# Extraction automatique de schéma pour des données massives

Redouane BOUHAMOUM, Zoubida KEDAD
Stéphane LOPES

DAVID - Université de Versailles Saint-Quentin-en-Yvelines
Versailles, France
prénom.nom@uvsq.fr

**Résumé.** Nous nous intéressons dans ce travail au problème d'extraction automatique de schéma pour les données du Web sémantique. La flexibilité des langages utilisés dans ce contexte, comme le langage RDF, peut rendre l'exploitation des données difficile. En effet, le schéma pour ces sources de données peut être incomplet ou manquant et, même s'il est présent, les données ne sont pas contraintes par ce schéma et ne sont pas tenues de le respecter. Nous avons proposé dans des travaux précédents une représentation condensée pour réduire la taille d'un jeu de données RDF en vue d'extraire son schéma. Cependant, pour certaines sources de données particulièrement hétérogènes, la taille de cette représentation demeure trop importante. Nous proposons dans ce papier SC-DBSCAN, un algorithme d'extraction de schéma inspiré de DBSCAN. La conception distribuée de notre algorithme le rend efficace sur de grandes sources de données RDF.

## 1 Introduction

Une quantité croissante de données sont disponibles sur le Web, décrites par les langages proposées par le W3C, comme RDF, RDFS et OWL. Ces langages permettent une description flexible des données, car ils n'imposent pas de structure à laquelle les instances doivent se conformer, contrairement aux bases de données relationnelles. Cette flexibilité lors de la création des données rend leur exploitation difficile. En effet, il peut exister des jeux de données où le schéma est incomplet ou absent. De plus, même si le schéma existe, les données ne sont pas contraintes de le respecter.

La découverte d'un schéma implicite décrivant les classes et les propriétés des entités du jeu de données est utile pour l'utilisation de ces sources de données. Certaines approches ont proposé l'utilisation d'algorithmes de clustering dans le but de regrouper les entités similaires en clusters représentant les classes du schéma. Cependant, l'utilisation de ces approches pour de grandes sources de données reste impossible à cause de la complexité des algorithmes utilisés.

Dans notre travail, nous abordons le problème du passage à l'échelle de la découverte de schéma pour les jeux de données RDF. Dans des travaux antérieurs, nous avons proposé une représentation condensée pour les données RDF sur laquelle un algorithme de clustering peut

être appliqué [Bouhamoum et al. (2018)]. Cependant, lorsque les entités sont décrites par des ensembles de propriétés très hétérogènes, la taille de la représentation condensée demeure trop importante.

Nous proposons dans ce travail SC-DBSCAN, un algorithme de clustering basé sur la densité et scalable. Cet algorithme est inspiré de l'algorithme de clustering DBSCAN [Ester et al. (1996)] et fournit les mêmes résultats que ce dernier. SC-DBSCAN est conçu et mis en œuvre dans un contexte Big Data pour assurer le passage à l'échelle. Il comprend les étapes suivantes : (i) les données sont partitionnées en fonction des propriétés décrivant les entités, (ii) les voisinages de chaque entité sont calculés au sein de chaque partition, (iii) des clusters partiels sont construits dans chaque partition, et (iv) les clusters partiels sont fusionnés.

La suite de cet article est structurée comme suit. La section 2 présente notre problématique. Notre approche est détaillée dans la section 3 et les expérimentations sont présentées dans la section 4. La section 5 présente les travaux existants sur l'extraction de schéma et sur le passage à l'échelle de DBSCAN. Une conclusion est présentée en section 6.

# 2 Problématique

Un jeu de données $D$ est défini par un ensemble de triplets RDF, $D \subseteq (R \cup B) \times P \times (R \cup B \cup L)$ où $R$, $B$, $P$ et $L$ représentent respectivement des ressources, nœuds vides (ressources anonymes), propriétés et littéraux. Dans un jeu de données RDF, tout nœud excepté les littéraux représente une entité. Les données publiées au format RDF peuvent être décrites par un schéma exprimé en RDF, RDFS ou OWL. Cependant, le langage RDF n'impose pas de contrainte sur la conformité des données au schéma, ce qui complique considérablement la compréhension et l'utilisation de ces jeux de données.

Différents travaux portant sur l'extraction de schéma ont été publiés, où les auteurs proposent d'utiliser des algorithmes de clustering afin de grouper dans une même classe les entités présentant des propriétés similaires [Kellou-Menouer et Kedad (2015); Kellou-Menouer et Kedad (2016); Christodoulou et al. (2013)]. L'utilisation de ces approches sur de grands jeux de données est impossible à cause de la complexité des algorithmes de clustering utilisés.

Parmi les algorithmes proposés, le clustering basé sur la densité répond aux exigences de l'extraction de schéma sur des données RDF. Tout d'abord, il permet de former des clusters de forme arbitraire, ce qui est important dans ce contexte où les entités peuvent être décrites par des ensembles de propriétés hétérogènes bien qu'elles soient de même type. Ensuite, il n'exige pas de préciser à priori le nombre de clusters, ce qui est également important, car le nombre de classes n'est en général pas connu. Enfin, il fournit un résultat déterministe et détecte le bruit, i.e. les entités qui ne sont pas assez importantes pour former un cluster. Toutefois, la question du passage à l'échelle de ce type d'algorithmes et leur utilisation dans un contexte big data restent un défi.

Notre problème peut donc être énoncé comme suit : étant donné un grand jeu de données RDF, comment regrouper les entités structurellement similaires pour former des classes et produire un schéma décrivant les données ? Des entités sont structurellement similaires si elles ont un certain nombre de propriétés en commun. Les mesures de similarité sont donc basées sur le nombre de propriétés partagées entre deux entités comme par exemple la similarité de Jaccard [Wikipedia].

# 3 SC-DBSCAN

Nous décrivons dans cette section notre approche de découverte de schéma pour les grands jeux de données RDF. Les différentes étapes de notre proposition sont présentées dans la figure 1.

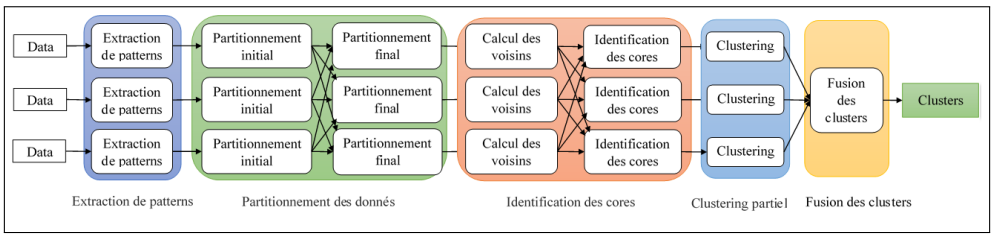



FIG. 1 – *Les étapes de SC-DBSCAN.*

Notre approche consiste tout d'abord à extraire une représentation condensée de l'ensemble de données RDF. Les étapes suivantes de la découverte de schéma seront effectuées sur cette représentation condensée au lieu du jeu de données initial. Cette étape consiste à extraire un ensemble de *patterns* représentant la structure des entités du jeu de données.

**Définition 1.** *Un* pattern $Pt$ *est un ensemble de propriétés distinctes tel qu'il existe au moins une entité décrite par l'ensemble de propriétés* $Pt$*.*

L'extraction de patterns d'un jeu de données produit en sortie toutes les structures (ensembles de propriétés) décrivant les entités du jeu de données. À chaque pattern est associé le nombre d'entités ayant la même structure que ce pattern. Le clustering est ensuite appliqué sur les patterns au lieu des entités pour permettre une exécution plus rapide tout en conservant la même qualité du schéma obtenu.

SC-DBSCAN est un algorithme de clustering distribué et déterministe basé sur la densité, inspiré de DBSCAN. Il permet de calculer efficacement les classes d'un jeu de données RDF et fournit les mêmes résultats que DBSCAN. SC-DBSCAN partitionne d'abord les données, identifie les *core patterns* (patterns ayant un nombre de voisins supérieur à un certain seuil), construit les clusters en parallèle dans chaque partition et enfin, fusionne les clusters partiels produits dans chaque partition pour fournir le résultat final.

Le partitionnement des données est fondé sur l'idée que des patterns similaires partagent au moins une propriété. Les partitions regroupent des patterns ayant des propriétés en commun en garantissant que les patterns similaires seront comparés au moins une fois.

En raison du partitionnement des patterns, le voisinage d'un pattern peut être réparti sur différentes partitions, empêchant ainsi l'identification des *core patterns*. Pour résoudre ce problème, SC-DBSCAN calcule le voisinage de chaque pattern avant l'étape du clustering, en assurant ainsi l'attribution à chaque pattern du rôle approprié (core, bordure ou bruit). Cette étape est effectuée en parallèle dans chaque partition, puis les voisins locaux découverts sur chaque partition sont regroupés par pattern, enfin les core patterns sont identifiés.

En utilisant les core patterns, des clusters partiels sont calculés dans chaque partition en parallèle, sans échange d'informations entre les nœuds de calcul. Les clusters finaux sont formés en fusionnant les clusters partiels qui ont des patterns en commun.

SC-DBSCAN est implémenté en utilisant Spark, un framework de calcul distribué adapté au traitement de grands ensembles de données. Le reste de cette section détaille notre proposition.

## 3.1 Extraction de patterns

Notre approche pour réduire la taille du jeu de données initial consiste à extraire un ensemble de patterns formant une représentation condensée des données.

Tout d'abord, le jeu de données est divisé et distribué sur les nœuds de calcul. À partir des triples RDF, l'identifiant du sujet et les propriétés des entités sont extraits pour former des paires de la forme $(entityID, property)$. Toutes les propriétés d'une même entité sont ensuite regroupées pour composer les entités et produire les paires $(entityID, \{p_1, p_2, p_3, \ldots\})$.

Les patterns sont ensuite extraits pour constituer un ensemble de paires $(pattern, nb)$, $nb$ étant le nombre d'entités décrites par le pattern. Pour cela, les paires $(entityID, \{p1, p2, p3, \ldots\})$ sont lues et le résultat $(pattern, 1)$ est généré, le pattern constituant l'ensemble des propriétés d'une entité. Le nombre 1 indique qu'une entité correspondant à ce modèle a été trouvée.

Enfin, le nombre d'entités décrites par un pattern est calculé en regroupant toutes les paires $(pattern, 1)$ ayant la même clé. À la fin de cette étape, la liste des patterns et le nombre d'entités pour chacun sont obtenus.

Étant donné que le clustering est basé sur la structure des entités et que la similarité est évaluée en fonction des propriétés les décrivant, le clustering de l'ensemble de patterns fournit le même schéma que celui produit par le clustering sur l'ensemble des entités.

La figure 2 représente un jeu de données RDF et les patterns correspondant aux entités du jeu de données. Par exemple, le pattern $pt_1$ représente les entités $e_1$ et $e_2$ décrites par le même ensemble de propriétés $\{b, c\}$.

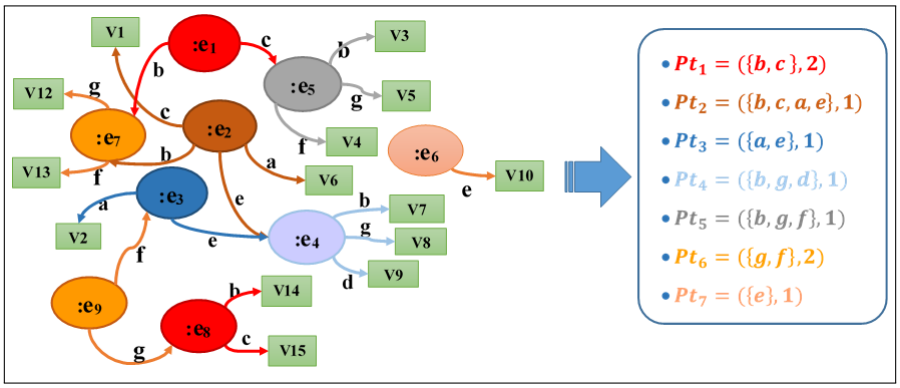



FIG. 2 – *Example d'un jeu données RDF et les patterns correspondant.*

## 3.2 Partitionnement des données

Le partitionnement des données joue un rôle important dans le traitement efficace des grands jeux de données. Cela permet de distribuer correctement les calculs sur les nœuds d'un

cluster. Dans notre contexte, il assure la division du jeu de données initial en sous-ensembles pour générer des tâches de clustering pouvant être traitées en parallèle. Lors du calcul des clusters, un partitionnement adéquat limite également les coûts de communication entre les partitions : le clustering des patterns dans une partition ne nécessite aucune donnée située dans une autre partition et il n'y a donc pas de transfert de données entre les noeuds de calcul. Enfin, le partitionnement des données fournit assez d'informations pour fusionner les clusters partiels. Notre méthode de partitionnement génère des partitions non disjointes et les patterns dupliqués sont utilisés pour fusionner les clusters partiels.

Dans notre approche, une partition est créée pour chaque propriété impliquée dans un pattern et contient tous les patterns décrits par cette propriété. De cette façon, tous les patterns qui pourraient être similaires sont regroupés dans la même partition. Les patterns qui ne se trouvent jamais dans la même partition ne partagent aucune propriété et leur comparaison est donc inutile.

**Définition 2.** *Une partition est un sous-ensemble des patterns obtenus à partir d'un jeu de données, choisis en fonction d'une propriété donnée. Partitionner un ensemble de patterns produit l'ensemble de partitions* $partitionSet = \{part_{p_x} \mid p_x \in P\}$ *où* $P$ *est l'ensemble de toutes les propriétés du jeu de données et où* $part_{p_x}$ *contient tous les patterns décrits par la propriété* $p_x$.

Si on considère l'ensemble de patterns obtenus dans l'exemple précédant, le partitionnement produit les partitions présentées dans la figure 2.

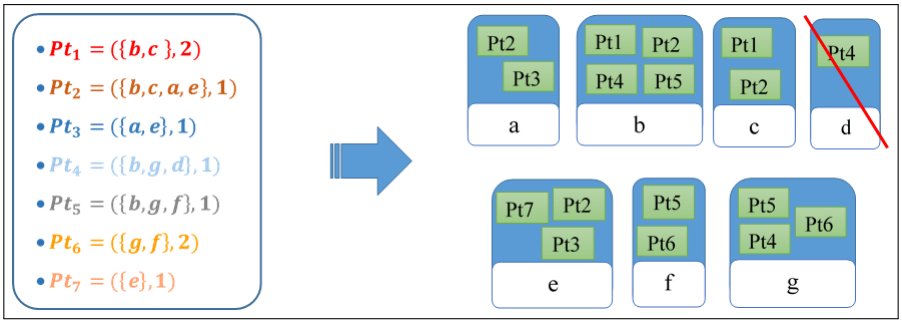



FIG. 3 – *Partitionement de l'ensemble de patterns.*

Le nombre d'éléments d'une partition $part_{p_x}$ peut être supérieur à la capacité d'un nœud de calcul. Cela rend le clustering de $part_{p_x}$ coûteux voire impossible. Dans ce cas, cette partition est subdivisée en fonction d'autres propriétés (autre que $p_x$). Ainsi, pour subdiviser $part_{p_x}$, une sous-partition est créée pour chaque propriété autre que $p_x$, et les patterns de $part_{p_x}$ sont distribuées sur les différentes sous-partitions par rapport aux propriétés les décrivant. Récursivement, les partitions dépassant la capacité de calcul sont subdivisées à nouveau jusqu'à ce que toutes aient un nombre d'éléments inférieur à la capacité. Dans notre exemple précédent, si la capacité d'un nœud est fixée à 3, la partition $part_b$ dépasse cette capacité. Elle est donc subdivisée en sous-partitions comme présenté dans la figure 4.

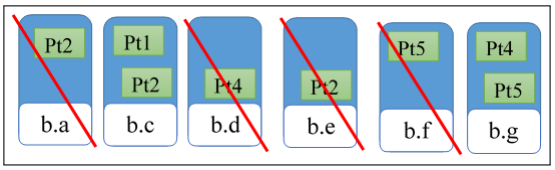



FIG. 4 – *Partitionnement de* $part_b$

## 3.3 Identification des cores

Dans SC-DBSCAN, le clustering est exécuté sur les patterns au lieu des entités. La définition d'un *core pattern* doit tenir compte du nombre d'entités représentées par ce pattern. Les autres patterns sont soit des *bordures*, i.e. voisins d'un core pattern, soit du bruit, i.e. n'appartenant à aucun cluster.

**Définition 3.** *Soient* $\epsilon$ *un seuil de similarité et* $minPts$ *un nombre d'entités. Un pattern est un* core pattern *si la somme de son nombre d'entités et du nombre d'entités des patterns dans son* $\epsilon$*-voisinage est supérieure au seuil* $minPts$*.*

Tout d'abord, pour chaque pattern, la liste de ses voisins dans chaque partition est calculée en parallèle. Ensuite, pour chaque pattern, tous les voisins trouvés dans chaque partition sont regroupés pour constituer la liste complète de ses voisins. Enfin, les patterns ayant une somme d'entités supérieure ou égale à $minPts$ sont définis comme étant des core patterns. Ce processus garantit que les rôles attribués à chaque pattern sont les mêmes que ceux qui auraient été attribués sans partitionner les données.

Si $\epsilon = 0.5$ et $minPts = 3$ dans notre exemple, les core patterns identifiés sont $pt_2$ et $pt_4$.

## 3.4 Clustering partiel au sein des partitions

Les core patterns fournissent suffisamment d'informations pour former les clusters partiels dans chaque partition. Seuls les core patterns produisent un cluster en ajoutant leurs voisins en tant qu'éléments du cluster. Les autres patterns sont soit des bordures dans le voisinage d'un core qui seront affectées au cluster, soit du bruit.

Pour chaque core pattern $pt_i$, un cluster $c_i$ contenant $pt_i$ et ses voisins est créé. Ensuite, parmi les patterns ajoutés au cluster, les cores sont sélectionnés et leurs voisins ajoutés au cluster $c_i$. Les clusters partiels sont identifiés en répétant ce processus de manière récursive sur les patterns nouvellement ajoutés jusqu'à ce qu'un pattern bordure soit trouvé. Tous les patterns non affectés à un cluster sont considérés comme du bruit.

La figure 5 présente les clusters calculés sur chaque partition.

## 3.5 Fusion des clusters partiels

La fusion des clusters partiels vise à identifier les clusters qui s'étendent sur plusieurs partitions et à les fusionner.

Un pattern $pt_x$ est attribué à un cluster $c_i$ si $pt_x$ est *density-reachable* à partir d'un core pattern de $c_i$.

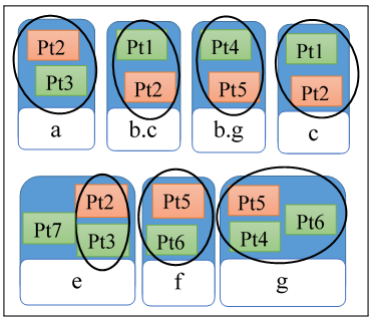



FIG. 5 – *Les clusters partiel obtenu sur chaque partition*

**Définition 4.** *(Density-reachable) Un pattern $pt_d$ est $density - reachable$ à partir d'un pattern $pt_s$ s'il existe une chaîne de patterns $pt_1, \ldots, pt_z$ avec $pt_1 = pt_s$, $pt_z = pt_d$ tel que $pt_{i+1}$ se trouve dans le voisinage de $pt_i$.*

Si ce même pattern $pt_x$ est affecté à un autre cluster $c_j$, cela signifie qu'il est density-reachable à partir d'un core pattern de $c_j$. Si $pt_x$ est un core, cela constituerait un pont entre les clusters $c_i$ et $c_j$. Ainsi, ces patterns doivent être affectés au même cluster et donc $c_i$ et $c_j$ doivent être fusionnés.

L'étape de fusion identifie les clusters qui s'étendent sur différentes partitions en recherchant les clusters locaux qui ont un core pattern en commun et en fusionnant ces clusters pour obtenir le résultat final.

Si un pattern bordure est affecté à différents clusters au cours de la phase du clustering, il sera attribué de manière aléatoire à l'un de ces clusters lors de la fusion.

SC-DBSCAN garantit un résultat de clustering identique à l'utilisation du DBSCAN séquentiel.

Considérant les clusters partiels obtenus dans notre exemple, les clusters des partitions $part_a$ et $part_{b.c}$ sont fusionnés puisqu'ils partagent le core pattern $pt_2$. Les clusters finaux sont présentés dans la figure 6 et représentent les classes du schéma ($Class1$ et $Class2$).

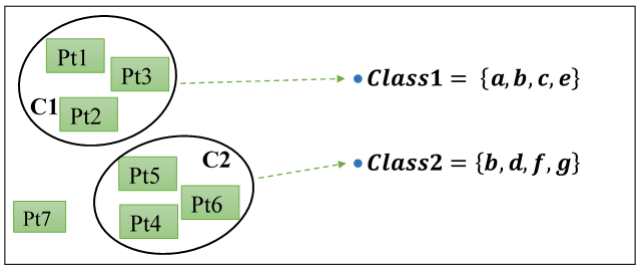



FIG. 6 – *Le resultat final du clustering.*

# 4 Expérimentations

Nous présentons dans cette section les évaluations de notre approche afin de valider son efficacité. Pour nos expérimentations, nous utilisons Apache Spark 2.3 installé sur un cluster de calculs de 5 nœuds équipés de 32Go de RAM.

Nous détaillons en premier lieu les évaluations effectuées sur la représentation condensée des données RDF pour montrer la rapidité du processus et le ratio de réduction. Par la suite, nous exposerons les résultats concernant notre algorithme de clustering SC-DBSCAN.

## 4.1 Réduction du jeux de données

Nous avons évalué notre représentation condensée sur différents jeux de données RDF réels : DBpedia [1] qui est une extraction de Wikipedia au format RDF ; DBLP [2] qui contient des données sur plus de 1.8 million de publications scientifiques ; Katrina et Charley [3] qui représentent des données d'observations de cyclones et tornades aux États-Unis.

Le tableau 1 montre, pour chaque jeu de données, le nombre de triplets RDF qui constituent les données, le nombre d'entités, le nombre de patterns extraits, et le temps nécessaire pour l'extraction des patterns.

| **Source** | **Triples** | **Entités** | **Patterns** | **Temps (s)** |
|---|---|---|---|---|
| DBpedia | 9 500 000 000 | 66 195 296 | 1 918 480 | 750 |
| DBLP | 222 375 855 | 16 086 516 | 351 | 163 |
| Katrina | 203 386 049 | 3 409 | 37 | 100 |
| Charley | 101 956 760 | 3 353 | 52 | 50 |

TAB. 1 – *Évaluations de la construction d'une représentation condensée.*

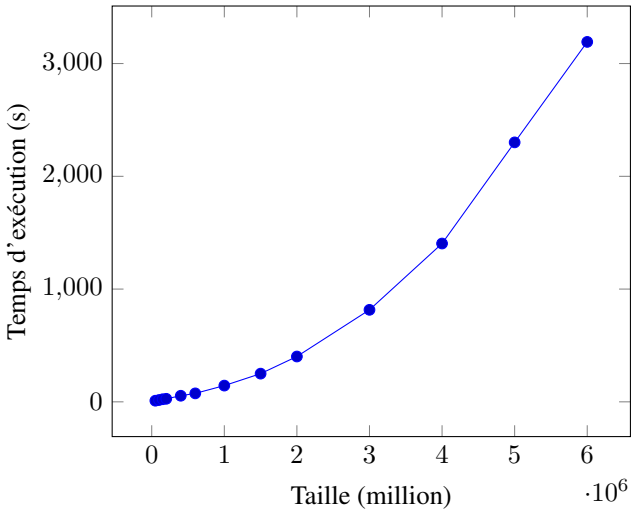
Le nombre de patterns qui constituent la représentation condensée est lié à l'hétérogénéité des entités : plus elles sont décrites par des ensembles de propriétés hétérogènes, plus le nombre de patterns est grand. Si on considère DBpedia, le nombre élevé de patterns s'explique par le fait que la source contient des entités très hétérogènes, contrairement à DBLP, Katrina et Charley qui sont moins hétérogènes et produisent un nombre réduit de patterns.

Nos évaluations montrent que pour certains jeux de données comme DBLP, Katrina et Charley, la taille initiale a été considérablement réduite, permettant une extraction de schéma en utilisant des algorithme de clustering existants. Cependant, pour des sources dont l'hétérogénéité est élevée comme DBpedia, l'extraction de schéma reste difficile.

En considérant le temps nécessaire pour extraire les patterns, les évaluations montrent que notre approche est capable de traiter de grandes sources de données dans un temps réduit (environ 12 minutes pour DBpedia qui est composée de plus de 9 milliards de triplets).

1. https ://old.datahub.io/dataset/dbpedia
2. https ://old.datahub.io/dataset/dblp
3. http ://wiki.knoesis.org/index.php/LinkedSensorData

FIG. 7 – *Scalabilité de SC-DBSCAN*




## 4.2 Clustering

Comme SC-DBSCAN produit exactement le même résultat que DBSCAN, nos évaluations porterons uniquement sur sa capacité à extraire un schéma sur de grands jeux de données.

Nous avons tout d'abord analysé le passage à l'échelle en utilisant des jeux de données de tailles différentes. Ensuite, nous avons étudié l'influence du nombre de propriétés décrivant les patterns.

Pour mener à bien ces différentes évaluations en contrôlant les paramètres, nous avons utilisé des jeux de données synthétiques générés en utilisant le générateur "IBM Quest Synthetic Data Generator" [IBM].

Dans ce qui suit, nous avons fixé $minPts$ à 3 et $\epsilon$ à 0.8 et nous utilisons, pour évaluer la similarité entre deux patterns, la formule de Jaccard (rapport entre le cardinal de l'intersection des propriétés de deux patterns et le cardinal de l'union de ces propriétés.

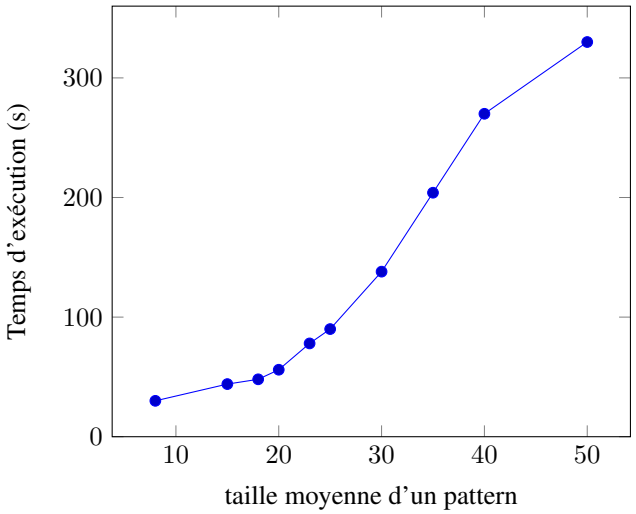
$$J(P1, P2) = \frac{|Prop(P1) \cap Prop(P2)|}{|Prop(P1) \cup Prop(P2)|}$$

Nous avons exécuté SC-DBSCAN sur des jeux de données de tailles différentes, chacun d'eux contenant des entités décrites en moyenne par 15 propriétés, i.e. la dimension moyenne d'une entité est de 15. La figure 2 présente les temps d'exécution de SC-DBSCAN.

Les résultats montrent l'efficacité de notre algorithme pour traiter de grands jeux de données. Le clustering d'un jeux de données contenant 5 millions d'entités prend environ 38 minutes. En comparaison, l'application de NG-DBSCAN sur un jeu de données de la même taille nécessite 30 minutes mais fournit un résultat approximatif [Lulli et al. (2016)].

Ces résultats s'expliquent par le fait que la première étape de SC-DBSCAN génère des partitions contenant un nombre de patterns pouvant être traité par un seul nœud du cluster :

FIG. 8 – *Influence de la dimensions des données.*




chaque nœud doit traiter un nombre de patterns inférieur à sa capacité dans une tâche. En outre, SC-DBSCAN ignore certaines comparaisons inutiles lors de la recherche du voisinage de chaque pattern, car les patterns ne sont comparés que s'ils partagent au moins une propriété.

Comme nous l'avons expliqué, notre approche de partitionnement est fondée sur les propriétés décrivant les données. Par conséquent, le nombre de propriétés influe sur le comportement de notre algorithme. La figure 3 montre les performances de SC-DBSCAN lorsque le nombre de propriétés varie. Pour cela, nous avons effectué des évaluations sur un jeu de données de 150000 patterns avec une capacité de 2000 en faisant varier la moyenne du nombre de propriétés décrivant un pattern (la dimension d'un pattern).

Le partitionnement distribue les patterns par rapport aux propriétés les décrivant. Plus un pattern est décrit un nombre important de propriétés, plus il sera distribué sur plusieurs partitions augmentant ainsi la charge des partitions, ce qui nécessite des divisions hiérarchiques sur plusieurs niveaux. Cela ce traduit par la génération d'un grand nombre de partitions, ce qui explique l'évolution de la courbe en augmentant la taille des patterns.

# 5 État de l'art

La découverte de schéma sur les jeux de données RDF a été abordée dans plusieurs travaux de recherche. Certain travaux proposent d'utiliser des algorithme de clustering afin de grouper les entités similaires dans des clusters représentant les classes du schéma. L'approche proposée dans [Kellou-Menouer et Kedad (2015); Kellou-Menouer et Kedad (2016)] utilise DBSCAN afin de regrouper les entités similaires représentant les differents type inclus dans un jeux de données RDF. Dans [Christodoulou et al. (2013)], les auteurs proposent d'utiliser le clustering hiérarchique pour la découverte des classes composant le schéma. Cependant, l'utilisation de ces approches sur de grandes masses de données est impossible à cause de la complexité des

algorithmes de clustering. D'autres approches ont été proposées pour le traitement de grandes masses de données RDF et ont été implémentées en utilisant des technologies big data [Ruiz et al. (2015); Baazizi et al. (2017)]. Ces approches utilisent les déclarations de type existantes dans le jeu de données RDF pour définir les entités similaires et ne s'appliquent pas dans les cas où ces déclarations sont absentes.

L'algorithme DBSCAN a été largement utilisé et étendu pour assurer sa scalabilité en proposant des versions parallèles. Dans [Patwary1 et al. (2012)], les données sont partitionnées de manière aléatoire et le clustering est ensuite appliqué sur chaque partition en parallèle en comparant les entités d'une partition avec l'ensemble du jeu de données. Dans [Luo et al. (2016)], S-DBSCAN partitionne les données aléatoirement, puis calcule les clusters dans chaque partition. Les clusters dont les centres sont proches les uns des autres sont ensuite fusionnés. L'approche proposée dans [Savvas et Tselios (2016)] est assez similaire à S-DBSCAN, mais fusionne les clusters dont l'intersection est non vide. Après le partitionnement et le calcul des clusters partiels dans chaque partition, [Han et al. (2016)] définit un intervalle pour chaque partition et considère les points hors de cet intervalle comme base pour fusionner les clusters partiels. MR-DBSCAN partitionne les données à l'aide du partitionnement binaire de l'espace, duplique les frontières de chaque partition dans les partitions voisines et calcule les clusters [HE et al. (2013)]. Les clusters sont finalement fusionnés s'ils partagent certains points. NG-DBSCAN procède en deux étapes [Lulli et al. (2016)] : tout d'abord, l'$\epsilon$-graphe est calculé en comparant chaque point avec $k$ points choisis aléatoirement et un arc est ajouté entre les plus proches. Ensuite, les sommets ayant le plus grand nombre de voisins sont considérés comme racine du cluster et tous les éléments connectés à cette racine constituent un même cluster.

Les versions existantes de DBSCAN scalable présentent certaines limitations : (i) PDS-DBSCAN compare une partition à l'ensemble de données, ce qui nécessite de dupliquer la totalité du jeux de données dans tous les nœuds de calcul, (ii) NG-DBSCAN est un algorithme probabiliste et ne fournit pas le même résultat que le DBSCAN séquentiel ; la même limitation existe avec S-DBSCAN et l'approche proposée dans [Savvas et Tselios (2016)], qui repose sur les centres pour fusionner les clusters partiels, (iii) il n'existe pas d'ordre relatif sur les ensembles de données Web, comme requis dans [Han et al. (2016)], (iv) MR-DBSCAN utilise le partitionnement binaire de l'espace, qui n'est pas adapté aux données de grande dimensionnalité telles que les jeux de données RDF.

# 6 Conclusion

Nous avons proposé SC-DBSCAN, une approche d'extraction de schéma à partir des données du web sémantique et applicable sur de grandes masses de données.

Afin d'atteindre cet objectif, nous avons proposé en un premier lieu de condenser les données RDF afin de réduire leur taille pour le clustering. Ensuite, nous avons proposé un partitionnement des données permettant de concevoir un algorithme distribué de clustering basé sur la densité. Nos expérimentations ont montré que SC-DBSCAN permet d'extraire un schéma à partir d'un grand jeu de données RDF.

Nous travaillons sur l'optimisation de SC-DBSCAN en améliorant l'étape de partitionnement afin de produire un nombre minimum de partitions et d'éviter les comparaisons inutiles.

## Références

Baazizi, M.-A., H. B. Lahmar, D. Colazzo, G. Ghelli, et C. Sartiani (2017). Schema inference for massive json datasets. *EDBT*.

Bouhamoum, R., K. K. Kellou-Menouer, S. Lopes, et Z. Kedad (2018). Scaling up schema discovery approaches. *International Conference on Data Engineering Workshops*.

Christodoulou, K., N. W. Paton, et A. A. Fernandes (2013). Structure inference for linked data sources using clustering. *EDBT/ICDT*.

Ester, M., H.-P. Kriegel, J. Sander, et X. Xu (1996). A density-based algorithm for discovering clusters in large spatial databases with noise. *KDD*.

Han, D., A. Agrawal, W. Liao, et A. Choudhary (2016). A novel scalable dbscan algorithm with spark. *International Parallel and Distributed Processing Symposium Workshops*.

HE, Y., H. TAN, W. LUO, S. FENG, et J. FAN (2013). Mr-dbscan : a scalable mapreduce-based dbscan algorithmfor heavily skewed data. *International Parallel and Distributed Processing Symposium Workshops*.

IBM. Ibm quest synthetic data generator. `https://sourceforge.net/projects/ibmquestdatagen/files/latest/download`. Accessed : 2018-10-01.

Kellou-Menouer, K. et Z. Kedad (2015). Schema discovery in RDF data sources. In *Conceptual Modeling - 34th International Conference, ER*, pp. 481–495. Springer.

Kellou-Menouer, K. et Z. Kedad (2016). A self-adaptive and incremental approach for data profiling in the semantic web. *T. Large-Scale Data- and Knowledge-Centered Systems 29*, 108–133.

Lulli, A., M. Dell'Amico, P. Michiardi, et L. Ricci (2016). Ngdbscan :scalable density based clustering forarbitrary data. *VLDB*.

Luo, G., X. Luo, et T. F. Gooch (2016). A parallel dbscan algorithm based on spark. *BDCloud*.

Patwary1, M. M. A., D. Palsetia, A. Agrawal, W. k. Liao, F. Manne, et A. Choudhary (2012). A new scalable parallel dbscan algorithm using the disjoint-set data structure. *International Conference for High Performance Computing, Networking, Storage and Analysis*.

Ruiz, D. S., S. F. Morales, et J. G. Molina (2015). Inferring versioned schemas from nosql databases and its applications. *ER*.

Savvas, I. K. et D. Tselios (2016). Parallelizing dbscan algorithm using mpi. *International Conference on Enabling Technologies : Infrastructure for Collaborative Enterprises*.

Wikipedia. Jaccard index. `https://en.wikipedia.org/wiki/Jaccard_index`. Accessed : 2018-10-15.

## Summary

The problem addressed in this paper is automatic schema discovery for semantic Web data. More and more datasets are published on the Web, in languages such as RDF. These languages provide a high flexibility: the schema describing the data can be incomplete or missing, and even if provided, the data is not constrained by this schema. Several approaches have been

proposed in order to discover the underlying schema for an RDF dataset; in our work, we focus on the scalability issues raised by such approaches. In previous works, we have proposed an approach for building a condensed representation of an RDF dataset prior to schema discovery. However, for some datasets, the size of this condensed representation remains too large. In this paper, we propose SC-DBSCAN, an approach for automatic schema discovery inspired from DBSCAN. SC-DBSCAN is suitable for large RDF dataset. We present an implementation of our approach using big data technology along with some experiments to show the effectiveness of our approach.