respect to the size of the ontology (resp. the locality module). Furthermore, our a binary classification module was trained by **L2SVM**, which is a linear supervised machine learning algorithm. The prediction step using this kind of ML models takes constant time. Hence, the overall complexity of predicting the hardness of is still polynomial w.r.t the size of the ontology module. Given these findings, we can assert that iteratively predicting the hardness of each extracted locality module won't yields to dramatical increase in the computing time.

5.6.3.2 Description of the extraction algorithm

Our hard locality module finding techniques are described in Algorithm 5. This is the main algorithm in the **ExOH** component of the ADSOR system. We should highlight that we decided to work with the Martín-Recuerda and Walther [MRW14] locality module extraction and ontology atomic decomposition methods. Their solutions are highly efficient and more scalable comparing to the other state-of-art solution. The algorithms of Martín-Recuerda and Walther are accessible from their Java

[17]The ML model is called M_{OOH} and was described in Chapter 4, Section 4.5.

tool, *Hys*[18]. These algorithms will be treated as black box tools to compute the required atoms and modules from an ontology under examination. Algorithm 5 takes as input a user ontology, $\mathcal{O}$ and

Algorithm 5: Hard modules identification algorithm

```
Input  : Ontology file $\mathcal{O}$, the minimum tolerated overlap $min_{ov}$.
Output: Predicted hardness status $h_{\mathcal{O}}$ and the set of the candidate hard modules $\mathbf{M}^H_{\mathcal{O}}$.
1  $\mathbf{M}^H_{\mathcal{O}} \leftarrow \emptyset$ ;
2  $h_{\mathcal{O}} \leftarrow predictTheEmpiricalHardness(\mathcal{O})$ ;
3  if $h_{\mathcal{O}} = \textbf{\textit{true}}$ then // Hard ontology
4      $\mathbf{ADH}_{\mathcal{O}} \leftarrow computeAxiomDependancyGraph(\mathcal{O})$ ;
5      $\mathbf{Atoms}^x_{\mathcal{O}} \leftarrow computeTheAtomicDecomposition(\mathcal{O}, \mathbf{ADH}_{\mathcal{O}})$ ;
6      foreach $\mathfrak{a} \in \mathbf{Atoms}^x_{\mathcal{O}}$ do
7          if $outDegree(\mathfrak{a}, \mathbf{Atoms}^x_{\mathcal{O}})=0$ and $inDegree(\mathfrak{a}, \mathbf{Atoms}^x_{\mathcal{O}}) \geq 1$ and
           $notIncludedAtom(\mathfrak{a}, \mathbf{M}^H_{\mathcal{O}})$ then
8              $\mathcal{M}_{\mathfrak{a}} \leftarrow extractLocalityModule(\mathfrak{a}, \mathbf{ADH}_{\mathcal{O}})$ ;
9              $h_{M_a} \leftarrow predictTheEmpiricalHardness(\mathcal{M}_{\mathfrak{a}})$ ;
10             if $h_{M_a} = \textbf{\textit{true}}$ then // Hard module
11                 $V_{M_a} \leftarrow computeTheOverLap(\mathbf{M}^H_{\mathcal{O}}, \mathcal{M}^{\mathfrak{a}})$;
12                 if $V_{M_a} \leq min_{ov}$ then // Acceptable overlap percentage
13                     $\mathbf{M}^H_{\mathcal{O}} \leftarrow \mathbf{M}^H_{\mathcal{O}} \cup \{\mathcal{M}_{\mathfrak{a}}\}$ ;
14                 end
15             end
16         end
17     end
18     if $\mathbf{M}^H_{\mathcal{O}} = \emptyset$ then // No hard modules
19         $\mathbf{M}^H_{\mathcal{O}} \leftarrow extractFromADGraphSubComponents(\mathbf{M}^H_{\mathcal{O}}, \mathbf{Atoms}^x_{\mathcal{O}})$ ;
20     end
21 end
22 return $(h_{\mathcal{O}}, \mathbf{M}^H_{\mathcal{O}})$;
```

a threshold value, min_{ov}. The latter one stands for the amount of the tailored overlap across the extracted modules. The min_{ov} value is a system parameter which varies from one ontology to another and it is set after preliminary experimentations. The algorithm starts by predicting the empirical hardness of the introduced ontology. This entails computing its features and then applying our binary

[18] The tool is available at http://lat.inf.tu-dresden.de/~dwalther/software/hys/

ML classification model. The prediction status computed by this model, $h_{\mathcal{O}}$, will be returned by the algorithm, together with the identified set of the locality hard modules, $\mathbf{M}^H_{\mathcal{O}}$. Initially, $\mathbf{M}^H_{\mathcal{O}}$ is an empty set. Whenever, $\mathcal{O}$ is predicted to be *hard*, Algorithm 5 begins the identification steps of the hard locality modules. Hence, the axiom dependency hypergraph, $\mathbf{ADH}_{\mathcal{O}}$, is computed at first. Then, the atomic decomposition is performed by invoking the HyS tool. Afterwards, all the atoms arranged in the atomic decomposition graph $\mathbf{Atoms}^x_{\mathcal{O}}$ are processed iteratively. As explained above, the locality modules will be extracted only from the signature of sink atoms. To ensure that an atom is a sink one, the algorithm compute its indegree and outdegree values given the atomic graph $\mathbf{Atoms}^x_{\mathcal{O}}$. The algorithm also checks that no previously extracted hard module contains this atom. This condition was put forward to reduce the overlap across the modules. Afterwards, the locality module, $\mathcal{M}_{\mathfrak{a}}$ is extracted based on the signature of the atom $\mathfrak{a}$ and using function `extractLocalityModule()`. If $\mathcal{M}_{\mathfrak{a}}$ is predicted to be hard and its overlap with the existing hard modules is below the threshold, min_{ov}, then this modules is added to the set of the potential hard modules $\mathbf{M}^H_{\mathcal{O}}$. Algorithm 5 repeats these steps until all the atoms are explored. Unlike the state-of-art ontology hotspot identification methods, we did not restrict the number of extracted modules nor limit their sizes. Indeed, our main goal is to find at least one hard locality module, even if it is a relatively large one. Whenever $\mathbf{M}^H_{\mathcal{O}}$ remains empty at the end of the first loop of this algorithm, then the second graph exploration strategy is applied. Function `extractFromADGraphSubComponents` achieves the steps of this strategy. The connected components of the atomic decomposition graph are first computed. Then, similar steps to those described in Lines 8 to 15 of Algorithm 5 will be iteratively applied to each connected component. The main difference is that the modules will be extracted based on the signature of all the atoms within a connected component. More specifically, function `extractLocalityModule()` (Algorithm 5, Line 8) will take as input a set of atoms rather than a single one.

5.6.4 Evaluations and discussion

The purpose of this section is to provide an empirical evaluation to prove the effectiveness of the proposed hard module extraction algorithm. We start by listing our evaluation methods and the data to be used. Afterwards, the achieved results are illustrated and discussed. We must highlight that we couldn't compare our results to those of the existing methods. On the one hand, the above discussed state-of-art algorithms of hotspot identification are not available for use. On the other hand, our definition of hard ontology module is quite different from the definition of ontology hotspot. We don't share neither the same properties, nor the identification mode. Hence, it would be quite absurd to try to establish any comparison.

5.6.4.1 Evaluation methods

To carry on the empirical evaluation of the ExOH system, the selected 11 distinct OWL ontologies, which varies in size, constructs being used as well as in complexity. Table 5.8 resumes the description of the considered ontologies. Some of these are already sub-parts extracted from the same initial ontology. This is the case of Sweet-P1 and Sweet-P2 which are modules retrieved from the Semantic Web for Earth and Environmental terminology (SWEET). All the test case ontologies have been predicted by our system as empirically hard with respect to the 6 reasoners of our study (Konclude, HermiT, MoRE, TrOWL, JFacT, FacT++). Then, these ontologies were processed by the ExOH system, and thus a set of potential hard modules were identified for each of them.

The validation process of the identified hard modules involves classifying them by the above listed reasoners in order to assess whether our predictions match the real empirical hardness of these modules. To achieve this purpose, we will conduct a new DL/EL classification challenge using the ORE Framework. The potential hard modules extracted from the different ontologies form the ontology corpus to be processed and the above listed reasoners are the challenge counterparts. Using this Framework, we can check both the correctness and the efficiency of these reasoners when classifying the modules. Finally, by referring to the reasoner evaluation results, the empirical hardness of the identified modules could be concretely adjudged. Using this method, we can outline the number of the potentially hard modules which were *really* found to be hard ones. A high number would obviously witness the more effectiveness of our extraction method.

5.6.4.2 Results and discussion

The experimental results of the hard module identification process and then, the validation results of these modules are all listed and discussed in this section. We start our inspections by depicting some of statistics that we have collected while trying to identify the hard modules of the 11 test case ontologies. Table 5.9 sketches some of the characteristics of the computed atomic decomposition (AD) graph of each of the testing ontologies. The number of the atoms (#Atoms) are listed together with some statistics relative to their sizes and degree of connectivity. The table reports also the number of the sink atoms. We recall that the locality modules will be extracted only from the signature of this kind of atoms. The number of the isolated atoms, (degree(atom)=0), is also put in evidence in this table. This is because these atoms will be ignored in the extraction process. The final column of Table 5.9 reports the connectivity of the AD graph (**D**: disconnect, **C**: connected), when the isolated atoms are discarded.

Throughout our experimentations, we observed that the average size of the isolated atoms is of 1,

Ontology	**Expressivity**	**Profile**	**#A**	**#C**	**#P**	**#I**
Chebi http://www.ontobee.org/ontology/CHEBI	$\mathcal{ALER}$	EL	129642	89926	191	28023
Galen (CO-ODE) http://owl.cs.manchester.ac.uk/research/co-ode/	$\mathcal{ALEHIF}+$	DL	37696	23136	950	0
Uberon http://www.ontobee.org/ontology/UBERON	$\mathcal{S}$	DL	32362	11786	5	0
Zebrafish http://www.ontobee.org/ontology/ZFA	$\mathcal{ALERIF}+$	DL	16414	6132	10	0
Digital Camera (DCO) http://db-tom.cs.uwaterloo.ca/	$\mathcal{ALCIF}(D)$	DL	55020	1920	94	280023
Phenotypic quality (PQO) http://www.ontobee.org/ontology/PATO	$\mathcal{SHF}(D)$	DL	172644	81111	32	3717
AEO (Anatomical Entity Ontology)http://www.ontobee.org/ontology/AEO	$\mathcal{SRIQ}$	DL	130460	58193	209	0
NCBI http://www.ontobee.org/ontology/NCBITaxon	$\mathcal{SHI}$	DL	144659	63981	30	0
Sweet-P1 https://sweet.jpl.nasa.gov/download	$\mathcal{SHOIN}(D)$	DL	6444	3022	533	1152
Sweet-P2	$\mathcal{SHOIN}(D)$	DL	6672	2991	533	1152
Gene Ontology (GO) http://www.obofoundry.org/ontology/go.html	$\mathcal{SHIF}$	DL	126796	18672	95	7

Table 5.8: Test ontology description for the empirical validation of ExOH (**A**= Axioms, **C**= Concepts, **P**= Properties, **I**= Individuals)

i.e. contains a unique axiom. Furthermore, their number is relatively high in some ontology cases, like the GO ontology with 83% of isolated atoms. This would entail the sparsity of its AD graph of the GO ontology. While in other ontologies such as Zebrafish, Galen and Uberon , very fewer isolated atoms were found, which would suggest that their AD graphs are dense ones. We can also notice the remarkably high percentage (99%) of the sink atoms in the AD graph of the DCO ontology. Based on this value and the value of average degree of atoms (2), we would assume that this is a *star* graph, where probably all the atoms are connected to one centric atom. Further interesting observations could be retrieved from this table. However, studying the attributes of the AD graphs is not one of our goals. Nevertheless, we will try to correlate some of the relevant observations about the structure of these graphs to the hard module identification results.

Table 5.10 and Table 5.11 respectively summarizes the achieved results of the hard module identification process. The former table illustrates the main characteristics of the modules extracted using our first exploration strategy, otherwise by relying on the sink atoms. This strategy have succeeded to unveil the hard modules for 7 out the 11 examined ontologies. The identification results using our

Ontology	**#Atoms**	**Size**			**Degree**			**Sink**	**Isolated**	**Connectivity**
		Min	**Max**	**Avg**	**Min**	**Max**	**Avg**	**(%)**	**(%)**	
Chebi	70103	1	41	1.85	1	4706	2.86	24.32	46.67	D
Galen	15211	1	13064	2.48		2476	5.33	59.81	0.01	D
Uberon	11351	1	61	2.85	1	974	5.14	58.92	0.03	D
Zebrafish	3073	1	19	5.34	1	3072	6.12	58.48	0	C
DCO	55001	1	15	1	1	14372	2	99.81	0.17	C
PQO	129150	1	100	1.34	1	17921	2.36	36.89	37.28	D
AEO	54740	1	1044	2.38	1	18851	5.52	53.14	2.99	D
NCBI	38531	1	12	3.75	1	29991	4.82	80.76	17.36	D
Sweet-P1	5129	1	13	1.26	1	90	2.26	66.74	1.59	D
Sweet-P2	5368	1	13	1.24	1	92	2.24	68.22	10.26	D
GO	108035	1	7	1.77	1	3917	0.80	7.72	83.25	D

Table 5.9: Result summary of the atomic decomposition.

second strategy, which was applied to the remaining 4 test case ontologies are illustrated in Table 5.11. We start by discussing the results in Table 5.10. We have reported the number of the identified hard modules (#Hard) and their size distribution. Two configurations were considered: with no module overlap control, and then with overlap control. First when no overlap control is applied, we can notice that the number of the potentially hard modules varies from 1 module (DCO ontology) up to 3651 modules (Uberon ontology). These values have known an important decrease when we have restricted the amount of the overlap. We can particularly remark that the number of Galen's potential hard modules have dropped from 2207 to only 1 unique module. Indeed, we have observed that the overlap percentage of the initially identified hard modules is at average about 95% of the module size. This is because Galen have one huge locality module (15564 axioms) which incorporate most of the remaining modules. Furthermore, Galen holds the biggest extracted atom (13064 axioms, Table 5.9), which may explain the emergence of a big module. The DCO ontology is another module centric ontology. Although the very small size of the DCO atoms (1 axiom at average), the unique extracted hard module from this ontology is of size 55020 axioms, which represents 98% of original ontology size. This finding confirms our previous assumption that the AD graph of the DCO ontology has a star shape. All of its atoms are directly related to 1 atom, which later have yielded to the extraction of this huge hard module.

Another bunch of interesting observations could be made from Table 5.11, where the results of our

Ontology	**#Mod**	**Without Overlap control**				**With Overlap Control**					
		#Hard	**Size**			**#Hard**	**Size**			**DL**	**EL**
			Min	**Max**	**Avg**		**Min**	**Max**	**Avg**	(%)	(%)
Galen	8508	**2207**	15336	16162	15408.74	**1**	15564	-	-	100	0
Uberon	6590	**3651**	116	1551	217.91	2909	**116**	1140	914.4	99.89	0.10
DCO	54896	1	**57033**	-	-	1	57033	-	-	100	0
PQO	47638	786	25	1245	22.10	18	25	2346	281.38	0	100
AEO	28699	1822	704	5228	1397.6	1111	704	7323	1627.40	100	0
Sweet-P1	3201	154	6	2293	33.12	61	6	2293	93.75	39.34	60.65
Sweet-P2	1971	260	6	2572	1055.06	164	13	2572	1178.72	79.26	20.72

Table 5.10: Results summary of hard module identification process based on the exploration of the AD sink atoms

Ontology	**#CC**	**#Mod**	**#HardMod**	**Size**			**DL(%)**	**EL(%)**
				Min	**Max**	**Avg**		
Chebi	4	4	1	96895	-	-	0	100
Zebrafish	3	3	2	16410	16412	16411	100	0
NCBI	6	6	1	137739	-	-	0	100
GO	6	6	0	-	-	-	-	-

Table 5.11: Results summary of hard module identification process based on the exploration of the AD connected components

second exploration strategy are illustrated. We can observe that few connected components were composing the AD graphs of the listed ontologies. By consequence, few number of modules were extracted and notably, almost a unique big hard locality module was identified per ontology. Surprisingly, our algorithm did not find any locality hard module in the GO ontology, neither by the first nor by the second search strategy. By referring to Table 5.9, we can observe that the AD graph of the GO ontology has uncommon treats comparing to the other graphs. As previously highlighted, this graph is a very sparse one with 83% isolated atoms and an average degree lower than 1. Yet, we can not provide a sound explanation about the nature of the correlation between these characteristics and the fact that we were enable to identify any hard locality module in this ontology.

Further conclusions could be retrieved from the previous tables. However at this stage of the study,

Ontology	**Expressivity**	**Profile**	**#Hard**	**Corr-Hard**	**#IncorrHard**	
				(%)	#DL	#EL
Chebi	$\mathcal{ALER}$	EL	1	100	0	0
Galen	$\mathcal{ALEHIF}+$	DL	1	100	0	0
Uberon	$\mathcal{S}$	DL	2906	99.96	0	1
Zebrafish	$\mathcal{ALERIF}+$	DL	2	100	0	0
DCO	$\mathcal{ALCIF}(\mathcal{D})$	DL	1	100	0	0
PQO	$\mathcal{SHF}(\mathcal{D})$	DL	18	100	0	0
AEO	$\mathcal{SRIQ}$	DL	1111	100	0	0
NCBI	$\mathcal{SHI}$	DL	1	100	0	0
Sweet-P1	$\mathcal{SHOIN}(\mathcal{D})$	DL	61	54.09	8	20
Sweet-P2	$\mathcal{SHOIN}(\mathcal{D})$	DL	164	17.68	115	20
GO	$\mathcal{SHIF}$	DL	0	0	0	0

Table 5.12: Results of the validation process of the identified hard modules

we need first to check that the identified modules are really hard ones. Table 5.12 depicts the results of the validation process of the potentially hard modules. The four first column of the table recall, for each examined ontology, its expressivity, OWL profile and the total number of the retrieved hard locality modules. Then, the percentage of the correctly identified hard modules is reported in column **Corr-Hard**. The last column of the table outline the distribution of the incorrectly identified modules over the OWL profiles. The percentage of the correctly identified hard modules is computed based on the evaluation results of the 6 different reasoners. We recall that a module is flagged as a hard one if at least one of these reasoners terminates the classification task of this module with a failure status. We can observe that **all** the identified hard modules of 8 out of 10 ontologies were found to be concretely hard ones. Only one hard module from Uberon was not approved. This is a EL module, while the initial ontology is DL. The reduction in the module expressivity may explain this result. On the other hand, the ExOH results were relatively poor regarding the modules extracted from the SWEET based ontologies. Although these are very similar ontologies, we have identified more correctly hard module from Sweet-P1, than from Sweet-P2. This is an interesting case from which we can start further investigations.

In overall, the achieved results are very encouraging ones, witnessing the effectiveness of our initial assumptions and proves the utility of the predictive modelling when applied to the ontology engineering problems of hard module extraction. More analysis of the collected data must be planned, to gain further insights about the possible correlation between the characteristics of the AD graphs and the hardness of the corresponding ontologies. In our opinion, there is some frequent graph patterns that when they appear in the AD graph, the ontology become hard to manage by DL reasoners.

5.7 Conclusion

In this chapter, we described our design ideas of an advising system, which offers guidance to users willing to select a reasoner for ontology based application. We called it, the *"Advisor System over Ontology Reasoners"* (**ADSOR**). It is composed from two main components: **RakSOR**, the ranking system of the ontology reasoners; and **ExOH** the extraction system of the empirically hard ontology modules. We should highlight that the implementation details of the ADSOR system and some screenshots of its corresponding Web application could be found in Annexe B.

We can point out four distinct contributions introduced in this chapter. First, we provided a novel framework which model the problem of automatic ranking of reasoners according to their expected robustness when classifying OWL ontologies under online execution environment. Indeed, both the correctness and the efficiency of a reasoner are considered in the ranking process. Our framework follows a meta-learning approach and applies bucket order rules. To the best of our knowledge, we are the first to consider more than one criteria in the reasoner ranking. Moreover, we succeeded to model and to put into practice two distinct supervised ranking methods which apply our reasoner preference rules: (1) a single label prediction based method; and (2) the extended multi-label ranking (EMLR) method. Our first method showed good ranking quality comparing to the default method. This quality was remarkably improved using our second method. The latter one has also outperformed well recognized state of art multi-label based solutions. However, the gain in the prediction precision and by consequence, in the ranking quality came at a price. Actually, our multi-label based solution provides less detailed information about the expected behaviours of the examined reasoners. It allows to predict the ranking and to distinguish between the successful/unsuccessful reasoners, while our first method further indicates the estimated reasoner runtime and the expected nature of failure. Both methods have approximately the same training and prediction computational complexity. For all of these reasons, it was hard to decide which method to embody in the RakSOR system. Finally, we chose the multi-label ranking (EMLR) method, since it provides more reliable predictions. Of course, we can still enrich the EMLR solution by using the established reasoner robutsness predictive profiles to get more finer grained predictions about the reasoners empirical behaviours. We should also highlight that techniques employed in the EMLR method are generic enough, which would suggest that it could be applied to the ranking problem of other kinds of systems or algorithms. Nevertheless, the feature space we employed to describe the input instances will no more be adequate. Furthermore, the ranking preference rules must also be reviewed and accommodated according to the functional characteristics of the new examined alternatives. Investigating the abilities of the EMLR method in other fields is next on our research agenda.

We also examined the feasibility of employing the ontology hardness predictive model to help extracting the ontology hard modules. We were looking to identify those ontology sub-parts that are the most computationally compelling ones. The solution relies on both the ontology atomic decomposition and locality module extraction techniques. Once the ontology atoms are computed and transformed into modules, the hardness of the latter ones is predicted using our trained model. We obtained primary encouraging results, which would testify the potential benefits of our solution. We also pinpointed some pitfalls in the ontology design structure which are probably causing its hardness. Nevertheless, we have observed that our algorithm was enable to identify the hard modules in some ontology test cases. Hence, we are intending to conduct a larger scale experimentations, in order to ensure whether this is an isolated case or a frequent one. Furthermore, we are planning to conduct some in depth investigations to figure out what distinguishes these ontologies from the ones where our algorithm have known a success. A further issue to be examined is the way to fix the amount of the tailored overlap. We have noticed, in various cases, that the extracted locality modules dramatically overlap with each others. Hence, we tried to restrict this undesirable side effect. As consequence, the number of the extracted hard modules have greatly dropped down. As a matter of fact, it still remains difficult to decide whether it is preferable to maintain a high number of relatively small sized overlapping modules or to wipe out the overlap and keep little number of relatively large modules?

Conclusions

In this concluding chapter, we will summarise the main results of the doctoral project described in this dissertation. First, we will reformulate the research questions and discuss how we have successfully answered them through the different chapters of this dissertation. Hence, we will provide a summary of our main contributions. Then, we will point out a number of future research directions.

Contributions Summary

Over the last decade, several semantic reasoning algorithms and engines have been proposed to cope with the computational complexity of inference tasks on expressive ontology languages such as OWL 2 DL. Huge research efforts have been dedicated to design efficient reasoning algorithms with highly sophisticated optimisation techniques. Nevertheless, annual reasoner competition events have revealed that there is no reasoning engine that can outperform in all input ontologies. Roughly speaking, the performances of a reasoner are closely depending on the success or the failure of its optimizations tricks. Hence, a reasoner could be optimized for some but not all the ontologies. By consequence, a variability phenomena of reasoner empirical performances is often observed in practice. Furthermore, there is a real lack of knowledge and support tools which would help pinpointing those ontology features which are reasoner performances degrading factors. Hence, experts pre-established decision rules about the appropriateness of a reasoner for an ontology based application are not efficient in practice, except for trivial cases. All of these findings gave rise to new challenges in the area of ontology reasoning. Indeed, a user is faced with the problems:

"How to gain insights about the empirical hardness of an ontology? How to choose the appropriate reasoner for a given ontology, in order to achieve better performances than simply selecting an arbitrary one?"

This thesis have dealt with these issues and tried to provide relevant and easy to understand answers to an end user. The proposed solutions rely on empirical algorithmic analysis and employ supervised learning techniques. One of the main advantages of the empirical algorithmic analysis is that it allows to investigate how algorithmic performance statistically depends on problem characteristics. Hence, while trying to track down the performances of reasoners, the ontology key features could be identified. The dependencies were modelled as predictive functions capable to anticipate the performances of a reasoner for any given ontology. The latter ones should be described by a set of relevant features. Another advantage of this kind of analysis is that it allows to compare and to automatically sort different reasoners according to their expected performances over the same ontology. Indeed, future reasoner empirical behaviours could be learned by an intelligent exploration of their past running. Hence, no further experimentations are needed to decide which reasoner to select.

This research has led to general contributions in the field of ontology reasoning. We can point out our contributions to distinct, but complementary, fields as follows:

(i) We proposed a rich set of comprehensive and efficiently computational features to model the design complexity of OWL Ontologies. Our features were arranged into four categories covering a broad range of syntactic, structural and semantic characteristic of the ontology components.

(ii) We proposed to handle to variability of reasoner performances by automatically predicting these performances for any input ontology. We investigated the robustness of reasoners. By robustness, we assign the ability of reasoner to correctly achieve a reasoning task within a fixed time limit. We argued that the robustness should be adjudged under some computational and functional constraints. This is why we chose to examine an online usage scenario where DL ontology classification is the main required task.

(iii) We applied supervised leaning concepts to the problem empirically predicting the reasoner robustness. We proposed a learning methodology which employs our ontology feature suite and relies on a rich set of advanced supervised machine learning techniques. Owing to the benefits of this proposal, we modelled the robustness profile of a reasoner, in terms of predictive models capable to anticipate its correctness and efficiency over any given ontology.

(iv) By employing the same methodology, we learned to identify the empirically hard ontologies. The latter ones are ontologies difficult to manage by a homogeneous group of reasoners. The reasoning methodology is the main criterion to assemble such kind of groups.

(v) The effectiveness of the proposed learning systems has been examined throughout a huge body of experimentations. We investigated 6 well known reasoners and over 2500 ontologies. We put into comparison various supervised learning algorithms and feature selection techniques in order to train best reasoner predictive models. All of our models showed to be highly accurate outperforming state-of-art solutions.

(vi) The trained models were compared and put into examination in order to reveal the most important ontology features. These are the ones found to be imperative to ensure the accuracy of the predictions. As a matter of fact, these features are the most correlated ones to the examined reasoner performances. We believe that such acquired knowledge would help revealing the reasoner performances degrading factors. It would also enhance the improvement of the ontology modelling methods, towards avoiding these factors.

(vii) We designed a novel system to support non-expert users in reasoner selection task. The main goal of this system is to be able to automatically rank a set of candidate reasoners according to their expected robustness over an input ontology. Our ranking framework follows a meta-learning approach and operates within two stages: the offline stage to acquire the knowledge; and the online stage to apply the ranking rules. The ranking is computed according to some preference rules which take into consideration both the correctness and the efficiency of the examined reasoners. The framework produces a consistent ranking with bucket order principals. To the best of our knowledge, we are the first to consider more than one criteria in the ranking process of reasoners.

(viii) We proposed two supervised ranking methods. The first one is based on single label prediction. It takes advantage of the learned reasoner profiles. After predicting the performances of each reasoner over the input ontology, an algorithm ranks them according to our preference rules. The second method is based on multi-label learning techniques. A novel ranking mechanism was introduced, that we called the Extended Multi-Label Ranking (EMLR) method. It combines multi-label classification and multi-target regression techniques. Both the ranking and the relevance of the reasoners are predicted in a consistent way. All of our experimental results witness the effectiveness of our proposals. We achieved high quality ranking and outperformed existing solutions.

(ix) We also examined the feasibility of employing the predictive model of the ontology overall hardness to help identifying its most computational challenging sub-parts. The motivation behind this proposal is to provide a user with some reduced subsets extracted from its empirically hard ontology. These subsets are particularly suspected to be most difficult parts of the ontology with respect to the reasoning tasks. To achieve this purpose, we first defined the concept of ontology *hard syntactic module*. We proposed an extraction and identification algorithm of these modules. Our solution relies on the atomic decomposition of an ontology and employs locality module extraction techniques. Once the ontology atoms are computed and transformed into modules, the hardness of the latter ones is predicted using our trained ML classification model. We obtained first encouraging results, which would testify the potential benefits of our solution.

(x) All of our proposals were gathered and modelled as distinct services offered by one system, called ADSOR, the "Advisor System over Ontology Reasoners". The system was designed to allow the different learning operations within an offline stage and to interact with the user requests in an online stage. We have developed a Web application which puts into practice the design ideas of the ADSOR system. The first version of this application is described in Annexe B.

Future Directions

Throughout the different stages of our doctoral research project, we made a number of choices to ensure the successful completion of our work. However, some of these choices might have limited the ultimate scope of the presented work. Consequently, a number of questions remain, and a range of directions for future research are conceivable. We discuss a number of these possibilities; but we stress that this list is by no means exhaustive.

Alternative reasoner correctness checking method Throughout our study, we highlighted the need to examine the correctness of the reasoning results since reasoners would provide quite distinct results over the same input ontology. Correctness is one important criteria when advising a user about the appropriateness of a reasoner. However, very few works have addressed the issue of automatic checking of the correctness. Researches in this field of study are just in their beginning. In our project, we employed one basic correctness checking method which relies on the majority vote mechanism. Surely, this not a faultless method, hence it will be worthwhile to investigate more advanced methods. We should highlight that shifting to different reasoner correctness checking method would not affect our learning or ranking respective methodologies. Only the reasoner evaluation process and by consequence the collected data will undergo the changes. We believe that using more reliable correctness checking method would grant more significant advices and thus the user satisfaction.

More feature and feature selection methods Our ontology feature suite could be extended by considering the time spent to compute some of the critical features as a further indication of its complexity. Such type of features was found to be relevant when predicting the performances of SAT solvers. We have already mentioned that the feature selection methods, which we have employed in the train of the reasoner robustness models, could further be extended. More specifically, we are intended to investigate more recent feature discretization methods which have found to be more performing than the one we used.

Advancing the reasoner ranking system: We introduced first design ideas of what could be considered as a recommender system of ontology reasoners. Nevertheless, further improvements are to be planned, in particular the following ones:

1. More reasoners to rank first of all, we are planning to examine much more reasoners in order to extend our meta-knowledge base, in particular ELK [KKS11] and Pellet [SPG+07]. We think that our suggested rankings would be further appreciated whenever they cover a more significant number of reasoners and more distinct reasoning methodologies. Therefore, for future work, we are planning to study a different kind of reasoners, those capable to handle ontologies written in fuzzy description logic [BS16]. Yet, to achieve this purpose, we must first extend our ontology feature set by new dedicated features, to better capture the specificity of the fuzzy DLs.
2. More reasoner usage scenarios In our study, we focussed on the online classification scenario of OWL ontologies. We fixed very tight time schedule and provided a relatively large amount of memory. For future work, we are intended to examine further scenarios such as the use of reasoners within mobile devices. Regarding this scenario, memory consumption should be reduced while keeping the same level of efficiency. This is a real challenge for reasoners. Hence, it would be worthwhile to predict their robustness under such constraints. Furthermore, our prediction models focus on the ontology classification task, a TBox reasoning task. As a future direction, we are intending to investigate the applicability of this technique to ABox reasoning tasks such as realisation and conjunctive query answering. Actually, our ultimate goal is to conduct different reasoner evaluation campaigns under a variety of machine, time and memory configurations and for different reasoning tasks. Once these data is gathered and their corresponding predictive models are trained, new ranking criteria could possibly be proposed.
3. Considering the user preferences in the ranking all the preference rules we used to rank the reasoners are default ones, encoded within the system. It would be interesting to ask the user about its own preferences and ranking conditions. Such information could be translated into criteria weights or other forms of preference rules. Thus, it would be favourable to employ multi-criteria decision making approaches, such as the Outranking or MAULT, to rank the reasoners.
4. Analysis of the user feedback in standard recommendation systems, the user feedbacks about the provided recommendations are often requested and analysed to adjudge the quality of the system. In some item recommender systems, these feedbacks are regarded as auxiliary valuable knowledge and are reused to improve the future recommendations. In our context, this could not be valuable, since our advises relies only on the observed behaviours of the ontology reasoners. However, it will be interesting to request the user feedback as an indication about how satisfactory are our advices.

Improving the hard model extraction approach We are planning to conduct larger scale experimentations to be able to validate our results and more importantly to collect more data about how the atomic graphs of the existing ontologies are structured. We believe that we need to grain more insights about these data structure in order to improve our hard modules identification mechanism. We have already discussed other possible improvement directions when concluding Chapter 5. Yet, we believe that the most interesting future prospects is to extend our work, so it could support the identification of ontology *hotspots* with respect to a particular reasoner. Actually, this is no arduous matter. It suffices to replace the predictive model of the ontology hardness by one of the already trained reasoner runtime regression models. In this way, our extraction algorithm will be looking to identify those ontology modules which are the most computationally expensive ones, comparing to the required runtime to classify the original ontology by the examined reasoner.

Even more domain application for the reasoner ranking methods We addressed the problem of reasoner selection from an end user perspective, and computed rankings to provide her/him with some guidelines. Nevertheless, we are convinced that our ranking methods could be employed to achieve different purposes. Two main future domain application could be investigated. These are:

- **Building meta-reasoning system:** a meta-reasoner could be based on the algorithm portfolio approach. In this case, a set of reasoners are gathered into the portfolio together with an intelligent selector capable to decide the most performing reasoner to handle an input ontology. Here, it would be worthwhile to examine the benefits of our ranking method when considered as an intelligent selector. However, to ensure that the meta-reasoner could outperform the existing engines, the time overheard due to the prediction steps should be reduced. Hence, further optimisation steps should be planned. One possible optimisation is to set a default reasoner to be launched whenever the ontology under examination is a trivial case. Chainsaw [TP12] is another kind of meta-reasoner. It extracts an ontology module for each user query and creates a suitable reasoner for this module to be able to answer the query. The authors highlighted that they are still looking for a strategy to choose the best suited reasoner for a given module. For the time being, they are working with a couple of reasoners in some arbitrary way, with no concrete automatic selection. We are willing to integrate our ranking method into this meta-reasoner and to conduct a new large scale study.
- **Intelligent parameter tuning of reasoners:** different reasoning strategies and configuration parameters might exist within one reasoner to handle different categories of ontologies. To best of our knowledge, none of the exiting reasoners employs the supervised modelling to achieve an intelligent selection of these parameters. Indeed, by varying the parameter setting, different version of the same reasoner emerges. The latter ones could be considered as independent reasoners. Thus, a reasoner ranking model could be trained having the different versions as the possible alternatives. Such model would make it possible to select the most performing reasoner variant, in other words,

the most appropriate parameter settings to handle the input ontology in the most efficient way. However, we should be aware that under such circumstances, the rank prediction is a highly time sensitive operation. So, it must be achieved as quick as possible without damaging the overall efficiency of the reasoner.

Bibliography

[Abb12] Sunitha Abburu. A survey on ontology reasoners and comparison. *International Journal of Computer Applications*, 57(17):33–39, 2012. (Cited in pages 2, 3, 16, 49, 70, 71, 95, 98).

[Abb14] Dean Abbott. *Applied Predictive Analytics: Principles and Techniques for the Professional Data Analyst*. John Wiley and Sons Publishing, 1 edition, 2014. (Cited in pages 13, 31, 75, 86, 105, 145, 147).

[AKA91] David W. Aha, Dennis Kibler, and Marc K. Albert. Instance-based learning algorithms. *Mach. Learn.*, 6(1):37–66, 1991. (Cited in pages 77, 79).

[AT16] Wasif Afzal and Richard Torkar. *Computational Intelligence and Quantitative Software Engineering*, chapter Towards Benchmarking Feature Subset Selection Methods for Software Fault Prediction, pages 33–58. Springer, 2016. (Cited in pages 31, 81, 144).

[AYL15a] Nourhène Alaya, Sadok Ben Yahia, and Myriam Lamolle. Predicting the empirical robustness of the ontology reasoners based on machine learning techniques. In *Proceedings of the International Conference on Knowledge Engineering and Ontology Development, (KEOD 2015), part of the 7th International Joint Conference on Knowledge Discovery, Knowledge Engineering and Knowledge Management (IC3K 2015), Volume 2, Lisbon, Portugal, November 12-14, 2015*, pages 61–73, 2015. (Cited in pages 141, 144).

[AYL15b] Nourhène Alaya, Sadok Ben Yahia, and Myriam Lamolle. What makes ontology reasoning so arduous? Unveiling the key ontological features. In *Proceedings of the 5th International Conference on Web Intelligence, Mining and Semantics, Larnaca, Cyprus*, pages 4:1–4:12. ACM, 2015. (Cited in page 116).

[BBFS09] James Bailey, François Bry, Tim Furche, and Sebastian Schaffert. Semantic web query languages. In *Encyclopedia of Database Systems*, pages 2583–2586. 2009. (Cited in page 68).

[BBM11] Franz Baader, Stefan Borgwardt, and Barbara Morawska. Unification in the description logic $\mathcal{EL}$ w.r.t. cycle-restricted TBoxes. Ltcs-report, Institute for Theoretical Computer Science, Technische Universität Dresden, Germany, 2011. (Cited in pages 70, 93, 125).

[BCLS06] Rainer Brüggemann, Lars Carlsen, Dorte B. Lerche, and Peter B. Sørensen. *A Comparison of Partial Order Technique with Three Methods of Multi-Criteria Analysis for Ranking of Chemical

Substance, pages 237–256. Springer Berlin Heidelberg, Berlin, Heidelberg, 2006. (Cited in pages 39, 177).

[BCM$^+$03] Franz Baader, Diego Calvanese, Deborah L. McGuinness, Daniele Nardi, and Peter F. Patel-Schneider, editors. *The Description Logic Handbook: Theory, Implementation, and Applications*. Cambridge University Press, USA, 2003. (Cited in pages 11, 24, 63, 69).

[BCM$^+$10] Franz Baader, Diego Calvanese, Deborah L. McGuinness, Daniele Nardi, and Peter F. Patel-Schneider. *The Description Logic Handbook: Theory, Implementation and Applications*. Cambridge University Press, NY, USA, 2nd edition, 2010. (Cited in pages 1, 11, 21, 48, 62, 65, 70, 113, 115, 122).

[BGCSV08] Pavel Brazdil, Christophe Giraud-Carrier, Carlos Soares, and Ricardo Vilalta. *Metalearning: Applications to Data Mining*. Springer Publishing Company, Incorporated, 1 edition, 2008. (Cited in pages 6, 35, 53, 105, 168).

[BGG$^+$13] Samantha Bail, Birte Glimm, Rafael S. Gonçalves, Ernesto Jiménez-Ruiz, Yevgeny Kazakov, Nicolas Matentzoglu, and Bijan Parsia, editors. *Informal Proceedings of the 2nd International Workshop on OWL Reasoner Evaluation (ORE-2013), Ulm, Germany, July 22, 2013*, 2013. (Cited in page 17).

[BGJ$^+$14] S. Bail, B. Glimm, E. Jiménez-Ruiz, N. Matentzoglu, B. Parsia, and A. Steigmiller. Summary ORE 2014 competition. In *the 3rd Int. Workshop on OWL Reasoner Evaluation (ORE 2014), Vienna, Austria*, 2014. (Cited in pages 3, 17, 18, 49, 95, 96, 97, 98, 130, 149).

[BLHL01] Tim Berners-Lee, James Hendler, and Ora Lassila. The semantic web. *Scientific American*, 284(5):34–43, 2001. (Cited in pages 1, 8, 9, 48, 55, 58, 59).

[BM76] J. A. Bondy and U. S. R. Murty. *Graph Theory with Applications*. Elsevier, New York, 1976. (Cited in page 244).

[Bre01] Leo Breiman. Random forests. *Mach. Learn.*, 45(1):5–32, 2001. (Cited in page 80).

[BS01] Franz Baader and Ulrike Sattler. An overview of tableau algorithms for description logics. *Studia Logica*, 69(1):5–40, 2001. (Cited in pages 2, 11, 24, 28, 34, 48, 69, 72, 73, 99, 121, 123, 125, 126, 151).

[BS16] Fernando Bobillo and Umberto Straccia. The fuzzy ontology reasoner *fuzzyDL*. *Knowledge-Based Systems*, 95:12 – 34, 2016. (Cited in page 229).

[BYRN99] Ricardo A. Baeza-Yates and Berthier Ribeiro-Neto. *Modern Information Retrieval*. Addison-Wesley Longman Publishing Co., Inc., MA, USA, 1999. (Cited in page 185).

[CC97] John C. Caruso and Norman Cliff. Empirical size, coverage, and power of confidence intervals for spearman's rho. *Journal of Educational and Psychological Measurement*, 57:637–654, 1997. (Cited in page 185).

[Cha10] Sadiq Charaniya. Facilitating DL reasoners through ontology partitioning. Master's thesis, The University of Georgia, USA, May 2010. (Cited in page 98).

[CHWW12] Weiwei Cheng, Eyke Hüllermeier, Willem Waegeman, and Volkmar Welker. Label ranking with partial abstention based on thresholded probabilistic models. In *Proceedings of the 26th Annual Conference on Neural Information Processing Systems 2012.*, pages 2510–2518, 2012. (Cited in page 192).

[Coo71] S. A. Cook. The complexity of theorem-proving procedures. In *Proceedings Proceedings of the Annual ACM Symposium on Theory of Computing*, pages 151–158, 1971. (Cited in page 107).

[CS14] Girish Chandrashekar and Ferat Sahin. A survey on feature selection methods. *Comput. Electr. Eng.*, 40(1):16–28, 2014. (Cited in pages 30, 142).

[CV95] Corinna Cortes and Vladimir Vapnik. Support-vector networks. *Machine Learning*, 20(3):273–297, 1995. (Cited in page 80).

[DCtTdK11] Kathrin Dentler, Ronald Cornet, Annette ten Teije, and Nicolette de Keizer. Comparison of reasoners for large ontologies in the OWL 2 EL profile. *Semant. web*, 2(2):71–87, 2011. (Cited in pages 3, 49, 71, 98).

[DGG$^+$15] Michel Dumontier, Birte Glimm, Rafael S. Gonçalves, Matthew Horridge, Ernesto Jiménez-Ruiz, Nicolas Matentzoglu, Bijan Parsia, Giorgos B. Stamou, and Giorgos Stoilos, editors. *Informal Proceedings of the 4th International Workshop on OWL Reasoner Evaluation (ORE-2015), Athens, Greece*, volume 1387, 2015. (Cited in pages 3, 49, 95, 98, 130, 149).

[Don03] Francesco M. Donini. The description logic handbook. chapter Complexity of Reasoning, pages 96–136. Cambridge University Press, NY, USA, 2003. (Cited in pages 21, 113, 121).

[DPSS11] Chiara Del Vescovo, Bijan Parsia, Ulrike Sattler, and Thomas Schneider. The modular structure of an ontology: Atomic decomposition. In *IJCAI 2011, Proceedings of the 22nd International Joint Conference on Artificial Intelligence, Spain*, pages 2232–2237, 2011. (Cited in pages xviii, 41, 173, 210, 211, 212, 214).

[EM02] E. J. Emond and D. Mason. A new rank correlation coefficient with application to consensus ranking problem. *Journal of Multicriteria Decision Analysis*, 11:17–28, 2002. (Cited in page 185).

[FH10] Johannes Fürnkranz and Eyke Hllermeier. *Preference Learning*. Springer-Verlag New York, Inc., 1st edition, 2010. (Cited in pages 108, 176, 189, 190).

[FHLMB08] Johannes Fürnkranz, Eyke Hüllermeier, Eneldo Loza Mencía, and Klaus Brinker. Multilabel classification via calibrated label ranking. *Machine Learning*, 73(2):133–153, 2008. (Cited in pages 190, 193, 204, 207).

[FHT98] Jerome Friedman, Trevor Hastie, and Robert Tibshirani. Additive logistic regression: a statistical view of boosting. *Annals of Statistics*, 28:2000, 1998. (Cited in page 79).

[FI93] Usama M. Fayyad and Keki B. Irani. Multi-interval discretization of continuous-valued attributes for classification learning. In *Proceedings of the International Joint Conference on Uncertainty in AI*, pages 1022–1027, 1993. (Cited in page 82).

[Fil08] P. Filzmoser. Linear and nonlinear methods for regression and classification and applications in R. Technical report, Department of Statistics and Probability Theory, Vienna University of Technology, Vienna, 2008. (Cited in page 79).

[FKM$^+$06] Ronald Fagin, Ravi Kumar, Mohammad Mahdian, D. Sivakumar, and Erik Vee. Comparing partial rankings. *SIAM Journal of Discrete Math.*, 20:628–648, 2006. (Cited in pages 40, 177).

[Fre70] P. R. Freeman. Algorithm as 26: Ranking an array of numbers. *Journal of the Royal Statistical Society. Series C (Applied Statistics)*, 19(1):111–113, 1970. (Cited in page 199).

[Fre78] Linton C. Freeman. Centrality in social networks conceptual clarification. *Social Networks*, 1(3):215 – 239, 1978. (Cited in page 119).

[FW10] Ensan Faezeh and Du Weichang. Canadian semantic web. chapter A Modular Approach to Scalable Ontology Development, pages 79–103. 2010. (Cited in pages 23, 120).

[GBJR$^+$13] Rafael S. Gonçalves, Samantha Bail, Ernesto Jiménez-Ruiz, Nicolas Matentzoglu, Bijan Parsia, Birte Glimm, and Yevgeny Kazakov. OWL reasoner evaluation (ORE) workshop 2013 results: Short report. In *ORE*, pages 1–18, 2013. (Cited in pages 3, 17, 38, 49, 50, 95, 98, 99, 130, 149, 171).

[GCCL06] Aldo Gangemi, Carola Catenacci, Massimiliano Ciaramita, and Jos Lehmann. Modelling ontology evaluation and validation. In *Proceedings of the 3rd European Semantic Web Conference*, 2006. (Cited in pages 21, 23, 113, 118, 119).

[GEJ16] Salvatore Greco, Matthias Ehrgott, and Figueira José Rui. *Multiple Criteria Decision Analysis: State of Art Surveys*. International Series in Operations Research & Management Science. Springer New York Inc., NY, USA, 2016. (Cited in pages 176, 177).

[GHKS08] Bernardo Cuenca Grau, Ian Horrocks, Yevgeny Kazakov, and Ulrike Sattler. Modular reuse of ontologies: Theory and practice. *Journal of Artificial Intelligence Research*, 31(1):273–318, 2008. (Cited in pages 41, 173, 209, 210, 211).

[GHM$^+$12] Birte Glimm, Ian Horrocks, Boris Motik, Rob Shearer, and Giorgos Stoilos. A novel approach to ontology classification. *Web Semantics*, 14:84–101, 2012. (Cited in pages 2, 48, 73, 149).

[GJ90] Michael R. Garey and David S. Johnson. *Computers and Intractability: A Guide to the Theory of NP-Completeness*. W. H. Freeman & Co., NY, USA, 1990. (Cited in pages 91, 104).

[GLPN93] Giorgio Gallo, Giustino Longo, Stefano Pallottino, and Sang Nguyen. Directed hypergraphs and applications. *Discrete Application Mathematics*, 42(2-3):177–201, 1993. (Cited in pages 41, 210).

[GLS$^+$13] S. Garcia, J. Luengo, J.A. Sáez, V. López, and F. Herrera. A survey of discretization techniques: Taxonomy and empirical analysis in supervised learning. *IEEE Transactions on Knowledge and Data Engineering*, 25:734–750, 2013. (Cited in pages 82, 144).

[GMPS13] Rafael S. Gonçalves, Nicolas Matentzoglu, Bijan Parsia, and Uli Sattler. The empirical robustness of description logic classification. In *Informal Proceedings of the 26th International Workshop on Description Logics, Ulm, Germany*, pages 197–208, 2013. (Cited in pages 3, 27, 49, 138).

[GPS12] Rafael S. Gonçalves, Bijan Parsia, and Ulrike Sattler. Performance heterogeneity and approximate reasoning in description logic ontologies. In *Proceedings of the 11th International Conference on The Semantic Web*, pages 82–98, 2012. (Cited in pages 2, 3, 4, 7, 16, 19, 41, 49, 50, 52, 54, 95, 97, 100, 101, 173, 209, 211, 212).

[Gro09] W3C OWL Working Group. *OWL 2 Web Ontology Language: Document Overview*. W3C Recommendation, 27 October 2009. Available at http://www.w3.org/TR/owl2-overview/. (Cited in pages 2, 49, 65, 93).

[Gru93] T. R. Gruber. A translation approach to portable ontology specifications. *Knowledge Acquisition archive, Academic Press Ltd. London, UK*, Volume 5,:199–220, 1993. (Cited in pages 10, 61).

[GS01] C. P Gomes and B. Selman. Algorithm portfolios. *Artificial Intelligence*, 126:43–62, 2001. (Cited in pages 6, 53, 107).

[GSCK00] Carla P. Gomes, Bart Selman, Nuno Crato, and Henry Kautz. Heavy-tailed phenomena in satisfiability and constraint satisfaction problems. *J. Autom. Reason.*, 24(1-2):67–100, February 2000. (Cited in page 107).

[GTH06] Tom Gardiner, Dmitry Tsarkov, and Ian Horrocks. Framework for an automated comparison of description logic reasoners. In *Proceedings of the International Semantic Web Conference, USA*, pages 654–667, 2006. (Cited in pages 3, 17, 19, 49, 50, 96, 97, 99, 141).

[HB91] B. Hollunder and F. Baader. Qualifying number restrictions in concept languages. In *Proceedings of the Second International Conference on Principles of Knowledge Representation and Reasoning, KR-91*, pages 335–346, Boston (USA), 1991. (Cited in pages 121, 127).

[HB11] Matthew Horridge and Sean Bechhofer. The OWL API: A java API for OWL ontologies. *Semantic Web*, 2(1):11–21, January 2011. (Cited in page 67).

[HD14] Bao-Gang Hu and Weiming Dong. A study on cost behaviors of binary classification measures in class-imbalanced problems. *CoRR*, abs/1403.7100, 2014. (Cited in pages 84, 146, 155).

[HFH$^+$09] Mark Hall, Eibe Frank, Geoffrey Holmes, Bernhard Pfahringer, and Peter Reutemann. The weka data mining software: An update. *SIGKDD Explor. Newsl.*, 11:10–18, 2009. (Cited in pages 31, 87, 88, 144).

[HKS06] Ian Horrocks, Oliver Kutz, and Ulrike Sattler. The even more irresistible $\mathcal{SROIQ}$. In *Proceedings of the 23rd Benelux Conference on Artificial Intelligence*, pages 57–67, 2006. (Cited in pages 2, 49, 65, 70, 93, 117).

[Hol92] Walter Holberg. The decomposition of graphs into k-connected components. *Discrete Mathematics*, 109(1):133 – 145, 1992. (Cited in page 215).

[Hor03] I. Horrocks. Implementation and optimisation techniques. In *The Description Logic Handbook: Theory, Implementation, and Applications*, chapter 9, pages 306–346. Cambridge University Press, 2003. (Cited in pages 2, 34, 49, 99, 121, 125, 126, 127).

[Hor07] Ian Horrocks. Semantic web: The story so far. In *Proceedings of the 2007 International Cross-disciplinary Conference on Web Accessibility (W4A)*, pages 120–125, 2007. (Cited in page 59).

[HPSV03] I. Horrocks, P.F. Patel-Schneider, and F. Van Harmelen. From SHIQ and RDF to OWL: The making of a web ontology language. *Web Semantics: Science, Services and Agents on the World Wide Web*, 1(1):7 – 26, 2003. (Cited in pages 1, 11, 24, 48, 62, 64).

[HTF01] Trevor Hastie, Robert Tibshirani, and Jerome Friedman. *The Elements of Statistical Learning*. Springer Series in Statistics. Springer New York Inc., NY, USA, 2001. (Cited in pages 13, 74).

[HXHLB14] Frank Hutter, Lin Xu, Holger H. Hoos, and Kevin Leyton-Brown. Algorithm runtime prediction: Methods & evaluation. *Artificial Intelligence*, 206:79–111, 2014. (Cited in pages 5, 52, 104).

[ISTV10] Marios Ioannou, George Sakkas, Grigorios Tsoumakas, and Ioannis P. Vlahavas. Obtaining bipartitions from score vectors for multi-label classification. In *Proceedings of the 22nd International Conference on Tools with Artificial Intelligence, ICTAI*, pages 409–416. IEEE Computer Society, 2010. (Cited in pages 190, 201).

[KKL15] Yong-Bin Kang, Shonali Krishnaswamy, and Yuan-Fang Li. R_2O_2: An efficient ranking-based reasoner for OWL ontologies. In *Proceedings of the 14th International Semantic Web Conference, ISWC*, pages 322–338, 2015. (Cited in pages 6, 22, 53, 101, 102, 103, 111, 159, 186, 187, 192, 203, 204).

[KKS11] Yevgeny Kazakov, Markus Krötzsch, and František Simancík. Concurrent classification of el ontologies. In *Proceedings of the 10th International Conference on The Semantic Web - ISWC'11*, pages 305–320, 2011. (Cited in pages 73, 149, 229).

[KLK12] Yong-Bin Kang, Yuan-Fang Li, and Shonali Krishnaswamy. Predicting reasoning performance using ontology metrics. In *Proceedings of the 11th International Conference on The Semantic Web*, pages 198–214, 2012. (Cited in pages 5, 24, 52, 100, 102, 103, 122, 144, 146, 163).

[KLK14] Yong-Bin Kang, Yuan-Fang Li, and Shonali Krishnaswamy. How long will it take? Accurate prediction of ontology reasoning performance. In *Proceedings of the 28th AAAI Conference on Artificial Intelligence*, pages 80–86, 2014. (Cited in pages 5, 7, 24, 41, 52, 54, 100, 101, 102, 103, 122, 209, 212, 215).

[Knu74] Donald E. Knuth. Estimating the efficiency of backtrack programs. Technical report, Stanford, CA, USA, 1974. (Cited in page 135).

[Kot07] S.B. Kotsiantis. Supervised machine learning: A review of classification techniques. In *Proceedings of the Emerging Artificial Intelligence Applications in Computer Engineering Conference.*, pages 3–24, The Netherlands, 2007. IOS Press. (Cited in pages 5, 13, 14, 52, 76, 77, 100, 140, 145).

[KP98] Ron Kohavi and Foster Provost. Glossary of terms. *Machine Learning*, 30(2-3):271–274, 1998. (Cited in page 83).

[Krö12] Markus Krötzsch. OWL 2 profiles: An introduction to lightweight ontology languages. In *Reasoning Web. Semantic Technologies for Advanced Query Answering - 8th International Summer School 2012, Vienna, Austria*, pages 112–183, 2012. (Cited in pages 65, 94).

[KZP06] S. B. Kotsiantis, I.D. Zaharakis, and P. E. Pintelas. Machine learning: A review of classification and combining techniques. *J. Artificial Intelligence Review*, 26:159–190, 2006. (Cited in page 139).

[LBG15] Christiane Lemke, Marcin Budka, and Bogdan Gabrys. Metalearning: A survey of trends and technologies. *Artificial Intelligence Review*, 44(1):117–130, June 2015. (Cited in pages 105, 106).

[LBHHX14] Kevin Leyton-Brown, Holger H. Hoos, Frank Hutter, and Lin Xu. Understanding the empirical hardness of NP-complete problems. *Communications of the ACM*, 57(5):98–107, May 2014. (Cited in pages 5, 52, 104, 107).

[LMPS15] Michael Lee, Nicolas Matentzoglu, Bijan Parsia, and Uli Sattler. A multi-reasoner, justification-based approach to reasoner correctness. In *Proceedings of the 14th International Semantic Web Conference, ISWC, Bethlehem, USA*, pages 393–408, 2015. (Cited in pages 3, 17, 19, 49, 50, 96, 97, 99).

[LNJ+10] P. LePendu, N. Noy, C. Jonquet, P. Alexander, N. Shah, and M. Musen. Optimize first, buy later: Analyzing metrics to ramp-up very large knowledge bases. In *Proceedings of The International Semantic Web Conference*, pages 486–501, 2010. (Cited in pages 21, 23, 113, 115, 118).

[LS08] Harris Lin and Evren Sirin. Pellint - a performance lint tool for pellet. In *Proceedings of the OWL Experiences and Directions Workshop at ISWC'08*, volume 432, Germany, 2008. (Cited in pages 98, 126).

[LSN10] Thorsten Liebig, Andreas Steigmiller, and Olaf Noppens. Scalability via parallelization of OWL reasoning. In *Proceedings of the 4th International Workshop on New Forms of Reasoning for the Semantic Web: Scalable and Dynamic (NeFoRS 2010)*, 2010. (Cited in page 28).

[MBP13] Nicolas Matentzoglu, Samantha Bail, and Bijan Parsia. A corpus of OWL DL ontologies. In *Informal Proceedings of the 26th International Workshop on Description Logics, Ulm, Germany*, pages 829–841, 2013. (Cited in pages 25, 130, 149).

[McB02] B. McBride. Jena: a semantic web toolkit. *IEEE Internet Computing*, 6(6):55–59, Nov 2002. (Cited in page 67).

[McG12] Catherine C. McGeoch. *A Guide to Experimental Algorithmics*. Cambridge University Press, NY, USA, 1st edition, 2012. (Cited in page 104).

[Mik09] Vàclav Mikolàšek. Dependability and robustness: State of the art and challenges. In *Software Technologies for Future Dependable Distributed Systems*, pages 25–31, 2009. (Cited in pages 27, 138).

[MLH+15] Nicolas Matentzoglu, Jared Leo, Valentino Hudhra, Uli Sattler, and Bijan Parsia. A survey of current, stand-alone OWL reasoners. In *Proceedings of the 4th International Workshop on OWL Reasoner Evaluation*, pages 68–79, 2015. (Cited in pages 2, 12, 48, 71, 72).

[MO97] Richard Maclin and David Opitz. An empirical evaluation of bagging and boosting. In *Proceedings of the Fourteenth National Conference on Artificial Intelligence and Ninth Conference on Innovative Applications of Artificial Intelligence*, pages 546–551, 1997. (Cited in page 80).

[MP88] Warren S. McCulloch and Walter Pitts. Neurocomputing: Foundations of research. chapter A Logical Calculus of the Ideas Immanent in Nervous Activity, pages 15–27. MIT Press, Cambridge, MA, USA, 1988. (Cited in page 78).

[MRW11] Francisco Martin-Recuerda and Dirk Walther. Towards understanding reasoning complexity in practice. In *Proceedings of Third International Conference of Knowledge Representation and Reasoning*, UK, 2011. (Cited in pages 2, 16, 49, 95).

[MRW14] Francisco Martín-Recuerda and Dirk Walther. Fast modularisation and atomic decomposition of ontologies using axiom dependency hypergraphs. In *Proceedings of the 13th International Semantic Web Conference - Part II*, pages 49–64. Springer, New York, 2014. (Cited in pages 41, 210, 211, 215).

[MSH09] Boris Motik, Rob Shearer, and Ian Horrocks. Hypertableau reasoning for description logics. *Journal of Artificial Intelligence Research*, 36:165–228, 2009. (Cited in pages 2, 11, 24, 48, 69, 72, 121).

[OAC+07] Leo Obrst, Benjamin Ashpole, Werner Ceusters, Inderjeet Mani, and Barry Smith. Semantic web. chapter The evaluation of ontologies - Toward Improved Semantic Interoperability, pages 139–158. Springer US, 2007. (Cited in page 117).

[OHH+08] Eoin O'Mahony, Emmanuel Hebrard, Alan Holland, Conor Nugent, and Barry O'Sullivan. Using case-based reasoning in an algorithm portfolio for constraint solving. In *19th Irish Conference on AI*, 2008. (Cited in page 107).

[OHL15] Richard Jayadi Oentaryo, Stephanus Daniel Handoko, and Hoong Chuin Lau. Algorithm selection via ranking. In *Proceedings of Twenty-Ninth AAAI Conference on Artificial Intelligence*, pages 1826–1832, 2015. (Cited in pages 107, 176, 192).

[ORGV15] Juan Luis Olmo, Cristóbal Romero, Eva Gibaja, and Sebastián Ventura. Improving meta-learning for algorithm selection by using multi-label classification: A case of study with educational data sets. *International Journal of Computational Intelligence Systems*, 8(6):1144–1164, 2015. (Cited in page 191).

[Pap03] Christos H. Papadimitriou. Computational complexity. In *Encyclopedia of Computer Science*, pages 260–265. John Wiley and Sons Ltd., UK, 2003. (Cited in page 91).

[PdSL11] Ricardo Bastos Cavalcante Prudêncio, Marcilio C. P. de Souto, and Teresa Bernarda Ludermir. Selecting machine learning algorithms using the ranking meta-learning approach. In Norbert

Jankowski, Włodzisław Duch, and Krzysztof Grabczewski, editors, *Meta-Learning in Computational Intelligence*, pages 225–243. Springer Berlin Heidelberg, 2011. (Cited in pages 7, 54, 175).

[PI11] Daniel Pop and Gabriel Iuhasz. Overview of machine learning tools and libraries. Technical report, Institute e-Austria Timisoara, Austria, 2011. (Cited in page 87).

[PKK11] Vinayaka Pandit, Sreyash Kenkre, and Arindam Khan. On discovering bucket orders from preference data. In *Proceedings of the Eleventh SIAM International Conference on Data Mining*, pages 872–883, Arizona, USA, 2011. (Cited in page 178).

[Pla99] John C. Platt. Advances in kernel methods. chapter Fast Training of Support Vector Machines Using Sequential Minimal Optimization, pages 185–208. MIT Press, MA, USA, 1999. (Cited in pages 77, 80).

[Pri02] Hilary A. Priestley. Ordered sets and complete lattices. In Roland Backhouse, Roy Crole, and Jeremy Gibbons, editors, *Algebraic and Coalgebraic Methods in the Mathematics of Program Construction: International Summer School and Workshop Oxford, UK*, pages 21–78. Springer Berlin Heidelberg, 2002. (Cited in page 175).

[Qui93] J. Ross Quinlan. *C4.5: Programs for Machine Learning.* Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 1993. (Cited in page 77).

[RGH12] Ana Armas Romero, Bernardo Cuenca Grau, and Ian Horrocks. MORe: Modular combination of OWL reasoners for ontology classification. In *Proceedings of the 11th International Conference on The Semantic Web*, 2012. (Cited in pages 74, 149).

[RHW86] D. E. Rumelhart, G. E. Hinton, and R. J. Williams. Parallel distributed processing: Explorations in the microstructure of cognition, vol. 1. chapter Learning Internal Representations by Error Propagation, pages 318–362. MIT Press, Cambridge, MA, USA, 1986. (Cited in pages 77, 78).

[Ric76] John R. Rice. The algorithm selection problem. *Advances in Computers*, 15:65–118, 1976. (Cited in pages 6, 53, 104, 105, 174).

[Ris01] Irina Rish. An empirical study of the naive bayes classifier. In *Proceedings of IJCAI 2001 workshop on empirical methods in artificial intelligence*, pages 41–46, NY, USA, 2001. (Cited in page 77).

[Ros91] Kenneth H Rosen. *Discrete Mathematics and Its Applications.* McGraw Hill Higher Education, NY, USA, 1991. (Cited in page 180).

[RZ13] Catherine Roussey and Ondrej Zamazal. Antipattern detection: how to debug an ontology without a reasoner. In *Proceedings of the Second International Workshop on Debugging Ontologies and Ontology Mappings, Montpellier, France*, pages 45–56, 2013. (Cited in pages 99, 124).

[SAC07] Mark D. Smucker, James Allan, and Ben Carterette. A comparison of statistical significance tests for information retrieval evaluation. In *Proceedings of the Sixteenth ACM Conference on Conference on Information and Knowledge Management*, pages 623–632, 2007. (Cited in page 187).

[Sam59] A. L. Samuel. Some studies in machine learning using the game of checkers. *IBM J. Res. Dev.*, 3(3):210–229, 1959. (Cited in pages 8, 9, 55, 58, 74).

[SKH11] Frantisek Simancik, Yevgeny Kazakov, and Ian Horrocks. Consequence based reasoning beyond horn ontologies. In *Proceedings of the 22nd International Joint Conference on Artificial Intelligence*, 2011. (Cited in pages 2, 11, 28, 48, 69, 72).

[SLG14] Andreas Steigmiller, Thorsten Liebig, and Birte Glimm. Konclude: System description. *Web Semantics: Science, Services and Agents on the World Wide Web*, 27(1), 2014. (Cited in pages 74, 149).

[SM07] H. Samulowitz and R. Memisevic. Learning to solve qbf. In *Proceedings of the AAAI National Conference on Artificial Intelligence*, pages 255–260, 2007. (Cited in page 107).

[SM09] Kate A. Smith-Miles. Cross-disciplinary perspectives on meta-learning for algorithm selection. *ACM Computing Surveys*, 41(1):6:1–6:25, 2009. (Cited in pages 38, 171, 172).

[SP13] Quan Sun and Bernhard Pfahringer. Pairwise meta-rules for better meta-learning-based algorithm ranking. *Machine Learning*, 93(1):141–161, 2013. (Cited in pages 106, 192, 204).

[SPG$^+$07] Evren Sirin, Bijan Parsia, Bernardo Cuenca Grau, Aditya Kalyanpur, and Yarden Katz. Pellet: A practical OWL-DL reasoner. *Web Semantic*, 5:51–53, 2007. (Cited in pages 2, 28, 48, 73, 99, 149, 229).

[SPS09] Heiner Stuckenschmidt, Christine Parent, and Stefano Spaccapietra. *Modular Ontologies: Concepts, Theories and Techniques for Knowledge Modularization.* Springer Publishing Company, Incorporated, 1st edition, 2009. (Cited in page 209).

[SSB14] Viachaslau Sazonau, Uli Sattler, and Gavin Brown. Predicting performance of OWL reasoners: Locally or globally? In *Proceedings of the Fourteenth International Conference on Principles of Knowledge Representation and Reasoning*, 2014. (Cited in pages 5, 52, 100, 101, 102, 116, 130, 144, 146, 163).

[STGV16] Eleftherios Spyromitros-Xioufis, Grigorios Tsoumakas, William Groves, and Ioannis Vlahavas. Multi-target regression via input space expansion: treating targets as inputs. *Machine Learning*, 104(1):55–98, 2016. (Cited in pages 191, 200, 208).

[Sun14] Quan Sun. *Meta-Learning and the Full Model Selection Problem.* PhD thesis, University of Waikato, Hamilton, New Zealand, 2014. (Cited in page 176).

[TAM$^+$05] Samir Tartir, I. Budak Arpinar, Michael Moore, Amit P. Sheth, and Boanerges Aleman-Meza. OntoQA: Metric-based ontology quality analysis. In *Proceedings of IEEE Workshop on Knowledge Acquisition from Distributed, Autonomous, Semantically Heterogeneous Data and Knowledge Sources*, 2005. (Cited in pages 21, 23, 24, 113, 118, 119, 121).

[TH06] Dmitry Tsarkov and Ian Horrocks. Fact++ description logic reasoner: System description. In *Proceedings of the Third International Joint Conference on Automated Reasoning*, pages 292–297, 2006. (Cited in pages 2, 24, 48, 73, 149).

[THPS07] Dmitry Tsarkov, Ian Horrocks, and Peter F. Patel-Schneider. Optimizing terminological reasoning for expressive description logics. *J. of Automated Reasoning*, 39(3):277–316, 2007. (Cited in pages 2, 21, 49, 113, 121, 126, 127, 128).

[TKV10] Grigorios Tsoumakas, Ioannis Katakis, and Ioannis P. Vlahavas. Mining multi-label data. In *Data Mining and Knowledge Discovery Handbook, 2nd ed.*, pages 667–685. 2010. (Cited in pages 189, 190, 191).

[TP12] Dmitry Tsarkov and Ignazio Palmisano. Chainsaw: a metareasoner for large ontologies. In *Proceedings of the 1st International Workshop on OWL Reasoner Evaluation, (ORE-2012), Manchester, UK*, 2012. (Cited in pages 47, 230).

[TPR10] Edward Thomas, Jeff Z. Pan, and Yuan Ren. Trowl: Tractable OWL 2 reasoning infrastructure. In *The Semantic Web: Research and Applications, 7th Extended Semantic Web Conference, Heraklion, Crete, Greece*, pages 431–435, 2010. (Cited in pages 73, 149).

[TSXVV11] Grigorios Tsoumakas, Eleftherios Spyromitros-Xioufis, Jozef Vilcek, and Ioannis Vlahavas. Mulan: A java library for multi-label learning. *Journal of Machine Learning Research*, 12:2411–2414, 2011. (Cited in page 202).

[Van11] Joaquin Vanschoren. Meta-learning architectures: Collecting, organizing and exploiting meta-knowledge. In *Meta-Learning in Computational Intelligence*, pages 117–155. 2011. (Cited in pages 38, 39, 171, 172).

[VLP08] Frank Van Harmelen, Vladimir Lifschitz, and Bruce Porter. *Handbook of Knowledge Representation (Foundations of Artificial Intelligence)*. Elsevier B.V., 1 edition,, 2008. (Cited in pages 59, 66).

[W3C] Semantic web - w3c. `http://www.w3.org/standards/semanticweb/`. 2015. (Cited in pages 59, 60).

[WLL$^+$07] Timo Weithöner, Thorsten Liebig, Marko Luther, Sebastian Böhm, Friedrich Henke, and Olaf Noppens. Real-world reasoning with OWL. In *Proceedings of the 4th European Conference on The Semantic Web*, pages 296–310, 2007. (Cited in pages 2, 3, 16, 49, 95).

[WM97] D. H. Wolpert and W. G. Macready. No free lunch theorems for optimization. *Trans. Evol. Comp*, 1(1):67–82, April 1997. (Cited in page 103).

[WP07] Taowei David Wang and Bijan Parsia. Ontology performance profiling and model examination: First steps. In *Proceedings of The 6th International Semantic Web Conference, 2nd Asian Semantic Web Conference, ISWC + ASWC*, pages 595–608, Korea, 2007. (Cited in pages 3, 49, 51, 98, 126).

[XHHLB08] Lin Xu, Frank Hutter, Holger H. Hoos, and Kevin Leyton-Brown. Satzilla: Portfolio-based algorithm selection for sat. *J. Artif. Int. Res.*, 32:565–606, 2008. (Cited in pages 107, 158, 159, 175).

[ZLT10] Hongyu Zhang, Yuan-Fang Li, and Hee Beng Kuan Tan. Measuring design complexity of semantic web ontologies. *J. Syst. Softw.*, 83(5):803–814, 2010. (Cited in pages 21, 101, 113, 118, 119).

[ZZ14] Min-Ling Zhang and Zhi-Hua Zhou. A review on multi-label learning algorithms. *IEEE Trans. Knowl. Data Eng.*, 26(8):1819–1837, 2014. (Cited in pages 189, 190, 191, 193, 208).

APPENDIX A

Graph Theory Terminology

Notations and the basic terminology about graph theory used in this thesis are retrieved from [BM76].

Graph basic properties. A **graph** $G = (V, E)$ is a non-empty set of vertices $V(G)$ (also denoted by V) and a set of edges $E(G)$ (also denoted by E). $E(G)$ is a two element subset of $V(G)$, i.e. $E(G) = \{(u, v)|u, v \in V(G)\}$. In other words, each edge $e \in E$ is said to *join* two vertices, (u,v) from V, which are called its **endpoints**. In this case, u and v are *adjacent* vertices and e is an incident edge to u and v. When all the edges in the graph are *unordered* pairs of vertices, then the graph is said to be **undirected**; otherwise, it is called a **directed** graph or a **digraph**. If multiple edges are allowed between two vertices, we obtain a **multigraph**. A **self-loop** or **loop** is an edge between a vertex and itself. An undirected graph without loops or multiple edges is known as a **simple** graph. A (partial) **subgraph** of graph G is a graph H with $V(H) \subseteq V(G)$ and $E(H) \subseteq E(G)$.

Degree of vertices. The degree of a vertex v, denoted by $deg(v)$, is equal to the number of edges incident to this vertex, with loops counted twice. A vertex v with degree 0 is called **isolated** vertex, however a vertex with degree 1 is called **endvertex** or a *leaf*. The **degree of a graph**, denoted by $\Delta(G)$, corresponds to the maximal computed degree of the graph vertices set: $\Delta(G) = max_{v \in V} deg(v)$.

In digraphs, if $e = (u, v)$ is a directed edge, then u is called the **tail** of e while v is said to be the **head**. Accordingly, the **indegree** of vertex v, denoted by $degin(v)$, is the number of the head endpoints adjacent to v. Analogously, the **outdegree** of vertex v, denoted by $degout(v)$, is the number of tail endpoints adjacent from v. Thus, the degree of v in a digraph is equal to the sum of its indegree and outdegree: $deg(v) = degin(v) + degout(v)$. A vertex v, for which the $degin(v) = 0$ and the $degout(v) > 0$ is called a **source**, whereas a **sink** represents a vertex having a $degin(v) > 0$ and a $degout(v) = 0$.

Paths and cycles. A **path** in graph (resp. digraph or multigraph) is a sequence of edges which connects a sequence of vertices. More explicitly, let $(v_1, v_2, \ldots, v_k)$ be a sequence of vertices such that $v_i \neq v_j$ whenever $i \neq j$, a sequence of edges $(e_1, e_2, \ldots, e_k)$ is a path of length k if $\forall i \in \{1, .., (k-1)\}$, $e_i = (v_i, v_{i+1})$ is an edge. In case where $v_1 = v_k$, then the path is said to be **closed**. A **cycle** is a closed path for which the edges and the vertices are distinct ones except for the starting and the ending vertices. A graph without cycles is called acyclic.

Graph Connectivity. A graph is **connected** if any two of its vertices can be joined by a path. A graph that is not connected is said to be **disconnected**. The resulting *"pieces"* of a disconnected graph are called the **components** of the graph. More specifically, a **connected component** is a maximal connected subgraph of a graph G. Each vertex belongs to exactly one connected component, as does each edge.

A digraph is called **weakly connected** if replacing all of its directed edges with undirected edges produces a connected (undirected) graph. It is connected if it contains a directed path from u to v or a directed path from v to u for every pair of vertices u, v. On the other hand, a digraph is **strongly connected** or strong if it contains a directed path from u to v and a directed path from v to u for every pair of vertices u, v. The **strong components** are the maximal strongly connected subgraphs.

A connected and acyclic graph is called a **tree**. Whereas, when all the pairs vertices in a graph are adjacent to each other, then this graph is said to be **complete**. Let K_n be a complete graph with n vertices. In this case, each vertex v in V_{K_n} has $deg(v) = (n-1)$ and the number of edges of this kind of graphs is equal to $|E_{K_n}| = \frac{n(n-1)}{2}$.

If erasing an edge of a connected graph would cause the graph to be disconnected, that edge is called a **bridge**. Similarly, a **cut** vertex in a graph is a vertex that when removed (with its incident edges) creates more components than previously in the graph. A graph is **biconnected** if it is connected and contains at least 3 vertices, but no cut vertex. A graph is **2-edge-connected** if it is connected and contains at least 2 vertices, but no bridge.

APPENDIX B

Implementation Details of the ADSOR System

We implemented the proposed **ADSOR** system as a Web based application to offer an online access to our services: the automatic ranking of reasoners and the extraction of ontology hard modules. ADSOR was implemented in Java and according to the well known J2EE architecture. Advanced Web technologies were put forward to achieve this implementation. All of the computing operations are coded in the background components, which have access to our meta-knowledge base, while the front-end components ensure the display of the Web pages.

The user needs to create an account to be able to login to the ADSOR system as shown in Figure B. All of the user past operations and obtained results are already saved in the system data repository. The user can access them from its personal space as depicted in Figure B. Using this interface, the user can also submit its new request after loading its ontology and choosing the operation to achieve: ranking or extraction. The Web interface which lists the ranking results is illustrated in Figure B. ADSOR outputs the computed feature values used to evaluate the user ontology. Then, the ranked reasoners are listed together with some indications such as the expected termination state (Success/Failure) and the estimated runtime in case of success. To ease the access to the suggested reasoners, the official download links of the latter ones also provided. In case of extraction request, the response does not come immediately. The hard module identification algorithm is executed offline, since the the calculation procedure is a time consuming one. Nevertheless, the user is instantly notified whether its ontology is hard or not. If this is the case, then it is informed that he will receive an email as soon as the extracted modules are ready for download. The identified set of the hard modules are accessible through the user personal space. The Web interface which lists the ranking results is depicted in Figure B. The computed features of the examined ontology are illustrated in this web page, followed by the list of the extracted modules. Some of basic characteristics of these modules are also outlined in this page.

We should highlight that this is an under development version of the ADSOR system. Some functionalities are still not operational. As soon as the implementation and the bug fixing tasks are achieved, ADSOR will be put online to be publicly used.

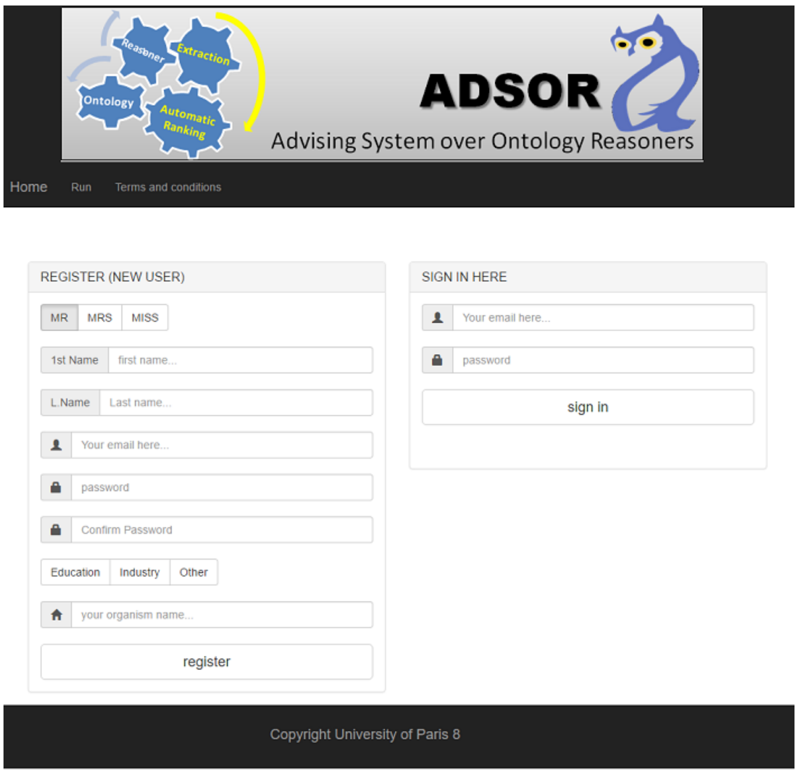
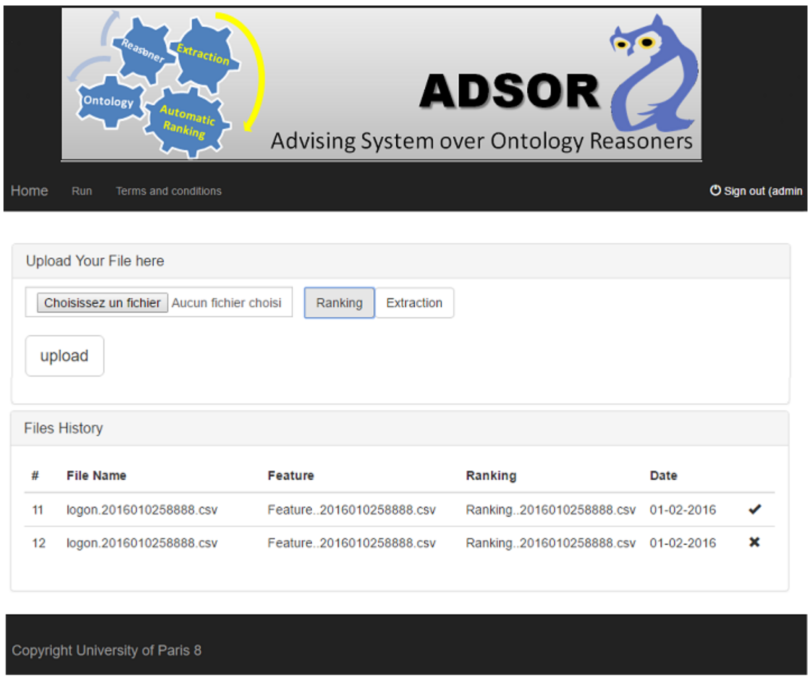
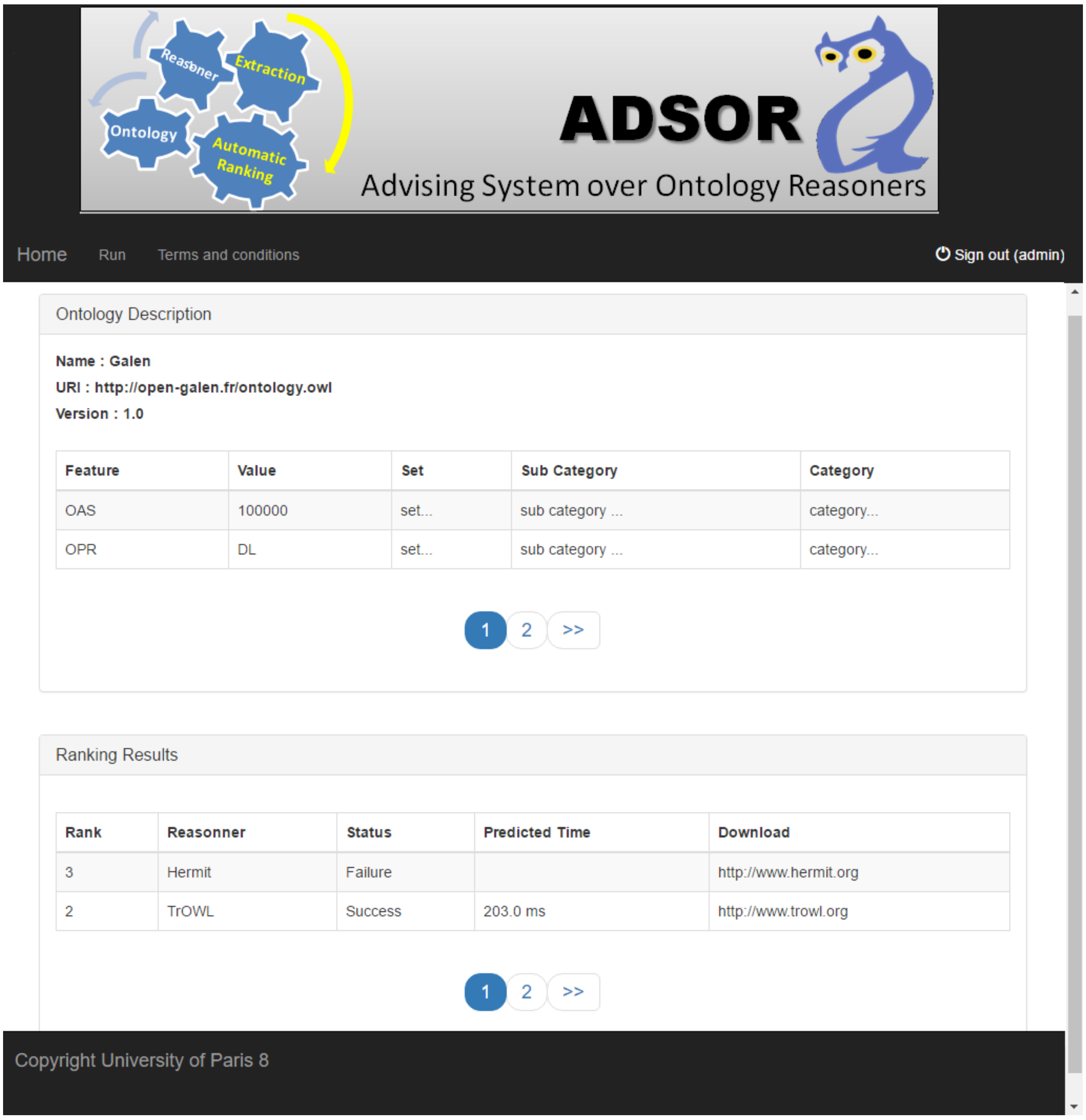


Figure B.1: The login interface of the ADSOR system.



#	File Name	Feature	Ranking	Date	
11	logon.2016010258888.csv	Feature..2016010258888.csv	Ranking..2016010258888.csv	01-02-2016	✓
12	logon.2016010258888.csv	Feature..2016010258888.csv	Ranking..2016010258888.csv	01-02-2016	✖

Figure B.2: The main running interface of the ADSOR system.



Ontology Description

Name : Galen
URI : http://open-galen.fr/ontology.owl
Version : 1.0

Feature	Value	Set	Sub Category	Category
OAS	100000	set...	sub category ...	category...
OPR	DL	set...	sub category ...	category...

1 2 >>

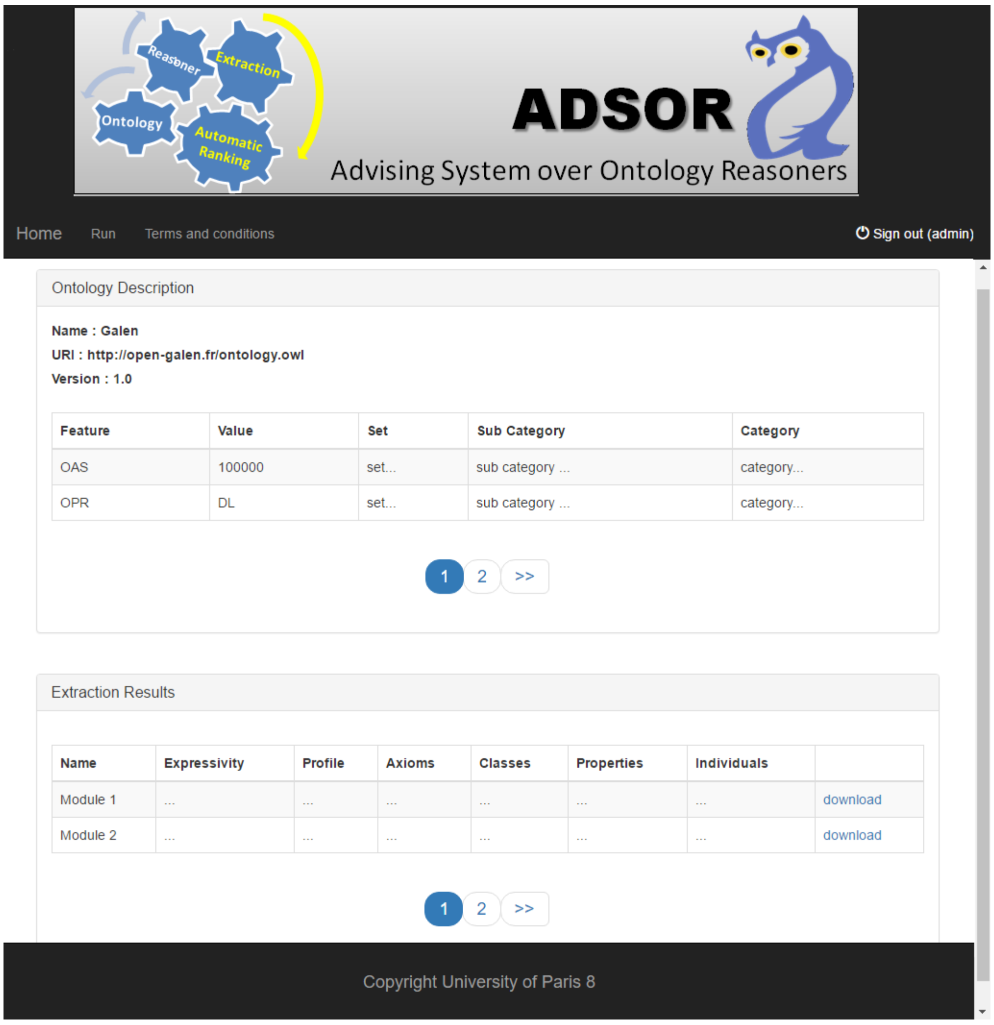
Ranking Results

Rank	Reasonner	Status	Predicted Time	Download
3	Hermit	Failure		http://www.hermit.org
2	TrOWL	Success	203.0 ms	http://www.trowl.org

1 2 >>

Copyright University of Paris 8

Figure B.3: ADSOR web interface of the reasoner ranking results.



Ontology Description

Name : Galen
URI : http://open-galen.fr/ontology.owl
Version : 1.0

Feature	Value	Set	Sub Category	Category
OAS	100000	set...	sub category ...	category...
OPR	DL	set...	sub category ...	category...

1 2 >>

Extraction Results

Name	Expressivity	Profile	Axioms	Classes	Properties	Individuals	
Module 1	...	...	...	...	...	...	download
Module 2	...	...	...	...	...	...	download

1 2 >>

Copyright University of Paris 8

Figure B.4: ADSOR web interface of the hard modules extraction results.