HAL open science

Découverte automatique de schéma pour les données irrégulières et massives

Redouane Bouhamoum

To cite this version:

Redouane Bouhamoum. Découverte automatique de schéma pour les données irrégulières et massives. Base de données [cs.DB]. Université Paris-Saclay, 2021. Français. NNT : 2021UPASG081 . tel-03526247

HAL Id: tel-03526247

https://theses.hal.science/tel-03526247

Submitted on 14 Jan 2022

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.

Découverte automatique de schéma pour les données irrégulières et massives

Automatic Schema Discovery for Large and Irregular Data

Thèse de doctorat de l'Université Paris-Saclay

École doctorale n° 580
Sciences et Technologies de l'information et de la communication (STIC)
Spécialité de doctorat : Informatique
Unité de recherche : Université Paris-Saclay, UVSQ, Données et Algorithmes pour une ville intelligente et durable, 78035, Versailles, France.
Référent : Université de Versailles Saint-Quentin en Yvelines

Thèse présentée et soutenue à Paris-Saclay, le 06/12/2021, par

Redouane BOUHAMOUM

Composition du jury

Jérôme Darmont Professeur, Université Lumière Lyon 2	Président
Mohand-Saïd Hacid Professeur, Université Claude Bernard Lyon 1	Rapporteur & Examinateur
Maguelonne Teisseire Directrice de recherche, INRAE	Rapporteur & Examinatrice
Fatiha Saïs Professeur, Université Paris-Saclay	Examinatrice

Direction de la thèse

Zoubida Kedad MCF-HDR, Université de Versailles	Directrice
Stéphane Lopes MCF, Université de Versailles	Co-encadrant

Thèse de doctorat

NNT : 2021UPASG081

ÉCOLE DOCTORALE
Sciences et technologies de l'information et de la communication (STIC)
université PARIS-SACLAY

Titre : Découverte automatique de schéma pour les données irrégulières et massives

Mots clés : Découverte de schéma ; Données RDF ; Clustering ; Big Data ; Découverte incrémentale de schéma ; Saturation des données RDF

Résumé : Le web des données est un espace dans lequel de nombreuses sources sont publiées et interconnectées, et qui repose sur les technologies du web sémantique. Cet espace offre des possibilités d'utilisation sans précédent, cependant, l'exploitation pertinente des sources qu'il contient est rendue difficile par l'absence de schéma décrivant leur contenu. Des approches de découverte automatique de schéma ont été proposées, mais si elles produisent des schémas de bonne qualité, leur complexité limite leur utilisation pour des sources de données massives. Dans notre travail, nous nous intéressons au problème du passage à l'échelle de la découverte de schéma à partir de sources de données RDF massives dont le schéma est incomplet ou absent. Nous nous intéressons également à l'incrémentalité de ces approches et à la prise en compte de connaissances implicites fournies par une source de données.
Notre première contribution consiste en une approche scalable de découverte de schéma qui permet l'extraction des classes décrivant le contenu d'une source de données RDF massive. Pour cela, nous avons d'abord proposé d'extraire une représentation condensée d'une source de données RDF qui servira en entrée du processus de découverte de schéma afin d'en améliorer les performances. Cette représentation est un ensemble de patterns qui correspondent à des combinaisons de propriétés décrivant les entités du jeu de données. Nous avons ensuite proposé une approche scalable de découverte de schéma fondée sur un algorithme de clustering distribué qui forme des groupes d'entités structurellement similaires représentant les classes du schéma.
Notre deuxième contribution a pour but de maintenir le schéma extrait cohérent avec les changements survenant au niveau des sources RDF, ces dernières étant en constante évolution. Nous proposons pour cela une approche incrémentale de découverte de schéma qui modifie l'ensemble des classes extraites en propageant dans ces dernières les changements survenus dans les sources.
Enfin, dans la troisième contribution de notre travail, nous adaptons notre approche de découverte de schéma afin qu'elle prenne en compte toute la sémantique portée par la source de données, qui est représentée par les triplets explicitement déclarés, mais également tous ceux qui peuvent en être déduits par inférence. Nous proposons une extension permettant de prendre en compte toutes les propriétés d'une entité lors de la découverte de schéma, qu'elles correspondent à des triplets explicites ou implicites, ce qui améliorera la qualité du schéma produit.

Title : Automatic Schema Discovery from large and irregular data

Keywords : Schema Discovery ; RDF Data ; Clustering ; Big Data ; Incremental Schema Discovery ; RDF Saturation

Abstract : The web of data is a huge global data space, relying on semantic web technologies, where a high number of sources are published and interlinked. This data space provides an unprecedented amount of knowledge available for novel applications, but the meaningful usage of its sources is often difficult due to the lack of schema describing the content of these data sources. Several automatic schema discovery approaches have been proposed, but while they provide good quality schemas, their use for massive data sources is a challenge as they rely on costly algorithms. In our work, we are interested in both the scalability and the incrementality of schema discovery approaches for RDF data sources where the schema is incomplete or missing. Furthermore, we extend schema discovery to take into account not only the explicit information provided by a data source, but also the implicit information which can be inferred.
Our first contribution consists of a scalable schema discovery approach which extracts the classes describing the content of a massive RDF data source. We have proposed to extract a condensed representation of the source, which will be used as an input to the schema discovery process in order to improve its performances. This representation is a set of patterns, each one representing a combination of properties describing some entities in the dataset. We have also proposed a scalable schema discovery approach relying on a distributed clustering algorithm that forms groups of structurally similar entities representing the classes of the schema.
Our second contribution aims at maintaining the generated schema consistent with the data source it describes, as this latter may evolve over time. We propose an incremental schema discovery approach that modifies the set of extracted classes by propagating the changes occurring at the source, in order to keep the schema consistent with its evolutions.
Finally, the goal of our third contribution is to extend schema discovery to consider the whole semantics expressed by a data source, which is represented not only by the explicitly declared triples, but also by the ones which can be inferred through reasoning. We propose an extension allowing to take into account all the properties of an entity during schema discovery, represented either by explicit or by implicit triples, which will improve the quality of the generated schema.

Maison du doctorat de l'Université Paris-Saclay
2ème étage aile ouest, Ecole normale supérieure Paris-Saclay
4 avenue des Sciences,
91190 Gif sur Yvette, France

Remerciements

Je tiens à remercier toutes les personnes qui ont contribué au succès de ma thèse et qui m'ont aidé à bien mener ce projet.

J'aimerais tout d'abord remercier mes directeurs de thèse, Zoubida Kedad et Stéphane Lopes pour leur encadrement tout au long de cette thèse, leurs remarques éclairées et leur soutien. Leurs conseils étaient très précieux et allaient bien au-delà de l'obtention d'un titre universitaire. J'ai beaucoup appris à leurs côtés et je leur adresse ma gratitude pour tout cela, ce fut un réel plaisir de travailler ensemble.

C'est un grand honneur pour moi d'adresser mes respectueux remerciements à Mohand-Saïd Hacid et Maguelonne Teisseire pour avoir accepté d'être les rapporteurs de ma thèse et pour le temps qu'ils auront bien voulu y consacrer.

Mes vifs remerciements vont également à Jérôme Darmont et Fatiha Saïs pour l'honneur qu'ils me font en acceptant de participer au jury de ma thèse.

Je remercie tous les membres de l'équipe ADAM, tous mes amis et collègues du laboratoire DAVID pour leur accueil et leur gentillesse. Je remercie également toute l'équipe pédagogique de l'université de Versailles, Saint-Quentin-en-Yvelines et les intervenants professionnels responsables de ma formation, pour avoir assuré la partie théorique de celle-ci.

Je remercie mes parents qui ont su croire en moi et qui m'ont apporté toute leur aide quand j'en ai eu besoin. Je remercie également toute ma famille qui a contribué de près ou de loin à ce que je suis devenu. Je remercie également ma belle famille pour leurs encouragements.

Enfin, je remercie mon épouse pour son enthousiasme à l'égard de mes travaux, son soutien quotidien et sa patience indéfectible, spécialement durant la période de rédaction de ce manuscrit et les préparations de la soutenance.

Table des matières

Table des figures **vi**

Liste des tableaux **ix**

Liste des algorithmes **x**

1 Introduction générale **1**

1.1 Contexte et motivations 1

1.2 Problématiques 5

1.3 Objectifs et contributions 7

1.4 Organisation du manuscrit 9

2 État de l'art **11**

2.1 Introduction 11

2.2 Approche de découverte de schéma 14

2.2.1 Approche de regroupement d'instances par similarité structurelle 15

2.2.2 Approche de regroupement de chemins similaires 21

2.2.3 Bilan des approches de découverte de schéma 25

2.3 Approches de découverte de patterns 28

2.3.1 Approche de découverte de patterns exacts 29

2.3.2 Approches de découverte de patterns approximatifs 33

2.3.3 Bilan des approches de découverte de patterns structurels 36

2.4 Approches connexes au problème de découverte de schéma 37

2.4.1 Approches d'enrichissement de schéma 37

2.4.2 Approches de typage d'instances 38

2.4.3 Bilan des approches connexes 39

2.5 Algorithmes de clustering basés sur la densité pour la découverte de schéma 40

2.5.1 Étude de l'adéquation des algorithmes de clustering pour la découverte de schéma 40
2.5.2 Principe de DBSCAN [28] 43
2.5.3 Algorithme DBSCAN scalables 44
2.5.4 Extensions incrémentales de DBSCAN 52
2.5.5 Bilan des extensions scalables ou incrémentales de l'algorithme DBSCAN 56
2.6 Conclusion 59

3 Découverte de schéma à partir des sources massives de données RDF 62
3.1 Introduction 62
3.2 Préliminaires 64
3.3 Problématique 67
3.4 Extraction d'une représentation condensée d'une source RDF 69
3.5 Approche générale pour la découverte de schéma 72
3.6 Distribution des données 74
3.6.1 Principe de distribution des entités 75
3.6.2 Optimisation de la distribution 77
3.6.3 Gestion des grands chunks 80
3.7 Identification des cores 82
3.8 Clustering Local 84
3.9 Génération des classes 86
3.10 La découverte de schéma appliquée à des patterns 88
3.11 Conclusion 89

4 Découverte incrémentale de schéma à partir de sources de données RDF 91
4.1 Introduction 91
4.2 Définition de la problématique 92
4.3 Principe d'une approche de découverte incrémentale de schéma 94
4.4 La distribution des données 95
4.4.1 Distribution des nouvelles entités 96
4.4.2 L'assignation des anciennes entités 97
4.5 Calcul de voisinage des nouvelles entités 99
4.6 Génération du nouveau schéma 102
4.6.1 Modification locale des clusters 102
4.6.2 Génération du nouveau schéma 106
4.7 Conclusion 108

5 Prise en compte des triplets implicites pour la découverte de schéma **111**

5.1 Introduction . . . 111

5.2 Définition de la problématique . . . 113

5.3 Préliminaires . . . 114

5.4 Principe général de notre approche . . . 118

5.5 Saturation des propriétés des entités . . . 120

5.6 Prise en compte des triplets implicites pendant la découverte de schéma . . . 125

5.6.1 Calcul de la sélectivité des propriétés . . . 126

5.6.2 Saturation et distribution des entités . . . 129

5.6.3 Découverte de schéma . . . 133

5.7 Conclusion . . . 134

6 Expérimentations **136**

6.1 Introduction . . . 136

6.2 Environnement de test . . . 137

6.2.1 Description des clusters de calcul . . . 137

6.2.2 Jeux de données . . . 137

6.3 Évaluation de la taille de la représentation condensée . . . 139

6.4 Évaluation de la scalabilité de l'approche de découverte de schéma . . . 141

6.4.1 Approche scalable de découverte de schéma . . . 141

6.4.2 Approche incrémentale de découverte de schéma . . . 146

6.5 Évaluation de la qualité du schéma . . . 149

6.6 Conclusion . . . 152

7 Conclusion **154**

7.1 Résumé des contributions . . . 154

7.2 Perspectives . . . 156

Liste des publications **159**

Bibliographie **161**

Table des figures

1.1 L'évolution du web sémantique entre 2007 et 2021 [35] 2
1.2 Exemple de graphe RDF 3
1.3 Exemple d'entités décrites par des propriétés explicites et implicites 7

2.1 Exemple de jeu de données RDF (a) et les clusters résultants de l'application de l'algorithme hiérarchique (b) [19] 17
2.2 Exemple de jeu de données et le treillis de concepts associé (composite (c), square (s),even (e), odd (o) et prime (p)) [69] 19
2.3 Exemple de jeu de données OEM et le treillis correspondant 20
2.4 Exemple d'un graphe de données OEM et son dataguide 22
2.5 Exemple de graphe OEM 23
2.6 Dataguide exact vs. Dataguide approximatif du graphe présenté dans la figure 2.5 24
2.7 Exemple de regroupement d'entités en considérant les propriétés sortantes et entrantes 24
2.8 Graphe de données RDF (a) et son graphe réduit par bisimulation (b) [96] 25
2.9 Graphe de données RDF et les patterns structurels qui représentent ses instances 29
2.10 Exemple de flux de données RDF et les patterns détectés en utilisant l'algorithme FreGraPad [10] . . 30
2.11 Exemple de jeux de données RDF et le résumé correspondant [15] 32
2.12 Exemple de jeu de données RDF et le Baseline Summary correspondant [16] 35
2.13 Le Refined Summary correspondant au jeu de données présenté dans la figure 2.12 [16] 36
2.14 Les concepts de base de DBSCAN 43
2.15 Illustration du partitionnement d'un espace de données en utilisant la méthode BSP [50] 45
2.16 Exemple de représentation de clusters [95] 48
2.17 Décomposition en cellules d'un espace de données par l'algorithme HPDBSCAN [42] 49
2.18 Exemple de graphe de cellules [100] 51
2.19 Interconnectivité entre deux régions 54

3.1 Exemple de sources de données RDF interconnectées . . . 65
3.2 Exemple de jeu de données RDF et le schéma correspondant . . . 66
3.3 Exemple de jeu de données RDF . . . 70
3.4 Vue globale de notre approche de découverte de schéma . . . 73
3.5 Exemple de jeu de données décrivant des auteurs et leurs publications dans des conférences . . . 76
3.6 La distribution de D dans des $chunks$ . . . 76
3.7 La distribution optimisée de D dans des partitions . . . 80
3.8 Formation des clusters locaux dans chaque $chunk$ . . . 85
3.9 Une illustration du principe de fusion des clusters locaux . . . 86
3.10 Les clusters finaux correspondant aux classes du schéma découvert à partir du jeu de données D . . . 87

4.1 Exemple d'un ensemble d'entités et le schéma correspondant . . . 93
4.2 Vue globale de l'approche incrémentale de découverte de schéma . . . 94
4.3 Les $chunks$ générés après la distributions des entités mises à jour Δ_D . . . 97
4.4 Assignation des anciennes entités aux $chunks$ créés . . . 98
4.5 L'identification des cores dans chaque $chunk$ . . . 101
4.6 Les différentes modifications possibles sur l'ensemble des clusters . . . 103
4.7 Clustering des entités dans les $chunks$ . . . 104
4.8 Résultat du clustering des entités de notre exemple . . . 107
4.9 La mise à jour du schéma après l'insertion des nouvelles entités . . . 108

5.1 Exemple de jeu de données RDF et les classes correspondantes . . . 112
5.2 Vue globale de l'approche de découverte de schéma avec prise en compte des propriétés implicites . 120
5.3 Ordre d'exécution des règles de saturation . . . 122
5.4 Exemple de déclarations schéma dans une source de données RDF . . . 123
5.5 Distribution des déclarations schéma . . . 124
5.6 L'exécution parallèle de la saturation des entités du jeu de données . . . 131
5.7 La description des entités du jeu de données enrichie par les propriétés implicites . . . 131
5.8 Les $chunks$ résultant de la distribution des entités après l'enrichissement de leurs descriptions . . . 133
5.9 Les classes décrivant les entités du jeu de données D après leur enrichissement . . . 134

6.1 Évaluation de la scalabilité de notre approche sur différents jeux de données synthétiques . . . 142
6.2 Évaluation de l'impact du nombre de propriétés sur le temps d'exécution . . . 143
6.3 Évaluation de l'accélération de l'algorithme exécuté sur des clusters de différentes configurations . . . 144
6.4 Évaluation du temps de clustering de sous-ensembles de DBpedia . . . 145

6.5 Comparaison de notre algorithme de clustering avec l'algorithme NG-DBSCAN 146
6.6 Algorithme incrémental vs. scalable 147
6.7 Le clustering d'un sous-ensemble de DBpedia 149
6.8 La qualité des classes découvertes pour différentes valeurs de ϵ 151

Liste des tableaux

2.1 Synthèse des approches de découverte de schéma 26
2.2 Synthèse des approches de découverte de patterns 37
2.3 Synthèse des versions scalables de l'algorithme DBSCAN 57
2.4 Synthèse des algorithmes DBSCAN incrémentaux 58

3.1 Les patterns extraits à partir du jeu de données de la figure 3.3 70

5.1 Règles de saturation liées aux déclarations RDFS 116
5.2 Règles de saturation liées aux déclarations OWL 117
5.3 Règles de saturation 119
5.4 Le nombre d'occurrences des propriétés décrivant les entités de notre exemple 128

6.1 Caractéristiques des jeux de données 138
6.2 Caractéristiques des sous ensembles de Dbpedia 139
6.3 Ratio de réduction de la taille des jeux de données 139
6.4 Temps d'exécution de chaque opération 140

Liste des algorithmes

1 Extraction de patterns . . . 71
2 Distribution des entités . . . 80
3 Divisions des grands $chunks$ . . . 82
4 Identification des cores . . . 83
5 Clustering local . . . 86
6 Fusion des clusters locaux . . . 87
7 Distribution des nouvelles entités . . . 97
8 Assignation des anciennes entités . . . 99
9 Calcul de voisinage . . . 101
10 Calcul des clusters locaux . . . 105
11 Restructuration des classes . . . 108
12 La saturation du schéma . . . 125
13 Calcul de sélectivité . . . 130
14 Saturation d'entité . . . 132
15 Distribution des entités . . . 133

Chapitre 1

Introduction générale

1.1 Contexte et motivations

Le web des données est une évolution du Web, proposée par Tim Berners Lee en 2001 [11] et promue par le W3C [1], dans le but de partager et d'interconnecter des sources de données qui seront utilisées non seulement par des humains mais aussi par des agents logiciels. Le Web sémantique repose sur l'utilisation d'un ensemble de standards et de technologies qui ont été développés par le W3C [38]. Les sources du web des données sont décrites en utilisant le langage RDF (Resource Description Framework [2]) qui propose un modèle standard pour la représentation de données sur le Web. Le langage RDFS (RDF Schema [3]) intègre essentiellement le typage des ressources et de leurs relations par des classes de ressources et de propriétés qui peuvent être organisées en hiérarchies. Au-dessus de ces langages, OWL (Web Ontology Language [4]) propose un langage de plus en plus expressif (au sens de la logique) pour décrire formellement les concepts utilisés dans les annotations RDF et les relations entre ces concepts. Les sources de données RDF sont interrogées en utilisant SPARQL, un langage de requête et un protocole pour interroger et accéder aux contenus RDF [108].

Les technologies du web sémantique proposées par le W3C ont permit la publication de sources de données liées (Linked Open Data, LOD [35]) dans lesquelles les sources de données incluent des liens vers d'autres sources, formant ainsi un espace unique de données. Le lien entre deux ressources qui se rapportent en réalité au même objet réel est représenté par des déclarations $owl{:}sameAs$. Ces données peuvent être partagées et utilisées par plusieurs applications, entreprises et groupes d'utilisateurs afin de créer de nouvelles connaissances. Depuis sa mis en œuvre en 2007, le web des données a connu une évolution importante et inclut à ce jour 1 301 jeux de données de différentes organisations et fournisseurs de données, avec 16 283 liens [35]. La figure 1.1 montre l'évolution du web sémantique depuis sa création.

1. https://www.w3.org/standards/semanticweb/
2. RDF : https://www.w3.org/RDF/
3. https://www.w3.org/TR/rdf-schema/
4. OWL : https://www.w3.org/OWL/

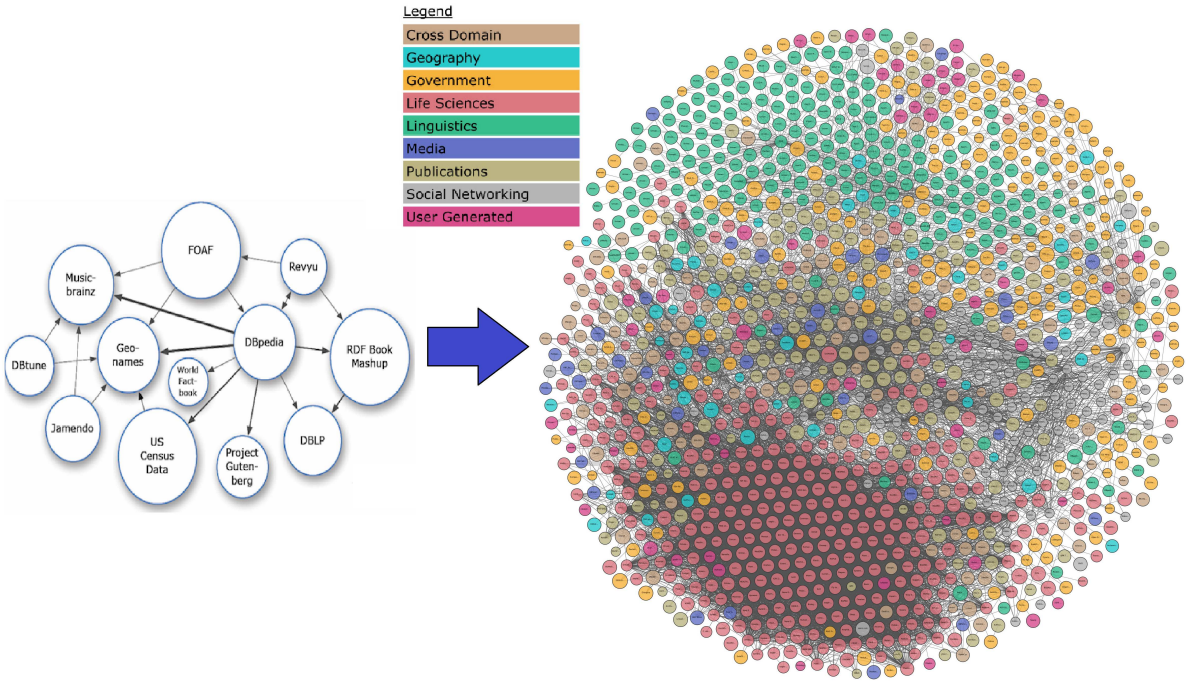


FIGURE 1.1 – L'évolution du web sémantique entre 2007 et 2021 [35]

Une des spécificités du langage RDF proposé pour la création des sources de données sur le web est qu'il permet une description flexible des données, et n'impose pas de contraintes structurelles comme c'est le cas par exemple des bases de données relationnelles, où un schéma est défini comme une contrainte sur la structure des données à insérer dans la base. Il est possible de spécifier un schéma pour des sources RDF, mais les données ne sont pas contraintes par ce schéma et il est possible qu'elles ne soient pas conformes à son contenu. Afin de faciliter l'exploitation des sources de données RDF, il est recommandé d'utiliser des vocabulaires standards comportant des classes, propriétés et relations qui peuvent être utilisés pour décrire les données et les méta-données. Le Linked Open Vocabularies (LOV) inventorie les ontologies et vocabulaires de description des jeux de données RDF mais également les relations qui lient ces vocabulaires [106] comme par exemple le vocabulaire *FOAF* permettant de décrire des personnes et les relations qui peuvent exister entre elles.

La figure 1.2 montre un exemple de jeu de données RDF représenté comme un graphe dont les sommets représentent les ressources et les arcs représentent des propriétés de ces ressources. Les ressources contenues dans une source de données RDF sont identifiées par des URI (Uniform Resource Identifiers). Les URI sont des identifiants uniques permettant de nommer et de faire référence à n'importe quelle ressource dans le Web des données. Par exemple dans DBpedia, Versailles est représenté par l'URI `https://dbpedia.org/page/Versailles,_Yvelines`.

Une source dans le web des données comporte des données ainsi que des déclaration schéma qui représentent des méta-données décrivant ces données. Le schéma nous renseigne notamment sur le type des entités de la source ainsi que leurs structures. Par exemple, les ressources "100007" et "106759" de la figure 1.2 repré-

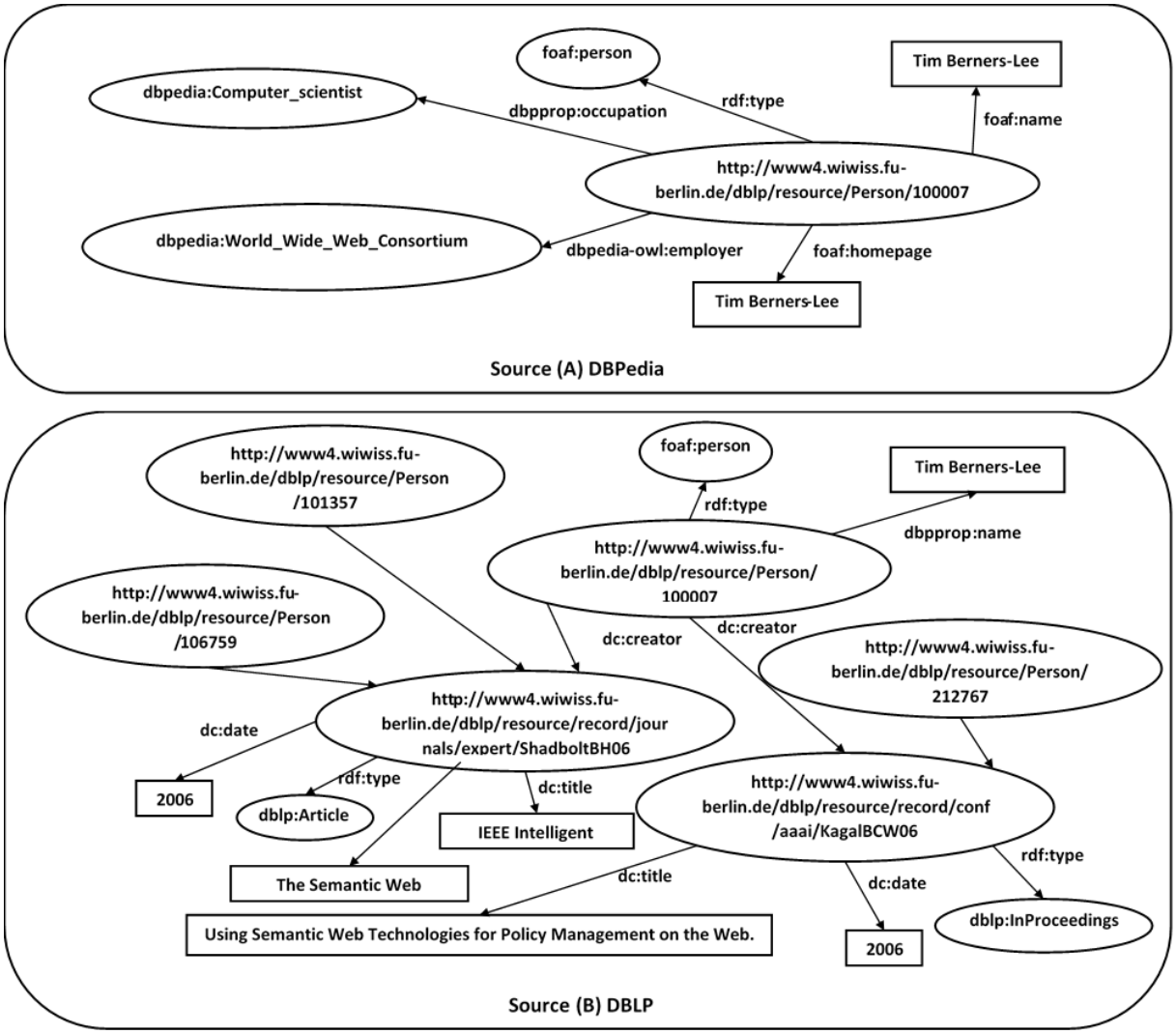


FIGURE 1.2 – Exemple de graphe RDF

sentent des données, et les ressources "$foaf{:}Person$" et "$dblp{:}Article$" représentent des déclarations schéma qui renseignent sur les types ou les classes des données contenues dans la source.

Cependant, le schéma associé à un jeu de données RDF peut être incomplet ou absent. Comme nous pouvons le voir dans l'exemple, les déclarations liées au schéma ne sont pas toujours fournies. Par exemple, les déclarations de type, renseignées par des liens $rdfs{:}type$, sont absentes pour certaines ressources, comme pour la ressource "106759". Le langage RDF utilisé pour la création des sources de données du web sémantique offre une grande flexibilité et n'impose pas que les données soient conformes au schéma. En conséquence, des entités appartenant à la même classe peuvent être décrites par des ensembles de propriétés hétérogènes, et ces entités peuvent en plus avoir dans leurs descriptions des propriétés non déclarées dans le schéma. Par exemple, les ressources "100007" et "212567" sont toutes les deux de type "$foaf{:}Person$" mais elles sont décrites par des ensembles différents de propriétés.

Dans le but d'aider les utilisateurs à exploiter les sources de données RDF, plusieurs approches ont été proposées pour la découverte du schéma qui décrit les instances contenues dans une source de données RDF. Ce schéma apporte des informations sur le contenu de la source et facilite son utilisation. Certaines de ces approches analysent la structure des entités afin de former les classes qui les décrivent. Cependant, les techniques utilisées par ces dernières sont coûteuses en temps d'exécution et leur utilisation sur de grandes sources de données est impossible. D'autres approches s'appuient sur des déclarations schéma pour former des patterns structurels représentant les entités d'une source de données RDF. Le problème de ces approches est que les déclarations sur le schéma ne sont pas toujours fournies. Il existe également des approches qui proposent d'enrichir le schéma existant dans une source de données RDF. Ces dernières complètent les déclarations décrivant les entités dans un jeu de données RDF, mais ne découvrent pas de nouvelles classes. Dans notre travail, nous nous intéressons à la découverte de schéma à partir des sources de données massives, qui évoluent fréquemment, sans faire d'hypothèse sur l'existence de déclarations liées au schéma dans la source. Nous visons également dans nos solutions à considérer aussi bien les informations explicites dans une source de données que les connaissances implicites qui peuvent être dérivées en utilisant des règles d'inférence.

Le schéma extrait peut être utilisé pour différents objectifs, nous citons dans ce qui suit quelques exemples :

Fournir une vue synthétique d'un jeu de données RDF. Le schéma découvert offre un résumé des classes des entités dans un jeu de données. Cette vue globale peut être utilisée afin de comprendre le contenu d'un jeu de données RDF et d'évaluer sa capacité à répondre aux besoins d'une application.

Interconnexion des jeux de données RDF. Une caractéristique clé des jeux de données RDF est qu'ils incluent des liens vers d'autres jeux de données, ce qui permet de naviguer dans le web des données comme dans un espace unique. Ces liens sont représentés par les propriétés $owl{:}sameAs$ [5], et la détermination de ces liens est un problème connu sous le nom d'interconnexion (interlinking). Des outils ont été proposés pour réaliser cette tâche, comme Knofuss [6] ou Silk [7], qui ont été utilisés pour interconnecter Yago [74] et DBpedia [7]. Ces outils exigent des informations sur le type et les propriétés du jeu de données afin de générer les liens `owl:sameAs` appropriés entre ces jeux de données. Le schéma d'une source contient ces informations et peut ainsi se révéler très utile pour interconnecter des jeux de données.

Interroger les jeux de données RDF. L'absence d'information sur les classes, propriétés et les données contenues dans les jeux de données RDF rend leur interrogation complexe. En effet, ces informations sont indispensables afin de formuler une requête dans le langage utilisé pour interroger les jeux de données RDF comme Sparql [108],

5. sameAs : `https://www.w3.org/2001/sw/wiki/SameAs.`
6. Knofuss : `https://technologies.kmi.open.ac.uk/knofuss`
7. Silk : `http://silkframework.org/`

dont les requêtes sont décrites par des patterns de graphe. Par exemple, le pattern *"?x name ?y"* retournera tous les objets ayant la propriété *name*. L'expression de ces patterns nécessite certaines informations sur le sujet, la propriété ou la valeur qui compose ces patterns.

Un schéma décrivant la structure des données fournit ces informations et facilite considérablement la formulation des requêtes. Il pourrait même être utilisé pour développer des outils qui assistent l'utilisateur dans la formulation des requêtes, comme celui proposé dans [15]. De plus, le schéma pourrait permet la création d'un index sur les entités afin d'accélérer l'exécution des requêtes. Il peut aussi permettre la sélection des sources pertinentes qui comportent des éléments de réponse aux requêtes exécutées sur un jeu de données distribué.

Les applications ci-dessus sont des exemples parmi beaucoup d'autres qui illustrent l'utilité d'un schéma décrivant un jeu de données RDF, et qui montrent les raisons pour lesquelles la découverte de schéma a été identifiée comme un défi important dans la gestion des données [3].

Dans notre travail, nous nous intéressons à la découverte de schéma décrivant les instances contenues dans une source de données massives, sur lesquelles les approches d'extraction de schéma existantes ne sont pas applicables en raison de leur complexité de calcul. Nous nous intéressons en particulier à proposer des solutions scalables et incrémentales au problème de découverte de schéma à partir de source de données RDF qui sont soumises à des évolutions au cours du temps ; ces solutions doivent garantir la qualité du schéma obtenu.

1.2 Problématiques

Le but de notre travail est de découvrir le schéma décrivant une source de données RDF massives. Différentes approches ont été proposées pour découvrir automatiquement le schéma implicite décrivant un jeu de données en analysant la description des entités. Cependant, nous montrerons dans l'état de l'art que la complexité des algorithmes sur lesquelles s'appuient ces approches rend leur utilisation impossible sur les jeux de données massives.

Dans le but de proposer un schéma décrivant la structure des entités contenu dans une source de données RDF massives, nous avons identifié et abordé les problématiques suivantes :

Découverte de schéma implicite à partir des données RDF massives

La première problématique à laquelle nous nous sommes intéressés est la découverte de schéma à partir des sources de données RDF massives en se basant sur l'analyse structurelle des instances. L'extraction de schéma consiste à construire les classes décrivant les instances à partir de la structure de ces dernières. Notre travail se situe dans la continuité des approches qui construisent les classes qui décrivent les données contenues dans une source RDF en regroupant les instances décrites par des ensembles de propriétés similaires et qui appartiennent potentiellement à la même classe [66, 65, 20, 21]. L'intuition est que les instances de la même classe possèdent des propriétés similaires.

Les approches existantes ont efficacement adopté des algorithmes de clustering dans le but d'identifier les groupes d'instances similaires, et ont fourni des classes de bonne qualité, avec une bonne précision et un bon rappel. Cependant, la complexité de ces algorithmes rend impossible leur utilisation pour le traitement de grandes sources de données du web. Même s'il existe des versions scalables de ces algorithmes de clustering, leur application direct pour la découverte de schéma n'est pas évidente car elles imposent des contraintes qui ne s'appliquent pas aux données RDF, ou parce qu'elles réduisent la qualité du schéma produit.

Par conséquent, le défi posé est d'assurer la scalabilité de la découverte de schéma pour permettre le traitement des grandes sources de données tout en offrant un schéma de bonne qualité, qui permet l'identification des classes du jeu de données avec une bonne précision et un bon rappel. Dans notre travail, nous nous intéressons à découvrir le schéma à partir de la description des entités, sans s'appuyer sur les déclarations du schéma dans la source de données. Le problème qui se pose est comment identifier les groupes d'entités similaires représentant les classes du schéma à partir de grande sources de données RDF, où les approches d'extraction de schéma existantes échouent dans le traitement du volume de ces dernières en raison de leur complexité ?

Incrémentalité de la découverte de schéma

En plus du grand volume de données, les sources de données RDF sont en constante évolution au fil du temps et de nouvelles instances sont régulièrement intégrées au jeu de données initial. Cette évolution peut rendre le schéma qui décrit une source de données RDF incohérent avec son contenu, et il ne représente plus alors une bonne description des entités qu'elle contient. Ainsi, le schéma décrivant une source de données RDF doit être modifié lorsque le jeu de données est mis à jour.

Les sources de données RDF peuvent aussi être créées et alimentées par des flux de données continus qui exigent l'extraction de schéma à intervalle régulier.

Le schéma décrivant une source de données RDF peut être extrait en appliquant une approche de découverte de schéma sur tout le jeu de données après chaque mise à jour. Cependant, lorsque les jeux de données RDF sont massives et que les mises à jour sont fréquentes, il serait très coûteux de ré-exécuter une approche de découverte de schéma sur la totalité des données afin d'en extraire un schéma à jour.

Les fréquentes mises à jour des données RDF font émerger le besoin de faire évoluer le schéma qui les décrit, en considérant seulement les nouvelles données ajoutées à la source. De plus, faire évoluer le schéma ne doit pas réduire sa qualité. Construire ce schéma incrémentalement doit produire un résultat de même qualité que celui généré par les approches qui s'exécutent sur la totalité du jeu de données.

Prise en compte des triplets implicites dans la découverte de schéma

Dans les sources de données RDF, les entités sont décrites par des propriétés explicitement renseignées par l'utilisateur, et des propriétés implicites dérivées en utilisant des règles d'inférences qui s'appuient sur des décla-

rations liées au schéma. La figure 1.3 représente un exemple d'entités RDF décrites par des propriétés explicitement renseignées par l'utilisateur, et des propriétés implicites dérivées en s'appuyant sur des déclarations liées au schéma. Par exemple, la propriété "$fatherOf$" de l'entités "$Paul$" est une propriété explicite. En considérant que la propriété "$fatherOf$" est une sous propriété de "$famillyMember$", nous déduisons implicitement que l'entité "$Paul$" est également décrite par $famillyMember$. Nous pouvons également déduire que l'entité "Sam" est décrite par la propriété "$sonOf$" si nous considérons que la propriété "$sonOf$" est une propriété inverse de "$fatherOf$".

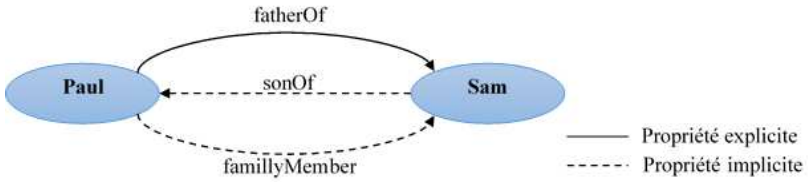


FIGURE 1.3 – Exemple d'entités décrites par des propriétés explicites et implicites

Cependant, les approches de découverte de schéma existantes s'appuient uniquement sur les propriétés explicites décrivant les données afin de regrouper les instances similaires et qui appartiennent à la même classe. La prise en compte des propriétés implicites enrichirait la description des entités et permettrait d'améliorer le résultat des approches de découverte de schéma.

Une solution pour cela serait de générer toute la connaissance implicite, puis de procéder à la découverte du schéma sur la source ainsi enrichie. Mais cela pourrait augmenter la taille de la source de façon significative. Le problème qui se pose est donc de déterminer ces connaissances implicites qui enrichissent les instances du jeu de données, et de les prendre en compte dans le processus de découverte de schéma sans recourir à une étape préalable où ces connaissances seraient générées et matérialisées. Afin de résoudre ce problème, nous devons relever les défis suivants : comment adapter l'approche de découverte de schéma scalable afin de tenir compte des propriétés dérivées pour construire les classes du schéma ? et comment assurer la scalabilité de l'approche proposée ?

Nous introduisons dans la section suivante nos contributions abordant ces problématiques.

1.3 Objectifs et contributions

Dans notre travail, nous proposons différentes contributions au problème de découverte de schéma. Nous nous sommes intéressés aux verrous posés par le passage à l'échelle, l'incrémentalité et la prise en compte de la sémantique implicite d'une source de données. Nos contributions sont les suivantes :

Extraction de patterns pour des données RDF [ICDEW-DESWEB 2018]

Afin de faire passer à l'échelle les approches de découverte de schéma, nous avons proposé de construire une représentation condensée du jeu de données initial en extrayant des patterns qui représentent toutes les combinaisons de propriétés décrivant les entités dans le but de réduire le nombre d'entrées de l'approche de découverte de schéma. Un pattern est un ensemble de propriétés distinctes tel qu'il existe au moins une entité décrite par cet ensemble de propriétés. L'approche de découverte de schéma est ainsi appliquée sur les patterns au lieu du jeu de données initial. Les patterns représentent toutes les structures décrivant les entités dans une source de données RDF, ce qui garantit que l'application d'une approche de découverte de schéma sur les patterns fournit le même résultat que sur les entités initiales.

Découverte scalable de schéma à partir des données massives[BDCSIntell 2018, TLDKS 2020]

Nous proposons de découvrir le schéma implicite d'une source de données RDF massive en regroupant les entités qui partagent des descriptions similaires et qui représentent des instanciations d'une même classe. En effet, les approches existantes ne traitent pas les grandes sources de données RDF en raison de la complexité des algorithmes qu'elles utilisent. Dans notre travail, nous avons introduit un algorithme de clustering basé sur la densité capable de calculer les clusters d'entités afin d'identifier et regrouper les entités similaires à partir de grandes sources de données RDF et de produire les classes composant le schéma décrivant la source.

Pour cela, nous avons proposé une méthode de distribution originale qui divise le jeu de données initial en sous-ensembles de données par rapport aux propriétés décrivant les entités. Notre intuition est de construire des sous-ensembles comportant des entités qui partagent des propriétés communes et qui sont susceptibles d'être similaires. Nous avons ensuite fourni un algorithme de clustering par densité scalable qui traite ces sous-ensembles de données en parallèle. En plus de sa conception parallèle, notre approche produit le même résultat que l'algorithme séquentiel, ce qui permet d'offrir non seulement un traitement rapide des grands sources de données, mais également un schéma de bonne qualité. Nous avons mené des expérimentations approfondies afin de montrer l'efficacité de notre proposition. Nous avons implémenté notre approche en utilisant la plateforme de calcul distribué Spark [104], dont le code source est disponible en ligne [8].

Construction incrémentale de schéma [ESWC 2021, BDA 2021]

Les données RDF sont en constante évolution et de nouvelles instances sont souvent ajoutées au jeu de données initial. En conséquence, le schéma décrivant le jeu de données peut devenir incohérent avec les données qui évoluent. Il ne représente donc plus une bonne description du contenu de la source si des nouvelles instances sont insérées.

8. https://github.com/BOUHAMOUM/SC-DBSCAN

Pour assurer la mise à jour du schéma suite à l'évolution des données, nous avons proposé une approche incrémentale de découverte de schéma à partir de jeux de données RDF massifs. Notre approche se base sur un algorithme de clutering basé sur la densité, scalable et incrémental qui propage les mises à jour survenant dans un jeu de données sur les clusters existants qui correspondent aux classes du schéma. L'objectif de notre approche est de mettre à jour le schéma en considérant uniquement les nouvelles entités et les anciennes entités impactées par les modifications. Ainsi, le clustering est restreint aux nouvelles entités et aux anciennes qui se trouvent dans le voisinage des nouvelles. Notre proposition a été conçue afin de permettre un calcul parallèle des nouveaux clusters et un traitement rapide des grandes sources de données tout en produisant un résultat de bonne qualité.

Utilisation des connaissances implicites pour la découverte de schéma

Les entités contenues dans une source de données RDF sont décrites par des propriétés explicites et d'autres implicites. En effet, les sources de données RDF comportent des des déclarations liées au schéma, que l'on peut considérer comme des méta-données qui offrent la possibilité de dériver de nouvelles informations à partir de celles qui sont explicitement définies dans le jeu de données. L'extraction des connaissances implicites pour une source de données RDF est connue dans la littérature sous le nom de saturation de graphe RDF ou raisonnement sur un graphe RDF [32, 44].

Dans notre travail, nous proposons une approche de découverte de schéma qui tient compte des propriétés explicitement renseignées dans le jeu de données, et également des propriétés implicites dérivées en utilisant les déclarations schéma dans une source de données RDF.

Nous adaptons notre approche de découverte de schéma afin qu'elle prenne en compte les descriptions explicites et implicites des entités contenues dans une source de données. Nous pouvons pour cela exploiter les déclarations sur le schéma fournies avec les données, ou encore une source extérieure comme par exemple une ontologie de domaine. L'approche doit considérer durant le clustering des entités aussi bien leurs propriétés explicites que leurs propriétés implicites. Elle doit également être scalable afin de permettre la découverte de schéma à partir des données massives.

1.4 Organisation du manuscrit

Le présent manuscrit est composé de sept chapitres.

Le chapitre 1 introduit le contexte général ainsi que les motivations de notre travail. Il résume nos objectifs et contributions.

Le chapitre 2 est consacré à l'état de l'art. Nous présentons tout d'abord les différentes approches relatives à l'extraction du schéma ainsi qu'un bilan de leurs limites. Nous analysons ensuite les algorithmes de clustering scalables et incrémentaux afin d'étudier leur utilisation pour la découverte de schéma à grande échelle.

Dans le chapitre 3, nous introduisons nos propositions pour la découverte de schéma pour des données RDF massives. Dans ce but, nous proposons deux approches complémentaires. Tout d'abord, nous présentons une méthode qui génère une représentation condensée d'une source de données RDF afin de réduire le nombre d'instances à traiter par l'approche de découverte de schéma. Nous présentons ensuite un algorithme de clustering scalable basé sur la densité, conçu pour la découverte de schéma à partir des grandes sources de données RDF, tout en assurant que le résultat est le même que celui produit par une version séquentielle.

Dans le chapitre 4, nous introduisons une approche incrémentale de découverte de schéma à partir des sources de données RDF massives qui s'appuie sur un algorithme de clustering basé sur la densité, scalable et incrémental. Cette approche assure la mise à jour du schéma décrivant une source RDF afin de le garder cohérent avec les données lorsque ces dernières évoluent par ajout de nouvelles entités.

Dans le chapitre 5, nous proposons une approche de découverte de schéma qui prend en compte les propriétés explicites et implicites décrivant les entités du jeu de données, dans le but d'améliorer la qualité du schéma produit. Notre approche exploite la sémantique incluse dans les sources de données RDF sous forme de déclarations schéma afin de dériver de nouvelles propriétés et ainsi enrichir la description des entités.

Dans le chapitre 6, nous présentons des évaluations expérimentales approfondies illustrant à la fois la qualité des résultats et les performances obtenues par nos approches de découverte de schéma.

Enfin, nous concluons le manuscrit dans le chapitre 7 par un résumé des contributions apportées et nous présentons des perspectives de recherche.

Chapitre 2

État de l'art

2.1 Introduction

Le web des données représente un espace d'information composé d'un nombre croissant de jeux de données interconnectés décrits avec des langages proposés par le W3C tel que RDF, RDFS et OWL. La caractéristique de ces langages est qu'ils sont flexibles et n'imposent pas de contraintes sur la structuration des données. Les sources de données du web comportent à la fois les données, et le schéma qui décrit ces données. Les bonnes pratiques lors de la création et de la publication des sources de données sur le web recommandent de fournir des déclarations liées au schéma, telles que les prédicats VoID [1], qui capturent différentes métadonnées sur le jeu de données. Ces déclarations aident les utilisateurs à comprendre la nature des entités contenues dans un jeu de données RDF et à documenter sur leur structure. Le problème est que ce schéma est souvent incomplet ou complètement absent, car les déclarations schéma ne sont pas obligatoires et ne sont donc pas toujours fournies. De plus, en raison de la nature flexible des langages du web sémantique, les données ne sont pas contraintes par le schéma même si ce dernier existe. Ainsi, des ressources de même type peuvent être décrites par des ensembles de propriétés différents de ceux spécifiés dans le schéma. L'absence de schéma contraignant les données offre une grande flexibilité lors de la création des sources, mais limite leur utilisation. En effet, la compréhension et l'exploitation d'une source de données RDF n'est pas triviale sans connaissance sur son contenu, des classes qu'elle comporte et des propriétés les décrivant. L'exploitation de ces sources serait facilitée si des informations décrivant leur contenu étaient disponibles.

L'extraction de représentations descriptives pour des sources de données irrégulières, semi-structurées ou non-structurées, comme c'est le cas des données RDF, a fait l'objet de plusieurs travaux de recherche. Ces approches peuvent être regroupées en plusieurs familles.

Les *approches de résumé de graphe RDF* regroupent les nœuds similaires afin de proposer une représentation condensée qui réduit la complexité des traitements effectués sur le jeu de données et le temps d'exécution des

1. VoID : The Vocabulary of Interlinked Datasets (`http://vocab.deri.ie/void/`)

requêtes en créant un index sur les instances contenues dans la source de données. Contrairement aux approches de découverte de schéma, les approches de résumé de graphe RDF n'analysent pas uniquement les propriétés qui représentent les dimensions décrivant les données afin de regrouper les nœuds similaires, mais s'intéressent aussi à la valeur de chaque propriété. Les nœuds regroupés sont ceux ayant des propriétés et des valeurs similaires. Ces approches sont ainsi moins concises car le résultat qu'elles produisent doit représenter tous les chemins possibles contenus dans le graphe de données [16, 15].

Les *approches de profilage de données RDF* consistent à extraire les caractéristiques d'une source de données. Elles décrivent une source de données et aident à sa compréhension en fournissant des statistiques sur son contenu, comme le nombre d'instances pour un type ou encore les propriétés les plus fréquentes dans la description des entités [1, 91, 76, 60, 27].

Les *approches de visualisation de graphes RDF* permettent la représentation graphique des données et offrent pour cela un ensemble de méthodes interactives. Ces approches permettent de résumer de manière visuelle les informations, comme les types contenus dans une source, et offrent la possibilité de naviguer dans les différentes entités de ce type. La représentation visuelle des données facilite l'extraction d'informations à partir des sources RDF [97, 103, 34].

Les *approches automatiques de découverte de schéma à partir des données RDF* analysent les structures qui décrivent les instances dans un jeu de données RDF afin d'extraire le schéma d'une source de données. Un schéma permet de représenter la structure des données d'une source en termes de classes et de liens entre elles. Dans le contexte du web des données, un schéma est vu comme un guide facilitant la compréhension du contenu d'une source de données et non pas comme une représentation à laquelle les données doivent se conformer.

Dans notre travail, nous nous intéressons à l'extraction d'informations sur le schéma, et en particulier les classes (types), à partir des sources de données RDF massives et irrégulières pour lesquelles les déclarations sur le schéma sont incomplètes ou absentes. Notre objectif est d'offrir une vue globale de la source de données et de faciliter la compréhension de son contenu. Ce problème a été identifié comme l'une des directions de recherche les plus importantes pour la gestion de données [2]. Ainsi, nous détaillons dans la suite du chapitre les approches de découverte de schéma à partir de données irrégulières. Les approches de résumé, de profilage et de visualisation des données RDF ne sont pas abordées dans ce chapitre car elles ne découvrent pas un schéma ou les structures qui décrivent les instances d'un jeu de données.

Plusieurs travaux de recherche s'intéressent à l'extraction de représentations descriptives des sources de données pour lesquelles le schéma est incomplet ou absent comme les données RDF, XML ou OEM. Ces approches tentent de construire une description structurelle des instances contenues dans une source de données afin d'offrir une vue globale sur les classes qu'elle comporte et les liens entre elles. Nous distinguons parmi ces approches trois catégories. La première catégorie regroupe des approches qui découvrent le schéma implicite d'une source de données en analysant la structure des données, sans utilisation des déclarations sur le schéma contenu dans la

source de données car ces dernières peuvent être absentes. Les approches de la deuxième catégorie proposent de découvrir des patterns structurels afin de représenter les structures qui décrivent les instances de la source de données. Les patterns ne représentent pas nécessairement des classes, mais des versions structurelles des instances du jeu de données. En d'autre terme, un pattern est un ensemble de propriétés représentant des entités ayant la même structure. Dans la troisième catégorie, les approches utilisent les déclarations existantes sur le schéma dans un jeu de données RDF afin de découvrir les types des entités non typées.

Pour analyser les différentes catégories d'approches, nous nous intéressons à la nature de la description structurelle produite, qui peut être des classes, des expressions régulières, des versions de classes, etc. Nous considérons aussi la question du passage à l'échelle des approches et leurs capacités à traiter des grandes sources de données. De plus, après avoir proposé un schéma, il est important de garder sa cohérence avec les données lorsque ces dernières évoluent. Aussi, nous étudions dans notre analyse l'incrémentalité des approches proposées. Enfin, nous étudions la prise en compte des triplets implicites : en effet, outres les triplets explicites décrivant les entités d'une source de données RDF, ces dernières sont décrites également par des triplets implicites dérivées en utilisant des règles d'inférences qui s'appuient sur des déclarations liées au schéma. Ces informations doivent être considérées dans le processus de découverte de schéma afin de représenter l'ensemble du contenu de la source.

Notre travail a pour but la découverte de schéma à partir d'une source de données RDF par analyse structurelle et regroupement des entités, sans aucune hypothèse sur la disponibilité des déclarations sur le schéma. Ainsi, nous discuterons dans notre état de l'art les algorithmes de clustering pouvant être utilisés par les approches de découverte de schéma afin d'explorer les structures des données, d'identifier les groupes d'entités similaires et d'en déduire le schéma définissant les classes et les propriétés qui décrivent un jeu de données. Bien que ces approches de découverte de schéma produisent un schéma de bonne qualité, la question du passage à l'échelle reste un problème ouvert en raison de la complexité des algorithmes de clustering sur lesquels elles s'appuient afin de regrouper les entités. Il existe des algorithmes de clustering scalables que nous étudierons dans cette état de l'art afin de déterminer leur adéquation à la découverte de schéma à partir des données RDF.

Parmi les algorithmes de clustering adaptés à la découverte de schéma, nous nous intéressons en particulier à DBSCAN, dont les caractéristiques le rendent bien adapté au contexte du web des données. En effet, son principe de regroupement par densité des entités similaires permet la construction de clusters de différentes formes et tailles. Cette caractéristique est importante dans un contexte où les entités appartenant au même type peuvent être décrites par des ensembles de propriétés hétérogènes. De plus, DBSCAN n'exige pas en entrée le nombre de clusters résultants, et détecte les entités bruitées qui ne sont pas assez importantes pour former une classe. Cependant, la complexité de DBSCAN ne permet pas son utilisation dans un contexte de données massives. Dans notre état de l'art, nous étudions les extensions proposées pour l'algorithme de clustering basé sur la densité afin d'assurer le passage à l'échelle, en particulier les approches parallèles qui pourraient être utilisées efficacement pour l'extraction de schéma à partir de jeux de données RDF massifs.

Une fois le schéma décrivant une source de données découvert, ce dernier doit être modifié lorsque le jeu de données est mis à jour, afin de rester cohérent avec les données. En effet, les données RDF sont sujettes à de fréquentes évolutions, et de nouvelles entités peuvent être ajoutées. Par exemple, entre la version 3.5 et la version 3.9 de DBpedia, le nombre d'entités de type $Person$ a été multiplié par 45 [89]. Suite à de telles évolutions, il est nécessaire de disposer d'approches qui mettent à jour de façon incrémentale un schéma décrivant les données RDF. Certaines des approches existantes exigent la disponibilité du jeu de données global pour définir le schéma, ou encore elles proposent de classifier les entités nouvellement insérées sans modifier le schéma existant. Dans la suite de l'état de l'art, nous présentons les algorithmes incrémentaux de clustering par densité, qui permettent de mettre à jour les clusters calculés en utilisant DBSCAN. Nous étudierons l'application de ces algorithmes pour assurer l'évolution du schéma et le garder cohérent avec les changements survenus dans la source de données.

Nous présentons dans la section 2.2 les différentes approches de découverte de schéma. Dans la section 2.3, nous présentons les approches qui découvrent les patterns structurels représentant les instances du jeu de données. Des approches connexes à la découverte de schéma sont présentées dans la section 2.4. Cette catégorisation des approches de découvertes de schéma a été inspirée de l'état de l'art présenté dans [64]. Nous discutons dans la section 2.5 l'utilisation de l'algorithme DBSCAN pour la découverte de schéma à partir des données RDF. Afin de pouvoir traiter de grands jeux de données, nous présentons dans la section 2.5.3 les versions scalables de DBSCAN qui proposent un traitement parallèle et qui peuvent être implémentées à l'aide de technologies Big Data. Les algorithmes DBSCAN incrémentaux sont présentés dans la section 2.5.4.

2.2 Approche de découverte de schéma

Dans la littérature, plusieurs travaux se sont intéressés à l'extraction de schéma à partir des données RDF. Ce schéma comporte les classes représentant les entités du jeu de données et les liens entres elles. Dans une source de données RDF, le schéma ne représente pas une contrainte sur les données, mais une description qui peut aider entre autres tâches à la compréhension d'une source de données, la formulation de requêtes ou encore à l'indexation des données. Certaines des approches proposées tentent d'analyser la structure des données afin d'identifier celles qui appartiennent à la même classe et ainsi reconstruire la description des classes et générer le schéma. Nous qualifions le schéma généré de schéma implicite car il est extrait en analysant le contenu de la source de données. Ces approches découvrent le schéma en se basant sur la similarité structurelle des instances et regroupent les données partageant des propriétés communes en utilisant des algorithmes de clustering ou des techniques d'analyse formelle de concepts. D'autres approches regroupent les chemins similaires dans un graphe RDF afin de proposer une représentation plus condensée qui facilite la compréhension du jeu de données initial.

Nous présentons dans cette section les approches de découverte de schéma à partir des sources de données

RDF. La section 2.2.1 portera sur les approches de découverte de schéma qui regroupent les instances structurellement similaires et discutera également la découverte de schéma à partir des données semi-structurées qui sont représentées par d'autres modèles, comme le modèle OEM (Object Exchange Model) [83]. Un graphe OEM ressemble dans sa représentation à un graphe RDF puisque les deux décrivent des objets et des liens entres ces objets. Cependant, contrairement au cas d'une source de données RDF, il n'est pas possible de spécifier de déclarations liées au schéma dans un graphe OEM. Des approches ont donc été proposées pour générer automatiquement de telles déclarations. La section 2.2.2 présente les approches qui regroupent les chemins similaires dans un graphe de données. Une discussion sur les approches de découverte de schéma est présentée dans la section 2.2.3. La différence entre ces deux catégories d'approches réside principalement dans le résultat qu'elles produisent. Les approches qui regroupent les instances similaires présentent un schéma comportant les classes décrivant les données et les liens entre ces classes. Tandis que les approches qui regroupent les chemins similaires visent à identifier tous les patterns de chemins qui caractérisent un graphe de données.

2.2.1 Approche de regroupement d'instances par similarité structurelle

Nous décrivons dans cette section les approches de découverte de schéma à partir de sources de données irrégulières en regroupant les entités structurellement similaires afin de déduire les classes formant le schéma. Il existe deux catégories d'approches qui découvrent le schéma en considérant la similarité structurelle des entités. La première inclut les approches qui utilisent les algorithmes de clustering pour former les groupes d'entités similaires, à partir desquels les classes du schéma seront construites. La deuxième catégorie concerne les approches qui s'appuient sur les techniques d'analyse formelle et les treillis pour la découverte de représentations structurelles de sources de données RDF. Nous présentons ces différentes approches dans la suite de la section.

Inférence de structures en utilisant le clustering hiérarchique (Christodoulou et al., 2013)

L'approche proposée dans [20, 21] est basée sur un algorithme de clustering hiérarchique ascendant afin d'extraire une représentation structurelle d'une source de donnée RDF. L'approche analyse la structure des entités afin de détecter celles qui sont potentiellement des instances d'une même classe (type). Ainsi, les entités similaires sont regroupées dans des clusters qui représentent les classes du schéma décrivant le jeu de données RDF.

Une classe est annotée par la valeur de la propriété $rdf : type$ la plus fréquente parmi l'ensemble des entités appartenant au même cluster. Dans le cas où la déclaration $rdf : type$ est absente pour toutes les entités du même cluster, la classe est annotée comme "*unknown*".

La similarité entre les entités est évaluée en utilisant l'indice de Jaccard qui reflète le nombre de propriétés communes entre deux ensembles de propriétés décrivant deux entités différentes. La similarité entre deux ensembles d'entités d_1 et d_2 est évaluée comme étant la moyenne des similarités entre chaque paire d'entités (e_i, e_j) tel que $e_i \in d_1$ et $e_j \in d_2$.

Comme l'algorithme hiérarchique forme plusieurs niveaux de clustering de façon ascendante, l'approche détermine la meilleure partition en utilisant le coefficient de silhouette (SC). Ce coefficient est calculé pour chaque entité afin d'évaluer la différence entre la similarité moyenne avec les entités du même cluster et la similarité moyenne avec les entités des autres clusters. La partition de clusters ayant le coefficient de silhouette le plus élevé est déterminée comme étant le meilleur clustering.

Les évaluations présentées dans cet article montrent que l'approche extrait un schéma d'une bonne qualité à partir d'une source de données RDF. Les auteurs le montrent en mesurant la précision, le rappel et le $FScore$ pour chaque classe extraite, les attributs et les relations entre les classes. En effet, l'algorithme hiérarchique construit progressivement les clusters afin de trouver ceux qui représentent mieux les classes des instances dans un jeu de données RDF, où on trouve des instances de la même classe décrites par des ensembles différents de propriété. L'algorithme de clustering hiérarchique permet de regrouper des entités structurellement similaires sans aucune connaissance à priori sur le nombre de clusters. Néanmoins, l'algorithme de clustering sur lequel se base l'approche est très coûteux en termes de temps d'exécution, sa complexité étant de $O(n^3)$. L'algorithme compare les paires d'entités afin de déterminer celles qui sont similaires, puis évalue les clusters calculés à chaque niveau afin de déterminer le meilleur clustering. Par conséquent, l'application de cette sur de grands jeux de données RDF est très coûteuse. Enfin, l'algorithme hiérarchique utilisé pour former les clusters n'est pas incrémental, ainsi, lorsque que le jeu de données évolue, l'algorithme devra être ré-exécuté sur la totalité du jeu de données afin d'en extraire le nouveau schéma.

Découverte de schéma en utilisant le clustering par densité, Kellou-Menouer et Kedad (2015)

Dans [65, 66, 64], les auteurs ont proposé une approche de découverte de schéma de source de données RDF qui s'appuie sur un algorithme de clustering basé sur la densité (DBSCAN) et qui détecte de façon automatique le seuil de similarité le plus adapté à un jeu de données RDF.

L'approche utilise DBSCAN afin de former des groupes d'entités similaires et construit les profils de type décrivant les entités du jeu de données. Un profil est composé d'un ensemble de propriétés. Une probabilité est attribuée à chaque propriété du profil. Elle exprime le nombre d'entités ayant cette propriété dans le cluster. La similarité entre les entités est évaluée en utilisant l'indice de Jaccard.

Le paramètre ϵ, qui représente le seuil de similarité entre deux entités, nécessaire au fonctionnement de DBSCAN est automatiquement détecté dans cette approche en évaluant la distance entre chaque entité et son plus proche voisin. Pour cela, le concept d'*entité seuil* est introduit : l'entité ayant le plus grand écart avec son plus proche voisin est l'*entité seuil*. La valeur de similarité entre l'entité seuil et son plus proche voisin est proposée comme paramètre ϵ.

Après le clustering, l'approche analyse les profils de types construits afin d'identifier les clusters ayant des propriétés communes et de détecter les entités ayant plusieurs types. Un type est attribué à une entité si elle est

décrite pas les propriétés fortes de ce type, définies comme les propriétés ayant une probabilité supérieure ou égale à un seuil fixé.

Afin de déterminer les liens sémantiques entre les types, l'approche utilise les propriétés sortantes des entités, c'est-à-dire les propriétés qui apparaissent dans des triplets dont le sujet est une entité, ainsi que les propriétés entrantes, c'est-à-dire les propriétés qui apparaissent dans des triplets dont l'objet est une entité. Un lien est créé entre deux types t_1 et t_2 si la propriété p est sortante dans t_1 et entrante dans t_2. Les liens hiérarchiques sont découverts en appliquant un algorithme de clustering hiérarchique sur les profils de types extraits.

Afin d'assurer la cohérence du schéma lors de l'évolution du jeu de données, l'approche propose de générer une entité fictive pour chaque type (cluster), composée des propriétés fortes du type. Chaque entité insérée dans le jeu de données est comparée aux entités fictives et affectée au cluster correspondant à l'entité fictive la plus proche.

Les évaluations présentées montrent que l'approche produit un schéma de bonne qualité en utilisant les métriques de précision et de rappel. En effet, l'approche identifie les profils des différents types décrivant les entités. Cependant, la complexité de DBSCAN ($O(n^2)$) limite son utilisation sur de grands jeux de données.

Détection de catégories à partir de données du web, Xi Chen et al. (2014)

L'approche proposée dans [19] utilise un algorithme de clustering hiérarchique sur les sources de données RDF afin de regrouper les entités les plus similaires à chaque itération de l'algorithme et produire une hiérarchie de clusters représentant les catégories contenues dans un jeu de données RDF. Le degré d'appartenance de chaque entité à son cluster est ensuite évalué. La figure 2.1 illustre la construction des clusters en utilisant l'algorithme hiérarchique.

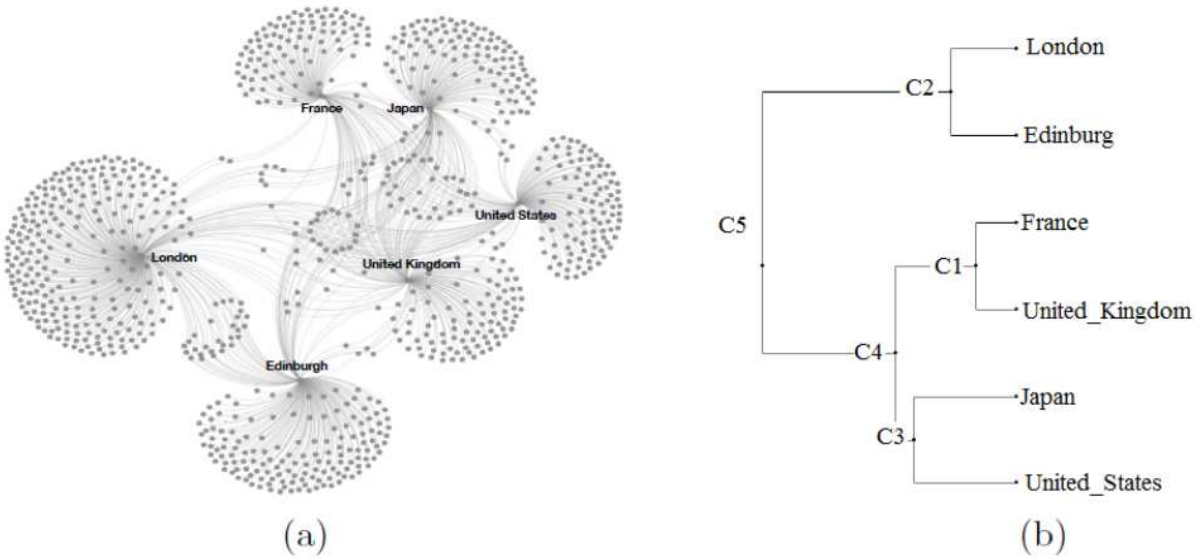


FIGURE 2.1 – Exemple de jeu de données RDF (a) et les clusters résultants de l'application de l'algorithme hiérarchique (b) [19]

L'algorithme de clustering hiérarchique se base sur une matrice de similarité. La similarité entre deux entités est évaluée en utilisant l'indice de Jaccard [56] qui calcule le nombre de propriétés communes entres les entités, deux propriétés sont considérées communes si elles possèdent la même valeur. La similarité entre deux entités est égale au nombre de paires (propriété, valeur) communes sur l'union des propriétés des deux entités.

Après la construction des clusters, le degré d'appartenance de chaque entité à son cluster est évalué. Ainsi, le centre de chaque cluster est calculé. Un centre est l'entité pour laquelle la moyenne des similarités avec les autres entités du même cluster est la plus élevée. Chaque cluster représente une catégorie dans le jeu de données RDF. Un degré d'appartenance de chaque entité à la catégorie est ensuite évalué comme le ratio de la similarité de l'entité avec les autres entités de la même catégorie et la similarité du centre avec les autres entités.

Cette approche présente une méthode qui permet l'extraction de catégories à partir d'une source de données RDF en utilisant l'algorithme de clustering hiérarchique. Cependant, la catégorie d'une entité ne représente pas son type/classe : la fonction de similarité utilisée par l'approche évalue les valeurs associées aux propriétés communes des entités et non leur similarité structurelle. Ainsi, deux entités ayant des propriétés communes ne sont pas forcément similaires, car elles doivent en plus avoir les mêmes valeurs liées à ces propriétés. Ceci diffère de la notion de type, ce dernier représentant un ensemble d'entités décrites par des propriétés similaires sans tenir compte des valeurs associées à ces propriétés. De plus, l'approche s'appuie sur le calcul d'une matrice de similarité coûteuse ce qui la rend inadaptée aux grands jeux de données RDF. Enfin, comme l'approche utilise un algorithme de clustering hiérarchique qui n'est pas incrémental, elle n'effectue pas la mise à jour des catégories découvertes si le jeu de données évolue.

Découverte de concepts formels dans le web des données, Kirchberg et al. (2012)

Dans [69], les auteurs présentent comment l'algorithme d'analyse formelle de concepts (FCA) peut être appliqué à un jeu de données RDF pour trouver les concepts qu'il contient. Un concept regroupe un ensemble d'entités décrites par les propriétés qui représentent ce concept. Les concepts sont calculés en utilisant l'approche FCA [92], permettant la visualisation des données et leurs structures en les représentant par un treillis.

La figure 2.2 représente le treillis de concepts extrait à partir d'un jeu de données $D = \{1, 2, 3, 4, 5, 6, 7, 8, 9, 10\}$ décrit par les propriétés $P = \{composite, \ even, \ odd, \ prime, \ square\}$. Le treillis inclut un concept pour chaque propriété ($composite$, $even$, odd, $prime$, $square$), ainsi que les concepts composés de plusieurs propriétés tels que $(composite, square)$ et $(composite, odd, square)$. Pour chaque concept on peut visualiser les objets qu'il décrit. Par exemple, on peut comprendre que les objets 1 et 9 sont décrits par les propriétés odd et $square$.

Cette approche regroupe les objets partageant un certain nombre de propriétés, puis construit un treillis correspondant au schéma décrivant le jeu de données. Le schéma permet l'analyse du jeu de données par instance et par propriété. L'approche ne traite pas le problème d'incrémentalité, mais il existe dans la littérature des techniques incrémentales de construction de treillis [102, 112] qui pourraient être utilisées afin d'assurer la mise à jour du schéma lorsque le jeu de données évolue.

	composite	even	odd	prime	square
1			✓		✓
2		✓		✓	
3			✓	✓	
4	✓	✓			✓
5			✓	✓	
6	✓	✓			
7			✓	✓	
8	✓	✓			
9	✓		✓		✓
10	✓	✓			

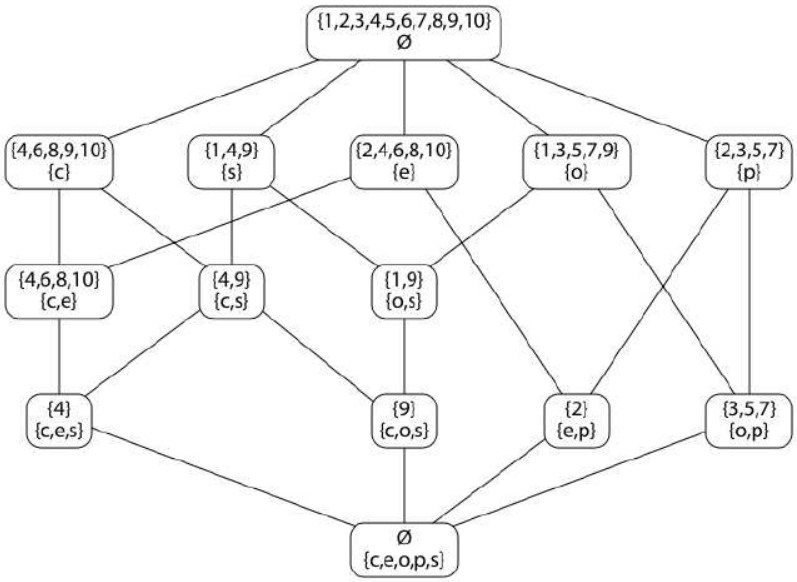


FIGURE 2.2 – Exemple de jeu de données et le treillis de concepts associé (composite (c), square (s),even (e), odd (o) et prime (p)) [69]

Nous pouvons cependant noter qu'un nœud du treillis ne représente pas forcément le type d'une entité, mais reflète le fait qu'un ensemble d'instances est décrit par un ensemble de propriétés. Ces mêmes entités peuvent avoir d'autres propriétés et peuvent ne pas être similaires. Par conséquent, elles ne représentent pas forcément des instances de la même classe. De plus, le treillis peut être très grand car il peut inclure jusqu'à $2^m \times 2^n$ concepts, où m est le nombre de propriétés et n le nombre d'objets.

Découverte de schéma sous la forme d'un index avec FCA, Brosius et Staab (2016)

Dans [14], les auteurs introduisent l'idée d'utiliser le schéma d'un jeu de données RDF comme un index afin d'optimiser le traitement des requêtes. Le schéma est extrait en utilisant l'algorithme d'analyse formelle de concepts (FCA), ce qui est justifié par le fait qu'il n'exige aucun paramètre pour son application comme dans le cas du clustering. Le treillis construit par l'algorithme FCA représente l'index sur les données.

Cet index est utilisé pour filtrer les sous-graphes de données qui composent potentiellement une réponse à une requête. Cependant, l'index représente la structure des entités et ignore les valeurs associées à chaque propriété décrivant une entité. Par conséquent, il ne peut pas être utilisé pour filtrer les sous-graphes par rapport aux valeurs renseignées dans une requête. Les auteurs proposent de réduire le nombre de sous-graphes retournés par l'index, cependant, l'optimisation prévue augmente le coût de la construction et de la navigation dans cet index. De plus, la taille du treillis correspondant à l'index rend difficile sa mise à jour à travers le temps.

Inférence de structures pour les données semi-structurées, Nestorov et al. (2001)

L'approche proposée dans [81] dérive une hiérarchie de types et de règles assignant les types aux entités dans un jeu de données semi-structurées qui peut être représenté par un graphe orienté comme c'est le cas des données OEM [83].

L'approche proposée est composée de quatre étapes principales. D'abord, les types sont identifiés. Un treillis L est créé à partir de l'alphabet composant tous les labels du jeu de données considéré D. Les labels concernés sont tous ceux décrivant les objets dans D. Ainsi, chaque nœud du graphe L représente un ensemble de labels et un nombre d'objets décrits par ces labels. La figure 2.3 représente un exemple de treillis construit à partir d'un jeu de données D.

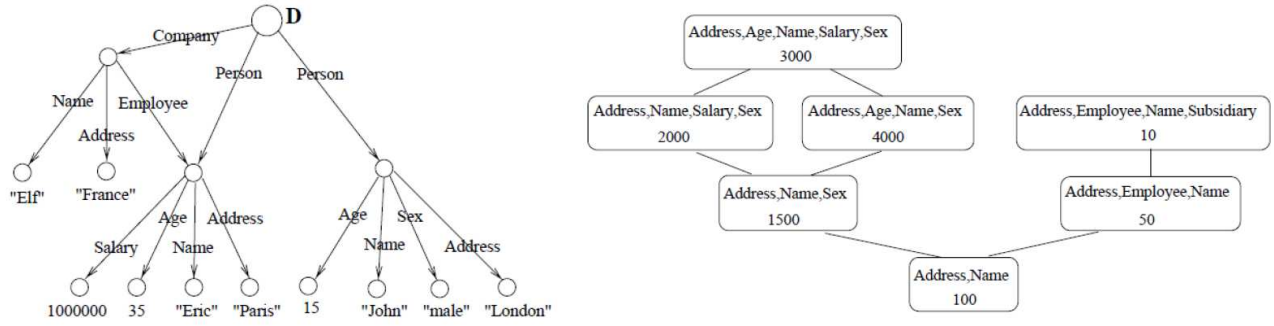


FIGURE 2.3 – Exemple de jeu de données OEM et le treillis correspondant

Après la création du treillis, l'approche identifie les types candidats en évaluant leur $saut$ $(jump)$. Le $saut$ permet d'évaluer l'homogénéité que peut avoir un ensemble d'objets avec un ensemble d'arcs. Afin d'évaluer le $saut$, on désigne $attribute(o)$ l'ensemble des labels des arcs sortants à partir de l'objet o, et $role(o)$ l'ensemble des labels des arcs entrant de l'objet o. Pour un ensemble de labels S, on définit $at(S)$ comme étant le nombre d'objets o dans D tel que $attributes(o) = S$, et $above(S)$ comme étant le nombre d'objets o dans D tel que $attribute(o) \supseteq S$. Ainsi, le $saut$ d'un ensemble de labels S est calculé comme $saut(S) = at(S)/above(S)$. Des types supplémentaires sont retrouvés en groupant les nœuds qui ne sont pas candidats et en tolérant une certaine hétérogénéité entre les ensembles de labels des objets.

La hiérarchie de types est ensuite construite. Durant cette étape, chaque type S est annoté par le label le plus fréquent des arcs entrants des objets o dans S. Cette annotation est définie comme le rôle principal du type, noté $p_{role(S)}$. Un candidat T est défini comme un type s'il n'existe pas d'autre type T' tel que $T' \subset T$. Un candidat est considéré comme un sous-type s'il n'est pas déjà un type et qu'il n'existe pas d'autre candidat S' tel que $S' \supset S$ et $p_{role(S)} = p_{role(S')}$.

Enfin, l'approche déduit les règles de typage des objets du jeu de données. Un objet est du type T s'il est décrit par les mêmes attributs que T. Si aucun type n'a les mêmes attributs qu'un objet o, les types et sous-types ayant le même p_{role} que l'objet o sont considérés afin de calculer la distance entre o et les types candidats. La distance

est calculée comme étant la proportion d'attributs différents. Un objet peut être de plusieurs types par rapport à la position de ce dernier dans la hiérarchie : si le type attribué à un objet hérite d'autres types, ces types génériques sont attribués à cet objet. Enfin, le typage des objets est évalué en calculant la précision de chaque type.

L'approche présentée extrait à partir d'un jeu de données semi-structurées une hiérarchie de types représentant les instances contenues dans le jeu de données. Elle permet le typage multiple des instances mais au sens de la hiérarchie de généralisation : une instance peut avoir plusieurs types s'il existe des liens de généralisation entre ces types. Tous les labels des arcs entrants sont considérés comme des propriétés d'un type. Ce traitement peut s'appliquer au modèle de données OEM mais pas aux données RDF. En effet, les données semi-structurées représentées par le modèle OEM ne distinguent pas les déclarations liées au schéma et les déclarations qui décrivent les entités. Les données RDF comportent des propriétés qui décrivent les instances, ainsi que des propriétés qui sont utilisées pour le raisonnement et la déduction de nouvelles informations. Par conséquent, toutes les propriétés dans une source de données RDF ne doivent pas être traitées de la même manière. Pour effectuer le typage, chaque objet est comparé à la structure d'un nœud dans le schéma, et si aucun nœud ne correspond, un nouveau nœud est créé pour représenter la structure de l'objet. Ce traitement est coûteux et ne peut pas être appliqué aux grands jeux de données.

Enfin, si les données sont très hétérogènes comme c'est le cas des données du web, le treillis représentant les objets sera de grande taille.

Extraction de schéma à partir de données semi-structurées, Nestorov et al. (1998) [80]

Dans cet article, les auteurs proposent une amélioration du travail présenté dans [81], d'une part pour rendre l'approche proposée robuste au bruit, et d'autre part afin de l'adapter aux grands jeux de données. Les auteurs proposent l'utilisation de l'algorithme de clustering $k - means$ pour la formation des clusters d'objets similaires. Cet algorithme exige comme paramètre le nombre de clusters à former, ce qui est une hypothèse forte car cela revient à connaître à priori le nombre de types existants dans un jeu de données. La méthode forme des types par regroupement, mais elle n'est pas incrémentale. Comme l'approche initiale [81], elle s'appuie sur les labels des arcs sortants afin d'annoter les types, ce qui n'est pas possible sur les données RDF où les arcs reflètent les propriétés des objets et pas nécessairement leur type.

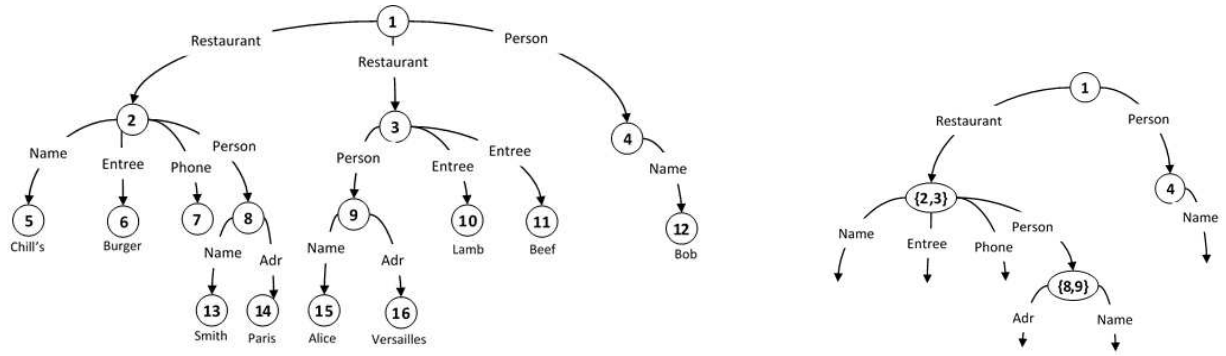
2.2.2 Approche de regroupement de chemins similaires

Nous présentons dans cette section une autre catégorie d'approches, qui visent à extraire un schéma implicite à partir d'un jeu de données irrégulières et/ou semi-structurées. Ces approches proposent de caractériser un graphe de données en regroupant les chemins similaires qui représentent des instances ayant des propriétés similaires.

Création de DataGuides pour l'optimisation des requêtes, Goldman et al. (1997)

Dans [39], les auteurs proposent un dataguide qui résume la structure d'une source de données irrégulière. Dans cette présentation, chaque chemin du graphe initial apparaît exactement une seule fois en regroupant les chemins composés de propriétés identique pour la même ressource.

Les auteurs proposent d'utiliser le dataguide pour interroger des sources de données irrégulières, mais il peut aussi être vu comme un résumé du graphe de données initial.



(a) Exemple de graphe OEM (b) Dataguide exact

FIGURE 2.4 – Exemple d'un graphe de données OEM et son dataguide

La figure 2.4 montre un exemple de graphe de données OEM (Object Exchange Model) et le dataguide correspondant. Dans une source de données irrégulière, les données de même type n'ont pas nécessairement la même structure. Par exemple, les instances 2 et 3 représentent toutes les deux des restaurants, mais elles sont décrites par des propriétés différentes, l'instance 2 possède les propriétés $Phone$ et $Owner$ qui sont absentes pour l'instance 3.

Afin de construire le dataguide, les chemins similaires dans le graphe initial sont fusionnés, ce qui permet de regrouper des instances similaires dans un même nœud, par exemple les instances 2 et 3 montrées dans la figure 2.4a sont regroupées dans le nœud $2, 3$ comme le montre la figure 2.4b. Un chemin représente une séquence de propriétés du graphe atteignant une feuille donnée. Dans un dataguide, des instances regroupées dans un même nœud n'ont pas forcément les propriétés décrivant le nœud, par exemple, l'instance 3 est regroupée avec l'instance 2 dans le nœud $2, 3$, alors que l'instance 2 ne possède pas la propriété $Phone$. De plus, des ressources de même type peuvent être représentées par plusieurs nœuds dans le dataguide si elles ne sont pas atteignables par le même chemin, par exemple, les ressources 4, 8 et 9 représentent toutes des ressources de type $Person$, mais on observe que 8 et 9 sont regroupées alors que ce n'est pas le cas de l'instance 4.

Le dataguide proposé par cette approche est un graphe concis et exhaustif représentant le graphe de données initial en termes de chemins. Autrement dit, on retrouve dans le dataguide exactement une seule fois tous les chemins du graphe initial. Un tel dataguide représente un résumé permettant la navigation dans un graphe de données afin de répondre à une requête. Contrairement à un schéma, le dataguide est construit en fusionnant les

chemins qui partent de la même donnée. Ceci permet de regrouper des instances similaires. De plus, le dataguide peut être plus grand que le graphe de données initial car les instances du graphe sont irrégulières et peuvent être décrites par des ensembles de propriétés très hétérogènes. Des nœuds de types différents peuvent être regroupés s'ils possèdent un chemin commun, par exemple, le nœud $\{2, 3, 4\}$ qui comporte les nœuds $\{2, 3\}$ de type $student$ et $\{4\}$ de type $professor$. Enfin, le coût de construction et de maintien du dataguide peut être élevé. En effet, la construction d'un automate fini déterministe est une opération coûteuse, comme expliqué dans [79].

Dataguide approximatif, Wang et al. (2000)

Pour pallier aux inconvénients d'un dataguide exact, les auteurs de [109] proposent un dataguide approximatif qui utilise une méthode de clustering hiérarchique incrémentale (COBWEB [33]) afin de regrouper les nœuds du graphe ayant des arcs entrants et sortants similaires. L'approche construit le dataguide approximatif de façon évolutive en analysant les arcs sortants et entrants de chaque nœud. Tout d'abord, le schéma approximatif comporte le nœud racine du graphe initial. Ensuite, le graphe initial est parcouru et chaque nœud est ajouté au graphe approximatif, en le fusionnant avec le nœud qui présente la plus grande valeur de la fonction d'utilité, ou en créant un nouveau nœud. Après chaque insertion, un arc est ajouté entre le nouveau nœud et les nœuds existant s'il existe un lien entre eux dans le graphe initial.

La fonction d'utilité U est définie comme suit : $U = \frac{1}{K}\sum_{i=1}^{K} E(v_i)$, v_i étant l'ensemble des nœuds du graphe de données. La fonction E reflète la probabilité qu'un arc l décrive un ensemble de nœuds v_i. Elle est définie comme suit : $E(v_i) = \frac{1}{\overline{v_i}}\sum_{l \in \overline{v_i}} P(l|v_i)P(v_i|l)$.

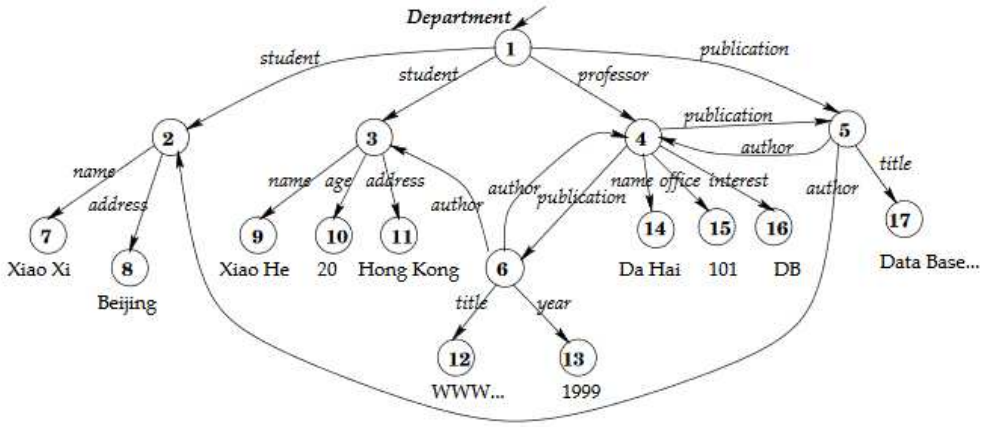
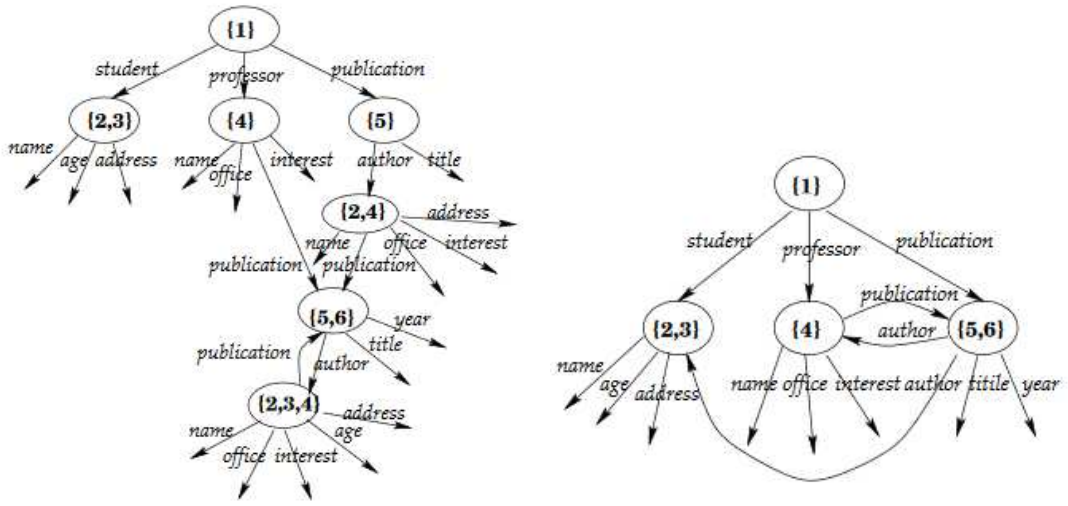


FIGURE 2.5 – Exemple de graphe OEM

Un dataguide approximatif est plus concis qu'un dataguide exact comme le montre la figure 2.6, ce qui optimise le temps de parcours du graphe et améliore le traitement des requêtes. Cependant, il peut contenir des chemins non existants dans le graphe initial, et par conséquent, il peut induire des faux positifs lorsqu'il est interrogé pour déterminer si une source possède bien une réponse à une requête.



(a) Dataguide exact (b) Dataguide approximatif

FIGURE 2.6 – Dataguide exact vs. Dataguide approximatif du graphe présenté dans la figure 2.5

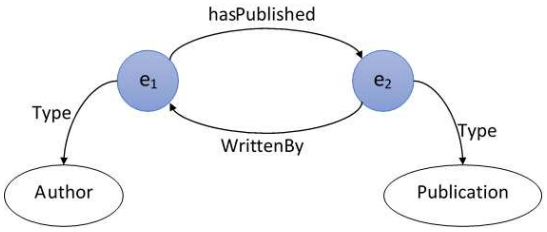


FIGURE 2.7 – Exemple de regroupement d'entités en considérant les propriétés sortantes et entrantes

Les nœuds qui composent le dataguide approximatif peuvent être considérés comme des classes qui représentent les instances du jeu de données. Cependant, l'approche a pour but de créer un graphe compact comportant un nombre minimum de classes (nœuds) alors que pour la découverte du schéma, le plus important est la qualité des classes découvertes et non leur nombre. De plus, l'approche considère de la même façon les propriétés entrantes et sortantes, par conséquent, des instances de classes différentes peuvent être regroupées si elles partagent des propriétés sortantes ou entrantes. L'exemple de la figure 2.7 montre deux entités e_1 et e_2 qui seront groupées car elles partagent les mêmes propriétés sortantes/entrantes, alors qu'elles ont des types différents. Enfin, l'algorithme de clustering COBWEB utilisé pour former le graphe approximatif peut donner des résultats différents selon l'ordre de parcours parcours des nœuds du graphe de données.

Bisimulation de graphes RDF à grande échelle, Schätzle et al. (2013)

L'approche [96] propose de réduire la taille du graphe initial afin de faciliter sa compréhension. L'approche utilise des techniques de bisimulation [77] afin de regrouper les nœuds similaires en considérant leurs arcs sortants. Deux nœuds sont bisimilaires s'ils possèdent les mêmes arcs sortants et si cette condition est également vraie pour leurs successeurs.

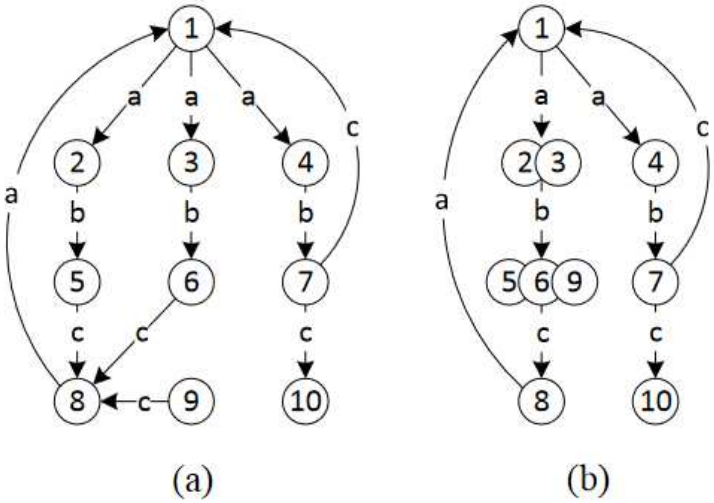


FIGURE 2.8 – Graphe de données RDF (a) et son graphe réduit par bisimulation (b) [96]

La figure 2.8 montre un graphe RDF et sa représentation réduite en utilisant le principe de bisimulation. Par exemple, les nœuds 2 et 3 sont bisimilaires car ils possèdent le même arc sortant b, et leurs successeurs 5 et 6 partagent le même arc sortant c. Le nœud 4 n'est pas regroupé avec le 2 et 3 car son successeur possède deux arcs sortants c.

Dans le graphe produit par cette approche, les nœuds du graphe réduit représentent des instances décrites par les mêmes arcs sortants, tandis que dans un jeu de données RDF, les instances de la même classe ne sont pas nécessairement décrites par les mêmes propriétés. Par conséquent, les nœuds du graphe réduit ne représentent pas les classes du jeu de données.

2.2.3 Bilan des approches de découverte de schéma

Les approches présentées dans cette section extraient le schéma implicite d'une source de données en analysant la structure des instances afin de regrouper celles qui sont similaires, sans faire aucune hypothèse sur l'existence de déclarations sur le schéma dans le jeu de données. Certaines de ces approches regroupent les entités similaires afin de découvrir les classes composant le schéma [20, 21, 65, 66, 19, 69, 14, 81, 80], alors que d'autres approches regroupent les chemins du graphe de données et tentent de réduire la taille du graphe initial en construisant un résumé de données [39, 96, 109]. Ces différentes approches sont synthétisées dans le tableau 2.1 qui présente la nature de la méthode utilisée par chaque approche, la mesure de similarité utilisée, la capacité de l'approche à traiter les grands jeux de données, l'incrémentalité de l'approche et la gestion du schéma lorsque les données évoluent, l'utilisation des déclarations sur le schéma tel que $rdf{:}type$, et le résultat produit par chaque approche.

TABLE 2.1 – Synthèse des approches de découverte de schéma

Approches	Algorithme utilisé	Formule de similarité	Scalable	Incrémental	Utilisation des déclarations schéma	Résultat
Clustering hiérarchique [20, 21]	Algorithme de clustering hiérarchique	Jaccard	Non	Non	Déclarations $rdf{:}type$ pour annoter les classes	Types, liens hiérarchiques et sémantiques entre les types
Extraction par DBSCAN [65, 66]	DBSCAN	Jaccard	Non	Oui, par classification des nouvelles entités	Non	Typage multiple, liens hiérarchiques et sémantiques
Détection de catégories [19]	Algorithme de clustering hiérarchique	Jaccard en considérant la valeur des propriétés	Non	Non	Non	Hiérarchie de types
Découverte de concepts [69]	Algorithme d'analyse de concepts formels (FCA)	/	Non	Non	Non	Treillis de concepts
Indexé un graph RDF avec FCA [14]	Algorithme d'analyse de concepts formels (FCA)	/	Non	Non	Non	Index sur un graphe RDF
Inférence de structures [81, 80]	Construction de treillis	Utilise le concept de $saut$ qui évalue l'homogénéité d'un ensemble d'arc	Oui	Non	Labels des arcs entrants pour annoter les types	Hiérarchie de types sous forme de treillis
L'approche bisimulation [96]	Technique de bisimulation	Similarité exacte	Non	Non	Non	Résumé du graphe initial
Dataguide approx [109]	COBWEB	Fonction d'utilité	Oui	Oui	Non	Dataguide
Dataguide [39]	Groupement de chemins similaires	Similarité exacte	Non	Non	Non	Dataguide

Les approches qui regroupent les entités similaires identifient les classes composant le schéma décrivant un jeu de données RDF. Ces approches comparent les structures décrivant les entités afin d'évaluer leurs similarités. Ces structures contiennent l'ensemble des propriétés des entités sauf pour [19] qui considère également la valeur des propriétés partagées entre les entités. Dans ce but, certaines de ces approches utilisent des algorithmes de clustering pour former des groupes d'entités similaires représentant les classes du schéma [21, 20, 65, 66]. Tandis que d'autres approches s'appuient sur l'analyse formelle afin de construire un treillis de concepts [69, 14, 81, 80]. La limite majeure de ces approches est la complexité des algorithmes utilisés afin de découvrir et regrouper les entités similaires. Ces algorithmes sont très coûteux, ce qui les rend inadaptés aux grands jeux de données RDF.

De plus, ces algorithmes ne sont pas incrémentaux et nécessitent la disponibilité de la totalité du jeu de données pour procéder à la découverte du schéma. La mise à jour d'un schéma implicite lors d'une évolution d'un jeu de données nécessiterait donc la ré-exécution de l'algorithme de découverte de schéma sur la source composée des anciennes et des nouvelles données. Pour pallier ce problème, l'approche proposée dans [66] classe les nouvelles entités dans le cluster comportant l'entité fictive la plus proche. Cependant, l'ajout d'une entité peut impliquer la fusion de classes existantes ou la création de nouvelles classes, cas qui ne sont pas couverts par cette approche, dans laquelle le schéma découvert n'est pas remis en question.

Les approches de la seconde catégorie regroupent les chemins similaires dans un graphe de données afin de générer un résumé sous forme d'un dataguide [39, 96] ou un dataguide approximatif [109].

Le problème de ces approches est la taille du résumé qui doit représenter tous les chemins possibles contenus dans le graphe initial. De plus, le coût de construction et de maintenance du dataguide est élevé car ce processus consiste à calculer un automate fini et déterministe du graphe de données initial. Enfin, les nœuds du graphe résumé regroupent des instances décrites exactement par le même ensemble de propriétés, ce qui les rend moins bien adaptées au cas d'une source de données RDF, où les instances d'une même classe peuvent être décrites par des ensembles de propriétés hétérogènes. L'approche décrite dans [109] propose un processus scalable qui construit de façon évolutive un dataguide approximatif en utilisant l'algorithme COBWEB. Le dataguide approximatif est plus concis comparé au dataguide exact, ce qui permet une optimisation des requêtes exécutées sur le dataguide. Cependant, l'utilisation de l'algorithme COBWEB peut produire un résultat différent pour des exécutions distinctes, résultat qui dépend de l'ordre de parcours du graphe initial.

L'étude des approches d'extraction de schéma pour des données RDF montre qu'aucune approche existante ne prend en compte les triplets implicites, qui peuvent être déduits à partir des triplets existants par inférence. La prise en compte de ces triplets est importante car ils font partie intégrante de la sémantique de la source de données. Ceci permettra de compléter la source de données et ainsi d'améliorer la qualité du schéma, qui sera plus représentatif des entités composant une source de données.

Parmi les catégories d'approches présentées dans cette section, nous constatons que celles qui procèdent par analyse de la structure des instances pour les regrouper par similarité sont les plus adaptées à l'extraction de schéma. Les approches qui regroupent les chemins similaires sont plus adaptées à des fins d'optimisation de requêtes sur les graphes de données car elles fournissent un plan d'accès à ces données.

L'analyse des approches de découverte de schéma fait apparaître un certain nombre de problèmes ouverts, qui sont encore peu adressés par les travaux existants. C'est le cas de la scalabilité. En effet, l'utilisation des algorithmes de clustering pour la découverte de schéma est coûteuse et ne permet pas un traitement scalable des sources de données massives. Un autre problème ouvert est l'incrémentalité des approches de découverte de schéma. L'évolution d'un jeu de données peut rendre le schéma associé incohérent avec les entités. Afin d'actualiser le schéma, il est nécessaire de ré-exécuter le processus de découverte sur la source de données après chaque évolution. Enfin, les solutions de découverte de schéma existantes considèrent uniquement les propriétés explicites dans la découverte de schéma et ignorent les informations qui peuvent être déduits par inférence à partir d'une source de données RDF.

2.3 Approches de découverte de patterns

En plus des approches de découverte de schéma, il existe des approches dont le but est de découvrir des patterns structurels à partir de sources de données irrégulières. Les approches de découvertes de schéma proposent de découvrir les classes représentant les entités du jeu de données, alors que les approches de découverte de patterns tentent d'identifier les structures qui décrivent les instances : un pattern est un ensemble de propriétés qui représente une version structurelle des entités et nous informe qu'il existe au moins une entité décrite par ce pattern. Les patterns sont utiles par exemple lors de l'interrogation d'une source de données. Si une requête porte sur un ensemble de propriétés et qu'aucun pattern ne correspond à cet ensemble de propriétés, cela signifie que cette requête ne possède pas de réponse dans la source considérée.

À l'inverse d'un schéma, les patterns ne représentent pas forcément les classes/types qui décrivent les instances du jeu de données. Un pattern structurel représente un ensemble de propriétés décrivant une ou plusieurs instances du jeu de données. Des instances d'une même classe peuvent être décrites par un ensemble hétérogène de propriétés, ainsi, une classe peut être représentée par plusieurs patterns. Certaines approches considèrent les patterns structurels comme des versions possibles d'une classe du schéma [94, 67, 68].

Un pattern structurel peut être exact ou approximatif. Un pattern exact décrit un ensemble d'instances ayant des ensembles de propriétés identiques. La figure 2.9 illustre des exemples de patterns exacts (figure 2.9.a) et approximatifs (figure 2.9.b) associés au graphe de données RDF. Dans cet exemple, les entités sont représentées par trois patterns exacts, chacun d'eux associé à des entités décrites par le même ensemble de propriétés, comme les entités e_3 et e_4. Les entités qui sont décrites par des ensembles de propriétés différents sont représentées par des patterns différents même si elles ont le même type, comme c'est le cas des entités e_1 et e_2. Un pattern approximatif est un ensemble de propriétés décrivant des instances, où chaque instance est décrite par un sous ensemble des propriétés composant le pattern. Un pattern approximatif décrivant un ensemble d'instances comporte des propriétés dites exactes, qui décrivent toutes les instances de cet ensemble, et des propriétés dites optionnelles qui décrivent une partie des instances de cet ensemble. Dans l'exemple de la figure 2.9, les entités e_1 et e_2 sont représentées par le même pattern pt_1 et les propriétés qui ne décrivent pas toutes les entités associées au pattern sont dites optionnelles et sont marquées par (?).

Cette section est organisée comme suit. Nous présentons tout d'abord les approches de découverte de patterns exacts d'un jeu de données (2.3.1). Nous introduisons ensuite les approches qui proposent la découverte de patterns approximatifs (2.3.2). Enfin, nous présentons une discussion sur les approches de découverte de patterns structurels (2.3.3).

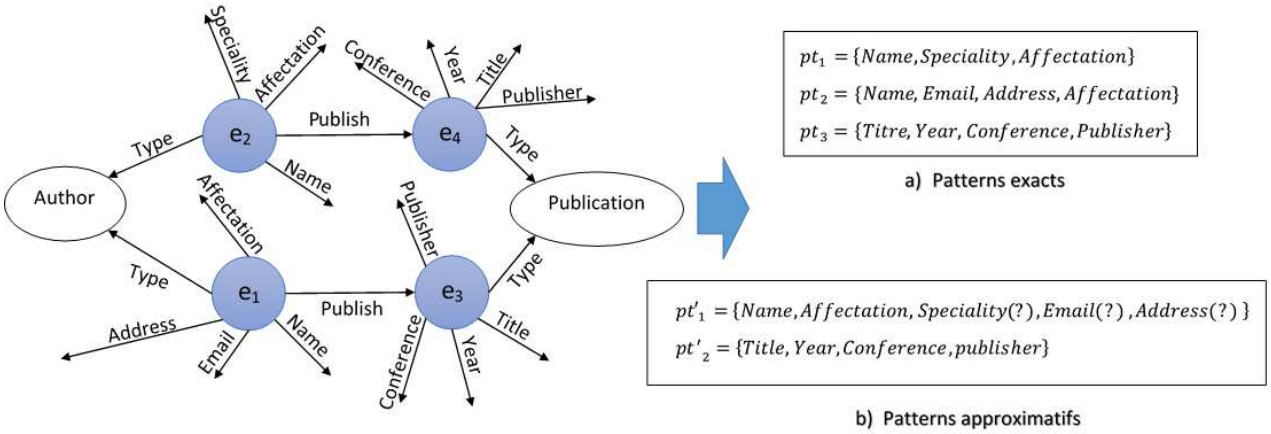


FIGURE 2.9 – Graphe de données RDF et les patterns structurels qui représentent ses instances

2.3.1 Approche de découverte de patterns exacts

Dans cette section, nous présentons les approches qui proposent de découvrir les patterns structurels exacts à partir d'un jeu de données comportant des instances irrégulières. Un pattern exact est représenté par un ensemble de propriétés qui décrivent au moins une instance du jeu de données.

FreGraPaD : détection de patterns fréquents pour les flux de données RDF, Belghaouti et al. (2016)

FreGraPad [10] est une approche d'extraction qui détecte et comptabilise les patterns fréquents à partir de flux de données RDF en une seule passe afin de produire une représentation compressée du jeu de données.

FreGraPad extrait les patterns en se basant sur la construction de trois structures de données : (1) la construction de vecteurs binaires (*bit vector*), utilisés pour détecter les patterns de graphe et optimiser l'espace mémoire, (2) la construction de la table de hachage de prédicats (*PHT*), utilisée pour détecter et sauvegarder les prédicats des patterns présents dans le flux de données, (3) la construction de la table de hachage de graphe (*GHT*), qui détecte et enregistre les patterns d'un graphe RDF.

Pour chaque instance e dans un flux de données, FreGraPad vérifie les prédicats (propriétés) décrivant e et les bits du vecteur binaire correspondant à ces prédicats sont mis à 1, les indices correspondant aux autres prédicats sont mis à 0.

Comme présenté dans la figure 2.10, le vecteur binaire de l'instance du graphe 1 est 1111 qui correspond aux prédicats a, b, c et d associés respectivement aux indices $0, 1, 2$ et 3 dans le tableau PHT.

Après la construction des patterns représentés par les vecteurs binaires, l'algorithme vérifie l'existence de ces vecteurs dans le tableau GHT. Si un pattern existe déjà, il voit sa fréquence augmentée, dans le cas contraire, un nouveau pattern de graphe est détecté et ajouté au tableau GHT avec une fréquence initiale de 1. Dans l'exemple de la figure 2.10, le pattern (15,1) est inséré dans le tableau GHT et 1 représente la fréquence de ce pattern.

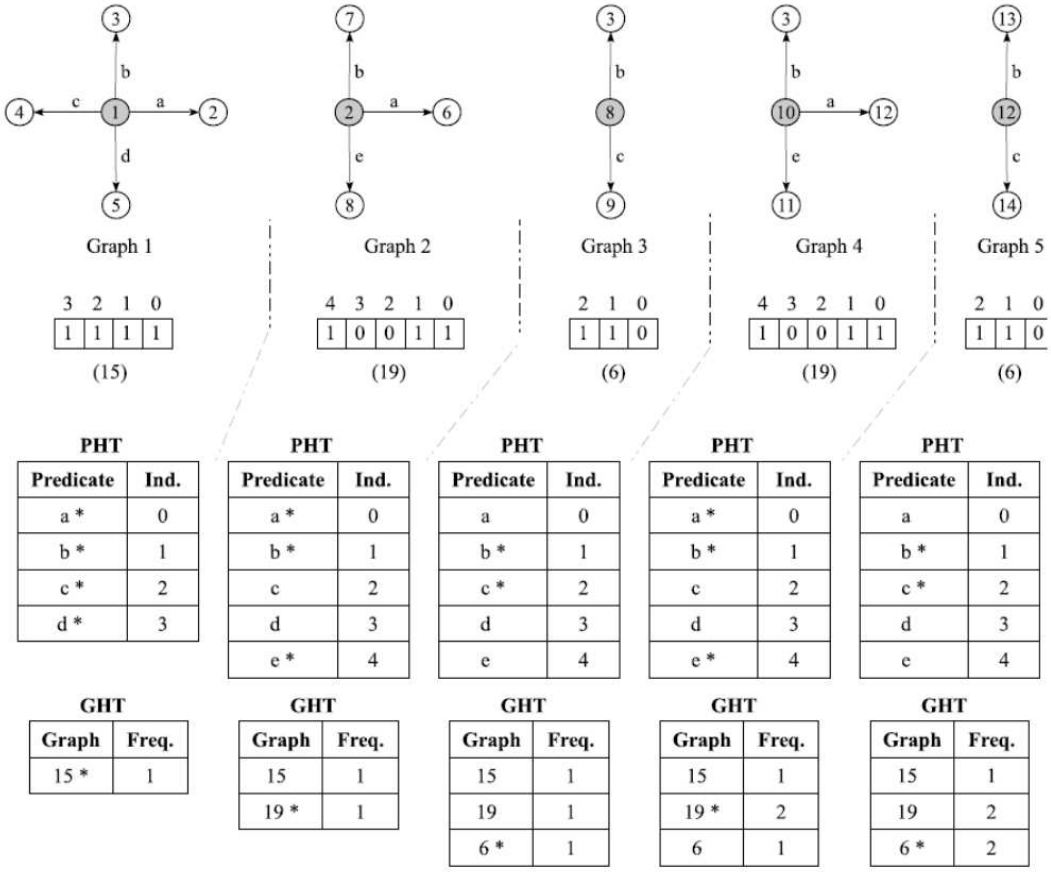


FIGURE 2.10 – Exemple de flux de données RDF et les patterns détectés en utilisant l'algorithme FreGraPad [10]

FreGraPad propose une représentation compressée du jeu de données initial. Cependant, cette compression ne représente pas un schéma pour cette source. En effet, les instances dans un jeu de données RDF sont très hétérogènes, et des instances de même type peuvent être décrites par des ensembles de prédicats différents, donc des patterns différents. Enfin, lorsque la source de données contient des instances très hétérogènes, le nombre de patterns résultant sera très grand ce qui rendra la représentation compressée difficile à interpréter.

Inférence des versions de classes à partir de jeux de données NoSql, Ruiz et al. (2015)

Cette approche [94] découvre les différentes versions des classes d'un schéma décrivant un jeu de données NoSql exprimé en Json. Une version d'une classe du schéma représente un ensemble de propriétés décrivant une ou plusieurs entités du jeu de données. L'approche découvre les versions des types déclarés dans la source de données RDF et n'identifie pas de nouveaux types. Elle considère les structures décrivant les entités de type T comme étant des versions de T.

Afin de découvrir les versions d'un schéma, l'approche regroupe les entités ayant la même déclaration de type. Ensuite, les différentes versions d'un type sont découvertes en parcourant les entités du même type, et chaque structure d'entité distincte sera considérée comme une nouvelle version du type. En plus de découvrir les versions

d'un type, un méta-modèle d'un schéma NoSQL est généré avec le type de base des valeurs de chaque propriété (chaîne de caractères, entier, etc.).

Cette approche est implémentée en utilisant la plateforme Map/Reduce et peut être appliquée sur des grands jeux de données. Cependant, elle s'appuie sur les déclarations de types afin de regrouper les entités similaires et ne peut pas être utilisée sur les sources de données où ces déclarations sont absentes. De plus, l'approche ne considère pas les entités ayant plusieurs types.

Résumé de graphe pour l'assistance à la formulation de requêtes, Campinas et al. (2012)

Dans [15], les auteurs proposent une approche qui assiste l'utilisateur lors de la formulation de requête SPARQL sur différentes sources de données hétérogènes, même dans le cas où l'utilisateur ignore la structure ou le vocabulaire décrivant ces données.

L'approche forme un résumé de graphe RDF en se basant sur le concept de collection de nœuds, défini comme un ensemble de nœuds partageant des caractéristiques communes. Deux types de caractéristiques sont considérées : la première, désignée par "$class-based$", groupe les entités ayant la même déclaration type, la deuxième, désignée par "$attribute-based$", groupe les entités décrites par les mêmes propriétés.

La figure 2.11 montre l'ensemble des entités et les liens entre ces dernières. Le graphe *Entity Layer* représente les jeux de données RDF et les liens entres les entités incluses dans ces jeux de données. Dans le graphe *Node Collection Layer*, chaque sommets représentent un ensemble d'entités du même type et les arcs représentent des liens entre elles. On remarque le nœud $Article$ produit suite au regroupement des entités $A1$ et $A2$ qui partagent le même type. En se basant sur ce graphe résumé, un système d'assistance à la formulation de requêtes SPARQL est proposé à l'utilisateur. Lors de la formulation de requêtes, le système analyse les différentes options qui peuvent compléter une requête et les propose à l'utilisateur. Cependant, la méthode qui résume un graphe RDF s'appuie sur les déclarations de type, qui ne sont pas toujours disponibles. En l'absence de ces déclarations, l'approche regroupe les entités ayant exactement le même ensemble de propriétés. Ceci peut poser problème pour une source de données RDF, qui peut contenir des entités appartenant à la même classe mais qui sont décrites par des ensembles de propriétés différents.

SchemaDecrypt : Découverte en ligne des versions de types, Kellou-Menouer et Kedad (2017)

SchemaDecrypt [67, 68] découvre les différents patterns structurels à partir d'une source de données distante, pour renseigner la co-occurrence entre les propriétés décrivant un type donné. Cette approche considère les patterns comme étant des versions structurelles des types contenus dans la source de données. Une version v_i d'un type t est un ensemble de propriétés qui décrivent certaines instances de t ayant exactement les mêmes propriétés. L'ensemble des versions V d'un type t représentent toutes les structures possibles des instances du type t.

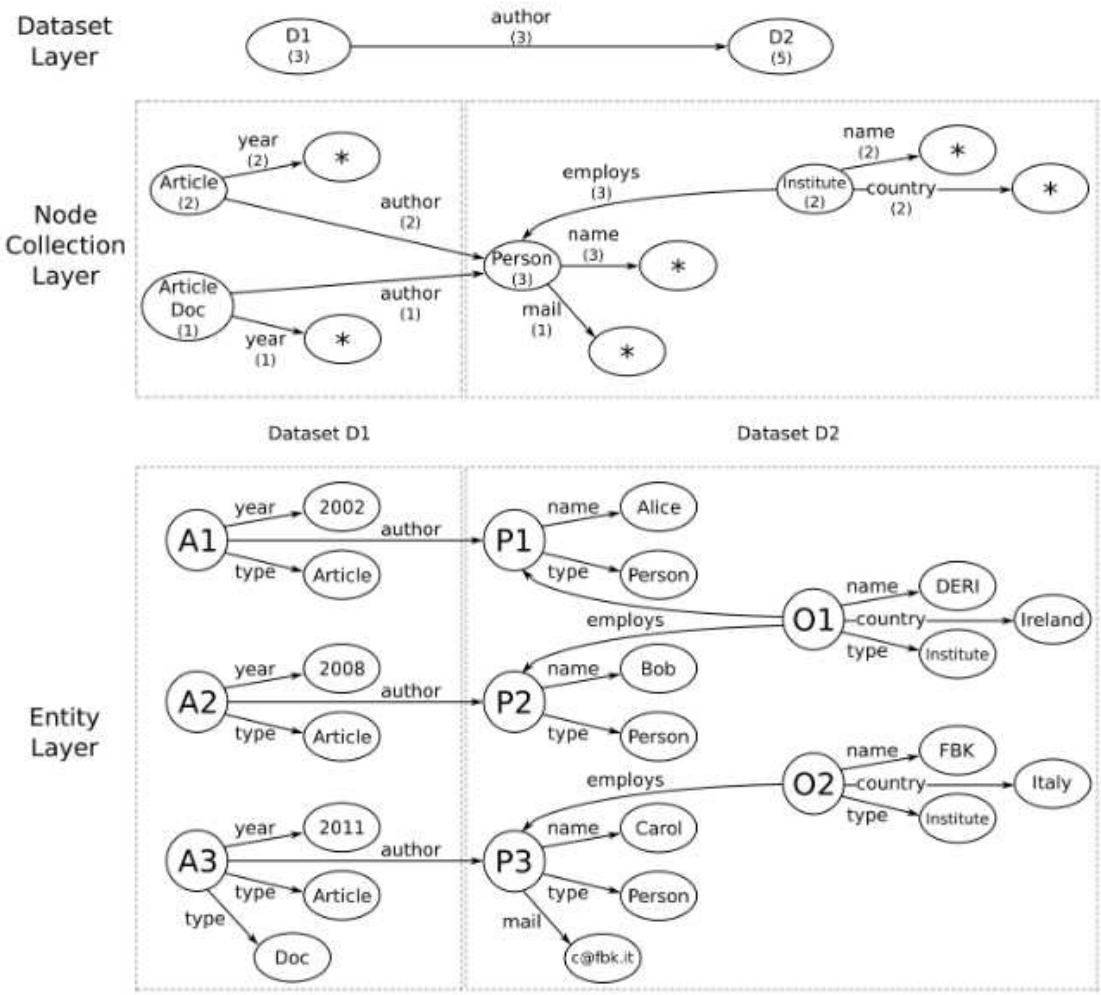


FIGURE 2.11 – Exemple de jeux de données RDF et le résumé correspondant [15]

L'approche découvre les versions de types en interrogeant une source de données distante avec des requêtes SPARQL et calcule le nombre d'occurrences de chaque version. Le principe général est de définir toutes les combinaisons de propriétés qui décrivent un type dans la source de données RDF. Les combinaisons de propriétés représentant un type $t_1 = \{p_1, p_2, \cdots, p_n\}$ correspondent à tous les ensembles de propriétés dérivés à partir de ces propriétés ($\{\}, \{p_1\}, \cdots, \{p_1, p_2\}, \cdots, \{p_1, p_2, \cdots, p_n\}, \cdots$). Ces différents ensembles sont considérés comme des versions candidates, et la source de données est ensuite interrogée afin de vérifier l'existence d'entités correspondant aux versions candidates. Les versions retenues sont celles pour lesquelles il existe au moins une instance conforme à cette version dans le jeu de données, et le nombre d'instances représente le nombre d'occurrences de la version.

L'objectif de l'approche est de trouver les versions d'un type d'une source de données distante qui impose des restrictions sur l'accès aux données comme le nombre de requêtes, le temps de traitement d'une requête ou encore le nombre de réponses à retourner. Comme l'approche interroge une source de données distante afin de construire

les versions des types, elle utilise un nombre minimum de propriétés dans une requête afin de réduire le temps d'exécution, et réduit le nombre de requêtes envoyées au serveur pour ne pas être restreint par le nombre de requêtes autorisées.

Pour garantir la découverte de versions en dépit des restrictions imposées par le serveur de la source de données, l'approche s'appuie sur la construction d'un profil probabiliste qui guide l'exploration des versions candidates en testant d'abord les versions les plus probables. Le profil probabiliste d'un type est formé par l'ensemble des propriétés associées au type avec leur probabilité qui représente pour chaque propriété p la probabilité qu'une instance soit décrite par p. Le profil d'un type est construit en interrogeant la source sur chaque propriété associée au type. Ensuite, l'approche réduit le nombre de combinaisons lors de la formation des versions candidates en diminuant le nombre de propriétés à tester, ceci en considérant les propriétés co-occurentes, i.e. qui apparaissent toujours ensemble, comme étant une seule propriété. Cette proposition réduit le nombre de propriétés à tester et donc le nombre de combinaisons et le nombre de versions candidates à explorer. De plus, l'approche découvre des règles de co-occurrence entre les propriétés associées à un type pour réduire le nombre de requêtes et minimiser le nombre de propriétés à considérer lors de la construction des versions candidates. Ces règles sont des règles d'inclusion, qui indiquent que toutes les instances ayant la propriété p_i ont aussi la propriété p_j, et des règles d'exclusion, qui indiquent que les propriétés p_i et p_j ne décrivent jamais la même instance. Ces règles permettent d'éliminer à priori des versions candidates, sans envoyer de requête à la source. Par exemple, il n'est pas nécessaire de vérifier les versions qui comportent des propriétés impliquées dans une règle d'exclusion.

Suivant les règles proposées, les différentes versions candidates de type sont générées progressivement à partir de l'ensemble réduit de propriétés jusqu'à trouver toutes les versions du type de la source de données.

Tout comme les approches de cette section, SchemaDecrypt n'a pas pour but la découverte de type/classe implicite décrivant les entités du jeu de données, mais découvre les versions des types déjà définis dans la source de données. En l'absence des déclarations de type, les versions de l'ensemble du jeu de données sont identifiées.

2.3.2 Approches de découverte de patterns approximatifs

Dans cette section, nous présentons les approches qui découvrent les patterns structurels approximatifs d'un jeu de données. Contrairement aux approches de la section précédente, un pattern approximatif est un ensemble de propriétés qui décrit un ensemble d'instances du même type. Il comporte des propriétés obligatoires et des propriétés optionnelles.

Inférence de schéma pour des jeux de données JSON massifs (Baazizi et al., 2017)

L'approche proposée dans [8, 9] infère un schéma qui offre une vue globale des structures des entités d'un jeu de données JSON. Il comprend également les types primitifs (entier, chaîne de caractères, etc.) des valeurs

des propriétés. Chaque structure est décrite par une expression régulière, avec la spécification du type de chaque propriété.

Afin de construire l'expression régulière décrivant les entités d'un jeu de données, l'approche procède en deux étapes : l'inférence des types primitifs des propriétés, puis la fusion de ces types. Les types primitifs sont les types de données utilisés dans les langages de programmation comme les types $String$, $Number$, $Bool$ etc.

Dans l'étape d'inférence, on considère qu'un type primitif est le type des valeurs de chaque propriétés (entier, chaîne de caractères, etc.) et non la déclaration de type d'une entité. Ainsi, cette étape consiste à extraire à partir de chaque entité un enregistrement qui représente les propriétés de l'entité et le type de valeur de chaque propriété. Par exemple, les entités $Person1\{id : 1, age : 14, admin : false, name : "John\ Smith", phone : 3132437\}$, $Person2\{id : 2, name : "Edmond\ Dantes", email : "ed@mc.com", admin : true\}$ et $Office\{id : 3, number : 3, address : "4\ rue\ armengaud"\}$ sont respectivement représentées par les enregistrements $Person1 = \{id : Number, age : Number, admin : Bool, name : String, phone : Number\}$, $Person2 = \{id : Number, name : String, email : String, admin : Bool\}$ et $Office = \{id : Number, number : Number, address : String\}$. À la fin de cette première étape, les enregistrements doublons, qui sont les enregistrements ayant la même structure sont éliminés.

Dans la seconde étape, l'approche fusionne itérativement les types produits lors de la première étape. La fusion se fait de façon distribuée en se reposant sur un opérateur de fusion commutatif et associatif. La fusion de deux enregistrements T_1 et T_2 produit un super-type structurel $T_{1,2}$. L'idée principale est de ne représenter qu'une fois les descriptions communes des entités en les fusionnant, tout en préservant les parties différentes qui sont propres à chaque type en les désignant par un point d'interrogation. Par exemple, la fusion des enregistrements $Person1$ et $Person2$ produit l'enregistrement $Person_{1,2} = \{id : Number, (age : Number)?, admin : Bool, name : String, (phone : Number)?, (email : String)?\}$.

Dans cette approche, un schéma est représenté par une expression régulière décrivant les différentes structures possibles des données ainsi que les types des valeurs des propriétés. Une propriété dans une expression régulière peut être obligatoire si elle est définie pour toutes les entités de la structure ou optionnelle si elle n'est pas définie pour toutes les entités et elle est dans ce cas annotée par un " ?". L'approche ne découvre pas de nouveaux types de données si ces derniers ne sont pas fournis dans le jeu de données. Cette approche a été implémentée en utilisant la plateforme map/Reduce ce qui la rend applicable sur les grands jeux de données. Cependant, elle s'appuie sur les déclarations de type afin de regrouper les entités et la qualité de l'expression régulière serait considérablement réduite lorsque ces déclarations sont absentes. En effet, l'approche ne fera dans ce cas de figure que des unions d'enregistrements et il résultera une expression régulière de la taille du jeu de données. Enfin, si les entités du jeu de données sont très hétérogènes, le schéma résultant sera complexe et difficile à lire.

Résumé de graphe RDF pour l'exécution de requêtes, Cebiric et al. (2015)

Dans [16], les auteurs proposent une méthode qui résume un graphe RDF afin de produire une représentation plus concise. Ce résumé peut être utilisé afin de donner une vue globale sur un jeu de données RDF, ou pour optimiser l'exécution de requêtes sur un graphe RDF.

L'approche exploite les déclarations $rdfs : domain, rdf : type, rdfs : range$ afin de regrouper dans des nœuds communs les propriétés partageant ces déclarations. Ces dernières ne sont pas forcément explicites et sont aussi déduites en utilisant la saturation de graphe RDF. Nous décrivons dans ce qui suit les étapes composant l'approche.

Tout d'abord, le graphe RDF est saturé pour extraire les déclarations implicites en utilisant des règles d'inférence RDFS. Par exemple, de nouvelles déclarations $rdf{:}type$ peuvent être déduites à partir des déclarations $rdfs{:}subclassOf$. Ensuite, les nœuds du graphe résumé Gs sont formés sur la base du domaine ou du co-domaine des propriétés, non pas par rapport aux déclarations $rdfs{:}domain/range$, mais selon le sujet et l'objet d'un triplet. Soient le graphe RDF G, et les propriétés p_1, p_2 ; $S(p)$ représentant un domaine dans le graphe résumé Gs et $T(p)$ représentant un co-domaine. L'approche propose deux types de résumés suivant différentes règles de formation des nœuds du graphe Gs :

- Baseline Summary : ce graphe regroupe les propriétés ayant un domaine ou un co-domaine en commun, cela en appliquant les règles suivantes :
  — $Si\ (sp_1o_1, sp_2o) \in G$, alors $S(p_1) = S(p_2)$ dans Gs
  — $Si\ (s_1p_1o, s_2p_2o) \in G$, alors $T(p_1) = T(p2)$ dans Gs
  — $Si\ (sp_1o_1, o_1p_2o_2) \in G$, alors $T(p_1) = S(p_2)$ dans Gs

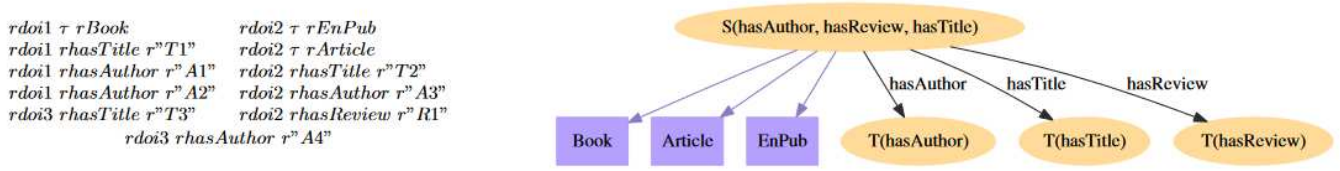


FIGURE 2.12 – Exemple de jeu de données RDF et le Baseline Summary correspondant [16]

La figure 2.12 présente un jeu de données RDF et le graphe *Baseline summary* correspondant. Chaque nœud annoté $S(p_1, p_2, \cdots)$ dans le schéma Gs représente le domaine d'un ensemble de propriétés. Le nœud créé dans l'exemple regroupe les ressources ayant les propriétés $hasAuthor$, $hasReview$ et $hasTitle$ qui sont représentées par des rectangles dans le graphe, bien que ces ressources aient des types différents.

- Refined Summary : dans ce graphe, les entités sont groupées comme pour le *Baseline Summary*, cependant, les déclaration $rdf : type$ sont considérées afin de ne pas regrouper des entités de types différents. Cela est fait en appliquant les règles suivantes :
  — $Si\ (spo, srdf : typec) \in G$, alors $S(p)Tc \in Gs$
  — $Si\ (spo, ordf : typec) \in G$, alors $T(p)Tc \in Gs$

— $Si(srdf : typec)2G$, alors ajouter un nouveau nœud de type c dans le schéma : $nTc \in Gs$

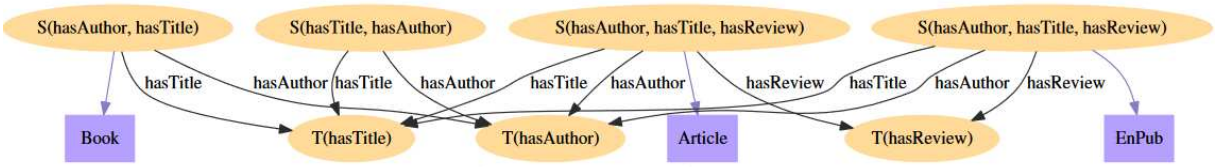


FIGURE 2.13 – Le Refined Summary correspondant au jeu de données présenté dans la figure 2.12 [16]

La figure 2.13 présente le graphe *Refined Summary* du jeu de données utilisé dans notre exemple. Contrairement au premier, celui-ci comporte quatre nœuds, trois d'entre eux correspondent aux différents types des ressources et le dernier représente les ressources n'ayant pas de type.

Le principe de cette approche est de regrouper des propriétés selon leur domaine et co-domaine. Ce regroupement est affiné en considérant les types des ressources afin de ne pas regrouper deux types différents en un même nœud. Cependant, les déclarations de type ne sont pas toujours fournies. Ainsi en l'absence des propriétés $rdf{:}type$, de nouveau nœuds sont créés. Ces nœuds peuvent être vus comme des patterns regroupant des entités décrites par les mêmes propriétés. Par conséquent, elles ne produisent pas un schéma décrivant les entités d'un jeu de données RDF. De plus, selon les auteurs l'approche est très coûteuse avec une complexité de $O(|G|^5)$, où $|G|$ représente le nombre de triplets dans la source de données. Le résumé peut être utilisé pour vérifier qu'une requête possède bien une réponse dans un jeu de données RDF. Cependant, une réponse positive suite à l'interrogation du résumé ne garantit pas que la source contient bien la réponse à la requête, car le résumé ne renseigne pas sur le co-domaine des propriétés ni sur leur valeur.

2.3.3 Bilan des approches de découverte de patterns structurels

Les approches de découverte de patterns identifient les différents ensembles de propriétés décrivant les instances d'une source de données. Un pattern nous renseigne sur l'existence d'entités décrites par l'ensemble de propriétés qui le compose. Il existe des approches qui proposent la découverte de patterns exacts [10, 94, 15, 67, 68], et d'autres qui proposent la découverte de patterns approximatifs [8, 9, 16]. Ces approches sont synthétisées dans le tableau 2.2.

Les approches de découverte de patterns proposent un processus scalable et incrémental, sauf l'approche [16] qui sature les données avant le regroupement des instances similaires. Ces approches s'appuient sur les déclarations de type afin d'identifier les instances d'une même classe dans le but de découvrir les structures décrivant les entités du jeu de données. Le problème avec cette proposition est que les déclarations schéma ne sont pas forcément fournis dans une source de données RDF. En l'absence de déclaration de type, ces approches regroupent les instances décrites par les mêmes propriétés. Cependant, les instances d'une même classe dans une source de

TABLE 2.2 – Synthèse des approches de découverte de patterns

Approches	Algorithme utilisé	Scalable	Incrémental	Utilisation des déclarations	Résultat
FreGraPad [10]	Regroupement par structure et comptage d'instances	Oui	Oui	Non	Patterns exacts avec le nombre d'instances associé à chacun
Versions de schéma [94]	Regroupement par type et identification des structures	Oui	Oui	S'appuie sur les déclaration $rdfType$	Schéma E/A
Query oriented Dans [15]	Regroupement par type ou par structure	Oui	Oui	S'appuie sur les déclaration $rdfType$	Patterns exacts sous forme de résumé de graphe
SchemaDecrypt [67, 68]	Découvertes des versions de type	Oui	Oui	S'appuie sur les déclaration $rdfType$	Versions structurelles des types
Expression régulière [8, 9]	Regroupement par type et identifications des patterns	Oui	Non	S'appuie sur les déclaration $rdfType$	Expression régulière décrivant les patterns avec des propriétés obligatoires et optionnelles
Résumé graphe [16]	Saturation des données et regroupement des instances selon le type de leur sujet	Non	Non	• S'appuie sur les déclaration $rdfType$ • Enrichissement de données	Patterns approximatifs sous forme de filtre pour les requêtes

données RDF ne sont pas nécessairement décrites par les mêmes propriétés. La découverte de pattern peut être utilisé pour réduire la taille du jeu de données, et nous pouvons faire la découverte de types sur les patterns au lieu de le faire sur les instances.

2.4 Approches connexes au problème de découverte de schéma

Les approches présentées dans les sections précédentes ont pour but de découvrir le schéma qui décrit les instances d'un jeu de données RDF, il existe d'autres approches qui proposent d'enrichir le schéma déclaré dans une source de données en inférant et ajoutant les informations absentes ou en identifiant les informations erronées dans la source.

Parmi ces approches, nous trouvons les approches d'enrichissement de schéma qui utilisent les déclarations sur le schéma fournies dans une source de données afin de compléter ces déclarations et enrichir le schéma [107, 85, 82, 116]. Il existe aussi des approches qui analysent la description des instances possédant des types afin de construire des modèles de prédictions qui attribuent des types aux instances non typées [88, 87, 31, 30, 62, 61, 12, 78, 111, 113, 114].

2.4.1 Approches d'enrichissement de schéma

Les approches d'enrichissement de schéma ont pour but d'inférer les déclarations manquantes en se basant sur les déclarations déclarées explicitement dans une source de données RDF. Une revue de ces approches est présentée dans [86]. Ces approches utilisent des algorithmes de fouille de données sur les déclarations explicites sur le schéma afin de les compléter et de les enrichir [107, 85, 82, 116]. Cependant, elles ne découvrent pas de types non définis dans le jeu de données mais attribuent les types déjà existants aux entités de la source.

Parmi les approches d'enrichissement de schéma nous pouvons citer celles qui infère des règles d'associations sur les triplets RDF pour acquérir des connaissances sur le schéma [107, 85]. L'inférence des règles d'associations à partir des données explicites vise à définir des corrélations entre les propriétés [4]. Ces règles d'association sont utilisées pour l'enrichissement du schéma.

L'approche présentée dans [107] utilise les déclarations $rdf{:}type$ pour inférer des règles d'associations sur les triplets RDF. Ces règles sont ensuite utilisées pour enrichir l'ontologie du jeu de données avec des déclarations RDFS ($domain$, $range$, $subClasseOf$, $subPropertyOf$) et OWL ($SymetricProperty$, $TransitiveProperty$). La solution proposée dans [85] suppose que la déclaration de type est absente pour certaines instances, et utilise les règles d'associations afin de déduire des nouveaux types pour les instances.

L'approche [116] utilise un algorithme de clustering hiérarchique afin de construire une hiérarchie des types déclarés dans le schéma du jeu de données.

L'approche [82] découvre les types des instances contenues dans DBpedia en analysant les types des instances des sources de données qui possèdent des liens avec DBpedia. Tout d'abord, les caractéristiques des types des ressources liées à DBpedia sont construites. Ensuite, l'algorithme de classification KNN [36] est utilisé pour typer les instances de DBpedia dont le type est absent.

2.4.2 Approches de typage d'instances

Les approches de typage d'instances dans une source de données RDF utilisent des techniques d'analyse statistique de la distribution des types pour attribuer un ou plusieurs types aux instances de la source [88, 87, 31], ou encore des algorithmes de machine learning afin de construire des modèles de prédictions pour typer les instances du jeu de données.

Nous trouvons dans cette catégorie l'approche SDtype [88, 87] qui dérive les types des instances en appliquant des règles de raisonnement sur les déclarations RDFS. Par exemple, si l'entité e est de type t_1 et que t_1 est un sous type de t_2 alors e est également de type t_2. Cependant, la qualité des déclarations de types générées est très dépendante de la qualité des types existants dans la source considérée. Pour cela, SDtype attribue un poids à chaque propriété décrivant un type, ce qui la rend robuste aux triplets non pertinents. L'approche applique pour chaque instance e les règles d'inférence de type déduites à partir des déclarations schéma (RDFS) dans la source de données RDF. Ensuite, pour chaque type inféré, son degré de confiance est calculé par rapport à l'instance e, selon les propriétés de cette dernière. Les poids des propriétés pour un type sont calculés comme la distribution des propriétés par rapport à la distribution des types. L'approche construit pour chaque propriété un tableau qui comporte la distribution des types du jeu de données par rapport à cette propriété. Le poids d'une propriété est ensuite calculé à partir du tableau de distribution. Les types attribués à une instance sont ceux dont la confiance est supérieure à un seuil donné.

L'approche présentée dans [30] identifie le type des instances de DBpedia en utilisant l'information sur sa catégorie renseignée par la déclaration $dcterms{:}subject$. Tout d'abord, la répartition statistique des catégories est calculée pour tous les types. Des types candidats sont ensuite générés pour chaque instance selon la probabilité de distribution de sa catégorie. Enfin, le type final est défini pour chaque instance en fonction de la probabilité, des mots clés de la catégorie et de la description de l'instance. Cependant, les auteurs indiquent que seulement 54% des instances de DBpedia sont décrites par la propriété $dcterms{:}subject$, et donc l'approche ne peut pas attribuer de type pour le reste des instances car l'information sur leur catégorie est absente.

Il existe des approches qui utilisent des algorithmes de machine learning afin d'extraire à partir des instances possédant déjà un type, les caractéristiques du type. Des modèles de prédiction sont ensuite construits et utilisés pour attribuer un type à chaque entité non typée. D'autres approches exploitent différentes informations pour caractériser une entité du jeu de données, comme sa description textuelle, ses propriétés et sa catégorie [78, 111]. Ensuite, une fonction de prédiction $p(t/e)$ est dérivée afin de déterminer si le type e s'applique à l'entité e. D'autres approches utilisent des corpus d'information pour construire la représentation des entités, puis le modèle de prédiction qui attribue les types au entités [113, 114]. Enfin, les approches [62, 61, 12] utilisent des réseaux de neurones afin de définir des corrélations entre la description des entités et les types. Ces corrélations sont ensuite utilisées afin de typer les entités.

2.4.3 Bilan des approches connexes

Les approches d'enrichissement de schéma se basent sur des déclarations schéma spécifiques fournies dans la source de données comme $rdf{:}type$ ou $owl{:}sameAs$ afin de dériver des informations supplémentaires sur les instances et enrichir le schéma qui les décrit. En l'absence des déclarations sur le schéma, ces approches ne peuvent pas être utilisées. De plus, ces approches ne peuvent pas dériver des types qui n'existent pas dans le jeu de données, sauf l'approche [30] qui est spécifique à DBpedia et qui ne peut pas être généralisée à d'autres sources de données.

Nous faisons face aux mêmes limitations avec les approches de typage d'instances qui construisent leur modèles de prédictions en fonction des caractéristiques extraites à partir des instances qui possèdent déjà un type. Une autre caractéristique de ces approches est qu'elles exigent la disponibilité de grande bases de connaissances afin de construire un modèle efficace durant l'étape d'apprentissage.

2.5 Algorithmes de clustering basés sur la densité pour la découverte de schéma

L'objectif de notre travail est de proposer des solutions au problème de découverte de schéma à partir des données RDF massives et fréquemment mise à jour en analysant la structure des entités que comporte le jeu de données afin de former des clusters d'entités similaires qui représentent les classes du schéma. Nous considérons dans notre contexte que les déclarations de types ne sont pas fournie, et que la découverte de type est fondée sur le regroupement d'instances similaires.

Dans cette section, nous présentons les différentes familles d'algorithmes de clustering qui peuvent être utilisées pour réaliser un tel regroupement, et nous montrerons pourquoi le clustering basé sur la densité (DBSCAN) est l'approche la plus adaptée à la découverte de schéma pour des sources de données RDF. Cependant, DBSCAN ne passe pas à l'échelle et ne s'applique pas dans notre contexte de données massives. Nous présenterons donc également les travaux existants qui proposent de paralléliser le traitement de DBSCAN afin de pouvoir l'appliquer aux grands jeux de données, et qui peuvent être implémentés en utilisant des technologies BigData.

Nous présentons dans la section 2.5.1 les différentes familles d'algorithmes de clustering afin d'expliquer le choix de l'algorithme retenu. La section 2.5.2 introduit le principe général de l'algorithme DBSCAN. La section 2.5.3 présente les algorithmes DBSCAN scalables applicables sur des grands jeux de données. Les algorithmes DBSCAN incrémentaux sont présentés dans la section 2.5.4. Enfin, un bilan de ces approches est présenté dans la section 2.5.5.

2.5.1 Étude de l'adéquation des algorithmes de clustering pour la découverte de schéma

Les algorithmes de clustering peuvent être utilisés pour découvrir les classes composant le schéma d'une source de données RDF par regroupement des entités structurellement similaires. Dans ce qui suit, nous étudions les critères imposés par la nature des données RDF, et nous analysons les différentes familles d'algorithmes de clustering afin d'identifier le plus adapté à la découverte de schéma.

Critères imposés par la nature des données RDF pour le choix de l'algorithme de clustering

Le but du clustering dans notre travail est de regrouper les entités structurellement similaires dans une source de données RDF et de former les classes du schéma. Le choix de l'algorithme de clustering est guidé par la nature des données RDF qui impose certains critères :

- Nous nous intéressons dans notre travail à comparer les structures des entités contenues dans une source de données RDF : l'algorithme de clustering doit donc comparer des ensembles de propriétés et non leurs valeurs. Ces propriétés sont de type catégoriel, non spatial et ne suivent aucune distribution.

- Les entités appartenant à la même classe peuvent être décrites par des propriétés différentes, ainsi, l'algorithme de clustering doit être capable de construire des clusters de formes et de tailles arbitraires pour regrouper les entités décrites par des ensembles de propriétés hétérogènes et qui appartiennent à la même classe.
- Le nombre de classes que comporte une source de données n'est pas défini à priori, l'algorithme ne doit donc pas exiger le nombre de clusters en entrée.
- Les données peuvent contenir des valeurs manquantes ou aberrantes, ce qui nécessite un algorithme robuste au bruit.

Analyse des algorithmes de clustering

Il existe dans la littérature différents types d'algorithme de clustering, qui varient selon plusieurs facteurs : la manière dont les clusters sont produits, le type de données qu'ils traitent, le volume de données qu'ils sont capables de traiter, la capacité à manipuler des données de haute dimension ayant un grand nombre d'attributs, leur déterminisme, etc. Nous distinguons plusieurs familles d'algorithmes de clustering [49, 58, 46]. Nous analysons dans ce qui suit l'adéquation des principaux algorithmes de clustering à la découverte de schéma à partir de données RDF afin de déterminer l'algorithme le mieux adapté.

- Algorithmes de partitionnement : Les algorithmes de partitionnement créent des clusters initiaux de façon aléatoire, puis les améliorent à chaque itération en déplaçant des objets d'un cluster vers un autre et en évaluant le résultat selon des critères comme le nombre des partitions ou leurs tailles [71, 63]. Ces algorithmes sont rapides car ils ne calculent la similarité des objets qu'avec les centres de clusters. Cependant, ils exigent le nombre de clusters en entrée, ne sont pas déterministes et présentent une sensibilité au bruit. De plus, les clusters sont formés autour des centres et leurs formes ne sont pas arbitraires.
- Algorithmes basés sur la densité : Les algorithmes qui suivent le principe de densité forment des clusters à partir des régions denses (régions comportant un seuil minimum d'objets similaires). Un objet est considéré dense si le nombre de ses voisins est supérieur au paramètre de densité [28, 6, 47, 53]. Ces algorithmes sont déterministes (le résultat ne dépend pas de l'ordre de parcours des objets), robuste au bruit, n'exigent pas le nombre de clusters à priori et forment des clusters de formes et de tailles arbitraires. Cependant, ces algorithmes comparent toutes les paires d'objets afin d'identifier ceux qui sont proches, ainsi, leur complexité limite leur utilisation sur les données massives.
- Algorithmes hiérarchiques : Les algorithmes hiérarchiques construisent les clusters graduellement en formant une arborescence et en regroupant les clusters les plus similaires à chaque itération. Ces méthodes sont itératives, et le critère d'arrêt peut être par exemple le nombre de clusters désirés, le nombre d'itérations, le nombre minimum ou maximum d'objets dans chaque cluster ou encore l'inertie. Les algorithmes hiérarchiques sont robustes au bruit et ils traitent différents types de données. Cependant, ils sont coûteux

et leur utilisation sur les grands jeux de données reste un challenge. De plus, ils ne forment pas des clusters de formes arbitraires.

- Algorithmes statistiques : Ces algorithmes utilisent des méthodes statistiques telles que les probabilités pour le regroupement des données. Par exemple, COBWEB [33] organise de façon incrémentale les objets en un arbre de classification. Chaque nœud dans un arbre de classification représente un cluster et il est étiqueté par des probabilités qui résument la distribution des valeurs des attributs des objets classés dans le nœud. Cependant, cet algorithme est non déterministe.
- Algorithmes utilisant les grilles : Ces algorithmes sont conçus pour les données spatiales dans le but de diviser l'espace de données en cellules [110, 98]. Ils créent une structure de grille à partir des données et évaluent la densité des grilles. Les régions jugées denses par rapport au paramètre de densité sont fusionnées. Celles qui ne sont pas denses permettent d'établir les frontières entre les cellules. Ces algorithmes sont rapides car ils fonctionnent sur des cellules au lieu des objets de départ, mais ils sont conçus pour les données spatiales et ne s'appliquent pas aux données RDF dont les propriétés sont de type catégoriel, non spatiale et qui ne suivent aucune distribution.
- Algorithmes flous : Ces algorithmes utilisent des techniques floues pour le clustering des données. Ils considèrent qu'un objet peut appartenir à plus d'un cluster afin de gérer l'incertitude qu'on peut rencontrer dans les cas de figure réels [23, 26]. Cependant, ces algorithmes effectuent plusieurs itérations pour avoir un bon résultat, et ils nécessitent la spécification à priori du nombre de clusters.

Les algorithmes de partitionnement de données et les algorithmes flous ne sont pas adaptés à l'extraction de schéma à partir des données RDF car ils nécessitent le nombre de clusters en entrée, alors que nous ne connaissons pas le nombre de classes que peut comporter une source de données. Les algorithmes statistiques comme COBWEB sont incrémentaux, mais ils ne sont pas déterministes, et ils produisent un schéma différent pour des ordres d'exploration différents du jeu de données. Les algorithmes hiérarchiques sont déterministes, cependant, ils ne forment pas des clusters de formes arbitraires, ne passent pas à l'échelle et ne sont pas incrémentaux. Les algorithmes basés sur la densité sont déterministes, robustes au bruit et ne requièrent pas la connaissance à priori du nombre de clusters. La principale caractéristique de ces algorithmes est leur capacité à former des clusters de formes et de tailles arbitraires. Cet atout est important dans notre contexte où les données RDF peuvent être décrites par des ensembles de propriétés très hétérogènes alors qu'elles appartiennent à la même classe. En effet, le principe de densité ne limite pas la formation de clusters aux objets très similaires, ce qui peut être le cas des données RDF. Enfin, notre travail se situe dans la continuité des travaux proposés dans [64], où l'algorithme DBSCAN a été adapté pour la découverte de schéma à partir de données RDF. Les résultats obtenu par ces travaux ont démontré l'efficacité de DBSCAN pour extraire un schéma à partir des données du web et pour produire les classes décrivant un jeu de données RDF avec une bonne précision et un bon rappel [66].

Pour toutes ces raisons, nous avons choisi de nous appuyer sur l'algorithme DBSCAN la découverte du schéma

implicite d'une source de données RDF. Néanmoins, la complexité de l'algorithme DBSCAN limite son utilisation sur les jeux de données massives. De plus, l'algorithme n'est pas incrémental et exige la disponibilité de la totalité des données pour s'exécuter. Comme les sources de données RDF publiées sur le web sont en constante évolution, il est important que le processus de découverte de schéma soit incrémental afin de permettre l'évolution du schéma sans avoir pour cela à exécuter un processus coûteux sur la totalité de la source de données. Nous étudions dans les sections suivantes les extensions scalables et incrémentales de l'algorithme DBSCAN qui ont été proposées pour le traitement des grands jeux de données.

2.5.2 Principe de DBSCAN [28]

DBSCAN est un algorithme de clustering qui se base sur la densité des points pour former les clusters. Le principe de l'algorithme est que le voisinage dans un rayon donné (ϵ) des points qui forment des clusters doit contenir un nombre minimum de points ($minPts$). En d'autres termes, la densité du voisinage doit excéder un certain seuil. L'algorithme DBSCAN utilise deux paramètres : le seuil de similarité ϵ et le nombre minimum de points $minPts$ devant se trouver dans un rayon ϵ pour que ces points soient considérés comme un cluster.

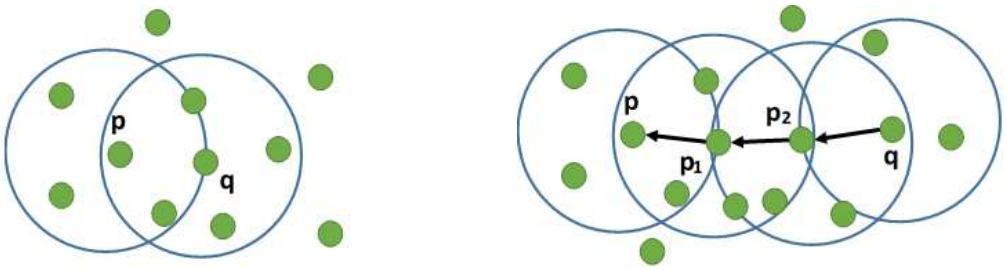
Définition 2.5.1 *L'*ϵ-voisinage *d'un point p est l'ensemble de points qui sont similaires à p selon un indice de similarité S et avec un seuil égal à ϵ.*

$$neighborhood_\epsilon(e) = \{e_i \in D \mid S(e, e_i) \geq \epsilon\}$$

Les points ayant un ϵ-*voisinage* dense, appelés points cores, seront utilisés pour initier les clusters.

Définition 2.5.2 *Un point p est dit point core si le nombre de points dans son ϵ-voisinage est supérieur au paramètre de densité $minPts$.*

$$|neighborhood_\epsilon(p)| \geq minPts.$$



(a) Atteignabilité directe (b) Atteignabilité

FIGURE 2.14 – Les concepts de base de DBSCAN

DBSCAN est fondé sur les concepts d'atteignabilité directe et d'atteignabilité qui assurent la formation des clusters suivant le principe de densité. Nous définissons dans ce qui suit ces deux concepts.

Définition 2.5.3 ***(Atteignabilité directe) :*** *Un point p est directement atteignable par densité à partir d'un point q par rapport à ϵ et $minPts$ si et seulement si q est un point core et p est dans son ϵ-voisinage, i.e. $|neighborhood_\epsilon(q)| \geq minPts$ et $p \in neighborhood_\epsilon(q)$. Ce principe est illustré par la figure 2.14a.*

Définition 2.5.4 ***(Atteignabilité) :*** *Un point p est atteignable par densité à partir de q par rapport à ϵ et $minPts$ s'il existe une chaîne d'entités $p_1, \ldots, p_z$, $p_1 = q$, $p_z = p$ tel que p_{i+1} est directement atteignable par densité à partir de $p_i, \forall i \in \{1, \ldots, z\}$. Ce principe est illustré par la figure 2.14b.*

DBSCAN itère sur les points du jeu de données. Pour chacun des points p qu'il analyse, il construit l'ensemble des points atteignables par densité depuis ce point, c'est-à-dire l'ϵ-*voisinage* de p. Si p est un point core, un cluster C est formé à partir de p et de ses voisins. Les points cores dans C sont identifiés et le cluster est étendu en ajoutant les voisins de chaque point core, et ainsi de suite, jusqu'à ne plus pouvoir agrandir le cluster. Les points qui ne sont affectés à aucun cluster sont considérés comme du bruit.

La complexité de DBSCAN est de $O(n^2)$, ce qui limite son utilisation sur les jeux de données massifs. Nous présentons dans ce qui suit des propositions d'algorithmes DBSCAN scalables et parallèles qui tirent avantage des plateformes de calcul distribuées.

2.5.3 Algorithme DBSCAN scalables

Dans cette section, nous présentons les algorithmes DBSCAN adaptés au traitement des grands jeux de données dans le but d'identifier un algorithme scalable adapté au clustering des données RDF massives. Nous nous intéressons dans notre étude aux algorithmes qui proposent un traitement parallèle et qui peuvent être implémentés en utilisant une technologie Big Data afin d'obtenir de meilleures performances.

Parallélisation de DBSCAN en utilisant une structure de données disjointe, Patwary et al. (2012)

PDSDBSCAN [84] (disjoint-set based parallel DBSCAN) partitionne d'abord aléatoirement les données pour ensuite appliquer l'algorithme DBSCAN sur les différentes partitions. Durant le calcul de voisinage des entités pour la formation des clusters, l'algorithme compare les entités d'une partition avec la totalité du jeu de données. Les entités ayant un voisinage dense sont considérées comme des cores.

La parallélisation des calculs est assurée en utilisant une structure de données disjointe. L'idée principale est d'exécuter DBSCAN en parallèle sur chaque partition afin de former des clusters locaux. Ensuite, les clusters locaux sont fusionnés afin de former le résultat final.

Durant le calcul de l'ϵ-voisinage des entités, les entités similaires appartenant à la même partition sont regroupées dans des clusters comme lors d'une exécution classique de DBSCAN. Dans le cas où des entités similaires appartiennent à des partitions différentes, ces entités sont assignées à différents clusters qui sont ensuite fusionnés.

Cet algorithme produit le même résultat de clustering que le DBSCAN séquentiel. Toutefois, il exige un accès au jeu de données complet à partir des différentes partitions, ce qui implique soit un surcoût lié aux échanges de données entre les partitions, soit la copie de l'intégralité du jeu de données dans les différentes partitions. Ces exigences pénalisent l'algorithme lors de son application à de grands jeux de données.

MR-DBSCAN : DBSCAN scalable basé sur la technologie Map/Reduce, Yaobin et al. (2011)

Dans [50, 51], les auteurs ont conçu et implémenté MR-DBSCAN, un algorithme DBSCAN parallèle qui utilise la technologie BigData Map/Reduce. MR-DBSCAN est composé de trois étapes : le partitionnement spatial des données, le clustering local dans chaque partition et la fusion des résultats locaux afin de produire le clustering final.

Pour partitionner les données, MR-DBSCAN utilise l'algorithme BSP (partition binaire de l'espace) [37] qui divise les données en sous-ensembles de tailles égales, en fonction de leur proximité spatiale. L'algorithme BSP divise récursivement en deux régions l'ensemble de données jusqu'à avoir des partitions de petite taille. Enfin, les points situés sur les bordures des régions produites durant le partitionnement sont dupliqués dans les régions voisines comme le montre la figure 2.15. Par exemple, la partition en vert correspondant à la région S_1 comportera les points bordures des régions S_0 et S_4. Ces points permettront la découverte des clusters divisés sur plusieurs régions.

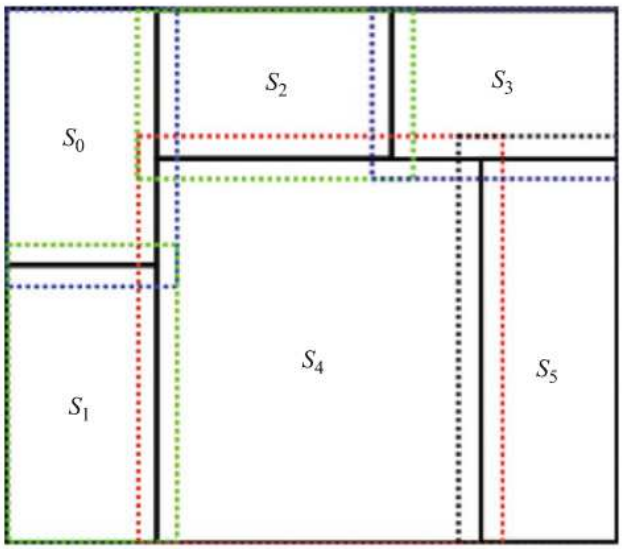


FIGURE 2.15 – Illustration du partitionnement d'un espace de données en utilisant la méthode BSP [50]

Après le partitionnement des données, MR-DBSCAN calcule les clusters locaux sur chaque partition en parallèle.

Enfin, MR-DBSCAN fusionne les clusters locaux en s'appuyant sur les points bordures dupliqués dans les différentes partitions. Si un point appartient à différents clusters, ces derniers sont ainsi fusionnés. Cette étape produit le résultat final du clustering.

Cette approche a été implémentée en utilisant Hadoop Map/Reduce [50, 51] et en utilisant Spark [22], deux technologies BigData qui permettent le passage à l'échelle de l'approche. Cependant, les propriétés des données RDF sont de type catégoriel et il n'est pas évident de les représenter dans un espace à n-dimensions comme requis dans l'approche de partition binaire de l'espace (BSP) afin de partitionner les données. L'idée d'utiliser cet algorithme afin d'extraire un schéma à partir des jeux de données RDF est donc exclue.

Parallélisation de DBSCAN par ordonnancement des entités, Dianwei et al. (2016)

Dans ce travail [48], les auteurs proposent un algorithme DBSCAN scalable, implémenté en utilisant la plateforme BigData Spark. L'idée de cet algorithme est de représenter les entités du jeu de données par des points permettant leur ordonnancement.

Ensuite, les données sont envoyées aléatoirement vers différents nœuds de calculs où le clustering se fera en parallèle. Chaque nœud calcule les clusters localement sans aucune communication avec les autres nœuds, ceci afin d'éviter le coût des transferts des données.

Après la distribution des données, les index des entités sont divisés en intervalle par rapport à leur ordre, chaque intervalle est défini pour une partition. Durant le clustering, les points ayant un index (ordre) non compris dans l'intervalle d'une partition sont considérés comme des points *SEED*. Les points $SEEDs$ introduits dans cet article sont utilisés afin de fusionner les clusters divisés sur plusieurs partitions. Ces points $SEEDs$ sont aussi dupliqués dans la partition correspondant à l'intervalle auquel ils appartiennent. Enfin, l'approche découvre les $SEEDs$ appartenant à plusieurs clusters afin de fusionner ces derniers. Cette étape produit le résultat final du clustering. Le problème posé par l'application de cette approche aux données RDF est la transformation des entités en points qui seront indexés et ordonnés dans des intervalles. Cette transformation ne s'applique pas aux données RDF qui comportent des propriétés de type catégoriel, et leur ordonnancement n'est pas trivial.

S-DBSCAN, Guangchun et al. (2016)

S-DBSCAN [73] est une version parallèle basée sur Spark de DBSCAN, qui est composé des étapes suivantes : le partitionnement aléatoire des données, le calcul des clusters localement sur chaque partition en parallèle, et la fusion des clusters locaux en s'appuyant sur leurs centres.

Le partitionnement est fait d'abord en déterminant le nombre de partitions à partir du nombre de nœuds de calcul. Sur la base de ce nombre, une fonction aléatoire distribue les données à travers les nœuds de calcul. Chaque partition comporte approximativement le même nombre d'entités.

Le clustering local est ensuite effectué dans chacune des partitions. DBSCAN est exécuté localement afin de grouper les entités similaires. Chaque partition produit un résultat de clustering indépendant des autres partitions.

La fusion des clusters locaux est effectuée en se basant sur les centres de ces clusters. Tout d'abord, les

distances entre les centres des clusters dans une même partition sont calculés puis ordonnés afin de définir la distance minimum qui sépare deux clusters dans une même partition. Ensuite, toutes les valeurs de distances minimum découvertes sur chaque partition sont ordonnées, et la plus petite de ces valeurs est définie comme le seuil D_{min} qui sera utilisé durant la fusion des clusters locaux. Les clusters locaux ayant des centres dont la distance est inférieure à D_{min} sont fusionnés pour produire le clustering final.

S-DBSCAN est une version scalable de DBSCAN, car il se base sur une fonction de distribution aléatoire qui permet la création rapide de partitions de même taille. Cependant, S-DBSCAN utilise les centres de clusters locaux afin de déterminer et fusionner les clusters distribués sur plusieurs partitions. En l'utilisant pour la découverte de schéma, cela conduirait à un schéma de moins bonne qualité que s'il avait été obtenu en utilisant DBSCAN. Définir un centre représentatif pour un ensemble d'entités RDF n'est pas évident. En effet, calculer une moyenne à partir des propriétés décrivant des données RDF n'est pas trivial car les propriétés sont de type catégoriel. De plus, choisir une entité afin de représenter un cluster ne garantit pas que le résultat de clustering est le même que celui produit par l'algorithme DBSCAN.

Algorithme DBSCAN parallèle en utilisant MPI, Savvas et Tselios (2016)

Cette approche [95] procède de façon similaire à S-DBSCAN [73] pour distribuer les données et former les clusters locaux. Mais la fusion des clusters locaux afin de produire le clustering final est différente.

Dans ce travail, l'idée principale pour paralléliser DBSCAN est de diviser les données en sous-ensembles aléatoirement pour ensuite exécuter DBSCAN dans chaque sous-ensemble de données indépendamment. Durant le processus du clustering, le centre et le rayon de chaque cluster local, formé sur chaque partition, sont calculés. Ensuite, les nœuds de calcul échangent la liste des clusters locaux afin de vérifier l'existence des clusters qui doivent fusionner pour former un cluster commun.

Un cluster est représenté par un cercle caractérisé par le centre et le rayon de ce cluster. L'idée est de trouver les cercles qui ont une intersection non vide. Cette idée est illustrée dans la figure 2.16.

Soient $C_1 = (c_1, r_1)$ et $C_2 = (c_2, r_2)$, deux cercles représentant deux clusters locaux différents, et d la distance entre leurs centres, et soient les points $c_1 = (x_1, y_1)$ et $c_2 = (x_2, y_2)$ représentant les coordonnées des centres dans un espace Euclidien à 2 dimensions. Les clusters locaux qui doivent fusionner sont identifiés en s'appuyant sur leurs centres et leurs rayons. On distingue de ce fait les cas suivants :

- Un cluster est inclus à l'intérieur d'un autre ou ils comportent des points communs si $d \leq |r_1 - r_2|$: ces clusters sont fusionnés s'il existe un point core d'un cluster qui est dans l'ϵ-voisinage d'un core dans l'autre cluster.
- Deux clusters s'intersectent en deux point si $d > |r_1 - r_2|$ et $d < r_1 + r_2$: dans ce cas, l'approche évalue la distance entre les points dans la bordure des clusters et fusionne les clusters s'il existe des points similaires.
- Deux clusters s'intersectent en un point si $d = r_1 + r_2$: alors ce point est identifié en résolvant les équations ci-dessous.

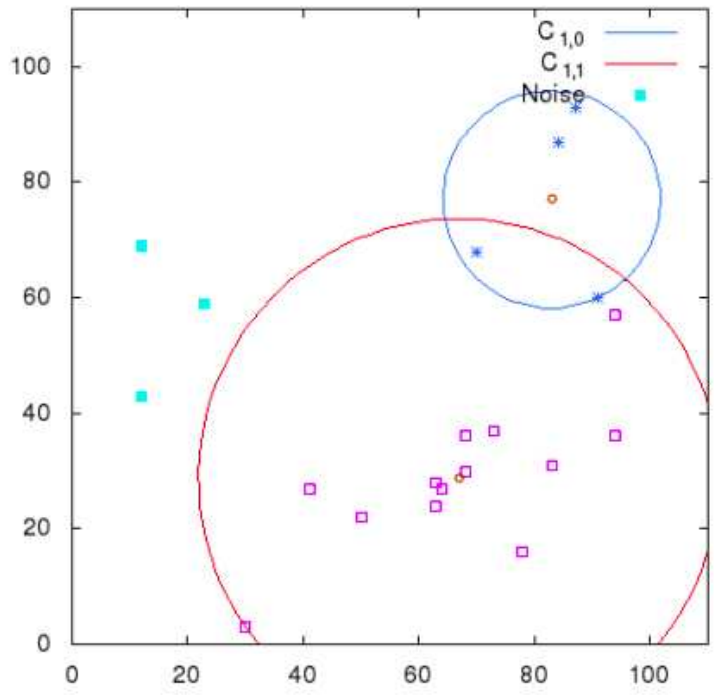


FIGURE 2.16 – Exemple de représentation de clusters [95]

- Les clusters ne s'intersectent pas si $d > r_1 + r_2$: dans ce dernier cas, aucune fusion n'est possible.

Le point ou les points qui se touchent sont identifiés en résolvant les équations :

$(x - x_1)^2 + (y - y_1)^2 = {r_1}^2$ et $(x - x_2)^2 + (y - y_2)^2 = {r_2}^2$.

Les résultats obtenus à partir de cette approche à travers des exemples concrets montrent que l'approche produit le même résultat que le DBSCAN séquentiel et qu'elle réduit sa complexité. Cependant, l'algorithme s'appuie sur les centres des clusters ainsi que la représentation des clusters dans un espace euclidien. Ces deux exigences ne peuvent pas être satisfaites par les jeux de données RDF qui sont de nature catégorielle et qui ne sont pas facilement représentés dans un espace à 2-dimensions.

HPDBSCAN, Götz et al. (2015)

HPDBSCAN [42] propose une exécution distribuée de DBSCAN en adoptant une technique de division de l'espace qui permet la distribution du calcul sur différents processus, et l'ordonnancement des données par rapport à leurs distances spatiales afin d'accélérer la recherche de voisinage de chaque entité.

Dans la première étape, un modèle de division de l'espace de données en cellules est calculé à l'aide de méthode tel que STING [110] et HACC [45]. Ensuite, ce modèle est superposé sur les données chargées dans les différents nœuds de calcul afin d'organiser les entités de façon à ce que toutes les entités dans une même cellule soient affectées à la même partition de données, et ainsi, au même nœud de calcul. Comme un nœud de calcul

peut traiter plusieurs cellules, l'approche assure que les nœuds ont une charge de travail équivalente en évaluant la complexité de chaque cellule. La complexité d'une cellule c est évaluée comme suit :

$Cost_{Cell}(c) = |c| \times |N_{Cell}(c)|$, $N_{Cell}(c)$ étant le voisinage de la cellule c.

La figure 2.17 montre une division équitable des données sur deux processeurs (nœuds de calcul).

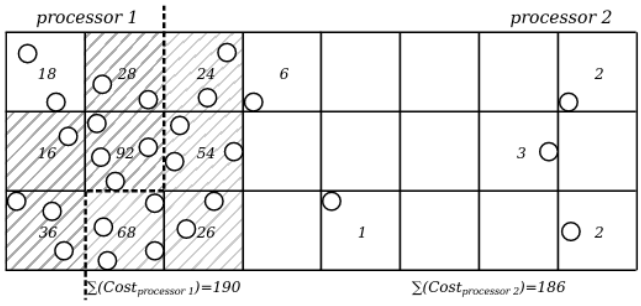


FIGURE 2.17 – Décomposition en cellules d'un espace de données par l'algorithme HPDBSCAN [42]

Chaque processeur forme les clusters localement en appliquant le principe de densité de DBSCAN. La recherche de voisinage nécessaire à la construction des clusters est accélérée tout d'abord par la parallélisation de son exécution, et par la recherche de voisinage rapide grâce à l'ordre défini sur les entités dans chaque partition. Dans cette étape, les clusters sont formés en une seule itération sur les entités contenues dans la partition. Le voisinage de chaque entité p est déterminée. S'il est dense et qu'aucun core ne fait partie de ce voisinage alors p et son voisinage sont étiquetés par le cluster de p. Dans le cas où un core q déjà étiqueté est dans le voisinage de p, alors p est étiquetée par le cluster de q. Lorsque plusieurs cores sont dans le voisinage de l'entité p, alors elle est étiquetée par les clusters de tous ces cores. Cette information est ainsi sauvegardée dans un espace partagé et sera utilisée pour la fusion des clusters locaux dans la prochaine étape.

Après la construction des clusters localement, les informations concernant les entités étiquetées par plusieurs clusters sont traitées et les clusters correspondants sont fusionnés. Ensuite, l'approche évalue la similarité des entités se trouvant dans la bordure des partitions afin de découvrir des entités similaires séparées par la division de l'espace effectuée lors de la première étape. Les entités bordures sont affectées aux clusters selon le principe de densité. Dans le cas où elles appartiennent à plusieurs clusters, ces derniers sont fusionnés.

HPDBSCAN améliore DBSCAN en parallélisant son exécution et en accélérant la recherche de voisinage qui représente l'opération la plus coûteuse de l'algorithme. Cependant, il procède en divisant l'espace des données et en ordonnant les entités. Cette contrainte ne permet pas l'utilisation de HPDBSCAN sur les données du web car ces dernières ne peuvent pas être divisées selon un critère de distance spatiale.

NG-DBSCAN : Clustering scalable basé sur la densité pour les données arbitraires, Lulli et al. (2016)

NG-DBSCAN [72] est un algorithme approximatif de clustering basé sur la densité qui opère sur des données arbitraires et avec une mesure de similarité symétrique quelconque. La conception distribuée de l'algorithme le rend scalable et adapté pour les plateformes distribuées comme Spark.

NG-DBSCAN évite le coût du calcul de l'ϵ-voisinage pour chaque point en procédant en deux étapes.

Tout d'abord, l'approche crée un ϵ-graphe, une structure de données où les nœuds sont des points de données et le voisinage de chaque nœud est un sous-ensemble de son ϵ-voisinage. Le processus de construction du ϵ-graphe converge à partir d'un graphe de départ d'une configuration aléatoire vers une approximation d'un graphe k plus proches voisins. Le graphe de voisinage est d'abord initialisé en connectant chaque nœud avec k autres nœuds, k étant un paramètre de NG-DBSCAN. L'ϵ-graphe est utilisé afin d'éviter de comparer toutes les entités entre elles. À chaque itération, toutes les paires de nœuds (x, y) séparées par deux sauts dans le graphe de voisinage sont considérées. Si la distance entre elles est inférieure au poids de l'arc sortant e le plus grand (le poids d'un arc représente la distance entre deux entités), alors e est remplacé par l'arc qui lie x et y. Grâce à cette étape, dès qu'une paire de nœuds à distance ϵ ou inférieure est découverte, l'arc correspondant est ajouté dans l'ϵ-graphe. Un nœud est considéré comme ayant assez de voisins lorsqu'il a un nombre M_{max} de voisins dans l'ϵ-graphe. Ainsi, il est enlevé de l'ϵ-graphe afin d'accélérer le processus. M_{max} est un paramètre de NG-DBSCAN.

Ensuite, l'ϵ-graphe est utilisé afin de calculer les clusters finaux. Durant cette étape, le coûteux calcul de l'ϵ-voisinage de chaque entité est remplacé par des requêtes de voisinage sur l'ϵ-graphe, beaucoup moins coûteuses. En considérant ce graphe, les rôles sont attribués à chaque nœud : les nœuds ayant plus que $minPts$ voisins sont considérés comme des nœuds cores ; les nœuds dans le voisinage des cores sont des nœuds bordures ; les autres nœuds sont considérés comme du bruit. Cette étape découvre les points de départ ($seeds$) de tous les clusters, et construit un ensemble d'arbres appelé la forêt de propagation qui lie les nœuds du graphe à leur $seed$. Afin d'identifier les $seeds$, chaque nœud i partage l'identifiant de son voisin ayant le plus grand nombre d'arcs, le propose comme étant un $seed$ et construit un arc entre ce $seed$ et les voisins du nœud i. Les nœuds bordures construisent uniquement un arc entre eux et leur $seed$. Les nœuds non proposés comme $seeds$ sont ensuite désactivés et un arc entre les nœuds désactivés et leurs $seeds$ est ajouté à l'arbre de propagation. Cette opération est répétée jusqu'à ce que tous les nœuds actifs soient des nœuds $seeds$. Enfin, chaque arbre dans la forêt de propagation représente un cluster qui regroupe tous les nœuds de l'arbre.

L'aspect distribué de NG-DBSCAN le rend scalable et adapté aux grands jeux de données et sa nature approximative le rend plus rapide. Cependant, il ne produit pas le même résultat de clustering que DBSCAN ce qui peut réduire la qualité des classes produites s'il est utilisé pour la découverte de schéma.

Un algorithme DBSCAN basé sur un partitionnement aléatoire, Hwanjun et Jae-Gil (2018)

RP-DBSCAN (Random Partitioning-DBSCAN) [100] est un algorithme DBSCAN parallèle qui utilise un partitionnement pseudo-aléatoire combiné avec un dictionnaire de cellules à deux niveaux.

L'algorithme RP-DBSCAN comprend trois phases. La première est la phase de partitionnement des données, qui distribue les entités pour un traitement parallèle. RP-DBSCAN utilise un partitionnement pseudo-aléatoire : il divise l'espace de données en cellules avec une diagonale ϵ et distribue aléatoirement les cellules à travers les nœuds de calcul. Plutôt que de traiter des points individuels, l'approche utilise le concept de sous-cellules où une sous-cellule est un hyper-cube à d-dimensions avec une diagonale de taille $\epsilon/2^{h-1}$, $h = 1 + \log_2(1/\rho)$, où $\rho(> 0)$ est le paramètre qui détermine la taille d'une sous-cellule. Les cellules sont représentées par leur densité (nombre de points à l'intérieur d'une cellule) et leur position (centre d'une cellule), comme le représente la figure 2.18. Les points sont affectés à une sous-cellule par rapport à leur position. Enfin, les données sont résumées dans un dictionnaire de cellules à deux niveaux, le premier niveau représentant les cellules et le deuxième les sous-cellules, comme le montre la figure 2.18. Le dictionnaire de cellules à deux niveaux est diffusé à tous les nœuds de calcul pour permettre l'exécution de requêtes de région sans échange d'information entre les nœuds de calcul.

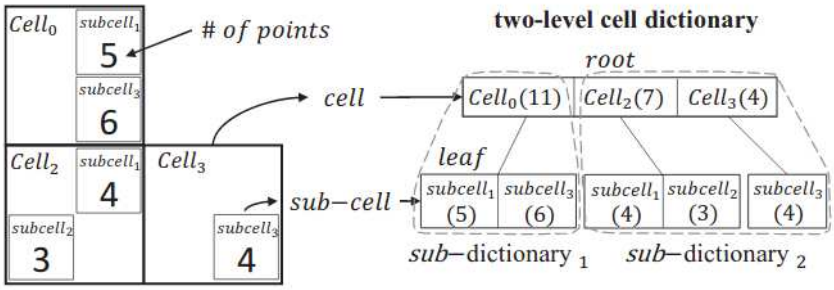


FIGURE 2.18 – Exemple de graphe de cellules [100]

Dans la deuxième phase, un graphe de cellules est construit en groupant, dans chaque partition toutes les cellules qui sont atteignables par densité, en d'autres termes, les cellules formant un cluster en s'appuyant sur le principe de densité. Un graphe de cellules représente un résultat de clustering local obtenu par une partition. En se basant sur le dictionnaire de cellules à deux niveaux, cette phase découvre aussi les cellules qui sont atteignables par densité et qui n'appartiennent pas à la même partition.

La troisième phase vise à combiner les graphes de cellules retournés par chaque nœud de calcul afin de produire le résultat du clustering global. Afin de rendre la phase de fusion scalable, l'approche ne considère pas tous les arcs de tous les sous-graphes de cellules afin d'obtenir les clusters finaux. Chaque arc qui lie deux cellules indique que les points dans ces deux cellules doivent appartenir au même cluster. Ainsi, les clusters auxquels appartiennent ces points sont fusionnés. Enfin, l'approche affecte les points aux clusters par rapport à la cellule à laquelle ils appartiennent et produit le résultat de clustering final.

RP-DBSCAN combine différentes techniques, car il partitionne les cellules de façon aléatoire, et crée ensuite un graphe en utilisant le BSP [37] afin d'accélérer la recherche de voisinage dans chaque partition. Enfin, il fusionne les clusters découverts dans chaque partition afin de produire les clusters finaux. Comme il utilise une structure basée sur des cellules, cet algorithme ne s'applique pas aux jeux de données RDF car il n'est n'est pas évident de représenter de telles données dans un espace à n-dimension. De plus, la qualité du clustering dépend fortement du paramètre ρ, et ce résultat n'est pas toujours le même que celui retourné par DBSCAN.

2.5.4 Extensions incrémentales de DBSCAN

L'un des objectifs de notre travail est de maintenir le schéma extrait à partir d'une source de données cohérent avec l'évolution des données. c'est-à-dire que les classes du schéma doivent permettre de décrire l'ensemble des entités contenues dans la source. Cependant, l'algorithme DBSCAN n'assure pas la mise à jour du schéma construit lorsqu'un jeu de données évolue. Cette section présente les extensions incrémentales de l'algorithme DBSCAN qui forment les clusters de façon évolutive et modifient le clustering existant des données afin d'assurer la cohérence avec le jeu de données.

Clustering incrémental dans un environnement d'entrepôts de données, Ester et al. (1998)

Afin de mettre à jour les connaissances découvertes à partir des entrepôts de données en utilisant les algorithmes de data mining, les auteurs présentent dans [29] le premier algorithme DBSCAN qui modifie de façon incrémentale les classes ou clusters extraits.

Dans ce travail, il est démontré que les clusters à modifier après insertion ou suppression d'une entité e sont ceux dans le ϵ-voisinage de l'entité e. Les entités dans le voisinage de e peuvent changer de nature : des entités non-cores peuvent devenir cores et vice-versa. Ensuite, le $2 \times \epsilon$-voisinage est analysé afin d'identifier les entités bordure qui changent de nature. Le $2 \times \epsilon$-voisinage d'une entité e est l'ensemble des entités similaires à e avec un seuil égal à $2 \times \epsilon$. En effet, les entités cores dans le $2 \times \epsilon$-voisinage de l'entité e gardent leur nature et restent inchangées. Mais des entités non-cores peuvent devenir des entités bruits si elles n'ont plus d'entité core dans leur $2 \times \epsilon$-voisinage. Dans le cas d'un ajout, une entité bruit peut avoir un nouveau core dans son voisinage et ainsi devenir une entité bordure. Il n'y a pas de changement pour le reste des entités.

En raison du principe de densité de DBSCAN, l'insertion ou la suppression d'entités affecte seulement les clusters existants qui sont dans le voisinage de ces entités. Ceci conduit à un algorithme de clustering incrémental qui produit le même résultat que DBSCAN.

Après insertion d'une entité e, les entités non-cores peuvent devenir cores, impliquant la création de nouvelles connexions suivant le principe de densité, et ainsi, la création de nouveaux clusters ou la modification de clusters existants. Les différents cas de figure à considérer après insertion d'une entité sont les suivants :

- Bruit : une entité insérée e est considérée comme bruit s'il n'existe pas d'entité core dans son voisinage, et le clustering reste inchangé.
- Création : un nouveau cluster est créé si le voisinage d'une entité insérée e contient un nouveau core qui n'appartient pas à un cluster existant. Le cluster est construit à partir d'entités considérées comme du bruit avant l'insertion de e.
- Absorption : le voisinage de l'entité insérée e contient un core qui appartient à un cluster existant, ainsi, e est absorbée par ce cluster.
- Fusion : des clusters existants sont fusionnés si le voisinage d'une entité insérée e contient des entités cores qui appartiennent à ces clusters. L'entité e est ajoutée au cluster résultant de la fusion.

D'autre part, la suppression d'une entité e implique que des entités cores peuvent devenir non-cores. En conséquence, des connexions par densité peuvent disparaître, et des clusters existants peuvent se diviser ou disparaître. Les auteurs distinguent les cas de figure suivants après la suppression d'une entité :

- Suppression : un cluster est supprimé si une entité core à l'intérieur de ce cluster change de nature et devient une entité bruit après la suppression de l'entité e.
- Réduction : les entités dans le voisinage de l'entité supprimée e peuvent devenir du bruit, ainsi, elles ne sont plus atteignables par densité et donc supprimées du cluster.
- Division : la suppression d'une entité e crée, à partir d'un cluster, différents ensembles d'entités qui peuvent former des clusters. Dans ce cas, le cluster reste le même si les entités de ces ensembles sont atteignables les unes des autres par densité. Dans le cas contraire, le cluster est divisé en plusieurs clusters selon les connexions entres les entités et suivant le principe de densité.

Ce travail représente le premier algorithme de clustering incrémental basé sur DBSCAN et qui produit le même résultat de clustering ce dernier. Cependant, cette approche traite une entité à la fois, et ne gère pas l'insertion ou la suppression d'ensembles d'entités. De plus, la mise à jour des clusters après l'insertion ou la suppression d'une entité implique la comparaison de cette entité avec le reste du jeu de données ; cette opération est coûteuse, ce qui rend son utilisation sur les grands jeux de données impossible.

Algorithme de clustering par densité incrémental pour des grands jeux de données, M.Bakr et al. (2015)

Dans [75], les auteurs proposent une amélioration de l'algorithme DBSCAN incrémental [29] introduit ci-dessus, en limitant la recherche de voisinage d'une entité insérée ou supprimée à certaines partitions de données plutôt qu'à la totalité du jeu de données.

Afin de partitionner les données, l'algorithme choisit k entités qui représentent les centres de chaque partition. Une nouvelle entité est affectée à la partition avec le centre le plus proche et comparée avec les entités de cette partition pour déterminer son voisinage. Le partitionnement peut se faire à l'aide d'algorithmes comme k-means ou encore celui proposé dans [115], qui partent de centres aléatoires de partitions et les optimisent jusqu'à trouver les

centres les plus représentatifs. Ces méthodes permettent aussi de recalculer les centres des partitions lorsque le jeu de données évolue.

Lorsqu'une nouvelle entité est ajoutée au jeu de données, elle est affectée à la partition avec le centre le plus proche, où est appliqué l'algorithme DBSCAN incrémental proposé dans [29] en ne considérant que les entités dans la partition contenant la nouvelle entité.

L'algorithme analyse ensuite les clusters résultants de chaque partition afin d'identifier les clusters qui doivent fusionner. Pour cela, il évalue l'inter-connectivité (IE) des entités aux bordures des clusters comme le montre la figure 2.19. L'inter-connectivité entre deux clusters A et B notée IE est évaluée comme suit :

$$IE(A, B) = \frac{N_{ab}}{(N_a + N_b)/2}$$

N_a et N_b représentent le nombre d'entités à la bordure des clusters A et B respectivement, et N_{ab} est le nombre d'arcs qui relient les bordures du cluster A à celles du cluster B.

Les clusters ayant une inter-connectivité supérieure à un seuil α sont fusionnés. Les entités non affectées à un cluster représentent du bruit.

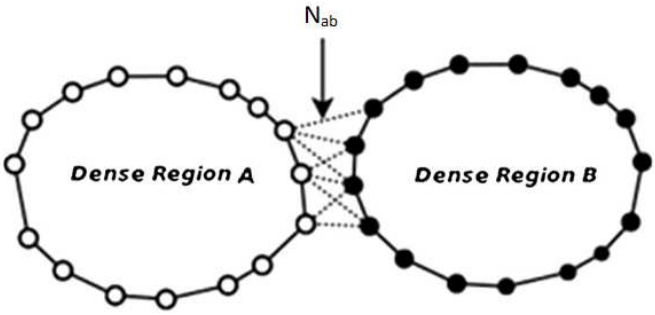


FIGURE 2.19 – Interconnectivité entre deux régions

Ce travail présente une amélioration de l'algorithme DBSCAN incrémental qui accélère la recherche de voisinage d'une entité en limitant la recherche à une partie du jeu de données. Ceci réduit le coût de la comparaison entre les entités, qui est l'opération la plus coûteuse dans DBSCAN. Le problème de cette approche se situe dans sa méthode de fusion des clusters produits dans chaque partition. En effet, l'approche compare les entités situées dans les bordures adjacentes des clusters et fait l'hypothèse que si les entités bordures de différents clusters ne sont pas proches, aucune autre entité à l'intérieur du cluster ne l'est. Ceci est vrai pour les données qui peuvent être représentées dans un espace à n-dimension, mais pas pour les données RDF. Dans des clusters formés à partir d'entités d'un jeu de données RDF, si deux entités situées dans les bordures de différents clusters ne sont pas similaires, rien ne permet d'affirmer que des cores dans ces clusters ne soient pas similaires.

Analyse et étude de DBSCAN incrémental, Chakraborty et Nagwani (2014)

Dans [18], l'approche proposée évalue le taux de changement effectué sur un jeu de données afin de décider si l'algorithme incrémental est plus performant que l'exécution de DBSCAN sur la totalité du jeu de données. Pour ce faire, le taux de changement, désigné par $\%\delta\ change\ in\ DB$, est calculé à l'aide de la formule suivante :

$\%\delta\ change\ in\ DB = \frac{new\ data - old\ data}{old\ data} \times 100.$

Dans le cas où le taux de changement dépasse le pourcentage des anciennes entités par rapport au jeu de données total, les auteurs recommandent l'utilisation de DBSCAN. Dans le cas inverse, l'utilisation de la version incrémentale de DBSCAN offre de meilleures performances.

Pour modifier les clusters découverts par DBSCAN de façon incrémentale après insertion de nouvelles entités, l'approche proposée dans [18] compare les nouvelles insertions avec les cores des clusters existants. Une nouvelle entité est donc affectée au cluster contenant le core le plus proche.

Ensuite, la similarité entre les entités bruitées est évaluée afin de former des clusters avec ces entités si l'une d'entre elles est une entité core.

Cette approche compare les nouvelles entités avec les cores des clusters existants afin d'accélérer l'affectation des entités vers les clusters les plus proches. Le nombre de cores étant plus réduit que le nombre des entités dans un jeu de données, la comparaison est plus rapide. Cependant, il peut exister des nouvelles entités qui ne sont affectées à aucun cluster, car elles ne sont similaires à aucun core dans ces clusters. Ces mêmes entités peuvent former un clusters avec d'autres entités bordures si leur voisinage est dense. De plus, l'insertion d'une nouvelle entité similaire à une ancienne entité bordure peut transformer cette dernière en une entité core qui étend le cluster en ajoutant ses voisines. Par conséquent, cette approche réduit la qualité du clustering produit et ne fournit pas le même résultat que DBSCAN. Son utilisation pour l'extraction de schéma dans le contexte du web des données implique la dégradation de la qualité du schéma produit.

RT-DBSCAN : Clustering parallèle en temps réel de jeux de données spatio-temporels en utilisant spark-streaming, Yikai et al. (2018)

RT-DBSCAN [41] est une extension de l'algorithme DBSCAN pour le clustering de données spatio-temporelles en temps réel implémentée en utilisant Apache Spark Streaming et en exploitant une plateforme appelée SMASH [40] qui gère le trafic des données à grande échelle.

L'objectif de cette approche est d'accélérer le calcul de voisinage pour chaque entité insérée. En effet, le voisinage d'une entité insérée représente le sous-ensemble de données nécessaire au clustering après l'insertion d'une entité. Le sous-ensemble de données, désigné par ϵ', à considérer lors de l'insertion d'une entité e est calculé comme suit :

$\epsilon' = ((minPts - 1) \times \epsilon)$-voisinage de e.

RT-DBSCAN accélère le calcul du ϵ'-voisinage de chaque entité insérée en parallélisant son exécution. Les flux de données entrants sont divisés en sous-ensembles de données ($ticks$) en fonction de leur heure d'arrivée ; à l'intérieur de chaque $tick$, les données sont traitées en parallèle et regroupées en clusters. Chaque $tick$ de RT-DBSCAN est exécuté en Map/Reduce en trois étapes : dans la première, l'espace de données est divisé géographiquement en plusieurs cellules. L'espace de données est divisé en utilisant le *Fast Clustering partitioning*. L'idée de cette méthode de partitionnement est de diviser itérativement l'espace $2D$ en quatre sous-cellules jusqu'à atteindre un seuil donné, défini par le nombre d'entités dans une cellule. Les cellules contenant un nombre d'entités inférieur à $minPts$ sont ensuite supprimées. Dans la deuxième étape, l'algorithme DBSCAN séquentiel est exécuté, en parallèle sur chaque cellule, afin de créer des clusters localement. La troisième étape fusionne les clusters locaux produits par les processus parallèles pour traiter le cas des clusters dont les éléments s'étendent sur plusieurs partitions. Les clusters ayant des entités dans la bordure d'une cellule peuvent avoir des entités similaires dans une autre cellule. Ainsi, la similarité des entités dans les bordures des cellules est évaluée afin de fusionner les clusters locaux comportant des entités similaires.

Dans le but d'assurer l'incrémentalité de l'approche, le résultat est stocké dans un $checkpoint$ qui maintient les différents clusters qui seront utilisés dans le clustering des futurs flux de données.

RT-DBSCAN représente un algorithme de clustering par densité rapide et incrémental, conçu pour le clustering des flux de données en continu. Cependant, la méthode de partitionnement utilisée pour la parallélisation du clustering dans chaque flux de données est conçue pour les données qui peuvent être représentées dans un espace à deux dimensions et ne s'applique pas pour les données RDF qui sont représentées par un grand nombre de propriétés.

2.5.5 Bilan des extensions scalables ou incrémentales de l'algorithme DBSCAN

Dans le but de construire des clusters suivant le principe de densité à partir de grands jeux de données, les approches existantes distribuent le traitement du clustering, et cette exécution parallèle accélère le processus. Ces algorithme sont résumés dans le tableau 2.3.

Cependant, les algorithmes DBSCAN scalables discutés dans cette section présentent des limites qui empêchent leur utilisation pour le clustering des données RDF pour la découverte de schéma, et offrir un schéma de bonne qualité :

- Les approches qui distribuent les données aléatoirement sur les différents processus offrent un traitement rapide, mais ne garantissent pas le regroupement de tous les points similaires dans des clusters. Par conséquent, la qualité du clustering est réduite [73, 95]. Dans notre contexte, ces approches produisent un schéma

TABLE 2.3 – Synthèse des versions scalables de l'algorithme DBSCAN

Approches	**Méthode de partitionnement**	**Principe de clustering**	**Avantages**	**Inconvénients**
PDSDBSCAN [84]	Partitionnement aléatoire	Clustering en comparant les points d'une partition avec tous les autres points du jeu de données	• Partitionnement rapide • Création de partitions de taille égale • Résultat égale à DBSCAN	Échange d'information important pour accéder à la totalité du jeu de données
MR-DBSCAN [50, 51]	Partitionnement binaire de l'espace et duplication des bordures	Clustering parallèle et fusion des clusters locaux en s'appuyant sur les points bordures	Création de partitions de tailles égales	Ne s'applique que sur les données qui peuvent être représentées sur un espace à n-dimension
DBSCAN parallèle en utilisant les $SEED$ [48]	Partitionnement aléatoire	Ordonnancement des données après leur clustering afin d'identifier les $SEED$ utilisés pour la fusion des clusters locaux	Partitionnement rapide et création de partitions de tailles égales	L'ordonnancement des entités limite son utilisation sur les données catégorielles
S-DBSCAN [73]	Partitionnement aléatoire	Clustering parallèle et fusion des clusters en s'appuyant sur leurs centres	Partitionnement et clustering rapide	Ne garantie pas de fournir le même résultat que DBSCAN
DBSCAN parallèle en utilisant MPI [95]	Partitionnement aléatoire	Clustering parallèle et fusion des clusters en s'appuyant sur leurs centres et leurs rayons	Partitionnement et clustering rapide	Ne garantie pas de fournir le même résultat que DBSCAN
HPDBSCAN [42]	Partitionnement en cellules en utilisant des techniques de division de l'espace	• Clustering parallèle • Recherche de voisinage accéléré	L'ordonnancement des entités améliore la recherche de voisinage	Application non triviale sur les données catégorielles
NG-DBSCAN [72]	Création aléatoire de graphes d'entités	Construction de l'ϵ-graphe pour construire les clusters	Compare uniquement les entités dans un même graphe afin d'offrir un processus rapide	Algorithme probabiliste qui ne fournit pas le même résultat que DBSCAN
RP-DBSCAN [100]	Division des données en cellules et distribution aléatoire	• Clustering en comparant les centres des cellules • Fusion des clusters locaux en utilisant un index sur les cellules	La recherche de voisinage en utilisant les centres offre un algorithme rapide	Ne garantit pas de fournir le même résultat que DBSCAN

de moins bonne qualité. En effet, le voisinage des entités est calculé dans chaque partition (sous-ensemble de données) et les voisins dans d'autres partitions ne sont pas découverts. Par conséquent, le résultat du clustering produit n'est pas le même que celui produit par l'algorithme DBSCAN, sauf pour PDSDBSCAN [84] ; mais cette approche exige une communication entre partitions.

- Les approches qui nécessitent un échange d'information entre les nœuds de calcul introduisent un surcoût de communication très important [84]. En effet, le transfert de données entre les différents nœuds de calculs qui composent un système distribué représente une opération coûteuse.
- Les approches qui utilisent les centres des clusters locaux pour reconstuire les clusters finaux ne fournissent pas le même résultat que l'algorithme DBSCAN et ainsi réduisent la qualité du clustering. En effet, la fusion dépend fortement du choix du centre de chaque cluster et de la représentativité de ce centre par rapport au reste des entités. De plus, les algorithmes qui partitionnent les données en utilisant des techniques de division d'espace comme le partitionnement binaire de l'espace (BSP) [37] ne s'appliquent pas aux données RDF. Ces techniques perdent leur efficacité lorsqu'elles sont utilisées sur les données de grande dimensionnalité comme les données RDF qui sont décrites par un nombre important de propriétés [50, 51, 22, 42, 100].

Le problème est le même pour les approches qui utilisent l'ordonnancement des données afin de les distribuer [48]. En effet, les données RDF sont de type catégoriel et ne peuvent pas être ordonnées.

- Enfin, les approches qui proposent un traitement probabiliste afin d'améliorer les performances [72] ne produisent pas le même résultat que l'algorithme DBSCAN et, dans notre contexte, cela réduit la qualité du schéma produit. Des travaux existants ont en effet montré qu'un schéma produit par l'algorithme DBSCAN est de bonne qualité en considérant le rappel et la précision de ses classes [65, 66]

En conclusion à l'analyse des extensions scalables de l'algorithme DBSCAN, nous pouvons voir qu'aucune des approches étudiées ne propose une solution parallèle qui s'applique aux données RDF et qui produit le même résultat que l'algorithme DBSCAN séquentiel. En effet, certaines de ces approches utilisent des méthodes de partitionnement d'espace qui ne s'appliquent pas aux données RDF [50, 51, 22, 42, 100], d'autres ne produisent pas les mêmes clusters que l'algorithme DBSCAN [72, 73, 95] et enfin certaines peuvent introduire un surcoût important [84].

Nous nous intéressons également dans notre travail à la mise à jour du schéma après son extraction. Dans ce but, nous avons étudié les algorithmes DBSCAN incrémentaux qui construisent de façon évolutive les clusters. Dans notre contexte, cela permettrait de modifier les classes du schéma extrait à partir d'une source de données RDF afin de garder la cohérence avec le jeu de données lorsque ce dernier est modifié. Notre étude est synthétisée dans le tableau 2.4.

TABLE 2.4 – Synthèse des algorithmes DBSCAN incrémentaux

Approches	**Méthode de partitionnement**	**Principe de clustering**	**Avantages**	**Inconvénients**
DBSCAN incrémentale initial [29]	/	Clustering du voisinage des entités ajoutées ou supprimées	Produit le même résultat que l'algorithme DBSCAN appliqué sur la totalité des données	Méthode non distribuée
DBSCAN incrémental distribué [75]	Partitionnement en partant de centres aléatoires qui sont ensuite optimisés jusqu'à trouver les centres de partitions les plus représentatifs.	La recherche de voisinage d'une entité insérée ou supprimée est restreintes à certaines partitions de données plutôt qu'à la totalité du jeu de données	Le partitionnement des données accélère la recherche de voisinage et ainsi le clustering.	Ne garantie pas de fournir le même résultat que DBSCAN
DBSCAN incrémental qui s'appuie sur les cores des clusters [18]	/	Comparaison des nouvelles entités uniquement aux entités cores dans les anciens clusters	Le nombre des cores étant plus petit que le nombre d'entités total, le clustering est plus rapide.	Ne garantie pas de fournir le même résultat que DBSCAN
RT-DBSCAN [41]	Partitionnement de l'espace de données en utilisant le *Fast Clustering partitioning*	Clustering parallèle des données et la fusion des résultats en comparant les régions bordures des partitions	La parallélisation des données offre un processus scalable.	La méthode de partitionnement des données ne s'applique pas aux données RDF à cause de leur nature

Cependant, les algorithmes DBSCAN incrémentaux abordés dans cet état de l'art présentent des limites :

- L'algorithme [29] propose un ensemble de règles à appliquer sur le voisinage des entités insérées ou supprimées afin d'adapter les clusters aux jeux de données, mais pour définir le voisinage d'une entité, l'algorithme la compare avec toutes les autres entités. De plus, l'algorithme est séquentiel ce qui rend le calcul de voisinage coûteux.
- Dans [75], les auteurs proposent d'optimiser la recherche de voisinage. Cependant, leur méthode de fusion

s'applique aux jeux de données qui peuvent être représentés dans un espace à n-dimensions. L'utilisation de cette méthode pour des données RDF réduirait considérablement la qualité des clusters construits et ainsi la qualité du schéma. Le même problème est observé pour l'approche [18] qui ne compare les nouvelles entités qu'avec les cores des clusters.

- RT DBSCAN [41] divise l'espace des données afin de paralléliser l'exécution et ne s'applique pas aux données RDF dont les propriétés sont de type catégoriel.

En conclusion, les algorithmes DBSCAN incrémentaux présentés dans cet état de l'art ne peuvent pas être directement utilisés pour la découverte incrémentale de schéma à partir de sources de données RDF massives. En effet, les approches qui ne proposent pas un traitement parallèle sont coûteuses et ne s'appliquent pas aux grands jeux de données. La comparaison des nouvelles entités avec quelques anciennes ne garantit pas de fournir un schéma de bonne qualité. Enfin, certaines des techniques utilisées pour la division de l'espace ne s'appliquent pas aux données RDF.

2.6 Conclusion

Le besoin d'un schéma décrivant les sources de données RDF afin de faciliter leur exploitation a motivé plusieurs travaux de recherche et des méthodes d'extraction de schéma ont été proposées.

Dans cet état de l'art, nous avons présenté les approches relatives à l'extraction de schéma à partir de données irrégulières. Nous avons d'abord étudié les approches qui extraient un schéma à partir de l'analyse structurelle des instances contenues dans une source de données, afin de regrouper celles qui représentent des instances de la même classe et ainsi former le schéma. Nous avons également présenté des approches qui proposent de découvrir les patterns représentant les versions structurelles possibles des instances d'un jeu de données RDF.

Les approches de découverte de schéma visent à identifier les entités partageant des propriétés communes afin de former des groupes d'entités structurellement similaires. Certaines de ces approches regroupent les instances selon leur similarité structurelle en utilisant des algorithmes de clustering ou des techniques d'analyse formelle. D'autres approches proposent le regroupement des chemins similaires dans un graphe de données afin de proposer un schéma sous forme de dataguide, exact ou approximatif, qui offre une vue globale sur les instances de la source de données. Les approches de découverte de schéma existantes présentent des limites. Celles qui utilisent des algorithmes de clustering ne s'appliquent pas sur les jeux de données de grande taille à cause de leur complexité. De plus, ces algorithmes ne sont pas incrémentaux et nécessitent pour garder le schéma cohérent avec les données, une ré-exécution du processus d'extraction de schéma sur la totalité du jeu de données. Les dataguides exacts ont pour but de représenter tous les chemins du graphe initial, leur taille peut donc être très grande. Pour pallier ce problème, le dataguide approximatif offre une représentation plus concise construite en utilisant l'algorithme COBWEB. Cependant, cet algorithme n'est pas déterministe.

Il existe des approches connexes au problème de découverte de schéma qui enrichissent la description existante avec de nouvelles déclarations. Parmi ces approches nous citons les approches d'enrichissement de schéma qui utilisent des algorithmes de fouille de données pour inférer de nouvelles déclarations schéma. Enfin, les approches de typage d'entités utilisent des techniques d'analyse statistique ou des algorithmes de machine learning afin d'attribuer un type aux entités pour lesquelles cette information est manquante. Ces approches ne découvrent pas un schéma ou les structures qui décrivent les instances d'un jeu de données, mais complètent un schéma déjà existant.

Dans notre étude, nous avons pu constater qu'aucune des approches de découverte de schéma existantes ne considère les propriétés implicites qui peuvent être dérivées par inférence, et qui font partie intégrante de la sémantique portée par une source de données RDF.

Nous avons également étudié un ensemble d'approches de découverte de patterns structurels, exacts ou approximatifs à partir de sources de données irrégulières. Ces approches s'appuient sur certaines déclarations du schéma afin d'identifier et de regrouper les entités similaires et de former les classes du schéma. Cependant, ces approches font l'hypothèse que les déclarations schéma sont fournies dans une source de données RDF, et par conséquent, ne s'appliquent pas sur les jeux de données RDF dont les déclarations schéma ne sont pas disponibles.

Dans notre travail, nous nous sommes intéressés à la découverte de schéma à partir de données RDF sans faire l'hypothèse de l'existence d'informations sur le schéma même partiellement, et en procédant par regroupement d'entités structurellement similaires.

Notre travail est axé sur la découverte de schéma par analyse structurelle des données en utilisant les algorithmes de clustering. Ceci nous a amené à étudier les algorithmes de clustering qui peuvent être utilisés pour l'extraction de schéma. Après avoir déterminé les critères exigés par les données RDF dans le choix de l'algorithme de clustering, nous avons montré que DBSCAN est l'algorithme de clustering le plus adapté aux des données RDF. Cependant, la complexité de cet algorithme limite son utilisation sur les grands jeux de données. De plus, il n'est pas incrémental et ne permettrait pas de faire évoluer le schéma pour prendre en compte l'ajout de données dans la source. Dans la deuxième partie de cet état de l'art nous avons présenté des propositions d'algorithmes DBSCAN scalables et incrémentaux.

Ces algorithmes tentent de distribuer l'exécution de DBSCAN afin de paralléliser les calculs et ainsi obtenir un processus plus rapide. Cependant, les algorithmes DBSCAN scalables existants présentent des limites lors de leurs utilisations sur les données RDF massives dans le but d'extraire un schéma. Ceux qui utilisent une distribution aléatoire des données réduisent la qualité du schéma résultant. Le même problème se pose pour les algorithmes probabilistes et ceux qui utilisent les centres pour construire le clustering final. D'autres approches partitionnent les données en utilisant des méthodes de division d'espace. Ces techniques ne s'appliquent pas sur les données RDF

qui ne sont pas facilement représentées dans un espace à n-dimensions.

Pour conclure, l'étude des approches liées au problème d'extraction d'informations sur le schéma à partir des données RDF massives nous a permet d'identifier les limites suivantes :

- Les approches de découverte de schéma existantes utilisent des algorithmes de clustering coûteux qui limitent leur utilisation sur des données massives. Les approches qui ont été proposées dans un contexte Big Data se basent sur certaines déclarations schéma, qui ne sont pas toujours fournies. Cependant, il n'existe pas dans cette catégorie d'approches scalables qui ne s'appuient pas sur des déclarations schéma pré-existantes. divisiondivision
- Les approches existantes ne sont pas incrémentales et nécessitent la disponibilité de toutes les données afin d'extraire le schéma. Certaines approches proposent une classification des nouvelles entités après l'extraction de schéma mais ne remettent pas en cause le schéma existant et ainsi l'écart de ce dernier avec les données peut devenir important. Par conséquent, la découverte incrémentale de schéma pour des données RDF qui évoluent dans le temps reste un problème ouvert.
- Les approches existantes se basent sur l'analyse des propriétés explicites dans une source de données RDF pour proposer un schéma décrivant les entités de la source. Or la sémantique d'une source de données RDF comporte non seulement les triplets existants, mais également tous ceux qui peuvent en être déduits par inférence, que l'on désigne par triplets implicites. Ces derniers ne sont pris en compte par aucune des approches existantes, ce qui revient à ignorer une partie de la sémantique portée par la source lors de la découverte de schéma. La prise en compte des triplets implicites dans ce processus est un problème ouvert.

Dans la suite de ce manuscrit, nous introduirons nos contributions pour pallier aux limites identifiées ci-dessus. Nous présenterons dans le chapitre 3 une solution au problème de scalabilité de la découverte de schéma à partir des grandes sources de données RDF. Nous détaillerons dans le chapitre 4 notre approche incrémentale de découverte de schéma qui met à jour le schéma décrivant une source de données RDF lorsque cette dernière évolue en ajoutant des nouvelles entités. Nous présenterons dans le chapitre 5 une contribution vers une approche hybride de découverte de schéma qui exploite les déclarations liées au schéma afin de dériver les propriétés implicites décrivant les entités et de les prendre en compte durant l'extraction des classes du schéma.

Chapitre 3

Découverte de schéma à partir des sources massives de données RDF

3.1 Introduction

Nous assistons à une prolifération de sources de données faiblement structurées, irrégulières et massives comme les données du web sémantique, décrites par des langages proposés par le W3C tel que RDF [1]. La différence principale de ces sources de données par rapport aux données structurées est qu'elles ne suivent aucun schéma prédéfini.

Une source de données RDF comporte à la fois les données, et les déclarations sur le schéma décrivant ces données. Les déclarations sur le schéma ne représentent pas des contraintes sur la structure des données mais aident l'utilisateur à comprendre le contenu de la source. Cependant, la nature flexible de ces langages permet la création d'une source sans aucune obligation de fournir des déclarations sur le schéma. Par conséquent, le schéma est souvent incomplet ou absent, et même s'il existe, les données ne sont pas contraintes de le respecter.

L'absence de schéma offre une grande flexibilité lors de la création de sources de données RDF, mais limite leur utilisation. En effet, l'exploitation ou l'interrogation d'une source de données est difficile en l'absence de connaissances sur cette source, sur les classes qu'elle comporte et les propriétés décrivant ces classes. L'exploitation des données RDF serait plus facile avec un schéma décrivant les données et apportant des méta-informations sur son contenu. C'est ce constat qui a fait émerger le besoin de caractériser et de comprendre le contenu des sources de données RDF, et qui a donné lieu aux approches automatiques de découverte de schéma. Dans le contexte du web des données, un schéma est vu comme un guide facilitant l'exploitation des sources de données RDF. Il renseigne sur le contenu de la source, mais ne pose pas de contrainte sur la façon dont les données sont décrites, comme

1. RDF : `https://www.w3.org/RDF/`

dans le cas des bases de données relationnelles par exemple.

Dans ce chapitre, nous nous intéressons au problème de découverte de schéma à partir de sources de données RDF massives. Certaines approches ont été proposées pour résoudre ce problème. Elles ne nécessitent pas de connaissances préalable mais explorent la structure des instances contenues dans une source de données afin d'en inférer le schéma. La structure d'une entité représente l'ensemble de propriétés qui la décrivent. Elles utilisent des algorithmes de clustering ou encore l'analyse formelle de concepts et fournissent un schéma de bonne qualité. Mais elles sont limitées par le coût des techniques utilisées et leur capacité à gérer de grandes sources de données reste un problème ouvert [21, 65, 66, 69, 14, 81, 80].

Notre thèse se situe dans la continuité des travaux de découverte de schéma utilisant un algorithme de clustering basé sur la densité pour découvrir le schéma d'une source de données RDF à partir de la structure explicite de ses entités ; le schéma produit est composé de classes et de liens entre elles [64]. Pour découvrir les classes, les auteurs proposent d'adapter l'algorithme de clustering DBSCAN afin d'identifier les clusters d'entités similaires qui représentent des classes.

Cette approche, comme celles qui utilisent le clustering pour la découverte de schéma [21, 20, 65, 66] produisent un schéma de bonne qualité, mais leur scalabilité demeure un problème ouvert car elles utilisent des algorithmes de clustering coûteux en temps d'exécution. Le même problème se pose pour les approches qui utilisent les techniques d'analyse formelle et les treillis de concepts [69, 14, 81, 80] qui sont des techniques coûteuses en temps d'exécution. Par conséquent, leur utilisation dans le but de découvrir le schéma implicite à partir de sources de données massives reste un défi. Enfin, les approches qui supposent l'existence préalable de déclarations liées au schéma pour en générer de nouvelles ne s'appliquent pas sur les sources de données dont le schéma est absent ou partiellement fourni.

Dans ce chapitre, nous nous intéressons au problème de scalabilité des approches de découverte de schéma à partir des sources de données pour lesquelles ce schéma est absent ou partiellement défini. Notre objectif est de proposer une approche scalable, adaptée aux grandes sources de données RDF sur lesquelles les approches existantes ne peuvent pas s'appliquer en raison de leur complexité.

Afin de découvrir le schéma d'une source de données RDF, nous proposons une approche qui découvre la structure implicite des données en regroupant les instances selon leur similarité structurelle pour définir les classes. Dans ce but, nous avons exploré deux pistes complémentaires. Nous introduisons tout d'abord dans ce chapitre une méthode qui génère une représentation condensée de la source de données, afin de réduire le nombre de points en entrée de l'algorithme de clustering. Nous présentons ensuite un algorithme de clustering basé sur la densité qui est scalable, et spécifiquement conçu pour la découverte de schéma à partir des grandes sources de données RDF. Notre approche parallélise le processus de clustering afin d'optimiser son exécution tout en assurant que le résultat est le même que celui produit par une exécution séquentielle de DBSCAN. Les principales contributions de ce chapitre sont :

- Une méthode d'extraction d'une représentation condensée d'une source RDF, qui comporte toutes les combinaisons de propriétés décrivant les entités existantes. Cette représentation sera utilisée pour la découverte de schéma.
- Un principe de distribution original qui divise le jeu de données initial en sous-ensembles qui peuvent être traités efficacement en parallèle, ainsi qu'une optimisation de cette méthode qui permet de réduire la taille des sous-ensembles, ce qui entraîne la limitation du nombre de comparaisons entre les entités durant le clustering, et donc l'accélération du processus.
- Un algorithme de clustering parallèle adapté à un environnement de calcul distribué qui limite les échanges d'informations entre les nœuds de calcul. Ces échanges représentent les opérations les plus coûteuses dans une plateforme de calcul distribué.
- Une implémentation scalable de notre algorithme en utilisant la plateforme de calcul distribué Spark [104], dont le code source est disponible en ligne [2].

Notre algorithme de clustering est inspiré de DBSCAN [28], qui répond aux critères imposés par la nature des données RDF, principalement car il produit des clusters de formes arbitraires. Cette caractéristique est importante dans notre contexte où les instances d'une même classe peuvent être décrites par des ensembles de propriétés hétérogènes. De plus, il ne nécessite pas la connaissance à priori du nombre de classes contenues dans une source de données, et détecte les entités bruits, qui ne sont pas assez importantes pour donner lieu à la création d'une classe.

Ce chapitre est organisé comme suit. Nous définissons dans la section 3.2 les concepts et notions nécessaires à la compréhension de notre approche. La problématique adressée dans ce chapitre est décrite dans la section 3.3. La section 3.4 présente la méthode d'extraction d'une représentation condensée à partir d'une source de données RDF. Nous introduisons notre approche scalable de découverte de schéma dans la section 3.5, où nous donnons une vue globale de notre proposition. Les sections qui suivent décrivent notre processus scalable de découverte de schéma. Ainsi, la distribution des données est détaillée dans la section 3.6, et l'identification des entités cores est décrite en section 3.7. La section 3.8 présente le processus de clustering local et la section 3.9 décrit l'étape de fusion qui produit le clustering final permettant de construire les classes du schéma. Enfin, nous concluons le chapitre en section 3.11 en présentant un bilan de notre approche scalable de découverte de schéma.

3.2 Préliminaires

Avant d'aborder le problème de découverte de schéma à partir de données RDF massives et de décrire notre solution, nous présentons dans ce qui suit quelques notions et définitions préliminaires.

2. https://github.com/BOUHAMOUM/SC-DBSCAN

Source de données RDF

Une source de données du web D est un ensemble de triplets RDF(S)/OWL $D \subseteq (R \cup B) \times P \times (R \cup B \cup L)$, tel que R, B, P et L représentent respectivement une ressource, une ressource anonyme, des propriétés et des littéraux. Un triplet est de la forme $\langle s, p, o \rangle$, où s est le sujet qui représente une ressource, p une propriété décrivant cette ressource, et o l'objet, ou la valeur de la propriété p pour la ressource s ; o peut être une ressource ou un littéral. Dans une telle source de données, nous considérons une entité comme une ressource ou un nœud anonyme, en d'autres termes, tout nœud dans le graphe correspondant à une ressource et non à un littéral.

Dans une source de données décrite en RDF(S)/OWL, nous trouvons à la fois les données et les déclarations liées au schéma. Des exemples de telles déclarations sont $rdf{:}type$ qui spécifie le type d'une entité, ou encore $rdfs{:}range$ et $rdfs{:}domain$ qui spécifient le domaine et le co-domaine d'une propriété. Le domaine et le co-domaine représentent respectivement le sujet et l'objet d'une propriété dans un triplet RDF. Une source de données peut être représentée par un graphe orienté et étiqueté, où chaque nœud est une ressource, une ressource anonyme ou un littéral. Chaque arc étiqueté par la propriété p dans le graphe et qui relie un nœud e_1 à un nœud e_2 représente un triplet (e_1, p, e_2) dans le jeu de données D.

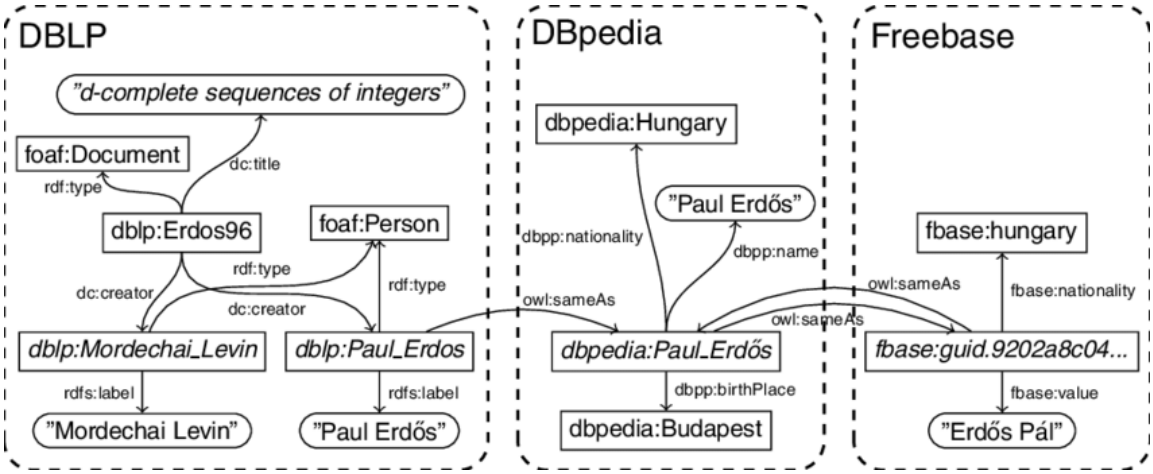


FIGURE 3.1 – Exemple de sources de données RDF interconnectées

Les informations sur le schéma peuvent être fournies par l'utilisateur, comme on peut les trouver dans des définitions de concepts importées à partir de sources externes. Ces déclarations nous apportent des informations sur les instances contenues dans une source de données RDF.

La figure 3.1 présente un exemple de source de données du web ; nous remarquons que la source contient les ressources "$dblp{:}Mordechai_Levin$" et $dblp{:}Paul_Erdos$ et des déclarations sur le schéma, comme par exemple les déclarations de classes $foaf{:}Person$ et $foaf{:}Document$. Chaque ressource est décrite par des propriétés. Par exemple, $dbpp{:}name$ et $dbpp{:}birthPlace$ décrivent la ressource "$dbpedia{:}Paul_Erdos$". Certaines ressources possèdent la propriété $rdf{:}type$ qui détermine le type de la ressource. C'est le cas de la ressource "$dblp{:}Erdos96$"

qui est de type $foaf{:}Person$. Mais on peut voir que ces déclarations ne sont pas fournies pour toutes les ressources, par exemple, nous ignorons le type de la ressource "$dbpedia{:}Paul_Erdos$". De plus, deux ressources de même type ne sont pas nécessairement décrites par le même ensemble de propriétés, comme on peut le voir pour "$dblp{:}Mordechai_Levin$" et "$dblp{:}Paul_Erdos$", qui sont toutes le deux de type $foaf{:}Person$ mais la ressource "$dblp{:}Paul_Erdos$" est décrite par la propriété $owl{:}sameAs$, ce qui n'est pas le cas de la ressource "$dblp{:}Mordechai_Levin$".

Nous introduisons une fonction notée $\overline{}$ qui retourne l'ensemble des propriétés qui décrivent une entité :

$$\begin{array}{rcl} \overline{} : \mathcal{R} \cup \mathcal{B} & \rightarrow & \mathcal{P} \\ e & \mapsto & \{p \in \mathcal{P} \mid \langle e, p, o \rangle \in D\} \end{array}$$

Par exemple, la ressource "$dbpedia : paul_Erdos$" représente une entité e décrite par l'ensemble de propriétés $\overline{e} = \{dbpp : name, dbpp : nationality, dbpp : birthPlace\}$.

Schéma d'une source de données RDF

Un schéma offre une vue synthétique d'une source de données en termes de classes et de liens entre chaque classe. Nous définissons un schéma comme suit.

Définition 3.2.1 *Un schéma S décrivant une source de données D est composé d'un ensemble de classes $C = \{C_1, C_2, \ldots, C_n\}$, où chaque classe C_i est décrite par un ensemble de propriétés $\{p_1^i, \ldots, p_m^i\}$ et représente un ensemble d'entités. De plus, le schéma contient l'ensemble de liens $\{p_1, p_2, \ldots, p_m\}$ entre les classes, où chaque p_i est une propriété pour laquelle le co-domaine et le domaine correspondent à des classes dans l'ensemble C.*

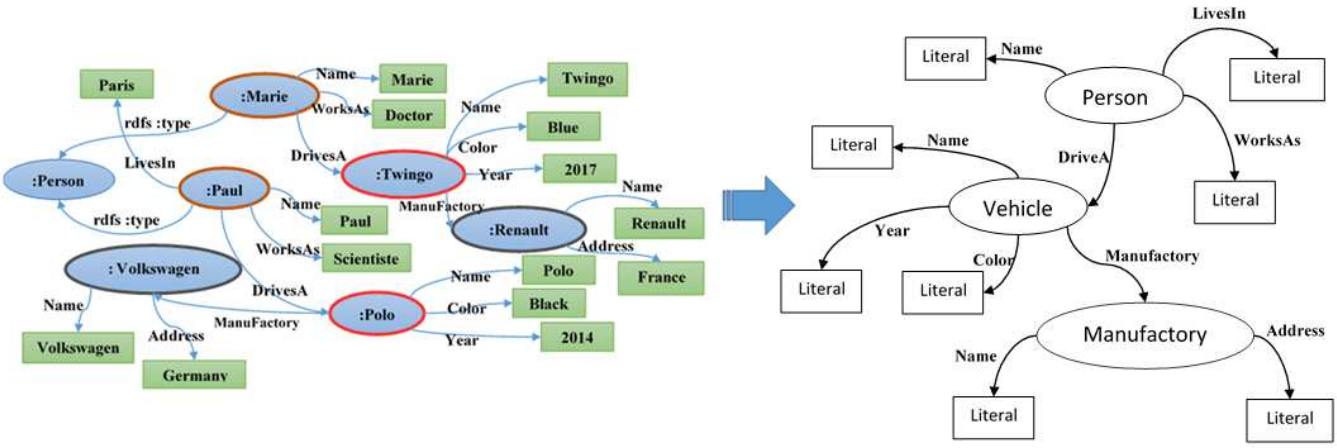


FIGURE 3.2 – Exemple de jeu de données RDF et le schéma correspondant

Exemple 3.2.1 *Dans la figure 3.2, le schéma associé au jeu de données indique que la source comporte des instances des classes $Person$, $Vehicle$ et $Manufacture$, ainsi que leurs structures.*

Similairement au concept de classes dans la modélisation des données, une classe dans une source de données RDF représente un ensemble d'entités ayant le même type, et ayant des propriétés communes. Ces entités sont des instances de cette classe. L'objectif de notre travail est de découvrir les ensembles d'entités structurellement similaires afin de reconstituer les classes correspondant à ces groupes d'entités et de générer le schéma implicite.

La similarité entre les entités peut être évaluée en utilisant un indice quelconque qui mesure la similarité entre des ensembles finis comme *l'indice de Sørensen–Dice* [43], *l'indice Overlap* [5] et *l'indice de Jaccard* [56]. Dans notre contexte, les propriétés décrivant les entités représentent des ensembles finis. Deux entités sont similaires si elles partagent un nombre de propriétés supérieur ou égal à un seuil donné. Dans notre travail, nous évaluons la similarité entre deux entités e_i et e_j en utilisant *l'indice de Jaccard*, qui représente le rapport entre la cardinalité de l'intersection des ensembles de propriétés respectifs de e_i et e_j et la cardinalité de leur union [56] :

$$J(e_i, e_j) = \frac{|\overline{e_i} \cap \overline{e_j}|}{|\overline{e_i} \cup \overline{e_j}|}$$

La valeur de similarité est comprise entre 0 et 1. Deux entités e_1 et e_2 sont similaires si $J(e_i, e_j) \geq \epsilon$, où ϵ est le paramètre qui définit le seuil de similarité. *L'indice de Jaccard* a été utilisé par plusieurs approches de découverte de schéma [21, 65, 66], conduisant à un schéma de bonne qualité.

La figure 3.2 représente un exemple de source de données RDF et le schéma correspondant. Les classes $Person$, $Vehicle$ et $Manufacture$ représentent respectivement des ensembles d'entités qui partagent des propriétés similaires, $C_1 = \{: Paul, : Marie\}$, $C_2 = \{: Twingo, : Polo\}$ et $C_3 = \{: Renault, : Volkswagen\}$. Les propriétés décrivant ces classes sont l'union des propriétés décrivant des entités similaires.

3.3 Problématique

Nous nous intéressons dans notre travail à la découverte de schéma à partir des sources de données RDF dont le schéma est absent. Nous adressons plus spécifiquement le problème de scalabilité des approches de découverte de schéma existantes : étant donné une source de données de grande taille décrite en RDF(S)/OWL où les déclarations sur le schéma sont incomplètes ou absentes, et où les approches d'extraction de schéma existantes échouent dans le traitement du volume de ces données en raison de leur complexité, comment extraire les classes qui constituent le schéma représentant les entités contenues dans cette source ?

En étudiant les approches de l'état de l'art, nous avons identifié deux catégories d'approches afin d'extraire le schéma décrivant une source de données RDF. Tout d'abord, les approches qui analysent la structure des données afin d'identifier les entités similaires, et qui n'exigent pas la présence de déclarations liées au schéma dans la source [20, 21, 65, 66, 19, 69, 14, 81, 80, 39, 96, 109]. La limite de ces approches est la complexité des algorithmes qu'elles

utilisent : elles sont fondées sur un regroupement par clustering qui nécessite généralement une comparaison de toutes les paires d'instances. Par conséquent, leur utilisation dans un contexte big data est coûteuse.

Les approches de la deuxième catégorie requièrent certaines déclarations sur le schéma afin d'extraire un ensemble de patterns représentant les entités du jeu de données [10, 94, 15, 8, 9, 16]. Le problème de ces approches est que les déclarations sur lesquelles elles s'appuient sont souvent manquantes ou incomplètes. En l'absence de ces déclarations les instances décrites par exactement les mêmes propriétés sont regroupées. Mais les patterns produits ne représentent pas nécessairement les classes décrivant les entités de la source de données, car dans une source RDF, les entités de la même classe ne sont pas nécessairement décrites par des ensembles de propriétés identiques.

Les déclarations sur le schéma étant souvent incomplètes ou absentes, nous proposons dans notre travail de découvrir le schéma implicite à partir des ensembles d'entités structurellement similaires, sans nous appuyer sur les déclarations du schéma dans la source de données. Notre approche est fondée sur un algorithme de clustering afin de former ces groupes d'entités similaires.

Exemple 3.3.1 *Dans l'exemple présenté dans la figure 3.2, l'algorithme de clustering doit former des clusters représentant les classes du schéma. Par exemple, le cluster formé à partir des entités* $:Twingo$ *et* $:Polo$ *représente la classe* $Vehicle$*.Ces entités sont structurellement similaires même si leur déclaration de type est absente.*

Parmi les algorithmes de clustering, ceux basés sur la densité sont les plus adaptés pour le clustering des sources de données du web, car ils permettent la formation de clusters de formes arbitraires. Ceci correspond aux ressources du web qui peuvent avoir le même type tout en étant décrits par des propriétés différentes. De plus, ces algorithmes sont déterministes, robustes au bruit et n'exigent pas le nombre de clusters à priori. Enfin, ces algorithmes ont été efficacement adoptés dans la découverte de schéma à partir des sources de données du web et ont fourni des résultats de bonne qualité [21, 20, 65, 66]. Le problème de ces algorithmes est leur complexité de calcul qui rend impossible leur utilisation dans notre contexte, celui du traitement de grandes sources de données du web.

Certaines approches ont utilisé des technologies big data telles que Hadoop et Spark afin de proposer des algorithmes de clustering parallèles et scalables. La parallélisation du clustering soulève les problèmes suivants :

- Comment et sur quel critère partitionner les données sur les nœuds de calcul afin de créer des sous-ensembles de données qui peuvent être traités rapidement ?
- Comment construire les clusters sur chaque sous-ensemble de données tout en limitant les échanges d'informations entre les différents nœuds de calcul, sachant que les voisins d'une entité peuvent être distribués sur plusieurs sous-ensembles ?
- Comment assurer que le résultat du clustering produit par l'exécution parallèle est le même que celui produit par l'algorithme de clustering séquentiel ?

Nous introduisons dans les sections suivantes nos propositions pour le passage à l'échelle de la découverte de schéma. Nous détaillons d'abord notre approche pour réduire la taille d'un jeu de données RDF. Nous présentons ensuite notre approche scalable de découverte de schéma qui s'inspire des algorithmes de clustering basés sur la densité et qui permet de traiter des sources de données RDF de grande taille tout en fournissant un schéma de bonne qualité.

3.4 Extraction d'une représentation condensée d'une source RDF

Une solution au problème de la scalabilité des approches de découverte de schéma est de réduire la taille du jeu de données sur lequel le clustering sera appliqué. Dans ce but, nous proposons dans cette section une approche qui transforme le jeu de données initial en une représentation plus condensée afin de réduire sa taille. Cette représentation condensée est composée de patterns, chacun correspondant à une combinaison de propriétés qui décrit au moins une entité dans la source de données. Des entités ayant exactement les mêmes propriétés seront représentées par un pattern unique.

Étant donnée une source de données RDF, l'extraction de patterns consiste à construire une représentation plus concise des données initiales, en identifiant toutes les combinaisons existantes de propriétés décrivant les entités. L'extraction de patterns prend en entrée les triplets de la source de données considérée et produit en sortie un ensemble de patterns, qui donne une représentation concise de la source de données initiale. Un pattern est défini comme suit :

Définition 3.4.1 *Un pattern pt est un ensemble de propriétés distinctes tel qu'il existe au moins une entité décrite par cet ensemble de propriétés. À chaque pattern pt est associé un nombre $|pt|$ qui représente le nombre d'entités décrites par la structure de ce pattern. Comme pour les entités, la fonction $\overline{}$ appliquée à un pattern retourne l'ensemble de propriétés le décrivant.*

Exemple 3.4.1 *Considérons le jeu de données RDF de la figure 3.3 qui représente des informations sur un groupe de personnes, comme leur nom, leur profession, leurs liens familiaux, etc.*

La représentation condensée de ce jeu de données est donnée dans le tableau 3.1, qui montre les patterns extraits et les entités correspondant à chaque pattern. Chaque ligne du tableau représente respectivement les entités ayant la même structure et le pattern correspondant. Un pattern est désigné par (E, Nb), où E représente l'ensemble des propriétés décrivant le pattern, et Nb représente le nombre d'entités décrites par E.

Dans la suite, les propriétés décrivant les entités de la figure 3.3 seront représentées par les abréviations suivantes : Address (adr), Color (clr), Country (Ctry), DriveA (DrA), HasChild (HsC), HasHusband (HsH), HasWife (HsW), LivesIn (LIn), Manufacture (Mnf), Name (Nm), Region (Rg), School (Scl), WorksAs (Wk), Year (Yr).

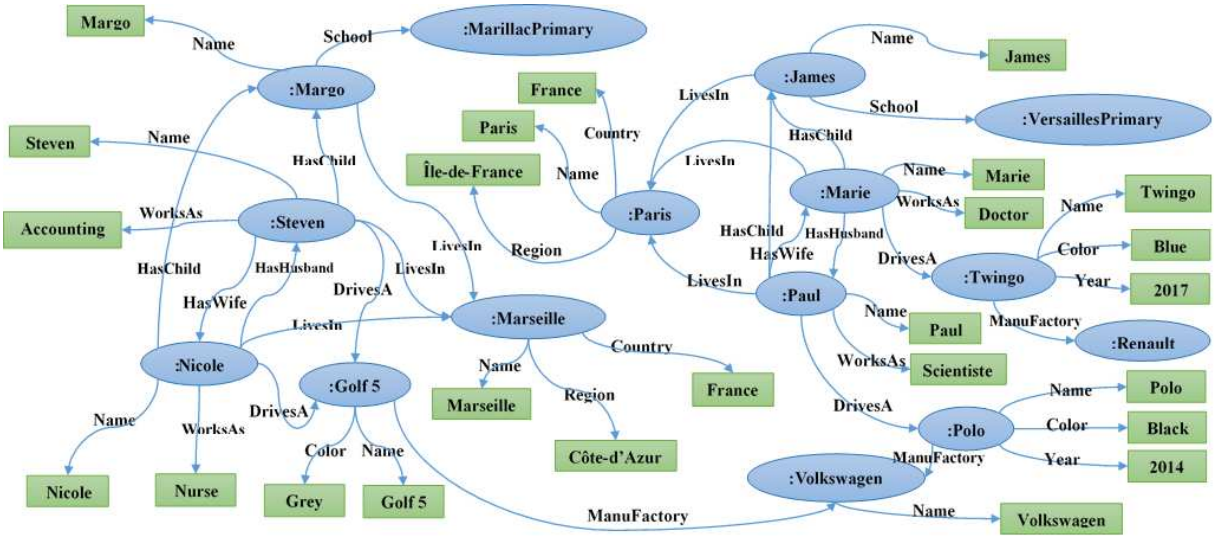


FIGURE 3.3 – Exemple de jeu de données RDF

TABLE 3.1 – Les patterns extraits à partir du jeu de données de la figure 3.3

Entities	**Pattern**
" :Marie", "Nicole"	$P1 = (\{Nm, Wk, DrA, HsH, LIn, HsC, \}, 2)$
" :Paul", " :Steven"	$P2 = (\{Nm, Wk, DrA, HsW, LIn, HsC, \}, 2)$
" :James", " :Margo"	$P3 = (\{Nm, LIn, Scl\}, 2)$
" :Polo", " :Twingo", " :Golf"	$P4 = (\{Nm, Clr, Yr, Mnf, \}, 3)$
" :Renault"	$P5 = (\{Nm, Ctry\}, 1)$
" :Volkswagen"	$P6 = (\{Nm\}, 1)$
" :Paris", " :Marseille"	$P7 = (\{Nm, Ctry, Rg\}, 2)$
" :VersaillesPrimary", " :MarseillePrimary"	$P8 = (\{Nm, adr\}, 2)$

L'identification des patterns à partir d'une source de données produit toutes les combinaisons de propriétés qui existent dans un jeu de données RDF. Le nombre de patterns étant plus réduit que le nombre d'entités initiales, exécuter le clustering sur les patterns au lieu de le faire sur les entités initiales lors de l'extraction de schéma donne lieu à une exécution plus rapide, sans dégrader la qualité du résultat produit. En effet, la découverte de schéma par clustering regroupe les entités par rapport à leur similarité structurelle : les entités formant un même cluster sont celles partageant des propriétés communes. Les patterns préservent toutes les structures décrivant les entités, ce qui garantit que l'exécution du clustering sur les patterns produit le même résultat que son application sur les entités.

Cependant, il est nécessaire d'adapter la définition de densité aux patterns. Ainsi, nous associons à chaque pattern le nombre d'entités décrites par ce pattern afin que cela soit pris en compte durant le calcul de voisinage et l'identification des entités ayant un voisinage dense. En effet, un pattern représente un ensemble d'entités, et si deux patterns pt_1 et pt_2 sont similaires, ceci implique que les entités représentées par pt_1 et pt_2 sont voisines.

Par conséquent, si nous voulons exécuter le clustering sur les patterns, la densité de voisinage de chaque pattern doit être évaluée en considérant le nombre d'entités dans ce voisinage.

Exemple 3.4.2 *Dans notre exemple (tableau 3.1), la taille du jeu de données a été réduite à* 53% *car nous avons obtenu* 8 *patterns décrivant les* 15 *entités en entrée. Ainsi, appliquer une approche de découverte de schéma sur les patterns serait un processus plus rapide que le faire sur les entités initiales.*

Nous avons proposé une exécution parallèle de notre approche et nous l'avons implémenté en utilisant Spark suivant le principe *Map/Reduce* [24, 25]. L'algorithme 1 décrit notre méthode d'extraction de patterns. Il prend comme entrée les triplets d'une source de données RDF et produit en sortie les patterns représentant les entités de la source. Les triplets RDF sont tout d'abord distribués sur les nœuds de calcul afin de permettre une exécution parallèle de l'algorithme. L'algorithme extrait à partir de chaque triplet le sujet et la propriété (ligne 3-5). Ensuite, il consolide les propriétés extraites sur chaque nœud par entités afin de former la structure de chaque entité (ligne 6). Enfin, l'algorithme extrait les patterns en regroupant les entités ayant la même structure (lignes 9-11) et calcule le nombre d'entités correspondant (ligne 13).

Algorithm 1 Extraction de patterns

Input: file data
1: $pattern : (Set(String) : propertySet, int : number)$
2: //read the files
3: **for all** t :triplet in data **do in parallel**
4: entities ← entities + Array(getSubject(t), getPredicat(t))
5: **end for**
6: //Group all properties that belong to the same entity
7: $entityList \leftarrow entities.groupByEntityId()$
8: //Extract the properties set from the list of entities
9: **for all** e :entity in entityList **do in parallel**
10: $pattern \leftarrow pattern + (getPropertySet(e), 1)$
11: **end for**
12: //Count the number of entity for each pattern
13: $pattern \leftarrow pattern.countEntityNumber()$
14: **return** pattern

L'extraction de patterns permet de réduire la taille d'une source de données RDF. Cependant, l'hétérogénéité de ces données peut avoir pour conséquence un un grand nombre de patterns. Les évaluations sur l'efficacité de l'approche d'extraction de patterns seront présentées dans le chapitre 6, où nous pourrons voir le taux de réduction de la taille d'un jeu de données RDF après l'extraction des patterns pour différentes sources.

Notre proposition est une première contribution au problème de la scalabilité des approches de découverte de schéma. Cependant, les données du web sont très hétérogènes, les entités appartenant à la même classe peuvent être décrites par des ensembles de propriétés différents. Par conséquent, ces entités ne seront pas représentées par le même pattern, mais plutôt par plusieurs. Cette hétérogénéité des données RDF implique que le nombre de

patterns dans une source de données peut être élevé. Le taux de compression d'un jeu de données peut donc être insuffisant, et les algorithmes de découverte de schéma existants peuvent ne pas être adaptés. Ainsi, l'extraction du schéma d'une source de données représentée par un grand nombre de patterns nécessite d'avoir un algorithme de clustering scalable. Pour cela, nous avons conçu un algorithme de clustering basé sur la densité distribué et implémenté en utilisant une technologie Big Data qui traite efficacement des grandes sources de données RDF. Cette proposition est décrite dans les sections suivantes.

3.5 Approche générale pour la découverte de schéma

Notre approche de découverte de schéma a pour objectif d'extraire un schéma qui décrit la structure des entités contenues dans une source de données RDF, dont la taille est trop importante pour pouvoir être gérée par les approches d'extraction de schéma existantes en raison de la complexité du traitement. L'approche extrait les classes implicites à partir des entités de la source de données ainsi que les propriétés décrivant chaque classe. Nous introduisons dans cette section le principe général de notre approche scalable de découverte de schéma.

Nous avons conçu un algorithme de clustering basé sur la densité, distribué et implémenté en utilisant une technologie Big Data qui traite efficacement des grandes sources de données RDF. Une exécution parallèle d'un algorithme de clustering basé sur la densité n'est pas triviale et soulève les problèmes suivants :

- Comment distribuer les données sur plusieurs nœuds de calcul lorsque la taille du jeu de données rend impossible son clustering sur un seul nœud ?
- Comment former les clusters à partir des données distribuées ? et comment limiter l'échange d'informations entre les nœuds de calcul durant ce processus, sachant que le voisinage des entités est distribué ?
- Comment garantir que l'algorithme de clustering parallèle produit le même résultat que l'algorithme séquentiel ?

Afin d'adresser les problèmes soulevés par une exécution distribuée d'un algorithme de clustering, nous proposons de diviser le jeu de données initial en sous-ensembles de données qui permettront la parallélisation du clustering des entités. Le clustering est ensuite exécuté dans chaque sous-ensemble, et des clusters locaux sont créés. Ces derniers sont fusionnés pour produire le résultat de clustering final. En plus de sa conception distribuée, notre algorithme produit le même résultat de clustering que l'algorithme DBSCAN séquentiel [28]. Les clusters formés par notre algorithme représentent les classes du schéma décrivant la source de données considérée.

La figure 3.4 donne un aperçu global de notre approche en insistant sur la parallélisation du processus et les communications entre les nœuds de calcul.

Dans la phase de distribution des données, des sous-ensembles d'entités sont créés en considérant les propriétés décrivant ces entités. Notre méthode de distribution assure le regroupement des entités similaires dans au moins un sous-ensemble de données, ainsi, toutes les paires d'entités similaires seront comparées au moins une

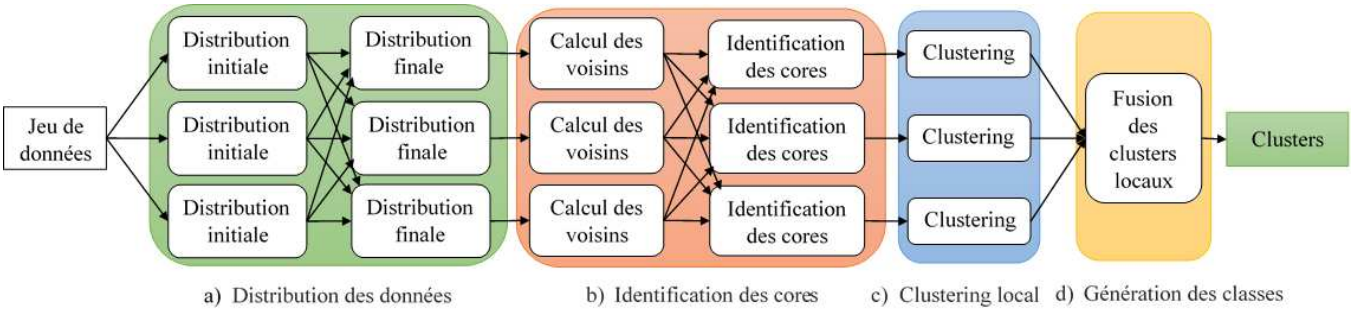


FIGURE 3.4 – Vue globale de notre approche de découverte de schéma

fois. De cette façon, toutes les comparaisons pertinentes seront effectuées durant le clustering des entités dans chaque sous-ensemble.

Après la distribution des données, le voisinage des entités est calculé dans chaque sous-ensemble. Pour chaque entité, les listes de ses voisins, qui peuvent être distribués sur plusieurs sous-ensembles, sont consolidées en une seule liste en échangeant les informations sur le voisinage des entités entre les nœuds de calcul. Durant cette étape, les entités ayant un voisinage dense, appelées entités cores, sont identifiées.

En s'appuyant sur les entités cores qui ont un voisinage dense, les clusters locaux sont formés dans chaque sous-ensemble en suivant le principe de densité. Pour créer les clusters locaux, l'approche part d'une entité arbitraire ayant un voisinage dense et retrouve toutes les entités qui lui sont similaires. Ensuite, ses voisins qui ont un voisinage dense sont retrouvés et récursivement ajoutés au cluster local.

Les clusters ayant des éléments distribués sur plusieurs sous-ensembles et affectés à des clusters locaux distincts sont formés. Les clusters locaux sont fusionnés afin de produire les clusters finaux. Deux clusters sont fusionnés s'ils partagent une entité core commune.

Notre proposition est implémentée en utilisant Spark, un framework open source de calcul distribué avec un moteur de traitement de données (principalement) en mémoire, adapté à la gestion de grands jeux de données [104]. Comme c'est toujours le cas pour les framework de calculs distribués, les opérations qui nécessitent l'échange d'informations entre les nœuds de calcul sont coûteuses. Certaines opérations dans Spark déclenchent un événement connu sous le nom de *shuffle*, qui sert à redistribuer les données sur les nœuds de calcul. Il implique la copie physique des données à travers les exécuteurs et les machines, ce qui en fait une opération lourde et coûteuse. Nous avons proposé une nouvelle méthode de distribution de données qui réduit à la fois les communications entre les nœuds de calcul et le recours à des opérations de type *shuffle* dans Spark.

Nous détaillons dans les sections suivantes les concepts et les algorithmes de notre approche scalable de découverte de schéma.

3.6 Distribution des données

La distribution des données joue un rôle très important dans notre processus de parallélisation de la découverte de schéma. Le jeu de données initial est d'abord divisé en sous-ensembles de données où le regroupement en clusters des entités peut se faire en parallèle sur les nœuds de calcul. Nous introduisons un principe original de distribution qui minimise les transferts de données entre les nœuds de calcul, et assure que le clustering d'un sous-ensemble ne nécessite aucune information qui se trouverait dans d'autres sous-ensembles. En conséquence, nous limitons les transferts de données entre les nœuds de calcul. La méthode de distribution doit fournir les informations nécessaires à l'identification et la restructuration des clusters qui s'étendent sur plusieurs sous-ensembles ; dans notre proposition, les entités dupliquées seront utilisées pour effectuer la fusion.

Les algorithmes de clustering discutés dans l'état de l'art proposent différentes méthodes de distribution des données pour paralléliser le calcul des clusters. Tout d'abord, certaines approches distribuent les données aléatoirement sur les différents nœuds de calcul [73, 84, 95]. Cette distribution permet la création de sous-ensembles de données de tailles égales ce qui offre un calcul de clustering rapide, cependant, elle ne garantit pas la comparaison de toutes les entités similaires. C'est également le cas des approches probabilistes [72]. Par conséquent, le résultat du clustering n'est pas le même que celui produit par un clustering séquentiel car certaines connections par densité qui auraient dû être formées entre les entités non comparées seront absentes, ce qui réduirait la qualité des clusters. D'autres approches proposent d'ordonner les entités et de les distribuer par rapport à cet ordre [48]. Suivant cette distribution, les entités qui peuvent être similaires et qui appartiennent à différents sous-ensembles peuvent être identifiées en comparant les entités dans les bordures des sous-ensembles. Mais cette distribution ne s'applique pas sur les données RDF, qui sont de type catégoriel, de grande dimension et qui ne respectent aucun ordre particulier. Enfin, il existe des approches qui utilisent le partitionnement de l'espace de données en régions, où chaque région comporte des entités proches dans un espace à n-dimension [50, 51, 100]. Ces méthodes assurent la production d'un résultat similaire à un clustering séquentiel, mais leur application sur les données RDF est impossible car ces données ne peuvent pas être représentées dans un espace à n-dimensions. Dans notre approche, nous proposons de distribuer les entités par rapport aux propriétés les décrivant. Ainsi, les sous-ensembles de données sont créés en se basant sur les propriétés du jeu de données, ensuite, les entités sont assignées à ces sous-ensembles de façon à ce que les entités qui présentent des propriétés communes et qui sont susceptibles d'être similaires soient groupées dans des sous-ensembles. Notre distribution assure que les entités similaires sont groupées dans au moins un sous-ensemble. De cette façon, toutes les paires d'entités similaires seront identifiées.

Dans cette section, nous introduisons d'abord notre principe de distribution des données qui divise le jeu de données initial en sous-ensembles tout en répondant aux exigences définies ci-dessus. Ensuite, nous présentons une optimisation de cette distribution qui permet de réduire la taille des sous-ensembles. Comme la distribution des données initiale peut créer des sous-ensembles comportant un nombre d'entités qui dépassent la capacité de

traitement d'un nœud de calcul, nous expliquons ensuite comment décomposer ces grands sous-ensembles.

3.6.1 Principe de distribution des entités

L'intuition derrière notre proposition est d'assigner toutes les entités similaires, qui partagent des propriétés communes, aux mêmes sous-ensembles. En effet, selon l'indice de *Jaccard*, deux entités sont similaires si elles partagent un nombre de propriétés supérieur à un seuil donné. Dans notre proposition, les entités qui peuvent être similaires sont assignées à au moins un sous-ensemble commun, et seront comparées durant le calcul de leurs voisinages. Les comparaisons des entités dans chaque sous-ensemble se feront dans une autre phase. Afin d'assurer un regroupement efficace des entités dans des clusters, les entités sont distribuées à travers plusieurs sous-ensembles. Deux entités qui ne sont jamais assignées à un même sous-ensemble sont des entités qui ne sont pas similaires, car le nombre de propriétés qu'elles partagent n'est pas suffisant.

Exemple 3.6.1 *Par exemple, deux entités e_1 et e_2 décrites par $\overline{e_1} = \{p_1, p_2\}$ et $\overline{e_2} = \{p_2, p_3\}$ partageant une propriété p_2 seront regroupées et comparées dans le sous-ensemble qui représente la propriété p_2.*

Dans notre travail, nous désignons un tel sous-ensemble par $chunk$ de données. Un $chunk$ est défini comme suit :

Définition 3.6.1 *Un $chunk$ créé pour un ensemble de propriétés $P \subseteq \mathcal{P}$ désigné par $[P]$ est un sous-ensemble d'entités ayant les propriétés de P dans leur description : $e \in [P] \implies P \subseteq \overline{e}$*

Nous décrivons d'abord une méthode naïve de distribution de données en $chunks$ pour introduire le principe de distribution. Ensuite, nous détaillons l'optimisation adoptée dans notre méthode de distribution.

L'approche naïve de distribution des données consiste à assigner les entités à des $chunks$ en considérant toutes les propriétés qui les décrivent. Ainsi, une entité e décrite par les propriétés $\overline{e} = \{p_1, p_2, \ldots, p_n\}$ sera assignée aux $chunks$ $[p_1], [p_2], \ldots, [p_n]$. e est donc groupée dans le $chunk$ $[p_i]$ avec toutes les entités qui partagent au moins une propriété avec e.

Définition 3.6.2 *Soit une entité e et son ensemble de propriétés $\overline{e}$. En adoptant la distribution naïve, e est assignée au $chunk$ $[p]$ pour chaque propriété p dans $\overline{e}$:*

$$\forall e, \forall p \in \overline{e}, e \textit{ est assignée à } [p].$$

Exemple 3.6.2 *La figure 3.1 présente un jeu de données D. Dans ce qui suit, les propriétés $name$, id, $publish$, $gender$, $title$, $conference$, $year$, $rank$ seront remplacées respectivement par $p_1, p_2, p_3, \ldots, p_8$. Ainsi, les entités $\{e_i \mid i \in [1, 7]\}$ sont décrites comme suit :*
$\overline{e_1} = \{p_1, p_2, p_3\}$, $\overline{e_2} = \{p_1, p_2, p_3, p_4\}$, $\overline{e_3} = \{p_2, p_3, p_4\}$, $\overline{e_4} = \{p_2, p_5, p_6, p_7\}$, $\overline{e_5} = \{p_2, p_5, p_6\}$, $\overline{e_6} = \{p_1, p_2, p_5, p_8\}$, $\overline{e_7} = \{p_2, p_5, p_7\}$.

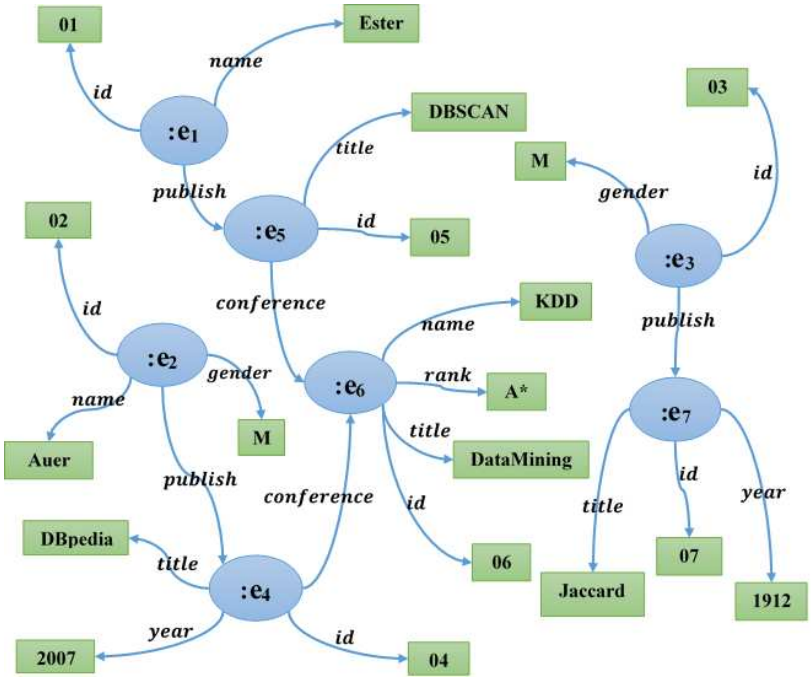


FIGURE 3.5 – Exemple de jeu de données décrivant des auteurs et leurs publications dans des conférences

La distribution des entités sur les $chunks$ en utilisant notre principe de distribution produit le résultat présenté dans la figure 3.6.

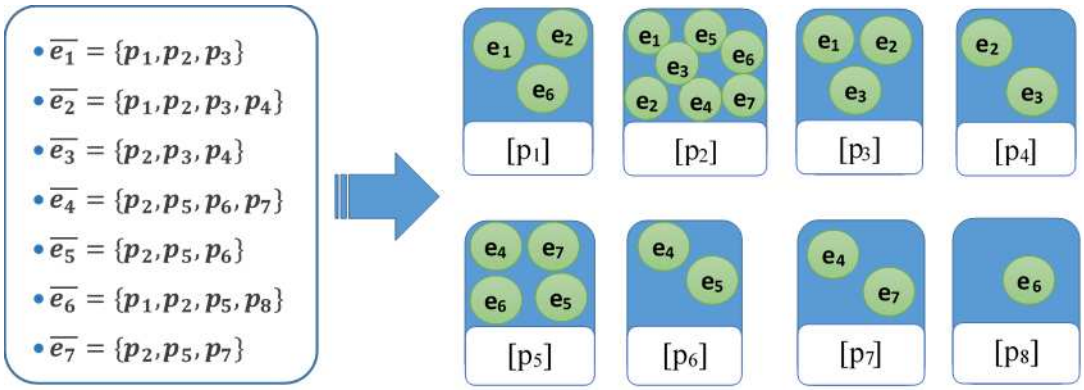


FIGURE 3.6 – La distribution de D dans des $chunks$

Par exemple, l'entité e_1 décrite par les propriétés p_1, p_2 et p_3 est assignée aux $chunks$ $[p_1]$, $[p_2]$ et $[p_3]$. Suivant cette distribution, e_1 est regroupée avec toutes les entités qui partagent au moins une propriété avec elle. L'entité e_2 est regroupée avec les entités qui lui sont similaires, e_2 et e_3 dans les $chunks$ $[p_1]$, $[p_2]$ et $[p_3]$.

Proposition 3.6.1 *(Distribution naïve (Exactitude)) En utilisant la distribution naïve, deux entités similaires seront groupées dans au moins un $chunk$, i.e. toutes les comparaisons nécessaires pour identifier des paires d'entités similaires seront effectuées.*

Preuve 3.6.1 *Selon l'indice de similarité, deux entités similaires doivent partager au moins une propriété. En utilisant la distribution naïve, ces entités seront assignées à au moins un $chunk$ commun, ainsi, elles seront comparées et identifiées comme voisines. Ceci garantit l'identification de tous les voisins pour chaque entité du jeu de données.*

Cependant, la distribution naïve souffre d'un inconvénient important. Deux entités similaires peuvent être groupées de façon redondante plusieurs fois. Dans l'exemple 3.6.2, les entités e_1 et e_2 décrites respectivement par $\overline{e_1} = \{p_1, p_2, p_3\}$ et $\overline{e_2} = \{p_1, p_2, p_3, p_4\}$ seront assignées aux $chunks$ $[p_1], [p_2], [p_3]$ et par conséquent, elles seront comparées trois fois alors qu'une seule comparaison est suffisante. La duplication peut être importante si les entités sont décrites par un grand nombre de propriétés. De plus, si une propriété décrit un nombre élevé d'entités, le $chunk$ créé à partir de cette propriété sera de grande taille, comme par exemple, le $chunk$ $[p_2]$ qui comporte toutes les entités du jeu de données. Le clustering des entités dans des $chunks$ de grande taille sera coûteux et pénalisera les performances. Afin de pallier ces problèmes, nous présentons dans ce qui suit une optimisation de notre méthode de distribution qui réduit le nombre d'entités dans chaque $chunk$ tout en garantissant l'identification de toutes les paires d'entités similaires.

3.6.2 Optimisation de la distribution

Notre méthode de distribution crée des $chunks$ sur la base des propriétés décrivant les entités. Ce principe induit la duplication des entités dans plusieurs $chunks$, car chacune est décrite par plusieurs propriétés. Pour diminuer la taille des $chunks$ produits par la distribution des données dans notre approche, nous ne considérons pas toutes les propriétés durant l'assignation des entités aux $chunks$ afin de limiter la duplication et de réduire le coût du processus de comparaison à l'intérieur des $chunks$.

Pour cela, nous définissons le nombre de propriétés minimum à considérer lors de la distribution d'une entité afin de garantir son regroupement avec chacune de ses voisines dans au moins un $chunk$. Nous définissons ce nombre comme étant le *seuil de dissimilarité*, qui définit le nombre de propriétés à considérer pour une entité e_i afin de déterminer les entités qui peuvent être similaires à e_i.

Par exemple, considérons l'entité e_2, telle que $\overline{e_2} = \{p_1, p_2, p_3, p_4\}$ et $\epsilon = 0.7$. Si e_2 possède dans sa description plus de deux propriétés différentes d'une autre entité quelconque, cette dernière ne peut être similaire à e_2. Par exemple, une entité e' décrite par $\overline{e'} = \{p_3, p_4, p_5\}$ n'est pas similaire à e_2 car $\overline{e_2} \setminus \overline{e'} = \{p_1, p_2\}$ comporte deux éléments. Nous montrons par la suite qu'il suffirait d'assigner e_2 aux $chunks$ $[p_1]$ et $[p_2]$ pour s'assurer que toutes les entités qui lui sont similaires se trouvent dans ces $chunks$. Les entités qui ne sont pas assignées à ces $chunks$ ne sont par conséquent pas similaires à e_2.

Cependant, les propriétés ne doivent pas être choisies aléatoirement, car, cela n'assurerait pas l'assignation d'entités similaires au même $chunk$ et des entités similaires pourraient ne pas être considérées comme voisines. Illustrons d'abord ce problème par l'exemple suivant.

Exemple 3.6.3 *Considérons les entités similaires e_2 et e_3 telles que $\overline{e_2} = \{p_1, p_2, p_3, p_4\}$ et $\overline{e_3} = \{p_2, p_3, p_4\}$. Supposons qu'il suffise de considérer deux propriétés lors de la distribution de chaque entité au lieu de considérer toutes les propriétés ; l'entité e_2 peut être affectée aux $chunks$ $[p_1]$, $[p_2]$ et e_3 à $[p_3]$ et $[p_4]$. Par conséquent, e_2 et e_3 ne sont pas regroupées dans un même $chunk$, et ne sont pas comparées alors qu'elles sont similaires.*

Nous pouvons ainsi noter que l'assignation aléatoire des entités ne garantit pas que les entités similaires seront comparées. Afin de résoudre ce problème, nous définissons une relation d'ordre total sur les propriétés et la sélection des propriétés à considérer lors de l'assignation d'une entité se fera par rapport à cet ordre.

Exemple 3.6.4 *En assignant les entités en respectant l'ordre des indices dans notre exemple, l'entité e_3 sera assignée à $[p_2]$ au lieu de $[p_3]$. Ainsi, e_2 et e_3 seront regroupées dans le $chunk$ $[p_2]$ et comparées durant le calcul de leurs voisinages.*

Ainsi, la distribution des données est faite en deux phases. Premièrement, nous calculons pour chaque entité son *seuil de dissimilarité*, qui permet de définir le nombre de $chunks$ auxquels une entité doit être assignée, en définissant le nombre de propriétés à considérer. Deuxièmement, nous définissons une relation d'ordre sur les propriétés ; les $chunks$ auxquels les entités sont assignées seront choisis par rapport à cet ordre.

Nous formalisons à présent l'intuition dernière l'optimisation de la distribution, et nous introduisons une proposition qui exprime que deux entités ne peuvent pas être similaires si la différence entre leurs ensembles de propriétés dépasse un certain seuil.

Proposition 3.6.2 *Soient e_1 et e_2 deux entités comparées en utilisant l'indice de* Jaccard. *Si $|\overline{e_1} \setminus \overline{e_2}| \geq |\overline{e_1}| - \lceil \epsilon \times |\overline{e_1}| \rceil + 1$ alors e_1 et e_2 ne peuvent pas être similaires.*

Preuve 3.6.2 *Supposons que $|\overline{e_1} \setminus \overline{e_2}| \geq |\overline{e_1}| - \lceil \epsilon \times |\overline{e_1}| \rceil + 1$. Nous avons $|\overline{e_1} \setminus \overline{e_2}| = |\overline{e_1}| - |\overline{e_1} \cap \overline{e_2}|$. Ainsi, $|\overline{e_1}| - |\overline{e_1} \cap \overline{e_2}| \geq |\overline{e_1}| - \lceil \epsilon \times |\overline{e_1}| \rceil + 1$. En soustrayant $|\overline{e_1}|$ des deux côtés, nous obtenons $|\overline{e_1} \cap \overline{e_2}| \leq \lceil \epsilon \times |\overline{e_1}| \rceil - 1$ qui implique que $|\overline{e_1} \cap \overline{e_2}| < \lceil \epsilon \times |\overline{e_1}| \rceil$. Selon la définition de l'indice de Jaccard, cette formule implique que e_1 et e_2 ne peuvent pas être similaires.*

Nous définissons maintenant la notion de *seuil de dissimilarité* pour une entité e. Dans notre travail, ce *seuil de dissimilarité* est basé sur l'indice de Jaccard. L'utilisation d'un autre indice de similarité nécessite une autre définition de ce seuil adapté à ce nouvel indice.

Définition 3.6.3 *Le* seuil de dissimilarité *pour une entité e est le nombre $dt(e) = |\overline{e}| - \lceil \epsilon \times |\overline{e}| \rceil + 1$.*

La définition suivante présente la distribution optimisée des données.

Définition 3.6.4 *s Soit $<_{\mathcal{P}}$ une relation d'ordre sur les propriétés décrivant un jeu de données, et soit e une entité avec $\overline{e} = \{p_1, p_2, \ldots, p_n\}$ et $p_i <_{\mathcal{P}} p_{i+1}$ pour $1 \leq i < n$. Avec la distribution des données optimisée, une entité e est*

assignée aux $chunks$ $[p_1]$, $[p_2]$, ..., $[p_{dt(e)}]$. *L'ensemble de comparaison de* e *noté* $cs(e)$ *est l'ensemble de propriétés* $\{p_1, p_2, \ldots, p_{dt(e)}\}$.

Proposition 3.6.3 *En utilisant la distribution optimisée des données, toutes les comparaisons nécessaires pour le clustering des entités seront effectuées au moins une fois dans l'un des* $chunks$ *existants.*

Preuve 3.6.3 *Nous devons montrer que si deux entités sont similaires, alors elles sont toutes les deux assignées à au moins un* $chunk$ *commun. Soient* e_1 *et* e_2 *deux entités similaires. Nous avons* $\frac{|\overline{e_1} \cap \overline{e_2}|}{|\overline{e_1} \cup \overline{e_2}|} \geq \epsilon$. *Ainsi,* $|\overline{e_1} \cap \overline{e_2}| \geq \epsilon \times |\overline{e_1} \cup \overline{e_2}|$ *qui implique que* $|\overline{e_1} \cap \overline{e_2}| \geq \lceil \epsilon \times |\overline{e_1}| \rceil$ *et* $|\overline{e_1} \cap \overline{e_2}| \geq \lceil \epsilon \times |\overline{e_2}| \rceil$. *Ceci implique que* $|\overline{e_1}| - |\overline{e_1} \cap \overline{e_2}| \leq |\overline{e_1}| - \lceil \epsilon \times |\overline{e_1}| \rceil$. *Comme* $|\overline{e_1} \setminus \overline{e_2}| = |\overline{e_1}| - |\overline{e_1} \cap \overline{e_2}|$, *nous obtenons* $|\overline{e_1} \setminus \overline{e_2}| \leq |\overline{e_1}| - \lceil \epsilon \times |\overline{e_1}| \rceil$. *Comme* $|cs(e_1)| = dt(e_1) > |\overline{e_1}| - \lceil \epsilon \times |\overline{e_1}| \rceil$, *nous avons donc* $cs(e_1) \cap \overline{e_2} \neq \emptyset$.

Nous pouvons montrer de la même manière que $cs(e_2) \cap \overline{e_1} \neq \emptyset$. *En conséquence,* $cs(e_1)$ *et* $cs(e_2)$ *contiennent tous les deux un élément de* $\overline{e_1} \cap \overline{e_2}$.

Si une relation d'ordre est définie sur les propriétés, nous pouvons choisir le minimum de $\overline{e_1} \cap \overline{e_2}$ *pour* $cs(e_1)$ *et* $cs(e_2)$. *Dans ce cas,* $cs(e_1) \cap cs(e_2) \neq \emptyset$. *Ce qui signifie qu'il existe au moins un* $chunk$ *contenant à la fois* e_1 *et* e_2.

Dans notre travail, nous proposons d'ordonner les propriétés par rapport à leur sélectivité qui représente le nombre d'entités décrites par chaque propriété. Ceci afin de créer des $chunks$ comportant le moins d'éléments possibles tout en assurant le regroupement des entités similaires dans au moins un $chunk$.

Définition 3.6.5 *La sélectivité d'une propriété est le ratio du nombre d'entités décrites par cette propriété, divisé sur le nombre total des entités. La sélectivité de la propriété* p *est calculée comme suit :*
$selec(p) = 1 - \frac{|p|}{|D|}$, D *étant le jeu de données.*

Dans notre approche, les propriétés sont ordonnées de la plus sélective à la moins sélective. Ainsi, un plus grand nombre de comparaisons inutiles (c'est-à-dire la comparaisons d'entités non similaires) seront ignorées et le clustering effectué dans chaque $chunk$ sera plus rapide. De plus, les $chunks$ créés comportent un nombre plus réduit d'entités car ils sont produits en considérant les propriétés les plus sélectives.

Exemple 3.6.5 *Considérons le jeu de données* D *présenté dans l'exemple 3.6.2 et comportant les entités* $\{e_i \mid i \in [1, 7]\}$. *Dans cet exemple, le seuil de similarité est fixé à* $\epsilon = 0.7$. *En considérant la sélectivité des propriétés, l'ordre défini sur ces dernières est le suivant :* $p_8 <_{\mathcal{P}} p_4 <_{\mathcal{P}} p_6 <_{\mathcal{P}} p_7 <_{\mathcal{P}} p_1 <_{\mathcal{P}} p_3 <_{\mathcal{P}} p_5 <_{\mathcal{P}} p_2$. *La distribution optimisée des entités sur les* $chunks$ *produit le résultat présenté dans la figure 3.7.*

Par exemple, le seuil de dissimilarité de l'entité e_2 *est égal à* $dt(e_2) = 4 - \lceil 0.7 \times 4 \rceil + 1 = 2$. *Les deux propriétés les plus sélectives décrivant* e_2 *sont* p_1 *et* p_4, e_2 *est ainsi assignée aux* $chunks$ $[p_1]$ *et* $[p_4]$. *Cette distribution assure que* e_2 *est regroupée avec toutes les entités similaires dans au moins un* $chunk$, *et sera donc comparée à chacune d'elles au moins une fois. L'entité* e_2 *est regroupée avec toutes ses voisines,* e_1 *et* e_3 *respectivement dans* $[p_1]$ *et* $[p_4]$.

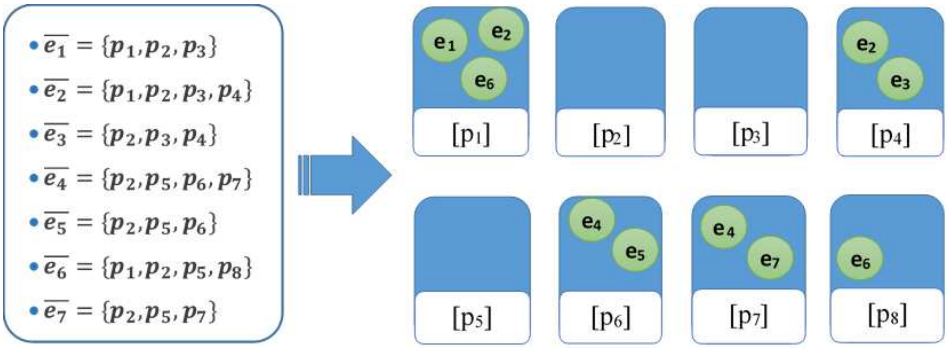


FIGURE 3.7 – La distribution optimisée de D dans des partitions

Les $chunks$ comportant moins de deux entités comme $[p_2]$ et $[p_8]$ sont supprimés, car un $chunk$ doit comporter au minimum deux entités pour pouvoir effectuer une comparaison.

L'algorithme 2 décrit la phase de distribution des données. Il prend en entrée le seuil de similarité ϵ, utilisé pour le calcul de seuil de dissimilarité, et pour définir les $chunks$ $cs(e)$ de chaque entité e.

Algorithm 2 Distribution des entités

Input: jeu de données D, seuil de similarité ϵ
1: **for all** entity e in D **do in parallel**
2: **for all** property p in $cs(e)$ **do**
3: $[p] = [p] \cup \{e\}$
4: **end for**
5: **end for**
6: Fusionner les chunks générés par l'exécution parallèle
7: **return** the chunks

La définition des $chunks$ auxquels une entité est affectée est effectuée en parallèle sur les différents nœuds de calcul afin d'assurer un traitement rapide (lignes 1-5). Chaque nœud de calcul distribue à travers des $chunks$ les entités qu'il traite. Les $chunks$ produits par cette exécution parallèle sont ensuite fusionnés afin d'obtenir les $chunks$ finaux (ligne 6).

Le processus de distribution peut créer des $chunks$ qui sont trop grands pour être traités sur un seul nœud de calcul. Cela nécessite un partitionnement supplémentaire, que nous décrivons dans la section suivante.

3.6.3 Gestion des grands chunks

Étant donnée un $chunk$ $[p]$ contenant un ensemble d'entités décrites par la propriété p, le nombre d'entités dans $[p]$ peut excéder la capacité de calcul d'un seul nœud, ce qui rend impossible l'exécution du clustering des entités contenues dans ce $chunk$ sur un nœud. Dans ce cas de figure, chaque $chunk$ $[p]$ comportant un grand nombre d'entités est divisé en considérant les autres propriétés.

Nous désignons par $capacity$ le paramètre qui détermine la capacité d'un seul nœud de calcul en nombre d'entités, plus précisément, le nombre de comparaisons qu'un nœuds de calcul peut faire.

Dans le cas d'un grand $chunk$ $[p]$ qui contient un nombre d'entités supérieur au paramètre $capacity$, nous proposons de créer des $sous$-$chunks$ pour chaque propriété décrivant les entités dans $[p]$ excepté la propriété p, ensuite chaque entité qui appartient à $[p]$ est assignée aux $sous$-$chunks$ si elle est décrite par la propriété utilisée pour créer le $sous$-$chunk$:

$$\forall e \in [p], \forall p_i \in \overline{e}, [\{p, p_i\}] = [\{p, p_i\}] \cup \{e\}$$

Récursivement, la taille de touts les $chunks$ résultants est évaluée et les $chunks$ excédant la capacité d'un nœud de calcul sont divisés jusqu'à l'obtention de $chunks$ comportant un nombre d'entités inférieur à la capacité d'un seul nœud de calcul.

La distribution récursive des données garantit que toutes les entités similaires sont regroupées dans les $sous$-$chunks$ tant que la taille d'un cluster ne dépasse pas la capacité d'un seul nœud de calcul. En effet, nous avons prouvé que notre méthode de distribution optimisée assure que toutes les comparaisons nécessaires pour le clustering des entités seront effectuées au moins une fois dans un $chunk$ (cf. preuve 3.6.3). Cette preuve est valable pour les éléments d'un $chunk$ $[p]$ qui sont distribués en $sous$-$chunks$. La preuve 3.6.3 garantit que toutes les entités similaires dans le jeu de données D sont identifiées après leur distribution en $chunk$. Si nous remplaçons D par $[p]$ dans cette preuve, nous concluons que toutes les entités similaires incluses dans $[p]$ sont identifiées après leur distribution en $sous$-$chunk$ suivant notre principe. Par conséquent, la distribution récursive de grands $chunks$ garantit que toutes les entités similaires appartenant à un $chunk$ $[p]$ seront groupées et comparées dans au moins un $sous$-$chunk$.

La distribution des données crée des $chunks$ du jeu de données initial, chacun des $chunks$ comportant un nombre d'entités qui peuvent être efficacement regroupées en clusters par un seul nœud de calcul. La distribution des entités à travers les $chunks$ ne nécessite aucun partage d'information entre les nœuds de calcul.

Exemple 3.6.6 *Par exemple, si la capacité d'un nœud est fixée à 3 et que nous considérons le* $chunk$ $[p_2] = \{e_1, e_2, e_3, e_4, e_5\}$ *dans notre exemple précédent, la taille de ce* $chunk$ *serait supérieure au paramètre* $capacity$*. Dans ce cas de figure,* $[p_2]$ *sera divisé en* $sous$-$chunks$*, par exemple les* $sous$-$chunks$ $[p_2, p_1] = \{e_2\}$ *et* $[p_2, p_3] = \{e_1, e_2\}$ *créés respectivement pour les propriétés* p_1 *et* p_3*.*

L'algorithme 3 évalue la taille de chaque $chunk$ en utilisant le paramètre $capacity$ et divise ceux qui dépassent la capacité de calcul d'un nœud. Ce processus est appliqué récursivement jusqu'à ce que la taille de touts les $chunks$ créés soit inférieure au paramètre $capacity$.

Après la génération des $chunks$ telles que la taille de chacun ne dépasse pas la capacité de calcul des nœuds, le calcul de voisinage de chaque entité sera effectué dans chacun des $chunks$. Ce processus est décrit dans la section suivante.

Algorithm 3 Divisions des grands $chunks$

Input: chs : les chunks, cap : la capacité de calcul d'un nœuds
1: **for all** $[P] \in chs \mid |[P]| > cap$ **do in parallel**
2: **for all** $e \in [P]$ **do**
3: **for all** $p_i \in \overline{e} \setminus P$ **do**
4: $[P \cup \{p_i\}] = [P \cup \{p_i\}] \cup \{e\}$
5: **end for**
6: **end for**
7: **end for**
8: Fusionner les chunks générés par l'exécution parallèle
9: **return** les chunks

3.7 Identification des cores

Dans un algorithme de clustering, les entités similaires sont groupées dans des clusters. Notre proposition se base sur la densité pour le regroupement des entités et la notion de proximité est liée à la densité du voisinage d'une entité. Afin de former un cluster à partir d'une entité donnée, le voisinage de cette entité doit contenir un nombre suffisant d'entités ; en d'autres termes, la densité de son voisinage doit dépasser un seuil fixé. Cette section décrit le processus de calcul de voisinage et l'identification des entités qui ont un voisinage dense, appelées entités cores.

Nous rappelons tout d'abord quelques définitions utilisées dans l'algorithme DBSCAN [28] dont s'inspire notre approche.

Définition 3.7.1 *L'*ϵ-voisinage *d'une entité* e *est l'ensemble d'entités dont la similarité avec* e *évaluée en utilisant* Jaccard *est supérieur ou égal à* ϵ*.*

$$neighborhood_\epsilon(e) = \{e_i \in D \mid J(e, e_i) \geq \epsilon\}$$

En considérant l'ϵ-voisinage de chaque entité, on distingue trois types d'entités : les entités cores qui comportent au moins $minPts$ entités dans leurs ϵ-voisinage, les entités bordures, qui ne sont pas des entités cores mais appartiennent à l'ϵ-voisinage d'une entité core, et les entités bruits qui ne font partie du ϵ-voisinage d'aucune entité core. Les entités bruits ne sont assignées à aucun cluster.

Définition 3.7.2 *Une entité* e *est dite entité core si le nombre d'entités dans son* ϵ*-voisinage est supérieur au paramètre de densité* $minPts$*, i.e.* $|neighborhood_\epsilon(e)| \geq minPts$*.*

Après le calcul de l'ϵ-voisinage de chaque entité, les entités cores sont identifiées. Cependant, comme les données sont distribuées, le voisinage d'une entité peut être distribué dans plusieurs $chunks$. Ainsi, le nombre de voisins d'une entité ne se calcule pas seulement dans un $chunk$, et nous devons donc considérer les voisins dans la totalité du jeu de données.

Exemple 3.7.1 *Si on fixe* $minPts$ *à* 2 *dans notre exemple, l'entité* e_2 *qui comporte* e_1 *et* e_3 *dans son* ϵ*-voisinage sera une entité core. Mais après l'assignation des entités aux* $chunks$*, le voisinage de* e_2 *est distribué dans les*

$chunks$ $[p_1]$ et $[p_4]$. Si les comparaisons entre les entités sont effectuées indépendamment dans les $chunks$, le nombre de voisins de l'entité e_2 dans chaque $chunk$ n'atteindra pas $minPts$ et e_2 ne sera pas considérée comme étant une entité core.

Dans notre approche, nous proposons d'identifier les entités cores en deux étapes. Tout d'abord, l'ϵ-voisinage de chaque entité est calculé en parallèle dans chaque $chunk$. Le calcul de l'ϵ-voisinage de chaque entité représente l'opération la plus coûteuse dans un algorithme de clustering basé sur la densité car cela nécessite la comparaison de toutes les paires d'entités. Notre algorithme opère sur des $chunks$ de données comportant un nombre d'entités suffisamment réduit pour permettre une exécution rapide. De plus, la distribution des données se fait en considérant un nombre minimal de propriétés pour chaque entité, grâce à cela, un certain nombre de comparaisons inutiles sont évitées : deux entités dont nous savons à priori qu'elles ne sont pas similaires, parce qu'elles ne sont dans aucun $chunk$ commun, ne sont pas comparées. Afin d'obtenir de meilleures performances, ce processus est parallélisé à travers les nœuds de calcul sans besoin d'échange d'informations entre ces nœuds.

Ensuite, les voisins découverts dans chaque $chunk$ sont consolidés par entité, et la liste des voisins correspondant à chaque entité dans le jeu de données complet est formée. Ainsi, bien que les données soient distribuées, nous calculons l'ϵ-voisinage pour chaque entité dans la totalité du jeu de données. Les entités cores ayant un nombre de voisins supérieur ou égale à $minPts$ sont enfin identifiées.

Exemple 3.7.2 *En fixant $minPts = 2$, les entités cores identifiées dans notre exemple 3.6.5 sont e_2 et e_4. L'algorithme identifie les voisins de e_2, qui sont e_1 et e_3 respectivement dans les $chunks$ $[p_1]$ et $[p_4]$. Ensuite, ces listes sont fusionnées afin de fournir la liste complète des voisins de e_2 : $neighborhood_\epsilon(e_2) = \{e_1, e_3\}$. Finalement, e_2 est identifiée comme une entité core car le nombre d'entités dans son voisinage est égal à $minPts$.*

L'algorithme 4 décrit la phase d'identification des entités cores, qui s'exécute en parallèle dans chaque $chunk$.

Algorithm 4 Identification des cores

Input: chs : les chunks, ϵ : seuil de similarité, $minPts$: seuil de densité

```
1:  for all [P] ∈ chs do in parallel
2:      for all e ∈ [P] do
3:          neighborhood_ε(e) = {e_i ∈ [P] | J(e, e_i) ≥ ε}
4:      end for
5:  end for
6:  Fusionner les voisins locaux afin de former la liste des voisins total pour chaque entité
7:  for all e ∈ D do
8:      if |neighborhood_ε(e)| ≥ minPts then
9:          cores = cores ∪ {e}
10:     end if
11: end for
12: return cores
```

Cet algorithme construit la liste de voisins pour chaque entité dans chaque $chunk$ (lignes 1-5), et consolide ensuite ces listes par entité (ligne 6). Les listes de voisins pour chaque entité sont échangées entre les nœuds de calcul afin de grouper chaque entité avec tous ses voisins. L'algorithme marque ensuite les entités ayant un nombre de voisin supérieur ou égale à $minPts$ comme entités cores (ligne 7-11).

Ayant calculé le voisinage de chaque entité et identifié les entités cores, le regroupement des entités en cluster est effectué localement dans chaque $chunk$. Ce processus est décrit dans la section qui suit.

3.8 Clustering Local

Le clustering local des entités est exécuté en parallèle et indépendamment dans les différents $chunks$; la stratégie de distribution assure que le clustering dans un $chunk$ ne nécessite aucune information qui se situe dans d'autres $chunks$. Ceci réduit le coût des communications entre les $chunks$ pour échanger des informations, et accélère le calcul des clusters. Un cluster local comporte des entités similaires qui se trouvent dans un même $chunk$.

Dans un algorithme de clustering par densité, les clusters sont formés suivant le principe d'atteignabilité par densité, introduit dans l'algorithme DBSCAN [28]. Nous rappelons ces définitions dans ce qui suit.

Définition 3.8.1 *Une entité e est directement atteignable par densité à partir d'une entité e' par rapport à ϵ et $minPts$ si et seulement si e' est une entité core et e est dans son ϵ-voisinage, i.e.*
$|neighborhood_\epsilon(e')| \geq minPts$ *et* $e \in neighborhood_\epsilon(e')$.

Définition 3.8.2 *: la structure d'une entité représente l'ensemble de propriétés qui la décrivent Une entité e est atteignable par densité à partir de e' par rapport à ϵ et $minPts$ s'il existe une chaîne d'entités $e_1, \ldots, e_n$, $e_1 = e'$, $e_n = e$ telles que e_{i+1} est directement atteignable par densité à partir de $e_i, \forall i \in \{1, \ldots, n\}$.*

Les clusters sont construits à partir des entités cores. Comme le voisinage des entités a été calculé et les entités cores identifiées, nous disposons de toutes les informations nécessaires à la génération des clusters localement dans chaque $chunk$. Seules les entités cores généreront un cluster en ajoutant leurs voisins comme des éléments de ce cluster. Les autres entités seront soit des entités bordures dans le voisinage d'une entité core, ou des entités bruits qui n'appartiennent à aucun cluster.

Pour chaque entité core e, un cluster C comportant e et ses voisins est créé. Les entités cores dans l'ϵ-voisinage de e sont ensuite identifiées et leurs voisins sont ajoutés au cluster C. Les voisins des entités cores dans C sont récursivement ajoutés au cluster jusqu'à ce que l'expansion du cluster s'arrête sur les entités bordures.

La figure 3.4.c montre la parallélisation de cette opération ; le clustering est effectué sur chaque $chunk$ indépendamment des autres et produit un clustering local.

Exemple 3.8.1 *Le clustering des $chunks$ obtenus à partir de l'exemple 3.6.5 en se basant sur les entités cores identifiées dans l'exemple 3.7.2 génère le résultat présenté dans la figure 3.8. Pour éviter de confondre les clusters créés dans différents $chunks$, les clusters sont annotés par l'identifiant du $chunk$ suivi d'un indice. Dans notre exemple, quatre clusters sont formés, $c_{p_1.1}$, $c_{p_4.1}$, $c_{p_6.1}$ et $c_{p_7.1}$ respectivement dans les $chunks$ p_1, p_4, p_6 et p_7.*

Par exemple, l'entité e_2 dans les $chunks$ $[p_1]$ et $[p_4]$ forme un cluster dans chaque $chunk$ en regroupant toutes les entités qui sont atteignables par densité à partir de e_2. Le même principe est appliqué pour toutes les entités cores dans le reste des $chunks$. Une entité qui n'appartient à aucun cluster, comme c'est le cas pour e_6, est considérée comme entité bruit. Mais nous ne pouvons pas affirmer cela à ce stade, car aucun échange d'informations n'est effectué, et une entité considérée comme bruit dans un $chunk$ peut être affectée à un cluster dans un autre $chunk$. Ainsi, l'identification des entités bruits se fera à la dernière étape.

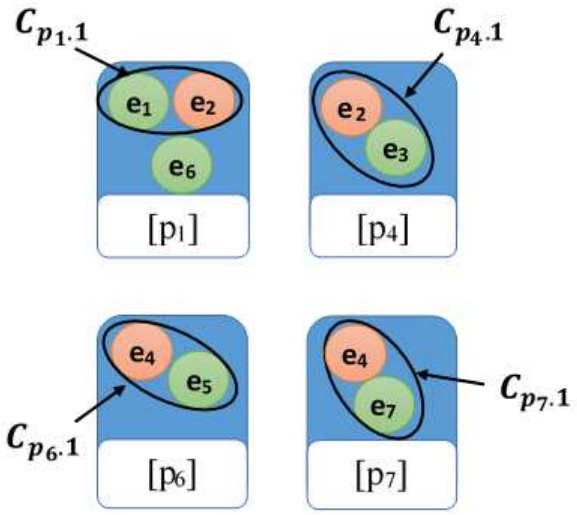


FIGURE 3.8 – Formation des clusters locaux dans chaque $chunk$

L'algorithme 5 calcule les clusters dans chaque $chunk$ généré lors de la distribution des données. Il itère sur l'ensemble des entités cores identifiées précédemment et crée pour chacune un cluster comportant l'entité core et ses voisins (ligne 6). Ensuite, l'algorithme vérifie parmi les voisins ajoutés ceux qui sont cores, et ajoute leurs voisins au cluster (lignes 7-9). L'algorithme ajoute récursivement les voisins des entités cores au cluster courant jusqu'à ce que toutes les entités cores soient vérifiées et l'expansion du cluster s'arrête sur les entités bordures. Le même processus est répété avec une autre entité core qui n'a pas encore été visitée, jusqu'à ce que toutes les entités cores soient affectées à des clusters. L'algorithme produit en sortie l'ensemble des clusters locaux.

Dans la prochaine section, nous montrons comment former les clusters finaux à partir des clusters locaux.

Algorithm 5 Clustering local

Input: chs : les chunks, cores : les entités cores

1: **for all** $[P] \in chs$ **do in parallel**
2: is-visited $= \emptyset$
3: **for all** $e \in [P]$ **do**
4: **if** $e \in cores$ and $e \notin$ is-visited **then**
5: is-visited $=$ isVisited $\cup \{e\}$
6: Create a new cluster $C = \{e\} \cup neighborhood_\epsilon(e)$
7: **for all** $e' \in C \mid e' \in cores$ and $e' \notin$ is-visited **do**
8: $C = C \cup \{e'\} \cup neighborhood_\epsilon(e')$
9: **end for**
10: **end if**
11: local-clusters $=$ local-clusters $\cup C$
12: **end for**
13: **end for**
14: **return** local-clusters

3.9 Génération des classes

Les clusters finaux représentent les classes composant le schéma. Dans notre approche, nous identifions les clusters qui s'étendent sur plusieurs $chunks$, et nous fusionnons les clusters locaux correspondant afin de former le clustering final. La fusion des clusters locaux est exécutée sur un seul nœud de calcul et produit le clustering final.

Dans notre approche, similairement aux algorithmes de clustering basés sur la densité, une entité e est assignée à un cluster C_i si e est atteignable par densité à partir d'une entité core dans C_i. Si cette même entité e se trouve aussi dans un autre cluster local C_j, cela signifie que e est aussi atteignable par densité à partir d'une entité core dans C_j. Si e est une entité core, alors elle représente un lien entre les entités dans les clusters C_i et C_j les rendant atteignables par densité les unes à partir des autres.

La figure 3.9 donne un aperçu du principe présenté ci-dessus ; les entités cores sont représentées en orange et les entités bordures en vert. Comme le montre cette figure, les entités dans les clusters C_1 et C_2 sont atteignables par densité à partir de l'entité core e_i, les rendant toutes atteignables par densité. Par conséquent, ces entités doivent former un même cluster dans le résultat final, et les clusters locaux correspondants sont donc fusionnés.

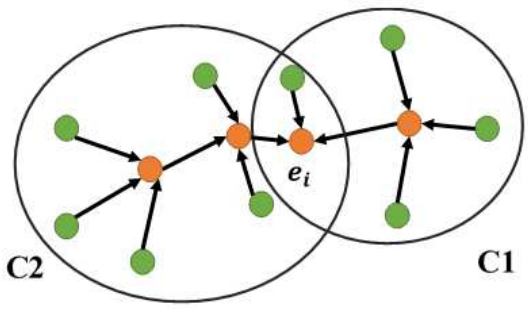


FIGURE 3.9 – Une illustration du principe de fusion des clusters locaux

Les clusters qui s'étendent sur plusieurs $chunks$ sont identifiés en recherchant les clusters locaux qui partagent une entité core commune et en les fusionnant. Si une entité bordure est assignée à différent clusters locaux durant l'étape de clustering, elle sera aléatoirement assignée à un de ces clusters durant l'étape de fusion. Toutes les entités qui ne sont pas assignées à un cluster sont considérées comme entités bruits. Ce processus produit les clusters finaux, assurant que le résultat est identique aux clusters générés par DBSCAN.

Exemple 3.9.1 *La figure 3.10 présente les clusters finaux obtenus en fusionnant les clusters locaux de l'exemple 3.8.1. Ainsi, les clusters $c_{p_1.1}$ et $c_{p_4.1}$ sont fusionnés car ils partagent une entité core commune e_2. Le résultat du clustering final représente les classes du schéma. Les propriétés de ces classes sont l'union de toutes les propriétés décrivant les entités à l'intérieur du cluster correspondant ($Classe_1 = \{p_1, p_2, p_3, p_4\}$ et $Classe_2 = \{p_2, p_5, p_6, p_7\}$). Les entités bruits comme e_6 ne sont pas considérées comme suffisamment représentatives pour générer une classe dans le schéma résultant.*

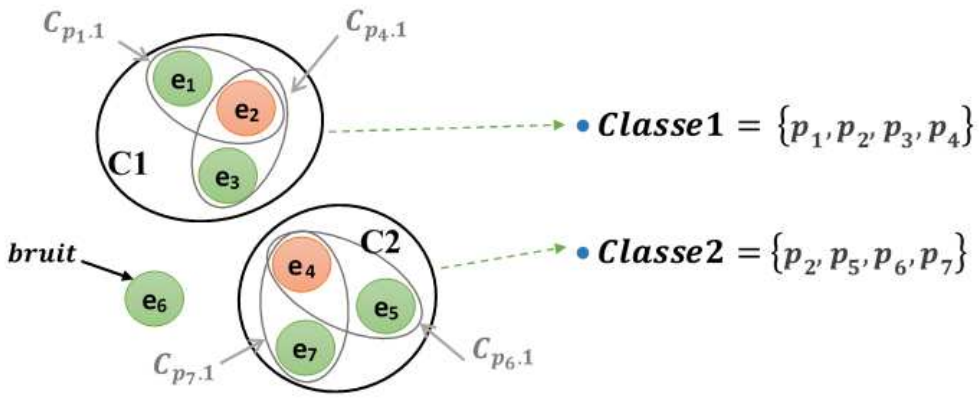


FIGURE 3.10 – Les clusters finaux correspondant aux classes du schéma découvert à partir du jeu de données D

L'algorithme 6 décrit le processus de fusion des clusters. Deux clusters sont fusionnés s'ils partagent une entité core commune. Ainsi, l'algorithme de fusion itère à travers les entités cores (lignes 2-6). Pour chaque core, les clusters contenant ce core sont identifiés (ligne 3) et fusionnés (ligne 5). L'étape finale est exécutée sur un seul nœud de calcul et n'est pas parallélisée.

Algorithm 6 Fusion des clusters locaux

Input: localClusters : les clusters locaux, cores : les entités cores

1: $clusters \leftarrow localClusters$
2: **for all** $e \in D \mid e \in cores$ **do**
3: $\quad lc_e = \{C \in clusters \mid e \in C\}$
4: $\quad clusters = clusters \setminus lc_e \cup (\cup_{C \in lc_e} C)$
5: **end for**
6: **return** clusters

3.10 La découverte de schéma appliquée à des patterns

Notre approche scalable de découverte de schéma forme des clusters d'entités similaires contenues dans une source de données RDF massive, qui ne peut pas être gérée par les approches d'extraction de schéma existantes à cause de la complexité du traitement. Le clustering est fait sur la base de la similarité structurelle des entités.

Nous avons également proposé une approche qui extrait une représentation condensée des entités d'un jeu de données RDF. Cette représentation consiste en un ensemble de patterns qui capturent toutes les combinaisons de propriétés qui représentent les entités dans la source de données.

Comme cette représentation conserve toutes les informations nécessaires à la découverte de schéma, notre algorithme de clustering peut s'appliquer sur les entités initiales comme sur les patterns et produire le même résultat de clustering. En effet, le clustering est fait en se basant sur la similarité structurelle des patterns, le clustering des patterns produit le même résultat que le clustering des entités initiales.

Cependant, le voisinage d'un pattern ne représente pas nécessairement des régions denses si nous ne considérons pas les entités qu'il représente. Durant le clustering des entités, nous définissons les cores en considérant le voisinage de ces dernières. Un pattern représente un ensemble d'entités et ceci doit être pris en compte durant le calcul de voisinage des patterns. Par conséquent, la définition des cores doit être adaptée afin de prendre en considération les patterns. Nous rappelons qu'un pattern est représenté par un ensemble de propriétés, et par le nombre d'entités décrites par cet ensemble de propriétés. Par conséquent, nous définissons un pattern core comme suit.

Définition 3.10.1 *Soit pt un pattern, et $N = \{pt_1, pt_2, \ldots, pt_n\}$ l'ensemble des voisins de pt. Le pattern pt est un core si la somme des entités représentées par pt et celles représentées par ses voisins est supérieure ou égale à $minPts$:*

$$|pt| + \sum_{i=1}^{n} |pt_i| \geq minPts$$

Dans la formule, la notation $|p|$ représente le nombre d'entités correspondant à un pattern p tel que défini dans 3.4.1.

La découverte de schéma appliquée aux patterns peut réduire d'un facteur important le nombre d'entrées pour l'algorithme de clustering. En effet, le nombre de patterns étant plus réduit que le nombre d'entités initiales, le clustering des patterns peut permettre une exécution plus rapide.

3.11 Conclusion

Dans ce chapitre, nous avons proposé une approche automatique de découverte du schéma implicite à partir de sources de données RDF massives. Notre approche s'appuie sur un nouvel algorithme de clustering basé sur la densité, distribué, qui forme des clusters d'entités structurellement similaires. Nous avons proposé une approche qui réduit le nombre de points en entrée de l'algorithme de clustering en remplaçant les entités par des patterns qui représentent toutes les combinaisons de propriétés décrivant les entités. Nous avons aussi introduit une nouvelle méthode de distribution des données permettant un clustering parallèle et rapide des entités d'un jeu de données RDF, tout en garantissant la comparaison de toutes les entités similaires, et ainsi en produisant le même résultat de clustering que l'algorithme DBSCAN. Les clusters résultants représentent les classes du schéma décrivant les entités d'une source de données RDF. Nous avons implémenté notre algorithme en utilisant Spark, une technologie big data offrant une exécution distribuée et rapide de notre algorithme qui permet le clustering de sources de données massives. Nous montrerons dans le chapitre des expérimentations à travers des évaluations détaillées la qualité du schéma produit par notre approche, ainsi que les performances obtenues par notre algorithme.

Notre approche permet de caractériser le contenu d'un jeu de données RDF en découvrant les classes qui représentent ses entités. Afin d'obtenir le schéma complet d'une source de données, le schéma extrait par notre approche peut être complété par la définition des liens entre les classes. Ceci peut être fait en analysant les liens existent entre les entités, comme proposé dans [65, 66], où deux types de liens ont été identifiés : (i) les liens sémantiques, correspondant à des propriétés qui lient deux entités (ii) les liens hiérarchiques, correspondant à la propriété $rdfs{:}subClassOf$. Enfin, les classes découvertes peuvent être annotées par les valeurs les plus fréquentes des propriétés $rdf{:}type$ [21, 20], ceci dans le but de définir la sémantique qu'elles représentent. Une classe créée suite au regroupement d'un ensemble d'entités $C = \{e_1, e_2, e_3, \ldots\}$ est annotée par la valeur la plus fréquente de la propriété $rdf : type$ décrivant les entités dans cet ensemble.

L'approche de découverte de schéma que nous proposons dans ce chapitre a été conçue pour les jeux de données RDF mais peut être utilisée sur d'autres formats de données comme *JSON* ou *XML*, dont les entités sont irrégulières et ne possèdent pas une structure prédéfinie.

Une amélioration intéressante pour notre approche serait de pondérer les propriétés en fonction de leur importance dans la description des entités. Ainsi, nous attribuerons un coefficient pour chaque propriété décrivant le jeu de données en utilisant leur distribution statistique. Ce coefficient serait utilisé dans la mesure de similarité qui détermine le voisinage des entités. Par exemple, une propriété p_1 qui décrit une grande partie des entités du jeu de données ne doit pas avoir le même poids sur le résultat de la similarité des entités qu'une propriété p_2 qui est plus sélective. Une propriété qui s'appliquerait à un grand nombre de classes ne devrait pas être considérée de la même façon qu'une propriété spécifique à une classe. Un tel coefficient offrirait plus de poids aux propriétés les plus spécifiques afin d'améliorer la qualité du schéma produit.

Une source de données RDF peut être sujette à des mises à jour fréquentes, où de nouvelles entités sont ajoutées aux données initiales. Suite à ces évolutions, le schéma généré pour la source peut rapidement devenir incohérent et ne plus constituer une bonne description de son contenu. Aussi, nous proposons dans le chapitre suivant une approche de découverte de schéma incrémentale, qui construit la description des données de façon évolutive suite aux mises à jour survenues dans une source de données.

Chapitre 4

Découverte incrémentale de schéma à partir de sources de données RDF

4.1 Introduction

Nous avons proposé dans le chapitre précédent une approche de découverte de schéma scalable capable d'extraire efficacement les classes implicites à partir des instances de grandes sources de données RDF. En plus du passage à l'échelle, les approches existantes de découverte de schéma sont confrontées à un autre problème qui est celui de l'évolution des sources. Ces dernières peuvent être sujettes à de fréquentes évolutions au fil du temps et faire l'objet d'insertion de nouvelles instances. Par exemple, entre la version 3.5 et la version 3.9 de DBpedia, le nombre de triplets ayant la classe $Person$ comme sujet a été multiplié par 45 [89]. En raison de cette évolution, la possibilité d'effectuer des mises à jour sur le schéma de façon incrémentale a émergé comme un nouveau défi. En effet, certaines approches existantes nécessitent la disponibilité de la totalité de la source de données après son évolution avant de procéder à l'actualisation du schéma [21, 65, 101]. D'autre approches introduisent la notion d'entité fictive associée à chaque cluster afin de classer les entités nouvellement insérées dans les clusters existants, sans remettre en cause le schéma déjà découvert ; les possibilité de créer de nouvelles classes ou de fusionner des classes existantes ne sont pas étudiées [66]. Dans les applications où le temps de réponse est un facteur important et où la taille du jeu de données est grande, exécuter le processus de découverte de schéma sur la globalité du jeu de données afin de modifier le schéma après une évolution est un processus très coûteux, particulièrement si l'approche de découverte de schéma s'appuie sur un algorithme de clustering. Cela met en évidence le besoin d'approches qui peuvent mettre à jour de manière incrémentale un schéma après chaque évolution des données de la source RDF considérée.

Dans ce chapitre, nous proposons, à partir de l'approche décrite au chapitre précédent, une version incrémentale

du processus de découverte de schéma qui permet de répercuter dans ce dernier les évolutions survenues au niveau des sources de données. Notre approche s'appuie sur un algorithme de clustering incrémental basé sur la densité, qui construit et met à jour les clusters représentant les classes du schéma. Notre approche met à jour les classes du schéma décrivant une source de données RDF de façon incrémentale après chaque insertion de données, afin de le garder cohérent avec l'évolution des données, ceci en assurant que le résultat est le même que celui produit par l'algorithme de clustering appliqué sur l'ensemble des données en une exécution. De plus, notre algorithme de clustering incrémental est conçu pour s'exécuter en parallèle afin de garantir son efficacité sur les grandes sources de données. Le code source de l'implémentation de notre approche en utilisant Spark est disponible en ligne [1].

Ce chapitre est organisé comme suit. Nous décrivons dans la section 4.2 la problématique adressée dans ce chapitre. La section 4.3 introduit notre approche incrémentale de découverte de schéma et donne une vue globale de notre proposition. La distribution des données est détaillée dans la section 4.4, l'identification des entités cores est décrite dans la section 4.5 et la section 4.6 montre comment le nouveau schéma est construit. Enfin, la section 4.7 conclut le chapitre et présente quelques perspectives.

4.2 Définition de la problématique

Considérons une source de données RDF D décrite par un schéma S. Supposons que l'ensemble d'entités noté Δ_D est ajouté à la source de données initiale D produisant l'ensemble $D' = D \cup \Delta_D$. Cette évolution du jeu de données D peut rendre le schéma S incohérent avec la source de données D'. En d'autres termes, le schéma S ne représente plus une bonne description de la source de données D'.

Exemple 4.2.1 *Considérons une source de données D (figure 4.1.a) décrite par l'ensemble de propriétés $\mathcal{P} = \{p_i \mid i \in [1, 9]\}$ et comportant les entités $D = \{e_i \mid i \in [1, 10]\}$.*

La figure 4.1 présente un exemple de source de données RDF (figure 4.1.a) et les classes du schéma correspondant (figure 4.1.c). La figure montre le regroupement en clusters (figure 4.1.b) des entités du jeu de données D en utilisant notre algorithme de clustering introduit dans le chapitre 3. Les clusters représentent les classes du schéma, et chaque classe est décrite par l'ensemble des propriétés décrivant les entités à l'intérieur du cluster correspondant.

Supposons qu'un ensemble d'entités $\Delta_D = \{e'_i \mid i \in [1, 7]\}$ est ajouté à D, où chaque entité est décrite de la façon suivante :

$\overline{e'_1} = \{p_1, p_5, p_8\}$, $\overline{e'_2} = \{p_1, p_3, p_5, p_8\}$, $\overline{e'_3} = \{p_9, p_{10}, p_{11}, p_{12}\}$, $\overline{e'_4} = \{p_9, p_{10}, p_{11}\}$, $\overline{e'_5} = \{p_{10}, p_{11}, p_{12}\}$,
$\overline{e'_6} = \{p_1, p_3, p_4\}$, $\overline{e'_7} = \{p_5, p_6, p_7\}$.

1. https://github.com/BOUHAMOUM/incremental_sc_dbscan.git

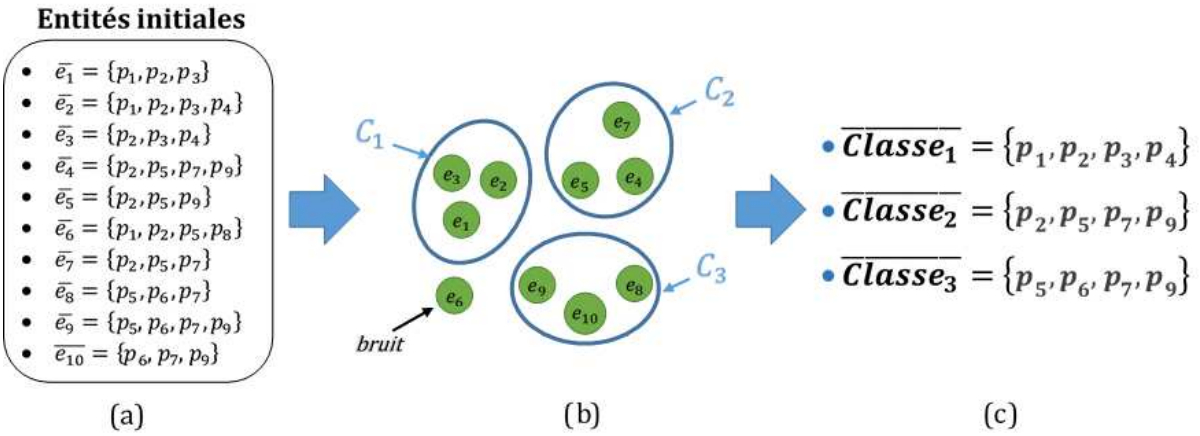


FIGURE 4.1 – Exemple d'un ensemble d'entités et le schéma correspondant

Nous pouvons voir qu'après l'ajout de l'ensemble d'entités Δ_D au jeu de données D, le schéma extrait pour D n'est plus cohérent avec les instances du jeu de données. Par exemple, il n'existe pas de classe dans le schéma représentant les entités e'_1, e'_2 et e'_3. Une solution serait de refaire le clustering sur la totalité du jeu de données formé à partir de l'union de D et Δ_D. Cependant, si les deux ensembles de données sont massifs et si les évolutions sont fréquentes, ce processus serait coûteux.

Pour faire face à ce problème, nous faisons les hypothèses suivantes :

1. Le jeu de données D et l'ensemble d'entités Δ_D nouvellement insérées peuvent être massifs. En effet, à l'ère du Big Data, les données RDF peuvent être produites en grand volume, tout comme les autres types de données.
2. Le schéma S décrivant le jeu de données D a été généré en utilisant une approche qui s'appuie sur un algorithme de clustering basé sur le principe de densité. Comme discuté dans le chapitre précédent, notre travail s'appuie sur l'algorithme de clustering basé sur la densité (DBSCAN) [28] dont nous nous sommes inspirés pour proposer notre approche d'extraction de schéma.
3. La similarité entre les entités est évaluée en utilisant *l'indice de Jaccard* [56] :

$$\forall e_i, e_j \in D, J(e_i, e_j) = \frac{|\overline{e_i} \cap \overline{e_j}|}{|\overline{e_i} \cup \overline{e_j}|}$$

4. Le voisinage de chaque entité $e \in D$ est connu. Nous supposons que le voisinage des entités regroupées dans des clusters a été calculé et sauvegardé lors des extractions précédentes du schéma en utilisant l'algorithme de clustering.

Dans ce chapitre, nous adressons le problème de découverte de schéma incrémentale à partir des sources de données RDF qui évoluent par l'ajout de nouvelles entités. Notre objectif est de proposer une approche qui

met à jour le schéma S décrivant le jeu de données D en considérant les entités Δ_D nouvellement insérées sans avoir à ré-exécuter le processus de clustering sur la totalité des données. Pour cela, il faut modifier les classes du schéma impactées par l'insertion des nouvelles entités ou, si nécessaire, créer de nouvelles classes décrivant ces dernières. Le schéma S' résultant de la propagation des modifications sur l'ensemble des classes existantes doit être un schéma descriptif qui représente la globalité du jeu de données, composé du jeu de données initial D et de l'ensemble d'entités Δ_D nouvellement insérées ($D \cup \Delta_D$). Nous détaillons notre proposition dans les sections suivantes.

4.3 Principe d'une approche de découverte incrémentale de schéma

Nous proposons dans ce chapitre une approche incrémentale de découverte de schéma, dont le but est de maintenir le schéma décrivant une source de données massive cohérent avec les données après les insertions d'entités dans celles-ci. Notre approche s'appuie sur un algorithme de clustering incrémental basé sur la densité et distribué. Afin de gérer les grands jeux de données qui évoluent fréquemment, le clustering est limité aux nouvelles entités et à leurs voisins dans l'ensemble des anciennes entités. Le clustering des nouvelles entités et la modification des clusters dans leurs voisinages assurent que le résultat est le même que celui produit par DBSCAN exécuté sur tout le jeu de données [29].

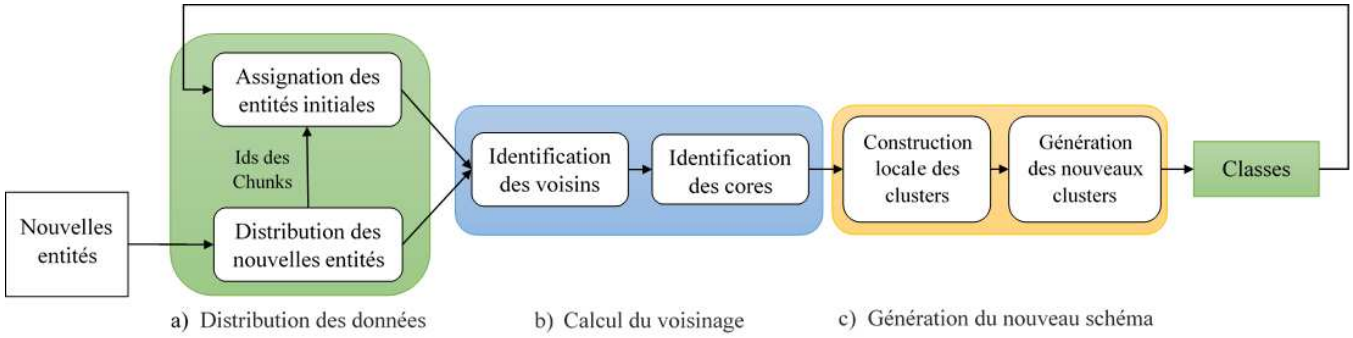


FIGURE 4.2 – Vue globale de l'approche incrémentale de découverte de schéma

Notre approche est composée de trois phases principales, parallélisées et implémentées en utilisant la technologie Big Data Spark [104]. La figure 4.2 illustre ces différentes phases.

Tout d'abord, les données sont divisées en sous-ensembles et distribuées sur les différents nœuds de calcul. Cette distribution des données crée les $chunks$ introduits dans le chapitre 3. Les $chunks$ comportent des entités partageant des propriétés communes et qui sont susceptibles d'être similaires. La distribution est limitée aux nouvelles entités de Δ_D et aux anciennes entités de D qui peuvent être similaires à une nouvelle entité de Δ_D. Comme présenté dans la figure 4.2.a, les nouvelles entités sont distribuées. Les anciennes entités contenues dans le jeu de données initial et qui peuvent être similaires aux nouvelles entités sont ensuite assignées aux $chunks$ créés.

De cette manière, toutes les entités qui sont susceptibles d'être similaires aux nouvelles, aussi bien dans D que dans Δ_D, sont groupées dans au moins un $chunk$. Ainsi, toutes les comparaisons nécessaires à l'identification des entités voisines seront effectuées durant le clustering.

Ensuite, en parallèle sur chaque nœud de calcul, le voisinage de chaque nouvelle entité est calculé. Les listes de leurs voisins qui peuvent être distribués sur plusieurs $chunks$ sont alors fusionnées. A la fin de cette étape, les entités cores ayant un voisinage dense sont identifiées. Cette étape est présentée dans la figure 4.2.b.

Enfin, en considérant sur le voisinage des nouvelles entités, l'ensemble des clusters est calculé localement sur chaque $chunk$: (i) d'anciens clusters peuvent être modifiés en ajoutant de nouveaux éléments, (ii) d'anciens clusters peuvent être fusionnés, et (iii) de nouveaux clusters peuvent être créés afin de représenter des nouvelles entités. Les clusters produits localement sur chaque $chunk$ sont alors fusionnés pour générer les nouveaux clusters qui représentent les classes du nouveau schéma, comme illustré dans la figure 4.2.c.

Nous avons implémenté notre algorithme en utilisant Spark [104] afin qu'il soit applicable à de grands jeux de données. Les sections suivantes détaillent notre proposition.

4.4 La distribution des données

Notre approche a pour but de modifier les clusters existants en tenant compte uniquement des nouvelles entités et de leurs voisinages. Par conséquent, nous devons calculer le voisinage des nouvelles entités. Cependant, les données traitées dans notre contexte sont massives et le calcul de voisinage des nouvelles entités requiert un nombre élevé de comparaisons, ce qui représente un processus coûteux. Dans notre approche, nous proposons de distribuer les nouvelles entités suivant le principe introduit dans le chapitre précédent, où le jeu de données est divisé en sous-ensembles par rapport aux propriétés qui décrivent les entités. La comparaison des entités est ainsi effectuée dans chaque $chunk$ en parallèle, ce qui accélère le processus de clustering.

Nous rappelons que l'intuition derrière notre méthode de distribution est de grouper dans un même $chunk$ toutes les entités partageant des propriétés communes afin d'assurer l'identification de toutes les paires d'entités similaires. Ainsi, les entités similaires sont regroupées dans au moins un $chunk$, et seront comparées durant le calcul de voisinage des entités. Dans notre approche incrémentale, la distribution des entités est limitée uniquement aux nouvelles entités, et aux anciennes entités partageant des propriétés avec les nouvelles et qui sont susceptibles d'être similaires à ces dernières.

Dans cette section, nous expliquons tout d'abord l'assignation des entités nouvellement insérées à différents $chunks$. Ensuite, nous montrons comment sont identifiées les anciennes entités qui peuvent être similaires aux nouvelles et comment elles sont assignées aux $chunks$.

4.4.1 Distribution des nouvelles entités

Dans notre algorithme incrémental, nous distribuons d'abord les nouvelles entités par rapport aux propriétés les décrivant. La comparaison d'une entité avec toutes ses voisines est assurée en la distribuant suivant ses propriétés sur différents $chunks$. Afin d'améliorer les performances de notre approche, nous ne considérons pas toutes les propriétés qui décrivent les entités. Ainsi, cette distribution est optimisée en utilisant le *seuil de dissimilarité* introduit dans le chapitre précédent afin de réduire le nombre de propriétés à considérer pour chaque entité. L'intuition derrière le *seuil de dissimilarité* est que deux entités sont similaires si elles partagent un nombre suffisant de propriétés. Ce nombre de propriétés dépend du seuil de similarité et du nombre de propriétés décrivant une entité. Le seuil représente le nombre de propriétés à considérer pour une entité e_i afin de décider si cette entité peut être similaire à une entité quelconque e_j. Lors de la recherche de voisinage, le *seuil de dissimilarité* permet la sélection d'entités candidates qui peuvent être similaires en considérant seulement un nombre réduit de propriétés. Par conséquent, le nombre d'entités à considérer lors de la recherche de voisinage d'une entité donnée est réduit. Ainsi, nous limitons la duplication des entités dans les différents $chunks$ et réduisons le coût du processus de comparaison en évitant des comparaisons inutiles.

Nous avons prouvé dans le chapitre précédent que l'utilisation du *seuil de dissimilarité* réduit la taille des $chunks$ tout en garantissant que toutes les comparaisons pertinentes, nécessaires à la détection des entités similaires sont effectuées au moins une fois. Dans notre approche incrémentale, ce principe de distribution assure que les nouvelles entités sont regroupées avec tous leurs voisins.

L'utilisation du *seuil de dissimilarité* exige de choisir les propriétés dans un ordre bien défini. Dans notre approche, les propriétés sont ordonnées selon leur sélectivité afin de créer des $chunks$ de tailles réduites tout en assurant le regroupement des entités similaires, et d'accélérer le calcul de voisinage dans chaque $chunk$. L'ordonnancement est fait en considérant la sélectivité des propriétés décrivant l'ensemble des nouvelles entités, ceci car la comparaison des nouvelles entités représente le traitement le plus coûteux dans notre approche et que nous visons à l'optimiser. Sachant que les comparaisons sont faites entre les nouvelles entités et les entités dans un même $chunk$, et que les anciennes entités ne sont pas comparées entre elles, en considérant la sélectivité des propriétés décrivant les nouvelles entités, nous créons des $chunks$ comportant moins de nouvelles entités et ainsi nous réduisons le nombre de comparaisons dans chaque $chunk$.

Exemple 4.4.1 *Considérons le jeu de données initial D et l'ensemble d'entités insérées Δ_D introduit dans l'exemple 4.2.1. La sélectivité des propriétés est définie suivant leur fréquence d'apparition dans la description des entités : $p_4 <_{\mathcal{P}} p_6 <_{\mathcal{P}} p_7 <_{\mathcal{P}} p_3 <_{\mathcal{P}} p_8 <_{\mathcal{P}} p_9 <_{\mathcal{P}} p_{12} <_{\mathcal{P}} p_1 <_{\mathcal{P}} p_5 <_{\mathcal{P}} p_{10}$. Par exemple, la propriété p_4 est la moins fréquente et ainsi la plus sélective.*

La distribution des nouvelles entités Δ_D à travers les $chunks$ produit le résultat présenté dans la figure 4.3. Par exemple, l'entité e'_3 est assignée aux $chunks$ $[p_9]$ et $[p_{10}]$. Sachant que $\epsilon = 0.7$, son seuil de dissimilarité est égal à

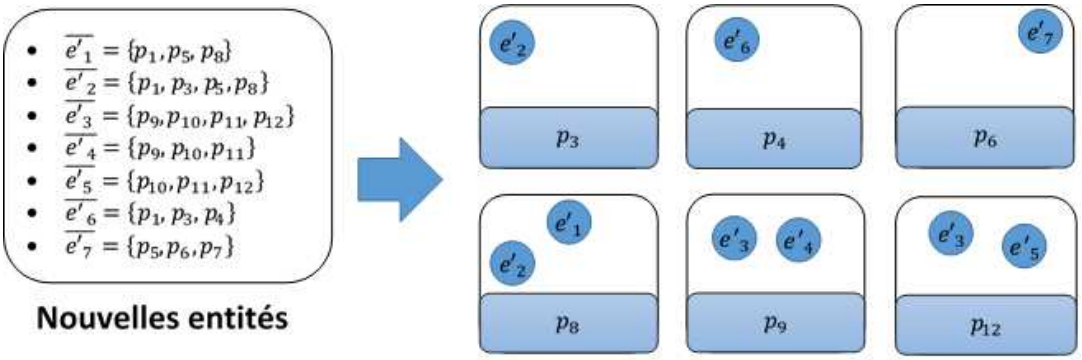


FIGURE 4.3 – Les $chunks$ générés après la distributions des entités mises à jour Δ_D

$dt(e'_3) = 4 - \lceil 0.7 \times 4 \rceil + 1 = 2$, *et les deux propriétés les plus sélectives décrivant* e'_3 *sont* p_9 *et* p_{10}*. De cette manière,* e'_3 *sera groupée et comparée avec toutes les autres entités à l'intérieur de ces* $chunks$*, assurant sa comparaison avec tous ses voisins.*

L'algorithme 7 décrit la distribution des nouvelles entités sur les $chunks$. Il prend en entrée l'ensemble des entités nouvellement insérées et le seuil de similarité ϵ. La distribution des entités est effectuée en parallèle sur l'ensemble des $chunks$ (ligne 1). L'algorithme définit les $chunks$ auxquels une entité est assignée en calculant son *ensemble de propriétés de comparaison* (lignes 2-4). L'exécution parallèle produit des $chunks$ sur chaque nœud de calcul. Ces derniers sont consolidés afin de produire les $chunks$ finaux.

Algorithm 7 Distribution des nouvelles entités

Input: the new dataset Δ_D, the similarity threshold ϵ
1: **for all** entity e' in Δ_D **do in parallel**
2: **for all** property $p \in cs(e')$ **do**
3: $[p] = [p] \cup \{e'\}$
4: **end for**
5: **end for**
6: Merge the chunks generated by the parallel execution for the same properties
7: **return** the chunks

Les entités de Δ_D sont maintenant distribuées sur un ensemble de $chunks$. Toutefois, l'ajout de nouvelles entités nécessite la mise à jour des clusters se trouvant dans le voisinage de ces dernières. Par conséquent, les entités de D susceptibles d'être similaires à celles de Δ_D doivent être identifiées et assignées aux $chunks$ créés afin qu'elles soient comparées. C'est l'objet de la section suivante.

4.4.2 L'assignation des anciennes entités

Comme expliqué précédemment, les clusters qui peuvent être modifiés suite à l'insertion de nouvelles entités sont ceux comportant des anciennes entités dans le voisinage des nouvelles. Les anciennes entités sont les entités du jeu de données D avant l'addition de l'ensemble des nouvelles entités Δ_D. Les entités dans l'ϵ-voisinage d'une

entité ajoutée peuvent changer de rôle dans la formation des clusters, i.e. les entités qui étaient bruits ou bordures peuvent devenir bordures ou cores. Ces changements de rôle impliquent la modification des clusters comportant ces entités. Par conséquent, les entités dans D qui sont voisines de nouvelles entités doivent être identifiées. Les anciennes entités partageant des propriétés communes avec les nouvelles peuvent être similaires aux nouvelles et sont ainsi distribuées sur les $chunks$ créés.

Afin de distribuer les entités de D, nous déterminons d'abord les propriétés à considérer : pour chaque entité $e \in D$, nous calculons l'ensemble de propriétés de comparaison $cs(e)$ (voir définition 3.6.4) qui comporte les propriétés à considérer afin de déterminer les $chunks$ auxquels l'entité doit être assignée. Remarquons qu'aucun nouveau $chunk$ n'est créé durant l'assignation des anciennes entités : les entités dans D sont seulement assignées aux $chunks$ déjà créés durant la distribution des nouvelles entités. En effet, l'assignation des anciennes entités a pour but l'identification de celles qui sont similaires aux nouvelles entités. Comme le voisinage des anciennes entités est connu, il n'est pas nécessaire de comparer ces dernières entre elles, et la création de $chunks$ comportant uniquement des anciennes entités est inutile. Par conséquent, une ancienne entité e est assignée au $chunk$ $[p]$ si e comporte la propriété p et que le $chunk$ $[p]$ a été créé durant la distribution des nouvelles entités : $e \in [p]$ si $p \in cs(e)$ et $\exists e' \in \Delta_D, e' \in [p]$

Le principe de distribution utilisé dans notre travail garantit que chaque entité est regroupée avec toutes les entités représentant des voisins potentiels dans $D \cup \Delta_D$. Ainsi, les nouvelles entités sont comparées avec toutes leurs entités similaires afin de définir leurs voisinages. Les clusters qui doivent être modifiés ou créés sont ainsi identifiés.

Exemple 4.4.2 *Dans notre exemple, le résultat de l'assignation des anciennes entités est présenté dans la figure 4.4 où les anciennes entités sont représentées en vert.*

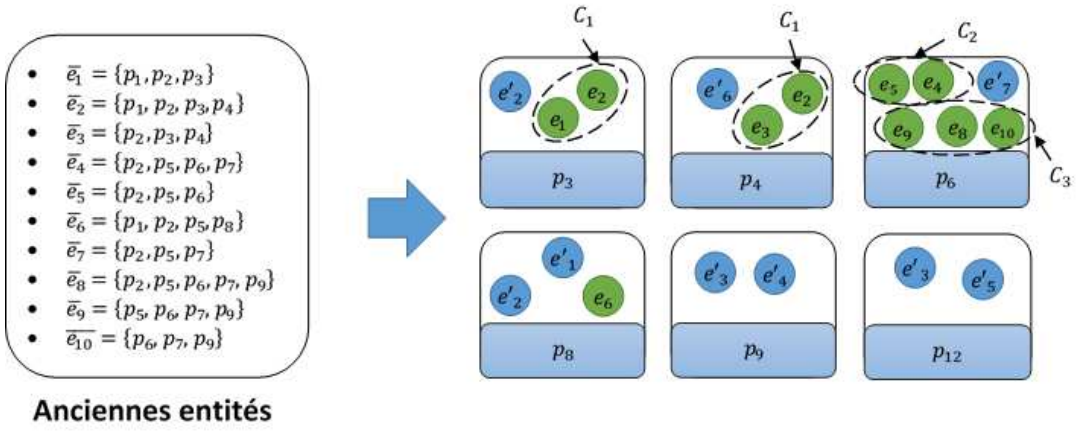


FIGURE 4.4 – Assignation des anciennes entités aux $chunks$ créés

Par exemple, le seuil de dissimilarité *de l'entité e_4 est égal à 2. Les deux propriétés les plus sélectives décrivant e_4 sont p_6 et p_7 mais e_4 est assignée uniquement au $chunk$ $[p_6]$. En effet, $[p_7]$ n'a pas été créée durant la distribution*

des nouvelles entités. Les $chunks$ créés après l'assignation des anciennes entités représentent les $chunks$ finaux.

L'algorithme 8 présente l'assignation des anciennes entités aux $chunks$ créés par la distribution des nouvelles entités de Δ_D. Il requiert l'ensemble des anciennes entités D, le *seuil de similarité* et les $chunks$ déjà créés.

L'algorithme identifie les $chunks$ auxquels une ancienne entité doit être assignée par rapport à l'ensemble de propriétés décrivant l'entité et au *seuil de dissimilarité* (ligne 2). Une entité e ayant la propriété p est assignée au $chunk$ $[p]$ si le $chunk$ existe (ligne 3). Enfin, les $chunks$ générés en parallèle par les algorithmes 7 et 8 sont fusionnés afin de produire les $chunks$ finaux.

Algorithm 8 Assignation des anciennes entités

Input: the old entities d, the similarity threshold ϵ, the list of created chunks CH
1: **for all** entity $e \in D$ **do in parallel**
2: **for all** property $p \in cs(e)$ **do**
3: **if** $[p] \in CH$ **then**
4: $[p] = [p] \cup \{e\}$
5: **end if**
6: **end for**
7: **end for**
8: Merge the chunks generated by the parallel execution for the same properties
9: **return** the chunks

La distribution des entités à travers des $chunks$ permet d'accélérer et de paralléliser la recherche de voisinage, qui représente l'opération la plus coûteuse dans un algorithme de clustering par densité. La section qui suit décrit le calcul de voisinage de chaque nouvelle entité.

4.5 Calcul de voisinage des nouvelles entités

La mise à jour du schéma suite à l'insertion de nouvelles entités revient à mettre à jour l'ensemble des clusters représentant les classes du schéma afin de les garder cohérentes avec le jeu de données. Pour cela, il est nécessaire de calculer le voisinage des nouvelles entités en considérant les nouvelles et les anciennes entités. Dans cette section, nous décrivons d'abord comment le voisinage de chaque nouvelle entité est calculé. Ensuite, nous montrons comment sont identifiées les entités cores afin de former les clusters.

Les $chunks$ créés durant la distribution des données contiennent des entités qui sont susceptibles d'être similaires. Dans cette section, nous présentons alors le calcul de la similarité entre les entités nouvellement ajoutées comparées à toutes les entités à l'intérieur d'un même $chunk$ afin d'identifier *l'ϵ-voisinage* des nouvelles entités. La similarité entre les entités est évaluée en utilisant *l'indice de Jaccard*.

Comme défini dans le chapitre précédent, une entité peut avoir les rôles suivant durant le clustering, selon son *l'ϵ-voisinage* :

- entité core ayant au moins $minPts$ entités dans son ϵ-*voisinage*,
- entité bordure, qui n'est pas core mais comporte au moins une entité core dans son ϵ-*voisinage*,
- entité bruit qui n'est pas core et ne comporte aucune entité core dans son ϵ-*voisinage*. Ces dernières ne sont assignées à aucun cluster.

L'ϵ*-voisinage* est calculé pour chaque nouvelle entité e' dans chaque $chunk$ en comparant e' à toutes les entités dans le même $chunk$. Cette comparaison inclut les nouvelles et les anciennes entités. *L'*ϵ*-voisinage* est calculé en parallèle dans les différents $chunks$ de façon indépendante sans échange d'informations entre les nœuds de calcul. Le calcul des voisinages des anciennes entités n'est pas nécessaire car ils ont déjà été calculés durant le calcul des anciens clusters. Les anciennes entités sont comparées aux nouvelles seulement durant le calcul de voisinage de ces dernières, et ainsi, le voisinage d'une ancienne est mis à jour si cette entité est similaire à une nouvelle entité. En effet, les anciennes entités bordures ou bruits peuvent devenir entités cores ou bordures, ce qui implique la modification des anciens clusters auxquels elles appartiennent. Cette opération produit la liste des voisins pour chaque entité à l'intérieur de chaque $chunk$.

Comme le voisinage d'une entité peut être distribué à travers différents $chunks$, les voisinages découverts dans chaque $chunk$ sont consolidés et la liste des voisins de chaque entité dans l'ensemble du jeu de données est construite. Grâce à cela, les voisins des nouvelles entités dans tout le jeu de données sont identifiés.

À partir des voisinages calculés, les entités cores, à partir desquelles les clusters seront initiés sont identifiés. Les entités cores sont les entités ayant un nombre de voisins supérieur ou égale au paramètre de densité $minPts$. Les anciennes entités bordures ou bruits qui sont similaires à de nouvelles entités peuvent devenir entités cores ou bordures : l'ajout de nouvelles entités à leur ϵ-*voisinage* peut rendre le nombre de leurs voisins supérieur ou égale à $minPts$ et ainsi elles deviennent des entités cores. Suite aux changements dans le voisinage des anciennes entités, les clusters existants avant l'insertion des nouvelles entités doivent être modifiés.

Les anciennes entités qui ne sont pas similaires à au moins une nouvelle entité dans les $chunks$ sont supprimées car elles n'impliqueront aucun changement dans les clusters existants : leurs clusters initiaux ne seront pas modifiés, et elles ne seront assignées à aucun nouveau cluster. De plus, les $chunks$ comportant uniquement des nouvelles entités dont le voisinage est vide sont supprimés, car aucun cluster ne sera modifié ou formé dans ces $chunks$.

Exemple 4.5.1 *Considérons les* $chunks$ *de notre exemple (4.4), les nouvelles entités (représentées dans un cercle bleu) sont comparées avec toutes les entités dans le même* $chunk$*. Par exemple, dans le* $chunk$ $[p_4]$*, la nouvelle entité* e'_6 *est comparée avec les entités* e_2 *et* e_3*.*

Dans notre exemple, le paramètre ϵ *est fixé à* $\epsilon = 0.7$ *et* $minPts$ *à* $minPts = 2$*. Les entités cores sont ainsi* e_2*,* e_4*,* e_9 *dans les anciennes entités, et* e'_1*,* e'_3*,* e'_7 *dans l'ensemble des nouvelles entités. Les entités cores sont représentées en rouge dans la figure 4.5. Par exemple, l'entité* e'_3 *possède deux voisins* e'_4 *et* e'_5 *respectivement*

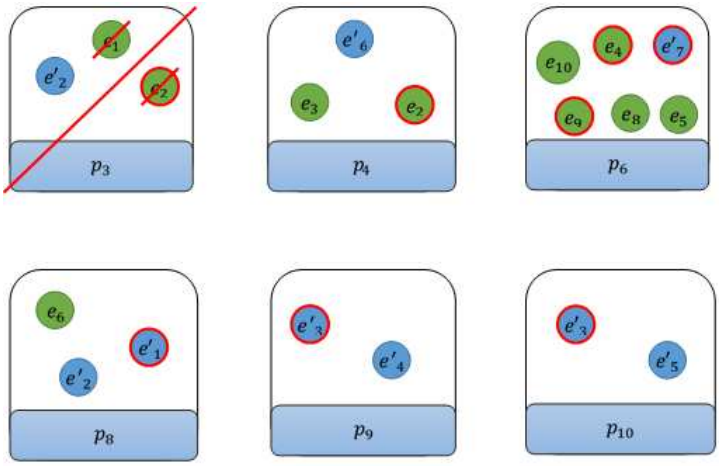


FIGURE 4.5 – L'identification des cores dans chaque $chunk$

dans les $chunks$ $[p_9]$ et $[p_{10}]$. La fusion des voisins découverts dans chaque $chunk$ produit la liste globale des voisins de l'entité e'_3.

Le $chunk$ $[p_3]$ est supprimé car les nouvelles entités que comporte le $chunk$, comme e'_2, ne possèdent aucun voisin parmi les nouvelles entités.

L'algorithme 9 décrit le calcul du voisinage des nouvelles entités en parallèle dans chaque $chunk$.

Algorithm 9 Calcul de voisinage

Input: les chunks CH, le seuil de similarité ϵ, le seuil de densité $minPts$

1: **for all** $[p] \in CH$ **do in parallel**
2: **for all** $e' \in [p] \mid e' \in \Delta_D$ **do**
3: $neighborhood_\epsilon(e') = \{e \in [p] \mid J(e, e') \geq \epsilon\}$
4: **end for**
5: **for all** $e \in [p] \mid e \in D$ **do**
6: $neighborhood_\epsilon(e) = neighborhood_\epsilon(e) \cup \{e' \in \Delta_D \mid e \in neighborhood_\epsilon(e')\}$
7: **end for**
8: **end for**
9: Merge the local neighborhoods to compute the complete list of neighbors of each entity
10: **for all** $e \in D$ **do**
11: **if** $neighborhood_\epsilon(e) \cap \Delta_D = \emptyset$ **then**
12: $D = D \setminus \{e\}$
13: **end if**
14: **end for**
15: **for all** $e \in D \cup \Delta_D$ **do**
16: **if** $|neighborhood_\epsilon(e)| \geq minPts$ **then**
17: $cores = cores \cup \{e\}$
18: **end if**
19: **end for**
20: **return** cores

L'algorithme calcule d'abord en parallèle dans chaque $chunk$ le voisinage des nouvelles entités (lignes 2-4). Si une entité est similaire à une ancienne, le voisinage de cette ancienne entité est modifié (lignes 5-7). Ensuite, les listes de voisins calculées dans chaque $chunk$ sont fusionnées : les différents nœuds de calcul s'échangent leurs

listes partielles de voisins afin de construire pour chaque entité sa liste de voisins dans tout le jeu de données. Les anciennes entités qui n'ont pas de nouvelles entités dans leurs voisinages sont supprimées des $chunks$ (lignes 10-14). Enfin, les entités ayant un nombre de voisin supérieur ou égale à $minPts$ sont identifiées comme étant des entités cores (lignes 15-19).

En calculant le voisinage des nouvelles entités, nous avons identifié les entités cores afin de former les nouveaux clusters, ainsi que les anciens clusters impactés par l'insertion des nouvelles entités. Nous décrivons dans la suite la restructuration des clusters existants, ainsi que la création de nouveaux clusters.

4.6 Génération du nouveau schéma

Pour générer le nouveau schéma décrivant la totalité du jeu de données incluant les anciennes et les nouvelles entités, nous modifions d'abord les clusters localement dans chaque $chunk$ en considérant le voisinage des nouvelles entités. Cette opération est effectuée en parallèle dans chaque $chunk$ et produit des clusters locaux qui regroupent des entités similaires à l'intérieur d'un même $chunk$. Dans un deuxième temps, les clusters locaux qui doivent fusionner sont traités. Enfin, le nouveau schéma est généré en propageant les modifications faites sur les anciens clusters à travers tout le jeu de données. Nous décrivons dans cette section les modifications appliquées sur les clusters dans les $chunks$, et la construction du nouveau schéma.

4.6.1 Modification locale des clusters

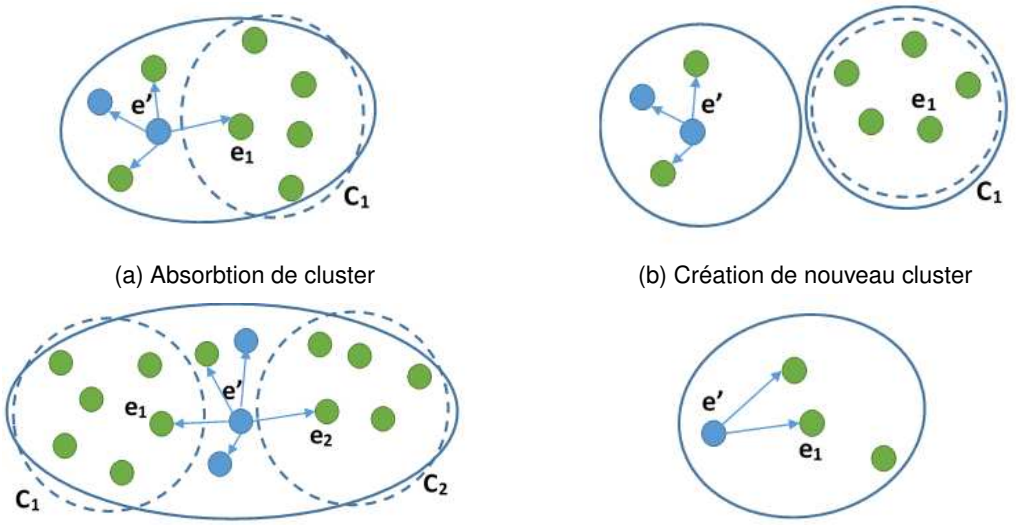
À partir des voisinages des nouvelles entités dans Δ_D calculés durant la phase précédente, nous distinguons différentes modifications possibles à appliquer sur l'ensemble les clusters extraits à partir du jeu de données initial D : (i) des clusters existants peuvent contenir de nouveaux éléments, (ii) des clusters peuvent fusionner et (iii) de nouveaux clusters peuvent être créés à partir des nouvelles entités cores suivant le principe d'atteignabilité par densité.

Les éléments à considérer dans notre algorithme incrémental sont les nouvelles entités et les anciennes entités dont le voisinage comporte des nouvelles entités : les clusters qui doivent être modifiés sont ceux qui comportent des éléments assignés dans des $chunks$ et qui sont voisins des nouvelles entités ; i.e. un cluster C est modifié si $\exists e \in C$, $\exists [p]$ tel que $e \in [p]$ et $\exists e' \in [p]$ tel que $e' \in neighborhood_\epsilon(e)$.

En effet, après l'insertion d'une nouvelle entité e', les entités dans le voisinage de e' peuvent créer de nouvelles connections par densité, par exemple, soient e_1 et e_2 deux entités non atteignable par densité. L'ajout de e' au jeu de données peut créer la chaîne $e_1, \cdots, e', \cdots, e_2$, ce qui crée une connexion entre les deux entités les rendant atteignables par densité. L'identification de nouvelles connections implique la modification des clusters existants. De plus, les entités qui ne sont pas dans le voisinage d'une nouvelle entité ne créent pas de nouvelles connections et ainsi leurs clusters restent inchangés.

En considérant les entités cores calculées durant la phase précédente, nous distinguons les modifications suivantes à appliquer sur l'ensemble des clusters existants :

- Si l'ϵ-*voisinage* d'une nouvelle entité $e' \in \Delta_D$ comporte une ancienne entité core $e \in D$ qui appartient à un ancien cluster C, alors l'entité e' est assignée à C. Si e' est une entité core alors le cluster C est également étendu en ajoutant toutes les entités atteignables par densité à partir de e'. Ce cas de figure est illustré dans la figure 4.6a. La nouvelle entité core e' est voisine d'une ancienne entité core e_1, ainsi, e' et ses voisins sont absorbés par le cluster C_1 comportant l'entité e_1.
- Si une nouvelle entité core $e \in D \cup \Delta_D$ n'a aucune ancienne entité core dans son ϵ-*voisinage*, alors un nouveau cluster est créé et les entités qui sont atteignables par densité à partir de e sont ajoutées à ce cluster. Dans la figure 4.6b, l'entité core e' crée un nouveau cluster à partir des entités qui lui sont atteignables par densité.
- Si l'ϵ-*voisinage* d'une entité core $e \in D \cup \Delta_D$ comporte deux anciennes entités cores ou plus, qui appartiennent à des clusters distincts, alors ces clusters sont fusionnés et le cluster résultant est étendu en ajoutant les entités atteignables par densité à partir de e. Dans la figure 4.6c, l'entité core e' possède deux anciennes entités cores dans son voisinage. Ainsi, les clusters C_1 et C_2 comportant respectivement e_1 et e_2 sont fusionnés et le voisinage de e' est ajouté au cluster résultant.
- Si une ancienne entité e devient core car une nouvelle entité non core est dans son voisinage, alors la nouvelle entité est absorbée par le cluster comportant e si cette dernière est affectée à un cluster. Dans le cas ou e était une entité bruit, un nouveau cluster est créé avec les entités atteignables par densité à partir de e. L'entité e_1 présentée dans la figure 4.6d crée un nouveau cluster comportant ses voisins et la nouvelle entité e'.



(a) Absorbtion de cluster

(b) Création de nouveau cluster

(c) Fusion de clusters

(d) Création de nouveau cluster

FIGURE 4.6 – Les différentes modifications possibles sur l'ensemble des clusters

Le nombre d'entités cores étant plus petit que le nombre total des entités dans un $chunk$, parcourir les entités cores au lieu de toutes les entités lors de la formation des clusters améliore l'efficacité du processus.

Durant cette phase, nous modifions les clusters dans le voisinage des nouvelles entités en suivant les règles définies ci-dessous :

- Si $e' \in \Delta_D$, e' est core et $\exists e \in neighborhood_\epsilon(e')$ et $e \in C$, et que e est core alors $C = C \cup e' \cup neighborhood_\epsilon(e')$.
- Si $e' \in D \cup \Delta_D$, si $\nexists e \in neighborhood_\epsilon(e')$ tel que e est core, alors créer un nouveau cluster $C' = e' \cup e$ tel que e atteignable par densité à partir de e'.
- Si $e' \in D \cup \Delta_D$, $\exists e_1, e_2, \cdots \in neighborhood_\epsilon(e')$ et $e_1 \in C_1$, $e_2 \in C_2, \cdots$ tel que $e_1, e_2, \cdots$ sont cores, alors créer un nouveau cluster $C' = C_1 \cup C_2 \cup \cdots \cup e' \cup neighborhood_\epsilon(e')$.
- Si $e' \in \Delta_D$, e' n'est pas core et $\exists e \in neighborhood_\epsilon(e')$ et $e \in C$, si e est core alors $C = C \cup e'$.

Ces règles sont appliquées en parallèle dans les différents $chunks$ en considérant les voisinages des entités calculés précédemment.

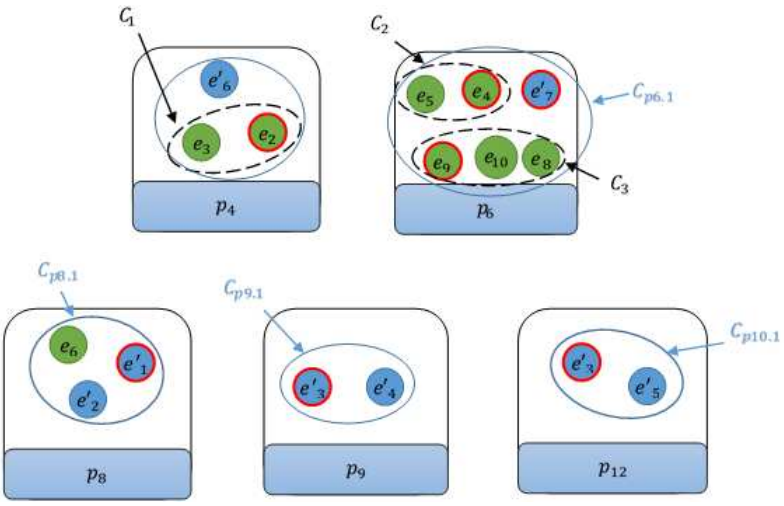


FIGURE 4.7 – Clustering des entités dans les $chunks$

Exemple 4.6.1 *Dans notre exemple, la modification des clusters dans les différents $chunks$ produit le résultat présenté dans la figure 4.7. Par exemple, l'entité e'_6 est absorbée par le cluster existant C_1 car elle n'est pas core, mais elle est voisine de l'entité core e_2. La nouvelle entité core e'_1 donne lieu à un nouveau cluster $C_{p_{8.1}}$ composé d'elle-même et de ses voisins. L'entité e'_7 est une entité core et son voisinage comporte deux anciennes entités cores e_4 et e_9. Par conséquent, les clusters C_2 et C_3 qui comportent respectivement e_4 et e_9 sont fusionnés pour former un nouveau cluster $C_{p_{6.1}}$.*

L'algorithme 10 décrit les modifications apportées sur l'ensemble des clusters dans chaque $chunk$. Il parcourt les entités cores dans les $chunks$ (ligne 3) : ces entités cores peuvent être de nouvelles entités ou d'anciennes

Algorithm 10 Calcul des clusters locaux

Input: CH : les chunks, $Cores$: les nouvelles entités cores

1: **for all** $[p] \in CH$ **do in parallel**
2: is-visited $= \emptyset$
3: **for all** $e \in Cores$ **do**
4: **if** $e \notin$ is-visited **then**
5: is-visited $=$ isVisited $\cup \{e\}$
6: **if** $e.cluster \neq null$ **then**
7: $C = e.cluster$
8: **else**
9: Create a new cluster $C = \{e\}$
10: **end if**
11: $C = C \cup neighborhood_\epsilon(e)$
12: **for all** $e' \in C \mid e' \in cores$ and $e' \notin$ is-visited **do**
13: **if** $e'.cluster = null$ **then**
14: $C = C \cup \{e'\} \cup neighborhood_\epsilon(e')$
15: **else**
16: $c' = e'.cluster$
17: $c = c \cup c'$
18: **end if**
19: **end for**
20: **end if**
21: local-clusters $=$ local-clusters $\cup C'$
22: **end for**
23: **end for**
24: **return** local-clusters

entités qui sont dans le voisinage d'une nouvelle entité. Ensuite, le cluster de l'entité core courante est identifié afin de l'étendre s'il existe (lignes 6-7). Si l'entité n'est affectée à aucun ancien cluster, un nouveau est alors créé (ligne 8). Le cluster est ensuite étendu en ajoutant les voisins de l'entité core (ligne 11). L'algorithme identifie ensuite parmi les entités ajoutées au cluster celles qui sont cores (ligne 12), et il ajoute leurs voisins au cluster si elles n'appartiennent pas déjà à un cluster (lignes 13-14). Si le cluster C créé comporte une entité core qui appartient également à un autre cluster C', alors C et C' sont fusionnés (lignes 16-17).

A la fin de cette phase, les clusters sont produits dans chaque $chunk$. Cependant, pour construire le résultat final, il est nécessaire d'identifier les clusters comportant des entités distribuées sur plusieurs $chunks$, et de fusionner les clusters locaux correspondant. En effet, la modification des clusters dans chaque $chunk$ est faite de façon indépendante, sans échange d'informations entre les nœuds de calcul et produit des clusters formés localement dans chaque $chunk$ et comportant des entités similaires dans un même $chunk$. Par conséquent, ces modifications doivent être consolidées afin de produire le nouveau schéma décrivant le jeu de données après sa mise à jour. Dans la section qui suit, nous allons décrire la formation des clusters finaux, qui correspondent aux classes du schéma, à partir des clusters locaux produits dans chaque $chunk$. Cette phase est exécutée sur un seul nœud de calcul et produit le résultat final du clustering.

4.6.2 Génération du nouveau schéma

Dans notre approche, le clustering est accéléré en parallélisant le processus sur les $chunks$. Cependant, ce calcul distribué des clusters peut produire des clusters qui s'étendent sur plusieurs $chunks$, et qui doivent être fusionnés en un même cluster final. En effet, des entités peuvent être atteignables par densité mais être distribuées dans des $chunks$ différents et donc affectées à différents clusters locaux.

Premièrement, les clusters créés à partir des nouvelles entités cores dans chaque $chunk$ de façon indépendante peuvent avoir des éléments atteignables par densité dans différents $chunks$ et qui appartiennent à d'autre clusters locaux. Par conséquent, ces derniers doivent appartenir à un même cluster. Ces clusters locaux partagent des cores communs et seront ainsi fusionnés. Les clusters des anciennes entités cores qui n'ont pas été fusionnés durant la modification des clusters ont le même identifiant sur toutes les $chunks$. Par conséquent, il n'est pas nécessaire de les fusionner.

Deuxièmement, les entités considérées durant le clustering sont les nouvelles entités et les anciennes entités qui sont dans le voisinage des nouvelles. Afin de produire le résultat final du clustering, les modifications effectuées sur les clusters doivent être propagées sur les anciennes entités qui n'ont pas été distribuées dans les $chunks$ et n'ont pas été considérées durant le clustering. Cette étape est exécutée en un seul processus et n'est pas parallélisée.

Dans cette section, nous visons d'abord à identifier les clusters distribués sur plusieurs $chunks$ et à fusionner les clusters locaux correspondants. Ensuite, nous présentons la réaffectation des anciennes entités dans le cas des clusters ayant fusionnés.

Le principe de l'identification et la fusion des clusters locaux a été introduit dans le chapitre précédent. Les clusters locaux forment un même cluster final s'ils partagent une entité core commune. En effet, une entité core appartenant à deux clusters distincts forme un lien entre les éléments de ces clusters rendant chaque entité de l'un des clusters atteignables à partir des entités de l'autre.

Par conséquent, les clusters dont les éléments sont répartis sur plusieurs $chunks$ sont identifiés en recherchant les clusters locaux produits dans chaque $chunk$, et qui partagent une nouvelle entité core commune. Cette opération concerne uniquement les clusters créés à partir de nouvelles entités cores. Durant cette étape, les nouveaux clusters locaux sont fusionnés afin de produire les clusters finaux.

Les entités bordures pouvant être affectées à plusieurs clusters durant le clustering sont insérées de façon aléatoire à un de ces clusters durant la fusion des clusters locaux.

Exemple 4.6.2 *Considérons le résultat du clustering présenté dans la figure 4.7. Les clusters $c_{p9.1}$ et $c_{p10.1}$ qui appartiennent à des $chunks$ distincts partagent une entité core e'_3 et doivent ainsi former un même cluster. Ils sont fusionnés en un cluster final C'_3 comme le montre la figure 4.8.*

Après avoir produit les clusters finaux qui représentent les nouvelles entités et leurs voisinages, ce résultat doit être intégré avec les anciens clusters pour produire le schéma final. Les anciens clusters à considérer durant cette

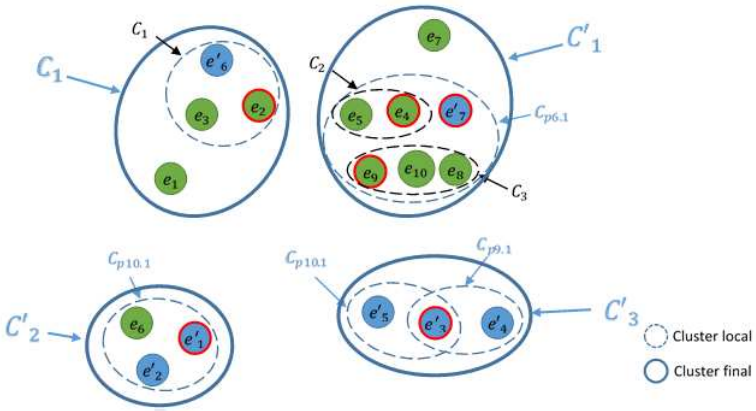


FIGURE 4.8 – Résultat du clustering des entités de notre exemple

opération sont ceux qui ont été fusionnés suite à l'insertion d'une nouvelle entité core qui est voisine d'une entité core appartenant au cluster. Lors de la fusion de deux clusters, les entités précédemment assignées à ces clusters doivent être ré-assignées au cluster résultant de leur fusion.

Si deux clusters C_i et C_j sont fusionnés en un nouveau cluster C', tous les éléments de ces deux clusters doivent être réassignés à C'. Cependant, durant la distribution des anciennes entités, toutes n'ont pas été affectées aux $chunks$, mais seulement celles partageant des propriétés communes avec de nouvelles entités. Toutes les entités qui appartiennent à des clusters ayant fusionné et qui n'ont pas été distribuées seront ré-assignées au cluster résultant de la fusion.

Exemple 4.6.3 *Dans la figure 4.7 présentant les clusters dans chaque $chunk$, les clusters C_2 et C_3 ont fusionné en un nouveau cluster $C_{p_6.1}$ car il existe une entité core e'_7 dans leur voisinage. Cependant, l'ancien cluster C_2 comporte une ancienne entité e_7 qui n'a été assignée à aucun $chunk$. Par conséquent, l'entité e_7 sera affectée à $C_{p_6.1}$ qui représentera le cluster C'_1 dans le résultat final.*

La figure 4.8 présente les clusters finaux obtenus à partir des clusters locaux de notre exemple.

Enfin, toutes les entités qui ne sont assignées à aucun cluster sont considérées comme du bruit.

Cette phase produit le résultat final du clustering, garantissant que l'algorithme incrémental conduit au même résultat que le DBSCAN séquentiel exécuté sur la totalité du jeu de données en un seul processus. En effet, les règles de construction des clusters assurent le regroupement de toutes les entités atteignables par densité à l'intérieur d'un même $chunk$. Ensuite, les clusters divisés sur plusieurs $chunks$ sont construits durant la consolidation des modifications effectuées en parallèle. Enfin, la propagation des mises à jour modifie l'assignation des entités qui n'ont pas été prises en compte durant le clustering.

L'algorithme 11 décrit la construction du résultat final du clustering. D'abord, il parcourt les nouvelles entités cores et si une entité core appartient à deux clusters ou plus, alors ces clusters sont fusionnés (lignes 2-5). Durant cette itération, les anciens clusters qui changent d'identifiant suite à une fusion sont sauvegardés (lignes 6-10).

Enfin, l'assignation des anciennes entités qui appartiennent aux clusters portant un nouvel identifiant est modifiée (lignes 12-17).

Algorithm 11 Restructuration des classes

Input: $local\text{-}clusters$: les clusters locaux, $Cores$: les entités cores, $oldClusters$: les anciennes entités qui ne sont pas dans des chunks et leur cluster

1: $clusters \leftarrow local\text{-}clusters$
2: $newIds : [oldId, newId] = \emptyset$
3: **for all** $e \in D \mid e \in Cores$ **do**
4: $\quad lc_e = \{C \in clusters \mid e \in C\}$
5: $\quad clusters = clusters \setminus lc_e \cup (\cup_{C \in lc_e} C)$
6: $\quad$ **for all** $c_i \in lc_e$ **do**
7: $\quad\quad$ **if** $c_i \in oldClusters$ **then**
8: $\quad\quad\quad newIds = newIds \cup (c_i, C)$
9: $\quad\quad$ **end if**
10: $\quad$ **end for**
11: **end for**
12: **for all** $e \in oldClusters$ **do**
13: $\quad$ **if** $e.cluster \in newIds$ **then**
14: $\quad\quad e.cluster = newIds.get(e.cluster)$
15: $\quad$ **end if**
16: **end for**
17: **return** clusters

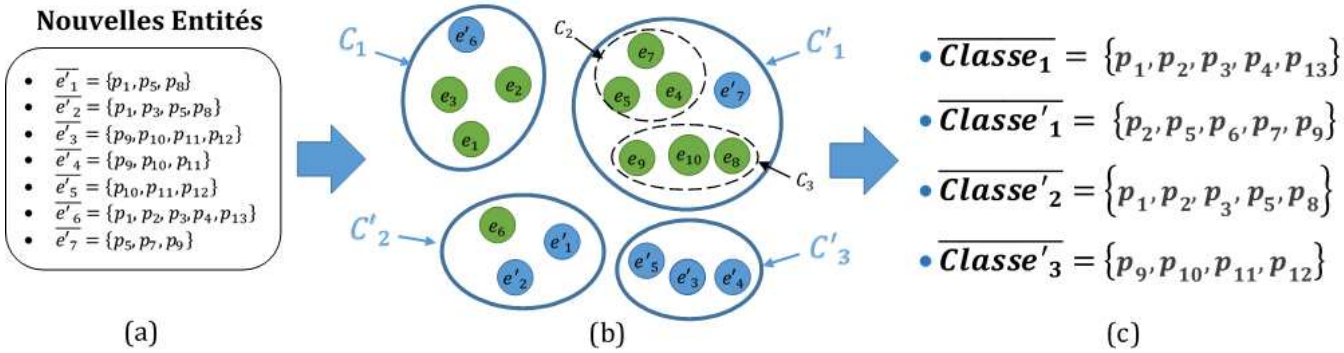


FIGURE 4.9 – La mise à jour du schéma après l'insertion des nouvelles entités

La figure 4.9 montre le résultat des modifications apportées aux classes introduites dans la figure 4.1 suite à l'insertion d'un ensemble de nouvelles entités. Par exemple, l'ensemble des propriétés décrivant $Classe_1$ a été modifié afin de représenter l'entité e'_6 qui fait partie du cluster C_1. Les classes $Classe_2$ et $Classe_3$ sont fusionnées en une nouvelle classe $Classe'_2$ représentant toutes les entités de ces deux classes. De plus, de nouvelles classes $Classe'_2$ et $Class'_3$ sont créées afin de représenter les nouveaux clusters générés.

4.7 Conclusion

Dans ce chapitre, nous nous sommes intéressés au problème de découverte incrémentale de schéma à partir des sources de données RDF lorsque de nouvelles entités sont ajoutées à cette dernière.

Notre avons pour cela proposé une approche de découverte de schéma, qui construit de façon incrémentale les classes décrivant les entités d'une source de données RDF qui évolue à travers le temps en ajoutant de nouvelles entités, ceci afin de garder le schéma cohérent avec les données lorsque ces dernières sont mises à jours. Notre approche est fondée sur un algorithme de clustering incrémental basé sur la densité qui forme les clusters regroupant les entités similaires en modifiant les clusters existants ou en créant des nouveaux clusters selon le voisinage des nouvelles entités ajoutées au jeu de données. L'approche garantit que l'ensemble des clusters résultant est le même que celui généré en utilisant l'algorithme DBSCAN sur la totalité du jeu de données en une seule exécution. Les clusters produits par notre approche représentent les classes du schéma qui décrivent les entités d'un jeu de données RDF.

Notre proposition a été implémentée en utilisant Spark, une technologie big data pour le calcul distribué, qui permet le passage à l'échelle de notre approche incrémentale de découverte de schéma et qui la rend adaptée à de grands jeux de données évolutifs. Nous montrerons dans le chapitre consacré aux expérimentations que l'extraction incrémentale de schéma à partir de jeu de données RDF en utilisant notre approche offre de meilleures performances que l'approche scalable de découverte de schéma appliquée sur la totalité du jeu de données.

La conception de notre approche permet son application aussi bien pour la mise à jour du schéma décrivant les entités d'une source de données RDF qui évolue fréquemment, que pour la découverte de schéma à partir de flux de données RDF. En effet, l'approche proposée peut être implémentée en utilisant une plateforme de gestion de flux de données comme Spark Streaming [2] ou Apache Flink [3], en considérant les nouvelles entités comme des flux de données. A chaque instant t, le schéma est extrait à partir du flux de données concerné et il est intégré au schéma découvert à l'instant $t-1$.

Tout comme le schéma produit par l'approche scalable de découverte de schéma, celui-ci peut être complété par les liens sémantiques et hiérarchiques entre les classes, et en annotant les classes. Une perspective à notre travail serait de faire évoluer de façon incrémentale les liens entre les classes du schéma, ainsi que l'annotation de ces derniers.

Nous avons supposé dans notre approche que le schéma initial avait été extrait en utilisant le principe de densité et en évaluant la similarité entre les entités en utilisant l'indice de *Jaccard*. Ces suppositions sont faites afin de garantir que le résultat de clustering est le même que celui produit par l'algorithme DBSCAN qui s'adapte à la nature des jeux de données RDF. Cependant, notre approche s'applique à n'importe quel schéma, produit en utilisant d'autres méthodes de découverte de schéma et d'autres indices de similarité. Cela permettrait d'intégrer dans tous les cas la représentation des nouvelles entités dans le schéma décrivant les données.

Dans nos travaux futurs, nous voulons étendre notre approche afin qu'elle prenne en compte d'autres évolutions des jeux de données RDF. Ces évolutions peuvent être des suppressions ou des modifications de structures d'en-

2. `https://spark.apache.org/docs/latest/streaming-programming-guide.html`
3. `https://flink.apache.org/`

tités existantes, c'est-à-dire l'ajout ou la suppression de propriétés décrivant une entité. La suppression d'entités à partir d'un jeu de données RDF peut impliquer la suppression ou la division de classes existantes. La modification structurelle d'une entité peut impacter les classes existantes, comme elle peut impliquer la création de nouvelles classes. Ces opérations nécessitent de recalculer le voisinage des entités modifiées d'où l'importance d'une approche adaptée aux grandes sources de données RDF.

Dans ce chapitre et le chapitre 3, nous avons adressé les problèmes de scalabilité et d'incrémentalité de la découverte de schéma à partir des sources de données RDF massives. Nous avons fait l'hypothèse que les déclarations schéma dans les jeux de données RDF sont absentes et que le schéma décrivant les entités est extrait en considérant uniquement la structure explicites des entités. Cependant, les entités d'une source de données RDF sont décrites par des propriétés explicites ainsi que des propriétés implicites qui peuvent être dérivées en utilisant les déclarations liées au schéma. Nous proposons dans le chapitre suivant une approche de découverte de schéma qui considère aussi bien les propriétés explicites qu'implicites. Notre approche exploite la sémantique contenue dans les jeux de données RDF dans le processus de découverte de schéma afin d'enrichir la description des entités et ainsi améliorer la qualité du schéma extrait.

Chapitre 5

Prise en compte des triplets implicites pour la découverte de schéma

5.1 Introduction

Dans les chapitres précédents, nous avons présenté des approches de découverte de schéma, scalables et incrémentales qui permettent l'extraction des classes décrivant les entités d'une source de données RDF massives, et qui évoluent à travers le temps. Ces approches s'appuient sur des algorithmes de clustering qui identifient les entités structurellement similaires qui peuvent représenter des instanciations d'une même classe. Le schéma est construit à partir de ces groupes d'entités similaires. Dans notre travail, nous avons fait l'hypothèse que les déclarations relatives au schéma sont absentes. C'est également le cas des approches existantes qui utilisent le clustering pour la découverte de schéma [21, 20, 65, 66]. Cette hypothèse permet à ces approches d'être appliquées sur toutes les sources de données RDF, même celles où le schéma est complètement absent. L'identification des entités similaires se fait en se basant sur les propriétés explicites décrivant les entités dans la source de données. Cependant, dans une source de données RDF, une entité est décrite par des propriétés explicites, définies lors de la création des données, et des propriétés implicites qui peuvent être dérivées à partir des propriétés explicites en utilisant des mécanismes de raisonnement. Ces propriétés implicites ne sont pas prises en compte par les approches de découverte de schéma existantes.

Exemple 5.1.1 *Par exemple, considérons le jeu de données RDF D présenté dans la figure 5.1 et les classes découvertes à partir de ce jeu de données en utilisant une approche qui s'appuie sur un algorithme de clustering par densité, et en fixant les paramètres ϵ à 0.6 et $minPts$ à 1. Nous remarquons dans ce schéma que certaines entités ne sont pas représentées par des classes. Par exemple, la classe qui représente les instances e_1 et e_2 est absente. Ceci est dû au fait que ces deux instances n'ont pas suffisamment de propriétés communes, et ainsi n'ont pas été regroupées dans un même cluster.*

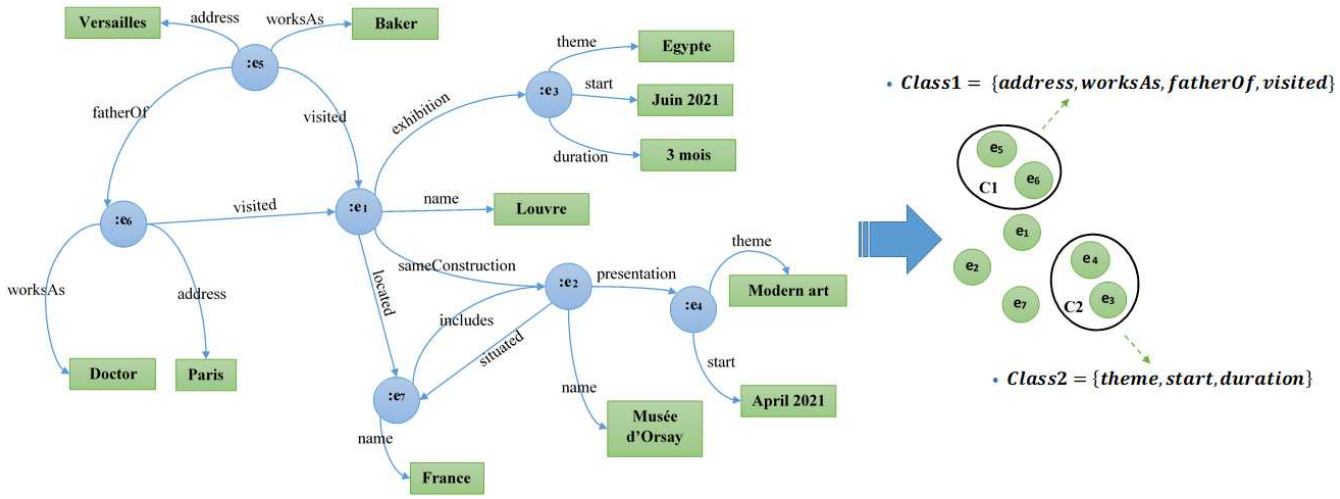


FIGURE 5.1 – Exemple de jeu de données RDF et les classes correspondantes

Supposons maintenant que nous disposons d'informations nous indiquant que les propriétés $presentation$ et $located$ sont équivalentes respectivement aux propriétés $exhibition$ et $situated$. En considérant ces informations, nous pouvons dériver que toutes les entités qui possèdent la propriété $presentation$ ont forcément la propriété $exhibition$ et vice-versa, de même pour les propriétés $located$ et $situated$. Ainsi, la description des entités e_1 et e_2 de notre exemple n'est plus réduite aux seules propriétés explicitement déclarées, mais comprend aussi les propriétés implicites qui peuvent en être déduites. En tenant compte des propriétés implicites décrivant ces e_1 et e_2, nous pouvons déduire par clustering que ces deux entités appartiennent bien à la même classe car elles partagent un nombre suffisant de propriétés.

Lorsque les déclarations sur le schéma sont partiellement disponibles, elles peuvent être utilisées afin de compléter la description des entités d'une source de données avec des propriétés implicites déduites par inférence.

Dans ce chapitre, nous proposons une première contribution vers une approche hybride de découverte de schéma qui est capable d'utiliser, lorsqu'elles existent les déclarations relatives au schéma afin de dériver des triplets implicites. Notre but est d'adapter notre approche de découverte de schéma afin qu'elle prenne en compte aussi bien les descriptions explicites qu'implicites des entités contenues dans une source de données, qui peuvent être déduites en exploitant soit les déclarations sur le schéma fournies avec les données, soit une source de connaissance externe si elle est disponible, comme une ontologie de domaine par exemple. En d'autres termes, l'approche doit considérer durant le clustering des entités leurs propriétés explicites et implicites. Dans notre contexte, où les sources de données sont massives, l'approche proposée doit être scalable.

La suite de chapitre est organisée comme suit. La section 5.2 définit la problématique abordée dans ce chapitre. Des préliminaires reprenant les travaux qui ont adressé le problème d'extraction des triplets implicites à partir des données RDF sont décrits dans la section 5.3. Dans la section 5.4, nous introduisons le principe général de notre approche hybride qui considère aussi bien les propriétés explicites qui décrivent une entité du jeu de données RDF,

que les propriétés implicites qui peuvent être déduites si des déclarations relatives au schéma sont disponibles, qu'elles soient complètes ou partielles. Nous décrivons dans la section 5.5 le processus d'extraction des triplets implicites à partir d'une source de données RDF. Dans la section 5.6, nous détaillons notre approche de découverte de schéma qui intègre la prise en compte des triplets implicites. Le chapitre est conclu par la section 5.7 qui présente le bilan de nos propositions, des améliorations possibles de notre approche ainsi que des perspectives de recherche.

5.2 Définition de la problématique

Dans ce chapitre, notre objectif est d'adapter notre approche de découverte de schéma afin qu'elle prenne compte des déclarations implicites. Notre approche se base sur le clustering des entités structurellement similaires et s'appuie sur les propriétés décrivant les entités afin de regrouper celles qui appartiennent à la même classe. Outre les propriétés explicites, la sémantique d'une source de données RDF inclut également d'autres propriétés, dites implicites, qui enrichissent la description des entités. Dans ce chapitre, nous abordons le problème de découverte de schéma à partir des sources de données RDF massives, en tenant compte des descriptions explicites et implicites des entités. Nous visons à exploiter la sémantique offerte par les sources de données RDF dans le but d'enrichir la description des entités en dérivant des propriétés implicites, et ainsi améliorer la qualité du schéma découvert.

Exemple 5.2.1 *Dans l'exemple 5.1.1, le fait de considérer que les propriétés $presentation$ et $located$ était équivalentes respectivement aux propriétés $exhibition$ et $situated$ a permis la découverte d'une nouvelle classe qui représente les entités e_1 et e_2, ceci en tenant compte de cette équivalence durant l'évaluation de leur similarité.*

Considérons un jeu de données RDF D comportant un ensemble d'entités $D = \{e_i | i \in [1, n]\}$, tel que n est le nombre d'entités dans D. Considérons également un ensemble de règles RG qui permettent de dériver à partir de D un ensemble de nouveaux triplets T. Dans ce chapitre, nous définissons une entité dans un jeu de données RDF comme étant décrite par l'ensemble des propriétés explicites et implicites. Nous introduisons dans ce chapitre une fonction dénotée par $\overline{\ }^{S}$ qui retourne l'ensemble de propriétés explicites et implicites qui décrivent une entité comme suit :

$$\begin{array}{lllll} \overline{\ }^{S} & : & \mathcal{R} \cup \mathcal{B} & \rightarrow & \mathcal{P} \\ & & e & \mapsto & \{p \in \mathcal{P} \mid \langle e, p, o \rangle \in D \vee \langle e, p, o \rangle \in T\} \\ & & & & \text{tel que } T \text{ dérivable à partir de } D \text{ en utilisant } RG \end{array}$$

Ainsi, nous visons à proposer une approche de découverte de schéma qui utilise les déclarations schéma Ds dans une source de données D afin de dériver les propriétés implicites décrivant une entité et les intégrer dans l'extraction des classes du schéma.

Une solution serait de dériver l'ensemble des triplets implicites à partir d'une source de données RDF en pré-traitement du processus de découverte de schéma. Cependant, une telle solution est coûteuse dans notre contexte, car les deux processus, la saturation comme la découverte de schéma, appliqués à des données massives représentent un traitement coûteux. De plus, les approches de saturation existantes appliquent toutes les règles de saturation, alors que dans notre travail, nous nous intéressons seulement aux règles qui dérivent des propriétés implicites. Ceci car notre approche de découverte de schéma se base sur le clustering des entités qui partagent des propriétés similaires. La similarité est évaluée en utilisant l'indice de *Jaccard* qui représente le rapport entre la cardinalité de l'intersection des ensembles de propriétés considérés et la cardinalité de leur union. Les règles qui produisent des nouvelles propriétés qui seront ajoutées à la description des entités ont un impact sur la similarité de ces dernières durant le clustering pour la découverte de schéma.

Par conséquent, notre objectif est d'adapter notre approche de découverte de schéma afin qu'elle tienne compte des propriétés implicites des entités en plus des propriétés explicites. Ceci dans le but d'améliorer la qualité du schéma produit tout en offrant les mêmes performances que notre approche scalable.

Une telle adaptation soulève les problèmes suivants :

- Concernant la prise en compte des propriétés implicites, quelles règles utiliser afin de les découvrir ? et quel ordre d'exécution de règles adopter afin de proposer un processus rapide qui produit l'ensemble exhaustif des propriétés implicites ?
- Comme notre principe de distribution des données est fondé sur les propriétés des entités, comment partitionner les données en tenant compte des propriétés explicites et implicites, sans faire un pré-traitement coûteux qui consisterait à générer la totalité des triplets implicites avant le clustering ?
- Lors de la distribution des données, le seuil de dissimilarité permet l'optimisation de l'approche en limitant la duplication des entités dans les $chunks$. Pour fonctionner, cette optimisation nécessite la définition d'une relation d'ordre sur les propriétés, sans avoir à extraire les propriétés implicites. Ainsi, comment définir à priori l'ordre des propriétés en tenant compte des propriétés explicites et implicites ?
- Comment assurer le clustering des entités dans chaque $chunk$ en comparant la similarité des entités par rapport à leurs propriétés explicites et implicites ?

Notre proposition exploite les déclarations relatives au schéma dans les sources de données RDF pour la découverte de schéma dans le but d'améliorer la qualité du schéma extrait.

5.3 Préliminaires

Cette section constitue un rappel des solutions apportées au problème de la génération des triplets implicites. Ce problème est connu dans la littérature sous la dénomination de "RDF Reasoning" [93, 59, 57, 44, 17], "RDF Triples Entailment" [99] ou encore "RDF Saturation" [32]. Dans le reste de ce chapitre, nous désignons l'opération

de l'extraction des triplets implicites par saturation. La saturation consiste à générer des triplets implicites à partir de ceux existants dans la source de données en appliquant un ensemble de règles [90, 13]. Cette opération implique l'ensemble des triplets instances qui décrivent les entités, et l'ensemble des triplets schéma (RDF, RDFS et OWL) qui décrivent les classes contenues dans la source de données. Nous définissons d'abord chacun de ces deux types de triplets.

Définition 5.3.1 *Un triplet instance est un triplet comportant un prédicat défini par l'utilisateur et qui ne fait pas partie du vocabulaire RDF, $RDFS$ ou OWL.*

Définition 5.3.2 *Un triplet schéma est un triplet comportant un prédicat primitif inclus dans le vocabulaire RDF, $RDFS$ ou OWL. Il décrit le schéma d'une source de données RDF en termes de classes, propriétés et contraintes.*

La saturation vise à inférer des triplets implicites à partir des triplets instances explicites en utilisant les triplets schéma. Les triplets schéma peuvent être fournis dans la source de données RDF, comme elles peuvent l'être par des ontologies externes à la source de données mais décrivant bien les concepts contenus dans la source. Une ontologie représente un vocabulaire spécifique à un sujet ou un domaine et définit les concepts utilisés pour décrire et représenter un champ d'expertise, permettant une intégration plus riche et garantissant l'interopérabilité des données.

Exemple 5.3.1 *Dans notre exemple, les connaissances qui indiquent que les propriétés $presentation$ et $located$ sont équivalentes respectivement aux propriétés $exhibition$ et $situated$ ne dépendent pas du contexte du jeu de données, et ont le même sens dans n'importe quel autre jeu de données.*

Afin de faire face au problème de saturation de grandes sources de données, des approches utilisant des technologies big data ont été proposées [17, 32, 44, 57, 59, 93]. Ces approches se basent sur les règles RDF définies dans le tableau 5.1 et les règles OWL définies dans le tableau 5.2.

En plus de l'implémentation qui se base sur des technologies big data afin d'assurer le passage à l'échelle, ces approches proposent des optimisations afin d'améliorer les performances de la saturation. Nous discutons dans ce qui suit les optimisations principales.

La saturation des triplets schéma avant la saturation des triplets instances. Le nombre de triplets schéma dans une source de données RDF est souvent petit comparé au nombre de triplets instances. Ainsi, les approches existantes saturent d'abord les triplets schéma. De plus, comme la taille du schéma est petite, les opérations de saturation qui ne peuvent pas être parallélisées sont effectuées en un traitement centralisé. En effet, certaines règles exigent plusieurs itérations afin de dériver tous les triplets implicites, comme par exemple la règle 5 du tableau 5.1 dont les conséquences d'une itération i sont utilisées comme prémisses de l'itération $i + 1$. À cet égard, les règles présentées dans les tableaux 5.1 et 5.2 sont catégorisées en deux parties : les règles concernant le schéma et

TABLE 5.1 – Règles de saturation liées aux déclarations RDFS

Règle Nº	Prémisse	Conséquence
1	$\langle s, p, o\rangle$	$\langle :n, rdf{:}type, rdfs{:}Literal\rangle$
2	$\langle p, rdfs{:}domain, x\rangle$, $\langle s, p, o\rangle$	$\langle s, rdf{:}type, x\rangle$
3	$\langle p, rdfs{:}range, x\rangle$, $\langle s, p, o\rangle$	$\langle o, rdf{:}type, x\rangle$
4	$\langle s, p, o\rangle$	$\langle s/o, rdf{:}type, rdfs{:}Resource\rangle$
5	$\langle p, rdfs{:}subPropertyOf, q\rangle$, $\langle q, rdfs{:}subPropertyOf, r\rangle$	$\langle p, rdfs{:}subPropertyOf, r\rangle$
6	$\langle p, rdf{:}type, rdf{:}Property\rangle$	$\langle p, rdfs{:}subPropetyOf, p\rangle$
7	$\langle s, p, o\rangle$, $\langle p, rdfs{:}subPropertyOf, q\rangle$	$\langle s, q, o\rangle$
8	$\langle s, rdf{:}type, rdfs{:}Class\rangle$	$\langle s, rdfs{:}subClassOf, rdfs{:}Resource\rangle$
9	$\langle s, rdf{:}type, x\rangle$, $\langle x, rdfs{:}subClassOf, y\rangle$	$\langle s, rdf{:}type, y\rangle$
10	$\langle s, rdf{:}type, rdfs{:}Class\rangle$	$\langle s, rdfs{:}subClassOf, s\rangle$
11	$\langle x, rdfs{:}subClassOf, y\rangle$, $\langle y, rdfs{:}subClassOf, z\rangle$	$\langle x, rdfs{:}subClassOf, z\rangle$
12	$\langle p, rdf{:}type, rdfs{:}ContainedMembershipProperty\rangle$	$\langle p, rdfs{:}subPropertyOf, rdfs{:}member\rangle$
13	$\langle o, rdf{:}type, rdfs{:}Datatype\rangle$	$\langle o, rdfs{:}subClassOf, rdfs{:}Literal\rangle$

les règles concernant les instances. Les règles concernant les instances désignent celles qui produisent d'autres triplets instances comme les règles 7 du tableau 5.1 et 3 du tableau 5.2. Par exemple, la règle 7 produit un triplet $\langle s, q, o\rangle$ qui comporte un prédicat défini par l'utilisateur. D'autre part, les règles concernant le schéma désignent les règles produisant des triplets schéma comme les règles 5 du tableau 5.1 et 13 du tableau 5.2. Par exemple, la règle 5 produit un triplet $\langle p, rdfs{:}subPropertyOf, r\rangle$ qui comporte un prédicat primitif faisant partie des déclarations $rdfs$.

Les dépendances entre les règles de saturation. La saturation des données est faite en appliquant les règles de façon itérative jusqu'à ce que tous les triplets implicites soient dérivés. Cependant, les performances de ce processus dépendent de l'ordre d'exécution des règles. Pour déterminer un ordre efficace d'exécution, les dépendances entre elles doivent être prises en considération. Une règle R_i précède une règle R_j si la conséquence de la règle R_i est utilisée comme prémisse de la règle R_j. Par exemple, dans le tableau 5.1, la règle 7 produit une déclaration qui sera utilisée comme prémisse des règles 2 et 3. En effet, la règle 7 produit des triplets instances qui sont utilisés avec les déclarations $rdfs{:}domain$ et $rdfs{:}range$ respectivement dans les règles 2 et 3 pour la production des déclarations $rdf{:}type$. Par conséquent, 7 doit être appliquée avant 2 et 3.

En tenant compte de ces optimisations, les approches existantes définissent un ordre d'exécution des règles pour saturer les triplets schéma en premier, et ensuite saturer les triplets instances. Cependant, la saturation des triplets est un processus itératif qui exige une comparaison de toutes les paires de triplets afin d'identifier ceux impliqués dans les règles de saturation. Afin d'accélérer ce traitement, certaines approches ont proposé de le paralléliser. Tout d'abord, la saturation des triplets schéma est faite en parallèle. Les triplets schéma sont saturés

TABLE 5.2 – Règles de saturation liées aux déclarations OWL

Règle Nº	Prémisse	Conséquence
1	$\langle p, rdf{:}type, owl{:}FunctionalProperty\rangle$ $\langle u, p, v\rangle$, $\langle w, p, u\rangle$	$\langle v, owl{:}sameAs, w\rangle$
2	$\langle p, rdf{:}type, owl{:}InverseFunctionalProperty\rangle$ $\langle v, p, u\rangle$, $\langle w, p, u\rangle$	$\langle v, owl{:}sameAs, w\rangle$
3	$\langle p, rdf{:}type, owl{:}SymmetricProperty\rangle$, $\langle v, p, u\rangle$	$\langle u, p, v\rangle$
4	$\langle p, rdf{:}type, owl{:}TransitiveProperty\rangle$, $\langle v, p, w\rangle$, $\langle w, p, v\rangle$	$\langle u, p, v\rangle$
5a	$\langle u, p, v\rangle$	$\langle u, owl{:}sameAs, u\rangle$
5b	$\langle u, p, v\rangle$	$\langle v, owl{:}sameAs, v\rangle$
6	$\langle v, owl{:}sameAs, w\rangle$	$\langle w, owl{:}sameAs, v\rangle$
7	$\langle v, owl{:}sameAs, w\rangle$, $\langle w, owl{:}sameAs, u\rangle$	$\langle v, owl{:}sameAs, u\rangle$
8a	$\langle p, owl{:}inverseOf, q\rangle$, $\langle v, p, w\rangle$	$\langle w, q, v\rangle$
8b	$\langle p, owl{:}inverseOf, q\rangle$, $\langle v, q, w\rangle$	$\langle w, p, v\rangle$
9	$\langle v, rdf{:}type, owl{:}Class\rangle$, $\langle owl{:}Class, owl{:}sameAs, w\rangle$	$\langle v, rdfs{:}subClassOf, w\rangle$
10	$\langle p, rdf{:}type, owl{:}Property\rangle$, $\langle p, owl{:}sameAs, q\rangle$	$\langle p, rdfs{:}subPropertyOf, q\rangle$
11	$\langle u, p, v\rangle$, $\langle u, owl{:}sameAs, x\rangle$, $\langle v, owl{:}sameAs, y\rangle$	$\langle x, p, y\rangle$
12a	$\langle v, owl{:}equivalentClass, w\rangle$	$\langle v, rdfs{:}subClassOf, w\rangle$
12b	$\langle v, owl{:}equivalentClass, w\rangle$	$\langle w, rdfs{:}subClassOf, v\rangle$
12c	$\langle v, rdfs{:}subClassOf, w\rangle$ $\langle w, rdfs{:}subClassOf, v\rangle$	$\langle v, rdfs{:}equivalentClass, w\rangle$
13a	$\langle v, owl{:}equivalentProperty, w\rangle$	$\langle v, rdfs{:}subProperty, w\rangle$
13b	$\langle v, owl{:}equivalentProperty, w\rangle$	$\langle w, rdfs{:}subProperty, v\rangle$
13c	$\langle v, rdfs{:}subProperty, w\rangle$ $\langle w, rdfs{:}subProperty, v\rangle$	$\langle v, owl{:}equivalentProperty, w\rangle$
14a	$\langle v, owl{:}hasValue, w\rangle$ $\langle v, owl{:}onProperty, p\rangle$, $\langle u, p, v\rangle$	$\langle u, rdf{:}type, v\rangle$
14b	$\langle v, owl{:}hasValue, w\rangle$ $\langle v, owl{:}onProperty, p\rangle$, $\langle u, rdf{:}type, v\rangle$	$\langle u, p, v\rangle$
15	$\langle v, owl{:}someValuesFrom, w\rangle$, $\langle u, p, x\rangle$ $\langle v, owl{:}onProperty, p\rangle$, $\langle x, rdf{:}type, w\rangle$	$\langle u, rdf{:}type, v\rangle$
16	$\langle v, owl{:}allValuesFrom, u\rangle$, $\langle w, p, x\rangle$ $\langle v, owl{:}onProperty, p\rangle$, $\langle w, rdf{:}type, v\rangle$	$\langle x, rdf{:}type, u\rangle$

d'abord car comme nous pouvons le remarquer sur les tableaux 5.1 et 5.2, les prémisses des règles concernant le schéma sont des triplets schéma alors que les prémisses des règles concernant les instances sont un triplet schéma et un triplet instance. Par conséquent, la saturation des triplets schéma peut se faire indépendamment des triplets instances, et fournit tous les triplets schéma nécessaire à la saturation des instances.

Les triplets schéma explicites et implicites sont ensuite utilisés pour la saturation des instances. Ainsi, les triplets instances sont distribués sur les différents nœuds de calcul, et les triplets schéma nécessaires à leur saturation sont diffusés sur ces nœuds. Enfin, les instances sont saturées en parallèle sur les différents nœuds offrant un processus de saturation performant et qui permet le traitement de grandes sources de données RDF.

5.4 Principe général de notre approche

Nous proposons dans ce chapitre l'adaptation de notre approche de découverte de schéma pour qu'elle considère les propriétés explicites et implicites décrivant les entités de la source. Pour cela notre approche exploite les triplets schéma disponibles dans une source de données RDF et les règles de saturation afin de dériver de nouveaux triplets. Comme notre approche de découverte de schéma se base sur la structure des entités (les propriétés décrivant une entité), nous nous intéressons dans la saturation des entités à l'inférence de nouveaux triplets comportant de nouvelles propriétés.

La saturation des entités offre une description plus complète de ces dernières et améliore ainsi la qualité du schéma produit. Dans notre approche, nous nous intéressons uniquement à dériver les propriétés implicites décrivant les entités. Par conséquent, nous ne considérons pas toutes les règles de saturation, mais seulement celles qui produisent de nouvelles propriétés.

Définition 5.4.1 *Considérons un jeu de données $D = e_i | i = [1, n]$ et l'ensemble RG qui représente les règles $RDFS$ et OWL. Dans notre approche, nous définissons un ensemble de règles $RG' \subseteq RG$ tel que $\forall r_i \in RG'$ il existe une chaîne $r_i, r_{i+1}, \ldots, r_{i+j}$ où chaque règle r_{i+1} est dérivée à partir de r_i et que r_{i+j} permet la dérivation de triplets $\langle e, p, o \rangle$ tel que $p \notin \overline{e}$.*

Les règles de saturation qui nous intéressent sont présentées dans le tableau 5.3. Nous désignons par p, q et r des propriétés, par s et o respectivement le sujet et l'objet d'un triplet RDF.

Nous avons identifié ces règles comme suit. Nous nous sommes intéressés tout d'abord aux règles de niveau instance qui s'appliquent aux propriétés des triplets instances et qui produisent des triplets instances comportant des nouvelles propriétés. Nous avons ensuite regardé les règles de niveau schéma qui produisent des triplets utilisés dans les règles de niveau instance préalablement identifiées.

Par conséquent, dans notre approche, nous nous intéressons aux déclarations schéma utilisées dans les règles identifiées dans le tableau 5.3 et qui seront utilisées pour l'extraction de nouvelles propriétés décrivant les entités :

TABLE 5.3 – Règles de saturation

Règle N°	type	Prémisse	Résultat	Niveau
1	rdfs	$\langle p, rdfs{:}subPropertyOf, q\rangle$ et $\langle q, rdfs{:}subPropertyOf, r\rangle$	$\langle p, rdfs{:}subPropertyOf, r\rangle$	Schéma
2	rdfs	$\langle p, rdfs{:}subPropertyOf, q\rangle$ et $\langle s, p, o\rangle$	$\langle s, q, o\rangle$	Instance
3	owl	$\langle s, p, o\rangle$ et $\langle p, rdf{:}type, owl{:}symmetricProperty\rangle$	$\langle o, p, s\rangle$	Instance
4	owl	$\langle p, owl{:}inverseOf, q\rangle$ et $\langle s, q, o\rangle$	$\langle o, q, s\rangle$	Instance
5	owl	$\langle p, owl{:}inverseOf, q\rangle$ et $\langle s, p, o\rangle$	$\langle o, p, s\rangle$	Instance
6	owl	$\langle p, owl{:}equivalentProperty, q\rangle$	$\langle p, rdfs{:}subProperty, q\rangle$	Schéma
7	owl	$\langle p, owl{:}equivalentProperty, q\rangle$	$\langle q, rdfs{:}subProperty, p\rangle$	Schéma
8	owl	$\langle p, rdfs{:}subProperty, q\rangle$ et $\langle q, rdfs{:}subProperty, p\rangle$	$\langle p, owl{:}equivalentProperty, q\rangle$	Schéma
9	owl	$\langle p, rdf{:}type, owl{:}Property\rangle$ et $\langle p, owl{:}sameAs, q\rangle$	$\langle p, rdfs{:}subPropertyOf, q\rangle$	Schéma
10	owl	$\langle v, owl{:}sameAs, w\rangle$	$\langle w, owl{:}sameAs, v\rangle$	Schéma
11	owl	$\langle p, rdf{:}type, owl{:}FunctionalProperty\rangle$ $\langle u, p, v\rangle$, $\langle w, p, u\rangle$	$\langle v, owl{:}sameAs, w\rangle$	Schéma
12	owl	$\langle p, rdf{:}type, owl{:}InverseFunctionalProperty\rangle$ $\langle v, p, u\rangle$, $\langle w, p, u\rangle$	$\langle v, owl{:}sameAs, w\rangle$	Schéma

- $rdfs{:}subProperty$: définit une propriété comme une spécialisation (sous-propriété) d'une autre propriété.
- $owl{:}equivalentProperty$: établit que deux propriétés possèdent le même objet.
- $owl{:}inverseOf$: définit une relation d'inversion entre les propriétés.
- $owl{:}symmetricProperty$: définit une relation de symétrie entre deux ressources RDF. Ainsi, le sujet d'un triplet RDF $\langle e_1, p, e_2\rangle$ représente l'objet d'un autre triplet $\langle e_2, p, e_1\rangle$.
- $owl{:}sameAs$: indique que deux propriétés sont similaires.
- $owl{:}Property$: associée à la propriété $rdf{:}type$, elle nous informe que le sujet du triplet représente une propriété.
- $owl{:}FunctionalProperty$: une propriété identifiée comme $FunctionalProperty$ signifie qu'elle possède un objet unique.
- $owl{:}InverseFunctionalProperty$: une propriété identifiée comme $InverseFunctionalProperty$ signifie qu'elle possède un sujet unique.

Seules ces déclarations schéma et les règles identifiées dans le tableau 5.3 sont considérées dans notre approche. Les déclarations schéma qui ne sont pas impliquées dans les règles qui dérivent des triplets instances comportant des nouvelles propriétés décrivant les entités ne sont pas prises en compte. Notre proposition est composée des étapes illustrées dans la figure 5.2 :

Tout d'abord, nous saturons les déclarations qui portent sur le schéma, et ce en parallèle. La saturation du schéma se base sur les règles de raisonnement qui s'appliquent aux triplets décrivant le schéma. Les triplets schéma sont saturés en premier car leurs saturation ne dépendent pas des triplets instances. Comme nous pouvons

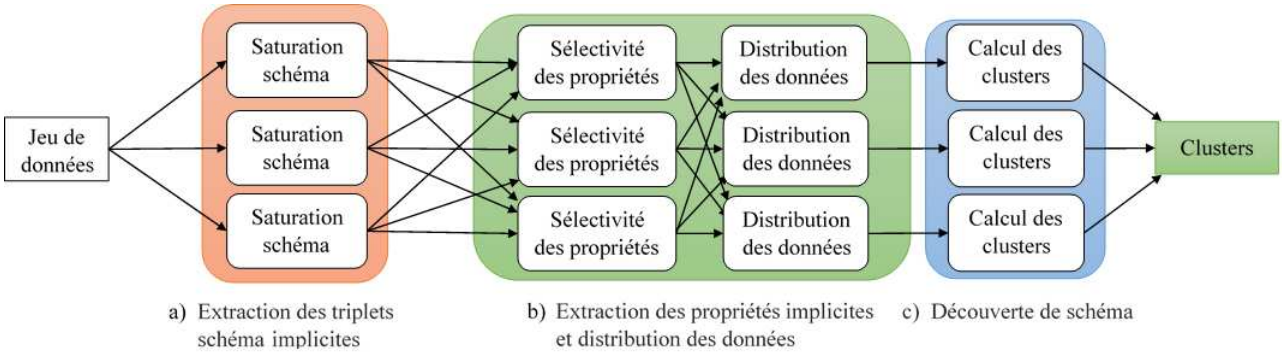


FIGURE 5.2 – Vue globale de l'approche de découverte de schéma avec prise en compte des propriétés implicites

le remarquer sur les tableaux 5.1 et 5.2, les prémisses des règles concernant le schéma sont uniquement des triplets schéma : les règles de saturation du schéma ne nécessitent pas de triplets instances, contrairement à la saturation des entités qui exige la connaissance de touts les triplets schéma. Par conséquent, les triplets schéma doivent être saturés avant la saturation des triplets instances. De plus, cette opération peut se faire indépendamment des triplets instances, et fournit tous les triplets schéma nécessaires à la saturation des instances. Enfin, le nombre de triplets schéma étant petit, il est possible de les diffuser sur les différents nœuds de calcul afin de paralléliser la saturation des triplets instances.

Nous distribuons ensuite les entités sur les différents nœuds de calcul afin d'assurer un traitement parallèle et rapide. La distribution des entités se fait suivant le principe introduit dans le chapitre 3, qui consiste à distribuer les entités du jeu de données par rapport aux propriétés qui les décrivent. Pour cela, nous calculons à priori la sélectivité des propriétés, en tenant compte des propriétés explicites et implicites. Ensuite, les entités sont saturées en parallèle et distribuées dans des $chunks$ de données en respectant les propriétés explicites et implicites.

Enfin, le schéma est extrait en calculant le clustering des entités en parallèle dans chaque $chunk$. Le regroupement en clusters des entités est fait en parallèle suivant le principe défini dans le chapitre 3. Tout d'abord, le voisinage de chaque entité est calculé et les entités cores sont identifiées. Une entité est décrite par ses propriétés explicites et implicites. Ensuite, les clusters locaux sont formés dans chaque $chunk$. Ces derniers sont ensuite fusionnés afin de produire le résultat du clustering final. Les clusters finaux représentent les classes du schéma.

5.5 Saturation des propriétés des entités

Nous distinguons deux types de triplets dans le processus de saturation. Les triplets instances qui représentent les entités du jeu de données, et les triplets schéma qui représentent des contraintes RDF, RDFS ou OWL décrivant le schéma des données.

Durant la saturation, toutes les règles qui produisent de nouveaux triplets sont appliquées itérativement jusqu'à ne plus provoquer de nouvelles modifications. Ceci nécessite un nombre élevé de comparaison entre les

sujets ou objets des triplets afin d'identifier ceux qui portent sur la même propriété, et ainsi implique un traitement coûteux. Dans un processus parallèle, les règles sont appliquées en faisant des opérations de jointure sur les triplets qui sont lus depuis différents nœuds de calcul ce qui provoque une surcharge importante liée aux échanges de données. Par exemple, l'application de la règle $\langle p, rdfs{:}subPropertyOf, q\rangle$, $\langle q, rdfs{:}subPropertyOf, r\rangle$ $\Rightarrow \langle p, rdfs{:}subPropertyOf, r\rangle$ nécessite l'identification des triplets $\langle s, p, o\rangle$, $\langle s', p, o'\rangle$ où $p = rdfs{:}subpropertyOf$ et de faire la jointure si $o = s'$, afin de produire un nouveau triplet $\langle s, p, o'\rangle$. Itérativement, les triplets résultants sont ensuite utilisés comme prémisses pour produire de nouveaux triplets.

Cependant, nous pouvons remarquer que les règles ont comme prémisse au plus un seul triplet instance. Ainsi, lors de la saturation, il n'y a jamais de jointures entre les triplets instance. Pour saturer une entité, il est nécessaire de disposer des triplets instance décrivant l'entité et des triplets schéma. De plus, des expériences ont montré sur des cas réels que le nombre de triplets schéma est petit et reste presque constant [52, 105, 70, 54]. Par conséquent, vu sa petite taille, l'ensemble des triplets schéma peut être saturé en premier et diffusé sur les différents nœuds de calcul afin de permettre une saturation parallèle de l'ensemble des triplets instances, qui est en général de grande taille et nécessite un traitement performant. De plus, la saturation des instances se fera en une seule itération sans aucune communication entre les nœuds de calcul.

Dans notre approche, nous saturons d'abord le schéma afin de produire tous les triplets schéma nécessaires à la saturation des instances. Les règles de saturation du schéma sont citées dans le tableau 5.3 et correspondent aux lignes pour lesquelles la colonne *niveau* indique *schéma*.

En plus des règles de saturation schéma existantes adoptées dans notre approche, nous introduisons une nouvelle règle qui permettra de produire tous les triplets schéma liés à la déclaration $owl{:}inverseOf$ dans le but de saturer les triplets instances en une seule itération. Dans le cas où le triplet instance $\langle e_1, p_1, e_2\rangle$ et les triplets schéma $\langle p_1, inverseOf, p_2\rangle$ et $\langle p_2, inverseOf, p_3\rangle$ sont contenus dans la source, nous obtiendrons en deux itérations les triplets $\langle e_2, p_2, e_1\rangle$ et $\langle e_1, p_3, e_2\rangle$. Pour éviter d'itérer plusieurs fois sur les règles durant la saturation des entités, en complément des règles utiles présentées dans le tableau 5.3, nous introduisons une nouvelle règle : $p_1\, inverseOf\, p_2$ et $p_2\, inverseOf\, p_3 \rightarrow p_1\, sameRole\, p_3$ qui implique qu'une entité ayant la propriété p_1 aura aussi la propriété p_3. Sans cette règle, cette implication sera obtenue en deux itérations en utilisant les déclarations $inverseOf$. Nous attribuons à cette règle le numéro 13, et elle représente une règle de niveau schéma.

Afin d'offrir une saturation efficace des triplets schéma, il est nécessaire de limiter le nombre d'itérations effectuées. Pour cela, nous définissons un ordre d'exécution des règles en fonction des dépendances existantes entre elles. Deux règles R_1 et R_2 sont dépendantes si le résultat de la règle R_1 est utilisé comme prémisse à la règle R_2. Dans ce cas, la règle R_1 doit être exécutée avant la règle R_2. Dans notre approche, nous définissons l'ordre suivant des règles de saturation :

- Les règles 11 et 12 ne considèrent pas d'autres triplets schéma et s'appliquent sans dépendance avec d'autres règles. Cependant, elles produisent des déclarations $owl{:}sameAs$ qui sont utilisées dans la règle 9 et doivent ainsi s'exécuter avant cette règle.
- La règle 10 utilise les triplets ayant la propriété $sameAs$ produits par les règles 11 et 12, et produit de nouveau triplets ayant la propriété $sameAs$ qui seront utilisés par la règle 9. Par conséquent, la règle 10 doit s'exécuter après les règles 11 et 12 et avant la règle 9.
- Nous remarquons que les règles 6, 7 et 9 du tableau 5.3 produisent des triplets ayant la propriété $subPropertyOf$. Par conséquent, ces règles doivent s'exécuter avant la règle 1 qui prend en prémisse des déclarations $subPropertyOf$. La règle 1 s'exécute en plusieurs itérations car les déclarations $subPropertyOf$ qu'elle produit peuvent être utilisées comme prémisses afin de dériver d'autres tripets.
- La règle 1 produit des triplets ayant la propriété $subProperty$, ainsi elle doit s'exécuter avant la règle 8 qui utilise cette propriété pour produire des triplets comportant des propriétés $equivalentProperty$.
- La règle 8 s'exécute après toutes les autres car elle représente une réduction et remplace deux triplets ayant la propriété $subProperty$ par un triplet $equivalentProperty$. Bien que cette règle produise des triplets $equivalentProperty$, les règles 6 et 7 ne sont pas ré-exécutées car aucun nouveau triplet ne sera produit, i.e., ces deux règles reproduiront les deux prémisses qui ont créé le triplet $equivalentProperty$.
- La nouvelle règle 13 est indépendante et peut être exécutée à une position quelconque par rapport aux autres règles.

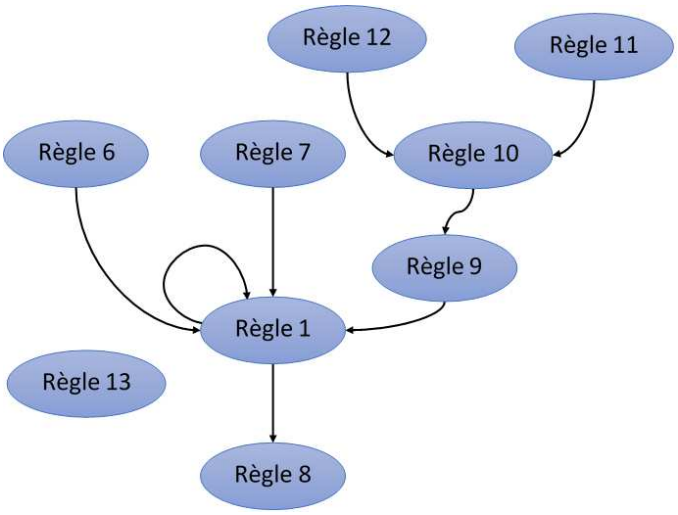


FIGURE 5.3 – Ordre d'exécution des règles de saturation

En considérant ces dépendances, nous définissons l'ordre des règles de saturation du schéma présenté dans la figure 5.3. La saturation du schéma est faite en exécutant ces règles dans l'ordre défini. Chaque règle est prise en charge par un processus parallèle séparé. Durant l'application de chaque règle, nous distribuons les triplets par rapport aux propriétés concernées par la déclaration, afin d'assurer un traitement parallèle. Ainsi, nous regroupons

dans un même processus tous les triplets, pour une déclaration donnée, qui concernent la même propriété. Cette distribution permet un processus de saturation parallèle. Ensuite, dans chaque partition, les règles sont appliquées afin de dériver de nouveaux triplets.

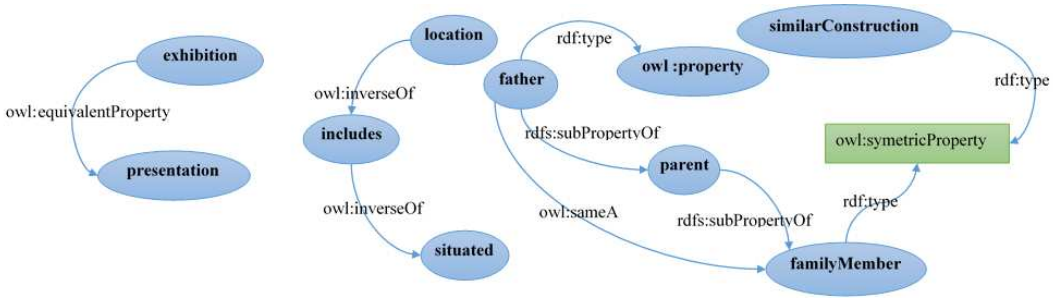


FIGURE 5.4 – Exemple de déclarations schéma dans une source de données RDF

Exemple 5.5.1 *Considérons que le jeu de données de la figure 5.1 comporte en plus des entités, les déclarations schéma présentées dans la figure 5.4.*

Dans cet exemple, afin de dériver les déclarations $subPropertyOf$ *à partir des déclarations schéma, nous distribuons tout d'abord les données pour assurer un traitement parallèle et rapide. Comme illustré dans la figure 5.5.b, nous assignons au même processus tous les triplets comportant des déclarations qui s'appliquent sur la même ressource. Par exemple, le processus* $parent$ *comporte tous les triplets qui peuvent dériver des triplets implicites comportant cette propriété.*

Ensuite, les triplets implicites sont dérivés dans chaque processus en parallèle. L'extraction des triplets implicites est faite sur les partitions comportant au moins deux triplets. Dans notre exemple, le triplet implicite $\langle father, rdfs{:}subpropertyOf, familyMember\rangle$ *est dérivé à partir des deux triplets de la partition* $parent$ *(figure 5.5.c).*

Enfin, l'opération est répétée en considérant les nouvelles déclarations $subProperty$*, jusqu'à ce que tous les triplets schéma implicites soient découverts. Dans notre exemple, le processus se répète en considérant les triplets initiaux plus le triplet dérivé dans la partition* $parent$*.*

Les processus gérant les règles $6, 7, 8, 9$ et 13 s'exécutent en une seule itération contrairement à la règle 1 qui nécessite une exécution itérative tant que de nouveaux triplets sont dérivés.

Exemple 5.5.2 *La saturation des déclarations schéma de notre exemple produit les triplets implicites suivant :*

$\langle location, sameRoleAs, situated\rangle$

$\langle father, rdfs{:}subProperty, familyMember\rangle$

Ces déclarations sont ajoutées à l'ensemble de triplets Ds *et constituent les déclarations du schéma du jeu de données* D*.*

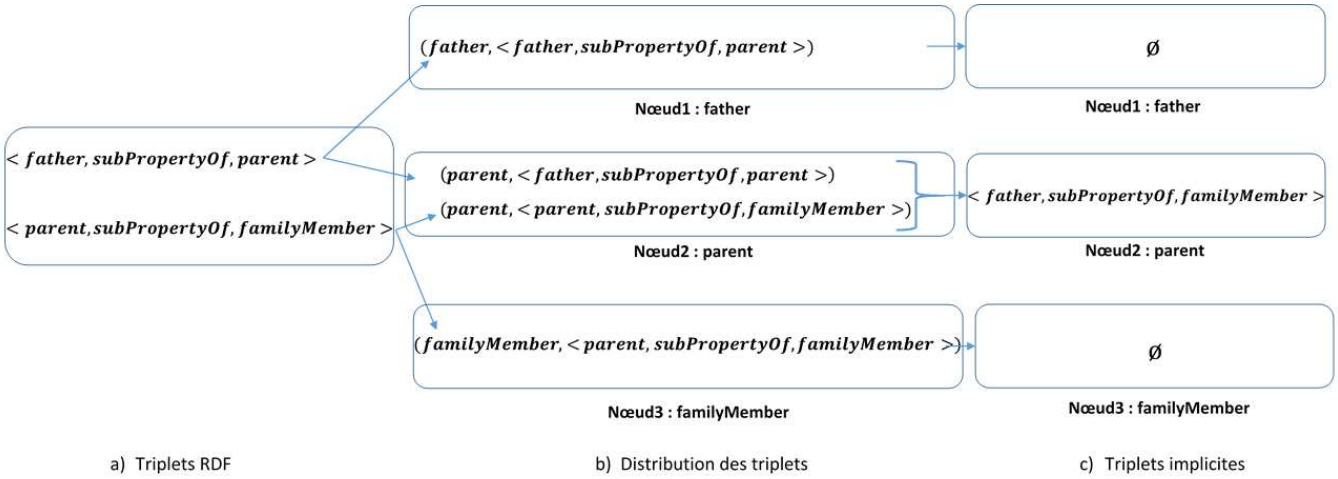


FIGURE 5.5 – Distribution des déclarations schéma

L'algorithme 12 décrit la phase de saturation du schéma, qui s'exécute en parallèle sur les triplets schéma. Ces processus s'exécutent en un pipeline de processus distribués.

Les triplets sont distribués de façon à regrouper tous ceux qui portent sur la même propriété dans la même partition (lignes 1-11). Durant cette étape, nous appliquons les règles 6, 7 et 9 qui consistent à produire des déclarations $subProperty$ à partir de déclarations $equivalentProperty$ (lignes 2-6). Les partitions produites en parallèle sont fusionnées pour que toutes les déclarations d'une propriété p_i soient traitées dans une même partition.

Ensuite, en parallèle sur chaque partition, nous appliquons la règle de saturation 1 afin de dériver les déclarations $subProperty$. Ainsi, pour chaque triplet ayant comme objet la propriété p_i, l'algorithme vérifie s'il existe un triplet où la même propriété p_i est un sujet, afin de dériver transitivement un nouveau triplet (lignes 13-22). Ce processus s'exécute itérativement tant que de nouveaux triplets sont dérivés.

Enfin, nous appliquons les règles 8 et 9 en un seul processus, car elles sont indépendantes l'une de l'autre (ligne 23). Ainsi, pour chaque triplet $\langle p_1, subProperty, p_2\rangle$, l'algorithme vérifie d'une part l'existence d'un $\langle p_2, subProperty, p_1\rangle$ afin d'appliquer la règle 8 et de dériver des déclarations $equivalentProperty$ (lignes 24-27). D'autre part, il vérifie l'existence d'un triplet $\langle p_2, inverseof, p_3\rangle$ afin d'appliquer la règle 13 et de dériver des déclarations $sameRole$ (lignes 28-30).

Les triplets de sortie incluent tous les triplets en entrée plus tous les triplets dérivés. Tous les triplets produits par ces processus sont fusionnés avec les triplets schéma initiaux pour former l'ensemble des déclarations schéma finales.

Dans notre approche de découverte de schéma, nous considérons les propriétés explicites et implicites dans l'évaluation de la similarité des entités et pour former le schéma. Les triplets schéma explicites dans la source de données, plus les triplets implicites dérivés par le processus de saturation du schéma sont utilisés par la suite dans la saturation des entités afin de dériver de nouvelles propriétés implicites. La taille des triplets schéma étant petite

Algorithm 12 La saturation du schéma

Input: ds : les triplets schéma

1: **for all** $\langle p_1, predicate, p_2 \rangle \in ds$ **do in parallel**
2: **if** $(predicate = equivalentProperty)\, OR\, (predicate = sameAs\, AND\, predicate.isAProperty())$ **then**
3: $partitions = partitions \cup (p_1, \langle p_1, subProperty, p_2 \rangle)$
4: $partitions = partitions \cup (p_2, \langle p_1, subProperty, p_2 \rangle)$
5: $partitions = partitions \cup (p_1, \langle p_2, subProperty, p_1 \rangle)$
6: $partitions = partitions \cup (p_2, \langle p_2, subProperty, p_1 \rangle)$
7: **else**
8: $partitions = partitions \cup (p_1, \langle p_1, predicate, p_2 \rangle)$
9: $partitions = partitions \cup (p_2, \langle p_1, predicate, p_2 \rangle)$
10: **end if**
11: **end for**
12: Fusionner les partitions locales par propriété
13: {this is a comment}
14: **while** $loop = true$ **do**
15: $loop = flase$
16: **for all** $\langle p_1, subProperty, p_2 \rangle \in part$ **do**
17: **if** $\exists \langle p_2, subProperty, p_3 \rangle \in part$ **then**
18: $part = part \cup \langle p_1, subProperty, p_3 \rangle$
19: $loop = true$
20: **end if**
21: **end for**
22: Fusionner l'ensemble des déclarations $subProperty$
23: **end while**
24: **for all** $\langle p_1, subProperty, p_2 \rangle\, AND\, \langle p_1, inverseOf, p_2 \rangle \in part$ **do**
25: **if** $\exists \langle p_2, subProperty, p_1 \rangle \in part$ **then**
26: $part = part \cup \langle p_1, equivalentProperty, p_2 \rangle$
27: $part = part \setminus \langle p_1, subProperty, p_2 \rangle$
28: $part = part \setminus \langle p_2, subProperty, p_1 \rangle$
29: **else if** $\exists \langle p_2, inverseOf, p_3 \rangle \in part$ **then**
30: $part = part \cup \langle p_1, sameRole, p_3 \rangle$
31: **end if**
32: **end for**
33: Fusionner l'ensemble des triplets produit sur chaque partition
34: **return** part

même après saturation, ces derniers sont alors distribués sur les différents nœuds de calcul afin de permettre une saturation rapide des entités, qui porte sur un grand nombre de triplets et nécessite un traitement parallèle scalable.

Nous décrivons dans la section suivante comment les propriétés décrivant les entités sont prises en compte dans notre approche afin de former des clusters d'entités structurellement similaires.

5.6 Prise en compte des triplets implicites pendant la découverte de schéma

La saturation des triplets schéma produit tous les triplets schéma nécessaires à la saturation des entités ; dans notre approche, la saturation des entités consiste à dériver de nouvelles propriétés décrivant les entités.

Dans notre contexte big data où les jeux de données sont massifs, nous distribuons les entités à travers différents nœuds de calcul afin d'offrir un algorithme efficace et rapide. Dans notre approche de découverte de schéma, les

entités sont divisées en $chunks$ selon les propriétés qui les décrivent. Ce principe a été introduit dans le chapitre 3.

Cependant, dans ce chapitre, nous tenons compte à la fois des propriétés explicites décrivant les entités et des propriétés implicites déduites à partir des déclarations relatives au schéma présentes dans le jeu de données. Ainsi, lors du partitionnement des données, nous devons tenir compte des propriétés explicites et de celles qui seront dérivées par la saturation.

Dans cette section, nous détaillons notre adaptation de notre méthode de distribution des données afin de tenir compte des propriétés implicites. Nous avons proposé dans le chapitre 3 une optimisation de la distribution des entités qui se base sur une relation d'ordre entre les propriétés Ici, nous présentons la définition de cette relation d'ordre en tenant compte non seulement des propriétés explicites mais également des propriétés implicites. Dans un deuxième temps, nous montrons comment les données sont distribuées en tenant compte des deux types de propriétés.

5.6.1 Calcul de la sélectivité des propriétés

Notre méthode de distribution se base sur les propriétés décrivant les entités pour diviser le jeu de données initial. Chaque $chunk$ créé comporte des entités partageant des propriétés communes et susceptibles d'être similaires. La distribution est optimisée en utilisant le *seuil de dissimilarité* qui limite le nombre de propriétés à considérer lors de la distribution de chaque entité et réduit la taille des $chunks$ produits. Pour choisir les propriétés à considérer pour chaque entité, nous définissons un ordre sur les propriétés. Dans notre approche, les propriétés sont ordonnées selon leur sélectivité.

Ainsi, pour pouvoir prendre en compte à la fois les propriétés explicites et implicites dans notre approche d'extraction de schéma, nous devons déterminer à priori, la sélectivité des propriétés et ainsi définir une relation d'ordre entre ces dernières.

Le calcul de sélectivité des propriétés est composé de trois étapes principales. Tout d'abord, les propriétés décrivant les entités sont extraites à partir des triplets du dataset. Ensuite, la sélectivité initiale de chaque propriété est calculée en considérant la co-occurrence de chaque propriété individuellement, ainsi que la co-occurrence des propriétés liées par des déclarations schéma. Cette sélectivité initiale ne prend pas en compte les triplets implicites. Enfin, nous calculons la sélectivité finale des propriétés explicites et implicites suivant les relations entre les propriétés que nous trouvons dans les déclarations sur le schéma.

Extraction des entités

Cette étape consiste à extraire à partir d'un jeu de données D, la description des entités à partir des triplets du jeu de données. La description d'une entité e consiste en l'ensemble des propriétés décrivant e noté $\overline{e}$. Durant cette étape, nous considérons deux types de propriétés : les propriétés sortantes, et les propriétés entrantes annotées par une flèche.

$$si\,(e_1, p_1, e_2) \in D \text{ alors } p_1 \in \overline{p_1}$$

$$si\,(e_2, p_1, e_1) \in D \text{ alors } \overleftarrow{p_1} \in \overline{p_1}$$

Les propriétés sortantes sont utilisées dans la saturation des entités ainsi que dans le processus de découverte de schéma, contrairement aux propriétés entrantes qui ne sont utilisées que pour la saturation des entités.

Exemple 5.6.1 *En considérant les propriétés entrantes et sortantes, les entités extraites à partir du jeu de données présenté dans la figure 5.1 sont décrites comme suit :*

$\overline{e_1} = \{name, exhibition, located, sameConstruction, \overleftarrow{visited}\}$,

$\overline{e_2} = \{name, presentation, situated, \overleftarrow{sameConstruction}, \overleftarrow{includes}\}$,

$\overline{e_3} = \{theme, duration, start, \overleftarrow{exhibition}\}$,

$\overline{e_4} = \{theme, start, \overleftarrow{presentation}\}$,

$\overline{e_5} = \{email, address, worksAs, fatherOf, visited\}$,

$\overline{e_6} = \{email, address, occupation, visited, \overleftarrow{fatherOf}\}$,

$\overline{e_7} = \{name, indicative, \overleftarrow{located}\}$,

$\overline{e_8} = \{address, city, \overleftarrow{situated}\}$.

Calcul de la sélectivité initiale des propriétés

La deuxième étape de calcul de la sélectivité des propriétés consiste à calculer le nombre d'occurrences de chaque propriété. Ainsi, nous définissons pour chaque propriété le nombre de fois où elle a été utilisée pour décrire une entité. Nous calculons le nombre d'apparition des propriétés sortantes, et les propriétés entrantes impliquées dans des déclarations $inverseOf$. Les autres propriétés entrantes ne sont pas nécessaires au calcul de la sélectivité comme nous allons le montrer dans la sous-section suivante, qui définit les formules de calcul de sélectivité.

Nous déterminons en plus des co-occurrences individuelles des propriétés, les co-occurrences des propriétés impliquées dans une déclaration schéma. Ainsi, nous calculons le nombre de fois où deux propriétés liées par une déclaration schéma sont présentes simultanément dans la description des entités. Si deux propriétés p_i et p_j sont impliquées dans une déclaration schéma $\langle p_i, schemaDeclaration, p_j \rangle$, nous sauvegardons pour chaque propriété p_i le nombre d'apparitions avec la propriété p_j. Nous utilisons pour cela le schéma saturé calculé durant l'étape précédente.

Exemple 5.6.2 *Dans notre exemple, la sélectivité initiale des propriétés est présentée dans le tableau 5.4. Par exemple, la propriété $name$ apparaît dans la description de trois entités, et l'ensemble des co-occurrences avec d'autres propriétés est vide car elle n'est impliquée dans aucune déclaration schéma. La propriété $situated$ décrit*

une entité, cette entité est impliquée dans une déclaration schéma avec la propriété $includes$, ainsi, nous sauvegardons le nombre de fois où ces deux propriétés décrivent la même entité. La propriété $situated$ est associée une fois avec la propriété $\overleftarrow{includes}$.

TABLE 5.4 – Le nombre d'occurrences des propriétés décrivant les entités de notre exemple

propriété	nombre d'apparition	combinaison avec autres propriété
$name$	3	$\emptyset$
$exhibition$	1	$\{(presentation, 0)\}$
$located$	1	$\{(\overleftarrow{includes}, 0)\}$
$sameConstruction$	1	$\{(\overleftarrow{sameConstruction}, 0)\}$
$presentation$	1	$\{(exhibition, 0)\}$
$situated$	1	$\{(\overleftarrow{includes}, 1)\}$
$theme$	2	$\emptyset$
$start$	2	$\emptyset$
$duration$	1	$\emptyset$
$email$	2	$\emptyset$
$address$	2	$\emptyset$
$worksAs$	1	$\{(occupation, 0)\}$
$fatherOf$	1	$\{(parent, 0)\}$
$visited$	2	$\emptyset$
$occupation$	1	$\{(worksAs, 0)\}$
$indicative$	1	$\emptyset$
$city$	1	$\emptyset$
$parent$	1	$\{(fatherOf, 0), (familyMember, 0)\}$
$familyMember$	1	$\{(parent, 0)\}$
$include$	1	$\{(situated, 1), located, 0)\}$
$\overleftarrow{include}$	1	$\emptyset$
$\overleftarrow{situated}$	1	$\emptyset$
$\overleftarrow{located}$	1	$\emptyset$

Calcul de la sélectivité finale des propriétés

Enfin, la sélectivité des propriétés est définie en utilisant les règles de calcul suivantes :

- si une propriété p_1 est *symétrique*, le nombre d'entités décrites par p est égale à $|[p_1]| = |[p_1]| * 2 - |[p_1] \cap [\overleftarrow{p_1}]|$. Dans un triplet $\langle e_1, p_1, e_2 \rangle$ une propriété p_1 symétrique fait partie de la description de l'entité e_1 explicitement, et de e_2 implicitement. Ainsi, la cardinalité de la propriété p_1 est calculée grâce à son nombre d'apparitions explicites multiplié par 2 en éliminant le cas où la propriété est comptée deux fois pour la même entité. En

effet, il faut tenir compte des cas où la propriété est en même temps entrante et sortante pour une même entité.

- si une propriété $p_1\, equivalentProperty\, p_2$, le nombre d'entités décrites par p_1 est égale $|[p_1]| = |[p_1]| + |[p_2]| - |[p_1] \cap [p_2]|$. Dans ce cas, chaque entité possédant p_1 aura forcément p_2 et vice versa. Ainsi, la cardinalité de chacune est la somme de leurs cardinalités, moins le cas où une entité possédait explicitement les deux propriétés.
- si $p_2\, subProperty\, p_1$, le nombre d'entités décrites par p_1 est égale $|[p_1]| = |[p_1] + [p_2]| - |[p_1] \cap [p_2]|$. Dans ce cas, chaque entité possédant p_1 aura forcément p_2. Ainsi, la cardinalité de p_1 est le nombre d'apparitions de p_1 plus p_2, moins le cas où une entité possédait explicitement les deux propriétés.
- si $p_1\, inverseOf\, p_2$ alors le nombre d'entités décrites par p_1 est égal $|[p_1]| = |[p_1] + [\overleftarrow{p_2}]| - |[p_1] \cap [\overleftarrow{p_2}]|$. Une entité possédant la propriété $\overleftarrow{p_2}$ entrante signifie qu'elle était l'objet de la propriété p_2 dans un triplet RDF. Si p_1 est l'inverse de p_2, alors toutes les entités ayant $\overleftarrow{p_2}$ entrante auront forcément p_1.

Les propriétés ayant la même sélectivité sont ordonnées selon l'ordre naturel.

Exemple 5.6.3 *Par exemple, le nombre final d'entités décrites par la propriété* $exhibition$ *est égale à* $|[exhibition]| = |[exhibition]| + |[presentation]| - |[exhibition] \cap [presentation]| = 1 + 1 - 0 = 2$ *car il existe une déclaration schéma* $\langle exhibition, owl{:}equivalentProperty, presentation\rangle$. *La propriété* $situated$ *décrit au total* 2 *entités,* $|[situated]| = |[situated]| + |[located]| + |[\overleftarrow{includes}]| - |[situated] \cap [\overleftarrow{located}]| - |[situated] \cap [\overleftarrow{includes}]| = 1+1-0-1 = 2$*, car cette propriété est impliquée dans les déclarations schéma* $\langle situated, sameRole, located\rangle$ *et* $\langle situated, inverseOf, includes\rangle$.

Enfin, en considérant la sélectivité des propriétés, l'ordre de ces dernières est le suivant : $city <_{\mathcal{P}} duration <_{\mathcal{P}} fatherOf <_{\mathcal{P}} indicative <_{\mathcal{P}} parent <_{\mathcal{P}} address <_{\mathcal{P}} email <_{\mathcal{P}} exhibition <_{\mathcal{P}} familyMember <_{\mathcal{P}} includes <_{\mathcal{P}} located <_{\mathcal{P}} occupation <_{\mathcal{P}} presentation <_{\mathcal{P}} sameCnstruction <_{\mathcal{P}} situated <_{\mathcal{P}} start <_{\mathcal{P}} theme <_{\mathcal{P}} visited <_{\mathcal{P}} worksAs <_{\mathcal{P}} name$.

Pour assurer un traitement rapide, ce processus est parallélisé, et la sélectivité de chaque entité est calculée par un processus indépendant. L'algorithme 13 présente le processus parallèle du calcul de sélectivité. Tout d'abord, il parcourt en parallèle les entités du jeu de données afin de calculer le nombre d'apparitions de chaque propriété p, et l'association de p avec les autres propriétés si elles sont impliquées dans la même déclaration schéma (lignes 1-11). Ensuite, les résultats obtenus sont fusionnés afin de calculer le nombre d'apparitions de chaque propriété. Enfin, la sélectivité finale de chaque propriété est calculée en parallèle, en appliquant les règles de calcul de sélectivité qui se basent sur les déclarations schéma du jeu de données.

5.6.2 Saturation et distribution des entités

Dans le but de découvrir le schéma par clustering, nous distribuons à travers des $chunks$, les entités du jeu de données par rapport aux propriétés les décrivant afin d'offrir un traitement scalable capable de gérer de grands

Algorithm 13 Calcul de sélectivité

Input: jeux de données D, déclaration schéma ds
1: **for all** e in D **do in parallel**
2: **for all** property p in $\overline{e}$ **do**
3: **for all** property p' in $\overline{e}$ **do**
4: **if** $\exists \langle p, pred, p' \rangle \in ds\, OR\, \langle p', pred, p \rangle \in ds$ **then**
5: $selecInit = selecInit \cup \{(p, (1, \{p', 1\}))\}$
6: **else**
7: $selecInit = selecInit \cup \{(p, (1, \emptyset))\}$
8: **end if**
9: **end for**
10: **end for**
11: **end for**
12: Fusionner les listes calculant le nombre d'apparition des entités
13: **for all** $(p, comb) in selecInit$ **do in parallel**
14: $selecFinal = selecFinal \cup calculeSelect(p, comb, ds)$
15: **end for**
16: **return** selecFinal

jeux de données. Ainsi, les entités similaires appartenant à la même classe sont identifiées dans chaque $chunk$ en parallèle. Enfin, nous avons introduit le *seuil de dissimilarité* afin de réduire le nombre de propriétés à considérer pour chaque entité. Ce principe de distribution a été détaillé dans le chapitre 3.

Dans notre approche, afin de tenir compte des propriétés implicites décrivant les entités et dérivées en exploitant la sémantique des jeux de données RDF, les entités sont saturées lors de leur distribution. Dans notre contexte, la saturation permet l'extraction des propriétés implicites décrivant les entités afin d'en tenir compte lors du clustering et de la distribution des entités.

Pour assurer la prise en compte des propriétés implicites durant la distribution des données tout en offrant un traitement scalable, les entités sont distribuées sur les différents nœuds de calcul. Afin de saturer une entité, nous avons besoin de connaître tous les triplets schéma qui comportent des déclarations liées aux propriétés de cette entité. Pour cela, nous diffusons sur tous les nœuds de calcul les triplets schéma saturés afin de permettre un traitement parallèle sans échange d'informations entre les nœuds. Ensuite, les entités sont saturées sur chaque nœud de calcul en parallèle et distribuées à travers les $chunks$ suivant les propriétés décrivant les entités. La saturation d'une entité est faite en appliquant les règles de saturation pour les instances identifiées dans le tableau 5.3. Comme la saturation du schéma produit tous les triplets nécessaires à la saturation des triplets instances, la saturation des entités se fait en une seule itération sur ces derniers.

Dans notre approche, les données sont divisées aléatoirement à travers les nœuds de calcul pour assurer un traitement parallèle. La division aléatoire des données assure la création de sous-ensembles de données de tailles égales ce qui permet une création rapide de partitions de tailles égales et le traitement rapide des données dans chaque partition. Dans chaque partition, chaque entité e_i est saturée afin de dériver les propriétés implicites décrivant e_i.

Pour saturer les entités, nous appliquons les règles de saturation sur chaque propriété décrivant les entités

et faisant partie des triplets schéma. Dans ce but, les triplets schéma initiaux ainsi que ceux dérivés durant la première étape sont diffusés sur les nœuds de calcul. Pour chaque entité e_i décrite par la propriété p_i, nous vérifions l'existence de triplets schéma $\langle p_i, schemaDeclaration, p_j \rangle$ comportant la propriété p_i. Ainsi, les propriétés implicites sont dérivées en appliquant les règles de saturation qui prennent en prémisse le triplet $\langle p_i, schemaDeclaration, p_j \rangle$.

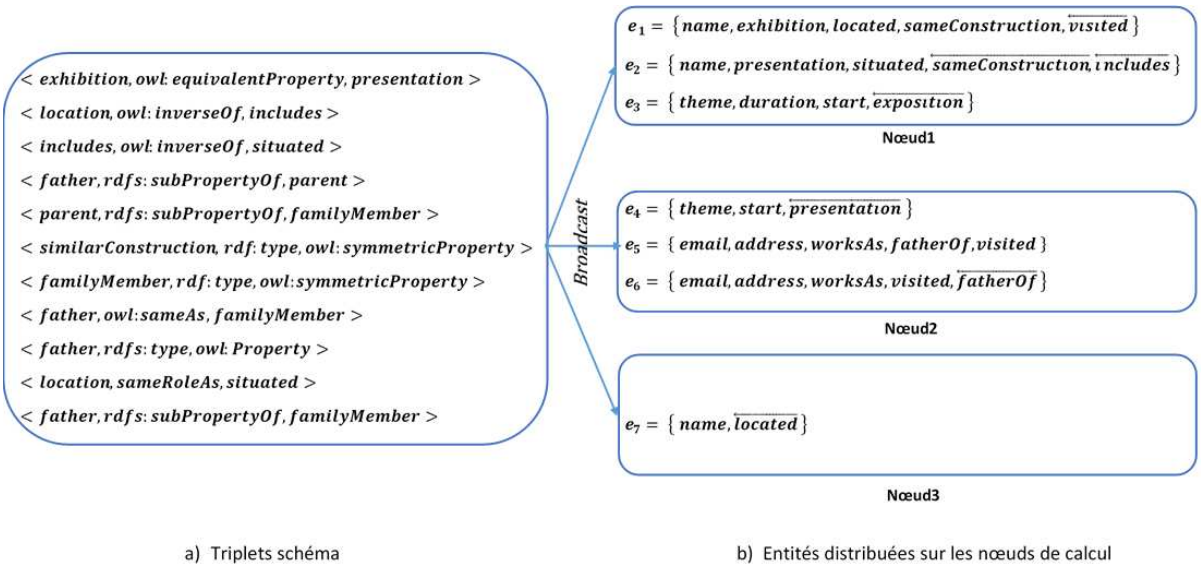


FIGURE 5.6 – L'exécution parallèle de la saturation des entités du jeu de données

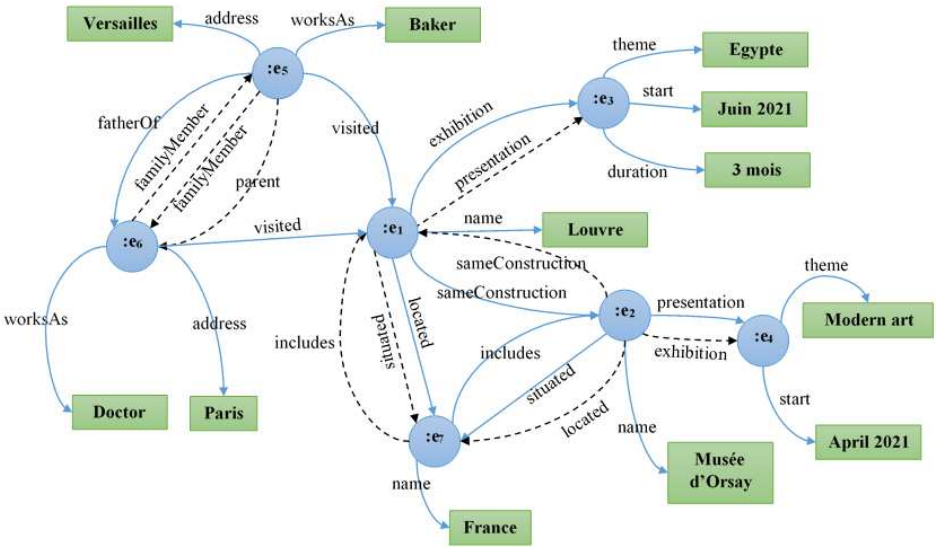


FIGURE 5.7 – La description des entités du jeu de données enrichie par les propriétés implicites

Exemple 5.6.4 *La saturation des entités de notre exemple produit le graphe présenté dans la figure 5.7, où les triplets implicites sont représentés par des pointillés. D'abord, les données sont divisées aléatoirement sur les différents nœuds de calcul. Dans la figure 5.6.b, les données sont partitionnées aléatoirement en 3 partitions. Durant cette étape, les triplets schéma saturés (figure 5.6.a) nécessaires à la saturation des entités sont diffusés*

sur les nœuds de calcul. Ensuite, les entités sont saturées en dérivant les propriétés implicites à partir des triplets schéma explicites et implicites. Par exemple, l'entité e_1 *est enrichie avec les propriétés* $situated$ *et* $presentation$*, car il existe des déclarations schéma* $located\, owl{:}sameRole\, situated$ *et* $exhibition\, owl{:}sameRole\, presentation$*. Ainsi, la règle* 13 *est appliquée pour dériver les propriétés* $situated$ *et* $presentation$ *qui sont ajoutées à la description de l'entité* e_1*,* $\overline{e_1^S} = \{name, exhibition, located, sameConstruction, situated, presentation\}$*.*

Algorithm 14 Saturation d'entité

Input: l'entité e, déclaration schéma ds

1: **for all** property p in $\overline{e}$ **do**
2: **if** $p.estEntrante$ **then**
3: **if** $\exists\langle p, pred, p'\rangle \in ds$ OR $\langle p', pred, p\rangle \in ds, /pred \in \{symetric, inverseOf\}$ **then**
4: $\overline{e} = \overline{e} \cup p'$
5: **end if**
6: **else**
7: **if** $\exists\langle p, pred, p'\rangle \in ds$ OR $\langle p', pred, p\rangle \in ds, /pred \in \{sameRole, equivalentPro\}$ **then**
8: $\overline{e} = \overline{e} \cup p'$
9: **else if** $\exists\langle p, subProperty, p'\rangle \in ds$ **then**
10: $\overline{e} = \overline{e} \cup p'$
11: **end if**
12: **end if**
13: **end for**
14: **return** e

L'algorithme 14 décrit l'opération de saturation d'entités. Pour chaque propriété décrivant l'entité e (ligne 1), l'algorithme vérifie les déclarations schéma qui portent sur cette propriété p. Dans le cas où p est une propriété entrante ($\overleftarrow{p}$), l'algorithme vérifie si p est impliquée dans une déclaration $inverseOf$ ou $symmetricProperty$ afin d'enrichir la déclaration de e avec p (lignes 2-5). Dans le cas contraire, la description de e est enrichie avec une nouvelle propriété p' si, p et p' font partie d'une description $sameRole$ ou $equivalentProperty$ (lignes 6-8). Enfin, si p est une $subProperty$ d'une propriété p', p' est ajoutée à la description de l'entité e (lignes 9-12).

Après l'enrichissement d'entités, les $chunks$ auxquels cette entité doit être assignée sont déterminés suivant notre principe de distribution et le *seuil de dissimilarité*, en considérant l'ensemble des propriétés explicites et implicites décrivant l'entité.

Exemple 5.6.5 *Si on considère dans notre exemple que* $\epsilon = 0.65$*, la distribution des données produit les* $chunks$ *présentés dans la figure 5.8.*

Par exemple, le seuil de dissimilarité de l'entité e_1 *est égal à* $dt(e_1) = 5 - \lceil 0.65 \times 5 \rceil + 1 = 2$*. Les deux propriétés les plus sélectives décrivant* e_1 *sont* $exhibition$ *et* $located$ *et ainsi* e_1 *est assignée aux* $chunks$ $[exhibition]$ *et* $[located]$*.*

L'algorithme 15 décrit la distribution des données en tenant compte des connaissances implicites. Il prend en entrée le seuil de similarité ϵ, utilisé pour calculer le seuil de dissimilarité de chaque entité.

Tout d'abord, il sature la description des entités en dérivant les propriétés implicites à partir de l'ensemble ds qui comporte les déclarations liées au schéma (ligne 2). Ensuite, les $chunks$ auxquels une entité est affectée

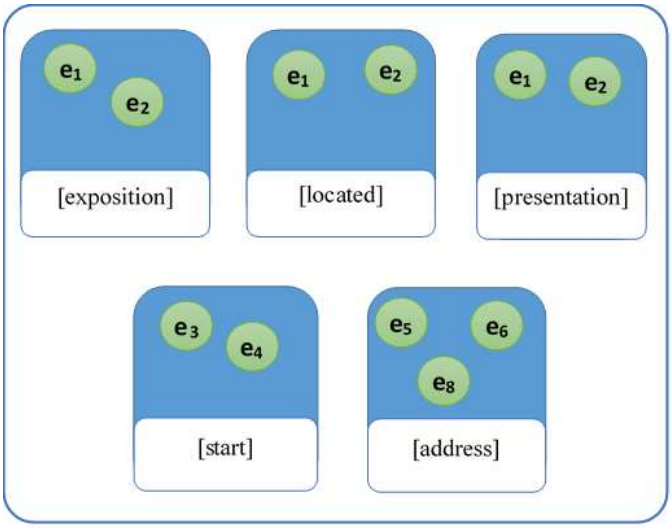


FIGURE 5.8 – Les $chunks$ résultant de la distribution des entités après l'enrichissement de leurs descriptions

Algorithm 15 Distribution des entités

Input: jeux de données D, seuil de similarité ϵ, déclaration schéma ds

1: **for all** e in D **do in parallel**
2: $e' = saturate(entity, ds)$
3: **for all** property p in $ch(e')$ **do**
4: $[p] = [p] \cup \{e'\}$
5: **end for**
6: **end for**
7: Merge the chunks generated by the parallel execution for the same properties
8: **return** the chunks

sont définies (lignes 3-5). Les $chunks$ créés en parallèle sont ensuite fusionnés afin d'obtenir les $chunks$ finaux (ligne 7).

5.6.3 Découverte de schéma

Après avoir complété la description des entités du jeu de données en utilisant la saturation pour dériver les propriétés implicites décrivant les données, et les avoir distribuées à travers les $chunks$, le schéma est découvert en utilisant le clustering parallèlement dans les différents $chunks$.

Pour cela, le voisinage des entités est calculé dans chaque $chunk$. La similarité entre les entités est évaluée en considérant l'ensemble des propriétés explicites et implicites, décrivant les entités. Ensuite, la liste des voisins de chaque entité est consolidée pour former la liste des voisins pour chaque entité dans le jeu de données global. Durant cette opération, les entités cores ayant un voisinage dense sont identifiées.

A partir de ces entités cores et leurs voisinages, les clusters sont formés en parallèle dans les différents $chunks$.

Enfin, les clusters qui s'étendent sur différents $chunks$ sont construits en fusionnant les clusters locaux partageant des entités cores en commun.

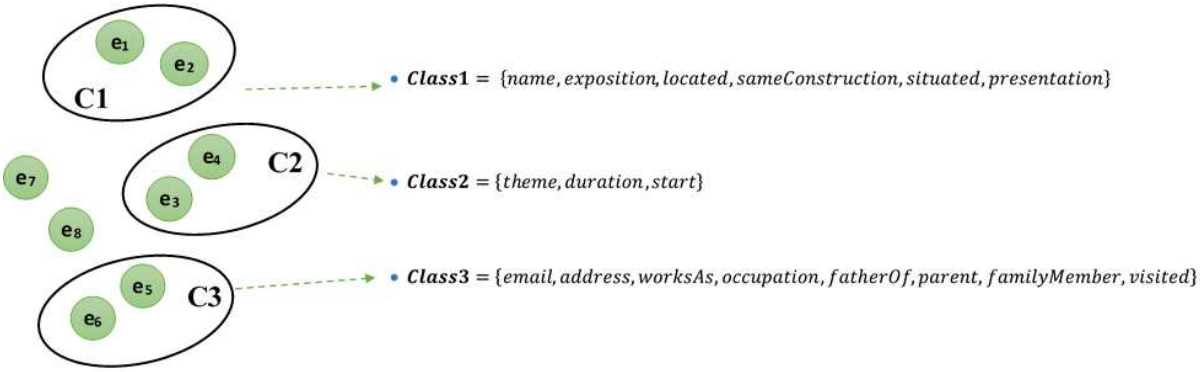


FIGURE 5.9 – Les classes décrivant les entités du jeu de données D après leur enrichissement

Les clusters formés par notre algorithme de clustering représentent les classes du schéma décrivant les entités contenues dans une source de données RDF.

Exemple 5.6.6 *Dans notre exemple, en considérant* $\epsilon = 0.65$ *et* $minPts = 1$*, les clusters formés sont présentés dans la figure 5.9.*

Par conséquent, le schéma découvert montre que le jeu de données RDF comporte des instances de la classe $museum$ *décrites par l'ensemble de propriétés* $\{name, exhibition, located, sameConstruction, situated, exhibition\}$*, des instances de la classe* $exhibition$ *décrites par les propriétés* $\{theme, duration, start\}$ *et des instances de la classe* $visitor$ *décrites par* $\{address, worksAs, visited, parent, fatherOf, familyMember\}$*.*

5.7 Conclusion

Les approches de découverte de schéma qui analysent la structure des entités dans un jeu de données RDF offrent un schéma de bonne qualité. Ces approches n'utilisent que les propriétés explicites qui décrivent les entités et ignorent la sémantique qui peut être dérivée à partir de ces sources de données.

Dans ce chapitre, nous avons adapté notre approche de découverte pour qu'elle tienne compte des triplets implicites en introduisant du raisonnement dans le but d'améliorer le schéma découvert. Nous avons proposé d'utiliser certaines déclarations trouvées dans les jeux de données RDF pour dériver les propriétés implicites et d'enrichir la structure des entités avec des propriétés implicites, qui n'ont pas été définies lors de la création du jeu de données. Notre approche extrait le schéma décrivant une source de données RDF en analysant les propriétés explicites et implicites décrivant les entités. Nous avons redéfini notre méthode de distribution des entités sur les $chunks$ afin qu'elle considère aussi bien les propriétés explicites qu'implicites tout en offrant un processus parallèle et scalable qui pourra bénéficier d'une implémentation en utilisant des technologies big data comme Spark.

Une autre perspective intéressante pour notre travail serait de proposer une version incrémentale de notre approche, qui fait évoluer le schéma découvert lorsqu'un jeu de données RDF est mis à jour. L'approche doit

considérer aussi bien les modifications qui portent sur les données que celles qui concernent les déclarations schéma.

La saturation des entités dans le but de perfectionner le schéma produit représente une amélioration par rapport aux approches qui analysent la structure des données afin de construire le schéma. Dans cette approche, nous n'avons considéré que les déclarations schéma utilisées pour dériver les propriétés implicites décrivant les entités. Une amélioration possible serait d'étudier les autres déclarations schéma afin de proposer une approche hybride qui exploiterait toutes les déclarations dans une source de données RDF. Par exemple, comment intégrer la description d'une classe déjà déclarée dans le jeu de données dans le processus de découverte de schéma ? La déclaration $owl{:}subClassOf$ peut également être utilisée afin de former les liens hiérarchiques. Les propriétés $rdfs{:}domain$ et $rdfs{:}range$ définissent respectivement la classe des sujets et objets liés à une propriété, et peuvent être utilisées afin d'enrichir le schéma avec des contraintes d'intégrité. Une étude détaillée des différentes déclarations schéma devra être réalisée pour déterminer la façon dont elles peuvent être prise en compte lors de la découverte de schéma.

Dans notre approche, nous avons considéré les déclarations schéma incluses dans une source de données RDF. Une alternative serait de considérer des déclarations fournies par une source de connaissances extérieures, telle une ontologie de domaine par exemple. Ceci permettrait d'adapter la découverte de schéma à un domaine spécifique. Il serait également possible de générer des schémas différents pour la même source de données si des ontologies différentes étaient utilisées, correspondant à des domaines d'application différents.

Chapitre 6

Expérimentations

6.1 Introduction

Nous nous sommes intéressés dans ce travail au problème de la découverte automatique de schéma à partir de sources de données RDF massives. Pour cela, nous avons proposé dans un premier temps une approche pour l'extraction d'une représentation condensée d'une source de données RDF. Pour extraire le schéma d'une source de données RDF massive ayant un nombre de patterns très élevé, nous avons proposé une approche fondée sur un algorithme de clustering basé sur la densité scalable. Nous avons ensuite proposé une approche incrémentale de découverte de schéma qui met à jour les classes du schéma en tenant compte des nouvelles insertions dans le jeu de données. Enfin, nous avons adapté notre approche de découverte de schéma afin d'exploiter les déclarations schéma partiellement fournies dans une source de données RDF et ainsi améliorer la qualité du schéma produit.

Nous présentons dans ce chapitre une étude expérimentale de nos différentes propositions dédiées à la découverte de schéma. Le but de nos expérimentations est de montrer l'efficacité de nos méthodes pour l'extraction d'un schéma de bonne qualité pour des sources de données RDF massives, qui sont de plus sujettes à des évolutions fréquentes. Pour cela, nous évaluons :

- le gain qu'apporte notre méthode d'extraction de patterns en termes de réduction de la taille d'un jeu de données RDF,
- les performances de notre approche incrémentale en la comparant avec notre approche scalable qui exige la disponibilité du jeu de données global pour s'exécuter.
- la qualité des classes que produisent nos approches de découvertes de schéma.

Ce chapitre détaille l'environnement d'expérimentation dans la section 6.2. Elle comporte une description des clusters de calcul ainsi que des jeux de données utilisés lors des évaluations. La section 6.3 présente les expérimentations sur l'approche d'extraction de patterns et montre le ratio de réduction de la taille des jeux de données RDF. Dans la section 6.4, nous présentons les expérimentations qui portent sur la scalabilité et l'incrémentalité de

nos approches de découverte de schéma afin de démontrer leur capacité à traiter des jeux de données massifs et en constante évolution. La qualité du schéma produit par nos propositions est discutée dans la section 6.5. Enfin, la section 6.6 conclut ce chapitre et présente une synthèse de nos évaluations.

6.2 Environnement de test

Cette section décrit d'abord les caractéristiques des clusters de calcul utilisés dans les différentes expérimentations. Ensuite, elle décrit les jeux de données sur lesquels nous avons conduit nos évaluations.

6.2.1 Description des clusters de calcul

Toutes nos expérimentations ont été conduites dans un environnement distribué permettant de concevoir des applications scalables permettant le traitement de grand volume de données en offrant une exécution parallèle sur plusieurs nœuds de calcul.

Nous avons utilisé dans nos expérimentations différentes configurations de clusters de calcul afin de montrer la scalabilité de nos approches lorsqu'elles sont exécutées sur des clusters de puissances différentes.

Cluster local

Nous avons tout d'abord conduit nos expérimentations sur un cluster local, exécutant Ubuntu Linux et composé de 5 nœuds (1 $master$ and 4 $workers$). Chaque nœud est équipé de 30 Go de RAM et un processeur 12-cores. Notre implémentation repose sur le framework Apache Spark 2.4.

Plateforme OpenStack

Afin de montrer la scalabilité de notre approche de découverte de schéma, nous avons créé en utilisant la plateforme OpenStack [1], plusieurs clusters comportant des nombres différents de nœuds de calcul. Chacun des clusters est composé d'un $master$ équipé de 4 Go de RAM et un processeur 4-cores. Le nombre de $workers$ des clusters varie entre 2 et 8 et chaque $worker$ est équipé de 16 Go de RAM et un processeur 6-cores. La même version du framework Apache Spark 2.4 a été installée sur ce cluster.

6.2.2 Jeux de données

Nous avons utilisé dans nos expérimentations différents jeux de données synthétiques ou réels afin de montrer l'efficacité de nos propositions lorsque celles-ci sont appliquées sur des jeux de données de tailles et de configurations différentes.

1. https://galactica.isima.fr/doc/

Afin d'évaluer notre approche d'extraction de patterns, nous avons utilisé des jeux de données RDF réels de tailles différentes :

- **DBpedia [2].** Dbpedia est un projet visant à proposer une version structurée et normalisée au format du web sémantique des contenus de Wikipédia, et à le rendre disponible sur le web sous la forme d'un jeu de données RDF. DBpedia est divisé en plusieurs sous-ensembles selon la langue utilisée pour la description des ressources.
- **DBLP [3].** DBLP est une extension au format RDF des données du catalogue de bibliographies en informatique publié par le site dblp [4]. La source de données RDF contient les méta-données pour plus de 1.8 millions de publications.
- **Katrina et Charley [5].** Ces deux sources de données RDF qui contiennent des données d'observations d'ouragans et de tempêtes aux État-Unis.

Le tableau 6.1 présente pour chaque jeu de données, le nombre de triplets et le nombre d'entités qu'il comporte.

TABLE 6.1 – Caractéristiques des jeux de données

Jeu de donées	Triples	Entités
DBpedia	9 500 000 000	66 195 296
DBLP	222 375 855	16 086 516
Katrina	203 386 049	3409
Charley	101 956 760	3353

Nous avons évalué la scalabilité de notre algorithme de clustering et de l'algorithme incrémental en utilisant des jeux de données synthétiques en complément des jeux de données réels. Les jeux de données synthétiques sont produits en utilisant le générateur "IBM Quest Synthetic Data Generator" [55]. Ce générateur a été très largement utilisé dans la communauté *Data Mining* afin d'évaluer les performances des algorithmes de découverte d'itemsets fréquents. Dans notre contexte, nous avons utilisé le générateur pour produire les propriétés de chaque entité utilisée dans nos expérimentations, et pour ajuster leurs caractéristiques.

Les différentes caractéristiques des données à considérer dans nos expérimentations sont (i) la taille du jeu de données afin d'étudier la scalabilité de nos algorithmes, (ii) le nombre total de propriétés décrivant un jeu de données et (iii) le nombre moyen de dimensions (propriétés) des entités.

En plus des données synthétiques, nous avons utilisé des jeux de données réels de différentes tailles extraits à partir de DBpedia [6]. DBpedia structure au format RDF le contenu de Wikipeda en 119 langues. Dans nos évaluations, nous avons extrait à partir de DBpedia des sous-ensembles de patterns qui représentent toutes les com-

2. https://old.datahub.io/dataset/dbpedia
3. https://old.datahub.io/dataset/dblp
4. https://dblp.uni-trier.de/
5. http://wiki.knoesis.org/index.php/LinkedSensorData
6. http://downloads.dbpedia.org/3.9/

binaisons de propriétés décrivant les entités dans le jeu de données. Exécuter le clustering sur les patterns au lieu des données initiales permet d'accélérer la découverte de schéma. Nous avons utilisé les versions anglaises, françaises, espagnoles, néerlandaises, britanniques et arabes de DBpedia dont les caractéristiques sont données dans le tableau 6.2.

TABLE 6.2 – Caractéristiques des sous ensembles de Dbpedia

Jeu de données	Entités	Patterns
DBpediaEn	23 634 425	1.23 million
DBpediaFr	6 765 834	626 381
DBpediaEs	6 014 121	529 434
DBpediaNl	4 405 538	268 603
DBpediaUk	1 611 489	129 762
DBpediaAr	1 202 321	63 000

Enfin, afin d'évaluer la qualité du schéma produit par notre approche, nous avons extrait à partir de DBpedia les entités possédant un type (class) connu, et nous les avons considérées comme une vérité terrain. Dans nos évaluations, nous avons considéré les entités pour les types suivants : Aircraft, Artist, Athlete, Book, Disease, Newspaper, Region et TelevisionStation. Ces entités représentent une référence à laquelle les clusters générés sont comparés.

6.3 Évaluation de la taille de la représentation condensée

Nous évaluons dans un premier temps l'efficacité de l'approche d'extraction de patterns pour réduire le nombre d'entités en entrée de l'algorithme de clustering d'extraction de schéma. Cette section présente les résultats de nos expérimentations pour l'approche d'extraction de patterns, en particulier pour déterminer le ratio de réduction de la taille du jeu de données et le temps d'exécution nécessaire à cette réduction.

Le tableau 6.3 illustre l'efficacité de notre approche d'extraction de schéma et montre le nombre de patterns extraits à partir de chaque jeu de données, le ratio de réduction (nombre de patterns divisé par le nombre d'entités) et le temps d'exécution pour le calcul des patterns.

TABLE 6.3 – Ratio de réduction de la taille des jeux de données

Jeu de données	Patterns	Ratio (%)	Temps (s)
DBpedia	1 918 480	2.89	750
DBLP	351	0.002	163
Katrina	37	1.08	100
Charley	52	1.55	50

Le nombre de patterns est la représentation condensée d'un jeu de données et dépend fortement de l'hétérogénéité des structures décrivant les entités. Plus les ensembles de propriétés décrivant les entités sont hétérogènes, plus le nombre de patterns est élevé. Si on considère DBpedia, le nombre de patterns est élevé car ce jeu de données comporte des entités très hétérogènes contrairement à DBLP, Katrina et Charley qui sont moins hétérogènes et produisent moins de patterns. En ce qui concerne le temps d'exécution, les expérimentations montrent que notre approche est capable de traiter de grands jeux de données RDF tels que DBpedia qui est composé de plus de 9 milliards de triplets.

Le temps d'exécution se décompose en trois parties principales présentées dans le tableau 6.4, qui sont le temps nécessaire à la composition des entités à partir des triplets (entité), l'extraction de patterns (Pattern) et le calcul du nombre d'entités représentées par chaque pattern (Stats). L'étape de composition des entités charge en parallèle les triplets et échange les informations entre les différents nœuds de calcul afin de définir pour chaque entité l'ensemble de propriétés la décrivant. Comme l'échange d'information entre les nœuds ($shuffle$) est une opération coûteuse, le temps d'exécution est plus élevé pour les entités décrites par un grand nombre de triplets. Cette étape de pré-traitement représente l'opération la plus coûteuse de notre approche. La même observation est faite sur l'étape d'extraction de patterns où un nombre élevé d'entités entraîne un temps plus élevé pour la découverte des patterns. Enfin, la dernière opération parcourt les patterns et compte le nombre d'entités ayant le même pattern. Comme l'extraction des patterns se fait en parallèle sur les différents nœuds de calcul, le nombre d'entités décrites par le même pattern est d'abord calculé sur chaque nœud, ensuite la somme de ces nombres est calculée en échangeant les informations entre les nœuds.

TABLE 6.4 – Temps d'exécution de chaque opération

Jeu de données	**Total (s)**	**Entités (s)**	**Pattern (s)**	**Stats (s)**
DBpedia	750	590	130	30
DBLP	163	120	40	3
Katrina	100	96	3	1
Charley	50	46	3	1

Le but de la représentation condensée d'un jeu de données est de permettre l'extraction de schéma en appliquant des algorithmes de clustering à partir de grandes sources de données dont la taille initiale ne permet pas l'utilisation de ces derniers. Nos évaluations montrent que pour certains jeux de données comme DBLP, Katrina et Charley, le nombre de patterns extraits permet l'application d'algorithmes de clustering sur les patterns pour en extraire le schéma. Cependant, l'extraction de schéma à partir de DBpedia reste hors de portée car les entités de ce jeu de données sont décrites par des ensembles de propriétés très hétérogènes, et elles sont ainsi représentées par un grand nombre de patterns. Sur ces derniers, un algorithme de clustering classique ne peut être directement appliqué.

Nous évaluons dans la suite la capacité de nos algorithmes de clustering à extraire le schéma à partir de sources de données RDF. Nous détaillons les expérimentations qui portent sur les algorithmes de clustering scalable dédiés à l'extraction de schéma à partir de grandes sources de données RDF, même dans le cas des jeux de données hétérogènes qui sont représentés par un grand nombre de patterns.

6.4 Évaluation de la scalabilité de l'approche de découverte de schéma

Cette section est dédiée à nos expérimentations pour évaluer les performances de notre approche appliquée à de grands jeux de données, qui représente l'objectif principal de nos travaux de thèse.

Nous avons tout d'abord évalué la scalabilité en montrant la capacité de notre algorithme à calculer des clusters à partir des données RDF et nous avons étudié son comportement sur des jeux de données de différentes configurations. Dans un deuxième temps, nous avons comparé les performances de notre algorithme de clustering avec celles obtenues par NG-DBSCAN, un algorithme de clustering par densité scalable, également implémenté en utilisant Spark. Enfin, nous évaluons l'efficacité de notre approche de découverte de schéma comparé à notre approche scalable, lorsqu'un jeu de données évolue en insérant un ensemble d'entités.

Dans nos expérimentations, nous avons utilisé *l'indice de Jaccard* pour évaluer la similarité entre les entités. Sauf indication contraire, les paramètres sont définis comme suit : le seuil de similarité ϵ à 0.8, le paramètre de densité $minPts$ à 3 et la capacité de calcul d'un nœud du cluster $capacity$ à 9000.

6.4.1 Approche scalable de découverte de schéma

Nous avons d'abord évalué la scalabilité de notre approche en utilisant plusieurs jeux de données synthétiques de différentes tailles. De plus, nous avons étudié le comportement de notre algorithme lors de son application sur des jeux de données ayant des caractéristiques différentes : (i) jeux de données comportant des entités de différentes dimensions (10, 20, 30 et 40 propriétés par entité) et (ii) jeux de données où les entités sont décrites par un nombre différent de propriétés. Nous avons aussi évalué l'évolution des performances de notre algorithme lorsqu'il est exécuté sur des clusters de calcul de différentes configurations, i.e. avec des nombres de *worker nodes* différents. Enfin, nous avons appliqué notre approche sur un jeu de données réel afin de montrer son efficacité sur des cas de figure réalistes.

Scalabilité de l'algorithme de clustering

La figure 6.1 montre le temps d'exécution de l'algorithme en fonction de la taille des jeux de données lorsque les entités sont décrites en moyenne par 10, 20, 30, et 40 propriétés.

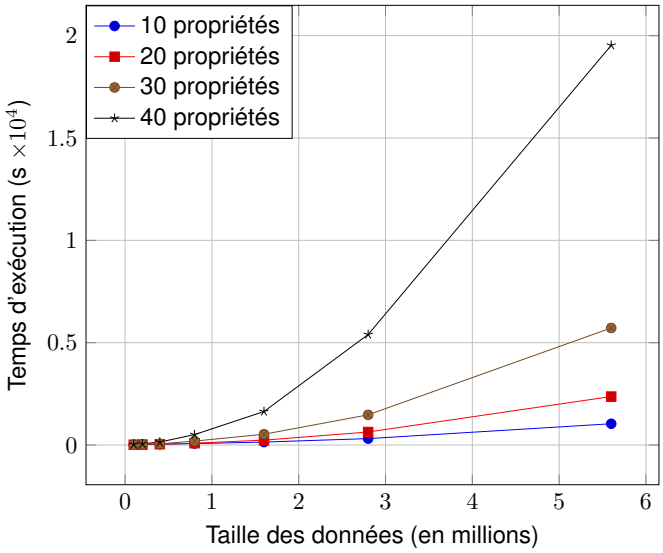


FIGURE 6.1 – Évaluation de la scalabilité de notre approche sur différents jeux de données synthétiques

Les résultats montrent l'efficacité de notre algorithme pour réaliser le clustering de grands jeux de données : il forme des clusters à partir d'un jeu de données comportant plus de 5 millions d'entités en 18 minutes, pour le jeu de données contenant des entités décrites en moyenne par 10 propriétés.

Les résultats s'expliquent par le fait que la distribution des données crée des petits sous-ensembles du jeu de données initial, sur lesquels le clustering est exécuté rapidement et en parallèle par les différents nœuds du cluster. Le calcul du voisinage des entités nécessite la comparaison de toutes les entités. Dans notre approche, cette opération est parallélisée, et chaque nœud de calcul évalue la similarité des entités pour une partie du jeu de données. De plus, des comparaisons inutiles sont évitées lors de la détermination du voisinage de chaque entité. En effet, les entités ne sont comparées que si elles se trouvent dans un même $chunk$. Chaque nœud de calcul identifie efficacement l'ϵ-voisinage des entités et les clusters partiels dans chaque $chunk$. Par ailleurs, les calculs sont distribués à travers les nœuds du cluster afin de minimiser les communications entre les nœuds, i.e. en évitant les coûteuses opérations de $shuffle$.

Quand la taille du jeu de données augmente, le processus nécessite plus de temps. Comme la distribution des données produit un grand nombre de $chunks$, chaque nœud de calcul doit traiter plusieurs $chunks$. De plus, les $chunks$ comportent un nombre élevé d'entités et peuvent être divisés récursivement afin de générer des $chunks$ ayant des tailles inférieures à la capacité des nœuds de calcul. Cette baisse de performance est plus visible quand le cluster atteint ses limites de calcul. Lorsque le cluster ne possède pas assez de ressources pour faire les calculs nécessaires au clustering d'un jeu de données, nous observons sur les différentes courbes une augmentation rapide du temps d'exécution. Cette limite est atteinte à différents stades en fonction des caractéristiques des jeux de données. Comme nous pouvons le voir sur la figure 6.1, lorsque les entités sont décrites en moyenne par 10 ou 20 propriétés, la limite du clusters est atteinte quand les jeux de données comportent plus de 2.8 millions d'entité,

tandis que pour les jeux de données qui comportent des entités décrites en moyenne par 30 ou 40 propriétés, la limite est atteinte lorsque que la taille des jeux de données dépasse 1.6 million d'entité.

Le même phénomène est observé quand le nombre de dimensions (i.e. propriétés) des entités augmente : le nombre d'entités augmente à l'intérieur des $chunks$ car les entités sont distribuées par rapport à leurs propriétés et le nombre de $chunks$ augmente aussi. Nous observons que les courbes ont le même comportement, mais la limite du cluster est atteinte pour des tailles différentes des jeux de données. La limite est atteinte lorsque l'algorithme est appliqué sur 5.8M d'entités pour le jeu de données où les entités sont décrites par 10, 20 et 30 propriétés, alors que cette limite est atteinte pour une taille de 2.8M du jeu de données quand les entités sont décrites par 40 propriétés.

Nous avons étudié l'impact du nombre total des propriétés décrivant un jeu de données. La figure 6.2 montre le temps d'exécution exigé par le clustering des jeux de données décrits par un nombre de propriétés qui varie entre 10k et 80k.

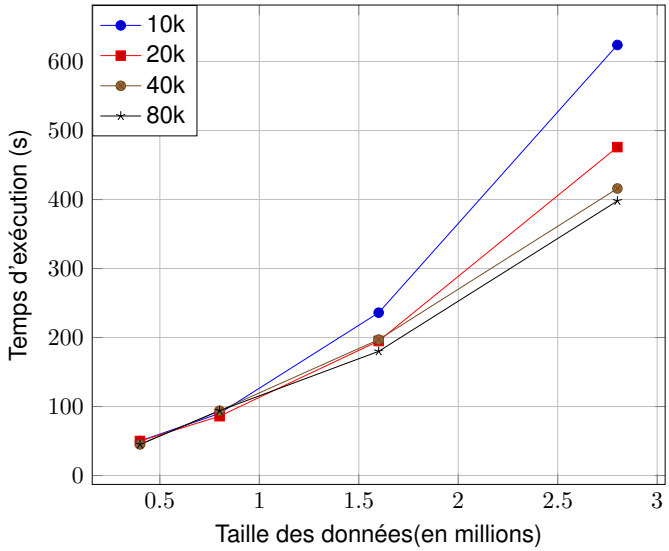


FIGURE 6.2 – Évaluation de l'impact du nombre de propriétés sur le temps d'exécution

Les expérimentations montrent que lorsque le nombre de propriétés augmente, le temps d'exécution baisse. Le fait d'avoir un grand nombre de propriétés implique la génération de plus de $chunks$ et une meilleure distribution des entités. Les entités étant réparties sur un plus grand nombre de sous-ensembles, leur nombre dans chaque $chunk$ est petit. Par conséquent, les $chunks$ créés sont de petites tailles dès la distribution initiale, ce qui ne requiert pas davantage de divisions des $chunks$. Ceci accélère la distribution des données et le clustering des entités.

Comportement de l'algorithme de clustering pour différentes tailles du clusters de calcul

Nous avons également étudié l'accélération de notre algorithme et l'impact du nombre de nœuds dans le cluster Spark sur ses performances, évaluées en termes de temps d'exécution. Nous avons fixé le temps de traitement

maximum à 5 heures et nous avons exécuté notre algorithme sur les jeux de données qui sont traités dans cet intervalle. Ces évaluations ont été conduites sur le cluster OpenStack dont nous avons fait varier le nombre de $workers$ entre 2 et 8.

La figure 6.3 montre l'évolution des performances de notre algorithme lorsque le nombre des nœuds $worker$ varie, en considérant des jeux de données de tailles comprises entre $500\,000$ et $3\,000\,000$ entités.

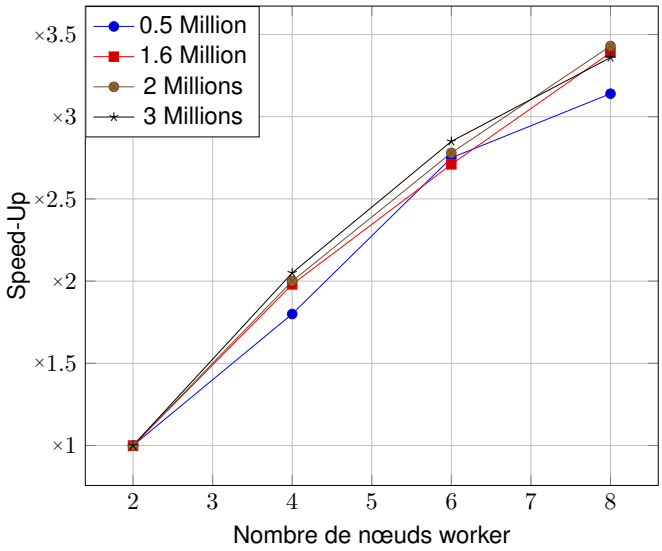


FIGURE 6.3 – Évaluation de l'accélération de l'algorithme exécuté sur des clusters de différentes configurations

Les expérimentations montrent que de meilleures performances et un clustering plus rapide sont obtenus lors de l'ajout de plus de nœuds $worker$ au cluster de calcul. Les résultats obtenus montrent que notre algorithme est scalable en dépit de la taille des jeux de données.

Application sur des jeux de données réels

Nous avons ensuite évalué l'efficacité de notre approche sur des jeux de données réels. La figure 6.4 montre la capacité de former des clusters à partir de jeux de données réels, comme DBpedia English qui représente une source de données RDF massive à partir de laquelle nous avons extrait plus de 1 million de patterns.

Ces résultats obtenus à partir des expérimentations indiquent que notre approche est scalable et adaptée aux grands jeux de données avec des caractéristiques variables. Le temps exigé pour le calcul du clustering dans les différentes expérimentations a été de l'ordre de quelques minutes, démontrant que notre approche est efficace dans plusieurs scénarios.

Comparaison avec NG-DBSCAN

Enfin, nous avons comparé notre approche à NG-DBSCAN, un algorithme de clustering scalable [72]. NG-DBSCAN est un algorithme DBSCAN scalable et parallèle qui offre de bonnes performances. Les comparaisons de

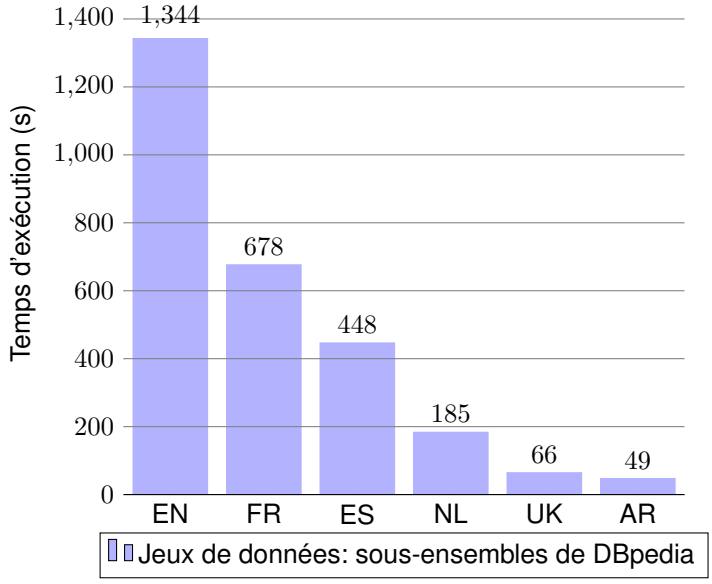


FIGURE 6.4 – Évaluation du temps de clustering de sous-ensembles de DBpedia

NG-DBSCAN avec les autres algorithmes DBSCAN scalables montrent qu'il surpasse les algorithmes de l'état de l'art. De plus, il est implémenté en utilisant le framework Apache Spark, qui représente le même environnement de développement utilisé dans l'implémentation de notre approche. En outre, contrairement à des algorithmes comme MR-DBSCAN et RP-DBSCAN, il peut être appliqué aux jeux de données RDF. Nous avons utilisé le code source fourni par les auteurs et disponible en ligne [7].

La figure 6.5 présente la fonction logarithmique du temps d'exécution nécessaire aux deux algorithmes pour effectuer le clustering de jeux de données de différentes tailles. Nous utilisons l'échelle logarithmique pour représenter le temps d'exécution car l'écart entre les performances des deux algorithmes est important et rend difficile leur comparaison.

Nos résultats montrent que les deux courbes ont une forme similaire, et que notre algorithme est toujours plus performant que NG-DBSCAN. Ceci est dû au fait que l'implémentation de NG-DBSCAN applique de nombreuses opérations $shuffle$, ce qui se traduit par de coûteux échanges d'informations entre les nœuds de calcul et ainsi augmente le temps d'exécution de l'algorithme. D'autre part, notre algorithme distribue intelligemment les données afin de réduire les communications entres les nœuds de calcul durant le calcul des clusters, et va ainsi réduire considérablement le temps d'exécution.

Les résultat obtenus par les différentes expérimentations montrent que notre proposition offre de bonnes performances en termes de vitesse d'exécution du processus de découverte de schéma, que ce soit sur les jeux de données synthétiques ou réels. Nous évaluons dans la section qui suit les performances de notre approche scalable et incrémentale de découverte de schéma.

7. `https://github.com/alessandrolulli/gdbscan`

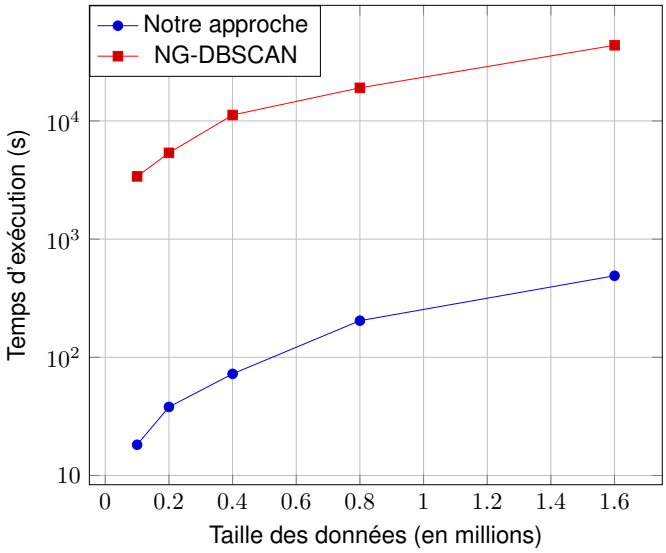


FIGURE 6.5 – Comparaison de notre algorithme de clustering avec l'algorithme NG-DBSCAN

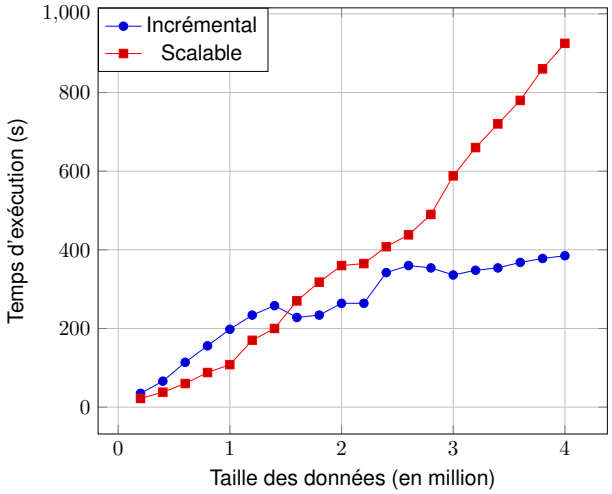
6.4.2 Approche incrémentale de découverte de schéma

Dans cette sous-section, nos expérimentations se concentrent sur les performances de nos approches appliquées aux grands jeux de données évolutifs. Ainsi, nous présentons les évaluations qui portent sur l'approche incrémentale de découverte de schéma. Nous présentons une comparaison entre l'approche de découverte de schéma scalable qui extrait le schéma à partir de la totalité d'un jeu de données et l'approche qui construit de façon incrémentale le schéma à partir d'un jeu de données qui évolue. Nous dérivons ainsi le facteur d'accélération lors de l'utilisation de l'approche incrémentale pour refléter l'insertion d'un ensemble d'entités sur les clusters au lieu d'utiliser notre approche de découverte de schéma scalable sur le jeu de données composé des anciennes entités et les nouvelles insérées.

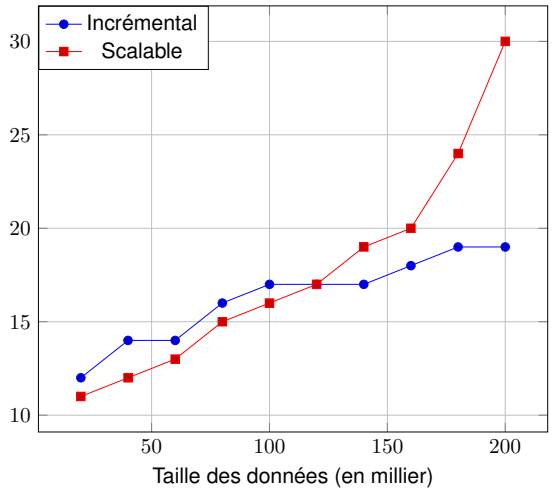
A chaque insertion d'entités Δ_D au jeu de données initial D, nous évaluons le temps d'exécution nécessaire à l'algorithme incrémental pour modifier le résultat du clustering obtenu à partir de D afin de refléter l'insertion de l'ensemble d'entités Δ_D. Le temps d'exécution de ce scénario est comparé au temps d'exécution nécessaire pour l'algorithme DBSCAN scalable afin de faire le clustering du jeu de données composé du jeu de données initial et de l'ensemble des entités insérées, i.e. $D \cup \Delta_D$.

Nous avons d'abord utilisé des jeux de données synthétiques comportant plus de 4 millions d'entités, générés en utilisant "IBM Quest Synthetic Data Generator" [55], introduit dans les expérimentations sur l'approche de découverte de schéma. Ce générateur produit les propriétés des entités que nous utilisons dans nos expérimentations. ensuite, comme la complexité de notre approche dépend du nombre d'entités insérées, nous avons également évalué notre approche de découverte de schéma incrémentale en insérant des ensembles de données de tailles différentes. Enfin, nous démontrons l'efficacité de notre approche sur des jeux de données réels. Dans ce but, nous appliquons notre approche sur 1.2 million d'entités extraites à partir de DBpedia [8] [7].

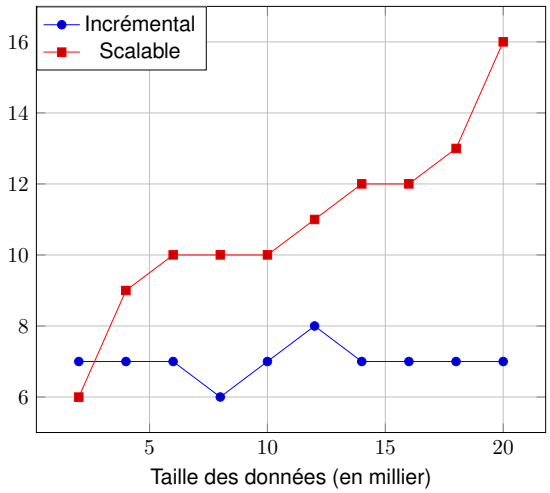
8. http://downloads.dbpedia.org/3.9/



(a) Par pas de 200k entités



(b) Par pas de 20k entités



(c) Par pas de 2k entités

FIGURE 6.6 – Algorithme incrémental vs. scalable

Évaluation des performances de l'algorithme incrémental

Nous avons tout d'abord évalué la scalabilité de notre approche et l'avons comparée à l'algorithme DBSCAN scalable proposé dans le chapitre 3 en utilisant plusieurs jeux de données synthétiques, auxquels nous avons ajouté des ensembles d'entités de différentes tailles. Les figures 6.6a, 6.6b et 6.6c montrent les temps d'exécution des deux algorithmes en fonction de la taille du jeu de données. L'algorithme scalable prend en entrée le jeu de données global alors que l'algorithme incrémental prend en entrée les clusters des exécutions précédentes et les entités nouvellement insérées.

Les résultats montrent que le clustering d'un petit jeu de données est plus rapide en utilisant l'approche scalable comparé à l'utilisation de l'approche incrémentale. Ceci est dû au fait que le clustering d'un petit nombre d'entités est très rapide et nécessite quelques secondes (22 secondes pour le clustering de 200k entités). D'autre part, quand le nombre d'anciennes entités est petit, les deux algorithmes ont presque le même nombre d'entités à gérer. Puisque

l'algorithme incrémental exécute des opérations supplémentaires, comme l'assignation des anciennes entités et l'union des données produites par cette assignation avec les $chunks$ créés durant la distribution des nouvelles entités, il est plus lent sur les petits jeux de données comparé à l'algorithme scalable.

Cependant, quand le nombre d'entités est plus grand, le clustering d'un jeu de données en utilisant l'algorithme DBSCAN incrémental est plus rapide. Ceci est dû au fait que le clustering est appliqué sur les nouvelles entités et leurs voisinages, ce qui contrebalance les opérations supplémentaires effectuées, alors que l'algorithme DBSCAN scalable doit former les clusters en calculant les voisinages de toutes les entités, ce qui représente une opération très coûteuse. De plus, l'approche incrémentale produit un plus petit nombre de nouveaux clusters comparé à l'approche scalable. Par conséquent, durant la fusion des clusters formés dans chaque $chunk$, qui s'effectue sur un seul nœud de calcul, l'algorithme incrémental gère moins de clusters ce qui le rend plus rapide.

Nous pouvons observer que plus le jeu de données est grand, plus l'écart entre les courbes des temps d'exécution des deux algorithmes est important, et plus le gain réalisé par l'approche incrémentale est grand.

Comme la complexité de l'algorithme incrémental est déterminée par le nombre de nouvelles entités et leurs voisinages, nous avons expérimenté l'insertion d'ensembles de données Δ_D de différentes tailles. Les résultats montrent que l'avantage de l'approche incrémentale comparé à l'approche scalable est observé à différents niveaux par rapport à la taille de l'ensemble d'entités ajouté. Plus les ensembles d'entités ajoutés sont petits, plus le clustering en utilisant l'algorithme incrémental est rapide. Dans nos expérimentations, lors de l'ajout de 200k entités à chaque exécution, l'algorithme incrémental devient plus rapide que l'algorithme scalable quand la taille du jeu de données globale atteint les 1.6M entités, alors que dans le cas de l'ajout de 20k entités à chaque exécution, il devient plus rapide quand le jeu de données atteint la taille de 140k entités (figure 6.6b). Quand la taille de l'ensemble des entités insérées est plus petite, le gain réalisé par l'algorithme incrémental est plus important, comme le montre la figure 6.6c, l'approche incrémentale surpasse l'approche scalable après la deuxième insertion. Ces résultats sont expliqués par le fait que l'algorithme incrémental produit les clusters seulement à partir des nouvelles entités et leurs voisinages et ne prend pas en considération tout le jeu de données. Plus l'ensemble de données insérées est petit, moins le nombre d'entités à gérer par l'algorithme est important, ce qui rend son exécution plus rapide.

Application sur des jeux de données réels

présente la fonction logarithmique du temps d'exécution nécessaire aux deux algorithmes pour effectuer le clustering

Enfin, nous avons évalué l'efficacité de notre approche sur des jeux de données réels. La figure 6.7 illustre la capacité de notre algorithme incrémental à faire le clustering de jeux de données réels, comme DBpedia, une source de données RDF volumineuse à partir de laquelle nous avons extrait 1.2 million de patterns. Le temps d'exécution nécessaire aux deux algorithmes est présenté par sa fonction logarithmique afin de montrer la différence

des performances des deux approches. Les patterns représentent toutes les combinaisons de propriétés décrivant les entités du jeu de données. Le clustering est ainsi appliqué sur les patterns. Similairement aux évaluations sur les jeux de données synthétiques, nous avons ajouté à chaque insertion au jeu de données initial D, un ensemble d'entités Δ_D comportant 100k entités. Ensuite, le temps d'exécution de l'algorithme incrémental est comparé à l'exécution de l'algorithme DBSCAN scalable appliqué à la totalité du jeu de données ($D \cup \Delta_D$).

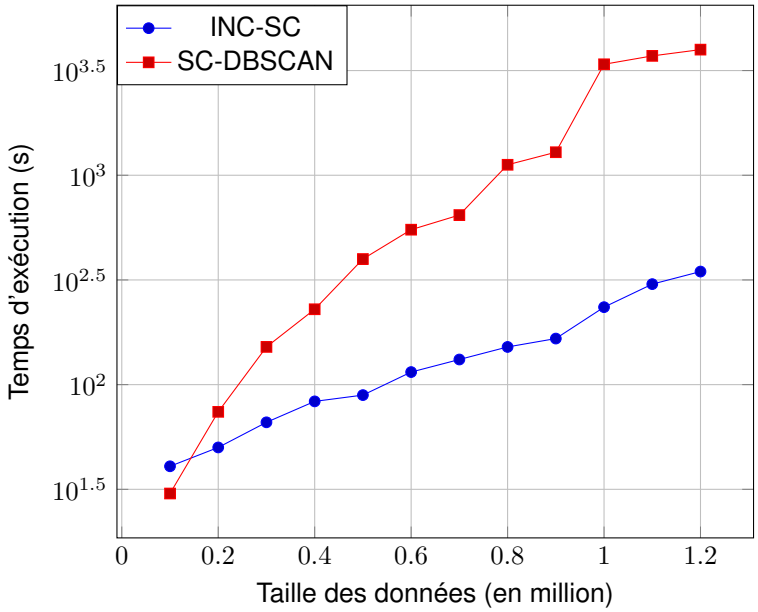


FIGURE 6.7 – Le clustering d'un sous-ensemble de DBpedia

Cette évaluation démontre que notre algorithme incrémental surpasse l'algorithme scalable en termes de performances. De plus, les patterns extraits à partir de Dbpedia sont décrits par un grand nombre de propriétés, certaines entités peuvent avoir plus de 600 propriétés. Par conséquent, l'algorithme scalable crée des $chunks$ de grande taille, ce qui impacte négativement les performances de l'algorithme car il atteint les limites de calcul du cluster lors de la recherche de l'$\epsilon-$voisinage des entités, comme nous pouvons le remarquer sur le jeu de données comportant 1 million d'entités. Cependant, l'algorithme incrémental n'est pas impacté par les entités ayant un grand nombre de propriétés car il ne gère à chaque clustering qu'un sous-ensemble du jeu de données comportant un nombre limité d'entités et calcule seulement l'$\epsilon-$voisinage des nouvelles entités.

6.5 Évaluation de la qualité du schéma

Comme expliqué dans les chapitres 3 et 4 qui détaillent nos approches de découverte et d'évolution du schéma, le clustering des données en utilisant ces approches produit le même résultat que le clustering des données en utilisant l'algorithme DBSCAN séquentiel appliqué sur toutes les entités en une seule exécution. Dans le chapitre 3, nous avons démontré que notre méthode de distribution des données garantit la comparaison de toutes les entités

voisines, et ainsi produit les mêmes clusters que l'algorithme DBSCAN. De même, dans le chapitre 4, nous avons expliqué que la modification des clusters dans le voisinage des entités impactées garantit que les clusters générés par notre algorithme incrémentale sont les mêmes que l'algorithme DBSCAN original [28]. Cette caractéristique de nos approches est importante car elle garantit la qualité du schéma extrait. En effet, il a été démontré dans d'autres travaux de recherche [65, 66] que DBSCAN produit un schéma de bonne qualité avec une bonne précision et un bon rappel.

Dans la même ligne que les expérimentations réalisées dans [65, 66], nous présentons dans cette section une évaluation de la qualité du schéma produit par nos approches. Nous avons pour cela considéré un jeu de données pour lequel les classes des entités sont connues, c'est-à-dire pour lequel toutes les entités possèdent une propriété définissant leur type. Nous avons utilisé ce jeu de données comme la vérité terrain à laquelle nous allons comparer le résultat de notre approche. Considérons l'ensemble $T = \{T_1, T_2, \cdots, T_m\}$, où chaque T_i représente l'ensemble d'entités ayant le type T_i. Cet ensemble de types est utilisé afin de comparer les classes produites par notre approche aux classes fournies par le jeu de données, et nous avons calculé la précision et le rappel de chaque classe découverte.

Nous avons appliqué le clustering sur les entités de DBpedia en utilisant notre algorithme sans considérer les types des entités. Nous avons fixé $MinPts$ à 1 car nous considérons qu'une entité doit avoir au moins une autre entité voisine afin de former une classe. Nous avons exécuté notre algorithme avec plusieurs valeurs de ϵ comprises entre 0.5 et 0.7. Dans le contexte de jeux de données RDF, le seuil ϵ représente le ratio des propriétés partagées requis pour que deux entités soient considérées comme voisines.

Nous avons annoté les classes découvertes avec le label du type le plus fréquent associé aux entités formant la classe.

Enfin, nous avons évalué la précision et le rappel de chaque classe. Dans notre travail, la précision et le rappel sont évalués en comparant les classes C générées par notre approche pour les entités aux types de ces entités T comme déclaré dans le jeu de données initial. La précision et le rappel sont évalués comme suit :

$$precision(C_i, T_i) = \frac{|C_i \cap T_i|}{|C_i|}$$

$$rappel(C_i, T_i) = \frac{|C_i \cap T_i|}{|T_i|}$$

Chacun des histogrammes a, b et c de la figure 6.8 montre pour une valeur donnée de ϵ à la fois la précision et le rappel.

Les résultats présentés dans la figure 6.8 montrent que notre approche est capable de détecter toutes les classes des entités contenues dans un jeu de données avec une bonne précision et un bon rappel quand la valeur de ϵ est bien définie (figure 6.8.b). Le rappel de la classe $Aircraft$ est plus bas que les autres classes car les entités dans cette classe sont décrites par des ensembles de propriétés très hétérogènes.

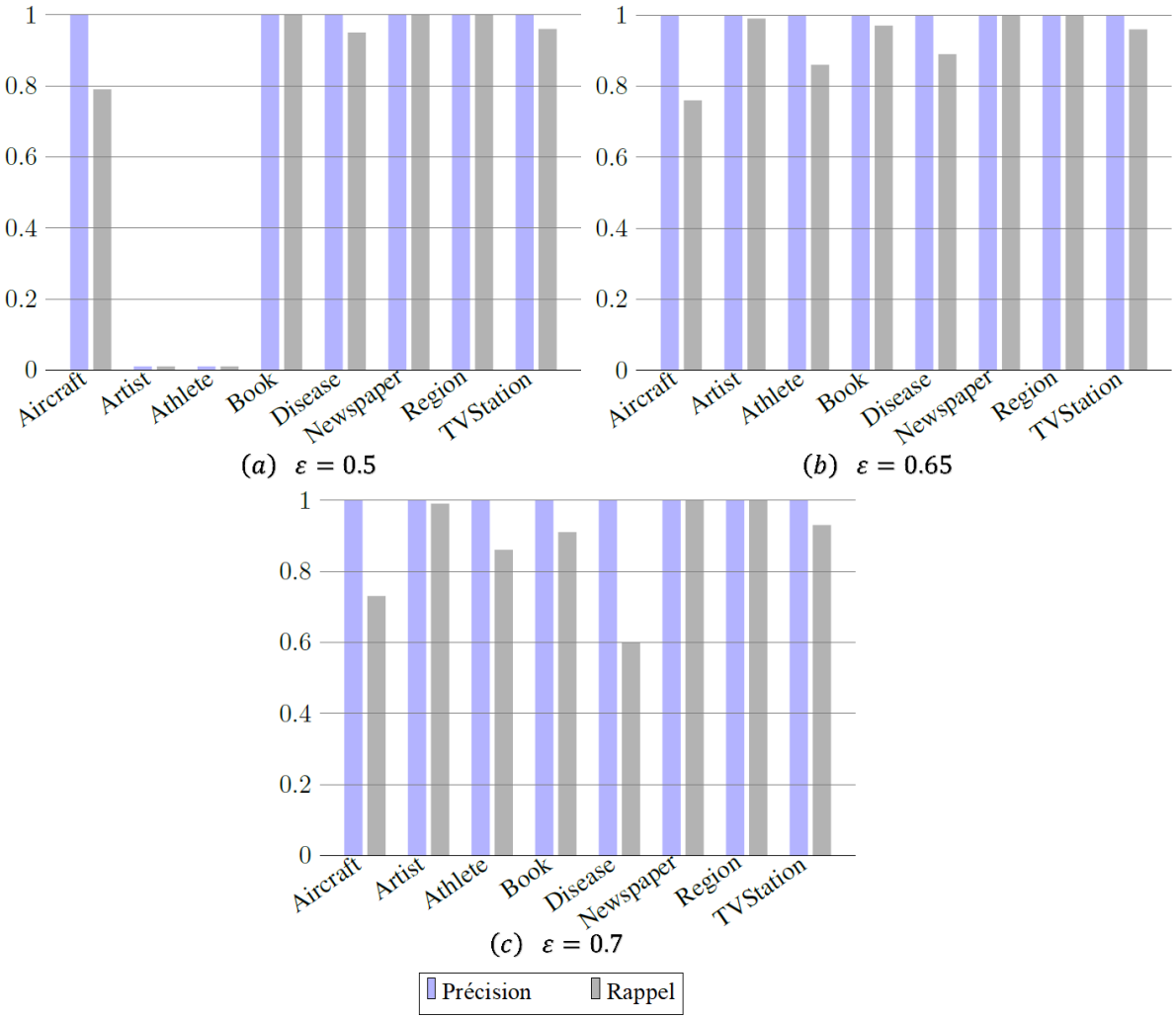


FIGURE 6.8 – La qualité des classes découvertes pour différentes valeurs de ϵ

Dans certains cas, les entités de différentes classes peuvent être décrites par des ensembles de propriétés similaires, et sont ainsi fusionnées dans une classe plus générique. Par exemple, les classes $Artist$ et $Athlete$ ont été groupées dans une classe plus générique $Person$, comme le montre la figure 6.8.a. Pour une valeur de ϵ plus élevée (figure 6.8.b), un plus grand nombre de propriétés partagées est requis afin que deux entités soient considérées comme similaires et les classes $Artist$ et $Athlete$ sont toutes les deux générées. Quand la valeur de ϵ est plus élevée, le rappel de certaines classes est plus faible (figure 6.8.c). Comme ces classes comportent des entités décrites par des ensembles de propriétés hétérogènes, elles ne sont pas considérées comme similaires et ne sont ainsi pas regroupées dans un même cluster. Une valeur plus élevée de ϵ rend l'algorithme très sensible aux petites différences dans les ensembles de propriétés décrivant les entités, ce qui peut impliquer que des entités du même type soient assignées à des clusters différents et ainsi réduire la qualité du schéma produit.

Pour conclure les expérimentations sur la qualité des classes résultantes, nous rappelons que le clustering d'un jeu de données en utilisant notre approche produit le même résultat que l'algorithme DBSCAN séquentiel.

Des travaux précédents ont démontrés que l'extraction de schéma à partir de jeux de données RDF en utilisant DBSCAN produit un résultat de bonne qualité, avec une bonne précision et un bon rappel, et détecte les classes qui n'ont pas été déclarées dans le jeu de données [66]. Ces résultats sont conformes à ceux fournis par notre approche et présentés dans cette section.

6.6 Conclusion

Notre objectif dans ce travail était de proposer des approches scalables de découverte de schéma à partir de sources de données RDF massives qui évoluent en ajoutant de nouvelles entités. Les résultats obtenus par les différentes expérimentations présentées dans ce chapitre montrent que nos propositions offrent de bonnes performances tant en termes de qualité des classes générées que de vitesse d'exécution du processus de génération de schéma. Nous avons utilisé pour cela des jeux de données synthétiques ayant des caractéristiques différentes, et des jeux de données du monde réel.

Les expérimentations montrent que notre approche de découverte de schéma produit un schéma de bonne qualité, avec une bonne précision et un bon rappel, et détecte les classes instanciées par les entités d'un jeu de données RDF. Les performances obtenues par nos algorithmes démontrent leur capacité à gérer des sources de données massives et d'extraire le schéma d'un jeu de données RDF en appliquant un clustering par densité rapide et efficace. De plus, l'environnement de développement et de test offre la possibilité d'accélérer le processus de découverte de schéma en ajoutant davantage de nœuds de calcul au cluster Spark. En effet, notre algorithme accélère et offre de meilleures performances lorsque d'avantage de nœuds de calcul sont ajoutés au cluster Spark, ce qui le rend applicable à de très grands jeux de données. Contrairement aux implémentations scalables existantes de DBSCAN, il produit le même résultat de clustering que celui produit par l'algorithme DBSCAN séquentiel. Ensuite, nous avons montré que notre algorithme de clustering scalable surpasse les algorithmes existants qui peuvent être appliqués sur les données du web.

Enfin, nous avons démontré que notre approche incrémentale de découverte du schéma qui met à jour les classes représentant les données lorsque qu'un jeu de données RDF évolue offre de meilleures performances, comparée à l'algorithme scalable qui exige la disponibilité de toutes les données pour s'exécuter.

Une perspectives intéressante à nos expérimentations est d'évaluer l'impact du nombre de clusters produits par l'algorithme de clustering scalable. En effet, la dernière étape de notre algorithme consiste à fusionner les clusters partageant une entité core. Cette opération n'est pas parallélisée et implique un coût important lorsque le nombre des clusters locaux est élevé.

La taille des $chunks$ produits par notre principe de distribution dépend de la valeur de ϵ, ainsi, une autre perspective est de conduire des expérimentations sur notre approche de découverte de schéma en considérant différentes valeurs de ϵ afin d'étudier son comportement.

Une autre amélioration possible à cette section est de comparer notre algorithme incrémentale à un autre algorithme DBSCAN évolutif. En effet, nous avons comparé les performances de notre approche incrémentale en évaluant le gain qu'elle apporte par rapport à notre approche scalable qui s'exécute sur la totalité du jeu de données. Par conséquent, il est intéressant de comparer notre algorithme à un autre qui propose une solution au même problème de l'incrémentalité de l'algorithme DBSCAN.

Chapitre 7

Conclusion

7.1 Résumé des contributions

Le web des données a émergé comme un grand espace de données qui évolue rapidement en interconnectant de plus en plus de sources de données. Cette grande quantité de données en a fait une ressource précieuse d'informations aux très nombreuses possibilités, mais l'exploitation de ces sources est difficile à cause de l'absence d'informations qui décrivent leur contenu. Différentes approches ont été proposées dans le but de découvrir le schéma décrivant les instances contenues dans une source de données RDF afin d'apporter l'aide nécessaire à leur exploitation. Cependant, les solutions proposées utilisent des algorithmes de clustering coûteux qui ne s'appliquent pas aux grandes sources de données, ou se basent sur des déclarations schéma fournies dans la source et ainsi ne s'appliquent pas lorsque ces déclarations sont absentes. Dans notre travail, nous avons apporté des contributions à la résolution du problème de la scalabilité de la découverte de schéma à partir des sources de données RDF massives, dont les déclarations schéma peuvent être incomplètes ou absentes. Nous avons également proposé une solution afin de rendre cette approche incrémentale, ce qui permet de garder le schéma cohérent avec les évolutions survenues dans les sources. Enfin, nous avons proposé une approche permettant de prendre en compte compte les triplets implicites lors de la découverte de schéma.

Nous avons d'abord proposé dans le chapitre 3 une approche automatique de découverte de schéma à partir des données RDF en analysant la structure explicite de ces instances. Notre approche est scalable et capable d'extraire les classes qui décrivent les entités à partir de sources de données massives, sur lesquelles les approches existantes ne s'appliquent pas à cause de leur complexité. Nous avons montré que l'algorithme DBscan est le plus adapté pour l'extraction de schéma à partir des données RDF, et qu'il répond mieux aux critères imposés par la nature de ces données. Cependant, son utilisation sur les jeux de données massifs est impossible en raison de sa complexité. Pour permettre l'extraction de schéma à partir de grandes sources de données, nous avons d'abord proposé une approche qui construit une représentation condensée du jeu de données initial composée d'un ensemble de patterns qui représente tous les ensembles de propriétés décrivant les entités du jeu de données. Le

but de notre approche est de permettre l'application de l'algorithme de clustering dédié à la découverte de schéma sur les grands jeux de données RDF en réduisant le nombre d'entités en entrée. Le processus de découverte de schéma est ensuite appliqué sur les patterns, ce qui permet de réduire le nombre de points à gérer et ainsi accélérer le traitement. Comme les patterns préservent toutes les structures décrivant les entités du jeu de données, la découverte de schéma appliquée aux patterns produit le même résultat que son application sur les entités initiales. Nous avons ensuite introduit une approche de découverte de schéma qui s'appuie sur un algorithme de clustering par densité, distribué et implémenté en utilisant une technologie Big Data afin d'offrir un traitement scalable aux grandes sources de données RDF. Notre algorithme de clustering permet la découverte de schéma implicite d'une source de données massive en identifiant les entités similaires et qui appartiennent à la même classe. Les classes du schéma sont ainsi construites à partir des clusters d'entités similaires. La conception distribuée de notre algorithme offre un traitement rapide des grandes sources de données RDF tout en garantissant que les clusters sont les mêmes que ceux produits par l'algorithme DBSCAN séquentiel.

Le résultat du clustering produit par notre algorithme est identique à celui calculé par l'algorithme séquentiel, ce qui permet d'offrir en plus du traitement rapide des données, un schéma de bonne qualité. Les expérimentations menées sur notre approche ont montré l'efficacité de notre proposition et la qualité du schéma produit.

Comme les sources de données RDF sont en constante évolution, le schéma décrivant une source de données RDF doit être modifié afin de maintenir sa cohérence avec les évolutions des données. Pour assurer la mise à jour du schéma suite à l'évolution des données, nous avons proposé une approche incrémentale de découverte de schéma à partir des données RDF massives. Nous avons introduit pour cela un algorithme de clustering par densité scalable et incrémental qui construit et met à jour les clusters représentant les classes du schéma. La construction incrémentale des clusters par notre approche produit le même résultat que l'utilisation de DBscan sur le jeu de données en une exécution, ce qui permet de fournir un schéma de bonne qualité. Enfin, notre proposition a été conçue afin de permettre un calcul parallèle des nouveaux clusters et un traitement rapide des grandes sources de données, comme nos différentes expérimentations l'ont démontré.

Nous nous appuyons dans notre travail sur les propriétés explicites décrivant les entités d'une source de données RDF afin de découvrir les classes composant le schéma. Cependant, la sémantique apportée dans les sources de données RDF offre la possibilité de déduire par raisonnement de nouvelles propriétés dites implicites, enrichissant par ce fait la description des entités et améliorant le processus de découverte de schéma. Dans le chapitre 5, nous avons proposé une approche de découverte de schéma qui exploite la sémantique des données RDF afin de compléter la description structurelle des entités par de nouvelles propriétés. Nous avons introduit une approche de découverte de schéma qui prend en compte les propriétés explicites et implicites décrivant les entités, en exploitant les déclarations sur le schéma fournies avec les données. Notre approche a été conçu pour effectuer un traitement parallèle et a été implémenté en utilisant une technologie big data afin d'assurer sa scalabilité aux grandes sources de données.

7.2 Perspectives

Dans cette section, nous présentons quelques travaux futures.

Notre approche de découverte de schéma permet de caractériser le contenu d'un jeu de données RDF en découvrant les classes qui représentent ses entités. Afin de compléter ce schéma, il serait intéressant de proposer une solution scalable pour la découverte des liens sémantiques et hiérarchiques entre les classes du schéma. Le schéma peut être également complété en représentant les contraintes qui s'appliquent sur les entités comme les restrictions de valeur que peut prendre une propriété, les types disjoints, etc.

Il serait aussi intéressant de généraliser nos solutions pour la découverte de schéma sur d'autres types de données graphe dont les entités sont irrégulières et ne possèdent pas une structure prédéfinie, comme les jeux de données *XML*, *JSON*, etc.

Une amélioration possible à notre approche de découverte de schéma serait de pondérer les poids des propriétés en fonction de leur importance dans la description des entités afin d'améliorer la qualité du schéma. En effet, les entités peuvent être décrites par des propriétés spécifiques à une classe donnée, et des propriétés qui ne décrivent pas spécifiquement une classe précise et peuvent être utilisées dans la description de différentes classes. Par exemple, la propriété $location$ est spécifique à un lieu tandis que la propriété $name$ peut décrire, des objets, des être humains, etc. Dans ce but, on pourrait affiner la mesure de similarité afin de considérer les propriétés par rapport à leur spécificité dans la description d'une classe, ou bien d'étudier d'autres façons d'évaluer la similarité entre les entités.

Dans notre travail, nous avons proposé une approche de découverte de schéma scalable et incrémentale qui fonctionne sur des sources de données disponibles en local et qui n'imposent pas de restrictions d'accès. Une perspective intéressante serait de proposer une approche qui découvre le schéma des sources de données distantes où l'utilisateur ne peut pas parcourir les données et le seul accès dont il dispose se fait via des requêtes envoyées au serveur Web qui gère la source de données. En effet, les sources distantes mettent en place des contraintes d'accès afin d'assurer que tout le monde puisse interroger équitablement les données du serveur et également pour se protéger des requêtes mal écrites et des robots. Une approche de découverte de schéma à partir des sources de données distantes doit tenir compte de ces restrictions.

L'approche incrémentale de découverte de schéma proposée met à jour le schéma extrait uniquement suite à des insertions de nouvelles entités au jeu de données initial. Il serait intéressant d'étendre notre approche afin de gérer les suppressions et les modifications d'entités. Notre approche de découverte de schéma s'appuie sur un algorithme de clustering basé sur la densité, où les clusters représentent les classes du schéma. Ainsi, la suppression d'entités peut impliquer la suppression de clusters existants et changer le schéma extrait. De plus, les entités peuvent être modifiées en ajoutant ou en supprimant des propriétés les décrivant, par conséquent, leurs similarités avec les autres entités doivent être réévaluées et les clusters modifiés en considérant les nouvelles distances.

Nous avons proposé dans ce travail une première contribution vers une approche hybride de découverte de schéma qui intègre certaines déclarations. Une perspective intéressante serait d'utiliser toutes les déclarations liées au schéma dans le processus de découverte de schéma. Par exemple, la prise en compte du schéma existant dans une source de données RDF, les classes déjà définies, ainsi que la considération des entités ayant une déclaration $rdf{:}type$ commune.

Liste des publications

- **[BDA2021]** Bouhamoum, R., Kedad, Z., Lopes, S. : Incremental Schema Discovery at Scale for RDF Data. Publié dans la 37ème Conférence sur la Gestion de Données – Principes, Technologies et Applications. Le 25-28 octobre 2021, en ligne et Ecole Normale Superieure de Paris (Session papiers publiés, BDA 2021).
- **[ESWC2021]** Bouhamoum R., Kedad Z., Lopes S. (2021) Incremental Schema Discovery at Scale for RDF Data. In : Verborgh R. et al. (eds) The Semantic Web. ESWC 2021. Lecture Notes in Computer Science, vol 12731. Springer, Cham. `https://doi.org/10.1007/978-3-030-77385-4_12`.
- **[TLDKS2020]** Bouhamoum R., Kedad Z., Lopes S. (2020) Scalable Schema Discovery for RDF Data. In : Hameurlain A., Tjoa A.M. (eds) Transactions on Large-Scale Data- and Knowledge-Centered Systems XLVI. Lecture Notes in Computer Science, vol 12410. Springer, Berlin, Heidelberg. `https://doi.org/10.1007/978-3-662-62386-2_4`.
- **[Journée thématique EGC & IA 2019]** Bouhamoum, R., Kedad, Z., Lopes, S. : Scalable Schema Discovery for RDF Data. Journée thématique EGC & IA : découverte de connaissances dans le Web de données. Le 10 mai 2019 à Paris Saclay, France. `https://jtegcafia.sciencesconf.org/resource/page/id/5`.
- **[AWD2019]** Bouhamoum, R., Kedad, Z., Lopes, S. : Extraction automatique de schéma pour des données massives. Atelier Web des Données (AWD 2019), Atelier de la conférence EGC. Le 22 janvier 2019 à Metz, France. `https://awd.ls2n.fr/wp-content/uploads/sites/55/2019/01/AWD_2018_paper_4.pdf`.
- **[BDCSIntell2018]** Bouhamoum, R., Kedad, Z., Lopes, S. : Schema discovery in large web datasources. In : Hojeij, M., Finance, B., Taher, Y., Zeitouni, K., Haque, R., Dbouk,M. (eds.) Proceedings of the 1st International Conference on Big Data andCyber-Security Intelligence, BDCSIntell 2018, Hadath, Lebanon, December 13-15,2018. CEUR Workshop Proceedings, vol. 2343, pp. 67–74. CEUR-WS.org (2018),`http://ceur-ws.org/Vol-2343/paper14.pdf`.
- **[ICDEW2018]** Bouhamoum, R., Kellou-Menouer, K., Lopes, S. and Kedad Z. : "Scaling Up Schema Discovery for RDF Datasets," 2018 IEEE 34th International Conference on Data Engineering Workshops (ICDEW), 2018, pp. 84-89, doi : 10.1109/ICDEW.2018.00021.

Bibliographie

[1] Z. Abedjan, T. Grütze, A. Jentzsch, and F. Naumann. Profiling and mining RDF data with prolod++. In I. F. Cruz, E. Ferrari, Y. Tao, E. Bertino, and G. Trajcevski, editors, *IEEE 30th International Conference on Data Engineering, Chicago, ICDE 2014, IL, USA, March 31 - April 4, 2014*, pages 1198–1201. IEEE Computer Society, 2014. doi : 10.1109/ICDE.2014.6816740. URL https://doi.org/10.1109/ICDE.2014.6816740.

[2] S. Abiteboul, M. Arenas, P. Barceló, M. Bienvenu, D. Calvanese, C. David, R. Hull, E. Hüllermeier, B. Kimelfeld, L. Libkin, W. Martens, T. Milo, F. Murlak, F. Neven, M. Ortiz, T. Schwentick, J. Stoyanovich, J. Su, D. Suciu, V. Vianu, and K. Yi. Research directions for principles of data management (abridged). *SIGMOD Rec.*, 45 (4) :5–17, May 2017. ISSN 0163-5808. doi : 10.1145/3092931.3092933. URL https://doi.org/10.1145/3092931.3092933.

[3] S. Abiteboul, M. Arenas, P. Barceló, M. Bienvenu, D. Calvanese, C. David, R. Hull, E. Hüllermeier, B. Kimelfeld, L. Libkin, W. Martens, T. Milo, F. Murlak, F. Neven, M. Ortiz, T. Schwentick, J. Stoyanovich, J. Su, D. Suciu, V. Vianu, and K. Yi. Research Directions for Principles of Data Management (Dagstuhl Perspectives Workshop 16151). *Dagstuhl Manifestos*, 7(1) :1–29, 2018. ISSN 2193-2433.

[4] R. Agrawal and R. Srikant. Fast algorithms for mining association rules in large databases. In J. B. Bocca, M. Jarke, and C. Zaniolo, editors, *VLDB'94, Proceedings of 20th International Conference on Very Large Data Bases, September 12-15, 1994, Santiago de Chile, Chile*, pages 487–499. Morgan Kaufmann, 1994. URL http://www.vldb.org/conf/1994/P487.PDF.

[5] C. Alcalde and A. Burusco. Study of the relevance of objects and attributes of l-fuzzy contexts using overlap indexes. In J. Medina, M. Ojeda-Aciego, J. L. Verdegay, D. A. Pelta, I. P. Cabrera, B. Bouchon-Meunier, and R. R. Yager, editors, *Information Processing and Management of Uncertainty in Knowledge-Based Systems. Theory and Foundations*, pages 537–548, Cham, 2018. Springer International Publishing.

[6] M. Ankerst, M. M. Breunig, H. Kriegel, and J. Sander. OPTICS : ordering points to identify the clustering structure. In A. Delis, C. Faloutsos, and S. Ghandeharizadeh, editors, *SIGMOD 1999, Proceedings ACM SIGMOD*

International Conference on Management of Data, June 1-3, 1999, Philadelphia, Pennsylvania, USA, pages 49–60. ACM Press, 1999. doi : 10.1145/304182.304187. URL https://doi.org/10.1145/304182.304187.

[7] S. Auer, C. Bizer, G. Kobilarov, J. Lehmann, R. Cyganiak, and Z. Ives. Dbpedia : A nucleus for a web of open data the semantic web. In *In Proceeding of the 6th International Semantic Web Conference (ISWC)*, pages 722–735. Springer Berlin Heidelberg, 2007.

[8] M.-A. Baazizi, H. B. Lahmar, D. Colazzo, G. Ghelli, and C. Sartiani. Schema inference for massive json datasets. In *In proceeding of the 20th Inter-national Conference on Extending Database Technology (EDBT)*, pages 222–233, 2017.

[9] M.-A. Baazizi, H. B. Lahmar, D. Colazzo, G. Ghelli, and C. Sartiani. Parametric schema inference for massive json datasets. In *The 28th VLDB Journal*, pages 497–521, 2019.

[10] F. Belghaouti, A. Bouzeghoub, Z. Kazi-Aoul, and R. Chiky. Fregrapad : Frequent rdf graph patterns detection for semantic data streams. In *proceeding of the 10th International Conference on Research Challenges in Information Science (RCIS)*, pages 1–9. IEEE, 2016.

[11] T. Berners-Lee and M. Fischetti. *Weaving the web - the original design and ultimate destiny of the World Wide Web by its inventor*. HarperBusiness, 2000. ISBN 978-0-06-251587-2.

[12] R. Biswas, R. Sofronova, M. Alam, N. Heist, H. Paulheim, and H. Sack. Do judge an entity by its name ! entity typing using language models. In *18th Extended Semantic Web Conference (ESWC), Poster and Demo Track*. Springer, 2021.

[13] T. Bosch, E. Acar, A. Nolle, and K. Eckert. The role of reasoning for RDF validation. In A. Polleres, T. Pellegrini, S. Hellmann, and J. X. Parreira, editors, *Proceedings of the 11th International Conference on Semantic Systems, SEMANTICS 2015, Vienna, Austria, September 15-17, 2015*, pages 33–40. ACM, 2015. doi : 10.1145/2814864.2814867. URL https://doi.org/10.1145/2814864.2814867.

[14] D. Brosius and S. Staab. Linked data querying through fca-basedschema indexing. In *5th International Workshop "What can FCA do for Artificial Intelligence" ? co-located with the European Conference on Artificial Intelligence, (FCA4AI@ECAI 2016)*, pages 63–68, 2016.

[15] S. Campinas, T. E. Perry, D. Ceccarelli, R. Delbru, and G. Tummarello. Introducing rdf graph summary with application to assisted sparql formulation. In *23rd International Workshop on Database and Expert Systems Applications*, pages 261–266, 2012. doi : 10.1109/DEXA.2012.38.

[16] S. Cebiric, F. Goasdoué, and I. Manolescu. Query-oriented summarization of RDF graphs. *Proc. VLDB Endow.*, 8(12) :2012–2015, 2015. doi : 10.14778/2824032.2824124. URL http://www.vldb.org/pvldb/vol8/p2012-cebiric.pdf.

[17] Y. Cetin and O. Abul. Distributed RDFS reasoning with mapreduce. In T. Czachórski, E. Gelenbe, and R. Lent, editors, *Information Sciences and Systems 2014 - Proceedings of the 29th International Symposium on Computer and Information Sciences, ISCIS 2014, Krakow, Poland, October 27-28, 2014*, pages 305–313. Springer, 2014. doi : 10.1007/978-3-319-09465-6_32. URL https://doi.org/10.1007/978-3-319-09465-6_32.

[18] S. Chakraborty and N. K. Nagwani. Analysis and study of incremental DBSCAN clustering algorithm. *CoRR*, abs/1406.4754, 2014. URL http://arxiv.org/abs/1406.4754.

[19] J. X. Chen and M. Z. Reformat. Learning categories from linked open data. In *proceeding of the 15th International Conference on Information Processing and Management of Uncertainty in Knowledge-Based Systems (IPMU)*, pages 396–405. Springer International Publishing, 2014.

[20] K. Christodoulou, N. W. Paton, and A. A. A. Fernandes. Structure inference for linked data sources using clustering. In *Proceedings of the Joint EDBT/ICDT 2013 Workshops*, EDBT '13, page 60–67, New York, NY, USA, 2013. Association for Computing Machinery. ISBN 9781450315999. doi : 10.1145/2457317.2457328. URL https://doi.org/10.1145/2457317.2457328.

[21] K. Christodoulou, N. W. Paton, and A. A. A. Fernandes. Structure inference for linked data sources using clustering. *Trans. Large Scale Data Knowl. Centered Syst.*, 19 :1–25, 2015. doi : 10.1007/978-3-662-46562-2_1. URL https://doi.org/10.1007/978-3-662-46562-2_1.

[22] I. Cordova and T. Moh. DBSCAN on resilient distributed datasets. In *2015 International Conference on High Performance Computing & Simulation, HPCS 2015, Amsterdam, Netherlands, July 20-24, 2015*, pages 531–540. IEEE, 2015. doi : 10.1109/HPCSim.2015.7237086. URL https://doi.org/10.1109/HPCSim.2015.7237086.

[23] T. P. Cronan and T. L. Means. System development : An empirical study of user communication. *Data Base*, 15(3) :25–33, 1984. doi : 10.1145/1040681.1040685. URL https://doi.org/10.1145/1040681.1040685.

[24] J. Dean and S. Ghemawat. Mapreduce : Simplified data processing on large clusters. In E. A. Brewer and P. Chen, editors, *6th Symposium on Operating System Design and Implementation (OSDI 2004), San Francisco, California, USA, December 6-8, 2004*, pages 137–150. USENIX Association, 2004. URL http://www.usenix.org/events/osdi04/tech/dean.html.

[25] J. Dean and S. Ghemawat. Mapreduce : simplified data processing on large clusters. *Commun. ACM*, 51(1) : 107–113, 2008. doi : 10.1145/1327452.1327492. URL http://doi.acm.org/10.1145/1327452.1327492.

[26] A. P. Dempster, N. M. Laird, and D. B. Rubin. Maximum likelihood from incomplete data via the em algorithm. *Journal of the Royal Statistical Society. Series B (Methodological)*, 39(1) :1–38, 1977. URL www.jstor.org/stable/2984875.

[27] M. B. Ellefi, Z. Bellahsene, J. G. Breslin, E. Demidova, S. Dietze, J. Szymanski, and K. Todorov. RDF dataset profiling - a survey of features, methods, vocabularies and applications. *Semantic Web*, 9(5) :677–705, 2018. doi : 10.3233/SW-180294. URL https://doi.org/10.3233/SW-180294.

[28] M. Ester, H.-P. Kriegel, J. Sander, and X. Xu. A density-based algorithm for discovering clusters in large spatial databases with noise. In *In proceeding of the Second International Conference on Knowledge Discovery and Data Mining (KDD)*, pages 226–231. AAAI Press, 1996.

[29] M. Ester, H. Kriegel, J. Sander, M. Wimmer, and X. Xu. Incremental clustering for mining in a data warehousing environment. In A. Gupta, O. Shmueli, and J. Widom, editors, *VLDB'98, Proceedings of 24rd International Conference on Very Large Data Bases, August 24-27, 1998, New York City, New York, USA*, pages 323–333. Morgan Kaufmann, 1998. URL http://www.vldb.org/conf/1998/p323.pdf.

[30] L. Fang, Q. Miao, and Y. Meng. Dbpedia entity type inference using categories. In T. Kawamura and H. Paulheim, editors, *Proceedings of the ISWC 2016 Posters & Demonstrations Track co-located with 15th International Semantic Web Conference (ISWC 2016), Kobe, Japan, October 19, 2016*, volume 1690 of *CEUR Workshop Proceedings*. CEUR-WS.org, 2016. URL http://ceur-ws.org/Vol-1690/paper107.pdf.

[31] L. Fang, Q. Miao, and Y. Meng. Dbpedia entity type inference using categories. In T. Kawamura and H. Paulheim, editors, *Proceedings of the ISWC 2016 Posters & Demonstrations Track co-located with 15th International Semantic Web Conference (ISWC 2016), Kobe, Japan, October 19, 2016*, volume 1690 of *CEUR Workshop Proceedings*. CEUR-WS.org, 2016. URL http://ceur-ws.org/Vol-1690/paper107.pdf.

[32] M. A. Farvardin, D. Colazzo, K. Belhajjame, and C. Sartiani. Streaming saturation for large RDF graphs with dynamic schema information. In A. Cheung and K. Nguyen, editors, *Proceedings of the 17th ACM SIGPLAN International Symposium on Database Programming Languages, DBPL 2019, Phoenix, AZ, USA, June 23, 2019*, pages 42–52. ACM, 2019. doi : 10.1145/3315507.3330201. URL https://doi.org/10.1145/3315507.3330201.

[33] D. H. Fisher. Knowledge acquisition via incremental conceptual clustering. *Mach. Learn.*, 2(2) :139–172, 1987. doi : 10.1007/BF00114265. URL https://doi.org/10.1007/BF00114265.

[34] F. Florenzano, D. Parra, J. L. Reutter, and F. Venegas. An interactive visualisation for RDF data. In T. Kawamura and H. Paulheim, editors, *Proceedings of the ISWC 2016 Posters & Demonstrations Track co-located with 15th International Semantic Web Conference (ISWC 2016), Kobe, Japan, October 19, 2016*, volume 1690 of *CEUR Workshop Proceedings*. CEUR-WS.org, 2016. URL http://ceur-ws.org/Vol-1690/paper84.pdf.

[35] I. C. for Data Analytics. The Linked Open Data Cloud. https://lod-cloud.net/, 2021. Accessed : 2021-08-01.

[36] J. H. Friedman, F. Baskett, and L. J. Shustek. An algorithm for finding nearest neighbors. *IEEE Trans. Computers*, 24(10) :1000–1006, 1975. doi : 10.1109/T-C.1975.224110. URL https://doi.org/10.1109/T-C.1975.224110.

[37] H. Fuchs, Z. M. Kedem, and B. F. Naylor :. On visible surface generation by a priori tree structures. In *Proceedings of the 7th Annual Conference on Computer Graphics and Interactive Techniques (SIGGRAPH)*, page 124–133. ACM Press, 1980.

[38] F. Gandon. Le web sémantique n'est pas antisocial. In M. Harzallah, J. Charlet, N. Aussenac-Gilles, and M. Lewkowicz, editors, *IC 2006 : Ingénierie des connaissances 2006 (Proceedings of the 17th French Knowledge Engineering Conference), Nantes, France, June 26-30, 2006*, pages 131–140. Université de Nantes, 2006. URL http://www.irit.fr/GRACQ/article.php3?id_article=216.

[39] R. Goldman and J. Widom. Dataguides : Enabling query formulation and optimization in semistructured databases. In M. Jarke, M. J. Carey, K. R. Dittrich, F. H. Lochovsky, P. Loucopoulos, and M. A. Jeusfeld, editors, *VLDB'97, Proceedings of 23rd International Conference on Very Large Data Bases, August 25-29, 1997, Athens, Greece*, pages 436–445. Morgan Kaufmann, 1997. URL http://www.vldb.org/conf/1997/P436.PDF.

[40] Y. Gong, L. Morandini, and R. O. Sinnott. The design and benchmarking of a cloud-based platform for processing and visualization of traffic data. In *Proceeding of the IEEE International Conference on Big Data and Smart Computing (BigComp)*, pages 13–20, 2017.

[41] Y. Gong, R. O. Sinnott, and P. Rimba. RT-DBSCAN : real-time parallel clustering of spatio-temporal data using spark-streaming. In Y. Shi, H. Fu, Y. Tian, V. V. Krzhizhanovskaya, M. H. Lees, J. J. Dongarra, and P. M. A. Sloot, editors, *Computational Science - ICCS 2018 - 18th International Conference, Wuxi, China, June 11-13, 2018, Proceedings, Part I*, volume 10860 of *Lecture Notes in Computer Science*, pages 524–539. Springer, 2018. doi : 10.1007/978-3-319-93698-7_40. URL https://doi.org/10.1007/978-3-319-93698-7_40.

[42] M. Götz, C. Bodenstein, and M. Riedel. Hpdbscan : Highly parallel dbscan. MLHPC '15, New York, NY, USA, 2015. Association for Computing Machinery. ISBN 9781450340069. doi : 10.1145/2834892.2834894. URL https://doi.org/10.1145/2834892.2834894.

[43] A. Gragera Aguaza and V. Suppakitpaisarn. Relaxed triangle inequality ratio of the sørensen–dice and tversky indexes. *Theoretical Computer Science*, 718, 01 2017.

[44] R. Gu, S. Wang, F. Wang, C. Yuan, and Y. Huang. Cichlid : Efficient large scale RDFS/OWL reasoning with spark. In *2015 IEEE International Parallel and Distributed Processing Symposium, IPDPS 2015, Hyderabad,*

India, May 25-29, 2015, pages 700–709. IEEE Computer Society, 2015. doi : 10.1109/IPDPS.2015.14. URL https://doi.org/10.1109/IPDPS.2015.14.

[45] S. Habib, V. Morozov, H. Finkel, A. Pope, K. Heitmann, K. Kumaran, T. Peterka, J. Insley, D. Daniel, P. Fasel, N. Frontiere, and Z. Lukic. The universe at extreme scale : Multi-petaflop sky simulation on the bg/q. *International Conference for High Performance Computing, Networking, Storage and Analysis, SC*, 11 2012. doi : 10.1109/SC.2012.106.

[46] M. Halkidi, Y. Batistakis, and M. Vazirgiannis. On clustering validation techniques. *J. Intell. Inf. Syst.*, 17(2-3) : 107–145, 2001. doi : 10.1023/A:1012801612483. URL https://doi.org/10.1023/A:1012801612483.

[47] M. Halkidi, Y. Batistakis, and M. Vazirgiannis. On clustering validation techniques. *J. Intell. Inf. Syst.*, 17(2-3) : 107–145, 2001. doi : 10.1023/A:1012801612483. URL https://doi.org/10.1023/A:1012801612483.

[48] D. Han, A. Agrawal, W. Liao, and A. N. Choudhary. A novel scalable DBSCAN algorithm with spark. In *2016 IEEE International Parallel and Distributed Processing Symposium Workshops, IPDPS Workshops 2016, Chicago, IL, USA, May 23-27, 2016*, pages 1393–1402. IEEE Computer Society, 2016. doi : 10.1109/IPDPSW.2016.57. URL https://doi.org/10.1109/IPDPSW.2016.57.

[49] J. Han and M. Kamber. *Data Mining : Concepts and Techniques, Second Edition*. The Morgan Kaufmann series in data management systems. Elsevier, 2006. ISBN 978-1-55860-901-3.

[50] Y. He, H. Tan, W. Luo, H. Mao, D. Ma, S. Feng, and J. Fan. MR-DBSCAN : an efficient parallel density-based clustering algorithm using mapreduce. In *17th IEEE International Conference on Parallel and Distributed Systems, ICPADS 2011, Tainan, Taiwan, December 7-9, 2011*, pages 473–480. IEEE Computer Society, 2011. doi : 10.1109/ICPADS.2011.83. URL https://doi.org/10.1109/ICPADS.2011.83.

[51] Y. He, H. Tan, W. Luo, S. Feng, and J. Fan. MR-DBSCAN : a scalable mapreduce-based DBSCAN algorithm for heavily skewed data. *Frontiers Comput. Sci.*, 8(1) :83–99, 2014. doi : 10.1007/s11704-013-3158-3. URL https://doi.org/10.1007/s11704-013-3158-3.

[52] N. Heino and J. Z. Pan. RDFS reasoning on massively parallel hardware. In P. Cudré-Mauroux, J. Heflin, E. Sirin, T. Tudorache, J. Euzenat, M. Hauswirth, J. X. Parreira, J. Hendler, G. Schreiber, A. Bernstein, and E. Blomqvist, editors, *The Semantic Web - ISWC 2012 - 11th International Semantic Web Conference, Boston, MA, USA, November 11-15, 2012, Proceedings, Part I*, volume 7649 of *Lecture Notes in Computer Science*, pages 133–148. Springer, 2012. doi : 10.1007/978-3-642-35176-1_9. URL https://doi.org/10.1007/978-3-642-35176-1_9.

[53] A. Hinneburg and H. Gabriel. DENCLUE 2.0 : Fast clustering based on kernel density estimation. In M. R. Berthold, J. Shawe-Taylor, and N. Lavrac, editors, *Advances in Intelligent Data Analysis VII, 7th In-*

ternational Symposium on Intelligent Data Analysis, IDA 2007, Ljubljana, Slovenia, September 6-8, 2007, Proceedings, volume 4723 of *Lecture Notes in Computer Science*, pages 70–80. Springer, 2007. doi : 10.1007/978-3-540-74825-0_7. URL https://doi.org/10.1007/978-3-540-74825-0_7.

[54] A. Hogan, A. Harth, and A. Polleres. Scalable authoritative OWL reasoning for the web. *Int. J. Semantic Web Inf. Syst.*, 5(2) :49–90, 2009. doi : 10.4018/jswis.2009040103. URL https://doi.org/10.4018/jswis.2009040103.

[55] IBM. Ibm quest synthetic data generator. https://sourceforge.net/projects/ibmquestdatagen/, 2015. Accessed : 2018-10-01.

[56] P. Jaccard. The distribution of flora in the alpine zone. *New Phytologist*, 11(2) :37–50, 1912.

[57] B. Jagvaral and Y. Park. Distributed scalable RDFS reasoning. In *2015 International Conference on Big Data and Smart Computing, BIGCOMP 2015, Jeju, South Korea, February 9-11, 2015*, pages 31–34. IEEE Computer Society, 2015. doi : 10.1109/35021BIGCOMP.2015.7072845. URL https://doi.org/10.1109/35021BIGCOMP.2015.7072845.

[58] A. K. Jain, M. N. Murty, and P. J. Flynn. Data clustering : A review. *ACM Comput. Surv.*, 31(3) :264–323, 1999. doi : 10.1145/331499.331504. URL https://doi.org/10.1145/331499.331504.

[59] B. Jang and Y. Ha. Transitivity reasoning for RDF ontology with iterative mapreduce. In L. Barolli, I. You, F. Xhafa, F. Leu, and H. Chen, editors, *Seventh International Conference on Innovative Mobile and Internet Services in Ubiquitous Computing, IMIS 2013, Taichung, Taiwan, July 3-5, 2013*, pages 232–237. IEEE Computer Society, 2013. doi : 10.1109/IMIS.2013.47. URL https://doi.org/10.1109/IMIS.2013.47.

[60] D. Janke, S. Staab, and M. Thimm. Koral : A glass box profiling system for individual components of distributed RDF stores. In R. Usbeck, A. N. Ngomo, J. Kim, K. Choi, P. Cimiano, I. Fundulaki, and A. Krithara, editors, *Joint Proceedings of BLINK2017 : 2nd International Workshop on Benchmarking Linked Data and NLIWoD3 : Natural Language Interfaces for the Web of Data co-located with 16th International Semantic Web Conference (ISWC 2017), Vienna, Austria, October 21st - to - 22nd, 2017*, volume 1932 of *CEUR Workshop Proceedings*. CEUR-WS.org, 2017. URL http://ceur-ws.org/Vol-1932/paper-05.pdf.

[61] H. Jin, L. Hou, J. Li, and T. Dong. Attributed and predictive entity embedding for fine-grained entity typing in knowledge bases. In E. M. Bender, L. Derczynski, and P. Isabelle, editors, *Proceedings of the 27th International Conference on Computational Linguistics, COLING 2018, Santa Fe, New Mexico, USA, August 20-26, 2018*, pages 282–292. Association for Computational Linguistics, 2018. URL https://www.aclweb.org/anthology/C18-1024/.

[62] H. Jin, L. Hou, J. Li, and T. Dong. Fine-grained entity typing via hierarchical multi graph convolutional networks. In K. Inui, J. Jiang, V. Ng, and X. Wan, editors, *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing, EMNLP-IJCNLP 2019, Hong Kong, China, November 3-7, 2019*, pages 4968–4977. Association for Computational Linguistics, 2019. doi : 10.18653/v1/D19-1502. URL https://doi.org/10.18653/v1/D19-1502.

[63] L. Kaufmann and P. Rousseeuw. Clustering by means of medoids. *Data Analysis based on the L1-Norm and Related Methods*, pages 405–416, 01 1987.

[64] K. Kellou-Menouer. *Découverte de schéma pour les données du Web sémantique. (Schema Discovery in Semantic Web Data Sources)*. PhD thesis, University of Paris-Saclay, France, 2017. URL https://tel.archives-ouvertes.fr/tel-01630962.

[65] K. Kellou-Menouer and Z. Kedad. Schema discovery in RDF data sources. In *In proceeding of the 34th International Conference on Conceptual Modeling (ER)*, pages 481–495. Springer International Publishing, 2015.

[66] K. Kellou-Menouer and Z. Kedad. A self-adaptive and incremental approach for data profiling in the semantic web. *Trans. Large Scale Data Knowl. Centered Syst.*, 29 :108–133, 2016. doi : 10.1007/978-3-662-54037-4_4. URL https://doi.org/10.1007/978-3-662-54037-4_4.

[67] K. Kellou-Menouer and Z. Kedad. On-line versioned schema inference for large semantic web data sources. In *Proceedings of the 29th International Conference on Scientific and Statistical Database Management, Chicago, IL, USA, June 27-29, 2017*, pages 9 :1–9 :12. ACM, 2017. doi : 10.1145/3085504.3085513. URL https://doi.org/10.1145/3085504.3085513.

[68] K. Kellou-Menouer and Z. Kedad. Schemadecrypt++ : Parallel on-line versioned schema inference for large semantic web data sources. *Inf. Syst.*, 93 :101551, 2020. doi : 10.1016/j.is.2020.101551. URL https://doi.org/10.1016/j.is.2020.101551.

[69] M. Kirchberg, E. Leonardi, Y. S. Tan, and S. Link. Formal concept discovery in semantic web data. In *proceeding of the 10th International Conference on Formal Concept Analysis (ICFCA)*, pages 164–179. Springer International Publishing, 2012.

[70] C. Liu, G. Qi, H. Wang, and Y. Yu. Large scale fuzzy pd * reasoning using mapreduce. In L. Aroyo, C. Welty, H. Alani, J. Taylor, A. Bernstein, L. Kagal, N. F. Noy, and E. Blomqvist, editors, *The Semantic Web - ISWC 2011 - 10th International Semantic Web Conference, Bonn, Germany, October 23-27, 2011, Proceedings, Part I*, volume 7031 of *Lecture Notes in Computer Science*, pages 405–420. Springer, 2011. doi : 10.1007/978-3-642-25073-6_26. URL https://doi.org/10.1007/978-3-642-25073-6_26.

[71] S. Lloyd. Least squares quantization in pcm. *IEEE Transactions on Information Theory*, 28(2) :129–137, 1982. doi : 10.1109/TIT.1982.1056489.

[72] A. Lulli, M. Dell'Amico, P. Michiardi, and L. Ricci. Ng-dbscan : Scalable density-based clustering for arbitrary data. *Proc. VLDB Endow.*, 10(3) :157–168, Nov. 2016. ISSN 2150-8097. doi : 10.14778/3021924.3021932. URL https://doi.org/10.14778/3021924.3021932.

[73] G. Luo, X. Luo, T. F. Gooch, L. Tian, and K. Qin. A parallel DBSCAN algorithm based on spark. In Z. Cai, R. A. Angryk, W. Song, Y. Li, X. Cao, A. G. Bourgeois, G. Luo, L. Cheng, and B. Krishnamachari, editors, *2016 IEEE International Conferences on Big Data and Cloud Computing (BDCloud), Social Computing and Networking (SocialCom), Sustainable Computing and Communications (SustainCom), BDCloud-SocialCom-SustainCom 2016, Atlanta, GA, USA, October 8-10, 2016*, pages 548–553. IEEE Computer Society, 2016. doi : 10.1109/BDCloud-SocialCom-SustainCom.2016.85. URL https://doi.org/10.1109/BDCloud-SocialCom-SustainCom.2016.85.

[74] F. M. Suchanek, G. Kasneci, and G. Weikum. Yago : a core of semantic knowledge. In *Proceedings of the 16th international conference on World Wide Web (WWW)*, pages 697–706. ACM Press, 2007.

[75] A. M.Bakr, N. M.Ghanem, and M. A.Ismail. Efficient incremental density-based algorithm for clustering large datasets. In *Alexandria Engineering Journal*, volume 54, pages 1147–1154. Elsevier B.V, 2015.

[76] N. Mihindukulasooriya, M. R. A. Rashid, G. Rizzo, R. García-Castro, O. Corcho, and M. Torchiano. Rdf shape induction using knowledge base profiling. In *Proceedings of the 33rd Annual ACM Symposium on Applied Computing*, SAC '18, page 1952–1959, New York, NY, USA, 2018. Association for Computing Machinery. ISBN 9781450351911. doi : 10.1145/3167132.3167341. URL https://doi.org/10.1145/3167132.3167341.

[77] R. Milner. *Communication and concurrency*. PHI Series in computer science. Prentice Hall, 1989. ISBN 978-0-13-115007-2.

[78] A. Neelakantan and M. Chang. Inferring missing entity type instances for knowledge base completion : New dataset and methods. In R. Mihalcea, J. Y. Chai, and A. Sarkar, editors, *NAACL HLT 2015, The 2015 Conference of the North American Chapter of the Association for Computational Linguistics : Human Language Technologies, Denver, Colorado, USA, May 31 - June 5, 2015*, pages 515–525. The Association for Computational Linguistics, 2015. doi : 10.3115/v1/n15-1054. URL https://doi.org/10.3115/v1/n15-1054.

[79] S. Nestorov, J. D. Ullman, J. L. Wiener, and S. S. Chawathe. Representative objects : Concise representations of semistructured, hierarchial data. In W. A. Gray and P. Larson, editors, *Proceedings of the Thirteenth International Conference on Data Engineering, April 7-11, 1997, Birmingham, UK*, pages 79–90. IEEE Computer Society, 1997. doi : 10.1109/ICDE.1997.581741. URL https://doi.org/10.1109/ICDE.1997.581741.

[80] S. Nestorov, S. Abiteboul, and R. Motwani. Extracting schema from semistructured data. *ACM SIGMOD Record*, pages 295—-306, 1998.

[81] S. Nestorov, S. Abiteboul, and R. Motwani. Inferring structure in semistructured data. *ACM SIGMOD Record*, 26, 06 2001. doi : 10.1145/271074.271084.

[82] A. G. Nuzzolese, A. Gangemi, V. Presutti, and P. Ciancarini. Type inference through the analysis of wikipedia links. In C. Bizer, T. Heath, T. Berners-Lee, and M. Hausenblas, editors, *WWW2012 Workshop on Linked Data on the Web, Lyon, France, 16 April, 2012*, volume 937 of *CEUR Workshop Proceedings*. CEUR-WS.org, 2012. URL http://ceur-ws.org/Vol-937/ldow2012-paper-13.pdf.

[83] Y. Papakonstantinou, H. Garcia-Molina, and J. Widom. Object exchange across heterogeneous information sources. In *Proceedings of the Eleventh International Conference on Data Engineering*, ICDE '95, page 251–260, USA, 1995. IEEE Computer Society. ISBN 0818669101.

[84] M. M. A. Patwary, D. Palsetia, A. Agrawal, W. Liao, F. Manne, and A. N. Choudhary. A new scalable parallel DBSCAN algorithm using the disjoint-set data structure. In J. K. Hollingsworth, editor, *SC Conference on High Performance Computing Networking, Storage and Analysis, SC '12, Salt Lake City, UT, USA - November 11 - 15, 2012*, page 62. IEEE/ACM, 2012. doi : 10.1109/SC.2012.9. URL https://doi.org/10.1109/SC.2012.9.

[85] H. Paulheim. Browsing linked open data with auto complete. In *Semantic Web Challenge*, 2012.

[86] H. Paulheim. Knowledge graph refinement : A survey of approaches and evaluation methods. *Semantic Web*, 8(3) :489–508, 2017. doi : 10.3233/SW-160218. URL https://doi.org/10.3233/SW-160218.

[87] H. Paulheim and C. Bizer. Type inference on noisy RDF data. In H. Alani, L. Kagal, A. Fokoue, P. Groth, C. Biemann, J. X. Parreira, L. Aroyo, N. F. Noy, C. Welty, and K. Janowicz, editors, *The Semantic Web - ISWC 2013 - 12th International Semantic Web Conference, Sydney, NSW, Australia, October 21-25, 2013, Proceedings, Part I*, volume 8218 of *Lecture Notes in Computer Science*, pages 510–525. Springer, 2013. doi : 10.1007/978-3-642-41335-3_32. URL https://doi.org/10.1007/978-3-642-41335-3_32.

[88] H. Paulheim and C. Bizer. Improving the quality of linked data using statistical distributions. *Int. J. Semantic Web Inf. Syst.*, 10(2) :63–86, 2014. doi : 10.4018/ijswis.2014040104. URL https://doi.org/10.4018/ijswis.2014040104.

[89] N. Pernelle, F. Saïs, D. Mercier, and S. Thuraisamy. Rdf data evolution : efficient detection and semantic representation of changes. In *Proceedings of the Posters and Demos Track of the International Conference on Semantic Systems - SEMANTICS*, volume 12, 2016.

[90] A. Polleres, A. Hogan, R. Delbru, and J. Umbrich. RDFS and OWL reasoning for linked data. In S. Rudolph, G. Gottlob, I. Horrocks, and F. van Harmelen, editors, *Reasoning Web. Semantic Technologies for Intelligent Data Access - 9th International Summer School 2013, Mannheim, Germany, July 30 - August 2, 2013. Proceedings*, volume 8067 of *Lecture Notes in Computer Science*, pages 91–149. Springer, 2013. doi : 10.1007/978-3-642-39784-4_2. URL https://doi.org/10.1007/978-3-642-39784-4_2.

[91] S. Pouriyeh, M. Allahyaril, G. Cheng, H. R. Arabnia, K. Kochut, and M. Atzori. R-lda : Profiling rdf datasets using knowledge-based topic modeling. In *2019 IEEE 13th International Conference on Semantic Computing (ICSC)*, pages 146–149, 2019. doi : 10.1109/ICOSC.2019.8665510.

[92] U. Priss. Formal concept analysis in information science. *Annu. Rev. Inf. Sci. Technol.*, 40(1) :521–543, 2006. doi : 10.1002/aris.1440400120. URL https://doi.org/10.1002/aris.1440400120.

[93] G. Rao, B. Zhao, X. Zhang, Z. Feng, and G. Xiao. PRSPR : an adaptive framework for massive RDF stream reasoning. In Y. Cai, Y. Ishikawa, and J. Xu, editors, *Web and Big Data - Second International Joint Conference, APWeb-WAIM 2018, Macau, China, July 23-25, 2018, Proceedings, Part I*, volume 10987 of *Lecture Notes in Computer Science*, pages 440–448. Springer, 2018. doi : 10.1007/978-3-319-96890-2_36. URL https://doi.org/10.1007/978-3-319-96890-2_36.

[94] D. S. Ruiz, S. F. Morales, and J. G. Molina. Inferring versioned schemas from nosql databases and its applications. In *In proceeding of the 34th International Conference on Conceptual Modeling (ER)*, pages 467–480. Springer International Publishing, 2015.

[95] I. K. Savvas and D. C. Tselios. Parallelizing dbscan algorithm using MPI. In S. Reddy and W. Gaaloul, editors, *25th IEEE International Conference on Enabling Technologies : Infrastructure for Collaborative Enterprises, WETICE 2016, Paris, France, June 13-15, 2016*, pages 77–82. IEEE Computer Society, 2016. doi : 10.1109/WETICE.2016.26. URL https://doi.org/10.1109/WETICE.2016.26.

[96] A. Schätzle, A. Neu, G. Lausen, and M. Przyjaciel-Zablocki. Large-scale bisimulation of RDF graphs. In R. D. Virgilio, F. Giunchiglia, and L. Tanca, editors, *Proceedings of the Fifth Workshop on Semantic Web Information Management, SWIM@SIGMOD Conference 2013, New York, NY, USA, June 23, 2013*, pages 1 :1–1 :8. ACM, 2013. doi : 10.1145/2484712.2484713. URL https://doi.org/10.1145/2484712.2484713.

[97] N. Schmitt, M. Niepert, and H. Stuckenschmidt. BRAMBLE : A web-based framework for interactive rdf-graph visualisation. In A. Polleres and H. Chen, editors, *Proceedings of the ISWC 2010 Posters & Demonstrations Track : Collected Abstracts, Shanghai, China, November 9, 2010*, volume 658 of *CEUR Workshop Proceedings*. CEUR-WS.org, 2010. URL http://ceur-ws.org/Vol-658/paper465.pdf.

[98] G. Sheikholeslami, S. Chatterjee, and A. Zhang. Wavecluster : A multi-resolution clustering approach for very large spatial databases. In A. Gupta, O. Shmueli, and J. Widom, editors, *VLDB'98, Proceedings of 24rd International Conference on Very Large Data Bases, August 24-27, 1998, New York City, New York, USA*, pages 428–439. Morgan Kaufmann, 1998. URL `http://www.vldb.org/conf/1998/p428.pdf`.

[99] M. A. Sherif, A. N. Ngomo, and J. Lehmann. Automating RDF dataset transformation and enrichment. In F. Gandon, M. Sabou, H. Sack, C. d'Amato, P. Cudré-Mauroux, and A. Zimmermann, editors, *The Semantic Web. Latest Advances and New Domains - 12th European Semantic Web Conference, ESWC 2015, Portoroz, Slovenia, May 31 - June 4, 2015. Proceedings*, volume 9088 of *Lecture Notes in Computer Science*, pages 371–387. Springer, 2015. doi : 10.1007/978-3-319-18818-8_23. URL `https://doi.org/10.1007/978-3-319-18818-8_23`.

[100] H. Song and J. Lee. RP-DBSCAN : A superfast parallel DBSCAN algorithm based on random partitioning. In G. Das, C. M. Jermaine, and P. A. Bernstein, editors, *Proceedings of the 2018 International Conference on Management of Data, SIGMOD Conference 2018, Houston, TX, USA, June 10-15, 2018*, pages 1173–1187. ACM, 2018. doi : 10.1145/3183713.3196887. URL `https://doi.org/10.1145/3183713.3196887`.

[101] I. Subhi, P. Pierre-Henri, H. Fayçal, and S.-S.-C. Samira. Revealing the conceptual schemas of rdf datasets. In *In proceeding of the 31st International Conference on Advanced Information Systems Engineering (CAiSE)*, pages 312–327. Springer International Publishing, 2019.

[102] J. Sun, Y. Xun, J. Zhang, and J. Li. Incremental frequent itemsets mining with fcfp tree. *IEEE Access*, 7 : 136511–136524, 2019. doi : 10.1109/ACCESS.2019.2943015.

[103] A. Telea, F. Frasincar, and G. Houben. Visualisation of rdf(s)-based information. In E. Banissi, K. Börner, C. Chen, G. Clapworthy, C. Maple, A. Lobben, C. J. Moore, J. C. Roberts, A. Ursyn, and J. J. Zhang, editors, *Seventh International Conference on Information Visualization, IV 2003, 16-18 July 2003, London, UK*, pages 294–299. IEEE Computer Society, 2003. doi : 10.1109/IV.2003.1217993. URL `https://doi.org/10.1109/IV.2003.1217993`.

[104] The Apache Software Foundation. Apache Spark. `https://spark.apache.org`, 2018. Accessed : 2018-10-20.

[105] J. Urbani, S. Kotoulas, E. Oren, and F. van Harmelen. Scalable distributed reasoning using mapreduce. In A. Bernstein, D. R. Karger, T. Heath, L. Feigenbaum, D. Maynard, E. Motta, and K. Thirunarayan, editors, *The Semantic Web - ISWC 2009, 8th International Semantic Web Conference, ISWC 2009, Chantilly, VA, USA, October 25-29, 2009. Proceedings*, volume 5823 of *Lecture Notes in Computer Science*, pages 634–649. Springer, 2009. doi : 10.1007/978-3-642-04930-9_40. URL `https://doi.org/10.1007/978-3-642-04930-9_40`.

[106] P. Vandenbussche, G. Atemezing, M. Poveda-Villalón, and B. Vatant. Linked open vocabularies (LOV) : A gateway to reusable semantic vocabularies on the web. *Semantic Web*, 8(3) :437–452, 2017. doi : 10.3233/SW-160213. URL https://doi.org/10.3233/SW-160213.

[107] J. Völker and M. Niepert. Statistical schema induction. In G. Antoniou, M. Grobelnik, E. P. B. Simperl, B. Parsia, D. Plexousakis, P. D. Leenheer, and J. Z. Pan, editors, *The Semantic Web : Research and Applications - 8th Extended Semantic Web Conference, ESWC 2011, Heraklion, Crete, Greece, May 29-June 2, 2011, Proceedings, Part I*, volume 6643 of *Lecture Notes in Computer Science*, pages 124–138. Springer, 2011. doi : 10.1007/978-3-642-21034-1_9. URL https://doi.org/10.1007/978-3-642-21034-1_9.

[108] W3C. Sparql query language for rdf. https://www.w3.org/TR/rdf-sparql-query/, 2013. Accessed : 2020-08-01.

[109] Q. Y. Wang, J. X. Yu, and K. Wong. Approximate graph schema extraction for semi-structured data. In C. Zaniolo, P. C. Lockemann, M. H. Scholl, and T. Grust, editors, *Advances in Database Technology - EDBT 2000, 7th International Conference on Extending Database Technology, Konstanz, Germany, March 27-31, 2000, Proceedings*, volume 1777 of *Lecture Notes in Computer Science*, pages 302–316. Springer, 2000. doi : 10.1007/3-540-46439-5_21. URL https://doi.org/10.1007/3-540-46439-5_21.

[110] W. Wang, J. Yang, and R. R. Muntz. STING : A statistical information grid approach to spatial data mining. In M. Jarke, M. J. Carey, K. R. Dittrich, F. H. Lochovsky, P. Loucopoulos, and M. A. Jeusfeld, editors, *VLDB'97, Proceedings of 23rd International Conference on Very Large Data Bases, August 25-29, 1997, Athens, Greece*, pages 186–195. Morgan Kaufmann, 1997. URL http://www.vldb.org/conf/1997/P186.PDF.

[111] B. Xu, Y. Zhang, J. Liang, Y. Xiao, S. Hwang, and W. Wang. Cross-lingual type inference. In S. B. Navathe, W. Wu, S. Shekhar, X. Du, X. S. Wang, and H. Xiong, editors, *Database Systems for Advanced Applications - 21st International Conference, DASFAA 2016, Dallas, TX, USA, April 16-19, 2016, Proceedings, Part I*, volume 9642 of *Lecture Notes in Computer Science*, pages 447–462. Springer, 2016. doi : 10.1007/978-3-319-32025-0_28. URL https://doi.org/10.1007/978-3-319-32025-0_28.

[112] Y. Xun, X. Cui, J. Zhang, and Q. Yin. Incremental frequent itemsets mining based on frequent pattern tree and multi-scale. *Expert Systems with Applications*, 163 :113805, 2021. ISSN 0957-4174. doi : https://doi.org/10.1016/j.eswa.2020.113805. URL https://www.sciencedirect.com/science/article/pii/S0957417420306217.

[113] Y. Yaghoobzadeh and H. Schütze. Corpus-level fine-grained entity typing using contextual information. In L. Màrquez, C. Callison-Burch, J. Su, D. Pighin, and Y. Marton, editors, *Proceedings of the 2015 Conference on Empirical Methods in Natural Language Processing, EMNLP 2015, Lisbon, Portugal, September 17-21,*

2015, pages 715–725. The Association for Computational Linguistics, 2015. doi : 10.18653/v1/d15-1083. URL https://doi.org/10.18653/v1/d15-1083.

[114] Y. Yaghoobzadeh and H. Schütze. Multi-level representations for fine-grained typing of knowledge base entities. In M. Lapata, P. Blunsom, and A. Koller, editors, *Proceedings of the 15th Conference of the European Chapter of the Association for Computational Linguistics, EACL 2017, Valencia, Spain, April 3-7, 2017, Volume 1 : Long Papers*, pages 578–589. Association for Computational Linguistics, 2017. doi : 10.18653/v1/e17-1055. URL https://doi.org/10.18653/v1/e17-1055.

[115] S. Young, I. Arel, T. P. Karnowski, and D. Rose. A fast and stable incremental clustering algorithm. In *2010 Seventh International Conference on Information Technology : New Generations*, pages 204–209, 2010. doi : 10.1109/ITNG.2010.148.

[116] N. Zong, D. Im, S. Yang, H. Namgoong, and H. Kim. Dynamic generation of concepts hierarchies for knowledge discovering in bio-medical linked data sets. In S. Lee, L. Hanzo, R. Ismail, D. S. Kim, M. Y. Chung, and S. Lee, editors, *The 6th International Conference on Ubiquitous Information Management and Communication, ICUIMC '12, Kuala Lumpur, Malaysia, February 20-22, 2012*, pages 12 :1–12 :5. ACM, 2012. doi : 10.1145/2184751.2184766. URL https://doi.org/10.1145/2184751.2184766.