HAL open science

# Advanced information extraction by example

Ngurah Agus Sanjaya Er

## To cite this version:

Ngurah Agus Sanjaya Er. Advanced information extraction by example. Information Retrieval [cs.IR]. Télécom ParisTech, 2018. English. NNT : 2018ENST0060 . tel-03194624

**HAL Id: tel-03194624**

**https://pastel.hal.science/tel-03194624v1**

Submitted on 9 Apr 2021

**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.

ParisTech
INSTITUT DES SCIENCES ET TECHNOLOGIES
PARIS INSTITUTE OF TECHNOLOGY

TELECOM ParisTech

2018-ENST-0060

EDITE - ED 130

# Doctorat ParisTech

# T H È S E

**pour obtenir le grade de docteur délivré par**

## TELECOM ParisTech

**Spécialité "Informatique et Réseaux"**

*présentée et soutenue publiquement par*

**Ngurah Agus Sanjaya ER**

le 17 Décembre 2018

# Techniques avancées pour l'extraction d'information par l'exemple

**Jury**

| | |
|---|---|
| **Mme. Salima BENBERNOU**, Professeur, Université Paris Descartes | Examinateur |
| **M. Ismail KHALIL**, Professor, Johannes Kepler University | Rapporteur |
| **M. Pierre SENELLART**, Professeur, École normale supérieure | Examinateur |
| **Mme. Genoveva VARGAS-SOLAR**, Chargé de recherche (HDR), CNRS | Rapporteur |
| **M. Talel ABDESSALEM**, Professeur, Télécom ParisTech | Directeur de thèse |
| **M. Stéphane BRESSAN**, Associate Professor, National University of Singapore | Co-directeur de thèse |

**TELECOM ParisTech**
école de l'Institut Mines-Télécom - membre de ParisTech
46 rue Barrault 75013 Paris - (+33) 1 45 81 77 77 - www.telecom-paristech.fr

T H È S E

# Abstract

Searching for information on the Web is generally achieved by constructing a query from a set of keywords and firing it to a search engine. This traditional method requires the user to have a relatively good knowledge of the domain of the targeted information to come up with the correct keywords. The search results, in the form of Web pages, are ranked based on the relevancy of each Web page to the given keywords. For the same set of keywords, the Web pages returned by the search engine would be ranked differently depending on the user. Moreover, finding specific information such as a country and its capital city would require the user to browse through all the documents and reading its content manually. This is not only time consuming but also requires a great deal of effort. We address in this thesis an alternative method of searching for information, i.e. by giving examples of the information in question. First, we try to improve the accuracy of the search by example systems by expanding the given examples syntactically. Next, we use truth discovery paradigm to rank the returned query results. Finally, we investigate the possibility of expanding the examples semantically through labelling each group of elements of the examples.

We begin with the problem of the set of tuples expansion by example. Set expansion deals with the problem of expanding a set given a set of examples of its members. Existing approaches mostly consider sets of atomic data. In the relational model, relation instances in first normal form are not sets of atomic values but rather are sets of tuples of atomic values. We extend the idea of set expansion to the expansion of sets of tuples, that is of relation instances or tables. Our approach enables the extraction of relation instances from the Web given a handful set of example tuples. For instance, when a user inputs the set of tuples < *Indonesia*, *Jakarta*, *IDR*>, <*China*, *Beijing*, *CYN*>, <*Canada*, *Ottawa*, *CAD*> the system that we have implemented returns relational instances composed of countries with their corresponding capital cities and currency codes.

Both set and set of tuples expansions require a ranking mechanism to present the most relevant candidates to the user. Such ranking mechanism typically analyses the heterogeneous graph of sources, seeds, wrappers, attributes, ontological information, and candidates, to automatically select the relevant candidates, or as we call them, true candidates. We propose of harnessing the truth finding algorithm as a possible method of finding the true candidates among all candidates extracted by the set of tuples expansion system. We empirically and comparatively evaluate the accuracy of several different state-of-the-art truth finding algorithms with several cases of the set of tuples expansion. From the result of the experiments, we find that the

Accu algorithm achieves the best performance. Accu also outperforms the classical ranking method based on lazy walk process on the heterogeneous graph typically applied by set expansion systems.

Web documents which contain information in the form of $n$-elements ($n > 2$) tuples rarely exist, while on the other hand, binary tuples are common to be found. We propose here the task of tuple reconstruction which leverages the binary tuples to reconstruct $n$-elements tuples. Starting with the given seeds of $n$-elements tuples, we break each tuple into binary ones by introducing the key element concept of a tuple on which the remaining elements depend on. We then search the Web to find and extract other binary tuples which have the same relation as the ones we have generated previously from the set of examples by leveraging the set of tuples expansion. Next, we reconstruct $n$-elements tuples from these extracted binary tuples. The tuple reconstruction task provides a method of discovering $n$-elements tuples which set of tuples expansion system fail to retrieve.

Each group of elements in the given set of examples describes a semantic class. However, the label of each semantic class of the seeds is unknown a priori. Having the labels of the semantic classes may help in expanding the set of examples itself as well as identifying the true member of the class. Following this intuition, as the first work, we leverage the topic labelling approach to label groups of elements of the given examples. We argue here that the set of examples can be viewed as the top-$N$ words of a topic model. We generate the candidate labels from the categories assigned in documents returned as the search result when querying an ontology using the set of words. We then again rely on truth finding algorithms to map each word to the best label. To label the topic, we select the label from each corresponding label of its top-$N$ words.

Finally, we propose the task of set labelling where we start from some example members of a set and then infer the most appropriate labels for the given set of terms or words. We illustrate the task and a possible solution with an application to the classification of cosmetic products and hotels. The novel solution proposed in this research is to incorporate a multi-label classifier trained from the labelled datasets. The multi-label classifier is responsible for retrieving the relevant labels for a given input. We use vectorization of the description of the data as input to the classifier as well as labels assigned to it. Given a previously unseen data, the trained classifier then returns a ranked list of candidate labels (i.e., additional seeds) for the set. These results could then be used to infer the labels for the set.

**Keywords**: set of tuples expansion, ranking, graph of entities, truth finding, set labelling, topic labelling

# Résumé

La recherche d'information sur le Web requiert généralement la création d'une requête à partir d'un ensemble de mots-clés et sa soumission à un moteur de recherche. Le résultat de la recherche, qui est une liste de pages Web, est trié en fonction de la pertinence de chaque page par rapport aux mots clés donnés. Cette méthode classique nécessite de l'utilisateur une connaissance relativement bonne du domaine de l'information ciblée afin de trouver les bons mots-clés. étant donnée une même requête, i.e. une liste de mots-clés, les pages renvoyées par le moteur de recherche seraient classées différemment selon l'utilisateur. Sous un autre angle, la recherche d'informations très précises telles que celle d'un pays et de sa capitale obligerait, sans doute, l'utilisateur à parcourir tous les documents retournés et à lire chaque contenu manuellement. Cela prend non seulement du temps, mais exige également beaucoup d'efforts. Nous abordons dans cette thèse une méthode alternative de recherche d'informations, c'est-à-dire une méthode qui consiste à des exemples d'informations recherchées. Tout d'abord, nous essayons d'améliorer la précision de la recherche des méthodes existantes en étendant syntaxiquement les exemples donnés. Ensuite, nous utilisons le paradigme de découverte de la vérité pour classer les résultats renvoyés. Enfin, nous étudions la possibilité d'élargir les exemples sémantiquement en annotant (ou étiquetant) chaque groupe d'éléments dans les exemples.

Nous commençons par le problème de l'extension d'un ensemble de n-uplets donné en exemple dans une requête de recherche d'information. L'extension d'ensembles traite le problème de l'élargissement d'une collection à partir d'un ensemble d'exemples de ses membres. Les approches existantes considèrent principalement des ensembles de données atomiques. Dans le modèle relationnel, cependant, les instances de relation de la première forme normale ne sont pas des ensembles de valeurs atomiques, mais plutôt des ensembles de n-uplets de valeurs atomiques. Nous étendons l'idée d'extension d'ensembles de données atomiques à celle d'ensembles de n-uplets, c'est- à-dire d'instances de relations ou de tables. Notre approche permet l'extraction d'instances de relations à partir du Web, étant donné quelques exemples de n-uplets. Par exemple, lorsqu'un utilisateur entre l'ensemble des n-uplets <Indonésie, Jakarta, IDR>, <Chine, Beijing, CYN>, <Canada, Ottawa, CAD>, le système que nous avons implémenté renvoie des instances relationnelles composées des noms de pays avec leur capitale et les codes de leurs devises.

L'extension d'ensembles et d'ensembles de n-uplets nécessite un mécanisme de tri pour pouvoir présenter les candidats (i.e. n-uplets extraits à l'issu de la recherche) les plus pertinents à l'utilisateur. Un tel mécanisme de classement analyse généralement le graphe hétérogène de sources, d'exemples, d'extracteurs, d'attributs, d'informations

ontologiques et de n-uplets candidats, afin de sélectionner automatiquement les candidats les plus pertinents, ou, comme nous les appelons, les vrais candidats. Nous proposons d'exploiter le paradigme de recherche de la vérité comme méthode possible pour trouver les vrais candidats parmi tous les candidats extraits par notre système d'extension de n-uplets. Nous évaluons de manière empirique et comparative la précision de plusieurs algorithmes de recherche de vérité populaire sur plusieurs cas d'extension de jeu de n-uplets. D'après les résultats des expériences, l'algorithme Accu obtient les meilleures performances. Accu surpasse également la méthode de classement classique basée sur le processus de marche paresseuse sur le graphe hétérogène généralement appliquée par les systèmes d'extension.

Les documents Web contenant des n-uplets de taille supérieure à 2 existent rarement, alors que, par contre, les n-uplets binaires sont fréquents. Nous proposons ici la tâche de reconstruction de n-uplets qui utilise les n-uplets binaires pour reconstruire des n-uplets de n éléments. En commençant par les exemples donnés de n-uplets à n éléments, nous décomposons chaque n-uplets en binaire en introduisant le concept d'élément clé d'un n-uplets dont dépend les éléments restants. Nous cherchons ensuite sur le Web pour trouver et extraire d'autres n-uplets binaires ayant la même relation que ceux que nous avons générés précédemment à partir de la série d'exemples en tirant parti de l'expansion de la série de n-uplets. Ensuite, nous reconstruisons des n-uplets à n-éléments à partir de ces n-uplets binaires extraits. La tâche de reconstruction de n-uplets fournit une méthode de découverte des n-uplets de n éléments que le système d'extension de n-uplets, présenté précédemment, ne parvient pas à récupérer.

Chaque groupe d'éléments dans l'ensemble donné d'exemples décrit une classe sémantique. Cependant, l'étiquette de chaque classe sémantique des exemples est inconnue à priori. Avoir les étiquettes des classes sémantiques peut aider à étendre l'ensemble des exemples plus aisément et à identifier le membre réel de la classe. Suite à cette intuition, et en tant que premier travail, nous exploitons l'approche d'étiquetage sur sujet (ou « topic labelling » en anglais) pour étiqueter des groupes d'éléments des exemples donnés. Nous soutenons ici que l'ensemble d'exemples peut être considéré comme les N premiers mots d'un modèle de sujet. Nous générons les étiquettes candidates à partir des catégories attribuées dans les documents renvoyés comme résultat de la recherche lors de l'interrogation d'une ontologie à l'aide de l'ensemble des mots. Nous nous basons ensuite sur des algorithmes de recherche de la vérité pour faire correspondre à chaque mot la meilleure étiquette. Enfin, pour trouver le bon le sujet pour chaque valeur, nous sélectionnons l'étiquette correspondante à ses N premiers mots.

Enfin, nous proposons la tâche d'étiquetage d'ensembles, en partant de quelques exemples puis en déduisant les étiquettes les plus appropriées pour cet ensemble de termes ou de mots. Nous illustrons la tâche et une solution possible avec une application à la classification des produits cosmétiques et des hôtels. La nouvelle solution proposée dans cette recherche consiste à incorporer un classifieur multi-étiquettes entrainé à partir de jeux de données étiquetées. Le classifieur multi-étiquettes est chargé de récupérer les étiquettes pertinentes pour une entrée donnée. Nous utilisons la vectorisation de la description des données en entrée du classifieur ainsi que

les étiquettes qui lui sont attribuées. En se basant sur des données nouvelles (i.e. non encore lues), le classifieur entraîné retourne ensuite une liste classée d'étiquettes candidates (c'est-à-dire des exemples supplémentaires) pour l'ensemble. Ces résultats pourraient ensuite être utilisés pour déduire les étiquettes de l'ensemble.

**Mots-clés**: extension de n-uplets, tri, graphe des entités, recherche de la vérité, annotation d'ensembles, étiquetage de sujets

# Contents

**Contents** ix

**List of Figures** xiii

**List of Tables** xv

**Acknowledgements** xvii

**1 Introduction** 1
- 1.1 Motivations . . . 2
- 1.2 Contributions . . . 4
- 1.3 Outline of the Thesis . . . 6

**2 Related Work** 7
- 2.1 Automatic Information Extraction . . . 7
  - 2.1.1 Taxonomy Based on Data Source . . . 7
  - 2.1.2 Taxonomy Based on Degree of Automation of the Process . . 10
- 2.2 Set Expansion . . . 12
  - 2.2.1 Taxonomy Based on Data Source . . . 12
  - 2.2.2 Taxonomy Based on Target Relations . . . 13
  - 2.2.3 Taxonomy Based on Pattern Construction . . . 13
  - 2.2.4 Taxonomy Based on Ranking Mechanism . . . 18
- 2.3 Truth Finding . . . 20
  - 2.3.1 Taxonomy of Truth Finding Approach . . . 20
  - 2.3.2 Ensemble Approach . . . 22
- 2.4 Topic Labelling . . . 22
- 2.5 Multi-label Classification . . . 24

**I Finding the Truth in Set of Tuples Expansion** 27

**3 Set of Tuples Expansion** 29
- 3.1 Problem Definition . . . 30
- 3.2 Proposed Approach . . . 30
  - 3.2.1 Crawling . . . 30
  - 3.2.2 Wrapper Generation and Candidates Extraction . . . 31

3.2.3 Ranking . . . 36
3.3 Performance Evaluation . . . 40
3.3.1 Experiment Setting . . . 40
3.3.2 Result and Discussion . . . 41
3.4 Comparison with state-of-the-art . . . 46
3.5 Conclusion . . . 47

**4 Truthfulness in Set of Tuples Expansion 49**
4.1 Problem Definition . . . 49
4.2 Proposed Approach . . . 51
4.3 Performance Evaluation . . . 53
4.3.1 Experiment Setting . . . 53
4.3.2 Result and Discussion . . . 57
4.4 Conclusion . . . 61

**5 Tuple Reconstruction 63**
5.1 Problem Definition . . . 64
5.2 Proposed Approach . . . 66
5.3 Performance Evaluation . . . 70
5.3.1 Experiment Setting . . . 71
5.3.2 Result and Discussion . . . 73
5.4 Conclusion . . . 75

**II Expanding the Set of Examples Semantically 77**

**6 Topic Labelling with Truth Finding 79**
6.1 Introduction . . . 79
6.2 Problem Definition . . . 80
6.3 Proposed Approach . . . 81
6.3.1 Generating Candidate Labels . . . 81
6.3.2 Ranking the Labels . . . 82
6.4 Performance Evaluation . . . 86
6.4.1 Experiment Setting . . . 86
6.4.2 Experiments . . . 87
6.4.3 Result and Discussion . . . 88
6.5 Conclusion . . . 92

**7 Set Labelling using Multi-label Classification 93**
7.1 Introduction . . . 93
7.2 Problem Definition . . . 94
7.3 Proposed Approach . . . 94
7.3.1 Training Process . . . 95
7.3.2 User Interaction . . . 95
7.4 Performance Evaluation . . . 96
7.4.1 Experiment Setting . . . 96

7.4.2 Result and Discussion . . . . . 99
7.5 Conclusion . . . . . 101

**8 Conclusions and Perspectives 103**
8.1 Conclusive Summary . . . . . 103
8.2 Perspectives . . . . . 105

**A Ranking of Sources 107**

**B Résumé en Français 113**
B.1 Ensemble d'expansion de n-uplets . . . . . 113
B.1.1 Robot d'aspiration de site Web . . . . . 114
B.1.2 Génération de wrappers et extraction de candidats . . . . . 115
B.1.3 Classement . . . . . 116
B.1.4 Comparaison avec l'état de l'art . . . . . 117
B.2 Vérité dans la série d'expansion de n-uplets . . . . . 119
B.2.1 Définition du problème . . . . . 119
B.2.2 Approche proposée . . . . . 121
B.2.3 Évaluation des performances . . . . . 123
B.3 Reconstruction de n-uplets . . . . . 123
B.3.1 Définition du problème . . . . . 125
B.3.2 Approche proposée . . . . . 126
B.3.3 Évaluation des performances . . . . . 128
B.4 Étiquetage de sujet avec recherche de vérité . . . . . 130
B.4.1 Approche proposée . . . . . 131
B.4.2 Évaluation des performances . . . . . 132
B.5 Définir l'étiquetage . . . . . 133
B.5.1 Définition du problème . . . . . 134
B.5.2 Approche proposée . . . . . 134
B.5.3 Évaluation des performances . . . . . 136
B.6 Conclusion . . . . . 137

# List of Figures


2.1 Plate diagram of LDA [1] . . . . . 23

3.1 Wrapper generation process . . . . . 31
3.2 An illustration of the graph of entities . . . . . 39

4.1 Average F-measure scores . . . . . 57

5.1 Tuple reconstruction illustration . . . . . 65
5.2 Graph of entities with reconstructed tuples . . . . . 71

6.1 *n-grams* average accuracy comparison for dataset DT1 . . . . . 89
6.2 Average accuracy of truth discovery algorithms for *2-grams* . . . . . 90
6.3 Average accuracy of Depen on varying the top-$l$ ($l$=1, 2, 3, 4, 5) . . . 91

7.1 Flowchart of the system . . . . . 94
7.2 Screenshots of the search engine . . . . . 100

B.1 Scores moyens de la F-mesure . . . . . 124

# List of Tables


1.1 An example of a customer table . . . . . 2

2.1 The set of seeds used in DIPRE . . . . . 13
2.2 A small example of multi-label dataset based on the Yves Rocher set 26

3.1 An example of calculating edit distance of two strings . . . . . 35
3.2 The list of entities and relationships of the graph of entities in STEP 38
3.3 Topics used for performance evaluation in the set of tuples expansion 41
3.4 Precision of top-$K$ ($K$ = 10, 25, 50, 100, 200, 300, 400) candidates (OR=Original order, PW=Permutation in wrapper) . . . . . 42
3.5 Recall of top-$K$ ($K$ = 10, 25, 50, 100, 200, 300, 400) candidates (OR=Original order, PW=Permutation in wrapper) . . . . . 43
3.6 Running time of STEP (in minutes) . . . . . 44
3.7 Performance comparison of DIPRE, Ext. SEAL, and STEP . . . . . 46

4.1 Excerpt of input dataset in AllegatorTrack . . . . . 53
4.2 Description of the different topics and corresponding ground truth . . 54
4.3 Precision measures per-attribute of the different truth finding algorithms . . . . . 55
4.4 Recall measures per-attribute of the different truth finding algorithms 56
4.5 Overall precision measures of the different truth finding algorithms . 56
4.6 Overal recall measures of the different truth finding algorithms . . . . 57
4.7 Average number of conflicts and sources of the different topics . . . . 58
4.8 Ranking of sources for topic DT1 - Calling Code . . . . . 59
4.9 Correlation of ranking of sources . . . . . 60

5.1 Topics used for performance evaluation in tuple reconstruction . . . . 72
5.2 Comparison of Precision, Recall, and F-measure of STEP with and without tuple reconstruction . . . . . 73

6.1 Excerpt of input for truth discovery algorithms . . . . . 84
6.2 Indexes for labels extracted from the same article . . . . . 85
6.3 Candidate labels generation . . . . . 87

7.1 Yves Rocher & Tripadvisor dataset . . . . . 98
7.2 Comparison of CountVectorizer (CV) and TFIDFVectorizer (TF) . . 99
7.3 Comparison of TFIDFVectorizer (TF) and doc2vec (DC) . . . . . 99

7.4 Set Labelling Accuracy on Yves Rocher Dataset . . . . . 101

A.1 Ranking of sources for DT2 . . . . . 107
A.2 Ranking of sources for DT3 . . . . . 108
A.3 Ranking of sources for DT4 . . . . . 109
A.4 Ranking of sources for DT5 . . . . . 110
A.5 Ranking of sources for DT6 . . . . . 110
A.6 Ranking of sources for DT7 . . . . . 111

B.1 Comparaison des performances de DIPRE, Ext. SEAL, et STEP . . . 117
B.2 Extrait du jeu de données en entrée dans AllegatorTrack . . . . . 122
B.3 Comparaison de la précision, du rappel et de la F-mesure de STEP avec et sans reconstruction du n-uplet . . . . . 128

# Acknowledgements

This thesis would not have been possible without the support and guidance from many people during my Ph.D. study. It is now my great pleasure to finally be able to thank them.

First and foremost, I would like to express my most profound gratitude to my supervisors, Talel Abdessalem and Stéphane Bressan for giving me the opportunity to do this Ph.D. research as well as their valuable guidance, continuous support, and great patience. They allowed me to pursue my research interests without having a lot of restrictions. They have never doubted and always trusted me in my research. They improved my research background through various collaborations and projects which I have been involved in.

I would also like to thank the member of my Ph.D. committee, particularly, Genoveva Vargas-Solar and Ismail Khalil, for reviewing this thesis; Salima Benbernou and Pierre Senellart for providing valuable insights.

During my Ph.D., I have the privilege of working both in Singapore and Paris. In Paris, I really enjoyed being a member of the DbWeb research group of the Department of Computer Science and Networks. The group provides an excellent research environment, thus I would like to thank Albert Bifet, Fabian M. Suchanek, Antoine Amarilli, Marie Al-Ghossein, and Atef Shaar. In Singapore, I conducted my research in the Database Research Lab at School of Computing Sciences in the National University of Singapore (NUS). Thanks to all members of the group for their invaluable discussions and friendships especially Ashish Dandekar, Debabrota Basu, Remmy Zen, Naheed Arafat, and Fajrian Yunus.

My acknowledgements also go to colleagues with whom I had the chance to collaborate in other contexts. I have to thank Mouhamadou Lamine Ba for our collaboration in incorporating truth discovery with my work in information extraction and your insights regarding this thesis. I would also like to thank Jesse Read for the valuable discussions on multi-label classification.

Last but not least, I have to be grateful to my family. In particular, I would like to dedicate this thesis to my dearest wife, Novi. This Ph.D. journey is full of all kinds of obstacles, only with your love and support have I managed to reach the end of the journey. My children, Reswa and Aurelia, may this achievement set an example for both of you to always do your best. My late father who has taught me that knowledge is of the utmost importance and paved the way for me to have a good education. My mother who has always given me her great support. My brother and sisters, thank you all for the support.

# Chapter 1

# Introduction

When the computer industry emerged in the late 1980s, it revolutionised the way organisations stored information. Before, any information, be it important or not, must be recorded manually through writings in papers. This method was highly time-consuming as well as prone to human errors. Moreover, data was highly unorganised, due to the lack of or missing structure of the data, and searching for a piece of information in a stack of papers seemed to be a very impossible task to achieve. The introduction of the database management system with the emergence of the computer industry changed all that. Storing information still needed human interaction but was done very efficiently through means of easy to use graphical user interface and organised into relational tables. For example, in the database of a retail company, a customer table records all *attributes* related to customer (name, address, telephone number, etc.) such as shown in Table 1.1. The company may also have a purchase table which stores other information detailing the purchase data (date, item, quantity, amount, etc.). In a table, the attributes are represented as *columns*, while *rows* contain instances of data. Each table in the database has a *primary* key which is an attribute, or a combination of attributes, that uniquely identifies each row in the table. The relationship between the two tables is defined using the *foreign* key, a column in the table which links to the primary key in other tables. Searching for any piece of information stored in the database was made possible using a specifically designed structured query language (SQL) that manipulates the relationship between tables. For example, if we want to know what items are purchased by a customer, then we have to find the foreign key that links the two tables and select the attributes that we are interested in from the relation.

Since the World Wide Web gained its popularity in the early 1990s and gradually becoming the ultimate source of information, more and more data are presented on Web pages. The simplest method of presenting data on a Web page is as text. This type of Web page is similar to the traditional paper-based method previously mentioned. The only difference is that it is represented in a Web page thus publicly available via the Internet. A more subtle way of presenting data on Web page is by enclosing the data using specific tags/markups (e.g. <TABLE> </TABLE> or <UL></UL>). An example of such HTML code for the data in Table 1.1 is presented in Listing 1.1. As we can see, each row in Table 1.1 is transformed into

a sequence of string starting and ending with the <tr> and </tr> tags. Each value of the column is then bracketed with the <td></td> tags. The HTML code is then rendered by the Web server and presented to the audience. This type of Web page can be categorised as the semi-structured document. A semi-structured document is a type of structured data which does not comply with the rules of relational databases or data tables but uses special tags or other markers to separate semantic data. Semi-structured data also ensures the hierarchical structure of rows and columns within the data. Each row in the table is enclosed with the <tr></tr>. Inside this tags, a column value of the table is presented by enclosing it with <td></td> tags. Seeking for information on a semi-structured document such as a Web page cannot be achieved easily. One needs to read through the whole page to find the information needed which is time-consuming and requires exhaustive workload. From the point of view of the audience, it is sometimes necessary to download the data from a Web page and maintain them in a database. This can be viewed as a reverse engineering of the process that is described earlier, which starts from a table in a database and ends with the data being presented on a Web page. The need of finding a specific piece of information from the vast amount of Web pages, extracting and maintaining them to a database leads to the introduction of the task of information extraction.

Table 1.1: An example of a customer table

| **Customer ID** | **First Name** | **Last Name** | **Address** | **City** |
|---|---|---|---|---|
| Cust-001 | Agus | Sanjaya | Tukad Melangit | Denpasar |
| Cust-002 | Remy | Zen | University Town | Singapore |
| ... | ... | ... | ... | ... |

## 1.1 Motivations

The goal of the information extraction task is to extract specific data from unstructured or semi-structured data. Information extraction on unstructured data or free text aims at simplifying the text by creating a structured view of the information contained in the text. Two popular sub-tasks of information extraction on unstructured data include Named Entity Recognition (NER) and Relationship Extraction (RE). The former of the two sub-tasks tries to identify information units like names (e.g. person, organisation, location names), numeric (date, money, percent) and temporal (times of day, duration) expressions [2] by leveraging domain-specific knowledge or extracting information from neighboring sentences. For example, in the sentence "Steve Jobs and Steve Wozniak were the co-founders of Apple Inc.", an NER system would identify Steve Jobs and Steve Wozniak as candidates for a person, and Apple Inc. as an organisation. Early approaches of named entity recognition were based on manually crafted rules [3, 4, 5, 6], while more recent studies focus on applying machine learning techniques including Conditional Random Fields [7, 8], Hidden Markov Model [9, 10], Neural Networks and Deep Learning [11, 12]. Relationship

extraction is related to the task of finding relations between entities, e.g. PERSON works for ORGANISATION, PERSON was born in LOCATION. From the previous example of a sentence, an RE system would extract the following relations "Steve Jobs works for Apple Inc.", "Steve Wozniak works for Apple Inc.". RE systems can be classified as supervised approach which requires labelled data [13, 14, 15], semi-supervised approach where a few examples of the targeted relation [16, 17, 18, 19] are given, and unsupervised approach which does not require labelled data or examples [20, 21, 22, 23].

Listing 1.1: Customer Data Presented in HTML Code

```
<html>
<table>
  <tr>
    <th>Customer ID</th>
    <th>First Name</th>
    <th>Last Name</th>
    <th>Address</th>
    <th>City</th>
  </tr>
  <tr>
    <td>C-001</td>
    <td>Agus</td>
    <td>Sanjaya</td>
    <td>Tukad Melangit</td>
    <td>Denpasar</td>
  </tr>
  <tr>
    <td>C-002</td>
    <td>Remmy</td>
    <td>Zen</td>
    <td>University Town</td>
    <td>Singapore</td>
  </tr>
<\table>
<\html>
```

The set expansion task is a sub-task of information extraction which generally applied on semi-structured documents such as Web pages. This specific type of information extraction task deals with finding elements of a semantic set given some example members (*seeds*). Google Sets was a popular Web-based set expansion system provided by Google Labs and later offered in Google Docs Spreadsheet that searched the Web to complete a set of examples given by the user. Indeed, when a given user gave the following set of examples <*IDR*>, <*CYN*> and <*CAD*>, Google Sets returned a list of currency codes including <*EUR*>, <*GBP*>, <*SGD*>, <*USD*>, etc. Existing set expansion approaches mostly only consider sets of atomic

data [24, 25, 26, 27] or binary relations [16, 28, 29, 30]. However, in the relational model, relation instances in the first normal form are not sets of atomic values but rather are sets of tuples of atomic values. Several solutions have been proposed to extract tuples from the Web by exploiting the structure of the data [31, 32, 33, 34] but they either require specific knowledge in linguistics, only work if certain structures present in the document, or need human supervision. This motivates us to generalise the set expansion problem to the set of tuples expansion problem.

## 1.2 Contributions

In this thesis, we focus on extracting information in the form of $n$-ary relations (*tuples*) from semi-structured data. We refer the task as tuples expansion. In the first part of the thesis, we detail our approach that starts with a few examples of tuples and extends them from the Web syntactically. To rank the candidates, we build a heterogeneous graph and apply a random walk on the graph. We then turn our attention to the reliability of the extracted candidates and apply a ranking mechanism using truth finding approach as an alternative to the previous ranking mechanism. In the second part of the thesis, we investigate ways of expanding the tuples semantically. To extend the tuples semantically, we need to be able to label the semantic class of each element of the tuples. Once the label is acquired, finding other members of the semantic class would be easier to achieve. We summarise our contributions as follows.

- Part I: On Finding the Truth in Set of Tuples Expansion
  We start with the definition of the set of tuples expansion task. Given a user seed inputs, for instance, the set of tuples <*Indonesia*, *Jakarta*, *Indonesian Rupiah*>, <*China*, *Beijing*, *Yuan Renminbi*>, <*Canada*, *Ottawa*, *Canadian Dollar*> the system that we have implemented returns relational instances composed of countries with their corresponding capital cities and currency names. We extend this idea of tuples expansion from set expansion problem. However, generalising the set expansion problem to set of tuples expansion introduces different challenges in the crawling, wrapper generation and candidate extraction, as well as the ranking phase.

  One of the main purposes of the set of tuples expansion is to be able to present to the user a reliable list of candidate tuples. In Chapter I, we adopt the ranking mechanism proposed by SEAL [24] on our approach of tuples expansion. However, this ranking mechanism is not able to reliably determine the true values amongst the extracted candidates. Distinguishing amongst true and false values in an automated way, usually known as truth finding, is a challenging problem that has been extensively studied in several contexts such as data integration and fact-checking. Consequently, we propose to incorporate a truth finding algorithm into our expansion process. The goal is to be able to present the true candidates to the user and achieve better performance in terms of precision, recall, and F-measure. We show that using the truth finding

algorithm, our approach achieves better performance in terms of precision, recall, and F-measure than our previous work.

We try to further improve the reliability of the extracted candidates by adding more knowledge. To achieve this, we propose the task of tuple reconstruction. The goal of the task is to be able to reconstruct reliable tuples from binary ones. Each tuple from the given set of examples has a key element which all the remaining elements of the tuple is depended on. For example, in the set of examples <*Indonesia*, *Jakarta*, *Indonesian Rupiah*>, <*China*, *Beijing*, *Yuan Renminbi*>, <*Canada*, *Ottawa*, *Canadian Dollar*>, obviously "Indonesia", "China", and "Canada" are the key elements of each tuple. These key elements are from the semantic class "country". All remaining elements are values that explain some attributes of the country. "Jakarta", "Beijing", and "Ottawa" denote the values of the semantic class "capital city", while "Indonesian Rupiah", "Yuan Renminbi", and "Canadian Dollar" are all values of the "currency name" class. We split the tuples into two sets of binary tuples of the form country-capital city and country-currency name. We use the two sets of binary tuples to form queries and try to retrieve other possible binary relations from a knowledge base. The extracted binary relations are then used to reconstruct the tuples.

- Part II: Expanding the Examples Semantically
  In the second part of the thesis, we explore the possibility of expanding the examples semantically. First, we leverage topic labelling as a mean to label the set of examples. Instead of giving examples member of a semantic set, we consider the top-$N$ words of topics generated by topic models. We argue that these top-$N$ words can be viewed as members of a semantic class as they have found to have strong correlations between them by the topic model. To collect candidate labels, we use words of each topic to query an ontology. The ontology then returns a collection of documents with their categories. We use these documents' categories as the candidate labels. We then leverage truth finding algorithms to infer a one-to-one relationship between each word in a topic with the label. The label for each topic is then selected from the label of its top-$N$ words.

  Next, we propose the task of set labelling. The set labelling task consists of finding the most suitable label for a set when given examples of its members. Our approach relies on a multi-label classifier to propose a set of relevant labels. We apply our approach to cosmetic products and hotel domain. In both domains, we use textual data (description of the cosmetic product and reviews of the hotel) as the features to train the classifier. We also take the labels assigned to each product and hotel as input to the classifier. For unknown data, we then ask the classifier to propose a set of labels for each data and we take the intersection of the two sets of labels as the label for the set.

## 1.3 Outline of the Thesis

In the next chapter, we provide a brief summary of research areas that are relevant to this thesis: information extraction, set expansion, truth finding, topic labelling, and multi-label classification. We discuss only areas of literature where the research have intersected with ours in some degree. Chapters 3-6 cover the work we have done on tuples expansion and investigate the reliability of the extracted candidates. Chapters 6-7 detail our work on expanding the examples semantically. We conclude the thesis and discuss possible future works in Chapter 8. All of these chapters present coherent research work and contributions on extracting tuples from the Web, assessing the reliability of the extracted candidates as well as investigating the possibility of expanding the examples semantically.

On all projects presented in this thesis, I was the lead researcher. Almost all of the research works have been published on conferences and journal. The work on tuples expansion has been published in iiWAS 2016 [35]. The work we have done on investigating the reliability of the extracted candidates using truth finding algorithms was presented in DEXA 2017 [36]. An extended version of the work with added datasets and experiments appeared in IJWIS 2017 [37]. We showed the work on tuples reconstruction in DASFAA-SeCoP 2018 workshop [38]. Our work on harnessing truth discovery algorithms on the topic labelling problem and set labelling using multi-label classification have both been submitted to iiWAS 2018 and will be published in November of 2018 [39, 40].

# Chapter 2

# Related Work

This section overviews the related work on automatic information extraction, set expansion, truth finding, topic labelling, and multi-label classification.

## 2.1 Automatic Information Extraction

Information extraction is the task consisting of the extraction of structured data from unstructured or semi-structured documents. In particular, extraction from Web documents generally leverages the HTML structure of Web pages to identify and extract data. The basic assumption here is that the general formatting of the page, and more specifically its HTML formatting, reflects the structure of the information that it contains and presents. A *wrapper*'s job is to extract the data contained in the Web pages. There have been many solutions proposed by researchers on information extraction. The solutions can be classified according to the data source and the degree of automation involved in the process.

### 2.1.1 Taxonomy Based on Data Source

Information extraction (IE) has been applied to different sources of data, including emails [41, 42, 43], Web documents [44, 45, 46, 47, 48], and para-relational data [49, 27, 50].

Zhang et al. [41] present an approach to extract structured data from emails. The repetitive structures of emails are used to induce the templates. The content of the email which is not part of the templates is then annotated for its semantic type. Types which are common such as dates, address, price, etc. are pre-annotated using existing annotators. Text without any pre-annotated type associated with it is then classified as a product name or not. The classification task is done by implementing a weak classifier using Conditional Random Fields and then applying the Expectation-Maximization algorithm. In [42], Sheng et al. propose Juicer which is a framework for extracting information from large-scale email service. It achieves the goal without hurting the privacy of the users. Juicer learns from a representative sample of emails by clustering them based on templates from which the emails are instantiated (purchase receipts, bills, hotel reservations, etc.). Juicer then applied

a set of classifiers as well as field extractors for each cluster. The result of this learning process is then aggregated into a set of static rules. This set of static rules is then used online for extracting the needed information from newly arrived emails. Mahlawi et al. [43] propose a novel approach to extract structured data from emails about a specific domain. The data extraction process consists of several sub-processes including keyword extraction, sentiment analysis, regular expression extraction, entity extraction, and summary extraction. Keywords in the email are extracted based on the word frequency analysis with refers to the individual email as well as the whole corpus. The sentiment analysis process leverages a dictionary corpus to evaluate the sentiment value of each email. Predefined regular expressions are used to extract important information such as the email of the sender and receiver, date, URL, and phone number. To extract entities from the email, it uses POS tagging technique. From the result of the POS tagging, common nouns are removed while the rest of the nouns are identified as entities. To summarise the content of the email, the authors build a graph using sentences as the vertices. If two sentences are overlapped in terms of content or have semantic similarity, an edge is drawn between the two vertices representing the words. The number of inbound edges is used to rank the vertices. The summary is extracted from the graph as the most important vertex and its corresponding edges.

Information extraction systems which extract information from Web documents assume that the collection of Web pages in the targeted domain are generated from the same unknown template. ExAlg [44] based its approach of extracting database values from the Web on this assumption. This unknown template is responsible for transforming the database values to a scripting language that is accepted by the server to render the Web page. Thus, given a set of template-generated Web pages, the algorithm then tries to deduce the template that is used to generate the Web pages in the first place. Once the template is deduced, the database values encoded can be easily extracted from the Web page. ObjectRunner [45], on the other hand, allows the user to freely specify the targeted data in the form of a Structured Object Description (SOD). A SOD consists of atomic entities and its type (set or tuple). For example, a book object can be specified as a tuple type SOD composed of three type of entities: *title*, *author*, and *price*. Each type of entity has a specific recognizer which is simply a regular expression. Based on the specified SOD and a corpus of Web pages, ObjectRunner builds the wrapper to extract the possible instances of the given SOD. Textual information extracted along with the structured data is stored in a repository for future query. Both of the previous approaches can be applied to any domain, but others focus on a very specific domain. DEXTER [46] specialises on finding product sites on the Web, detecting, and extracting product specifications from these sites. Starting with a small set of seed Web pages from large and popular Websites, DEXTER iteratively finds new Websites which publish the product specifications. Each of the product in the seed Web pages is used as the query to the search engine. DEXTER then produces two rankings of the Websites. First, the retrieved Websites are then ranked based on the number of times each Website is present in the results. From the search results, DEXTER identifies hubs and analyses the backlinks of the hubs to give more score to relevant Websites

and produces the second ranking. The two rankings are then combined based on union and intersection to produce the final ranking of Websites. To detect product specifications Websites as well as to locate product category entry pages and index pages, DEXTER trains a classifier using the anchor text of the links in the home page as the features. Another classifier is trained to detect the portion of the Web page which contains the product specifications. The features used for training include the number of links, number of items, the size of the text, etc. To extract attribute-value pairs from the specifications, DEXTER implements a heuristic approach to find common HTML structures from different specifications and combines this with pattern inference extraction based on noisy annotations. Although ExAlg, ObjectRunner, and DEXTER have been proved to achieve their respective goals, they do not take into consideration the large volume of the Web. DIADEM [47] is a system that can automatically extract large scale (thousands of Websites) of structured data. To explore Websites, identify structured data, and induce wrapper automatically, DIADEM incorporates phenomenological and ontological knowledge. This phenomenological knowledge enables DIADEM to understand that, for example, some attributes are necessary or some values of the attributes are just wild cards for form filling. The ontological knowledge, on the other hand, helps DIADEM to recognize domain schema as well as the entity recognizers for each of the schema type taking into account the Web sites' appearance. The adaptability of DIADEM to various Websites is made possible by the self-adaptive network of relational transducers implemented in each of the components of DIADEM. Previous knowledge of a Web site is used to rearrange the order of execution of the transducers or to react to some specific exceptions. To efficiently extract information from the Web, IE systems need to avoid visiting unimportant Web pages. Unfortunately, most of the approaches to IE implement the conventional crawling technique of using search engines. ACEBot (Adaptive Crawler Bot for Data Extraction) [48] is an intelligent crawling technique to find only important Web pages with content-rich areas. In the learning phase, ACEBot first derives the DOM tree of the pages. It then establishes connections between the pages based on the root-to-link paths of the DOM tree. Each of these paths may or may not lead to important Web pages, i.e. pages with rich content. The paths are then ranked based on its importance. The most important path is selected to navigate the crawling process and ACEBot then performs massive downloading from the targeted Web pages.

Recently, information extraction has also been applied to para-relational data. This data type refers to nearly relational data with the characteristics of relational data but differs in terms of formatting, for example, spreadsheets [49] and diagrams [50]. In [49], Chen et al. propose Senbazuru which is a prototype spreadsheet database management system. Senbazuru allows the user to extract relational data from a large collection of spreadsheets. The collection of spreadsheets is indexed based on the text contained. When a user query arrives, Senbazuru retrieves the relevant datasets using the inverted index and Term Frequency-Inverse Document Frequency (TF-IDF) style ranking. Converting the data in each spreadsheet to relational format is done by first locating data frames (value region, attribute regions, etc.). A data frame is simply non-empty rows region of the spreadsheet. Assigning

semantic labels to the data frames is accomplished by Conditional Random Field (CRF). CRF is also responsible for identifying the parent-child relationship between the data frames. Each value in the value region is then extracted as a tuple by pairing it with the relevant attributes from the attribute region. Finally, the relational tuples are assembled into relational tables. The attributes in different tuples are clustered together into consistent columns. The clustering model takes into consideration the hierarchical relationship as well as schema from knowledge bases such as Freebase [51] and Yago [52]. DiagramFlyer [50] extracts elements of the graphical specification of a diagram, e.g. title, legend, caption, scale, $x$-axis and $y$-axis labels, and type from PDFs. Words are extracted from PDFs by PDF processor with its corresponding two-dimensional bounding box of coordinates. These words boxes are then grouped together as segments based on the proximity and orientation of neighboring word boxes. Unnecessary segments that do not indicate a diagram are removed. Each segment is then assigned a label from the previously mentioned set of labels by using a classifier. The input to the classifier is a set of textual features derived from the segments.

### 2.1.2 Taxonomy Based on Degree of Automation of the Process

Earlier information extraction systems require the user to manually write the extraction rules or wrappers. This manual method has since been left behind as machine learning was introduced to help in generating the rules automatically. From the point of view of how the wrappers are generated, information extraction systems can be classified as supervised, semi-supervised, and unsupervised systems.

In supervised IE systems, a set of Web pages are annotated with the data to be extracted and used as input. The IE systems then infer the wrapper from the labelled Web pages. RAPIER [53] generates the extraction rules in a bottom-up manner. It starts with the most specific rules and then iteratively replaces them with more general rules. The generated rules are only used to extract single-slot data. The rules are comprised of three different patterns, a pattern to match text preceding the filler, a pattern to match the actual filler, and a pattern to match the text following the filler. In this learning process, RAPIER combines the syntactic and semantic information of the input Web pages. The semantic information comes from the part of speech tagger and lexicon such as Wordnet [54]. As opposed to RAPIER, WHISK [55] can extract multi-slot information. The rules are generated using a covering learning algorithm and can be applied to various sources of documents including structured and free text. When applied to free text documents, the input data needs to be labelled with a syntactic analyser and semantic tagger. WHISK employs a top-down manner when generating the rules. The rules are in the form of regular expressions. The initial rules are the most general rules which encompass all instances, then WHISK continually adds a new term to extend the initial rules. In NoDoSe [56], a user can interactively decompose semi-structured document hierarchically. This helps NoDoSe to handle nested objects. It separates the text from the HTML code and applies heuristic-based mining components to

each group. The goal of the mining component is to find the common prefix and suffix that identify different attributes. A tree describing the document structure is the output of this mining task.

The semi-supervised approach requires a small set (instead of complete as in supervised systems) of example from the users to generate the extraction rules. IEPAD [57] tries to find the repetitive patterns in a Web page. The intuition is that data records are commonly rendered using the same template for good presentation. The repetitive patterns can be identified by using a binary suffix tree. This type of tree only considers the exact match of suffixes, therefore the center start algorithm is then applied to align multiple strings. This alignment of strings starts at the beginning of each repetitive pattern and ends before the next repeat. Olera [58] tries to discover blocks with the record similar to the ones supplied by the user. Upon finding such blocks, it then generalises the pattern using multiple string alignment techniques. This helps Olera to extract those pages with only single data records which IEPAD fails to accomplish. Tresher [59] requires the help of user to highlight the examples of semantic contents as well as labelling them. These examples are then represented by a DOM tree on which a tree edit distance is used to generate the wrapper.

Labelled examples and human interactions are completely unnecessary in the unsupervised approach. RoadRunner [28] argues that a Web page is generated from a template by encoding the database values into HTML code. Extracting the data is considered as the decoding process. RoadRunner infers a universal wrapper for a set of HTML pages of the same class, i.e. pages that are generated by the same template, by iteratively comparing pairs of pages. The ACME (for *Align*, *Collapse under Matching*, and *Extract*) matching technique is used to compare the two pages in each iteration. The goal of the comparison is to align matched tokens as well as to collapse any mismatches. The mismatched tokens can be of two types: string mismatches or tag mismatches. String mismatches lead to the discovery of data values, while tag mismatches correspond to iterator or optional. The comparison process can yield several alignments, thus RoadRunner employs universal free regular expression to reduce the complexity. The alignments serve as wrappers and can be used to extract the data. ExAlg [44] deduces the unknown template from a given set of Web pages and the values to be extracted. ExAlg proposes two techniques, differentiating role and equivalence class, to detect the template. ExAlg also defines a token as either a word or HTML tag. The first technique relies on differentiating the role of a particular token. For example, the role of “Name” in “Book Name” and “Reviewer Name” is different. The latter technique finds a set of tokens that have exactly the same number of occurrences across the training pages. This is denoted as equivalence class (EC). The intuition is that a data tuple must be encompassed by a set of template tokens from the same equivalence class. ECs that do not have a minimum support, i.e. tokens in the EC do not appear in a sufficient number of pages, are removed. ECs must also have a minimum number of tokens to be accepted, otherwise, they are filtered. It is also necessary that all tokens in an EC are ordered and ECs are mutually nested. The original template is then reconstructed using the remaining valid ECs. DeLa [60] is an extension of IEPAD and tackle the

problem of nested object extraction. The wrapper generation process is conducted in two phase. In the first phase, data-rich sections of pairs of Web pages are identified based on the DOM tree. Nodes with identical sub-trees are removed. Next, repeated patterns are discovered using suffix trees. From each of the discovered patterns, the last occurrence of the pattern is used to find the new repeated pattern from the new sequence. A repeated pattern is enclosed inside a pair of brackets and closed with a "*" symbol as in regular expression. DeLa then selects the pattern with the most page support to extract the data.

## 2.2 Set Expansion

The goal of the set expansion task is to find elements of a semantic class given some examples (*seeds*) of its members. It has also been studied under various different names such as named entity extraction [61, 62], concept expansion [26, 63], or entity expansion [64, 65]. Most of the set expansion systems only consider the problem of extracting atomic values, while others try to extract binary or even $n$-ary relations. Generally, set expansion approaches implement a three-step framework. It starts with collecting the relevant documents from the data source taking into consideration the set of seeds provided by the user. The next step is to infer wrappers from the collected documents and extract the candidates using the generated wrapper. The last step is applying a ranking mechanism on the extracted candidates. Based on this framework, the set expansion system can be classified in terms of the data source, the target relations, the pattern construction, and the ranking mechanism.

### 2.2.1 Taxonomy Based on Data Source

In terms of the data source, set expansion systems can be viewed as two separate groups, i.e. corpus-based and Web-based systems. Corpus-based set expansion systems typically deal with building domain-specific semantic lexicons from a collection of specific domain textual documents. This is understandable because domain-specific documents are likely to contain specialised terminology. However, as the number of specific-domain corpora available is relatively small then the accuracy of those systems, although acceptable, are quite low. The World Wide Web or the deep Web, on the other hand, is a source of vast knowledge repository which is highly distributed with varying quality and granularity. It is then an excellent yet challenging source of data especially for finding and extracting specialised terminologies.

Corpus-based set expansions systems generally can only be applied to free text documents, thus require specialised natural language processing (NLP) techniques, including parsing, part-of-speech tagging (POS), named-entity-recognition (NER), etc. In [66, 67], the authors compute co-occurrence statistics of the text collection source to identify and extract candidate entities from the given set of seed entities. Syntactic relationships, such as Subject-Verb and Verb-Objects, are later used to extract specific elements from the corpus [68]. The previous approach is then extended to incorporate lexicon knowledge to generate patterns and extract the targeted rela-

Table 2.1: The set of seeds used in DIPRE

| **Author** | **Book Title** |
|---|---|
| Isaac Asimov | The Robots of Dawn |
| David Brin | Startide Rising |
| James Gleick | Chaos: Making a New Science |
| Charles Dickens | Great Expectations |
| William Shakespeare | The Comedy of Errors |

tions [29]. A few Web-based set expansion systems focus on extracting candidates of a semantic class by using hyponym patterns [69, 70], while others propose approach applied on specific type of Web documents, i.e. Weblogs [70], Web list and query logs [25], Web tables [26], and XML [33]. To extract the targeted entities or relations, several set expansion systems leverage the structural information of the Web page such as DIPRE [16], SEAL [24], Extended SEAL [30], RoadRunner [28], LyreTail [27], FOIL [34]. Igo et al. [71] take the advantages of both the corpus-based and Web-based techniques. From the set of seeds, they first expand a semantic lexicon on the corpus specific to the domain. Later, they calculate the Pointwise Mutual Information (PMI) between the candidates and the seeds based on queries to a search engine. The PMI scores are then used to filter out irrelevant candidates.

### 2.2.2 Taxonomy Based on Target Relations

From the point of view of target relations, many solutions have been proposed to extract atomic values (i.e. unary relation) from the data source, e.g. SEAL [24], SEISA [25], ConceptExpand [26], and LyreTail [27]. These systems focus on the task of either building a semantic lexicon [71, 70], identifying certain named entities [64, 62], expanding a semantic concept [26] or solving long-concept queries [63]. There are also some solutions that have been proposed to extract binary relations, e.g. DIPRE [16], RoadRunner [28], TextRunner [29], and Extended SEAL [30]. The proposed systems mainly approach the problem of identifying relations between pairs of entities by exploiting character-level features [30], structural information [16, 28], or distant supervision [22]. In comparison to unary and binary relation, $n$-ary relations have not been explored intensively. Only a few solutions have been proposed to tackle this problem by exploiting the graph [31, 32], XML [33], or unranked ordered tree [34] structure of the data.

### 2.2.3 Taxonomy Based on Pattern Construction

With regards to the pattern construction process, set expansion systems can be classified into several groups including Semantic Similarity (SS), Labelled Learning (LL) and Wrapper Inference (WI).

The basic assumption in the SS approach is that words describing the same concept would occur together in a similar context. SS approach computes the feature vector of all the words distribution or the surrounding words of a context window.

**Algorithm 1:** DIPRE's Algorithm

```
Input  : Set of binary seeds S and documents D
Output: Set of binary relations R
R = ∅; O = ∅;
R = R ∪ S;
while R is not large enough do
    foreach r ∈ R do
        O = O ∪ FindOccurrences(r, D);
    end
    P = InferPatterns(O);
    R' = ExtractCandidates(P, D);
    R = R ∪ R';
end
return R;
Function InferPatterns(O)
    O' = {Group occurrences of O based on the order and middle};
    P = ∅;
    foreach O_i ∈ O' do
        p =GenerateAPattern(O_i);
        if (specificity(p) satisfies Equation 2.2) then
            P = P ∪ p;
        else
            Group occurrences based on the URL: O'_i = {O_i1, O_i2, ..., O_in};
            O' = O' ∪ {O'_i};
        end
        O' = O' \{O_i};
    end
    return P;
end
Function GenerateAPattern(O = {o_1, o_2, ..., o_n})
    if (o_1.order=o_2.order=...=o_n.order) &
      (o_1.middle=o_2.middle=...=o_n.middle) then
        order = o_1.order;
        middle = o_1.middle;
        urlprefix = LongestCommonPrefix(o_1.url, o_2.url, ..., o_n.url);
        prefix = LongestCommonSuffix(o_1.prefix, o_2.prefix, ..., o_n.prefix);
        suffix = LongestCommonPrefix(o_1.suffix, o_2.suffix, ...,
          o_n.suffix);
    end
    return p;
end
```

To find candidates, SS calculates the similarity score of the seeds vectors against all of the other words vectors using specific metrics such as PMI [64], Cosine [66], etc.

Approaches in the LL category view set expansion problem as a binary classification. To be more precise, the binary classifier is trained using a set of *positive* examples from a particular class and responsible for assigning the class of *unlabelled* examples [72]. The Bayesian Sets [73, 74], which can be considered as a specialised LL, uses a *Reliable Negative Set* together with the positive examples to train the classifier.

Set expansion systems that fall into the Wrapper Inference category generate wrappers by exploiting the character-level features [24] or other structural information (HTML tags) [16]. Next, we look into the details of how both approaches infer the wrappers. In [16], Brin proposes Dual Iterative Pattern Relation Expansion (DIPRE) which looks for attribute-value pairs. In the paper, Brin shows how DIPRE extracts pairs of author and book title by giving a set of five seeds as shown in Table 2.1. DIPRE exploits the duality between patterns and target relations. Assuming that the set of seeds given by the user are good instances of the target relations, DIPRE will generate a good set of patterns. The set of instances found using the generated set of patterns are also good candidates for the target relations. The construction of the search string(s) from the given seeds and the search engine used by DIPRE are not specified by the author in the paper. Algorithm 1 shows the approach in DIPRE. From each of the Web pages collected, DIPRE locates all occurrences of all seed pairs of author and book title (line 4-6). Specifically, each occurrence of a seed pair is recorded as a 7-tuple. For instance, the 7-tuple for the author and book title example is of the form: <*author*, *title*, *order*, *url*, *prefix*, *middle*, *suffix*>. The value of *order* is equal to 1 if the occurrence of author precedes the occurrence of title, or 0 otherwise. The *url* denotes the URL of the Web page, while *prefix*, *suffix*, and *middle* are the characters preceding, following and separating the elements of the seed in the Web page. DIPRE limits the number of characters for the prefix and suffix to only 10 characters. DIPRE then infers patterns based on the set of occurrences (line 7). The patterns are then used to extract candidates from the Web page (line 8). This procedure can be repeated until the candidates are considered large enough.

The input to the pattern inference process in DIPRE is groups of occurrences ($O'$) based on their respective *order* and *middle* as shown in function *InferPatterns* of Algorithm 1. For each group ($O_i$) in $O'$, DIPRE tries to infer a pattern (line 15). If a pattern $p$ is successfully inferred then DIPRE calculates its $specificity$ using Equation 2.1. The specificity of a pattern $p$ is basically the length multiplication of all its components. DIPRE also checks the number of seeds ($n$) which can be retrieved using the pattern $p$. A pattern $p$ is considered as potential if it satisfies Equation 2.2 (line 16), where $t$ is a predefined threshold. Otherwise, $O_i$ is divided into groups based on the URL and the same process is repeated on each of the groups (line 19-20).

$$specificity(p) = |p.urlprefix||p.middle||p.prefix||p.suffix| \tag{2.1}$$

$$n > 1 \,\&\, specificity(p) * n > t \tag{2.2}$$

Function *GenerateAPattern* of Algorithm 1 shows how DIPRE infers a pattern $p$ from a set of occurrences ($O$) of the seeds. First of all, it ensures that all entries in $O$ have the same *order* and *middle* (line 28). If the condition is not satisfied then the procedure returns no pattern, otherwise, the *order* and *middle* of the pattern $p$ are set equal to $o_1.order$ and $o_1.middle$ (line 29-30). The *urlprefix* of the pattern is set as the longest common prefix of all the URLs of the set of occurrences. The *prefix* and *suffix* of the pattern are set to the longest common suffix and prefix of all the prefixes and suffixes respectively (line 31-33). DIPRE does not apply any ranking mechanism to the extracted candidates. Thus instead of a ranked list, it returns a set of candidates.

Wang et al. [24] propose Set Expander for Any Language (SEAL) to expand a set of atomic entities from a semi-structured corpus. The authors claim that SEAL is able to extract candidates regardless of the language used in the corpus. To retrieve relevant documents, SEAL uses the concatenation of all seeds as a keyword to the search engine (Google). For example, given the set of seeds of car manufacturers $\{Ford, Toyota, Nissan\}$, the query to the Google is "Ford Toyota Nissan". The search engine then returns a number of Web pages (URL) that contain the seeds but SEAL only considers the top $n$ pages.

SEAL generates specific wrappers for each Web page. Given the set of seeds, SEAL first locates the occurrences of all seeds in the Web page. Each of such occurrences is assigned a unique *id*. SEAL stores the contexts of an occurrence in a *trie* data structure. A *trie* data structure [75], also known as digital/radix/prefix tree, is a tree for storing strings in which there is one node for every common prefix. The strings are stored in extra leaf nodes. The *left* and *right* contexts (characters preceding and following the occurrence of a seed) are then inserted into a separate *left* and *right* trie. In the *left* trie, the context is stored inversely. Each node in the *left* trie maintains a list of *id*s which are preceded by the string from root to that particular node. SEAL defines a wrapper as a pair of maximally long common *left* and *right* context from which at least a seed is located. To find the maximally long common *left* contexts, SEAL traverses nodes on the *left* trie with a minimum of one entry in its list of *id*s and none of its child nodes has the same property. For each such longest common strings, SEAL traverses the *right* trie to find all the maximally long common *right* contexts. The wrapper generation process in SEAL is shown in Algorithm 2 where *seeds* and $l$ denote the set of seeds and minimum length of the context respectively. To rank the candidates, SEAL first builds a heterogeneous graph consisting of seeds, Web pages, wrappers, and candidates as the nodes and relationships among them as edges. A random walk [76] is then performed on the graph. After reaching the steady state, the weight assigned by the random walk to each node is used as the score on which the ranking is based on.

**Algorithm 2:** SEAL's Algorithm

```
Input  : A pair of trie (t1, t2)
Output: A set of wrappers W
foreach n1 ∈ TopNodes(t1,l) do
    foreach n2 ∈ BottomNodes(t2, n1) do
        foreach n1 ∈ BottomNodes(t1, n2) do
            w = ConstructWrapper(n1, n2);
            W = W ∪ w;
        end
    end
end
return W;
Function BottomNodes(Trie t, Node n')
    N = ∅;
    foreach n ∈ t do
        n_seeds = CommonSeeds(n, n');
        n_child = {children nodes of n};
        n_child_seeds = {∀n_i ∈ n_child & CommonSeeds(n_i, n') == |seeds|};
        if (n_seeds == |seeds|) & (n_child_seeds == ∅) then
            N = N ∪ n;
        end
    end
    return N;
end
Function TopNodes(Trie t, int l)
    N = ∅;
    foreach n ∈ t do
        n_parent = {parent node of n};
        text_n = {string from root to n};
        text_n_parent = {string from root to n_parent};
        if (|text_n| ≥ l) & (|text_n_parent| < l) then
            N = N ∪ n;
        end
    end
    return N;
end
Function CommonSeeds(Node n1, Node n2)
    R = ∅;
    I = n1.indexes ∩ n2.indexes;
    foreach i ∈ I do
        R = R ∪ {seeds at index i};
    end
    return |R|;
end
```

### 2.2.4 Taxonomy Based on Ranking Mechanism

The last step of the three steps framework for set expansion approach is to apply a ranking mechanism on the extracted candidates. However, there exists several set expansion systems that return a set of candidates instead of a ranked list of candidates. DIPRE [77], RoadRunner [28], TextRunner [29], FOIL [34] are examples of such system that do not apply any ranking mechanism. On the other hand, set expansion systems that do apply a ranking mechanism can be classified into two large groups, i.e. statistical-based and graph-based ranking. The former depends on specific metrics such as PMI [64], Jensen-Shannon [62], co-occurrence statistics [66, 67, 27], iterative similarity aggregation [25] to rank the extracted candidates, while the latter computes the similarity between the candidates and the seeds on a graph data structure [24, 26, 32, 31].

In [67], a context window is defined as the noun words directly preceding and following an occurrence of a seed. For the given context windows of a category, the category score for a word is calculated as the conditional probability of the word appears in a category context. In [62], an occurrence of a seed in a Web search query is represented as a search-query which is the concatenation of the remaining words preceding (prefix) and following (postfix) the seed in the query. This search-query is added as an entry to a query template vector and acts as a search-signature of the candidates with respect to the targeted class. Each entry in the vector is assigned a weight which corresponds to the number of times the query occurs in the query logs. A similarity function, i.e. Jensen-Shannon, is calculated between the search-signature vector of each candidate and search-signature vector of the class and used to rank the candidates. Sarmento et al. [66] define a membership function to rank candidate entities for the given set of seeds. The basic assumption of the approach is that members of the same semantic class tend to occur together in the text. Thus, the membership function is based on the co-occurrence statistics between the candidate and the set of seeds computed from the text corpus. LyreTail [27] measures the co-occurrence statistics between the candidates and the set of positive and negative seeds. If a candidate is likely to co-occur with the positive seeds then it is a positive candidate. On the contrary, if the candidate co-occurs often with the negative seeds, then it is not a positive candidate. Zhang et al. [63], rank candidate entities of a long concept based on its relevancy to shorter concepts. Entities that appear in more shorter concepts have higher relevancy scores thus are good candidates to expand the long concept. In [64], the authors represent terms (entities) as its NP chunks. The features of the terms are defined as the left and right most stemmed chunks. PMI score is calculated between a term $w$ and a feature $f$. A PMI vector is then constructed for each term by calculating its PMI against all possible features. The similarities between the terms and the seeds are measured using these PMI vectors and similarity metrics such as Cosine, Jaccard, and Dice. SEISA [25] is a set expansion system which specifically targets Web Lists and Query Logs. To evaluate the goodness of the candidates, SEISA applies an iterative similarity aggregation based on the previously mentioned metrics. The similarity aggregation is calculated in terms of relevance and coherence between the

set of seeds and the set of candidates. The relevance score is defined as the weighted similarity between the two sets. The coherence score is the similarity among the members of the candidate set. A set of candidates is considered as *good* if the members are similar to the seeds and they belong to a consistent concept.

In [68], nodes in the constructed graph represent nouns extracted from the corpus. An edge between nodes is created whenever the two represented nouns co-occur separated only by conjunctions "*and*" or "*or*". The frequency of the co-occurrence is used as the weight for the edges which has to satisfy a certain threshold. Thus, the weight is used as the ranking mechanism for candidates. Kozareva et al. [70] build a hyponym pattern linkage graph which consists of only candidates as the nodes. An edge between two nodes ($u$, $v$) denotes a "*generated by*" relationship between the two candidates ($v$ is generated by $u$). This edge is weighted with the number of times $u$ generates $v$. The authors also evaluate different scoring function that takes into account both the in-degree and out-degree of each node (Betweenness [78] and PageRank [77]). SEAL [24] considers a heterogeneous graph of seeds, Web pages, wrappers, and candidates. The edges of the graph represent relationships among vertices. A seed and a Web page have a *find* relationship because the Web page is retrieved by using the seed as the query to a search engine. A Web page and a wrapper have a *derive* relationship as a wrapper is inferred from the Web page. A wrapper and a candidate have an *extract* relationship because a wrapper is used to extract the candidate. SEAL calculates the rank of the candidates using a lazy walk process on the graph similar to that of PageRank [77]. ConceptExpand [26] takes as input an ad-hoc concept name and a few seeds of entities. It then uses the concept name as the keyword to query a Web table search system (e.g. Google's Web Tables[1] or Microsoft's Excel Power Query[2]). From the resulting tables along with the candidate entities, they build a bipartite graph of tables and entities mentioned on each table. A table is said to be exclusive if the entities contained are all relevant to the concept. Likewise, an entity is said to be relevant, if it is contained in at least one exclusive table. The relevancy of entities and the exclusivity of the tables are then inferred iteratively and used to rank the entities and the tables. Jayaram et al. [32] propose Graph Query by Example (GQBE), a system that queries knowledge graphs (e.g. DBpedia [79], Yago [52], Freebase [51], Probase [80]) when given examples of entity tuples. For instance, if one wants to find the name of the CEO of IT companies, he or she can give an example tuple such as <*Tim Cook*, *Apple*> to the system. GQBE captures the user's query intent by deriving a neighborhood graph from the knowledge graph based on the example tuple. Query graphs which contain all entities of the query are then identified from the neighborhood graph. For each of the query graphs, GQBE finds answer graphs which are edge-isomorphic to the query graph. The quality of an answer graph is based on two components, i.e. the structure and content score. Structure score of an answer graph is the total weight of all edges, while the content score is the total matching nodes. The system then returns a ranked list of candidate tuples based on the sum of the two scores (e.g. <*Mark Zuckerberg*, *Facebook*>, <*Sundar Pichai*, *Google*>, etc.).

[1] `https://research.google.com/tables`

[2] `http://office.microsoft.com/powerbi`

## 2.3 Truth Finding

Truth finding also referred to as truth discovery [81, 82, 83, 84], fact checking and data fusion [85, 86, 87, 88], solves the problem of automatically determining true (or actual) information and trustworthy sources by only analysing provided data values, source overlapping, and conflicts. It has numerous application domains such as data integration [89, 83], information retrieval [90], Open Linked Data [91], and set expansion [36].

Truth finding's main objective is to deterministically integrate conflicting statements and automatically compute a confidence value for each statement built on the trustworthiness level of its sources. The trustworthiness level is itself unknown in general and therefore should be estimated by the process. This leads to an iterative algorithm with the specified convergence criteria. At convergence, the truth finding function determines the truth as the statements with the highest confidence scores. Example 1 shows and explains a general application case of truth finding.

**Example 1.** *As an example, consider three sources which give statements about the capital city and the currency of the country Algeria. Source_1 and Source_3 state that "Alger" and "Dinar Algerian" are the correct values while Source_2 gives "Alger" and "Leka Algerian". We can observe that while all sources agree on the capital city, they disagree on the currency. A typical majority voting-based process will automatically break down this disagreement by choosing "Dinar Algerian" as the correct value for the currency of Algeria because it is voted by 2 out of the 3 sources. The accuracy of each source is then calculated as the proportion of statements given by the source which conform to the correct value, while the confidence of each statement is defined as the weighted accuracy of sources that support the statement. Thus, the accuracy values of Source_1, Source_2, and Source_3 are 1, 1/2, and 1 respectively, while the confidence scores for "Alger", "Leka Algerian", and "Dinar Algerian" are set to 5/6, 1, and 1/2.*

### 2.3.1 Taxonomy of Truth Finding Approach

Majority Voting is a naive truth finding model. It assumes an identical source trustworthiness and takes the values provided by the majority as the truth. Such a straightforward and very intuitive approach, however, does not capture the different levels of trustworthiness of the sources, especially Web sources. This has led to the adoption of a weighted voting strategy, with weights being the quality levels of the input sources, by most of the truth finding algorithms. In this setting, as source quality is unknown a priori, it is estimated based on the level of the truthfulness of the provided statements, yielding an iterative process. According to the manner, such an iterative process is implemented, one can classify, to some extent, truth finding algorithms as agreement-based, MAP estimation-based, and Bayesian-inference method.

Algorithms in the agreement-based class use the notion of majority to iteratively compute source trustworthiness and confidence score until convergence. In [81], Yin et al. propose TruthFinder which extends majority voting. It considers a

similarity metric in its weighting model by leveraging the intuition that the levels of the truthfulness of two similar values should influence each other in a certain sense. An important feature of the truth is *hardness*. Providing the correct values for *easy* object attributes should not increase significantly the trustworthiness of the sources. In contrast, when the attributes are *hard*, sources that provide the corresponding correct values should become more trustworthy as proposed by Galland et al. in [92]. The authors propose three algorithms: Cosine, 2-Estimates, and 3-Estimates which differ in the series of parameters estimated. The first two algorithms only estimate the correct value of facts and the trustworthiness of the sources, while the last algorithm also estimates the error of each fact.

All of the approaches in MAP estimation-based class are built on Maximum A Posteriori (MAP) paradigm. In LTM [83], the authors regard a source as a classifier because each source makes true/false claims for the statements. The quality of the sources is represented by the *sensitivity* and *specificity*. The two features are associated with the false negatives and false positives made by the sources respectively. Both quality features and the truth values of the facts are modelled as latent variables in a Bayesian probabilistic graphical model. Given the observed data, these latent variables are then approximated using MAP estimation which maximises the posterior likelihoods of the corresponding latent variables. MLE [93] is applicable to the crowd or social sensing. Any observations made by a user is modelled as binary variables. The default state of each variable is always "negative". The truth finding task in this scenario is transformed into a maximum likelihood estimation problem. The problem is then solved using the Expectation-Maximization (EM) algorithm. LCA models [94] assume that the truth values of facts are latent variables, while the credibility of sources is represented by a set of modelled parameters including honesty, difficulty, mistake, and lying. SimpleLCA considers honesty of sources which is the probability of the source being honest. GuessLCA takes into consideration the difficulty of telling the truth. When telling the truth becomes harder, some sources tend to guess to make the claims. MistakeLCA asserts how likely an honest source make a mistake. LieLCA differentiates between sources that intentionally lie and that of making an honest mistake. These parameters give different aspects of credibility to the sources.

In Bayesian-inference methods, the family of algorithms is based on the Bayesian analysis [95]. Depen model and four variants have been proposed in [89, 82]. Depen investigates and finds, based on a Bayesian analysis, source dependence via copying relationships. The copying relationships should not significantly increase the confidence in any information and more credit must be given to data from independent sources. Accu algorithm extends Depen by introducing different source accuracy level. It then estimates the level of accuracy of a source as a function of the vote count of their provided values. The other variants of Depen, i.e. AccuNoDep, AccuSim, and AccuPr, relax the source dependence assumption while also consider value similarity and source popularity respectively. Sources are not only correlated by dependence, but negative correlations can also occur, as studied by the authors in [87]. In [96, 97], Fang et al. propose a multi-valued truth finding model called SmartMTD (Smart Multi-valued Truth Discovery) that considers four

important implications to conduct truth discovery. Two graphs are constructed and further used to derive two aspects of source reliability via random walk computations. Fang also presents in [96] a general approach, which utilises Markov chain models with Bayesian inference, for comparing the existing truth discovery methods. Liu et al. applied truth finding on Open Linked Data. They demonstrated in [91] TruthDiscover which is a novel system that identifies the truth in Linked Data with a scale-free property. TruthDiscover estimates the prior beliefs of sources by leveraging topological properties of the Source Belief Graph. The estimates are then utilised to smooth the trustworthiness of the sources. To model inter-dependencies among objects which are used for accurately estimating the trust values of objects, TruthDiscover exploits Hidden Markov Random Field.

### 2.3.2 Ensemble Approach

The main limitation of the current truth discovery approaches is that none of them outperforms the others in all application scenarios. Consequently, Ba et al. have considered in [98] possible structural correlations among data attributes to present a complementary study of the proposed truth discovering algorithms. Any specific truth finding algorithm can be used in their model. In the same spirit, VERA [90] is a Web platform enabling to verify the veracity of Web facts (of numerical types) using an ensembling of several truth finding algorithms. It combines the best of several algorithms which enables to overcome the weakness of each of them [99]. [90] employs, for the same purposes, existing truth finding algorithms as [99].

## 2.4 Topic Labelling

Every document has an underlying topic which the author based their content on. For example, upon writing a scientific paper, a researcher may choose a broad topic such as "machine learning" or a very specific one like "topic labelling" to base the content of the paper. However, it is rarely seen that a document only influenced by a single topic. Instead, most of the time the document is built on a mixture of topics. In the case of the scientific paper, the writer may also write some applications of the machine learning algorithms in "biology" or "medicine". This is the general assumption adopted by topic modelling algorithms.

Topic modelling approaches learn the "abstract" topics that appear in a collection of documents. One of the most well-known topic modelling algorithms is Latent Dirichlet Allocation (LDA) [1]. LDA considers every topic as a distribution over the top-$N$ words and every document as a distribution over $K$ topics where $K$ and $N$ are input to the model. This makes LDA a probabilistic clustering technique. Intuitively, LDA works on co-occurrence of words in the documents. If a certain set of words co-occur in multiple documents then this set gets a higher probability for a certain topic then the rest of the words. Additionally, there is an intrinsic duality in the LDA: the documents is a distribution over topics whereas a topic is also considered as a distribution over documents. The plate diagram such as shown

in Figure 2.1 is an illustrative way of explaining the LDA model. The boxes in the diagram denote "plates", whereas plate here represents replication. The $D$ plate is the corpus or the collection of documents, the $M$ plate represents the repeated choice of topics and words in a document, and $K$ denotes the topics.

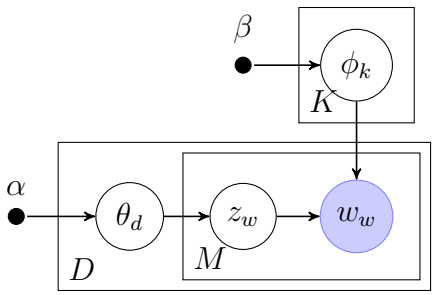


Figure 2.1: Plate diagram of LDA [1]

To generate a new document based on this model, one must first decide the number of words $M$ that the new document contains. We can draw $M$ from a Poisson distribution [100]. Next, select a mixture of $K$ topics for the new document. The distribution of the $K$ topics is drawn from a Dirichlet distribution. Let us assume $K$=3. A draw from the Dirichlet distribution would return 0.3, 0.5, 0.2 where the mixture of topics is 30%, 50%, and 20% for Topic 1, Topic 2, and Topic 3 respectively. The next step is we generate each word in the new document by first selecting the topic (from the multinomial distribution of topics we sampled before), and then randomly choose the word under the selected topic (according to the topic's distribution of words). In reality, we are only presented with the collection of documents. Assuming that the collection of documents are generated by this generative model, LDA then tries to learn the topic representation of each document as well as the words associated with each topic. There are several methods that can be used to achieve this goal including variational Bayes [1], Gibbs sampling [101] and expectation propagation [102].

As we have previously mentioned, LDA represents a topic via the top-$N$ words with refers to their marginal probability in that topic. For example, {*charles, prince, king, diana, royal, queen, family, parker, british, bowles*} is a set of words which describes a topic. The set of words are extracted from a corpus of news articles. Unfortunately, LDA does not provide us with the label for the topic. Although one can guess what the topic is about from the words (e.g. "British royal family"), it requires a significant cognitive knowledge with the risk of subjectivity. Thus, automatically finding an appropriate label for the topic model is more preferable.

Automatically labelling a topic model is the task of topic labelling. The task was proposed by Mei et al. [103]. In the paper, they extracted bigram phrases from the corpus. They use these bigrams as the candidate labels. To decide which bigram phrases to extract, they applied a lexical association measure. Next, they calculated the Kullback-Leibler (KL) divergence [104] between pairs of label candidate and topic and used them to rank the labels. Lau et al. [105] applied a different approach of extracting candidate labels. They queried Wikipedia and also performed a site-

restricted search on Google using the top-$N$ topic words as the keywords. The initial set of primary candidate labels were constituted from the articles titles of the search result. A set of secondary candidate labels were then generated from each of the primary candidates. They first extracted noun chunk from the label by using OpenNLP chunker[3]. *n-grams* were then generated from only these noun chunks. They discarded *n-grams* which were not article titles in Wikipedia. RACO scores [106] were then computed for each pair of secondary and primary labels within the same topic and used to remove unrelated labels. The ranking process takes into consideration features from the candidate labels. These features include lexical association measures, the raw number of words, and the relative number of words in the label candidate that are in the set of topic words. They experimented with an unsupervised model by using each feature individually to rank the label candidates and then selecting the top-$N$. The combination of the features is used as input to a support vector regression model. The model was trained over topics of which they have the gold-standard labels for the candidate labels.

Bhatia et al. [107] represented a Wikipedia title by its documents as well as words embedding. The model was built using the entire collection of English Wikipedia as the training set for doc2vec [108] and word2vec [109]. The candidate labels were ranked using a supervised learn-to-rank model. They trained a support vector regression model using four features including cosine similarity of letter trigram vector of the label and topic words, PageRank value of each label on a directed graph based on hyperlinks found within the article text, number of words in the candidate label, as well as the lexical overlap between the candidate label and the top-$N$ topic words. Kou et al. [110] extracted phrases which contain the top-$N$ words of a topic from a collection of topic-related documents as the candidate labels. They mapped both topics and candidate labels to vector space and ranked them based on the cosine similarity. Hulpus et al. [111] constructed a graph from DBpedia concepts corresponding to the top-$N$ words of each topic. They removed noised from the graph based on connectivity. They then calculated defined several graph centrality measures which are based on the closeness and betweenness [112], information [113] as well as random walk betweenness [114] centrality. The best label for a topic was then selected using these graph centrality measures. Automatic topic labelling has also been applied to Twitter data [115] by first calculating words relevance of documents. The documents in consideration must be related to the topic. Next, a summarisation algorithm was used to generate the relevant labels.

## 2.5 Multi-label Classification

Classification is the task in machine learning which assigns unknown objects to one of several predefined categories (labels) by learning a *classifier* from a training dataset. The training dataset contains a set of observations whose category of membership is known. The classification task can be applied in diverse areas such as categorizing emails as “spam” or “not spam”, identifying a new species of animal as either

[3] `http://opennlp.soundforge.net`

"mammal", "reptile", "bird", "fish" or "amphibian" based on the characteristics or features of the email and animal respectively. In the first example, there are only two categories ("spam" or "not spam") which one of them can be assigned to unknown objects. This is the typical setting of the classification task, i.e. *binary classification*. In the second example, there are more than two categories ("mammal", "reptile", "bird", "fish" or "amphibian"), but still, an animal can only be classified to one of the categories. This setting is referred to as *multi-class classification*. Consider the case where the classifier is asked to predict the genre of a song. There are plenty of music genres such as "rock", "pop", "country", "jazz", etc. to choose from. A song cannot be exclusively categorized into one of the genres but instead into multiple categories. This type of setting is known as *multi-label classification*.

In multi-label classification, multiple predictions must be made for each data object, corresponding to the number of labels. Each label takes a binary value, where a 1 represents the presence or *relevance* of a label, and a 0 otherwise. Previous works on multi-label classification are often categorized into two approaches, i.e. *problem transformation*, and *algorithm adaptation*. The first approach remodels the multi-label problem as one or more single-label problems. Any *off-the-shelf* single-label classifier can then be chosen to make the single-label classifications. The predictions made by the single-label classifiers are then mapped into multi-label representations. Naive Bayes [116] and Support Vector Machine [117] are some of the algorithms employed in earlier research using problem transformation approach. In the latter approach, an existing single-label algorithm is modified to fit the purpose of multi-label classification. Decision trees [118] and AdaBoost [119] are among the earliest single-label algorithms adapted to multi-label classification problem. Problem transformation approach is more preferable than algorithm adaptation due to its flexibility.

*Binary relevance* method (BM) [117, 120] is one of the most recognized problem transformation methods. To present the BM method we first formalize the multi-label classification problem. Consider $X^d \subset \mathbb{R}$ as the input domain of all possible attribute values. To represent an object $\mathbf{x}$, we use the vector notation $\mathbf{x} = [x_1, ..., x_d]$, where $d$ is the number of attribute values. The output domain of all possible labels is represented by the set $\mathcal{L} = \{1, ..., L\}$. An $L$-vector $\mathbf{y} = [y_1, ..., y_L]$, is used to represent labels which are associated with $\mathbf{x}$. The value of each $y_j$ is equal to 1 if object $\mathbf{x}$ is associated with label $j$, and 0 otherwise. For a training dataset $D$ of $N$ labelled examples, we represent it as $D = \{(\mathbf{x}^i, \mathbf{y}^i) | i = 1, ..., N\}$. The superscripts $i$ and $j$ are used to differentiate between the data and label dimension. Thus, $y_j^i$ corresponds to the binary relevance of label $j$ for the $i$-th example. BM method trains a classifier $\mathbf{c}$ from $D$. $\mathbf{c}$ itself is composed of $L$ binary classifiers, where each $c_j$ predicts whether $\mathbf{x}$ is associated with label $j$ or not. Hence, the output of $\mathbf{c}$ for any object $\mathbf{x}$ is the vector $\hat{\mathbf{y}} \in \{0, 1\}^L$.

Although BM is arguably one of the most popular problem transformation methods, it is consistently cast aside. The reason is that it is believed that BM does not capture the relationships among the labels, i.e. it assumes that labels are independent. Any correlations between labels that exist in the training data are lost during the transformation process. This affects the resulting set of labels predicted by BM

| ID | Description | | | | | Skin Care | Anti-aging | Nail Care | Nail Polish |
|---|---|---|---|---|---|---|---|---|---|
| $\mathbf{x}$ | $x_1$ | $x_2$ | $x_3$ | $x_4$ | $x_5$ | $y_1$ | $y_2$ | $y_3$ | $y_4$ |
| 1 | 0 | 1 | 1 | 1 | 0 | 1 | 1 | 0 | 0 |
| 2 | 1 | 1 | 1 | 0 | 1 | 0 | 1 | 1 | 0 |
| 3 | 1 | 0 | 1 | 0 | 1 | 0 | 1 | 0 | 0 |
| 4 | 0 | 0 | 1 | 0 | 1 | 0 | 1 | 0 | 0 |
| 5 | 1 | 1 | 1 | 0 | 1 | 0 | 0 | 1 | 1 |

Table 2.2: A small example of multi-label dataset based on the Yves Rocher set

which can contain too few or too many labels and sometimes produce labels that are unlikely to occur together in the real world. But Read et al. [121] claim that using their proposed chaining method the BM-based methods can overcome the previous concerns.

As in BM, the Classifier Chain model (CC) trains $L$ binary classifiers. The fundamental difference between the two methods is that in BM, each classifier $c_j$, which is responsible for the binary relevance problem for label $j \in \mathcal{L}$, is independent to each other, while in CC, a chain connects all classifiers. In the learning phase for CC, each classifier $c_j$ is trained sequentially using a transformed dataset $((\mathbf{x}, y_1, ..., y_{j-1}), y_j)$. Suppose that we have a dataset as in Table 2.2. To train $c_1$ for label $y_1$ we use objects $\mathbf{x}^1$, ..., $\mathbf{x}^5$, where the initial input space for each object consists of $x_1$, ..., $x_5$. Next, to train $c_2$, we augment $y_1$ to the initial input space of each object. We do the same process for all $c_j$ by adding $y_1, ..., y_{j-1}$ to the initial input space. In general, to train $c_j$ we add $y_1$, ... $y_{j-1}$ to the initial input space. The classification process is done in a similar fashion. Using the feature vector of an unknown object as the input space for $c_1$, we first predict $y_1$. We add the predicted value of $y_1$ to the initial input space and use it as the input for $c_2$ to predict $y_2$. This process is repeated until we have predicted all $y_j$.

# Part I

# Finding the Truth in Set of Tuples Expansion

# Chapter 3

# Set of Tuples Expansion

The task of set expansion consists of extracting, from a corpus, elements of a particular semantic class characterised by some examples of its members. More precisely, given a set of examples, also referred to as *seeds* (e.g. names) characterising a particular semantic class (e.g. US Presidents) and a collection of documents (e.g. HTML pages), the set expansion problem is to extract more elements of the particular semantic class from the collection of documents. For example, we may want to find the names of presidents of the United States of America by giving the three examples Barrack Obama, George Bush, and Bill Clinton as seeds. The goal here is to extract the names of all the other US Presidents from the Web.

Set expansion approaches use a three steps framework. The first step is crawling. Given the set of seeds, crawling finds relevant documents containing the seeds. The different methods of composing the queries from the seeds can be applied to crawl the search engine and collect the Web pages. The next step is the wrapper generation and candidate extraction. The former involves finding all occurrences of the seeds in the collected Web pages. It then identifies patterns (wrappers) from contexts surrounding the occurrences of the seeds. The latter uses these wrappers to find and extract other possible candidates from the Web pages. The last step is ranking. The wrapper generation and candidate extraction processes are likely to extract extraneous and erroneous data because of the imperfection of the patterns inferred. Ranking the candidates according to their plausibility is then paramount to presenting satisfactory results to users.

Existing set expansion approaches (DIPRE [16], SEAL [24]) largely deals with atomic seeds. Each seed in the set of examples given by the user consists of only one element. However, most of the time, a user demands information in a form of relation instances much like in the relational model of a database. In this case, a relation instance or tuple is composed of a set of atomic values. The aforementioned set expansion approaches need to be generalised to tackle this problem. We propose our approach of extending the set expansion problem to the expansion of sets of tuples. This chapter is organised as follows. We start with a formal definition of our proposed task of the set of tuples expansion in Section 3.1. Section 3.2 explains in details our approach of the set of tuples expansion. We show the experiment results in Section 3.3. We conclude the chapter by highlighting the work we have done and

introduce some future works in Section 3.5.

## 3.1 Problem Definition

To be precise, we formulate the task of the set of tuples expansion as follows. Let $D$ be a collection of documents (*corpus*), while $S$ is the set of all semantic classes. A member (*element*) of a semantic class is denoted by $e$ where $e \in s_i$ and $e \notin S \setminus s_i$. The previous definition is basically saying that $e$ can only be a member of exactly one semantic class. A tuple, $T$, is a set of $n$ elements ($n > 1$) and each of these elements belongs to a different semantic class. When given the set of seeds $R = \{T_1, T_2, ..., T_{Ns}\}$, the goal of set of tuples expansion is to extract a candidate set of tuples, $\mathcal{T} = \{T'_1, T'_2, ..., T'_{Nc}\}$, from $D$. $Nc$ and $Ns$ are the size of the seeds and candidate set respectively, where $Nc \geq 2$ and generally $Nc > Ns$. As a running example, consider the following set of seeds, $R = \{$<*Indonesia*, *Jakarta*, *Indonesian Rupiah*>, <*Singapore*, *Singapore*, *Singapore Dollar*>$\}$. Each tuple in $R$ consists of three elements. The first, second, and third element of the tuples belong to the semantic class of "country", "capital city", and "currency name" respectively. Each element of the tuples in the candidate set $\mathcal{T}$ must also be in the same semantic class as the element of the seeds.

## 3.2 Proposed Approach

We propose to design, implement and evaluate an extension of the set expansion approach to set of tuples expansion (STEP). The generalisation of the set expansion raises new challenges in every stage of the set expansion framework: retrieving the relevant documents, wrapper generation to extract the candidate tuples, and ranking of the candidate tuples. The challenges mainly arise due to the fact that the seeds are constituted of multiple elements (*composite seeds*) and each of the element may appear arbitrarily on a Web page. Furthermore, there is no consistent structure that separates each element of the seed on the Web page. We tackle this problem by adopting *regular expression*-based wrapper where we leverage a slightly modified version of the edit distance algorithm (Section 3.2.2). To rank the candidate, we base our mechanism on SEAL [24] where we add a new node (*domain*) and its set of relations (Section 3.2.3).

### 3.2.1 Crawling

Considering composite seeds rather than atomic ones raises the problem of the order of the elements in the seed. The issue is neither discussed by the authors of [16] nor of [24] for crawling. Indeed, search engines like Google are relatively robust when given keywords in different orders. This robustness, however, suffers from long lists of keywords as it happens with seeds composed of numerous elements. We have tried and considered search strings formed with permutations of the elements of the seeds. Although it can improve the performance in some cases, we are not considering such

permutations due to the high number of possible permutations. In the wrapper generation phase (Section 3.2.2), however, such permutations is important in finding the occurrences of the seeds. We use Google search engine to collect Web pages. The search query is the concatenation of all the elements of all the seeds. For the set of seeds <*Indonesia*, *Jakarta*, *Indonesian Rupiah*>, <*Singapore*, *Singapore*, *Singapore Dollar*>, the input query for Google is "Indonesia" + "Jakarta" + "Indonesian Rupiah" + "Singapore" + "Singapore" + "Singapore Dollar"'.

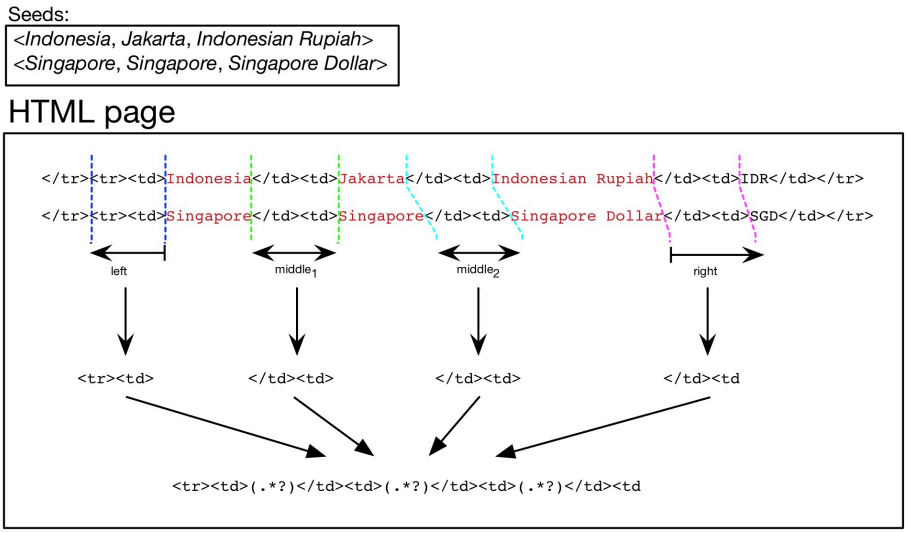


Figure 3.1: Wrapper generation process

### 3.2.2 Wrapper Generation and Candidates Extraction

We want to extract the candidates from documents (Web pages) in our corpus. For the sake of simplicity, in this work without loss of generality, we consider that the Web pages are HTML documents. In order to be able to extract candidates from Web pages, we must first infer the underlying wrapper. This wrapper generally consists of *contexts* (characters) surrounding the attributes of the given seeds and the candidate tuples that are going to be fetched later. DIPRE [16] enforces a constraint that a wrapper can only be generated if it brackets all the occurrence of the seeds on the pages. This is a really strict constraint that drastically decreases the *recall* (the percentage of correct results among all possible correct results) of the system. SEAL [24], on the other hand, relaxes the constraint while constructing the wrappers. A wrapper is generated if it brackets at least one occurrence of each seed on a page. By relaxing the constraint, SEAL outperforms DIPRE, especially in the *recall*. Based on this, we adopt SEAL's approach to wrapper induction. However, SEAL only works on seeds with atomic values or binary relations. For seeds with more than one element, we must first find the contexts between each element, infer the wrapper and use the inferred wrapper to extract the candidates. An illustration of the wrapper generation process can be seen from Figure 3.1 where we have a set

of seeds $R$, each of which has $n$ elements and a document $d$ that contains the seeds. Each occurrence of a seed is represented by a *left* and *right* context, as well as $n$-1 *middle* contexts. We generate the wrapper by finding the longest common string for the left and right contexts and obtaining the common regular expression for the middle contexts from the set of occurrences. As can be seen from Figure 3.1, the *left* contexts for both seeds are the same which is "r><tr><td>". The $middle_1$ and $middle_2$ contexts are also the same, while the right contexts only differ in the last character ("I" and "S"). The result of comparing the *left*, $middle_1$, $middle_2$, and *right* contexts are "r><tr><td>", "</td><td>", "</td><td>", and "</td><td>" respectively. The detail of the process is explained next.

Algorithm 3 shows how we find the occurrences of the seeds. To find an occurrence of a tuple seed with $n$ elements in a Web page, we use a regular expression in the form of (.{1, *max_context_length*})$e_1$(.*)$e_2$(.*) ... $e_n$(.{1, *max_-context_length*}) (line 2) where *max_context_length* is a variable controlling the number of characters. The order of each element $e_i$ in the regular expression is the same as the order of each element of the tuple seeds. Each occurrence of a seed is denoted by a ($n$+1)-tuple: <*left*, $middle_1$, $middle_2$, ..., $middle_{n-1}$, *right*>. The *left* represents a sequence of characters *max_context_length* preceding the first element ($e_1$) (line 5), *right* represents a sequence of *max_context_length* characters following the last element ($e_n$) (line 6), and $middle_i$ represents a sequence of characters between the $i^{th}$ and the $(i+1)^{th}$ element of this occurrence (line 9-12).

**Algorithm 3:** FindSeedsOccurrences

**Input :** A set of seeds $R = \{T_1, T_2, ..., T_{Ns}\}$, a document $d$
**Output:** A set of left contexts $LC$, a set of right contexts $RC$, and sets of middle contexts $MC_1, MC_2, ..., MC_{n-1}$

```
1  foreach T_i ∈ R do
2      O_i = {occurrences of T_i in d};
3      if O_i ≠ ∅ then
4          foreach o ∈ O_i do
5              left = {characters of length max_context_length preceding e_1};
6              right = {characters of length max_context_length following e_n};
7              LC = LC ∪ left;
8              RC = RC ∪ right;
9              for i ← 1 to n − 1 do
10                 middle_i = {characters between i and i+1 element};
11                 MC_i = MC_i ∪ middle_i;
12             end
13         end
14     end
15 end
16 return LC, RC, MC_1, ..., MC_N;
```

Once all occurrences of each of the tuple seeds are found, we then try to infer

pattern from each of the set of contexts as shown in Algorithm 4. We do a character-wise comparison for pairs of left and right contexts (line 1-8) and take the longest common string as the result ($LCWrap$, $RCWrap$). For the right context, we do the comparison from left to right while for the left part we have to inverse the process. As for the middle contexts, we generate a common regular expression from pairs of the context using a slightly modified edit distance algorithm as shown in Algorithm 5 (line 9-14). The results are denoted as $MCWrap_1$, $MCWrap_2$, ..., $MCWrap_{n-1}$ for each of the middle contexts. The set of wrappers is then defined as the combinations of each of the $LCWrap$, $MCWrap_1$, $MCWrap_2$, ..., $MCWrap_{n-1}$, and $RCWrap$ (line 15). We use the generated wrapper to extract candidates from the Web page. The extraction process is fairly straightforward. We add a capturing group, denoted by (.*?) in the regular expression, in between the $LCWrap$, $MCWrap_1$, $MCWrap_2$, ..., $MCWrap_{n-1}$, and $RCWrap$. We then use regular expression parser to search for its occurrences in the Web page and extract the intended information from the capturing group.

**Algorithm 4:** WrapperGeneration

**Input :** A set of left contexts $LC$, a set of right contexts $RC$, and sets of middle contexts $MC_1, MC_2, ..., MC_{n-1}$

**Output:** A set of wrappers $Wrappers$

```
1  foreach pairs of lc ∈ LC do
2      LCSLeft = LongestCommonSuffix(lc);
3      LCWrap = LCWrap ∪ LCSLeft;
4  end
5  foreach pairs of rc ∈ RC do
6      LCPRight = LongestCommonPrefix(rc);
7      RCWrap = RCWrap ∪ LCPRight;
8  end
9  for i ← 1 to n − 1 do
10     foreach pairs of mc ∈ MC_i do
11         MCRegex = CommonRegularExpression(mc);
12         MCWrap_i = MCWrap_i ∪ MCRegex;
13     end
14 end
15 Wrappers = {combinations of LCWrap, MCWrap_1, MCWrap_2, ...,
       MCWrap_{n−1}, RCWrap};
16 return Wrappers;
```

Algorithm 5 shows the process of generating the common regular expression for pairs of the middle contexts. The algorithm is based on the edit distance (Levenshtein distance [122]) algorithm. The true purpose of the edit distance algorithm is to calculate the minimum number of operations (insert, update, delete) needed to transform a string A to string B. In our perspective, however, this algorithm can be used to generate a common regular expression of two strings by replacing when-

**Algorithm 5:** CommonRegularExpression

```
Input  : Two strings s[1..m], t[1..n]
Output: A regular expression
for i ← 1 to m do
    d[i, 0] ← i;
end
for j ← 1 to n do
    d[0, j] ← j;
end
min ← 0; result ← "";
for j ← 1 to n do
    currmin ← min;
    min ← length(s)+1;
    for i ← 1 to m do
        if s[i] = t[j] then
            d[i, j] ← d[i − 1, j − 1];
        else
            d[i, j] ← minimum(d[i − 1, j] + 1, d[i, j − 1] + 1, d[i − 1, j − 1] + cost);
        end
        if d[i, j] < min then
            min ← d[i, j];
        end
    end
    if min > currmin then
        result ← result + ".";
    else
        result ← result + t[i − 1];
    end
end
return result;
```

Table 3.1: An example of calculating edit distance of two strings

| | | < | / | a | > | _ |
|---|---|---|---|---|---|---|
| | **0** | 1 | 2 | 3 | 4 | 5 |
| : | 1 | **1** | 2 | 3 | 4 | 5 |
| _ | 2 | **2** | 2 | 3 | 4 | 4 |
| < | 3 | **2** | 3 | 3 | 4 | 5 |
| / | 4 | 3 | **2** | 3 | 4 | 5 |
| b | 5 | 4 | 3 | **3** | 4 | 5 |
| > | 6 | 5 | 4 | 4 | **3** | 4 |
| _ | 7 | 6 | 5 | 5 | 4 | **3** |

ever there is an operation of insert, update or delete with a ".". The dot character in regular expression stands for any character. The input to the algorithm is two strings with size $m$ and $n$. The algorithm then starts with the initialization of an $m * n$ matrix $d$ (line 1-6). The usual edit distance process is then carried out (line 8-19) with the exception of line 9-10. We add the two lines of code to calculate the current minimum score of the $j$-th row. We add a check (line 21-25) to see whether the current row's minimum score is greater than the previous row's minimum. If yes, this means that there is an operation performed on the current row and thus we add a "." to the *result* string. Otherwise, the "edit cost" for the current row is the same as in the previous one, then we just add the current character to the *result* string. To illustrate the idea, let us use ":_</b>_" and "</a>_" as the two input strings. The edit distance matrix for these two strings is shown in Table 3.1. We start at the top left corner of the matrix where the value is 0 and this is what we call the current minimum (*currmin*). Next, on row 1, we check that the minimum value of row 1 is 1 and this is greater than the *currmin* which is 0, thus we can conclude that there is an operation performed here. We add "." to the result and set *currmin* to 1. Next, in row 2, the minimum value is 2 and this is greater than the *currmin*, thus we add another "." to the result. We continue finding the minimum value of each row and compare it with the *currmin* until we reach the last row. The result of the algorithm for the example above is "..</.>_".

We briefly mentioned in Section 3.2 that one of the challenges that arise due to the composite seeds is that each of the element of the seed may appear arbitrarily on a Web page. Consider the seed, <*Indonesia*, *Jakarta*, *Indonesian Rupiah*>. The regular expression used by Algorithm 4 to locate the occurrences of the previous seed would be: "(.{1, *max_context_length*})Indonesia(.*)Jakarta(.*)Indonesian Rupiah(.{1, *max_context_length*})". This regular expression would fail to detect an occurrence of the seed if the element 'Indonesian Rupiah' appears at the beginning of the Web page and then is followed by 'Indonesia' and 'Jakarta' respectively. If we do not consider such a case then we would not find any occurrence of the tuple in the Web page and can not generate any wrapper. To tackle this problem, we also consider locating the occurrences of the permutations of elements of the seed. Algorithm 6 shows how we generate wrappers for all of the possible permutations of the elements of the seed. For the set of seeds with $n$ elements, we first generate all the possible permutation of order $n$ (line 1). We denote the order of the elements from the given seeds as the "actual" order. Next, for each generated order $o$, we reorder the elements of each seed accordingly (line 4-7). Using the new set of seeds

we then infer the wrapper from the Web page (line 8). We return as the output of the algorithm pairs of the inferred wrappers and the order of elements of the seed which is used to infer the wrappers (line 10). We refer the wrappers generated by Algorithm 6 as *permutated wrappers*. All of the candidates extracted using these permutated wrappers need to put back into the order of the given seeds.

**Algorithm 6:** GenerateWrappersWithPermutation

**Input :** A set of seeds $R = \{T_1, T_2, ..., T_{Ns}\}$, a document $d$
**Output:** A set of left contexts $LC$, a set of right contexts $RC$, and sets of middle contexts $MC_1, MC_2, ..., MC_{n-1}$

```
1  Order = {generate permutation of order n};
2  foreach o ∈ Order do
3      R_new = ∅;
4      foreach r ∈ R do
5          r' = Reorder(r, o);
6          R_new = R_new ∪ r';
7      end
8      Wrappers =
         {WrapperGeneration(FindSeedsOccurrences(R_new, d))};
9  end
10 return Wrappers, Order;
```

### 3.2.3 Ranking

We adopt the ranking mechanism introduced in SEAL [24]. The nodes in the graph represent entities, i.e. seeds, Web page, wrapper, candidate and domain. The domain entity shows to which a Web page belongs to. This type of entity is not mandatory in the graph used by SEAL. We argue differently because this relationship is important to produce a ranked list of domains which are relevant to the seeds. This is byproducts of our ranking mechanism where we can not only answer the previous query but also others such as the most relevant Web pages to the seeds, etc. An edge relates two entities if there is a relationship between them. For example, an edge exists from seeds to Web pages because the corresponding Web pages are retrieved by using the seeds as the query to search engine. The entities and its relationships used in our graph of entities are summarised in Table 3.2.

The process of building the graph of entities is shown in Algorithm 7. To have a better understanding of the algorithm, consider Figure 3.2 where we have a set of three seeds <*Indonesia*, *Jakarta*, *Indonesian Rupiah*>, <*Singapore*, *Singapore*, *Singapore Dollar*>, and <*Malaysia*, *Kuala Lumpur*, *Malaysian Ringgit*>. The first step of building the graph is to query the search engine using the three seeds (line 2). Suppose that we find two Web pages, `http://1min.in/content/international/currency-codes` and `http://www.science.co.il/International/Currency-codes.asp` after running the query. Each of the set of Web pages is then added as a new

**Algorithm 7:** BuildGraph

```
Input  : A set of seeds R = {T_1, T_2, ..., T_Ns}
Output: A graph G = {V, E}
1  V = ∅; E = ∅; V = V ∪ R;
2  P = FindDocuments(R);
3  foreach P_i ∈ P do
4      V = V ∪ P_i;
5      d = FindDomain(P_i);
6      V = V ∪ d;
7      E = E ∪ {< P_i, BELONG_TO, d >, < d, INV_BELONG_TO, P_i >};
8      foreach r in R do
9          E = E ∪ {< P_i, INV_RETRIEVE, r >, < r, RETRIEVE, P_i >};
10     end
11     if UsePermutation then
12         Wrappers, Order = {GenerateWrappersWithPermutation(R, P_i)};
13     else
14         Wrappers = {WrapperGeneration(FindSeedsOccurrences(R, P_i))};
15     end
16     if Wrappers ≠ ∅ then
17         foreach w ∈ Wrappers do
18             V = V ∪ w;
19             E = E ∪ {< P_i, GENERATE, w >, <
                w, INV_GENERATE, P_i >};
20             Candidates = {extract tuples from P_i using w};
21             if Candidates ≠ ∅ then
22                 foreach c ∈ Candidates do
23                     if UsePermutation then
24                         c = PutToOriginalOrder(c, Order);
25                     end
26                     V = V ∪ c;
27                     E = E ∪ {< w, EXTRACT, c >, <
                        c, INV_EXTRACT, w >};
28                 end
29             end
30         end
31     end
32 end
33 return G;
34 Function FindDocuments(A set of seeds R = {T_1, T_2, ..., T_Ns})
35     P ={retrieve set of pages containing R};
36     return P;
37 end
```

Table 3.2: The list of entities and relationships of the graph of entities in STEP

| **Source Node** | **Relation** | **Target Node** |
|---|---|---|
| Seeds | *RETRIEVE* | Web page |
| Web page | *GENERATE* | Wrapper |
| | *BELONG_TO* | Domain |
| | *INV_RETRIEVE* | Seeds |
| Wrapper | *EXTRACT* | Candidate |
| | *INV_GENERATE* | Web page |
| Candidate | *INV_EXTRACT* | Wrapper |
| Domain | *INV_BELONG_TO* | Web page |

node to the graph (line 4). We find the domain to which the Web page belongs to, and then add the domain to the set of nodes (line 5-6). We add edges between the Web page and the domain (line 7). We also connect the Web page to each of the seeds (line 8-10). Using the set of seeds and the Web page, we then generate the set of wrappers (line 11-15). Recall that there are two ways of generating the wrappers: using Algorithm 4 or Algorithm 6. The latter is used when we also take into consideration that the elements of the seeds can be found arbitrarily in a Web page and not necessarily be in the same order as the given seeds. As can be seen from Figure 3.2, from `http://1min.in/content/international/currency-codes` we generate two wrappers. Each of the wrappers is then added as a node to the graph (line 18) and its relationships to the Web page is added as edges (line 19). For each of the wrappers, we try to extract candidates from the Web page (line 20). If we are succeeded to do so, then we add each candidate as a node as well as create the edge between the candidate and wrapper (line 22-28). Note that if we extract the candidates using a permutated wrapper, then we need to put the order of the elements of the extracted candidates the same as the order of the given seeds (line 23-25). From the leftmost wrapper in Figure 3.2, we extract the three candidate tuples <*United Kingdom*, *London*, *Poundsterling*>, <*USA*, *Washington DC*, *US Dollar*>, <*France*, *Paris*, *Euro*> and create edges between the nodes.

$$A^* = (A - I)^{-1} \tag{3.1}$$

After the graph is generated, an iterative process is run to do a random walk [76] on the graph. The weight assigned to each node, once the iterative process has been completed or reaches a stationary probability, is then used to produce a ranking list of entities. However, the exact solution to finding the stationary probability of a random walk in a graph consisting of calculating Equation 3.1, where $A$ and $I$ are the transition matrix and the identity matrix respectively. The random walk problem is not only computationally expensive but also does not work for special cases of graph, i.e. graph whose nodes have no outgoing edges (dangling nodes) and graph with disconnected components. In this case, all nodes receive 0 as their final weight. This is counter intuitive because only nodes that have no incoming edge should receive 0 as their final weight. In the latter case, the notation of ranking

pages from one connected component to another connected component becomes ambiguous. Page et. al. [77] give the solution to this problem by introducing a positive constant $p$ between 0 and 1, which they call the damping factor. They also define the PageRank matrix (or Google matrix) of the graph by Equation 3.2. A denotes the initial distribution of the nodes while B is the "teleportation" probability. The teleportation probability is an $n$-by-$n$ matrix with every entry equals to $\frac{1}{n}$. The matrix $M$ models the random walk as follows. If we are at a node $i$, on the next move, most of the time, we follow the outgoing edges from $i$ to one of its neighboring nodes. A smaller, but a positive percentage of the time, we leave the current node and choose arbitrarily a new node from the graph and "teleport" to this node. It has also been proven in [123] that PageRank algorithm performed competitively against other ranking methods, i.e. Random Walk with Restart [124], Bayesian Sets [125], and Wrapper Length [123]. Due to these reasons, we choose to apply PageRank on the graph of entities. The weight computed by PageRank for each node can then be used to rank nodes of the same type. The candidates presented as the output of the system are ranked based on their PageRank weight. We are also able to produce a ranked list of the domain given the seeds. This can be viewed as the most *relevant* domain for a particular user's query.

$$M = (1 - p) \cdot A + p \cdot B \tag{3.2}$$

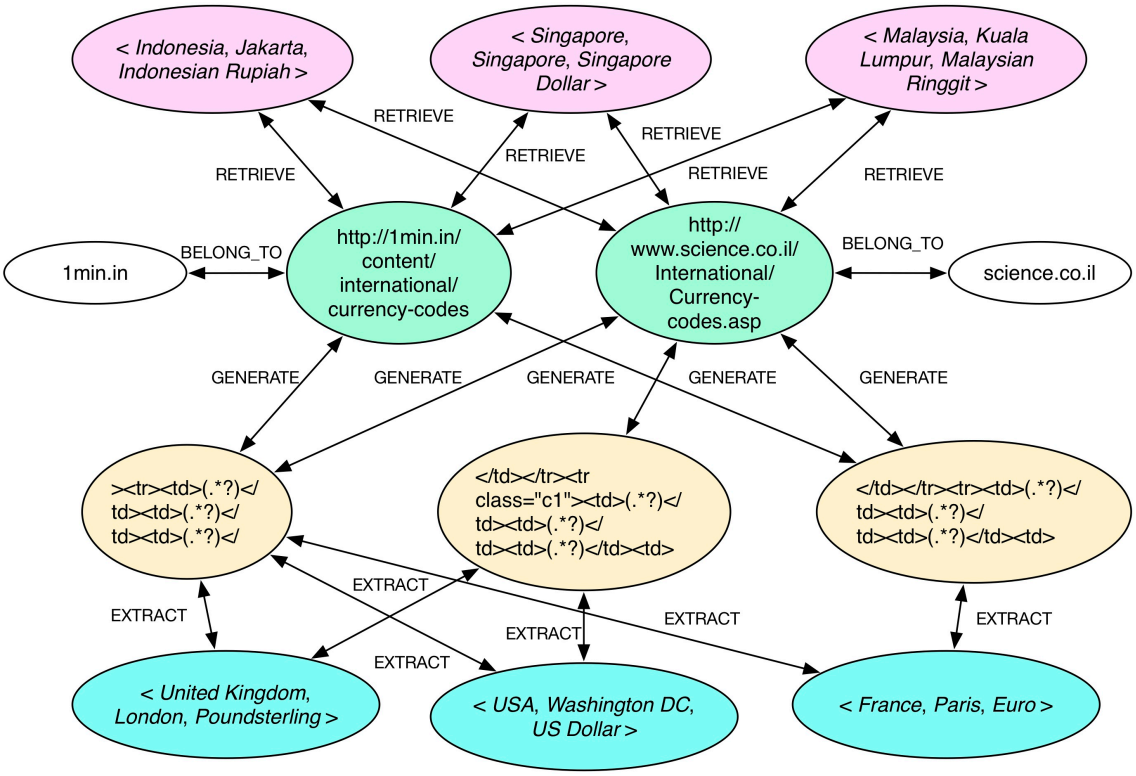


Figure 3.2: An illustration of the graph of entities

## 3.3 Performance Evaluation

### 3.3.1 Experiment Setting

We implement our approach into a system of extracting tuples from the Web. To evaluate the performance of the system, we first manually define several topics where each topic defines a set of tuples or table. To compare the candidates extracted by the system, we manually construct the ground truth for each topic by collecting data from various reference Web pages. These reference Web pages are manually identified using various search engines including Google[1] and Google Tables[2]. We have inspected and fixed typos and errors in the resulting tables. All the tuples that they contain are correct. The reference tables are generally complete to the extent discussed below for individual topics. For each topic, we define a query consisting of two to four example tuples from the reference table. For the sake of fairness, in the experiments presented here, we artificially exclude the Web pages that are used to construct the ground truth for the crawling results. The eleven topics, queries, and their corresponding reference ground truth are presented in Table 3.3.

We use two metrics to measure the effectiveness of the system. We measure the *precision* and *recall* of the system for the top-$K$ results for varying values of $K$ ($K$ denotes the number of candidates). While the candidates are ranked, the list of references given as ground truth is not. Let $O$ be the candidates extracted by the system and $B$ is the ground truth. The *Precision*, $p$, is the percentage of correct candidates among $O$. The *Recall*, $r$, is the percentage of correct candidates among all possible correct candidates. $p$ and $r$ are defined in Equation 3.3.

$$p = \frac{\sum_{i=1}^{|O|} Entity(i)}{|O|}; r = \frac{\sum_{i=1}^{|O|} Entity(i)}{|B|} \tag{3.3}$$

*Entity(i)* is a binary function that returns 1 if the $i$-th candidate belongs to the ground truth and 0 otherwise.

There are several options for crawling. The simplest is to present all seeds as they are given as a set of keywords to the search engine. Alternatively, we can consider permutations of the elements in the seeds. As we have discussed in Section 3.2.1, we do not consider such permutations in the crawling phase due to the high number of possible permutations. However, we take into consideration the permutation of the elements in the wrapper generation and extraction phases. In the first experiment denoted by "OR" in Table 3.4 and 3.5, the keywords are ordered as they appear in the seed tuple. We generate the wrapper for the "OR" experiment using Algorithm 4. The result is shown in the corresponding rows. In the experiment denoted by "PW" in tables 3.4 and 3.5, we use all permutations of keywords. For example, if the seed is <*Indonesia*, *Jakarta*, *Indonesian Rupiah*>, then we also search for <*Indonesia*, *Indonesian Rupiah*, *Jakarta*>, <*Jakarta*, *Indonesia*, *Indonesian Rupiah*>, etc. In this second approach, we use Algorithm 6 to generate the wrappers. Note that if the candidates are extracted by what we refer as permutated wrappers then before

[1] `http://www.google.com`
[2] `https://research.google.com/tables`

Table 3.3: Topics used for performance evaluation in the set of tuples expansion

| Topics | Seeds | Reference |
|---|---|---|
| DT1 - Airports | <*London Heathrow Airport*, *London*>, <*Charles De Gaulle International Airport*, *Paris*>, <*Schipol Airport*, *Amsterdam*> | `http://www.kccusa.com/airport-codes` |
| DT2 - Universities | <*Massachusetts Institute of Technology (MIT)*, *United States*>, <*Stanford University*, *United States*>, <*University of Cambridge*, *United Kingdom*> | `https://www.theguardian.com/higher-education-network/2011/sep/05/top-100-universities` |
| DT3 - Car brands | <*Chevrolet*, *USA*>, <*Daihatsu*, *Japan*>, <*Kia*, *Korea*> | `http://www.autosaur.com/car-brands-complete-list` |
| DT4 - US agencies | <*ARB*, *Administrative Review Board*>, <*VOA*, *Voice of America*> | `http://www.fedjobs.com/chat/agency_acronymns.html` |
| DT5 - Rock bands | <*Creep,Radiohead*>, <*Black Hole Sun*, *Soundgarden*>, <*In Bloom*, *Nirvana*> | `https://en.wikipedia.org/wiki/List_of_downloadable_songs_for_the_Rock_Band_series` |
| DT6 - MLM | <*mary kay*, *usa*>, <*herbalife*, *usa*>, <*amway*, *usa*> | `https://www.businessforhome.org/2015/04/100-solid-top-mlm-companies/` |
| DT7 - Olympic | <*1896*, *Athens*, *Greece*>, <*1900*, *Paris*, *France*>, <*1904*, *St Louis*, *USA*> | `https://www.google.com/mymaps/viewer?mid=1rb34I18jAquXiPT_EP9E4diE6QU&hl=en` |
| DT8 - FIFA Player | <*2015*, *Lionel Messi*, *Argentina*>, <*2014*, *Cristiano Ronaldo*, *Portugal*>, <*2007*, *Kaka*, *Brazil*>, <*1992*, *Marco van Basten*, *Netherlands*> | `http://www.topendsports.com/sport/soccer/list-player-of-the-year-fifa.htm` |
| DT9 - US Governor | <*Rick Scott*, *Florida*, *Republican*>, <*Andrew Cuomo*, *New York*, *Democratic*> | `https://en.wikipedia.org/wiki/List_of_current_United_States_governors` |
| DT10 - Currency | <*China*, *Beijing*, *Yuan Renminbi*>, <*Canada*, *Ottawa*, *Canadian Dollar*>, <*Iceland*, *Reykjavik*, *Iceland Krona*> | `http://1min.in/content/international/currency-codes` |

adding the candidates to the graph we have to change the order of its elements to match the order of the elements of the given seeds. This is done to ensure that each element of the candidates matches the semantic class of the corresponding element in the seeds.

The rows of Table 3.4 and 3.5 correspond to topics DT1 to DT11. For each topic, we report the results of both experiments, OR and PW. We select 10, 25, 50, 100, 200, 300, 400 as the values for $K$. The value inside the brackets, for example (441) for row DT1 and $K$=400 for the "OR" experiment, gives the number of the extracted tuples from the Web ($|O|$=441).

### 3.3.2 Result and Discussion

From Table 3.4 we can see that, for topics DT1, DT3, DT4, and DT10, we achieve a precision of at least 0.78. When we look closely at the details of the result of the experiments, we find that the extracted candidates come from a single Web page except for topic DT3. The system extracts 441 candidates from `http://www.airportcodes.us/airports.htm` for topic DT1, but seven candidates are not included in our ground truth. We check and find that these 7 candidates are actually spelled differently. For example, N'Djamena Airport is written as N_Djamena

Airport, Marseille / Provence Airport is written as Marseille/Provence Airport and Dakar - Yoff International Airport is written as Dakar _ Yoff International Airport. For topic DT4, 332 candidates are extracted from `http://www.onvia.com/business-resources/articles/guide-government-acronyms`. 18 of the extracted candidates are rejected by our ground truth from which 13 are again spelled differently. For example, Bio-informatics is written as Bioinformatics, Organization as Organisation. Other problems we encounter include the omission of an apostrophe, special characters ("&amp"), additional information and acronyms. This leaves only five of the extracted candidates that objectively are not included in the ground truth. For topic DT10, we extract 274 candidates from `http://www.science.co.il/International/Currency-codes.asp`. 240 out of the 274 extracted candidates are correct. Four entries from our ground truth are missing in the extracted candidates because we extract the encoded character "&" as "&" as part of the data instead of "&". For topic DT3, 87 candidates are extracted from four different Web pages of which 68 are included in the ground truth. Two out of four Web pages from which we extract the candidates for this topic are Youtube pages (`https://www.youtube.com/watch?v=dF0bTRbTEDE`, `https://www.youtube.com/watch?v=o6U7KIgkt_c`). This case illustrates the versatility of sources of information on the World Wide Web.

Table 3.4: Precision of top-$K$ ($K$ = 10, 25, 50, 100, 200, 300, 400) candidates (OR=Original order, PW=Permutation in wrapper)

| **Data** | | **Top-K** | | | | | | |
|---|---|---|---|---|---|---|---|---|
| | | **10** | **25** | **50** | **100** | **200** | **300** | **400** |
| **DT1 - Airports** | OR | 1.0 | 1.0 | 1.0 | 0.99 | 0.985 | 0.98 | 0.984 (441) |
| | PW | 1.0 | 1.0 | 1.0 | 0.99 | 0.98 | 0.98 | 0.984 (441) |
| **DT2 - Universities** | OR | 0.7 | 0.44 | 0.3 | 0.24 | 0.13 | 0.1 | 0.08 (473) |
| | PW | 0.7 | 0.4 | 0.26 | 0.23 | 0.135 | 0.1 | 0.07 (542) |
| **DT3 - Car brands** | OR | 0.9 | 0.84 | 0.92 | 0.78 (87) | 0.78 (87) | 0.78 (87) | 0.78 (87) |
| | PW | 0.9 | 0.84 | 0.84 | 0.76 | 0.75 (102) | 0.75 (102) | 0.75 (102) |
| **DT4 - US agencies** | OR | 1.0 | 1.0 | 0.96 | 0.97 | 0.935 | 0.943 | 0.945 (332) |
| | PW | 1.0 | 1.0 | 0.98 | 0.94 | 0.94 | 0.95 | 0.945 (332) |
| **DT5 - Rock bands** | OR | 0.2 | 0.28 | 0.32 | 0.32 | 0.19 | 0.156 | 0.156 (319) |
| | PW | 0.2 | 0.28 | 0.34 | 0.3 | 0.225 | 0.186 | 0.133 (1813) |
| **DT6 - MLM** | OR | 0.6 | 0.52 | 0.66 | 0.59 | 0.365 | 0.403 | 0.39 (330) |
| | PW | 0.6 | 0.44 | 0.28 | 0.35 | 0.36 | 0.243 | 0.182 (884) |
| **DT7 - Olympic** | OR | 0.9 | 0.56 | 0.44 | 0.23 | 0.135 | 0.135 (200) | 0.135 (200) |
| | PW | 0.9 | 0.64 | 0.44 | 0.22 | 0.11 | 0.073 | 0.044 (624) |
| **DT8 - FIFA player** | OR | 0.2 | 0.24 | 0.12 | 0.07 | 0.075 | 0.069 (215) | 0.069 (215) |
| | PW | 0.3 | 0.24 | 0.12 | 0.1 | 0.06 | 0.056 (284) | 0.056 (284) |
| **DT9 - US governor** | OR | 0.6 | 0.68 | 0.46 | 0.23 | 0.125 | 0.113 (220) | 0.113 (220) |
| | PW | 0.5 | 0.48 | 0.48 | 0.24 | 0.13 | 0.116 (223) | 0.116 (223) |
| **DT10 - Currency** | OR | 1.0 | 1.0 | 0.66 | 0.83 | 0.91 | 0.875 (274) | 0.875 (274) |
| | PW | 1.0 | 1.0 | 0.66 | 0.83 | 0.91 | 0.875 (274) | 0.875 (274) |
| **DT11 - Formula 1** | OR | 0.9 | 0.36 | 0.18 | 0.19 | 0.18 | 0.152 (289) | 0.152 (289) |
| | PW | 0.7 | 0.48 | 0.24 | 0.12 | 0.11 | 0.073 | 0.055 (798) |

For other topics, the precision is below 0.5. There are several reasons why the performance of these topics is low. Topic DT2 is concerned with universities and their country. We rely on Google table (http://research.google.com/tables) to provide us with the source for our ground truth. We searched "NUS Singapore Oxford UK" and used the data from the top-100 QS ranking for computer science and information

Table 3.5: Recall of top-$K$ ($K$ = 10, 25, 50, 100, 200, 300, 400) candidates (OR=Original order, PW=Permutation in wrapper)

| Data | | Top-K | | | | | | |
|---|---|---|---|---|---|---|---|---|
| | | **10** | **25** | **50** | **100** | **200** | **300** | **400** |
| **DT1 - Airports** | OR | 0.022 | 0.056 | 0.1133 | 0.2244 | 0.4467 | 0.66 | 0.984 (441) |
| | PW | 0.0226 | 0.056 | 0.1133 | 0.2244 | 0.44 | 0.66 | 0.984 (441) |
| **DT2 - Universities** | OR | 0.07 | 0.11 | 0.15 | 0.24 | 0.26 | 0.3 | 0.38 (473) |
| | PW | 0.07 | 0.1 | 0.13 | 0.23 | 0.27 | 0.3 | 0.38 (542) |
| **DT3 - Car brands** | OR | 0.086 | 0.201 | 0.442 | 0.653 (87) | 0.653 (87) | 0.653 (87) | 0.653 (87) |
| | PW | 0.086 | 0.201 | 0.403 | 0.73 | 0.74 (102) | 0.74 (102) | 0.74 (102) |
| **DT4 - US agencies** | OR | 0.014 | 0.035 | 0.067 | 0.136 | 0.262 | 0.397 | 0.441 (332) |
| | PW | 0.014 | 0.035 | 0.068 | 0.132 | 0.264 | 0.4 | 0.441 (332) |
| **DT5 - Rock bands** | OR | 0.001 | 0.0036 | 0.0083 | 0.0167 | 0.0199 | 0.0246 | 0.0277 (319) |
| | PW | 0.001 | 0.0036 | 0.0089 | 0.015 | 0.023 | 0.029 | 0.1269 (1813) |
| **DT6 - MLM** | OR | 0.0625 | 0.135 | 0.343 | 0.614 | 0.76 | 1.0 | 1.0 (330) |
| | PW | 0.0625 | 0.1145 | 0.1458 | 0.3645 | 0.75 | 0.76 | 1.0 (884) |
| **DT7 - Olympic** | OR | 0.3 | 0.46 | 0.73 | 0.76 | 0.9 | 0.9 (200) | 0.9 (200) |
| | PW | 0.3 | 0.53 | 0.73 | 0.73 | 0.73 | 0.73 | 0.93 (624) |
| **DT8 - FIFA player** | OR | 0.08 | 0.24 | 0.24 | 0.28 | 0.6 | 0.6 (215) | 0.6 (215) |
| | PW | 0.12 | 0.24 | 0.24 | 0.4 | 0.48 | 0.64 (284) | 0.64 (284) |
| **DT9 - US governor** | OR | 0.12 | 0.34 | 0.46 | 0.46 | 0.5 | 0.5 (220) | 0.5 (220) |
| | PW | 0.1 | 0.24 | 0.48 | 0.48 | 0.52 | 0.52 (223) | 0.52 (223) |
| **DT10 - Currency** | OR | 0.04 | 0.102 | 0.135 | 0.34 | 0.74 | 0.98 (274) | 0.98 (274) |
| | PW | 0.04 | 0.102 | 0.135 | 0.34 | 0.74 | 0.98 (274) | 0.98 (274) |
| **DT11 - Formula 1** | OR | 0.136 | 0.136 | 0.136 | 0.287 | 0.54 | 0.66 (289) | 0.66 (289) |
| | PW | 0.106 | 0.181 | 0.181 | 0.181 | 0.33 | 0.33 | 0.66 (798) |

systems departments[3]. Our system is able to extract 234 candidates. However, most of them are not in the ground truth. For example, we find <*Swiss Federal Institute of Technology Zurich*, *Switzerland*>, <*University of Illinois at Urbana-Champaign*, *United States*>, <*Technical University of Denmark*, *Denmark*>, <*Delft University of Technology*, *Netherlands*>, <*The University of Tokyo*, *Japan*> in the top 20 extracted candidates. They are indeed universities but are not in the top 100 QS ranking for computer science and information systems departments. This illustrates the inherent ambiguity of the concept set amplified by the set of seeds in a hierarchy of concepts. The top 100 universities for computer science and information systems department are universities. We face a similar problem with topics DT5 and DT6. For topic DT5, the ground truth has a large number of entries (1907). Our system is able to extract 289 candidates where only 43 of the extracted candidates are in the ground truth. Topics DT6, DT7, DT8, and DT11 yield poor precision yet achieve good recall.

The presentation of data on a Web page clearly also has an impact on the precision and recall of our system. With topic DT9, where we look for US governor names, the states governed and the political parties, we retrieve a Web page at the URL `http://www.ontheissues.org/2014_State.htm`. We process the page and generate a wrapper of the form ".htm'>(.*?)</a> (.*?) (.*?) Governor". The three capturing groups, denoted by (.*?) in the wrapper, correspond to the three elements in the tuple. When we look at the wrapper, the second and third element of the tuple is separated by a blank space. In this case, the separator (blank character) is very common which leads to the failing of the wrapper in extracting the correct infor-

[3]`https://www.theguardian.com/higher-education-network/2011/sep/05/top-100-universities-world-computer-science-and-information-systems-2011`

Table 3.6: Running time of STEP (in minutes)

| | | DT1 | DT2 | DT3 | DT4 | DT5 | DT6 | DT7 | DT8 | DT9 | DT10 | DT11 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| **Crawling** | **OR** | 0.1639 | 0.8237 | 0.9403 | 0.0707 | 0.7330 | 1.210 | 0.7831 | 0.4848 | 0.7375 | 0.2926 | 0.7584 |
| | **PW** | 0.2628 | 0.6322 | 0.7095 | 0.0546 | 0.4996 | 0.7929 | 0.5590 | 0.3833 | 0.6147 | 0.1351 | 0.5262 |
| **Wrapper Generation** | **OR** | 0.0009 | 0.0044 | 0.0028 | 0.0003 | 0.0010 | 0.0010 | 0.0013 | 0.0022 | 0.0004 | 0.0003 | 0.0012 |
| | **PW** | 0.0008 | 0.0029 | 0.0071 | 0.0005 | 0.0020 | 0.0017 | 0.0014 | 0.0036 | 0.0011 | 0.0004 | 0.0020 |
| **Extraction** | **OR** | 0.0004 | 0.5695 | 0.0056 | 0.0001 | 0.0257 | 0.0136 | 0.0037 | 0.0117 | 0.0013 | 0.0002 | 0.1912 |
| | **PW** | 0.0005 | 0.7493 | 0.0080 | 0.0001 | 0.0985 | 0.0182 | 0.0043 | 0.0145 | 0.2852 | 0.0003 | 0.6291 |
| **Ranking** | **OR** | 0.0079 | 0.0244 | 0.0037 | 0.0054 | 0.0119 | 0.0096 | 0.0091 | 0.0059 | 0.0036 | 0.0033 | 0.0096 |
| | **PW** | 0.0043 | 0.0168 | 0.0036 | 0.0043 | 0.0564 | 0.0208 | 0.0132 | 0.009 | 0.0078 | 0.0064 | 0.0272 |

mation. For example, if we have the text "...martinez.htm'>Susana Martinez</a> New Mexico Republican Governor" and apply our wrapper to the text, the three matching groups correspond to "Susana Martinez", "New" and "Mexico Republican". Thus the candidate extracted from the text is <*Susana Martinez*, *New*, *Mexico Republican*>. This extracted candidate is rejected by the system because the correct entry in the ground truth is <*Susana Martinez*, *New Mexico*, *Republican*>.

Using seeds with an element that is a member of more than one semantic class (*multifaceted* element) can also hurt the precision of the system. In Section 3.1 we consider an element $e$ as a member of a semantic class $s_i$ if and only if $e$ is $\in s_i$ but not in $S \setminus s_i$. Nevertheless, we come across cases where this multifaceted element affects the performance of our system. For example, when we want to extract cities in the United Kingdom (UK) and use <*Liverpool*>, <*Manchester*>, <*Norwich*> as the input to the system, the system fails to extract the correct candidates. Instead of extracting other cities in the United Kingdom, our system extracts football clubs <*Chelsea*>, <*Arsenal*>, <*Tottenham*>. This and the issue raised by the results for topics DT2, DT5 and DT6 compel for the use of ontological information and suggest the new problem of set labelling (Chapter 7). From table 3.5, we can see that the recall values for topics DT1, DT3, DT6, DT7, DT8, DT9, DT10, and DT11 are more than 0.5. For topic DT6, we register a perfect recall score of 1.0. This means that our system can extract candidates correctly. Nevertheless, due to the lack of source Web pages with the information sought, the number of extracted candidates is at least half of that in the ground truth. The lack of source Web pages is related to the search in Google. In this experiment, we use the concatenation of all the seeds as keywords to the Google search engine. This limits the number of Web pages returned by Google. For example, in the experiment for topic DT4, we use "ARB Administrative Review Board VOA Voice of America" as the search keyword to Google and only retrieve one Web page. From this Web page, we extract 332 candidates and 314 are in the ground truth. It receives less than 0.5 on the recall score because the ground truth contains 712 entries. For topic DT4, the lack of Web page retrieved from Google affects the recall of our system.

In topic DT5, we have 1907 entries of songs and the singers. Our seeds consist of <*Creep*, *Radiohead*>, <*Black Hole Sun*, *Soundgarden*>, <*In Bloom*, *Nirvana*>. All of the seeds come from a typical genre of music which is rock. But in our experiment, the extracted candidates come from a different music genre. For example, <*Can This Be Love*, *Jennifer Lopez*> and <*Louie*, *Nelly*> are from R&B and rap genre respectively. This happens because the Web pages retrieved from Google do not specifically belong to one music genre. Furthermore, some Web pages provide

information about music in general. To improve the recall values of the system we need to be able to direct our search on Google to match the intended target information or choose a reliable Web page as the source for ground truth. Topic DT2 receives low recall because the ground truth only has 100 entries whilst we extract many candidates. Although the extracted candidates belong to the same semantic group (universities) since it is not included in the ground truth, the recall value becomes low. For the experiment with topic DT6 (MLM), the system receives the maximum score (1.0) for recall. Our ground truth includes 96 entries and the system manages to extract it from various Web pages. In fact, when we check on the extracted candidates, the system is able to extract many "true" candidates which are not in the list of ground truth. In this context, the word "true" means that the candidate belongs to the same semantic group as the seeds, but is missing from the ground truth.

Now we look at the impact of elements permutation in tuple seeds for wrapper generation. In general, permutation has a negative impact on the precision of the system while the recall is positively affected. The precision of the system becomes lower because we extract more candidates. However, this also increases the possibility of extracting "junk". This happens to topics DT2, DT3, DT5, DT6, DT7, DT8, and DT11. From table 3.4, the precision values for topics DT1, DT4, and DT10 remain the same for both experiments (OR and PW). For topic DT9, the precision for experiment PW slightly increases compared to that of OR because the system manages to extract more correct candidates.

To see whether doing the permutation of elements of tuple seeds in wrapper generation process really improves the system's performance, we conduct another experiment using topic DT4. The actual seeds are <*ARB*, *Administrative Review Board*>, <*VOA*, *Voice of America*> and we change the seeds to <*Administrative Review Board*, *ARB*>, <*Voice of America*, *VOA*>. We run the OR experiment and get no result. There is only one Web page retrieved by Google. The system cannot find any occurrence of the seeds as the order of elements in the seeds is not the same as the order of occurrences of each element in the Web page. We then run the PW experiment with the same topics. The system manages to retrieve 26 candidates from the Web page. This shows us that doing permutation of each element of a tuple when generating the wrapper is indeed improving the performance of our system.

We conduct the above experiments with queries with two to four seeds. We vary and increase the number of seeds in a query in other experiments whose results are not reported here. The performance is negatively affected by more seeds in the current implementation. The crawling phase returns fewer candidate Web pages. We can see this problem with DT8. We use four seeds as the input and the number of Web pages retrieved by the search engine is 15 (half of the maximum number of 30 pages that we collected for each experiment). This and the above highlight that we need to find more effective ways of collecting candidate Web pages for a sustainable solution. We are currently working on various ways to improve the crawling phase accordingly, for instance by decomposing the query into several subqueries.

The running time for each phase of our system for all topics can be seen in Table 3.6. The crawling phase consumes most of the time as retrieving Web pages

from Google depends highly on the stability of the connection. Considering permutations of elements of the seeds affect the running time in the wrapper generation and candidate extraction, as well as the ranking. In wrapper generation, the system has to do more runs in order to find the occurrences of the seeds due to the number of possible permutations. The more wrappers, the more extractions, and the more entities added to our final graph. This is why the ranking process also needs more time to complete.

All of the above experiments are run under the setting where we exclude the Web pages used to build the reference tables constituting the ground truth. We do not consider these reference pages in the crawling and subsequent phases for the sake of fairness. We also run experiments in which we do not exclude the Web pages used to build the ground truth. It is noticeable that these Web pages are not always discovered by the crawling phase (that these pages are not in the top 30 pages returned by the search engine presented with the query string formed from the examples). Furthermore, when these reference pages are identified by the crawler and used for the expansion, the performance results (precision, recall, and running time) do not differ significantly. This means that, for the eleven topics, there are generally sufficiently enough sources of information on the Web.

Table 3.7: Performance comparison of DIPRE, Ext. SEAL, and STEP

| | **Algorithm** | **Topic** | | | | | | **Average** |
|---|---|---|---|---|---|---|---|---|
| | | **DT1** | **DT2** | **DT3** | **DT4** | **DT5** | **DT6** | |
| Precision | DIPRE | 0 | 0 | 0 | 0.945 | 0.256 | 0 | 0.2 |
| | Ext. SEAL | 0 | 0.6 | 0.46 | 0.93 | 0 | 0.6 | 0.43 |
| | STEP | 0.97 | 0.07 | 0.74 | 0.945 | 0.136 | 0.182 | **0.51** |
| Recall | DIPRE | 0 | 0 | 0 | 0.845 | 0.005 | 0 | 0.14 |
| | Ext. SEAL | 0 | 0.24 | 0.125 | 0.83 | 0 | 0.88 | 0.345 |
| | STEP | 0.986 | 0.38 | 0.98 | 0.441 | 0.126 | 1.0 | **0.65** |

## 3.4 Comparison with state-of-the-art

In this section, we compare our proposed approach of the set of tuples expansion with two state-of-the-art in set expansion systems: DIPRE [16] and Extended SEAL [30]. Both DIPRE and Extended SEAL can only extract binary relations, thus to make a fair comparison, we evaluate the performance of DIPRE, Extended SEAL (Ext. SEAL), and STEP in terms of its precision and recall only for extracting binary tuples. For this experiment, we consider only DT1-DT6 from Table 3.3 using the given seeds. We run each of the algorithms by giving the seeds as input and retrieve all Web pages (without excluding any reference Web pages). All approaches in comparison find the occurrences of the seeds by locating each element in the same order as the order of the elements in the seeds. Next, all algorithms also find the occurrences of the seeds where the elements are in reversed order. As DIPRE does not employ any ranking mechanism, we compare the complete list of candidates produced by each algorithm without taking into consideration the ranking of each candidate. We set all parameters required by each algorithm to its default values.

Table 3.7 shows the result of the comparison of the three approaches. In terms of the average precision and recall, STEP outperforms DIPRE and Ext. SEAL. STEP consistently achieves higher precision than DIPRE on topic DT1, DT2, DT3, DT4, and DT6. Among these topics, DIPRE receives the lowest precision score of 0 in DT1, DT2, DT3, and DT6. As it has been pointed out in Algorithm 1, DIPRE first cluster the occurrences of the seeds based on their *middle* contexts. It then enforces exact matching on the middle contexts before generating the wrapper. Furthermore, it can only infer wrappers from Web pages that use semi-structured templates (e.g. HTML table, ordered or unordered list) to present the data, otherwise, it will fail to achieve the goal. On the other hand, out of the six topics used in this experiment, STEP achieves higher precision scores than Ext. SEAL in topics DT1, DT3, DT5 and DT6. As a matter of fact, in topics DT1 and DT5, Ext. SEAL does not extract any candidates, whilst STEP manages to extract 448 and 1759 candidates out of which 435 and 240 are in the ground truth respectively. Ext. SEAL achieves higher precision than STEP for topics DT2 and DT6. We take a look into the specifics and find that Ext. SEAL only extracts 40 and 140 candidates compare to 542 and 884 candidates extracted by STEP. In DT2, the number of candidates which conform with the ground truth for Ext. SEAL is 24 while STEP 38. However, STEP extracts an order of magnitude larger candidates than SEAL, thus it receives a lower precision score. We also browse the list of candidates extracted by STEP and find that it contains a large number of university names which do not appear in the ground truth. STEP dominates Ext. SEAL in terms of recall. Indeed, the one case where Ext. SEAL achieves higher recall than STEP is in topic DT4. In this topic, Ext. SEAL manage to extract 637 candidates. This number is almost double the number of candidates extracted by STEP. In DT6, STEP receives a recall score of 1.0 while SEAL only 0.88 despite the fact that it has a much lower precision. This once again proves the fact that STEP extracts more candidates that appear in the ground truth and relevant to the given seeds. The reason behind this is that SEAL implements an exact matching algorithm to infer the common middle contexts of all the occurrences of the seeds. This is a very strict requirement because Web pages are generated from different templates thus they are highly unlikely to have the same middle contexts. This failure cause SEAL to not be able to construct wrappers and hence can not extract any candidates. Our proposed approach to finding common middle contexts are based on regular expression. This allows STEP to be more flexible in constructing the wrapper and extracting the candidates. From the comparison result, we conclude that our approach is able to extract more candidates that are relevant to the seeds. This, in turn, helps the performance of our approach in optimising the recall while giving comparable precision to the state-of-the-art.

## 3.5 Conclusion

We present an approach to the set of tuples expansion from the World Wide Web. The approach consists of three phases: crawling, wrapper generation and data ex-

traction, and ranking. Examples given in the form of tuples create a combinatorial issue for the crawling, wrapper generation, and data extraction. The empirical evaluation shows that the system is efficient, effective, and practical. The comparison with two state-of-the-art approaches shows that STEP consistently extracts more correct candidates with refers to the ground truth. Nevertheless, we also highlight some of the shortcomings of the current system and suggests new engineering and research issues to be studied. One of them is the characterisation of the concept characterised by the examples. Additional semantics knowledge in the form of integrity constraints, such as candidate keys, admissible values and ranges, and dependencies, also has the potential to improve crawling, wrapper generation and data extraction.

# Chapter 4

# Truthfulness in Set of Tuples Expansion

In the previous chapter, we present the set of tuples expansion problem. We propose an approach to tackle the problem by generalising the set expansion problem. We present and detail our approach in each phase, i.e. crawling, wrapper generation and candidate extraction, as well as the ranking method. We extend the idea originally proposed by Wang et al. in [24] and applied a random walk on a heterogeneous graph of Web pages, wrappers, seeds, and candidates in order to rank the candidates according to their relevance to the seeds. However, this ranking mechanism is not able to reliably determine the true values amongst the extracted candidates. Distinguishing amongst true and false values of some given facts in an automated way, usually known as truth finding, is a challenging problem that has been extensively studied in several contexts such as data integration and fact-checking. In this chapter, we propose to incorporate a truth finding algorithm into our expansion process. The goal is to be able to present the true candidates to the user and achieve better performance in terms of precision, recall, and F-measure. We first formulate the problem we are addressing in Section 4.1. We then present our proposed approach in Section 4.2. We show the performance of a system implementing our approach in Section 4.3, while Section 4.4 concludes this chapter.

## 4.1 Problem Definition

We address here the problem of finding the true values amongst the candidates extracted by the set of tuples expansion system. Before we present our approach of leveraging truth finding algorithm into the expansion process, we start with the definition of the problem.

We fix the set of candidate tuples $\mathcal{T}$ extracted by STEP. These extracted tuples represent instances of real-world objects characterised by some attributes (or properties). We restrict ourselves in this work to the common *one-truth* framework where any attribute of every object has only *one correct value* and *many possible wrong values*. We consider fixed finite sets of *attribute labels* $\mathcal{A}$ and *values* $\mathcal{V}$, as well

as a finite set of *object identifiers* $\mathcal{I}$. Then, we have formally:

**Definition 1.** *A* tuple *$t$ is a pair $(i, v)$ where $i$ is a given object identifier in $\mathcal{I}$ and $v$ a mapping $v : \mathcal{A} \to \mathcal{V}$.*

Set of tuples expansion algorithm inspects documents using wrappers to extract a collection of tuples, i.e., its statements about the values of *some* attributes of *some* objects: see Example 2. In this setting, we describe a *data source* as modelled by a given document (where the value has been extracted) and a wrapper (the extractor of the value). At attribute level, we define indeed a data source as follows:

**Definition 2.** *A* source *is a partial function $S : \mathcal{I} \times \mathcal{A} \to \mathcal{V}$ a with non-empty domain. A (candidate)* ground truth *is a total function $G : \mathcal{I} \times \mathcal{A} \to \mathcal{V}$.*

Given a data source, a particular tuple $t$ is represented by the set of couples of the object identifier and attribute value which share the same object identifier. That is, given a source $S$ and two distinct attributes $a$, $a' \in \mathcal{A}$ we state that $S(i_1, a)$ and $S(i_2, a')$ refer to statements about attributes $a$ and $a'$ of the same tuple if $i_1 = i_2$ where $\mathrm{i}_1$ and $\mathrm{i}_2$ are object identifiers in $\mathcal{I}$.

**Example 2.** *Reconsidering tuples in Example 1 of Section 2.3, one can observe these are about the real world object "Country" with their properties (attributes) "Capital City" and "Currency": we have three candidate tuples <Algeria, Alger, Dinar Algerian>, <Algeria, Alger, Leka Algerian> and <Algeria, Alger, Dinar Algerian> from three different data sources* Source_1, Source_2 *and* Source_3. *Note that in Example 1, for simplicity reasons, the object identifier, i.e. "*Algeria*" was omitted while provided attribute values are "*Alger*", "*Dinar Algerian*" and "*Leka Algerian*" respectively.*

Truth finding problem's main goal is to determine the actual ground truth based on the statements of a number of sources. Sources are specifically defined as possibly incomplete; in particular, they may not cover all attributes:

**Definition 3.** *The* attribute coverage *of a source $S$ with respect to $X \subseteq \mathcal{A}$ is the proportion of attributes $a \in \mathcal{A}$ for which there exists at least one object $o \in \mathcal{O}$ such that $S(o, a)$ is defined: $Cov(S, X) := \frac{|\{\, a \in X \mid \exists o \in \mathcal{O}\ S(o,a)\ \textit{defined}\}|}{|X|}$.*

Two sources $S$ and $S'$ are *contradicting* each other, i.e. disagree, whenever there exists $i \in \mathcal{I}$, $a \in \mathcal{A}$ such that $S(i, a)$ and $S'(i, a)$ are both defined and $S(i, a) \neq S'(i, a)$. A source is *correct* with respect to the ground truth $G$ on $i \in \mathcal{I}$, $a \in \mathcal{A}$ when $S(i, a) = G(i, a)$, and *wrong* when $S(i, a)$ is defined but $S(i, a) \neq G(i, a)$. A truth finding process, such as those reviewed in Section 2.3, is formally defined as follows:

**Definition 4.** *A* truth finding algorithm *$F$ is an algorithm that takes as input a set of sources $\mathcal{S}$ and returns a candidate ground truth $F(\mathcal{S})$, as well as an estimated* accuracy *$Accuracy_F(S)$ for each source $S \in \mathcal{S}$ and confidence score $Confidence_F(S(i, a))$ for every statement $S(i, a)$ made by a source $S$ on any attribute $a$ of the object $i$.*

A large number of truth finding processes are fixed-point iteration algorithms which compute a candidate ground truth based on source accuracy (or trustworthiness) values and confidence scores. If a truth finding algorithm $F$ does not specifically use source accuracy values (this is the case, for instance, of majority voting), one can respectively define source accuracy and confidence for $F$ simply as:

$$Accuracy_F(S) := \frac{|\{\, i \in \mathcal{I}, a \in \mathcal{A} \mid S(i,a) = F(\mathcal{S})(o,a)\}|}{|\{\, i \in \mathcal{I}, a \in \mathcal{A} \mid S(i,a) \text{ defined}\}|} \tag{4.1}$$

For a fixed object identifier $i \in \mathcal{I}$ and attribute $a \in \mathcal{A}$, we have:

$$Confidence_F(S(i,a)) := \frac{\sum_{S' \in \mathcal{S} | S'(i,a) = S(i,a)} Accuracy_F(S')}{|\{\, S' \in \mathcal{S} \mid S'(i,a) = S(i,a)\}|} \tag{4.2}$$

**Example 3.** *Applying Equation 4.1 on candidate tuples in Example 2, we obtain the accuracy values of* 1, 1/2 *and* 1 *for Source_*1*, Source_*2*, and Source_*3 *respectively. Indeed the true values are chosen by* voting *thus the values for attributes "Capital City" and "Currency" will be "*Alger*" and "*Dinar Algerian*". According to Equation 4.2, the confidence scores of the statements "*Alger*", "*Algerian*" and "*Leka Algerian*" will be set to* 5/6, 1 *and* 1/2*. Obviously, "*Dinar Algerian*" comes from top-ranked sources, i.e. Source_*1 *and Source_*3*, thus is chosen instead of "*Dinar Algerian*".*

This is how the simplest approach of truth finding, i.e *voting*, determines the true values of some given facts. In our setting, the facts are some values of attributes of some real-world objects. We, therefore, leverage the truth finding algorithm in the tuples expansion process to find the true values of these attributes and in turn decide the correct candidates. We formally present next such a process.

## 4.2 Proposed Approach

As detailed in Section 3.2.3, we build the graph of entities and apply PageRank on the graph before finally rank the candidates based on the weight assigned by PageRank. In this work, we propose to incorporate truth finding algorithm into the expansion process to decide the true candidates amongst the extracted candidates and present a ranked list of candidates as the end result. To be able to achieve this goal, we need to relate the graph of entities to the truth finding setting.

From Definition 4.1, a truth finding algorithm takes as input a set of statements made by sources on any attribute of an object. We consider here the statements as the tuple candidates and the sources as the Web pages where we derive the wrapper used to extract the corresponding candidates. The overall objective of embedding truth finding algorithm into our approach is to improve the candidates presented as the end result. However, in the set of tuples expansion problem, the concept of an object and its attributes are not explicitly modelled. However, elements in the seeds and in extracted tuples do have some kind of relationships. Considering the tuple <*Angola*, *Luanda*, *Kwanza*>, one can easily argue that Luanda and Kwanza are the attributes of Angola. To be consistent with the truth finding setting, we then

technically decompose the object and its attributes in the tuples extracted by our system. An object is represented by the first element of the tuple, which is denoted as the *object key*, while the other remaining elements serve as the attributes. We further use this key element to decompose a tuple into *key-attribute* binary tuples. The previous tuple is decomposed into <*Angola*, *Luanda*> and <*Angola*, *Kwanza*>. In general, if a tuple is composed of $n$ elements, then after decomposition we obtain $n$-1 key-attribute tuples.

**Algorithm 8:** GenerateFacts

**Input :** A graph of entities $G = \{V, E\}$
**Output:** A set of facts tuples $B$

1 $B = \emptyset$;
2 $C$ = {retrieve all nodes from $G$ where the type of node=="candidates"};
3 **foreach** $c \in C$ **do**
4 $\quad p = FindSourceParentOf(c)$;
5 $\quad k = c[1]$;
6 $\quad$ **for** $i \leftarrow 2$ **to** $n$ **do**
7 $\quad\quad b =< k, Attr_{i-1}, c[i], p >$;
8 $\quad\quad B = B \cup \{b\}$;
9 $\quad$ **end**
10 **end**
11 **return** $B$;

To experiment with *state-of-the-art* truth finding algorithms presented in Section 2.3 we rely on the DAFNA platform[1]. DAFNA provides AllegatorTrack [126] which is a Web application setting up an implementation of most of the state-of-the-art truth finding algorithms within the same setting. AllegatorTrack accepts as input a set of entries whose main elements consist of an object key, an object attribute, an attribute value, and a data source. Thus, in each of the key-attribute tuple, we have to also include the object's attribute and the source. The source is the Web page from which the wrapper used to extract the candidate is derived from. Since the object's attribute is not known a priori, we simply give "*Attr*" followed by an index as the label.

We use Algorithm 8 to generate the facts from the graph of entities. We first retrieve all candidate nodes from the graph (line 2). Then for each node $c$, we find the Web page $p$ from which the candidate is extracted from (line 4). We do this by following the edge from the candidate node to the wrapper and then to the Web page. The object key $k$ is the first element of the $n$ elements tuple candidate $c[1]$, while the rest of the elements $(c[2], ..., c[n])$ are the attribute values. Thus, for each of the attribute values, we form the tuple in the form $<k, Attr_{n-1}, c[i], p>$ and add it to the result (line 6-9).

STEP produces wrappers for a Web page by combining the left, right and middle contexts (see Section 3.2.2). This leads to the possibility of extracting the identical

[1] http://da.qcri.org/dafna/#/dafna/home_sections/home.html

Table 4.1: Excerpt of input dataset in AllegatorTrack

| Object | Attribute | Value | Source |
|---|---|---|---|
| Angola | *Attr_1* | Luanda | `http://1min.in/content/international/currency-codes_1` |
| Angola | *Attr_2* | Kwanza | `http://1min.in/content/international/currency-codes_1` |
| Angola | *Attr_2* | Kuanza | `http://1min.in/content/international/currency-codes_2` |

or distinct tuples from the same Web page by means of different wrappers. One has to observe that a given source cannot provide more than one statement for the same object attribute within our truth finding setting. As a result, we refer to the source of a given candidate tuple by indexing the name of the inspected Web page with the number of the used wrapper. Table 4.1 shows an excerpt of an input dataset to AllegatorTrack in which each row consists respectively of the object key, followed by the attribute of the object, then the value of the attribute, and finally the source.

## 4.3 Performance Evaluation

### 4.3.1 Experiment Setting

We intensively performed our evaluation on seven topics consisting of a set of tuples extracted by our STEP system for distinct domains; we have detailed in Section 3.2 how our system returns a set of tuples in a certain domain given some seeds. Table 4.2 summarises the details of the seven topics used in our performance evaluation. Its columns indicate the name of the topic, the number of inspected Web pages, the number of wrappers generated from the Web pages, the number of extracted candidate tuples, the number of distinct candidate tuples, and the number of entries in the ground truth. Next, we briefly summarise every topic.

- DT1 - Calling Code. We extracted tuples composed of country and the prefix country calling code from the Web. We used the following four seeds as input to STEP: <*United States*, *1*>, <*Indonesia*, *62*>, <*France*, *33*>, <*Afghanistan*, *93*>.

- DT2 - Nobel. We extracted from the Web tuples consisting of a year and the name of Nobel prize winner in Physics. We used the following three seeds as input to our STEP system: <*1921*, *Albert Einstein*>, <*1971*, *Dennis Gabor*>, <*1938*, *Enrico Fermi*>.

- DT3 - Chat Abbr. We extracted from the Web tuples composed of texting abbreviations or internet acronyms and its descriptions. We used the following seeds as input: <*brb*, *be right back*>, <*afk*, *away from keyboard*>.

- DT4 - Currency. We extracted from the Web tuples composed of countries with their corresponding capital cities and currency names. We used the following seeds as input to the STEP System: <*Angola*, *Luanda*, *Kwanza*>, <*Bolivia*, *La Paz*, *Boliviano*>, <*India*, *New Delhi*, *Rupee*>,<*Cuba*, *Havana*, *Peso*>.

- DT5 - FIFA PLAYER. We extracted tuples from the Web composed of a year, the winner of the FIFA player of the year award, and his country of origin. The seeds are the following: *<2015, Lionel Messi, Argentina>*, *<2014, Cristiano Ronaldo, Portugal>*, *<2007, Kaka, Brazil>*.

- DT6 - MISS UNIVERSE. We collected tuples composed of a year, the winner of Miss Universe beauty pageant, and her country of origin from the Web. We used the following seeds as input: *<2000, India, Lara Dutta>*, *<2012, USA, Olivia Culpo>*,*<2015, Philippines, Pia Wurtzbach>*.

- DT7 - OSCAR. We extracted tuples from the Web composed of a year, the academy award winner for best actor, and the name of the movie in which the actor is nominated in. We used the following seeds as input: *<2016, Leonardo DiCaprio, The Revenant>*, *<2002, Adrien Brody, The Pianist>*, *<1995, Tom Hanks, Forrest Gump>*, *<1999, Kevin Spacey, American Beauty>*.

- GROUND TRUTH DATASETS. For performance evaluation of the truth finding algorithms and PageRank, we also gathered ground truth for each tested topic. To do so, we selected authoritative Web source as a base for the ground truth of each topic and manually extracted golden standard for each extracted object instance. The number of entries in each ground truth is indicated in the last column of Table 4.2.

Our tests intend to compare the precision, recall, and F-measure (a.k.a. the harmonic mean) of PageRank and eleven truth finding algorithms in terms of percentage of true and false candidates according to a certain ground truth. We compute the precision and recall using Equation 3.3, while F-measure is calculated using Equation 4.3.

$$\text{F-measure} = 2 * \frac{p * r}{p + r} \tag{4.3}$$

In Equation 3.3, *Entity(i)* is a binary function that returns 1 if the $i$-th candidate is found in the ground truth and 0 otherwise, $|O|$ is the number of distinct extracted candidates, and $|B|$ is the size of the ground truth. $|O|$ and $|B|$ is the fifth and last column in Table 4.2.

Table 4.2: Description of the different topics and corresponding ground truth

| Topic | #Web_Page | #Wrapper | #Candidate_Tuples | #Distinct_Tuples | Ground_Truth |
|---|---|---|---|---|---|
| **DT1 - Calling Code** | 16 | 98 | 5572 | 1001 | 239 |
| **DT2 - Nobel** | 5 | 7 | 446 | 237 | 107 |
| **DT3 - Chat Abbr.** | 12 | 11 | 5758 | 5167 | 2697 |
| **DT4 - Currency** | 9 | 39 | 2473 | 927 | 244 |
| **DT5 - FIFA Player** | 10 | 13 | 260 | 141 | 61 |
| **DT6 - Miss Universe** | 8 | 3 | 130 | 82 | 65 |
| **DT7 - Oscar** | 6 | 5 | 368 | 337 | 87 |

We investigate the correlation of ranking of sources between PageRank, truth finding algorithms, and the ground truth ranking in terms of precision, recall, and F-measure by computing the weighted Kendall's Tau score [127] using Equation 4.4.

$C$ and $D$ are the numbers of concordant and discordant pairs from any two ranked lists in comparison.

$$\tau = \frac{C - D}{C + D} \tag{4.4}$$

Table 4.3: Precision measures per-attribute of the different truth finding algorithms

| | | CO | 2E | 3E | DP | AC | AS | AN | TF | SL | GL | CM | PR |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| **DT1** | **Attr1** | 0.172 | 0.361 | 0.368 | 0.360 | 0.368 | 0.368 | 0.368 | 0.360 | 0.368 | 0.360 | 0.361 | 0.368 |
| **DT2** | **Attr1** | 0.137 | 0.147 | 0.168 | 0.147 | 0.153 | 0.153 | 0.153 | 0.147 | 0.153 | 0.153 | 0.158 | 0.153 |
| **DT3** | **Attr1** | 0.246 | 0.194 | 0.201 | 0.241 | 0.231 | 0.236 | 0.231 | 0.283 | 0.214 | 0.220 | 0.255 | 0.064 |
| **DT4** | **Attr1** | 0.620 | 0.667 | 0.702 | 0.647 | 0.676 | 0.676 | 0.676 | 0.632 | 0.702 | 0.673 | 0.647 | 0.620 |
| | **Attr2** | 0.084 | 0 | 0.017 | 0.003 | 0 | 0 | 0 | 0.005 | 0 | 0.003 | 0.005 | 0.020 |
| **DT5** | **Attr1** | 0.730 | 0.888 | 0.873 | 0.841 | 0.888 | 0.888 | 0.888 | 0.793 | 0.888 | 0.793 | 0.841 | 0.888 |
| | **Attr2** | 0.873 | 0.841 | 0.809 | 0.888 | 0.873 | 0.873 | 0.873 | 0.873 | 0.873 | 0.873 | 0.873 | 0.920 |
| **DT6** | **Attr1** | 0.924 | 0.878 | 0.924 | 0.924 | 0.924 | 0.924 | 0.924 | 0.924 | 0.924 | 0.924 | 0.924 | 0.939 |
| | **Attr2** | 0.818 | 0.787 | 0.818 | 0.787 | 0.838 | 0.838 | 0.848 | 0.803 | 0.818 | 0.848 | 0.787 | 0.803 |
| **DT7** | **Attr1** | 0.442 | 0.863 | 0.915 | 0.536 | 0.915 | 0.915 | 0.915 | 0.547 | 0.915 | 0.884 | 0.915 | 0.410 |
| | **Attr2** | 0.410 | 0.810 | 0.831 | 0.452 | 0.831 | 0.831 | 0.831 | 0.494 | 0.915 | 0.778 | 0.831 | 0.336 |

We started by experimenting with truth finding algorithms via AllegatorTrack. We considered each of our seven topics and performed on it the following algorithms: Cosine (CO), 2-Estimates (2E), 3-Estimates (3E), Depen (DP), Accu (AC), AccuSim (AS), AccuNoDep (AN), TruthFinder (TF), SimpleLCA (SL), GuessLCA (GL), and Combiner (CM). We ran those tests without changing the default values of the mandatory parameters for each algorithm, i.e. the *prior source trustworthiness* and *convergence threshold* which are set by default to 0.8 and 0.001 respectively: these are default values proven to be optimal. The output of every truth finding algorithm is, for every object's attribute, a confidence score and a truth label (true or false). To evaluate the performance of truth finding algorithm w.r.t to the ground truth, we only consider the tuples whose truth labels are set to true. For the sake of fairness, we consider the same extracted candidates for each topic and perform PageRank to it. We only consider tuples with the highest score for each object returned by the PageRank algorithm. We use the same implementation as well as the identical experimentation setting as in our previous work [35].

In the first phase of the experiment, we are interested in finding the best truth finding algorithm for the set of tuples expansion task. To achieve this, we conduct the following two experiments. In the first experiment we are interested in doing tests on topics per attribute, i.e., if a given topic contains objects with several attributes, we have considered sub-topics per attributes and run experiments on each sub-topic separately. We do such a splitting of each tested topic because we focus on the performance of the algorithms per attribute, as we later explain with the results in Table 4.3 and Table 4.4. In this perspective, we use "Attr", the abbreviation for an attribute, and add an index to differentiate the different attributes belonging to the same given topic when the latter has been split into several sub-topics. In the second experiment, we conduct tests using every overall topic, i.e. without doing

the splitting according to the attributes. The goal is to study the performance of the ranking algorithms on any given entire input topic.

Table 4.4: Recall measures per-attribute of the different truth finding algorithms

| | | CO | 2E | 3E | DP | AC | AS | AN | TF | SL | GL | CM | PR |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| **DT1** | **Attr1** | 0.423 | 0.887 | 0.904 | 0.883 | 0.904 | 0.904 | 0.904 | 0.883 | 0.904 | 0.883 | 0.887 | 0.895 |
| **DT2** | **Attr1** | 0.252 | 0.271 | 0.308 | 0.271 | 0.280 | 0.280 | 0.280 | 0.271 | 0.280 | 0.280 | 0.290 | 0.280 |
| **DT3** | **Attr1** | 0.247 | 0.195 | 0.201 | 0.241 | 0.232 | 0.236 | 0.232 | 0.284 | 0.215 | 0.221 | 0.255 | 0.064 |
| **DT4** | **Attr1** | 0.873 | 0.939 | 0.988 | 0.910 | 0.951 | 0.951 | 0.951 | 0.889 | 0.988 | 0.947 | 0.910 | 0.873 |
| | **Attr2** | 0.119 | 0 | 0.025 | 0.004 | 0 | 0 | 0 | 0.008 | 0 | 0.004 | 0.008 | 0.029 |
| **DT5** | **Attr1** | 0.754 | 0.918 | 0.902 | 0.869 | 0.918 | 0.918 | 0.918 | 0.820 | 0.918 | 0.820 | 0.869 | 0.918 |
| | **Attr2** | 0.902 | 0.869 | 0.836 | 0.918 | 0.902 | 0.902 | 0.902 | 0.902 | 0.902 | 0.902 | 0.902 | 0.951 |
| **DT6** | **Attr1** | 0.938 | 0.892 | 0.938 | 0.938 | 0.938 | 0.938 | 0.938 | 0.938 | 0.938 | 0.938 | 0.938 | 0.954 |
| | **Attr2** | 0.831 | 0.800 | 0.831 | 0.800 | 0.862 | 0.862 | 0.862 | 0.815 | 0.831 | 0.862 | 0.800 | 0.815 |
| **DT7** | **Attr1** | 0.442 | 0.863 | 0.916 | 0.537 | 0.916 | 0.916 | 0.916 | 0.547 | 0.916 | 0.884 | 0.916 | 0.411 |
| | **Attr2** | 0.411 | 0.811 | 0.832 | 0.453 | 0.832 | 0.832 | 0.832 | 0.495 | 0.916 | 0.779 | 0.832 | 0.337 |

Table 4.5: Overall precision measures of the different truth finding algorithms

| | CO | 2E | 3E | DP | AC | AS | AN | TF | SL | GL | CM | PR |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| **DT1** | 0.172 | 0.361 | 0.368 | 0.360 | 0.368 | 0.368 | 0.368 | 0.360 | 0.368 | 0.360 | 0.361 | 0.366 |
| **DT2** | 0.137 | 0.147 | 0.168 | 0.147 | 0.153 | 0.153 | 0.153 | 0.147 | 0.153 | 0.153 | 0.153 | 0.153 |
| **DT3** | 0.261 | 0.208 | 0.213 | 0.256 | 0.245 | 0.244 | 0.245 | 0.459 | 0.226 | 0.233 | 0.242 | 0.069 |
| **DT4** | 0 | 0 | 0.003 | 0 | 0 | 0 | 0 | 0.003 | 0.000 | 0.003 | 0.055 | 0.128 |
| **DT5** | 0.841 | 0.809 | 0.809 | 0.777 | 0.809 | 0.809 | 0.809 | 0.730 | 0.809 | 0.809 | 0.460 | 0.079 |
| **DT6** | 0.787 | 0.742 | 0.818 | 0.757 | 0.818 | 0.818 | 0.818 | 0.772 | 0.787 | 0.818 | 0.742 | 0.696 |
| **DT7** | 0.378 | 0.842 | 0.915 | 0.442 | 0.925 | 0.915 | 0.915 | 0.484 | 0.915 | 0.915 | 0.263 | 0.010 |
| **Average** | 0.368 | 0.444 | 0.471 | 0.391 | **0.474** | 0.472 | 0.473 | 0.422 | 0.465 | 0.470 | 0.325 | 0.214 |

In the second phase of the experiment, we want to investigate the performance of the best truth finding algorithm found in the first phase. We start by studying the evolution of the truth finding algorithm according to the number of conflicting values and sources. As previously explained, the goal of truth finding algorithms is to deterministically integrate conflicting statements and automatically compute a confidence value for each statement built on the trustworthiness level of its sources. Based on this, our hypothesis is that there is a positive relationship between the average number of conflicts and sources with the precision, recall, and F-measure. Next, we evaluate the correlation of the ranking of sources given by the truth finding algorithm and other rankings based on the precision, recall and F-measure scores. We also study the same correlation from the ranking produced by PageRank. To achieve this, we first get the ranked list of sources from PageRank and the truth finding algorithm. An example of the ranked lists for DT1 is shown in Table 4.8. The first column lists all the Web pages' URL for this topic. We then show the weight assigned by PageRank for each Web page along with its corresponding rank. The next two columns show the accuracy and rank inferred by the truth finding algorithm. In the last three columns, we show the ranked list of Web pages based on the precision, recall, and F-measure scores. We calculate the precision, recall, and F-measure score of a Web page by comparing the candidates extracted from it with the ground truth. We then evaluate the correlation between the ranking of

sources given by PageRank and the truth finding algorithm with the ranking from the precision, recall, and F-measure.

### 4.3.2 Result and Discussion

Measures in Table 4.3 and Table 4.4 show that at least one truth finding algorithm is better or equal in performance to PageRank, except for DT4 (Attr2), DT5 (Attr2), and DT6 (Attr1). In topic DT4, five truth finding algorithms (2E, AC, AS, AN, and SL) receive 0 in precision, while the highest score is achieved by Cosine (CO). They achieve the minimum precision score because these algorithms affect a false true label to the second attribute (Attr2) of the object.

According to Table 4.5 and Table 4.6, PageRank achieves the lowest average for the overall precision and recall. The best truth finding algorithms, in this case, are Accu and TruthFinder respectively (indicated by bold in Table 4.5 and Table 4.6). However, in terms of the trade-off between precision and recall, Figure 4.1 proves that Accu is an obvious choice. We conclude our analysis by stating that the set expansion problem can significantly gain effectiveness with truth finding, as demonstrated by the different results of our intensive experiments on various topics.

Table 4.6: Overal recall measures of the different truth finding algorithms

| | **CO** | **2E** | **3E** | **DP** | **AC** | **AS** | **AN** | **TF** | **SL** | **GL** | **CM** | **PR** |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| **DT1** | 0.423 | 0.891 | 0.908 | 0.887 | 0.908 | 0.908 | 0.908 | 0.887 | 0.908 | 0.891 | 0.787 | 0.891 |
| **DT2** | 0.252 | 0.271 | 0.308 | 0.271 | 0.280 | 0.280 | 0.280 | 0.271 | 0.280 | 0.280 | 0.280 | 0.280 |
| **DT3** | 0.262 | 0.209 | 0.213 | 0.257 | 0.245 | 0.245 | 0.245 | 0.460 | 0.226 | 0.233 | 0.242 | 0.069 |
| **DT4** | 0 | 0 | 0.004 | 0 | 0 | 0 | 0 | 0.004 | 0 | 0.004 | 0.078 | 0.180 |
| **DT5** | 0.869 | 0.836 | 0.836 | 0.803 | 0.836 | 0.836 | 0.836 | 0.754 | 0.836 | 0.836 | 0.475 | 0.082 |
| **DT6** | 0.800 | 0.754 | 0.831 | 0.769 | 0.831 | 0.831 | 0.831 | 0.785 | 0.800 | 0.831 | 0.754 | 0.692 |
| **DT7** | 0.379 | 0.842 | 0.916 | 0.442 | 0.916 | 0.916 | 0.916 | 0.484 | 0.916 | 0.916 | 0.263 | 0.011 |
| **Average** | 0.434 | 0.493 | 0.517 | 0.498 | 0.517 | 0.517 | 0.517 | **0.527** | 0.508 | 0.513 | 0.436 | 0.366 |

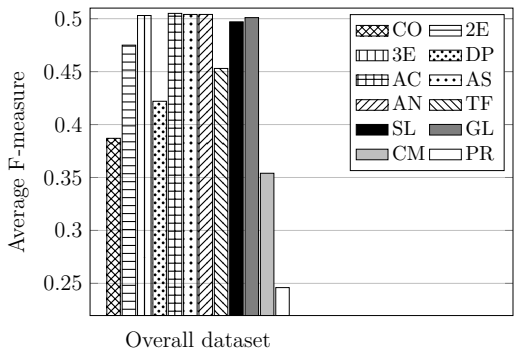


Figure 4.1: Average F-measure scores

Figure 4.1 compares average F-measure of truth finding algorithms and PageRank given the F-measure scores of every approach over our seven topics. Obviously, truth finding algorithms outperforms PageRank with an improvement ranging from 10% to 26% in terms of both precision and recall. Accu (AC) is the best truth finding model for ranking w.r.t. computed F-measure scores. The only exception is

Table 4.7: Average number of conflicts and sources of the different topics

| Topic | Avg_#Conflict | Avg_#Source | Precision | Recall | F-measure |
|---|---|---|---|---|---|
| **DT1** | 1.71 | 3.77 | 0.368 | 0.908 | 1.816 |
| **DT2** | 1.2 | 2.09 | 0.153 | 0.280 | 0.56 |
| **DT3** | 2.15 | 2.19 | 0.244 | 0.245 | 0.49 |
| **DT4** | 2.97 | 5.65 | 0 | 0 | 0 |
| **DT5** | 2.26 | 3.58 | 0.809 | 0.836 | 1.672 |
| **DT6** | 1.24 | 1.96 | 0.818 | 0.831 | 1.662 |
| **DT7** | 3.55 | 3.87 | 0.925 | 0.916 | 1.832 |

T4 where Accu receives the minimal score for all three metrics. We investigate this special case further by first running Majority Voting on the extracted candidates and then calculating the precision, recall, and F-measure of the result. We observe that the score is very low indicating that the candidates extracted from most of the sources are not correct yielding a biased truth finding process. The candidates extracted also contain a lot of noises, for example, two sources extracted "Singapore Dollar" and "Dollar" as the value of one of the attributes. While a person would consider these two values are referring to the same currency name for the country Singapore, it would be a challenge for a system to infer this without any human interaction. Nevertheless, Accu improves the precision, recall, and F-measure of our approach by 0.54%, 0.29%, and 0.51% respectively. Based on this, we conclude that leveraging a truth finding algorithm can indeed improve the performance of our approach.

For the next two experiments, we only select Accu (AC) as the representative of the truth finding algorithms since it is shown from earlier experiments that Accu achieves the best score for the three metrics (precision, recall, F-measure). Table 4.7 summarises the description of each topic in terms of the average number of conflict and the average number of sources (the second and third column of Table 4.7). The next three columns correspond to the precision, recall, and F-measure scores acquired by Accu for each dataset. From Table 4.7, we can see that DT2 has the lowest number of average conflicts, thus we expect it would achieve the lowest score in terms of precision, recall, and F-measure, but DT4 receives the lowest score instead. Although DT4 has an average number of conflicts of 2.97, it achieves a minimum score of 0 for all metrics among all the topics. We have explained a possible reason for DT4 achieving such a low score in the previous experiment. The same conclusion can be inferred when we relate the average number of sources with the precision, recall, and F-measure of the system. DT4 which has the highest average number of sources records the lowest scores for all three metrics, while topic DT7 which has the second highest average number of sources achieves the highest score in precision. Based on Table 4.7, we can not find any correlation (positive or negative), between the average number of conflicts and sources with the precision, recall, and F-measure scores. We conclude our analysis by stating that the average number of sources or conflicts do not influence the performance of the system in terms of precision, recall,

Table 4.8: Ranking of sources for topic DT1 - Calling Code

| URL | PageRank | | Accu | | Precision | | Recall | | F-measure | |
|---|---|---|---|---|---|---|---|---|---|---|
| | Weight | Rank | Acc. | Rank | Score | Rank | Score | Rank | Score | Rank |
| https://countrycode.org | 0.003 | 1 | 0.916 | 7 | 0.850 | 2 | 0.854 | 1 | 0.852 | 1 |
| http://www.howtocallabroad.com/codes.html | 0.002 | 2 | 0.925 | 5 | 0.832 | 3 | 0.828 | 2 | 0.830 | 2 |
| http://www.howtocallinternationally.com/codes.html | 0.002 | 3 | 0 | 10 | 0 | 10 | 0 | 10 | 0 | 10 |
| https://www.telstra.com.au/home-phone/international-calling/international-dialling-codes | 0.001 | 4 | 0.886 | 8 | 0.744 | 6 | 0.707 | 6 | 0.725 | 6 |
| http://www.areacodehelp.com/world/country-calling-codes.shtml | 0.001 | 5 | 0.929 | 4 | 0.758 | 4 | 0.812 | 3 | 0.784 | 3 |
| https://weendeavor.com/telephone/international-country-codes/ | 0.001 | 6 | 0.792 | 9 | 0.185 | 9 | 0.142 | 9 | 0.161 | 9 |
| http://wikitravel.org/en/List_of_country_calling_codes | 0.001 | 7 | 0.942 | 3 | 0.702 | 8 | 0.611 | 7 | 0.653 | 7 |
| http://www.nationsonline.org/oneworld/international-calling-codes.htm | 0.001 | 8 | 1 | 1 | 0.854 | 1 | 0.711 | 5 | 0.776 | 4 |
| http://www.1areacodescountrycodes.com/all-country-code-numbers-countrycodes.htm | 0.001 | 9 | 0.981 | 2 | 0.704 | 7 | 0.339 | 8 | 0.458 | 8 |
| http://www.ldpost.com/countries/ | 0.001 | 10 | 0.918 | 6 | 0.746 | 5 | 0.774 | 4 | 0.760 | 5 |

Table 4.9: Correlation of ranking of sources

| | | DT1 | DT2 | DT3 | DT4 | DT5 | DT6 | DT7 | Average |
|---|---|---|---|---|---|---|---|---|---|
| **PageRank** | **Precision** | -0.10 | -1.00 | 0.09 | 0.39 | -0.14 | -0.36 | -1.00 | -0.30 |
| | **Recall** | -0.03 | -0.29 | 0.33 | 0.32 | -0.38 | -0.36 | -1.00 | -0.20 |
| | **F-measure** | 0.03 | -1.00 | 0.33 | 0.39 | -0.38 | -0.36 | -1.00 | -0.28 |
| **Accu** | **Precision** | 0.61 | 0.53 | -0.26 | -0.35 | 0.44 | -0.36 | 1.00 | 0.23 |
| | **Recall** | 0.50 | 1.00 | -0.43 | -0.46 | 0.22 | -0.36 | 1.00 | 0.21 |
| | **F-measure** | 0.52 | 0.53 | -0.43 | -0.35 | 0.22 | -0.36 | 1.00 | 0.16 |

and F-measure.

Table 4.8 shows the ranking of sources for topic DT1. The ranking of sources for the rest of the topics can be found in Appendix A. We show the weight assigned by PageRank, the accuracy inferred by Accu, and also the precision, recall, and F-measure for each Web page along with its corresponding rank. All of the rankings produced by PageRank, Accu, or rankings by precision, recall, and F-measure do not agree on the order of importance of the URLs. This is also true for all of the ranking of sources for the rest of the topics as shown in Appendix A. An interesting case can be seen in Table 4.8 where a Web page (row 3 in Table 4.8) receives a weight of 0.002 from PageRank although it achieves 0 scores for precision, recall, and F-measure. Note that when we build our graph of entities, we add any Web page from which we can infer wrappers and the corresponding candidates. This opens the possibility of adding Web pages which extract "junk" candidates. Web pages are assigned a weight after we run the PageRank algorithm on the graph. When we calculate the precision with refers to the ground truth, a Web page receives a 0 score because it extracts only irrelevant candidates.

In Table 4.8, we are presented with five different rankings for the same topic DT1 (rankings for other topics can be found in Appendix A). We further investigate the correlation between the rankings in Table 4.8 by calculating their Kendall's Tau score. The result is shown in Table 4.9. Please note that a Kendall's Tau score of less than zero indicates that there is a negative association between the two lists which are in comparison. In general, the ranking of sources given by Accu has a positive association with the ranking of sources using the precision, recall, and F-measure scores. In DT6, the performance of PageRank and Accu is equal. We look at the experiment result closely and find that Accu infers its ground truth based on three Web pages. Among these three Web pages, the one that ranks the highest actually extract candidates which contain many special characters such as é, &amp, etc. Subsequently, this Web page achieves low rank in the precision, recall, and F-measure rankings. Normalising special characters into its standard form before inserting the candidates into our graph of entities or perform truth finding algorithm is a possible way of improving the result. On the other hand, PageRank outperforms Accu in topics DT3 and DT4. In both cases, Accu suffers from the fact that the majority of the sources are constantly wrong which resulted in a biased truth finding process. Moreover, we find that the sources have many candidates which are not included in the ground truth. Nevertheless, we can conclude that leveraging truth finding algorithm into our approach can also improve the ranking of sources.

## 4.4 Conclusion

We have proposed to leverage the truth finding algorithm to candidates extracted by the set of tuples expansion. We empirically showed that truth finding algorithms can significantly improve the truthfulness of the extracted candidates. We intend to use truth finding algorithms into our STEP system, by also tackling difficult cases. However, an in-depth analysis of the truth finding algorithm integrated into our approach shows that the average number of sources or conflicts do not affect the performance of the system. The ranking of sources is also positively affected by leveraging truth finding algorithm. We have shown this by computing the Kendall's Tau score of the ranking of sources provided by Accu with the ranking of sources in terms of the precision, recall, and F-measure scores.

# Chapter 5

# Tuple Reconstruction

Set of tuples expansion system, such as STEP [35], extracts information from the World Wide Web in the form of tuples. Particularly, given a set of tuple examples <*Indonesia*, *Jakarta*, *Indonesian Rupiah*>, <*Singapore*, *Singapore*, *Singapore Dollar*>, STEP returns a list consisting of the name of a country, its capital city, as well as the name of the currency. STEP applies a voting ranking mechanism based on PageRank on the tuple candidates. PageRank [77] is primarily used by Google to measure the importance or relevance of Web pages with refers to a query given by the user. Each link from other Web pages to a target Web page is treated as a vote for the target Web page. The Web pages are then ranked according to its corresponding vote count. In the particular case of STEP [35], a graph is built by defining a set of entities (Web pages, wrappers, seeds, domains, and tuple candidates) as nodes and the relationships between entities as links (edges). A tuple candidate node is only linked with wrapper nodes. This means that the rank of a tuple candidate in the final list only depends on the number of wrappers used to extract that particular candidate. The wrappers in STEP are regular expression-based wrappers. They are generated by comparing the contexts of a pair of occurrences of seeds in a Web page. This method may produce more tuple candidates, but on the other hand, it is highly likely to extract *false* ones, i.e. candidates whose elements are not from the same semantic class as the seeds. These false candidates may receive as many vote count as the true candidates due to the nature of the graph of entities. They may have more votes if they are extracted by many wrappers from the same page or by other wrappers from other Web pages. The reverse situation where true candidates have more votes than the false ones may also occur. Nevertheless, ensuring that extracting true candidates rather than false ones is utmost importance. This is the motivation of this research where we propose a solution to ensure that true candidates will always have more votes than the false ones and rank higher on the list. Our proposed solution incorporates a tuple reconstruction phase by combining binary tuples. We explain briefly our proposed method next.

Tuples such as those given as examples earlier consist of $n$ elements, i.e. the term $n$-elements tuple. Each element is a value for an attribute of a real-world *entity* (object). We use the term element and attribute interchangeably throughout this chapter. Among these $n$ elements, one of them can be considered as the key

attribute while all the remaining elements are the values of the properties of the key attribute. Consider the tuple <*Indonesia*, *Jakarta*, *Indonesian Rupiah*> where $n$=3, one can surely identify that the key attribute in the tuple is the country "Indonesia" while "Jakarta" and "Indonesian Rupiah" are the values for attributes "capital city" and "currency" of Indonesia respectively. By noticing this key attribute notion, the previous tuple can be decomposed into two *2*-elements tuples (*binary* tuples). We accomplish this by pairing the attribute values with the key attribute: <*Indonesia*, *Jakarta*> and <*Indonesia*, *Indonesian Rupiah*>. Suppose that we have the knowledge of the two binary tuples a priori (by querying an ontology or extracting binary tuples with the same relation), add them to the graph of entities as new nodes, and create links to the the candidate tuple <*Indonesia*, *Jakarta*, *Indonesian Rupiah*>, then the candidate tuple should gain more weight after we perform the PageRank algorithm. Subsequently, the candidate tuple should also rank higher in the final list presented to the user. Based on this intuition, we propose the problem of tuple reconstruction. Starting from the binary tuples, we reconstruct tuples of $n$-elements ($n$>2). The reconstructed tuples can then be added to the graph of entities with the purpose of enriching the graph. We show that the addition of these reconstructed tuples can improve the confidence in the extracted candidate tuples. We organise this chapter as follows. Section 5.1 gives a formal definition of the tuple reconstruction problem. We then detail our proposed approach to solving the problem in Section 5.2. In Section 5.3, we evaluate the performance of the proposed solution while Section 5.4 concludes this chapter.

## 5.1 Problem Definition

We consider fixed finite sets of *attribute labels* $\mathcal{A}$ and *values* $\mathcal{V}$. Let $n$ represents the number of attributes (*elements*). We formally defined the following.

**Definition 5.** *A* tuple *$t$ consists of a set of $v$ where $v$ is a mapping $v : \mathcal{A} \rightarrow \mathcal{V}$.*

To form a tuple, we must first select a set of attributes for a real-world entity. For instance, we select the entity to be a country, and for this entity, we choose $n$=3 attributes which include the capital city, currency code, and the currency name. Each of these attributes can take exactly one value from the $\mathcal{V}$ domain. Suppose we take "Indonesia", "Jakarta", "IDR", and "Indonesian Rupiah" as the values for the attributes then we form the tuple <*Indonesia*, *Jakarta*, *IDR*, *Indonesian Rupiah*>. We define next the key attribute of a tuple. Let $i$ denotes the index of each element in a tuple respectively.

**Definition 6.** *The* key attribute *is the identifier of a tuple and comprised of the first n-1 elements for n-elements tuple where n≥2.*

The key attribute acts as the identifier of the tuple. For example, in the tuple <*Indonesia*, *Jakarta*>, the key attribute is "Indonesia" where it represents the name of a country.

Following the above definitions, we can proceed with defining the tuple reconstruction problem. The tuple reconstruction problem deals with the task of constructing $n$-elements tuples from binary ones. We propose here two methods of reconstructing tuples, i.e. *strict* and *loose*. As we described briefly earlier, an $n$-elements tuple can be constructed from a pair of ($n$-1)-elements tuples for $n$>2. These two ($n$-1)-elements tuples should have the same elements ordered from index $i$=1 to $n$-2.

**Definition 7.** *A pair of n-elements tuples can be merged to construct an (n+1)-elements tuple if they both have the same n-1 key attribute with n≥2 (strict).*

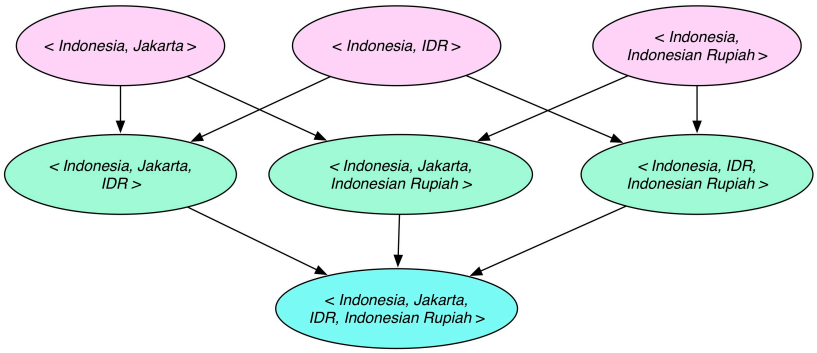


Figure 5.1: Tuple reconstruction illustration

Consider Figure 5.1 where on the first level of the graph we have three binary tuples <*Indonesia*, *Jakarta*>, <*Indonesia*, *IDR*>, and <*Indonesia*, *Indonesian Rupiah*>. By applying Definition 7 on the binary tuples, we generate the nodes in the second level, i.e. <*Indonesia*, *Jakarta*, *IDR*>, <*Indonesia*, *Jakarta*, *Indonesian Rupiah*>, and <*Indonesia*, *IDR*, *Indonesian Rupiah*>. Two out of the three tuples in the second level have the same key attribute by Definition 6. Thus, again by applying Definition 7 on the second level, we get in the third level the tuple <*Indonesia*, *Jakarta*, *IDR*, *Indonesian Rupiah*> by combining the leftmost and middle nodes.

**Definition 8.** *Two n-elements tuples (n > 2) can be merged to form (n+1)-elements tuples if they have the same values for the first n-2 elements and differ in one value of the remaining elements (loose).*

The *strict* method of generating tuples may lead to the possibility of overlooking some potential candidates. We can not generate the previous tuple by combining either the leftmost or middle node with the rightmost node on the second level of the tree because it violates Definition 6 and 7. If we take a closer look at the three nodes, we can see that they share the same $n$-2 key attribute and only differ in one of the attribute values in the remaining $n$-1 attributes. Thus, we define another method of generating the tuples as in Definition 8. Using Definition 8, we are able to generate the tuple on the third level by combining the rightmost node on the second level with any of the other two nodes. We refer to the tuples generated from Definition 7 and Definition 8 as *reconstructed tuples*.

## 5.2 Proposed Approach

The tuple reconstruction process as described in the previous section requires the availability of binary tuples at the start of the process. These binary tuples can be retrieved by leveraging any of the set expansion systems that are able to extract binary relations, i.e. DIPRE [16], Ext. SEAL [30], STEP [35], etc. However, our motivation in proposing the tuple reconstruction problem is to assist STEP in generating a more trustworthy list of candidates. For this reason, we incorporate the tuple reconstruction as a part of the process of building the graph of entities in STEP [35]. This tuple reconstruction process can only be carried out for seeds with the number of elements $n > 2$. We first show the tuple reconstruction process can be applied by leveraging *FindSeedsOccurrences* and *WrapperGeneration* algorithms of STEP (Algorithm 3 and 4). Later, we incorporate directly the tuple reconstruction process into building the graph of entities by modifying Algorithm 7.

**Algorithm 9:** FindBinaryTuples

**Input :** A set of seeds $R = \{T_1, T_2, ..., T_{Ns}\}$
**Output:** A list of binary tuples $R'$

```
1  B = ∅;
2  for i ← 1 to n − 1 do
3      R' = ∅;
4      foreach t ∈ R do
5          R' = R' ∪ {< t[0], t[i] >};
6      end
7      D = {FindDocuments(R')};
8      foreach d ∈ D do
9          Wrappers = {WrapperGeneration(FindSeedsOccurrences(R', d))};
10         foreach w ∈ Wrappers do
11             C = {extract binary tuples from d using w};
12             B = B ∪ C;
13         end
14     end
15 end
16 return B;
```

Algorithm 9 details the process of finding binary tuples given a set of seeds with elements $n > 2$. We use as our running example the following set of tuple seeds: <*Indonesia*, *Jakarta*, *Indonesian Rupiah*, *IDR*>, <*Singapore*, *Singapore*, *Singapore Dollar*, *SGD*> and <*Malaysia*, *Kuala Lumpur*, *Malaysian Ringgit*, *MYR*>. We assume that for each tuple seed the key element is always the first element of the tuple. We first decompose the seeds into binary ones starting from the second element until the $n$-th element (line 2). Next, for each of the tuple seeds, we compose the binary seeds by pairing the first element of the tuple with the $i$-th element (line 4-6). Thus, the tuple seeds given as the running example will be decomposed into <*Indonesia*, *Jakarta*>, <*Singapore*, *Singapore*>, and <*Malaysia*, *Kuala Lumpur*>

on the first run, while <*Indonesia*, *Indonesian Rupiah*>, <*Singapore*, *Singapore Dollar*>, and <*Malaysia*, *Malaysian Ringgit*> on the second, and finally <*Indonesia*, *IDR*>, <*Singapore*, *SGD*>, and <*Malaysia*, *MYR*>. Using these binary seeds as the new set of seeds, we retrieve a set of Web pages containing them (line 7). For each of the Web page retrieved we find the occurrences of the seeds using Algorithm 3. The occurrences and the document are then fed into Algorithm 4 to infer a set of wrappers. We then extract binary tuples from the document using the set of wrappers (line 8-14).

**Algorithm 10:** ReconstructTuple

**Input :** A set of binary tuples $B$, an integer $n$
**Output:** A set of reconstructed tuples $B'$

1 $B' = \emptyset$;
2 $B' = B' \cup B$;
3 **for** $i \leftarrow 2$ **to** $n - 1$ **do**
4 $\quad b = \{$retrieve tuples from $B'$ where number of elements $= i\}$;
5 $\quad$ **foreach** $b_i \in b$ **do**
6 $\quad\quad c = \emptyset$;
7 $\quad\quad$ **if** *strict* **then**
8 $\quad\quad\quad c = \{$retrieve tuples from $b$ where elements $e_1, ..., e_{i-1}$ of the tuples are the sames as in $b_i\}$;
9 $\quad\quad$ **else**
10 $\quad\quad\quad c = \{$retrieve tuples from $b$ where elements $e_1, ..., e_{i-2}$ of the tuples are the sames as in $b_i$ and for elements $e_{i-1}, ..., e_i$ they differ in only one value$\}$;
11 $\quad\quad$ **end**
12 $\quad\quad$ **foreach** $c_i \in c$ **do**
13 $\quad\quad\quad t = \{$combine $b_i$ with $c_i\}$;
14 $\quad\quad\quad B' = B' \cup t$;
15 $\quad\quad$ **end**
16 $\quad$ **end**
17 **end**
18 **return** $B'$;

We start the reconstruction process by combining the binary tuples using Definition 7 or Definition 8 to form tuples with $n$=3. Next, we combine *3*-elements tuples to form *4*-elements tuples, and so on. The process is stopped when we finally have constructed $n$-elements tuples. We use Algorithm 10 to reconstruct $n$-element tuples from the set of binary tuples. At the start of the process, $B'$ consists of all the binary tuples extracted by Algorithm 9. We repeat the following process for $i$=2 to $n - 1$ (line 3). On the first iteration, we retrieve a set of binary (*2*-elements) tuples $b$ from $B'$ (line 4). For each element $b_i$ of $b$, depending on the tuple reconstruction method, we collect a set of tuples $c$ from $b$ that satisfy Definition 7 or Definition 8 (line 6-11). We then combine $b_i$ with each $c_i$ of $c$ and add the reconstructed tuple

to $B'$ (line 12-15). On the second iteration, we retrieve *3*-elements tuples from $B'$ and repeat the process (line 5-16). We continue repeating the process until we have constructed all $n$-elements tuples.

We explain next how we incorporate the tuple reconstruction process directly to build the graph of entities of STEP. STEP takes as input a set of tuple seeds where each of the seeds consists of $n$-elements. The *strict* method of tuple reconstruction requires that $n \geq 2$ while from the definition of the *loose* method the requirement is at least $n = 3$. Let us reconsider the set of tuple seeds of our running example: <*Indonesia*, *Jakarta*, *Indonesian Rupiah*, *IDR*>, <*Singapore*, *Singapore*, *Singapore Dollar*, *SGD*> and <*Malaysia*, *Kuala Lumpur*, *Malaysian Ringgit*, *MYR*> where $n = 4$. Our goal is to reconstruct tuples by first extracting binary tuples from the given seeds and recursively combine the binary tuples to generate $n$ elements tuples. We modify Algorithm 7 to incorporate tuple reconstruction into the process of building the graph of entities as shown in Algorithm 11 and Algorithm 12. In Algorithm 11, we find binary tuples from the given set of seeds. We decompose each tuple seeds into groups of binary seeds constructed from pairing the key element and one of the remaining elements alternately (line 4-6). Next, we retrieve a set of Web pages containing the binary seeds. For each of the Web page, we generate wrappers and extract the binary tuples. We add the entities to the graph as well as create the edges between them as summarised in Table 3.2 (line 8-27). The result of the algorithm is a graph with the addition of binary tuples as the nodes as well as the relationships between the nodes as the edges and the set of binary tuples. We then reconstruct the tuples from the set of binary tuples using Algorithm 12. We add all reconstructed tuples as new nodes on the graph but leaving out the binary ones as it has been added to the graph earlier (line 3). We start adding the reconstructed tuples to the graph from tuples of length 3 to $n$ (line 3-4). For each tuple in the selected set of the reconstructed tuples, we find its *ancestors* and *descendants*. The ancestors, in this case, are the set of ($n$-1)-elements tuples that can be used to construct the corresponding $n$ element tuple. Suppose that <*United Kingdom*, *London*, *Poundsterling*> is a reconstructed tuple, the ancestors of this tuple are <*United Kingdom*, *London*>, <*United Kingdom*, *Poundsterling*>. We can also add <*London*, *Poundsterling*> to the set of ancestors if we generate the corresponding tuple using Definition 8. For each of the ancestors, we create an edge to the corresponding tuple and label it as *CONSTRUCT* (line 8-10). On the other hand, the descendants are the set of $n$+1 reconstructed tuples that are generated from the corresponding tuple. For the reconstructed tuple <*United Kingdom*, *London*, *Poundsterling*>, an example of its descendant is <*United Kingdom*, *London*, *Poundsterling*, *GBP*>. We then create an edge from each of the descendants to the corresponding tuple and assign *INV_CONSTRUCT* as the label (line 12-14).

Figure 5.2 shows an example of the graph produced by Algorithm 12. The lower part of the figure (denoted by the rectangle with the dashed line) is the tuple reconstruction process, while the upper part is the graph of entities constructed by STEP. The tuple reconstruction is carried out in parallel with the process of building the graph of entities. Both processes start with the given set of tuple seeds as its input. For the tuple reconstruction, we then decompose the tuple seeds into binary seeds.

**Algorithm 11:** RetrieveBinaryTuples

**Input :** A graph of entities $G = (V, E)$, a set of $n$ elements tuple seeds $R = \{T_1, T_2, ..., T_{Ns}\}$

**Output:** A graph of entities $G' = (V', E')$, a set of binary tuples $B$

```
1  G' = G;
2  for i ← 1 to n − 1 do
3      R' = ∅;
4      foreach t ∈ R do
5          R' = R' ∪ {< t[0], t[i] >};
6      end
7      P = {FindDocuments(R')};
8      foreach P_i ∈ P do
9          V' = V' ∪ P_i;
10         d = FindDomain(P_i);
11         V' = V' ∪ d;
12         E' = E' ∪ {< P_i, BELONG_TO, d >, <
             d, INV_BELONG_TO, P_i >};
13         foreach r in R' do
14             E' = E' ∪ {< P_i, INV_RETRIEVE, r >, < r, RETRIEVE, P_i >
               };
15         end
16         Wrappers = {WrapperGeneration(FindSeedsOccurrences(R', P_i))};
17         foreach w ∈ Wrappers do
18             V' = V' ∪ w;
19             E' = E' ∪ {< P_i, GENERATE, w >, <
               w, INV_GENERATE, P_i >};
20             C = {extract binary tuples from P_i using w};
21             foreach c_i ∈ C do
22                 V' = V' ∪ c_i;
23                 E' = E' ∪ {< w, EXTRACT, c_i >, <
                     c, INV_EXTRACT, w >};
24             end
25             B = B ∪ C;
26         end
27     end
28     return G', B;
29 end
```

**Algorithm 12:** BuildGraphWithReconstructedTuples

**Input :** A graph of entities $G = (V, E)$, a set of $n$ elements tuple seeds $R = \{T_1, T_2, ..., T_{Ns}\}$

**Output:** A graph of entities with reconstructed tuples $G' = (V', E')$

```
1  G', B = RetrieveBinaryTuples(G);
2  B' = {ReconstructTuple(B, n)};
3  V' = V' ∪ B' \ B;
4  for i ← 3 to n do
5      B_new = {Select tuples of length i from B'};
6      foreach t ∈ B_new do
7          Anc = {Find ancestors of t};
8          foreach a ∈ Anc do
9              E' = E' ∪ {< a, CONSTRUCT, t >};
10         end
11         Des = {Find descendants of t};
12         foreach a ∈ Des do
13             E' = E' ∪ {< a, INV_CONSTRUCT, t >};
14         end
15     end
16 end
17 return G';
```

Since a binary seed contains a key element (the first element of the tuple seed) and an attribute value, after decomposing the tuple seeds we will have three sets of binary seeds. In Figure 5.2, we only show one set of the binary seeds (<*Indonesia*, *Jakarta*>, <*Singapore Singapore*>, and <*Malaysia*, *Kuala Lumpur*>). Using this binary seeds then we retrieve a set of Web pages, infer wrappers from each of the Web pages, and finally extract other binary tuples (<*United Kingdom*, *London*>, <*United Kingdom*, *Poundsterling*>, and <*United Kingdom*, *GBP*>). From these binary tuples, we then reconstructed *3*-elements tuples (<*United Kingdom*, *London*, *Poundsterling*>, <*United Kingdom*, *London*, *GBP*>. We then relate the reconstructed tuples to its ancestors and descendants. In this setting, the descendants are $n$-elements tuples which are extracted by STEP. By reconstructing tuples from the set of binary seeds, we are actually giving more votes to the $n$-elements tuple of STEP which in turn increase the confidence of the extracted tuples.

## 5.3 Performance Evaluation

Our goal in this performance evaluation is to show that the proposed approach of tuple reconstruction can be directly integrated into the set of tuples expansion problem. Furthermore, we also intend to prove that tuple reconstruction present a few advantages to the expansion process: (1) it introduces new $n$-elements candidate tuples which might not be extracted by STEP and in turns can improve the pre-

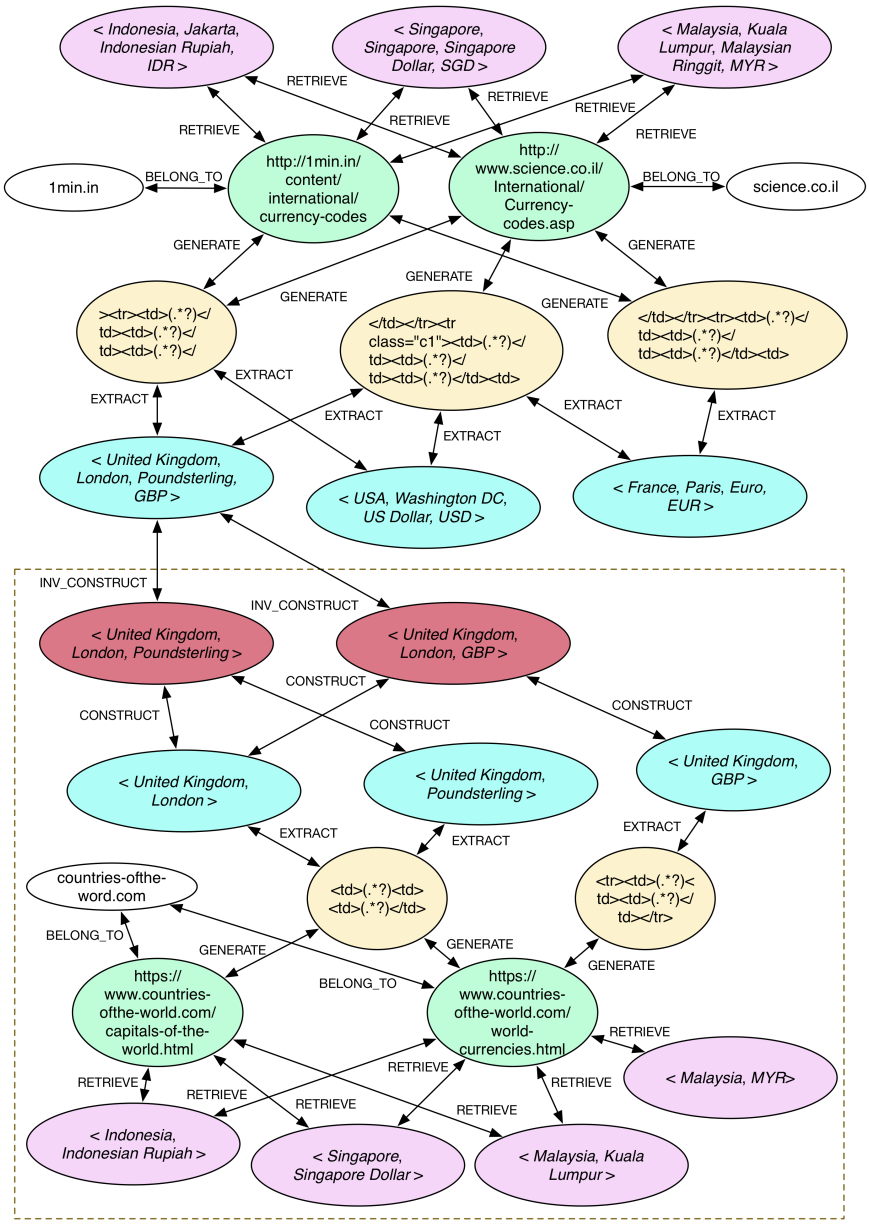


Figure 5.2: Graph of entities with reconstructed tuples

cision or recall, (2) provided that the *2*-elements, ..., ($n$-1)-elements reconstructed tuples are correct, they can boost the confidence of the $n$-elements tuples which are extracted by STEP.

### 5.3.1 Experiment Setting

We implement the proposed approach of tuple reconstruction as a system which can extract tuples from the Web as well as reconstruct tuples from binary ones. We evaluate the performance of the system based on the extracted candidates. We define several topics and compare the extracted candidates to the manually constructed

Table 5.1: Topics used for performance evaluation in tuple reconstruction

| Topics | Seeds | Reference | No. Entries |
|---|---|---|---|
| DT1 - Currency | *<Indonesia, Jakarta, Indonesian Rupiah, IDR>, <Singapore, Singapore, Singapore Dollar, SGD>, <France, Paris, Euro, EUR>* | `https://www.ibpsguide.com/country-capitals-and-currency` | 244 |
| DT2 - Car brands | *<Alfa Romeo, Italy, Fiat S.p.A, alfaromeo.com>, <Chevrolet, USA, General Motors, chevrolet.com>* | `http://www.autocarbrands.com/car-brands-information-list/` | 104 |
| DT3 - Airports | *<CGK, Soekarno-Hatta International Airport, Jakarta, Indonesia>, <CDG, Charles De Gaulle International Airport, Paris, France>, <SIN, Changi International Airport, Singapore, Singapore>* | `http://http://www.nationsonline.org/oneworld/IATA_Codes` | 9220 |

ground truth for each topic. The ground truth of each topic is collected from a reference Website and excluded from the set of tuples expansion process. We have inspected and verified that the resulting tables are correct and generally complete. The topics, the set of seeds, the corresponding reference, and the number of entries contained in the reference are presented in Table 5.1. We briefly summarise every topic as follows.

- DT1 - Currency. This topic associates countries with their corresponding capital city, currency name, and currency code. We choose `https://www.ibpsguide.com/country-capitals-and-currency` as the reference Website which has 244 entries. We use the following set of seeds: *<Indonesia, Jakarta, Indonesian Rupiah, IDR>, <Singapore, Singapore, Singapore Dollar, SGD>, <France, Paris, Euro, EUR>*.

- DT2 - Car brands. This topic associates automotive brands with their manufacturing country, the parent company, and its official Website. We choose `http://www.autocarbrands.com/car-brands-information-list/` as the reference Website which has 104 entries. We use the following set of seeds: *<Alfa Romeo, Italy, Fiat S.p.A, alfaromeo.com>, <Chevrolet, USA, General Motors, chevrolet.com>*.

- DT3 - Airports. This topic associates airport codes with their name, the location, and the country. We choose `http://http://www.nationsonline.org/oneworld/IATA_Codes` as the reference Website which has 9220 entries. We use the following set of seeds: *<CGK, Soekarno-Hatta International Airport, Jakarta, Indonesia>, <CDG, Charles De Gaulle International Airport, Paris, France>, <JFK, John F. Kennedy International Airport, New York, USA>*.

The metrics used in the experiment are precision, recall, and F-measure. The precision and recall are calculated using Equation 3.3, while the F-measure using Equation 4.3.

In STEP [35], we use Algorithm 7 to build the graph of entities consisting of seeds, Web pages, domains, wrappers and candidates as the nodes and the relationships between the entities as its edges. We then introduce the tuple reconstruction problem and integrate a solution to the problem directly to STEP in Algorithm 12. To evaluate the performance of our proposed approach of tuple reconstruction, we compare the candidates extracted by STEP which uses Algorithm 7 and Algorithm 12

Table 5.2: Comparison of Precision, Recall, and F-measure of STEP with and without tuple reconstruction

| Topic | Number of Candidates | | Precision | | Recall | | F-measure | |
|---|---|---|---|---|---|---|---|---|
| | w/o TR | With TR | w/o TR | with TR | w/o TR | With TR | w/o TR | With TR |
| DT1 | 193 (187) | 304 (207) | 0.968 | 0.68 | 0.766 | 0.848 | 0.855 | 0.754 |
| DT2 | 53 (44) | 53 (44) | 0.83 | 0.83 | 0.423 | 0.423 | 0.526 | 0.526 |
| DT3 | 451 (326) | 1235 (968) | 0.035 | 0.104 | 0.722 | 0.783 | 0.066 | 0.183 |

in building the graph of entities. We calculate the precision and recall using Equation 3.3, as well as the F-measure using Equation 4.3 on the extracted candidates with refers to the ground truth. First, we run STEP that implements Algorithm 7 on each topic with the given set of seeds and then extract the candidates. We refer to this as STEP without tuple reconstruction (w/o TR). Next, we repeat the same process but we replace Algorithm 7 with Algorithm 12 to build the graph of entities (with TR). Table 5.2 compares the number of candidates extracted, precision, recall, and F-measure of STEP with (with TR) and without tuple reconstruction (w/o TR). The number in brackets in the number of candidates denotes the number of correct candidates, i.e. candidates that are present in the ground truth.

### 5.3.2 Result and Discussion

From Table 5.2, we can see that integrating tuple reconstruction into the set of tuples expansion system improve the number of extracted candidates and the recall of the system, especially in topics DT1 and DT3. In topic DT2, the performance of STEP with and without the tuple reconstruction is the same. We investigate further the impact of the tuple reconstruction for each topic and detail the result next.

We start with topic DT2 as this is an interesting case where using tuple reconstruction does not improve the performance of STEP in terms of the candidates extracted as well as the three metrics we use. As we give the set of seeds <*Alfa Romeo*, *Italy*, *Fiat S.p.A*, *alfaromeo.com*>, <*Chevrolet*, *USA*, *General Motors*, *chevrolet.com*> as input to STEP, we notice that the search engine only returns several Web pages. `http://www.autosaur.com/car-brands-complete-list/` is one of the URLs returned from which STEP manages to extract 53 candidates. Out of these candidates, 9 are rejected with refers to the ground truth for various reasons, i.e. noise in the candidates (HTML tags), minor difference in the writing ("&amp" instead of "&"), or no reference in the ground truth ("adical sportscars" is not contained by our ground truth). On the second run of STEP (with tuple reconstruction), the number of extracted candidates is exactly the same. This means that the tuple reconstruction fails to add even a single candidate. We take a look into the process to find out the reason for the failure. We decompose each of the tuple seeds into three groups of binary seeds, i.e. first group: <*Alfa Romeo*, *Italy*>, <*Chevrolet*, *USA*>, second group: <*Alfa Romeo*, *Fiat S.p.A*>, <*Chevrolet*, *General Motors*> and <*Alfa Romeo*, *alfaromeo.com*>, <*Chevrolet*, *chevrolet.com*>. From the first group of the binary seeds, we try to find relations containing the name of an automobile brand and their country of origin. We manage to extract 84 can-

didates. Using the second group of binary seeds, from the set of URLs returned by the search engine, we only manage to extract 22 candidates. If we combine the two sets of candidates, we expect to have 22 reconstructed tuples of length $n$=3 with the relations of automobile car brands, the country of origin and the parent company. However, when we use the last group of binary seeds, we fail to extract any candidates from the set of URLs. This prevents us from reconstructing tuple of length $n$=3 with the official Website of the automobile brand as one of its element. Furthermore, this failure also impedes us from constructing tuples of length $n$=4. We find this as the reason why the tuple reconstruction does not contribute any candidate to the final set of candidates.

Tuple reconstruction shows its potential of improving the number of candidates on topics DT1 and DT3. In the former case, STEP without tuple reconstruction extracts 193 candidates. This number is improved by tuple reconstruction by 57% to 304 candidates. From the three groups of binary seeds, i.e. <*Indonesia*, *Jakarta*>, <*Singapore*, *Singapore*>, <*France*, *Paris*>; <*Indonesia*, *Indonesian Rupiah*>, <*Singapore*, *Singapore Dollar*>, <*France*, *Euro*>; <*Indonesia*, *IDR*>, <*Singapore*, *SGD*>, <*France*, *EUR*>; we manage to extract 297, 276, and 164 candidates respectively. From these three groups of binary candidates, we are then able to reconstruct 111 tuples of length $n$=4, out of which 20 of them are accepted by the ground truth. The 20 new candidates help to improve the recall of STEP from 0.766 to 0.848. However, the remaining 91 reconstructed tuples contribute to the decrease of the precision from 0.968 to 0.68. The most significant improvement of tuple reconstruction is shown in topic DT3. Without tuple reconstruction, STEP only extracts 451 candidates. This number is improved more than tripled to 1235 candidates when we integrate tuple reconstruction into the expansion process. The reconstructed tuples also contribute to the improvement of the recall from 0.722 to 0.783. However, the precision scores of both runs of STEP (with and without tuple reconstruction) are really low. In topic DT3, we try to extract relations of all airports code with their respective names, locations, and countries with `http://www.nationsonline.org/oneworld/IATA_Codes` as the reference Website. The reference Website itself consists of 9220 entries. This total number of entries in the ground truth is a magnitude higher than the extracted candidates of 451 and 1235 for STEP without and with tuple reconstruction respectively. We investigate further the candidates extracted by STEP without tuple reconstruction and find that they consist of only the major airports in the world. Tuple reconstruction then helps in adding 784 new candidates. We find that these new candidates are actually all airports in the USA alone. Indeed, when we check on the URLs returned by the search engine using the decomposed binary seeds <*CGK*, *Soekarno-Hatta International Airport*>, <*CDG*, *Charles De Gaulle International Airport*>, <*JFK*, *John F. Kennedy International Airport*>; <*CGK*, *Jakarta*>, <*CDG*, *Paris*>, <*JFK*, *New York*>; <*CGK*, *Indonesia*>, <*CDG*, *France*>, <*JFK*, *USA*>; we manage to extract binary relations of airport codes and its names, airport codes and the corresponding locations of city and country from the same URL `http://www.airportcodes.us/airports-by-name.htm`. We are certain that if the URLs returned by the search engine point to *relevant*

Web pages, i.e. Web pages which contain the targeted binary relations, the tuple reconstruction would add many more new candidates. Nevertheless, from the result of the performance evaluation, we conclude that tuple reconstruction has the potential of finding candidate tuples which are not extracted by the set of tuple expansion system.

## 5.4 Conclusion

We propose the tuple reconstruction problem which deals with the task of reconstructing tuples of length $n$ from binary ones. We present two methods for the task of tuple reconstruction. The solution of the problem is integrated directly into the process of building the graph of entities in a set of tuples expansion system. We evaluate the performance of the solution in terms of precision, recall, and F-measure on three real-world topics. Empirical results show that new candidates can indeed be added by the tuple reconstruction which previously are not found by the set of tuples expansion system without the tuple reconstruction. However, the integration of tuple reconstruction into the set of tuples expansion system must be carefully considered as it improves the recall but on the other hand can also negatively impact the precision.

# Part II

# Expanding the Set of Examples Semantically

# Chapter 6

# Topic Labelling with Truth Finding

## 6.1 Introduction

In the first part of the thesis, we focus our effort on expanding the set of $n$-elements tuples ($n$ is the number of elements) by giving a set of tuple seeds of equal length and a corpus. To achieve this, we infer wrappers from the set of occurrences of the seeds in the corpus and apply the inferred wrappers on the corpus to retrieve the candidates. This approach relies heavily on the availability of documents in the corpus which contain the seeds. Failure to retrieve any occurrences of the seeds in the documents leads to none of the candidates being extracted. In the second part of the thesis, we investigate the possibility of expanding the tuple seeds semantically, i.e. by finding other corresponding elements of the different groups of elements in the seeds. Consider the following tuple seeds <*Indonesia*, *Jakarta*, *Indonesian Rupiah*>, <*Singapore*, *Singapore*, *Singapore Dollar*>, <*France*, *Paris*, *Euro*>. Looking at the first, second, and third element of each seed, one can assume that these group of elements belongs to the semantic class "country", "capital city", and "currency name" respectively. However, these class labels are not defined a priori but can be inferred based on the examples of the elements provided. Once the correct label of each group of elements is identified, we can then expand the set of examples using the other members of the class. For example, "Malaysia", "United Kingdom", "Germany" are all members of the semantic class "country" and can, therefore, be used to expand the first group of elements. This type of task of automatically finding the label for a set of examples is closely related to topic labelling.

A topic modelling approach learns groups of words that best describe latent topics underlying a collection of documents. The process of finding these groups of words constitutes of learning the topic-word distributions as well as the mixture of topics in the documents. Upon learning the set of words describing each topic, one then has to figure out the right label for that topic. While feasible, manual labelling requires hard work and may be fairly time-consuming in some scenarios. The given label can be quite subjective with respect to the knowledge possessed by the person performing the task. For the following set of words {*charles, prince, king,*

*diana, royal, queen, family, parker, british, bowles*}, a person who is knowledgeable in world affairs would group the words as "prince charles", "diana", "royal queen", "parker bowles", "british" and then suggest "British royal family" as the label. On the other hand, "royal family" might be chosen by many as the relevant label by only considering "king", "prince", "queen", and "family" and disregarding the rest of the words. In this scenario, we see that a different person has a different opinion on the label. This motivates the task of topic labelling.

Topic labelling deals with the task of automatically finding labels for the topic models. Generally, the task mainly focuses on two subtasks, candidate labels generation, and ranking. The former focuses on retrieving or generating a set of candidate labels, while the latter applies a ranking mechanism on the candidate labels to select the most relevant label for the topic. Many methods have been proposed to tackle the task including those that leverage knowledge bases[111, 128, 105] and use data mining or machine learning algorithms [107, 110, 115]. In [105], the authors generate a set of primary and secondary candidate labels from Wikipedia, while the ranking process is carried out on features of the candidate labels using a support vector regression model. Bhatia et al. [107] also leverage Wikipedia as a source for the candidate labels, but they use documents and words embedding to represent the labels in the ranking process.

In this research, we propose to leverage topic labelling to automatically label the set of elements in the given tuple seeds. We treat the semantic class as the topic while the elements of the class as the set of words describing the corresponding topic. We transform the problem of proposing the label of a topic into truth finding (truth discovery) domain where we find the label for each of the top-$N$ words in the topic. The label for a topic is then selected from the labels of the corresponding top-$N$ words. The rest of the chapter is structured as follows. We begin with the problem definition of the task we are addressing in Section 6.2. We then detail our proposed method of generating and ranking candidate labels in Section 6.3. Section 6.4 explains the experiments we conducted to evaluate the performance of our proposed method, and Section 6.5 concludes the paper.

## 6.2 Problem Definition

We properly define the task we are addressing in this research as follows. Let $S$ and $L$ be the set of all semantic classes and a fixed finite set of class labels respectively. An element $e$ of a semantic class $s_i \in S$ is defined as $e \in s_i$ and $e \notin S \setminus s_i$. We further define a tuple $T$ as a set of $n$ elements ($n > 1$), where each of the $n$ elements belongs to a different semantic class of $S$. Topic labelling is then deals with the task of mapping an attribute label $l \in L$ to the semantic class $s_i$ given some examples of its members $(e_1, e_2, ..., e_i)$. Given a set of tuples $T_1, T_2, ..., T_j$ then topic labelling has to find a set of labels $\{l_1, l_2, ..., l_n\}$ from the set of examples members $\{e_{11}, e_{12}, ..., e_{1j}\}$, $\{e_{21}, e_{22}, ..., e_{1j}\}$, ..., $\{e_{11}, e_{12}, ..., e_{ij}\}$. Given the set of tuple seeds as follows <*Indonesia*, *Jakarta*, *Indonesian Rupiah*>, <*Singapore*, *Singapore*, *Singapore Dollar*>, <*France*, *Paris*, *Euro*>, we expect topic labelling to return

"country", "capital city", and "currency name" for the set of {Indonesia, Singapore, France}, {Jakarta, Singapore, Paris}, and {Indonesian Rupiah, Singapore Dollar, Euro} respectively.

**Algorithm 13:** FindCandidateLabels

**Input :** A set of words $W = \{w_1, w_2, ..., w_k\}$
**Output:** A set of candidate labels $L$

```
1 L = ∅;
2 foreach w_i ∈ W do
3     D = {FindDocuments(w_i)};
4     foreach d ∈ D do
5         L = L ∪ {ExtractCategories(d)};
6     end
7 end
8 return L;
```

## 6.3 Proposed Approach

In this research, we propose to automatically label topics generated via topic models by analysing a heterogeneous graph of relating the different objects and concepts characterising the problem: topics, words, Wikipedia articles, ancillary semantic information, and candidate labels. The graph is processed using truth discovery algorithms to produce a ranked list of candidate labels for each topic. We first describe how we generate candidate labels and then detail the ranking process.

### 6.3.1 Generating Candidate Labels

Our method of generating candidate labels is inspired by Lau et al. [105]. We leverage a knowledge base, in our case, it is Wikipedia[1], as the source for the candidate labels. Each article in Wikipedia has a list of categories associated with it. The categories are placed at the bottom of the page near the end of every article. We use these categories as our candidate labels. We perform the following two methods of extracting candidate labels for the topic model. To detail each method, we consider the following set of words {*charles, prince, king, diana, royal, queen, family, parker, british, bowles*} which are sorted in descending order based on the weight given by LDA.

Our first method of generating candidate labels is detailed in Algorithm 13. The algorithm accepts as input a set of $k$ words $W$. We use each of the words $w_i$ in $W$ as the search query to Wikipedia (line 2-3). From each of the search result return by Wikipedia, we consider only the top ten articles. We extract categories from these articles as the set of candidate labels for the topic (line 5). Since each topic given

[1] http://en.wikipedia.org

by LDA is described by a set of ten words, thus we have ten runs of the previously described method for every topic. The algorithm returns the set of candidate labels $L$ (line 8).

Lau et al. [105] stated that a good label for a topic is usually better expressed using a combination of words. Intuitively, using a combination of words to query Wikipedia may lead to more relevant articles. These relevant articles obviously have more relevant categories. This is the motivation behind our next method which is presented in Algorithm 14. The algorithm accepts a set of $k$ words $W$ and an integer $n$. $n$ controls the *n-grams* we want to create and is differ of $n$ in $n$-element tuples. We build *n-grams* from the set of words (line 2). Using the running example set of words and $n$=2, the bigrams constructed are {*charles prince*}, {*prince king*}, ..., {*british bowles*}, etc. As in the previous method, we then search Wikipedia using each of the bigrams and retrieve the categories from the articles in the search result as the candidate labels (line 3). We experiment with $n$=3, 4 and 5 to investigate the most effective *n-grams* to use as the query. We also investigate which of the two methods is better in terms of accuracy in proposing the most relevant label. We present the details of the result of the experiments in Section 6.4.2.

**Algorithm 14:** FindCandidateLabelsNGrams

**Input :** A set of words $W = \{w_1, w_2, ..., w_k\}$, an integer $n$
**Output:** A set of candidate labels $L$

1 $L = \emptyset$;
2 $W = CreateNGrams(W, n)$;
3 $L = \{FindCandidateLabels(W)\}$;
4 **return** $L$;

### 6.3.2 Ranking the Labels

After generating a set of candidate labels, the next step is to rank the labels and select the most relevant label for each topic. To achieve the task, we first construct a graph of entities with topics, words, Wikipedia articles, and candidate labels as the nodes while relationships among them as the edges. We then generate facts from the graph of entities as input to the truth discovery algorithm. Finally, we select the label for each topic based on the result of the truth discovery algorithm. We detail each of the steps in the process next.

#### Building Graph of Entities

Algorithm 15 details the process of building the graph of entities. We start the graph with the $K$ topics which we denote in the graph as Topic-*1*, Topic-*2*, ..., Topic-$K$ (line 2). Each of the topics has a set of words, thus we add all of the words to the graph as nodes (line 4). We relate each topic to its set of words by drawing edges between them (line 5-7). As detailed in Algorithm 13 and Algorithm 14 in Section 6.3.1, we propose two methods of retrieving candidate labels, i.e. using each

**Algorithm 15:** BuildGraph

**Input :** A set of topics $T$, where each topic $t_i \in T$ is a set of words $W_i$
**Output:** A graph of entities $G = (V, E)$

```
1  V = ∅; E = ∅;
2  V = V ∪ {T};
3  foreach t_i ∈ T do
4      V = V ∪ {W_i};
5      foreach w ∈ W_i do
6          E = E ∪ {< t_i, CONSISTS_OF, w >};
7      end
8      if useNGrams then
9          W'_i = CreateNGrams(W_i, n);
10         V = V ∪ {W'_i};
11         foreach w ∈ Desc do
12             Desc = FindDescendant(w, W'_i);
13             foreach w' ∈ W'_i do
14                 E = E ∪ {< w, GENERATE, w' >};
15             end
16         end
17         W_i = W'_i;
18     end
19     foreach w ∈ W_i do
20         D = {FindDocuments(w)};
21         V = V ∪ {D};
22         foreach d ∈ D do
23             E = E ∪ {< w, RETRIEVE, d >};
24             L = {ExtractCategories(d)};
25             V = V ∪ {L};
26             foreach l ∈ L do
27                 E = E ∪ {< d, EXTRACT, l >};
28             end
29         end
30     end
31 end
32 return G;
```

Table 6.1: Excerpt of input for truth discovery algorithms

| **Word** | **Attribute** | **Candidate Label** | **Wikipedia Articles** |
|---|---|---|---|
| serum | Label | serology | clinical_pathology |
| serum | Label | blood plasma | hematology |
| serum | Label | serology | epidemiology |

word or *n-grams* to query Wikipedia. In the latter approach, we first generate the *n-grams* from the set of words $W_i$, add the *n-grams* to the graph, and link each word that is used to compose the *n-grams* (line 8-16). We then replace $W_i$ with the set of *n-grams* (line 17). From each element (word/bigram) of $W_i$, we retrieve a set of Wikipedia articles (line 20). Thus, we link each word/bigram and the resulting Wikipedia articles in the graph (line 23). We then extract categories on each article as the candidate labels, add them to the graph, and create links between the articles and the candidate labels (line 24-28).

**Algorithm 16:** GenerateFacts

**Input :** A graph of entities $G = \{V, E\}$
**Output:** A set of facts tuples $B$

```
1  B = ∅;
2  W = {retrieve "word" node from G};
3  foreach w ∈ W do
4      D = {retrieve child nodes of w and the type of node="document"};
5      foreach d ∈ D do
6          L = {retrieve child nodes of d and the type of node="label"};
7          foreach l ∈ L do
8              b =< w, "label", l, d >;
9              B = B ∪ {b};
10         end
11     end
12 end
13 return B;
```

### Generating Facts

Truth discovery algorithm takes as input a collection of facts proposed by some sources regarding properties values of some objects. The goal of the truth discovery algorithm is to determine the true value of the properties as well as the level of trustworthiness of each source. In accordance with this definition, in this research, we define objects, sources, and facts as words describing the topics, documents (Wikipedia articles), and candidate labels extracted from the articles respectively. We generate the input to the truth discovery algorithms from the graph of entities constructed in our candidate labels generation process using Algorithm 16. We start by retrieving a set of nodes $W$ with the type of "word" (line 2). For each word $w$ in

Table 6.2: Indexes for labels extracted from the same article

| **Word** | **Attribute** | **Candidate Label** | **Wikipedia Articles** |
|---|---|---|---|
| serum | Label | blood sugar level | blood_test_1 |
| serum | Label | blood plasma | blood_test_2 |
| serum | Label | serology | blood_test_3 |

$W$, we retrieve its child nodes $D$ where the nodes' type is "document" (line 4). In the constructed graph, a "word" type node can be directly linked to a "documents" or via the *n-grams*. In the latter case, we first find the "*n-grams*" nodes which are derived from the "word" node and then collect the "document" nodes to the *n-grams*. Truth discovery algorithms do not acknowledge the relationships in between the objects and sources (words and documents), thus we disregard the *n-grams* when generating the facts. For each of the document, we then retrieve the labels and generate the facts as tuples consisting of the word, the label and the document (line 5-11). An example of the generated input for truth discovery algorithms is shown in Table 6.1.

From Table 6.1, we can see that there are three sources where each of the sources proposes a label for the word "serum". An agreement-based algorithm such as MAJORITY VOTING would choose "serology" as the best label for the word based on the facts given. The decision is based entirely on the number of sources which support the fact and disregard the trustworthiness level of the sources. However, this can be a problem since sources can propagate false information among them deliberately or unintentionally. Advance truth discovery approaches, on the other hand, not only consider the number of votes received by a fact but also the trustworthiness of the sources. For the same facts as in Table 6.1, an advance truth discovery algorithm could choose "blood plasma" as the best label depending on the level of trustworthiness of each of the sources. The confidence of the facts and the trustworthiness of the sources are calculated iteratively. As an initial setup, all sources are considered to have the same level of trustworthiness. The trustworthiness level of a source is updated based on the number of facts contributed by the source which are considered as true on each iteration. After completion, the truth discovery algorithm returns the confidence level of each fact and the level of trustworthiness of each source.

As described earlier in Section 6.3.1, we extract categories from Wikipedia article as candidate labels. Wikipedia can assign more than one category to an article. Thus in our graph of entities, a Wikipedia article can be connected to more than one label. In the truth discovery algorithm setting, a source can only contribute one fact of an object's property. In our research, if we apply the setting strictly, this means that a Wikipedia article can only contribute a label for a given word. This could potentially remove good candidate labels as we have to choose only a label from the article. To accommodate this with the truth discovery algorithms, we add indexes at the end of the article name for labels extracted from the same Wikipedia article as shown in Table 6.2. We generate the facts for input to the truth discovery algorithms using Algorithm 16.

**Selecting the Topic Label**

A truth discovery algorithm labels facts which are considered as the true values of the properties of the objects as well as its confidences. In this research, the truth discovery algorithm helps us in determining which candidate label can best describe a word of a topic. To select the best label for a word, we first rank the candidate labels based on the confidence given by the truth discovery algorithms and then select only the true label. Since a topic consists of a set of words and each word is associated with a label, at the end of the process, we have a ranked list of candidate labels for a topic. To label the topic, we simply take the top-$l$ ($l$=1) from the list. In one of the experiments in Section 6.4, we vary the top-$l$ labels which we consider as the relevant labels. We conduct the experiments using several truth discovery algorithms including a simple majority voting based on PageRank, Cosine, Truth Finder [81], 3-Estimates [92], Depen [89], and LTM [83]. We compare the performance of the algorithms in section 6.4.2.

## 6.4 Performance Evaluation

We detail here the experiments we conducted to evaluate the performance of our proposed approach. We describe the datasets and the metric we use for our experiment as well as the procedure. We then analyse and discuss the result.

### 6.4.1 Experiment Setting

We conducted our experiments for the purpose of evaluating the performance of our proposed approach on the following datasets. Next, we summarise each dataset as follows.

- DT1 - Ohsumed corpus [129][2]; this dataset consists of the first 20,000 documents from 50,216 medical abstracts of the year 1991. The documents are classified into 23 Medical Subject (MeSH) categories of the cardiovascular group.
- DT2 - British Academic Written English [130][3]; this dataset contains almost 3,000 university-level students' written assignments with a proficient standard. The assignments are distributed evenly over four disciplinary areas including arts and humanities, physical sciences, life sciences, and social sciences.
- DT3 - Monster.com job listings [4]; the dataset consists of more than 20,000 job listings taken from Monster.com Website as a subset of a bigger dataset (more than 4.7 million job listings).

[2] http://disi.unitn.it/moschitti/corpora/ohsumed-all-docs.tar.gz

[3] https://www.coventry.ac.uk/research/research-directories/current-projects/2015/british-academic-written-english-corpus-bawe/

[4] https://www.kaggle.com/PromptCloudHQ/us-jobs-on-monstercom

Table 6.3: Candidate labels generation

| Dataset | Method | Topic | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|
| | | **T1** | **T2** | **T3** | **T4** | **T5** | **T6** | **T7** | **T8** | **T9** | **T10** |
| DT1 | Keywords | **0.27** | 0 | 0.13 | 0 | 0.44 | 0.13 | 0.22 | 0 | 0 | **0.06** |
| | *2*-grams | 0.23 | 0 | **0.23** | 0.11 | **0.50** | **0.20** | **0.28** | **0.22** | 0 | 0 |
| DT2 | Keywords | 0.27 | 0 | 0.10 | 0.14 | 0.17 | 0.19 | 0.37 | 0.25 | 0.07 | 0.17 |
| | *2*-grams | **0.43** | **0.29** | **0.17** | **0.31** | **0.26** | **0.39** | **0.43** | **0.42** | **0.40** | **0.54** |
| DT3 | Keywords | 0.21 | 0.44 | 0.13 | 0.21 | 0.38 | 0.28 | 0.17 | 0.22 | 0.36 | 0.27 |
| | *2*-grams | **0.33** | **0.50** | **0.57** | **0.54** | **0.46** | **0.67** | **0.33** | **0.47** | **0.56** | **0.40** |

$$Acc = \frac{\sum_{i=1}^{|R|} Label(i)}{\mid G \mid}; \tag{6.1}$$

We evaluate the performance of the truth discovery algorithms for the task of topic labelling in terms of accuracy. To be able to achieve this we need to have baselines. In this research, we manually build our baselines for each dataset. We first run LDA on each corpus and learn $K$ topics where each topic is represented by the top-10 words. For example, the following top-10 words are given by LDA for the 2nd topic: {*patients, levels, serum, renal, less, plasma, insulin, blood, normal, increased*}. For each topic learned by LDA, we select labels from the set of candidate labels which are relevant to the topic. For the previous set of top words, the relevant labels are "diabetes", "blood plasma", "kidney failure", "disorders of endocrine pancreas", and "insulin". We then compare the list of candidate labels generated by the truth discovery algorithms with the baselines to calculate the accuracy using Equation 6.1. *Label(i)* returns 1 if the $i$-th candidate label is found in our baseline, otherwise, it returns 0. $|G|$ and $|R|$ denotes the size of the baseline and the total number of candidate labels respectively.

### 6.4.2 Experiments

We start the experiment by investigating the candidate labels generation process. We apply the following approach for the three datasets we use in the performance evaluation where each dataset consists of ten topics. The candidate labels are generated by extracting the categories from each article returned by Wikipedia. We apply the two methods detailed in Section 6.3.1 to retrieve the Wikipedia articles. At first, we use each word as a separate query to Wikipedia. Next, we generate *n-grams* from the set of words. After each run of the search, we build our graph of entities. For each of the graphs generated from the previous step, we apply truth discovery algorithms including majority voting, Cosine, Truth Finder [81], 3-Estimates [92], Depen [89], and LTM [83]. We then measure the accuracy of each algorithm using Equation 6.1. We then calculate the average accuracy for each candidate label generation method for each topic on the datasets across different truth discovery algorithms. Table 6.3 shows the result of this experiment. The first column in Table 6.3 corresponds to the three datasets which we use in our experiments. For each dataset in column 1, we perform two methods of candidate labels generation including using each word

as the query to Wikipedia (Keywords) and building *2-grams* (*bigrams*) from the set of top-$N$ words. The columns T1, T2, ..., T10 refer to the topic number of each dataset.

Next, we investigate the impact of using different *n-grams* for $n$=2, 3, 4, 5 to query Wikipedia. Our intuition is that using more words as a query would give more specific Wikipedia articles. In the end, this would lead to more specific labels. The result of this experiment is shown in Figure 6.1. We evaluate the impact of using different *n-grams* on the following truth discovery algorithms majority voting (MV), cosine (CO), TruthFinder (TF), 3-Estimates (3E), Depen (DP) and LTM (LT). We show in Figure 6.1 the average accuracy ($y$-axis) of each algorithm ($x$-axis) across the topics in each dataset.

In this experiment, we are interested in investigating the best truth discovery algorithm for our task of topic labelling. To focus our examination, we conduct the experiments using the best candidate label generation method. The method selection is based on the previous experiment. Figure 6.2 shows the comparison result of the truth discovery algorithms in terms of average accuracy ($y$-axis). The truth discovery algorithms in comparison are shown as $x$-axis (MV=majority voting, CO=cosine, TF=TruthFinder, 3E=3-Estimates, DP=Depen, LT=LTM).

As our last experiment, we investigate the possibility of improving the performance of our proposed method by varying the selection of the top labels from the list of candidate labels for the topic. For this experiment, we only use the best truth discovery algorithm using the best method for generating the candidate labels both of which are based on the results of the previous experiments. The result of the experiment is shown in Figure 6.3. The $x$-axis of Figure 6.3 shows the number of top labels ($l$) we consider when calculating the average accuracy ($y$-axis) for each topic of the datasets. We show the average accuracy for each dataset across the ten topics.

### 6.4.3 Result and Discussion

From Table 6.3, we can see that generally using *2-grams* for querying Wikipedia is more effective than queries using each individual word. For T2, T4, T8, T9 on dataset DT1 as well as T2 on DT2, querying each word to Wikipedia yield a score of 0 on accuracy. We then take a closer look at each of these topics and datasets. We find that among the set of words in each topic of the datasets, there are some very general words such as *women*, *disease*, *patients*, *god*, *magnetic*, etc. When we query Wikipedia using one of the words, for example *magnetic*, we retrieve the candidate label "magnetism" from the article. This label itself is very general. The term *magnetic* is a member of the topic T8 of dataset DT1. The other set of words are {*patients, imaging, nerve, syndrome, brain, may, cerebral, spinal, clinical*}. We would not consider "magnetism" as a relevant label as it is clear that it can not describe the topic. The relevant labels in our baseline for the topic T8 of dataset DT1 are "magnetic resonance imaging", "medical imaging", and "neurology". The first and second label in the baseline can be retrieved by querying *magnetic imaging* and *patients imaging* to Wikipedia. This is where our proposed method of using *2-grams*

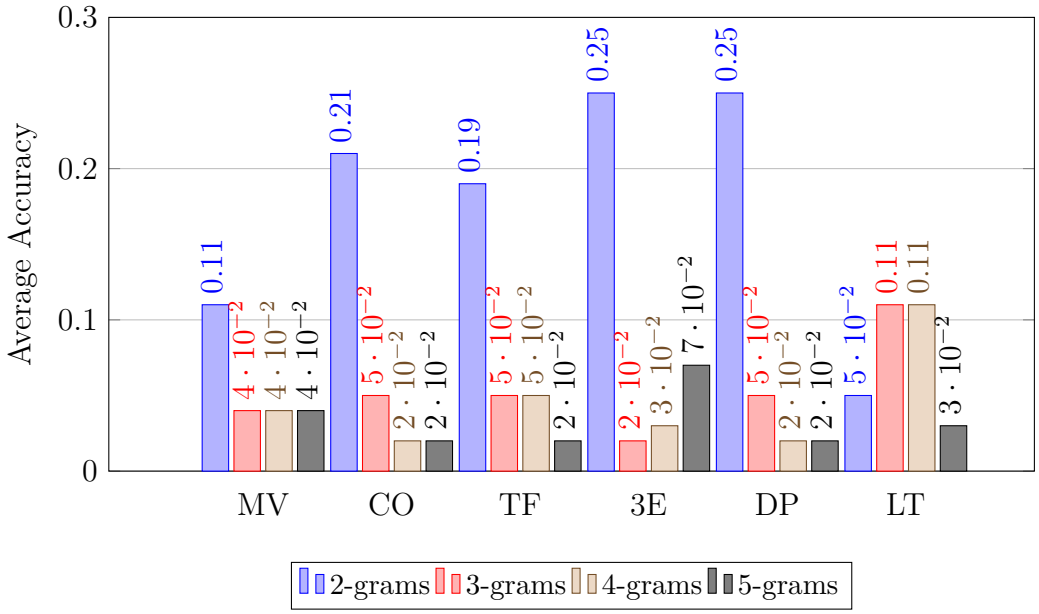


Figure 6.1: *n-grams* average accuracy comparison for dataset DT1

as the query to Wikipedia yield more relevant labels than querying each word. One special case where querying each word achieves better accuracy than *2-grams* is on topic T10 for dataset DT1. The set of words for this topic are the following {*induced, cells, activity, effect, animals, rats, mice, response, increased, effects*}. The labels in our baseline for topic T10 are "germ cells", "induced mice", and "invasive mammal species". Out of the three labels, "invasive mammal species" can be retrieved from Wikipedia articles using the word *rats* as the query. Using the *2-grams* generated from the set of words, we fail to retrieve any of the labels in our baseline. Both methods, however, fail to propose relevant labels on dataset DT1 for topic T2 and T9. Both topics are specific topics related to "diabetes" and "cognitive behavioral therapy" respectively. For topic T2, the closest labels proposed by both methods are "kidney cancer" or "insulin therapies". A human judgment would probably relate both labels to "diabetes" but since our method is completely unsupervised, it rejects the labels. In topic T9, our methods could only propose a more general labels such as "psychotherapy" and "medical terminology" for the set of topic words {*patients, treatment, therapy, group, treated, dose, study, months, less, received*} whereas the labels expected are "cognitive behavioral therapy", "group psychotherapy", "therapy group" or "psychology". Nevertheless, based on the examination of the experiment result we conclude that using *2-grams* as the query to Wikipedia is better than querying each word.

Figure 6.1 shows the average accuracy comparison of varying the $n$=2, 3, 4, 5 for the *n-grams* on dataset DT1. In general, forming *2-grams* from the set of top-$N$ words as query to Wikipedia gives better accuracy than using $n$=3, 4 or 5. *2-grams* achieves 2 to 10 times higher than *3-grams*, *4-grams* or *5-grams* for majority voting (MV), cosine (CO, TruthFinder (TF), 3-Estimates (3E) and Depen (DP). We look at the result of the experiments and find that querying Wikipedia using *3-grams*

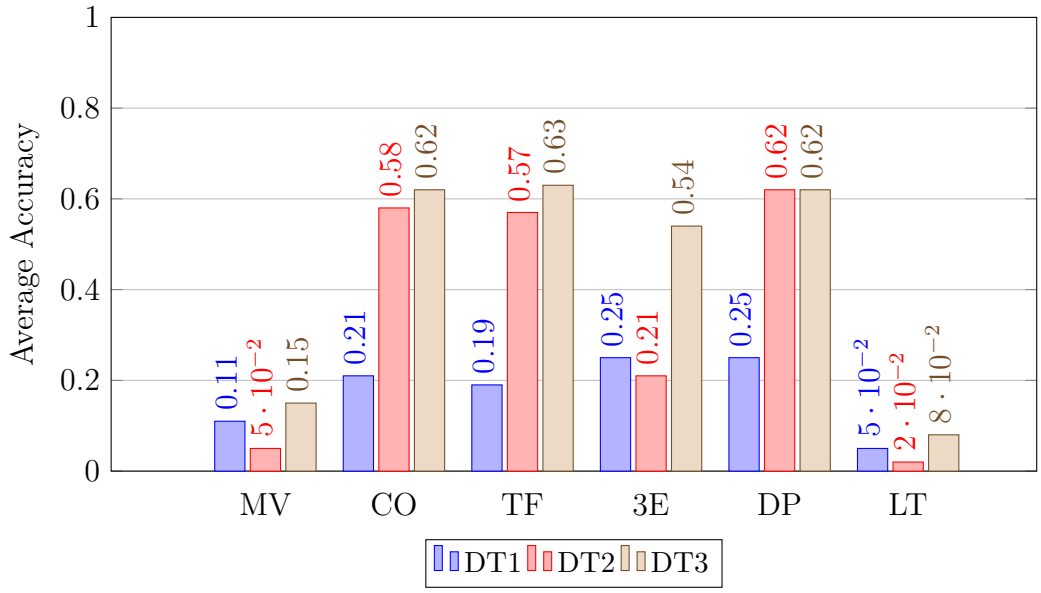


Figure 6.2: Average accuracy of truth discovery algorithms for *2-grams*

returns more articles. This also means that we extract more candidate labels. Unfortunately, most of the labels are not correct according to our baseline. For example, in T1 of dataset DT1, the *3-grams* "surgery treatment years" retrieves the following labels "cardiac surgery", "thoracic surgery" and "vascular surgery" among others, whereas "complications patient postoperative" retrieves "anesthesia", "vomiting" and "intensive care medicine". From the former *3-grams*, we are given the specific type of surgeries which probably need years of treatment after the operative procedure. The latter *3-grams* retrieves a symptom and ways to handle complications experienced by the patients. None of the retrieved labels are in the baseline. This contributes to the fact that *3-grams* achieves low average accuracy score. Moreover, as we have mentioned previously, most of the labels extracted by the *3-grams* are quite specific. However, a special case where *2-grams* achieves lower average accuracy than *3-grams* and *4-grams* can be seen on algorithm LT. The average accuracy achieves by LT on *3-grams* is 0.11. We investigate the result of the experiments to explain this case. In T2, T8 and T9 of dataset DT1, LT selects "coronary artery diseases", "magnetic resonance imaging" and "group psychotherapy" as the label respectively. All of the selected labels are in our baseline. In this case, the *3-grams* again retrieves specific labels but since they are all in the baseline the average accuracy of LT increases. This shows that a more sophisticated method of generating the baseline is necessary. The method must also balance the granularity level of the selected labels. Using too specific or too general labels could hurt the accuracy of the topic labelling model.

We have established from the previous experiment that the best method for candidate generation is by using *n-grams* of the set of words. We now focus our attention to find the best truth discovery algorithm for the task of topic labelling. For this examination, we only consider the results of the experiments in which we use

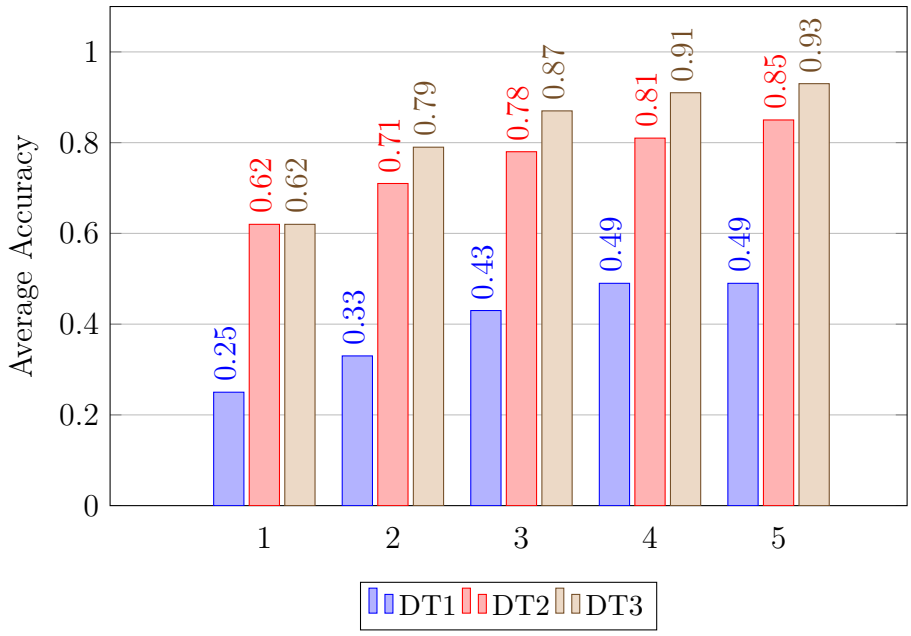


Figure 6.3: Average accuracy of Depen on varying the top-$l$ ($l$=1, 2, 3, 4, 5)

*2-grams* as the query to Wikipedia as shown in Figure 6.2. Looking at the accuracy of the truth discovery algorithms in Figure 6.2, we find that LTM and majority voting suffers in all the datasets. The average accuracies of both algorithms are in the range of 0.02-0.08 and 0.05-0.15. These values are considerably less than the other truth discovery algorithms in all three datasets. In dataset DT2, the accuracies of LTM and majority voting are as low as 0 for most of the topics. We take a look at these topics and find that majority voting only retrieves two to three labels for each topic. From the retrieved labels, a maximum of one label is true according to our baselines which means that the algorithm constantly voting for the wrong labels. The same case also applies for LTM where the approach mostly discovers wrong labels. In dataset DT1, 3-Estimates receives the same average accuracy as Depen, while TruthFinder ranks the highest in terms of accuracy in dataset DT3. However, both algorithms fall short of Depen in the other two datasets. Interesting cases where all truth discovery algorithms receive 0 in terms of accuracy can be found in dataset DT1 (topic T2, T9, and T10). The set of words describing the topic T2 of dataset DT1 are as follows {*patients, levels, serum, renal, less, plasma, insulin, blood, normal, increased*}. The set of labels in our baseline are "blood plasma", "diabetes", "disorders of endocrine pancreas", "insulin", "kidney failure". We take a look at each algorithm and find that most of the algorithms actually consider "insulin therapies" or "kidney anatomy" as the relevant labels. Both of these labels, from human judgment point of view, can be considered as having a relationship with the labels in our baseline. "insulin therapies" is a more specific topic than "insulin" whereas "kidney anatomy" is more general than "kidney failure". Both of these cases show hierarchical relationships. This opens up a new possibility of a future work in finding relationships among the extracted labels. From this set of experiments, we

conclude that the best truth discovery algorithm for the task of topic labelling is Depen.

The goal of the last experiment is to investigate the impact of varying the number of top-$l$ labels selected for each topic. Figure 6.3 shows the average accuracy of Depen across the ten topics in each dataset when we vary the top-$l$ from 1 to 5. The general conclusion that can be inferred from Figure 6.3 is that the correct labels tend to appear in the top-5 of the list. We take a closer look at datasets DT1 where Depen receives relatively low score than on the other datasets. When we set $l$=1, in every topic of dataset DT1 we find that there is at least one label which is referred to as the true value for some of the words on the topic. For example, in Topic 5 where the set of words are {*patients, tumor, disease, cancer, carcinoma, cases, tumors, cell, diagnosis, lesions*}, the words diseases, cancer, and cases all refer to the same label "infectious causes of cancer". We refer to this type of labels where it is referred to by many words in the topic as the dominant label. We search through all of the other topics and find the same case where there is at least one dominant label in it. These dominant labels are extracted from different Wikipedia articles using different *2*-grams queries. Our method then select "infectious causes of cancer" for Topic 5 label because it has the highest weight given by Depen. Indeed, this is not the correct label because our baselines are "carcinoma diagnosis", "oncology" and "medical terminology". When we vary the number of labels to be selected for each topic ($l$=2 or 3), our method retrieves "medical terminology" which is one of the baselines. This helps improve the accuracy score of Depen.

## 6.5 Conclusion

We propose to leverage truth discovery algorithms for the task of finding and ranking candidate labels of the topics in a topic model. We use Wikipedia as the source to extract the candidate labels by composing queries using the set of words of the topics. We apply different techniques of composing the query including generating the *n-grams* as the queries from the set of topic words. Several truth discovery algorithms are then used to retrieve the top-$l$ labels for each of the word in the topic. We evaluate the performance of the truth discovery algorithms in terms of accuracy on the retrieved labels. From the result of the empirical evaluation, we show that our approach is both efficient as well as practical. We are investigating the possibility of improving the approach by detecting relationships between candidate labels. Acquiring the knowledge of relationships between candidate labels of the topic domain a priori may have a positive impact on the performance of our approach.

# Chapter 7

# Set Labelling using Multi-label Classification

## 7.1 Introduction

In this chapter, we investigate a different approach of labelling the set of tuple seeds. Instead of considering each group of tuple seeds as words describing topic models as in Chapter 6, we treat them as elements constituting a set and then attempt to infer the appropriate label for the set using some features of the elements. To achieve the goal of inferring the label for the set, we leverage a multi-label classifier. We illustrate the task which we address in this research by the following simple real-life scenario. Consider Alice who is searching for cosmetics products. She browses Kiko's Website and is interested in two products that she finds on the Website: "Bright Lift Night" and "Bright Lift Mask". Unfortunately, the two products seem to be out-of-stock and nowhere to be shopped online. Alice decides to search for similar products on other brands' Websites. She visits Yves Rocher's Website. She could use the Website's search box or browse it by categories, but not only the names of the products but also their categories are different and Alice is not sure about what keywords to use. The problem which Alice faces in the scenario described previously is how to infer a *label* (category) for a *set* (the set of products) given some *examples* of its members (products). Of course, in the case of Alice, inferring the label requires more than just the name of the products. It is therefore imperative to find other *characteristics* (features) of the products to infer the label. We properly define the problem in the next section as well as propose a possible solution to the problem. In the case of Alice, we apply the proposed solution that allows Alice to discover the categories of products in which she is interested. The solution, is applied in the form of a search tool, finds the corresponding categories in Yves Rocher's Website to the categories of the products of Kiko's Website.

We organised this chapter as follows. In Section 7.2, we define the task of set labelling, propose a solution to the problem as well as implements the approach to two real-life scenarios in Section 7.3. We then evaluate the performance of a system implementing our approach in Section 7.4. We conclude the chapter in Section 7.5.

## 7.2 Problem Definition

We properly define the task of set labelling that we are addressing in this chapter. Set labelling deals with finding the most suitable label for a set, given some examples of its members. The example members are referred to as *seeds* while the set of seeds itself is the *query*. The task assumes that the set is homogeneous, i.e. each member of the set belong to the same class of objects or concepts. Set labelling tries to find an appropriate characterisation, a name or label, for the class. For this work, we consider sets of terms (words). For instance, given the seeds "Indonesia", "Singapore", and "Malaysia", set labelling can infer that these three objects belong to a set of countries, which can, therefore, be labelled as "country". One can also argue that the label should or could be "Asian country", "Southeast Asian country", "ASEAN members" as all of the seeds also belong to the aforementioned sets. Thus, set labelling returns a set of labels as the result instead of only a single relevant label. There is also inherent uncertainty regarding the sources that are used for the labelling. Therefore, it is imperative to return results ranked according to some degree of confidence or certainty.

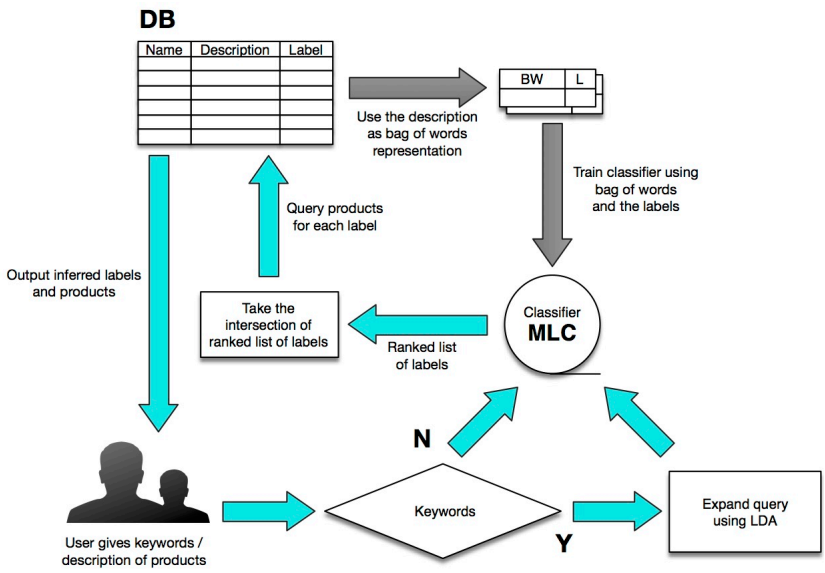


Figure 7.1: Flowchart of the system

## 7.3 Proposed Approach

We propose a solution to the task of set labelling by leveraging a multi-label classifier. In the traditional multi-class problem, the classifier must decide for a given input a single label which is considered as the most relevant. However, in the set labelling task, a given set of seeds can be assigned to more than one relevant label. In this setting, it is more appropriate to adopt the multi-label classification to our task as a multi-label classifier returns a set of relevant labels for the given input. The

classifier also ranks the labels according to their level of confidence which is also in line with the definition of the set labelling task.

To be able to evaluate our proposed approach of the set labelling task, we apply it to two real-life domains, i.e. cosmetic products and hotels. Alice's scenario is the motivation behind the first domain, while the similar setting can also happen in the second. Consider a person who is planning his holiday where he needs to arrange the accommodation. He has traveled to various places and enjoyed the hotel where he stayed in each place. Based on this past experiences, he already has the specific criteria for his favorite hotel. However, upon booking the accommodation, the Website does not provide the search tool that accommodating to all of his criteria. In this scenario, the problem that is faced by the person is how to find other hotels similar to the ones he has stayed in before based on his preferences. This problem can easily be tackled, that is, considering that we can retrieve the labels for all the previous hotels that the person has visited and then return other hotels under the same labels.

To propose a solution to both domains, we implement our approach into a search engine. This search engine can be embedded in various domains, not only the two we use as examples. The flowchart of the proposed solution is shown in Figure 7.1. In Figure 7.1, we show both the *training* and the *user interaction* process as a single flowchart.

### 7.3.1 Training Process

The process starts with training the multi-label classifier using labelled dataset. Each entry in the dataset consists of an object, the labels assigned and the features of the object. In this implementation, we choose the features as a set of words describing the corresponding object, i.e. descriptions of the cosmetic products and reviews given by users for the hotels. We then transform the features to a lower dimension as vectorized bag-of-words. To acquire the bag-of-words vectors, we use Scikit-learn [131] using the module CountVectorizer and TFIDFVectorizer [132]. We also conduct experiments using gensim's doc2vec [133] as a comparison. The result of the experiments using different methods of vectorization is presented in Section 7.4. This vectorized bag-of-words and the labels are the input to train the classifier.

### 7.3.2 User Interaction

Once the classifier has been trained, the user can interact with the system by giving either keywords or a description of the unknown objects as input. Since the multi-label classifier is trained using descriptions of the objects and its assigned labels, when interacting with the system, it is better to give also a form of description of the unknown objects as the queries. Nevertheless, if the user only provides keywords for each object then we need to be able to expand the given keywords and use it as the features of the unknown object instead. We achieve this by leveraging Latent Dirichlet Allocation (LDA) [1] into the expansion process.

LDA is a generative probabilistic model of a corpus. LDA assumes that a document is composed of random mixtures over latent topics, where each topic itself is a distribution over words. In the cosmetics products and hotels scenario, we consider the product labels and hotel categories as topics respectively. We use the descriptions of all products or reviews of hotels as input and ask LDA to learn $K$ topics (number of product labels or hotel categories). The output of LDA is $K$ topics where each topic is a distribution of words. We use the distribution of words in each topic in the query expansion process which is detailed next.

**Algorithm 17:** Query Expansion

**Input :** A set of words $I$, topic distribution $K$, integer $T$
**Output:** A set of words $O$

1 $O = I$;
2 **foreach** $w \in I$ **do**
3 | $z \leftarrow$ {select from $K$ where $w$ has the highest probability};
4 | $m \leftarrow$ {select top-$T$ words from $z$};
5 | $O = O \cup m$;
6 **end**
7 return $O$;

To expand the query, we use Algorithm 17 and give as input a set of keywords $O$, $K$ topic-words distribution, and a number $C$. For each word $o$ in the set of keywords $O$, we select topic $z$ among $K$ topics where $o$ has the highest probability (line 3). Word $o$ may appear in all of the $K$ topic-words distribution but has different probabilities. Our intuition is to select the topic where $o$ has the highest probability, i.e. $o$ most likely belongs to that topic. From topic $z$, we take the top-$T$ words to append $o$ (line 4). This means that the $T$ words are the best words to describe topic $z$. We do the same procedure for every words $o$ and return the union of all the words as the expansion result (line 7).

The same procedure is repeated as many times as required depending on the number of unknown objects. Each query to the multi-label classifier returns a ranked list of labels considered relevant by the classifier with refers to the given input. The proposed labels for the set of the unknown objects is then retrieved by intersecting all ranked list of labels from each run of the query. From the set of proposed labels, we retrieve other objects under those labels and present them to the user.

## 7.4 Performance Evaluation

### 7.4.1 Experiment Setting

Performance evaluation is done on a system implementing our proposed solution using two datasets from the cosmetics products and hotels domains. Table 7.1 shows a snippet of the Yves Rocher and Tripadvisor dataset. Each row in the dataset consists of an object name, a description, and labels assigned to it.

- DT1 - Yves Rocher. We collect our corpus of product names, descriptions, and labels assigned to the products from four Yves Rocher's Web pages (`http://yvesrocher.com.my`, `http://yvesrocherusa.com`, `http://yves-rocher.co.uk`, `http://www.yvesrocher.ca`). The total number of distinct products is 692. Each product can be assigned to at least a label (hence the *multi-label* aspect), where the number of distinct labels is 182. We also include comments given by users who have tried the product into the description. We also retrieve Yves Rocher's product reviews from `https://www.makeupalley.com`.

- DT2 - Tripadvisor. We collect the data of hotel reviews from Tripadvisor. There are 2,943 distinct hotels from this data with reviews. However, the review data does not have any information regarding the hotel labels. Thus, we manually collect labels for each hotel from Tripadvisor Web page. There are 52 unique hotel labels, such as 'Hotels with Free Wifi, 'Cheap Hotels', 'Hotels with Kitchenette', 'Apartment Hotels', etc.

In the multi-class (i.e., *single-label*) classification problems, the standard evaluation metric is *accuracy* where we measure the number of correct predictions from all of the predictions. In addition, *precision*, *recall*, *F-measure* and *ROC area*, complete the list of metrics for single multi-class classification problems [134]. However, none of this traditional metrics can be used to evaluate multi-label classification. The prediction for an object in multi-label problems is a set of labels instead of just a single label. The traditional metrics failed to capture the *partially correct* notion where only some of the predicted labels are correct. Godbole et al. in [117] extend the definitions for *accuracy*[1] *(A)*, *precision (P)*, *recall (R)*, and *F-measure ($F_1$)* as the following. For each object $i$, let $S_i$ and $\hat{S}_i$ denote the correct and the predicted set of labels respectively.

$$A = \frac{1}{N}\sum_{i=1}^{N}\frac{|S_i \cap \hat{S}_i|}{|S_i \cup \hat{S}_i|}; P = \frac{1}{N}\sum_{i=1}^{N}\frac{|S_i \cap \hat{S}_i|}{|\hat{S}_i|} \tag{7.1}$$

$$R = \frac{1}{N}\sum_{i=1}^{N}\frac{|S_i \cap \hat{S}_i|}{|S_i|}; F_1 = \frac{1}{N}\sum_{i=1}^{N}\frac{2|S_i \cap \hat{S}_i|}{|S_i| + |\hat{S}_i|} \tag{7.2}$$

We use the metrics in Equation 7.1 and Equation 7.2 to evaluate the performance of the multi-label classifier.

In multi-label classification problem, the confidence output for each label is usually of real-valued. Thus, we need a threshold to be able to separate the relevant and irrelevant labels. For a given vector $\hat{\mathbf{w}} \in \mathbb{R}^L$ of real-valued confidence outputs, a threshold function $f_t(\hat{\mathbf{w}})$ mapped a multi-label prediction $\hat{y}$ such that:

$$\hat{y}_j = \begin{cases} 1 & \text{if } \hat{w}_j \geq t \\ 0 & \text{if } \hat{w}_j < t \end{cases} \tag{7.3}$$

In [121], the author claims that using a single threshold throughout the evaluation is more efficient and just as effective as using a threshold vector. Thus, in our

[1] Also known as Jaccard index in this formulation

Table 7.1: Yves Rocher & Tripadvisor dataset

| OBJECT NAME | DESCRIPTION | LABELS |
|---|---|---|
| Anti Age Global Kit | This care struggle against the main characteristics of skin aging such as wrinkles, dark circles, sagging skin and dull skin. Plant native cells bring you their full potential self-regeneration. Set includes: Complete Anti-aging Day Care 15ml Complete Anti-aging Night Care 15ml | Anti Age Global, Skin Care |
| ... | ... | ... |
| ADAGIO ACCESS BORDEAUX RODESSE | The apartments were in a very good location for the city. Rooms clean and comfortable and staff friendly. There is a basic kitchen in the room which is a real bonus. Secure on site parking. We did not use any of the hotel facilities. The apartment was very good value for money. It had a single bed ... | Hotels with Free Wifi, Cheap Hotels, Hotels with Kitchenette, Apartment Hotels |
| ... | ... | ... |

performance evaluation, we calibrate the threshold $t$ using Equation 7.4 [121].

$$t = \arg\min_{t} \left\| \frac{1}{N} \sum_{i=1}^{N} \sum_{j=1}^{L} y_j^i - \left( \frac{1}{N} \sum_{i=1}^{N} \sum_{j=1}^{L} 1_{\hat{w}_j \geq t} \right) \right\| \tag{7.4}$$

In Equation 7.4, $N$ refers to the number of examples in the *test* set while $L$ is the number of labels. The first part of Equation 7.4 is basically just the average number of labels associated with each example [120]. The second part of the equation also calculates the same value, but in this case in the predicted labels with threshold $t$. We choose $t$ such that it minimises Equation 7.4. We find that $t = 0.2$ satisfies Equation 7.4 for both datasets.

We set the maximum and minimum document frequency (*max_df* and *min_df*) to 0.95 and 2 for both CountVectorizer and TFIDFVectorizer. If either *max_df* or *min_df* is of real value, the parameter represents a proportion of documents, otherwise, it is considered as absolute counts. Setting the two parameters ensures that words which appear too frequent or infrequent are going to be ignored when building the vocabulary.

To vectorize the description of an object using doc2vec, we set the hyperparameters as suggested in [135]. The initial ($\alpha$) and minimum learning rate ($\alpha_{min}$) are set to 0.025 and 0.001 respectively. The vector size for each description is 300 and the number of epoch we use is 20. We train our multi-label classifier using a chaining method proposed in [121]. By using the chaining method instead of BM, we can still capture the correlations that exist among labels without the expense of computational complexity. The base algorithm we use for each classifier in the chain is the random forest with 100 estimators.

Since the multi-label classifier is the core of our proposed solution, we focus on evaluating its performance on two datasets as summarised in Table 7.1. Initially, we compare the unnormalised vector (CountVectorizer) against the normalised (TFIDFVectorizer). The result is shown in Table 7.2. We use 100, 300, and 500 as the number of features extracted from each of the descriptions. Secondly, we compare

Table 7.2: Comparison of CountVectorizer (CV) and TFIDFVectorizer (TF)

| | | Accuracy | | | Precision | | | Recall | | | $F_1$ | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | | **100** | **300** | **500** | **100** | **300** | **500** | **100** | **300** | **500** | **100** | **300** | **500** |
| **Yves Rocher** | **CV** | 0.27 | 0.32 | 0.33 | 0.3 | 0.36 | 0.36 | 0.53 | 0.61 | 0.61 | 0.59 | 0.65 | 0.66 |
| | **TF** | 0.26 | 0.31 | 0.33 | 0.29 | 0.35 | 0.36 | 0.51 | 0.62 | 0.62 | 0.55 | 0.67 | 0.67 |
| **Tripadvisor** | **CV** | 0.35 | 0.35 | 0.35 | 0.38 | 0.38 | 0.38 | 0.7 | 0.71 | 0.71 | 0.74 | 0.75 | 0.75 |
| | **TF** | 0.35 | 0.35 | 0.35 | 0.38 | 0.38 | 0.38 | 0.71 | 0.71 | 0.71 | 0.75 | 0.75 | 0.75 |

Table 7.3: Comparison of TFIDFVectorizer (TF) and doc2vec (DC)

| | | **Accuracy** | | **Precision** | | **Recall** | | **$F_1$** | |
|---|---|---|---|---|---|---|---|---|---|
| | | **300** | **500** | **300** | **500** | **300** | **500** | **300** | **500** |
| **Yves Rocher** | **TF** | 0.31 | 0.33 | 0.35 | 0.36 | 0.62 | 0.62 | 0.67 | 0.67 |
| | **DC** | 0.04 | 0.03 | 0.05 | 0.04 | 0.11 | 0.09 | 0.12 | 0.1 |
| **Tripadvisor** | **TF** | 0.35 | 0.35 | 0.38 | 0.38 | 0.71 | 0.71 | 0.75 | 0.75 |
| | **DC** | 0.32 | 0.32 | 0.37 | 0.37 | 0.64 | 0.62 | 0.68 | 0.67 |

the best vectorization method from the first experiment with doc2vec. In both experiments, we split the dataset as train and test set. We fix the training and test set to 85% and 15% respectively. From the result of the second experiment, we choose the best vectorization method and implement it to the search engine.

To evaluate the performance of our proposed method of set labelling, we create $n$ unknown products from each category in the Yves Rocher dataset. To produce the description of each unknown product, we randomly select two products within the same category and concatenate the product descriptions. Next, we ask the classifier to assign the labels for each of these unknown products. We then take the intersection of the labels and compare the result with the original labels assigned by Yves Rocher to calculate the accuracy. Table 7.4 shows the result of the experiments.

### 7.4.2 Result and Discussion

From Table 7.2, we can see that for Yves Rocher dataset, CountVectorizer performs better for a small number of features. On the other hand, the performance of TFIDFVectorizer increases in corresponds to the number of features. The increase in performance for both CountVectorizer and TFIDFVectorizer is noticeable when we change the number of features from 100 to 300. However, a higher number of features than 300 does not improve the performance. For Tripadvisor data, on the other hand, varying the vectorization method or the number of features does not affect the performance. We take a closer look at the predicted labels for this dataset. We find that the classifier almost always returns a particular label as the prediction. This label is 'Hotels'. This is something that is quite 'predictable' because $\frac{2}{3}$ of the dataset is labelled as 'Hotels'. We intend to find a better source of labels for the Tripadvisor dataset. From this result, we can also see that we achieve better recall when we use TFIDFVectorizer and set the number of features to 300 or 500. Thus, we choose TFIDFVectorizer to compare with doc2vec in the second experiment.

Table 7.3 shows the result of the second experiment. For Yves Rocher dataset, we can see that using doc2vec to vectorize the description yields low result in all of the

metrics. Increasing the number of features to 500 does not improve the performance either. On the contrary, doc2vec achieves considerably the same performance as TFIDFVectorizer for Tripadvisor dataset. This result is possibly affected by the large number of the Tripadvisor dataset, which is four times greater than the Yves Rocher dataset. Moreover, the number of reviews for each hotel in the Tripadvisor dataset is also likely to have a positive impact on the final result. This result agrees with the claim that doc2vec needs more words to learn and produce good vector representation of the data as indicated by Lau et al. [135].

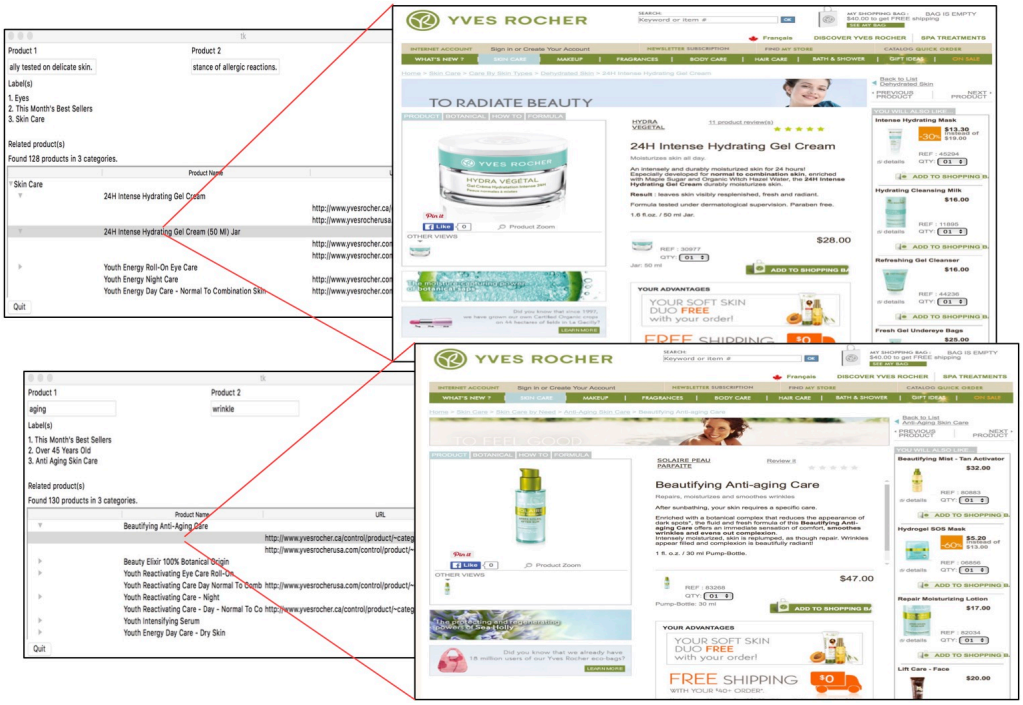


Figure 7.2: Screenshots of the search engine

Using the result of the experiments, we choose TFIDFVectorizer as the vectorization method for our multi-label classifier to use in the search engine for Yves Rocher and Tripadvisor. A few snapshots of the implemented search engine can be seen in Figure 7.2. In the first snapshot, we can see that the input to the system is descriptions. From each of the descriptions, our system infers a ranked list of labels through the use of the multi-label classifier. We take the intersection of these ranked lists of labels as the final set of labels. The final set of labels is then used by the system to query the database and retrieve all objects under each label in the set. We present the final set of labels and its objects to the user. In the second snapshot, the input to the system is only one keyword for each object. The system then expands the keyword using Algorithm 17 and feeds the result of the query expansion into the classifier. The same process as described earlier is then carried out to present the user with the relevant objects.

Table 7.4 shows the result of our experiments in finding the label of $n$ unknown products for the Yves Rocher dataset. The first column denotes the number of $n$

Table 7.4: Set Labelling Accuracy on Yves Rocher Dataset

| $n$ | Trial | | | | | | | | | | Average |
|---|---|---|---|---|---|---|---|---|---|---|---|
| | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | |
| 2 | 0.95 | 0.93 | 0.97 | 0.93 | 0.95 | 0.95 | 0.97 | 0.95 | 0.94 | 0.96 | 0.95 |
| 3 | 0.93 | 0.95 | 0.94 | 0.92 | 0.94 | 0.93 | 0.95 | 0.93 | 0.95 | 0.94 | 0.94 |
| 4 | 0.95 | 0.94 | 0.93 | 0.90 | 0.92 | 0.91 | 0.94 | 0.93 | 0.94 | 0.91 | 0.93 |

unknown products ($n$=1, 2, 3). In the second column, for each value of $n$, we conduct ten trials and calculate the accuracy of the set labelling approach. The last column shows the average accuracy of the experiments. From the experiment results, we find that our set labelling approach achieves an average accuracy of more than 90%.

## 7.5 Conclusion

We introduce the set labelling problem and propose a possible solution to the problem by using multi-label classification. We show that the multi-label classifier produces a relatively good result in accuracy, precision, recall, and F-measure for Yves Rocher and Tripadvisor dataset. The set labelling approach achieves an average accuracy of more than 90% in finding the labels of $n$ unknown products of Yves Rocher dataset. We apply the solution of this set labelling problem to a search engine for Yves Rocher and Tripadvisor. We are currently investigating various ways of improving the precision and recall of the multi-label classifier as well as evaluating the performance of our implemented solution in comparison to the traditional search engine provided on the site.

# Chapter 8

# Conclusions and Perspectives

## 8.1 Conclusive Summary

In this thesis, we have addressed several challenges in searching information by examples, particularly information in the form of $n$-ary relations (tuple). We presented a number of contributions that aid in extracting and ranking the extracted candidate tuples. We addressed the following important problems: the set of tuples expansion, the truthfulness of the extracted candidates, tuple reconstruction and semantically expanding the set of examples, which make a promising step towards realizing examples-oriented information extraction.

In the first part of the thesis, we generalised the set expansion problem to set of tuples expansion [35]. The proposed method takes as input a set of $n$-elements tuples as examples whereas traditional set expansion approaches mainly only consider atomic values. The generalisation process raises challenges in the crawling, wrapper generation, and candidate extraction phases. Our proposed method applies several strategies to retrieve relevant documents based on the given seeds including permutation of the seeds as keywords to the search engine. Locating occurrences of the seeds in the documents also consider the fact that each element of the seeds may appear arbitrarily. We implemented our proposed approach to a system (STEP) and demonstrated its effectiveness using eleven topics where each topic defines a set of tuples or table. We also compared our system to state-of-the-art set expansion systems (DIPRE [16] and Extended SEAL [30]) and showed that it achieves better performance in terms of precision and recall in several topics.

In [35], we adopted the ranking mechanism of SEAL [24] which is based on a graph of entities where the nodes represent entities while the edges between them denote the relationships. We then added a new entity, i.e. domain, into the graph and its relationships. The candidates are then ranked based on the weight assigned to the nodes after the walk has been completed. However, this ranking mechanism does not guarantee that the candidates with higher weight are true candidates as STEP may also extract false candidates. We proposed to leverage truth finding approach to find the true candidates among all candidates extracted by STEP [36]. We generated facts as input to the truth finding algorithms from the graph of entities. Our experiments using several state-of-the-art truth finding algorithms on seven

topics showed that Accu [89] is the best algorithm. Accu also gives significant improvement in terms of precision, recall, and F-measure compared to the graph-based ranking mechanism. In [37], we also compared the ranking of sources produced by the graph-based method and Accu. We found that leveraging truth finding algorithm, particularly Accu, can indeed improve the confidence in the produced ranking based on the Kendall's Tau scores.

We proposed the tuple reconstruction [38] task with the purposing of reconstructing $n$-elements tuples from binary ones. The motivation behind this task is that documents (e.g. Web pages) that contain information in the form of $n$-elements tuples are rarely to be found, while on the contrary binary tuples are very common. We presented two methods to reconstruct the $n$-elements tuples. To evaluate the performance of the proposed approach, we embedded the tuple reconstruction into the process of building the graph of entities in STEP. Experiment results on three topics showed that the tuple reconstruction can propose new candidates which are not extracted by STEP. Indeed, this integration of tuple reconstruction into STEP has increased its recall. However, this improvement may also impact the precision negatively.

In the second part of the thesis, we focus our attention on expanding the set of examples semantically. A set of $n$-elements tuples can be considered as $n$ groups of words describing $n$ topics. Having the knowledge of the label of each topic would make expanding the set of examples semantically be possible. As the first work, we leveraged topic labelling to give relevant labels for the topics. We transformed the problem of proposing the label of a topic into truth finding domain [40]. We used the words as the query to a knowledge base and retrieved categories of the returned documents as the candidate labels. We then associated a label for each word in the topic by leveraging truth finding approach. The topic label is then selected as the label of the words in the topic with the highest confidence given by the truth finding approach.

Finally, we addressed the set labelling problem which deals with the task of finding an appropriate label for a set given some examples of its members. We considered elements of a group in the set of examples as members of a set. Thus, by inferring the label of the set from features of the elements, we could expand the examples semantically. We proposed a solution to the problem by first training a multi-label classifier using labelled dataset. Each entry in the dataset denotes an object, the labels assigned and features of the object. We used the description of the object as its feature. Once we have trained the multi-label classifier, for a set of unknown or unseen objects, the classifier then inferred the relevant labels for each of the objects. The labels for the set is then selected as the intersection of all the labels returned by the classifier for all unknown objects. We applied our solution of the set labeling problem into a search engine [39] and experimented with two real-world scenarios, i.e. cosmetic products and hotels. We showed how the search engine can infer the labels and then present the user with all objects under the corresponding labels.

All of the research presented in this thesis provide coherent work on extracting and ranking candidate tuples. The research work also helps in achieving examples-

oriented information extraction.

## 8.2 Perspectives

We propose in this section the most promising research directions identified from the study conducted during this thesis. We discuss the following main challenges on targeted crawling, expansion on sibling pages, multiple facts from a source setting in truth discovery, and harnessing the hierarchy of labels for topic or set labelling.

**Targeted crawling** One of the main challenges we face in the set of tuples expansion is to collect the relevant documents with refers to the given tuple seeds. We have experimented on a search engine by querying it using the concatenation of all the seeds and then changing the order of each word in the query. However, this approach does not have a significant impact on finding the relevant documents. Since a rich collection of documents containing the seeds is critical in order to generate wrappers, the method of finding the relevant documents is paramount.

**Expansion on sibling pages** Information in the form of $n$-ary relations frequently is presented as paginated Web pages (*sibling* pages). However, these paginated Web pages do not appear in the search result. Failure of identifying and retrieving such paginated Web pages may lead to the loss of potential candidates.

**Multiple facts from a source** In a single truth setting of truth finding problem, a source can only contribute to a single fact regarding an attribute of an object. Embedding truth finding into the ranking process of our set of tuples expansion approach forces us to adopt the single truth setting by considering facts from a single source as multiple facts from different sources. One possible direction for dealing with this challenge is by harnessing current research on truth finding which can accommodate multiple facts from a single source. This could potentially improve the confidence of candidates ranked by the truth finding.

**Hierarchy of labels** An ontology generally groups elements of a set into a hierarchy of labels. It is interesting to investigate for the set labelling or topic labelling problem if we could harness this hierarchy of labels to propose the relevant labels for the set given the set of seeds.

# Appendix A

# Ranking of Sources

Table A.1: Ranking of sources for DT2

| URL | PageRank | | Accu | | Precision | | Recall | | F-measure | |
|---|---|---|---|---|---|---|---|---|---|---|
| | Weight | Rank | Acc. | Rank | Score | Rank | Score | Rank | Score | Rank |
| http://www.upi.com/List-of-Nobel-Prize-in-Physics-winners/68891381229493/ | 0.00653 | 1 | 0.5526 | 4 | 0.0896 | 4 | 0.1214 | 4 | 0.1031 | 4 |
| https://simple.wikipedia.org/wiki/List_of_Nobel_Prize_winners_in_Physics | 0.00561 | 2 | 1.000 | 2 | 0.2385 | 3 | 0.2429 | 2 | 0.2407 | 3 |
| https://en.wikipedia.org/wiki/List_of_Nobel_laureates_in_Physics | 0.00561 | 3 | 1.000 | 1 | 0.2385 | 2 | 0.2429 | 1 | 0.2407 | 2 |
| https://www.nobelprize.org/nobel_prizes/physics/laureates/ | 0.005 | 4 | 0.7446 | 3 | 0.4893 | 1 | 0.2149 | 3 | 0.2987 | 1 |
| http://math.ucr.edu/home/baez/physics/Administrivia/nobel.html | 0.00425 | 5 | - | - | - | - | - | - | - | - |

Table A.2: Ranking of sources for DT3

| URL | PageRank | | Accu | | Precision | | Recall | | F-measure | |
|---|---|---|---|---|---|---|---|---|---|---|
| | Weight | Rank | Acc. | Rank | Score | Rank | Score | Rank | Score | Rank |
| http://www.safesurfingkids.com/chat_room_internet_acronyms.htm | 0.0005 | 1 | 0.5074 | 4 | 0.00325 | 5 | 0.0011 | 5 | 0.00165 | 5 |
| http://www.netlingo.com/acronyms.php | 0.00036 | 2 | 0.1875 | 6 | 0.4488 | 1 | 0.4167 | 1 | 0.4322 | 1 |
| http://www.webopedia.com/quick_ref/textmessageabbreviations.asp | 0.00035 | 3 | 0.4718 | 5 | 0.0163 | 4 | 0.01186 | 2 | 0.0137 | 2 |
| http://www.smart-words.org/abbreviations/text.html | 0.00027 | 4 | 0.7669 | 1 | 0.1951 | 2 | 0.0059 | 3 | 0.0115 | 3 |
| http://mistupid.com/internet/chattalk.htm | 0.00027 | 5 | 0.5713 | 3 | 0 | 6 | 0 | 6 | 0 | 6 |
| http://www.txtdrop.com/abbreviations.php | 0.00026 | 6 | 0.6884 | 2 | 0.045 | 3 | 0.00185 | 4 | 0.0035 | 4 |

Table A.3: Ranking of sources for DT4

| URL | PageRank | | Accu | | Precision | | Recall | | F-measure | |
|---|---|---|---|---|---|---|---|---|---|---|
| | Weight | Rank | Acc. | Rank | Score | Rank | Score | Rank | Score | Rank |
| http://www.worldclasslearning.com/general-knowledge/list-countries-capital-currencies-languages.html | 0.00127 | 1 | 0.4252 | 6 | 0.1246 | 1 | 0.1762 | 1 | 0.14601 | 1 |
| https://www.spotthelost.com/country-capital-currency.php | 0.00062 | 2 | 0.5533 | 4 | - | 4 | - | 4 | - | 4 |
| http://www.science.co.il/international/Currency-codes.php | 0.0006 | 3 | 0.8635 | 2 | - | 4 | - | 4 | - | 4 |
| http://www.currentaffairsandgk.com/list-of-countries-and-capitals-with-currency-and-official-languages/ | 0.0006 | 4 | 0.4774 | 5 | 0.0714 | 2 | 0.0983 | 3 | 0.0827 | 2 |
| http://1min.in/content/international/currency-codes | 0.0006 | 5 | 0.9049 | 1 | - | 4 | - | 4 | - | 4 |
| http://self.gutenberg.org/articles/list_of_countries_and_capitals_with_currency_and_language | 0.00052 | 6 | 0.1237 | 7 | 0.0627 | 3 | 0.1106 | 2 | 0.0801 | 3 |
| http://bankersadda.in/country-capital-currency-list/ | 0.00045 | 7 | 0.7088 | 3 | - | 4 | - | 4 | - | 4 |

Table A.4: Ranking of sources for DT5

| URL | PageRank | | Accu | | Precision | | Recall | | F-measure | |
|---|---|---|---|---|---|---|---|---|---|---|
| | Weight | Rank | Acc. | Rank | Score | Rank | Score | Rank | Score | Rank |
| http://english.ahram.org.eg/NewsContent/6/55/251949/Sports/World/List-of-Ballon-dOr-winners.aspx | 0.0042 | 1 | 0.9603 | 2 | 0.7627 | 2 | 0.7377 | 1 | 0.75 | 1 |
| https://sg.news.yahoo.com/list-ballon-dor-winners-191013973--sow.html | 0.00359 | 2 | 0.9749 | 1 | 0.5 | 5 | 0.0163 | 6 | 0.0317 | 6 |
| http://www.totalsportek.com/list/fifa-ballon-dor-winners-of-all-time/ | 0.00302 | 3 | 0.099 | 6 | 0.0806 | 6 | 0.0819 | 5 | 0.0813 | 5 |
| http://www.nairaland.com/3520320/list-previous-ballon-dor-winners | 0.00297 | 4 | 0.8039 | 4 | 0.8275 | 1 | 0.3934 | 3 | 0.533 | 3 |
| http://www.topendsports.com/sport/soccer/list-player-of-the-year-fifa.htm | 0.00287 | 5 | 0.6923 | 5 | 0.5384 | 4 | 0.1147 | 4 | 0.1891 | 4 |
| http://www.topendsports.com/sport/soccer/list-player-of-the-year-ballondor.htm | 0.00283 | 6 | 0.8136 | 3 | 0.6229 | 3 | 0.6229 | 2 | 0.6229 | 2 |

Table A.5: Ranking of sources for DT6

| URL | PageRank | | Accu | | Precision | | Recall | | F-measure | |
|---|---|---|---|---|---|---|---|---|---|---|
| | Weight | Rank | Acc. | Rank | Score | Rank | Score | Rank | Score | Rank |
| https://www.youtube.com/watch?v=qxzIeQ_y5lE | 0.00505 | 1 | 0.4715 | 3 | 0.3846 | 3 | 0.0769 | 3 | 0.1282 | 3 |
| http://www.betdsi.com/events/entertainment/miss-universe | 0.00505 | 2 | 0.8788 | 2 | 0.796875 | 1 | 0.78461 | 1 | 0.79069 | 1 |
| https://en.wikipedia.org/wiki/List_of_Miss_Universe_titleholders | 0.00475 | 3 | 0.9373 | 1 | 0.7924 | 2 | 0.6461 | 2 | 0.7118 | 2 |

Table A.6: Ranking of sources for DT7

| URL | PageRank | | Accu | | Precision | | Recall | | F-measure | |
|---|---|---|---|---|---|---|---|---|---|---|
| | Weight | Rank | Acc. | Rank | Score | Rank | Score | Rank | Score | Rank |
| http://www.widescreenings.com/who-beat-oscar-best-actor.html | 0.001675 | 1 | 0.00069 | 3 | 0 | 3 | 0 | 3 | 0 | 3 |
| http://www.1728.org/page8.htm | 0.00165 | 2 | 0.3889 | 2 | 0.1554 | 2 | 0.3448 | 2 | 0.2142 | 2 |
| http://www.nndb.com/honors/511/000032415/ | 0.00163 | 3 | 0.7804 | 1 | 1.0 | 1 | 1.0 | 1 | 1.0 | 1 |

# Appendix B

# Résumé en Français

Pour rechercher des informations sur le Web, il faut généralement créer une requête à partir d'un ensemble de mots-clés et l'envoyer à un moteur de recherche. Cette méthode traditionnelle nécessite que l'utilisateur connaisse relativement bien le domaine de l'information ciblée pour trouver les bons mots clés. Les résultats de la recherche, sous la forme de pages Web, sont classés en fonction de la pertinence de chaque page Web par rapport aux mots clés donnés. Pour le même ensemble de mots-clés, les pages Web renvoyées par le moteur de recherche seraient classées différemment en fonction de l'utilisateur. De plus, pour trouver des informations spécifiques telles que le nom d'un pays et sa capitale, l'utilisateur devrait parcourir tous les documents et en lire le contenu manuellement. Cela prend non seulement du temps, mais exige également beaucoup d'efforts. Nous abordons dans cette thèse une méthode alternative de recherche d'informations, c'est-à-dire en donnant des exemples des informations en question. Tout d'abord, nous essayons d'améliorer la précision de la recherche à l'aide d'exemples de systèmes en développant syntaxiquement les exemples donnés. Ensuite, nous utilisons le paradigme de découverte de vérité pour classer les résultats de la requête renvoyée. Enfin, nous étudions la possibilité d'étendre sémantiquement les exemples en étiquetant chaque groupe d'éléments des exemples.

## B.1 Ensemble d'expansion de n-uplets

La tâche d'extension des ensembles consiste à extraire d'un corpus des éléments d'une classe sémantique particulière caractérisée par quelques exemples de ses membres. Plus précisément, étant donné un ensemble d'exemples, également appelés semences (par exemple, noms) caractérisant une classe sémantique particulière (par exemple, les présidents américains) et une collection de documents (par exemple, des pages HTML), le problème de l'extension de l'ensemble est d'extraire davantage d'éléments de la classe sémantique particulière de la collection de documents. Par exemple, nous voudrons peut-être trouver les noms des présidents des États-Unis d'Amérique en donnant comme exemples les trois exemples Barrack Obama, George Bush et Bill Clinton. Le but ici est d'extraire les noms de tous les autres présidents américains du Web.

Les approches d'expansion d'ensemble utilisent un cadre en trois étapes. La première étape est l'exploration. Compte tenu de l'ensemble des graines, l'exploration recherche les documents pertinents contenant les graines. Les différentes méthodes de composition des requêtes à partir des sources peuvent être appliquées pour analyser le moteur de recherche et collecter les pages Web. La prochaine étape est la génération de wrapper et l'extraction de candidats. La première consiste à rechercher toutes les occurrences des graines dans les pages Web collectées. Il identifie ensuite des modèles (wrappers) à partir de contextes entourant les occurrences des semences. Ce dernier utilise ces wrappers pour rechercher et extraire d'autres candidats possibles des pages Web. La dernière étape est le classement. Les processus de génération d'enveloppe et d'extraction de candidats sont susceptibles d'extraire des données superflues et erronées en raison de l'imperfection des modèles inférés. Classer les candidats en fonction de leur vraisemblance est alors primordial pour présenter des résultats satisfaisants aux utilisateurs.

Les approches d'expansion d'ensembles existantes (DIPRE [16], SEAL [24]) traitent en grande partie des semences atomiques. Chaque graine dans l'ensemble d'exemples donnés par l'utilisateur est constituée d'un seul élément. Cependant, la plupart du temps, un utilisateur demande des informations sous la forme d'instances de relations, comme dans le modèle relationnel d'une base de données. Dans ce cas, une instance de relation ou un n-uplet est composé d'un ensemble de valeurs atomiques. Les approches d'expansion des ensembles susmentionnées doivent être généralisées pour résoudre ce problème. Nous proposons notre approche consistant à étendre le problème de l'extension des ensembles à celui des ensembles de n-uplets.

Nous proposons de concevoir, mettre en œuvre et ávaluer une extension de l'approche d'extension d'ensemble à l'extension d'ensemble de n-uplets (STEP). La généralisation de l'extension de l'ensemble soulève de nouveaux défis à chaque étape du cadre d'expansion de l'ensemble: récupération des documents pertinents, génération d'encapsuleurs pour extraire les n-uplets candidats et classement des n-uplets candidats. Les difficultés proviennent principalement du fait que les graines sont constituées de multiples éléments (graines composites) et que chacun des éléments peut apparaître de manière arbitraire sur une page Web. En outre, aucune structure cohérente ne sépare chaque élément de la graine sur la page Web. Nous abordons ce problème en adoptant un wrapper basé sur des expressions régulières dans lequel nous tirons parti d'une version légèrement modifiée de l'algorithme de modification de distance (Section 3.2.2). Pour classer le candidat, nous basons notre mécanisme sur SEAL [24] où nous ajoutons un nouveau nœud (domaine) et son ensemble de relations (Section 3.2.3).

### B.1.1 Robot d'aspiration de site Web

Considérer les semences composées plutôt que atomiques pose le problème de l'ordre des éléments dans la graine. La question n'est ni discutée par les auteurs de [16] ni par [24]. En effet, les moteurs de recherche tels que Google sont relativement robustes lorsqu'on leur attribue des mots-clés dans des ordres différents. Cette robustesse, cependant, souffre de longues listes de mots-clés, comme c'est le cas avec

des graines composées de nombreux éléments. Nous avons essayé et pris en compte les chaînes de recherche formées avec la permutation des éléments des graines. Bien que cela puisse améliorer les performances dans certains cas, nous n'envisageons pas de telles permutations en raison du nombre élevé de permutations possibles. Dans la phase de génération de l'emballage (Section 3.2.2), cependant, de telles permutations sont importantes pour déterminer l'occurrence des semences. Nous utilisons le moteur de recherche Google pour collecter des pages Web. La requête de recherche est la concaténation de tous les éléments de toutes les graines. Pour l'ensemble des graines <Indonésie, Jakarta, Rupiah Indonésienne>, <Singapour, Singapour, Dollar de Singapour>, la requête de saisie de Google est «Indonésie» + «Jakarta» + «Rupiah Indonésienne» + «Singapour» + «Singapour» + «Dollar de Singapour».

### B.1.2 Génération de wrappers et extraction de candidats

Nous voulons extraire les candidats des documents (pages Web) de notre corpus. Par souci de simplicité, dans ce travail sans perte de généralité, nous considérons que les pages Web sont des documents HTML. Afin de pouvoir extraire les candidats des pages Web, nous devons d'abord déduire le wrapper sous-jacent. Ce wrapper est généralement constitué de contextes (caractères) entourant les attributs des graines données et des n-uplets candidats à récupérer ultérieurement. DIPRE [16] impose une contrainte selon laquelle un wrapper ne peut être généré que s'il met toutes les occurrences des graines sur les pages. C'est une contrainte très stricte qui diminue considérablement le rappel (le pourcentage de résultats corrects parmi tous les résultats corrects possibles) du systm̀e. SEAL [24], en revanche, relâche la contrainte lors de la construction des wrappers. Un wrapper est généré s'il met entre parenthèses au moins une occurrence de chaque graine sur une page. En relâchant la contrainte, SEAL surpasse DIPRE, notamment lors du rappel. Sur cette base, nous adoptons l'approche de SEAL en matière d'induction par emballage. Cependant, SEAL ne fonctionne que sur des graines avec des valeurs atomiques ou des relations binaires. Pour les graines avec plus d'un élément, nous devons d'abord trouver les contextes entre chaque élément, déduire l'encapsuleur et utiliser l'encapsuleur inféré pour extraire les candidats.

Une fois que toutes les occurrences de chacune des graines du n-uplet sont trouvées, nous essayons ensuite d'inférer un motif de chacun des ensembles de contextes. Nous appliquons une comparaison caractr̀e-caractr̀e pour des paires de contextes gauche et droit et prenons comme résultat la chaîne commune la plus longue (LCWrap, RCWrap). Pour le contexte de droite, nous faisons la comparaison de gauche à droite, tandis que pour la partie de gauche, nous devons inverser le processus. En ce qui concerne les contextes intermédiaires, nous générons une expression régulière commune à partir de paires du contexte en utilisant un algorithme de modification de la distance légèrement modifié. Les résultats sont notés $\text{MCWrap}_1$, $\text{MCWrap}_2$, ..., $\text{MCWrap}_{n-1}$ pour chacun des contextes du milieu. L'ensemble de wrappers est ensuite défini comme les combinaisons de chacun des LCWrap, $\text{MCWrap}_1$, $\text{MCWrap}_2$, ..., $\text{MCWrap}_{n-1}$ et RCWrap. Nous utilisons le

wrapper généré pour extraire les candidats de la page Web. Le processus d'extraction est assez simple. Nous ajoutons un groupe de capture, noté (.*?) dans l'expression régulière, entre LCWrap, $\text{MCWrap}_1$, $\text{MCWrap}_2$, ..., $\text{MCWrap}_{n-1}$ et RCWrap. Nous utilisons ensuite un analyseur d'expression régulière pour rechercher ses occurrences dans la page Web et extraire les informations souhaitées du groupe de capture.

L'un des problèmes liés aux semences composites est que chaque élément de la graine peut apparaître de manière arbitraire sur une page Web. Considérez la graine <Indonésie, Jakarta, Rupiah Indonésienne>. L'expression régulière utilisée pour localiser les occurrences de la graine précédente serait: «(. {1, max_context_length}) Indonesia(.*)Jakarta(.*)Roupie indonésienne(.{1, max_context_length})». Cette expression régulière ne parviendrait pas à détecter l'occurrence de la graine si l'élément «Roupie indonésienne» apparaît au début de la page Web, suivi de «Indonésie» et «Jakarta», Si nous ne considérons pas un tel cas, nous ne trouverons aucune occurrence du n-uplet dans la page Web et nous ne pourrons générer aucun wrapper. Pour résoudre ce problème, nous envisageons également de localiser les occurrences des permutations des éléments de la graine. Tous les candidats extraits à l'aide de ces enveloppes permutées doivent être replacés dans l'ordre des graines données.

### B.1.3 Classement

Nous adoptons le mécanisme de classement introduit dans SEAL [24]. Les noeuds du graphique représentent des entités, à savoir des graines, une page Web, un wrapper, un candidat et un domaine. L'entité principale indique à quelle page Web appartient. Ce type d'entité n'est pas obligatoire dans le graphique utilisé par SEAL. Nous discutons différemment parce que cette relation est importante pour produire une liste classée des domaines pertinents pour les semences. Ceci est un sous-produit de notre mécanisme de classement dans lequel nous pouvons non seulement répondre à la requête précédente, mais également à d'autres, telles que les pages Web les plus pertinentes, etc. Par exemple, il existe une frontiére entre les semences et les pages Web, car les pages Web correspondantes sont extraites en utilisant les semences comme requête du moteur de recherche. Les entités et leurs relations utilisées dans notre graphique d'entités sont résumées dans le tableau 3.2.

Une fois le graphique généré, un processus itératif est exécuté pour effectuer une marche aléatoire [76] sur le graphique. Le poids attribué à chaque nœud, une fois que le processus itératif est terminé ou atteint une probabilité stationnaire, est ensuite utilisé pour produire une liste de classement des entités. Cependant, la solution exacte pour trouver la probabilité stationnaire d'une marche aléatoire dans un graphique consistant à calculer l'équation $A^* = (A - I)^{-1}$, où A et I sont respectivement la matrice de transition et la matrice d'identité. Le problème de la marche aléatoire est non seulement coûteux en calcul, mais ne fonctionne pas non plus dans les cas particuliers de graphe, c'est-à-dire un graphe dont les noeuds n'ont pas de bord sortant (nœuds pendants) et un graphe avec des composants déconnectés. Dans ce cas, tous les nœuds reçoivent 0 comme poids final. Ceci est contre-intuitif car seuls les nœuds qui n'ont pas de bord entrant doivent recevoir 0 comme poids final. Dans ce dernier cas, la notation des pages de classement d'un composant connecté

à un autre composant connecté devient ambiguë. Page et. Al. [77] apportent la solution à ce probléme en introduisant une constante positive p entre 0 et 1, qu'ils appellent le facteur d'amortissement. Ils définissent également la matrice PageRank (ou matrice Google) du graphique par l'équation $M = (1-p) \cdot A + p \cdot B$. A désigne la distribution initiale des nœuds tandis que B est la probabilité de «téléportation». La probabilité de téléportation est une matrice n-par-n avec chaque entrée égale à $\frac{1}{n}$. La matrice n M modélise la marche aléatoire comme suit. Si nous sommes à un nœud i, au mouvement suivant, la plupart du temps, nous suivons les bords sortants de i vers l'un de ses nœuds voisins. Un pourcentage du temps plus petit mais positif, nous quittons le nœud actuel et choisissons arbitrairement un nouveau nœud dans le graphe et le «téléportent» vers ce nœud. Il a également été prouvé dans [123] que l'algorithme PageRank était compétitif par rapport à d'autres méthodes de classement, telles que Random Walk with Restart [124], Bayesian Sets [125] et Wrapper Length [123]. Pour ces raisons, nous avons choisi d'appliquer Pagerank au graphique des entités. Le poids calculé par PageRank pour chaque nœud peut ensuite être utilisé pour classer les nœuds du même type. Les candidats présentés comme sortie du système sont classés en fonction de leur poids en tant que PageRank. Nous sommes également en mesure de produire une liste classée du domaine en fonction des semences. Cela peut être considéré comme le domaine le plus pertinent pour la requête d'un utilisateur particulier.

Table B.1: Comparaison des performances de DIPRE, Ext. SEAL, et STEP

| | **Algorithme** | **Sujet** | | | | | | **Moyenne** |
|---|---|---|---|---|---|---|---|---|
| | | **DT1** | **DT2** | **DT3** | **DT4** | **DT5** | **DT6** | |
| Précision | DIPRE | 0 | 0 | 0 | 0.945 | 0.256 | 0 | 0.2 |
| | Ext. SEAL | 0 | 0.6 | 0.46 | 0.93 | 0 | 0.6 | 0.43 |
| | STEP | 0.97 | 0.07 | 0.74 | 0.945 | 0.136 | 0.182 | **0.51** |
| Rappel | DIPRE | 0 | 0 | 0 | 0.845 | 0.005 | 0 | 0.14 |
| | Ext. SEAL | 0 | 0.24 | 0.125 | 0.83 | 0 | 0.88 | 0.345 |
| | STEP | 0.986 | 0.38 | 0.98 | 0.441 | 0.126 | 1.0 | **0.65** |

### B.1.4 Comparaison avec l'état de l'art

Nous comparons notre approche proposée de l'extension de jeu de n-uplets à deux systèmes d'extension de jeu à la pointe de la technologie: DIPRE [16] et Extended SEAL [30]. DIPRE et Extended SEAL ne peuvent extraire que des relations binaires. Par conséquent, pour une comparaison équitable, nous évaluons les performances de DIPRE, Extended SEAL (Ext. SEAL) et STEP en termes de précision et de rappel uniquement pour l'extraction de n-uplets binaires. Pour cette expérience, nous ne considérons que DT1-DT6 du tableau 3.3 en utilisant les germes donnés. Nous exécutons chacun des algorithmes en donnant les graines en entrée et en récupérant toutes les pages Web (sans exclure aucune page Web de référence). Toutes les approches en comparaison trouvent les occurrences des graines en plaçant chaque élément dans le même ordre que celui des éléments dans les graines. Ensuite, tous les algorithmes trouvent également les occurrences des germes où les éléments sont inversés. DIPRE n'utilisant aucun mécanisme de classement, nous comparons la liste

complète des candidats produite par chaque algorithme sans prendre en compte le classement de chaque candidat. Nous définissons tous les paramètres requis par chaque algorithme à ses valeurs par défaut.

Le tableau B.1 montre le résultat de la comparaison des trois approches. En termes de précision moyenne et de rappel, STEP surpasse DIPRE et Ext. JOINT. STEP obtient systématiquement une précision supérieure à celle de DIPRE sur les sujets DT1, DT2, DT3, DT4 et DT6. Parmi ces sujets, DIPRE reçoit le score de précision le plus faible, à savoir 0 pour DT1, DT2, DT3 et DT6. Comme cela a été souligné dans l'algorithme 1, DIPRE regroupe d'abord les occurrences des graines en fonction de leurs contextes intermédiaires. Il applique ensuite la correspondance exacte sur les contextes intermédiaires avant de générer le wrapper. De plus, il ne peut déduire que des pages Web qui utilisent des modèles semi-structurés (tableau HTML, liste ordonnée ou non ordonnée, par exemple) pour présenter les données. Sinon, l'objectif ne sera pas atteint. D'autre part, sur les six sujets utilisés dans cette expérience, STEP obtient des scores de précision plus élevés que Ext. SCEAU dans les sujets DT1, DT3, DT5 et DT6. En fait, dans les sujets DT1 et DT5, Ext. SEAL n'extrait aucun candidat, tandis que STEP parvient à extraire 448 et 1759 candidats dont 435 et 240 sont dans la réalité, respectivement. Ext. SEAL atteint une précision supérieure à STEP pour les thèmes DT2 et DT6. Nous examinons les détails et trouvons que Ext. SEAL extrait seulement 40 et 140 candidats, contre 542 et 884 candidats extraits par STEP. Dans DT2, le nombre de candidats qui sont conformes à la vérité au sol pour Ext. SEAL est égal à 24 alors que STEP 38. Cependant, STEP extrait un ordre de grandeur plus grand que les candidats SEAL, il obtient donc un score de précision inférieur. Nous parcourons également la liste des candidats extraits par STEP et constatons qu'elle contient un grand nombre de noms d'universités qui ne figurent pas dans la vérité au sol. STEP domine Ext. SEAL en termes de rappel. En effet, le seul cas où Ext. SEAL obtient un rappel supérieur à celui de STEP dans la rubrique DT4. Dans ce sujet, Ext. SEAL parvient à extraire 637 candidats. Ce nombre est presque le double du nombre de candidats extraits par STEP. En DT6, STEP reçoit un score de rappel de 1,0 alors que SEAL n'atteint que 0,88 alors que sa précision est bien inférieure. Cela prouve encore une fois le fait que STEP extrait davantage de candidats qui apparaissent dans la vérité au sol et qui sont pertinents pour les semences données. La raison en est que SEAL implémente un algorithme de correspondance exacte pour déduire les contextes intermédiaires communs de toutes les occurrences des semences. Ceci est une exigence très stricte car les pages Web sont générées à partir de modèles différents et qu'il est donc très peu probable qu'elles aient les mêmes contextes intermédiaires. En raison de cet échec, SEAL ne peut pas créer de wrappers et ne peut donc extraire aucun candidat. Notre approche proposée pour trouver des contextes intermédiaires communs repose sur l'expression régulière. Cela permet à STEP d'être plus flexible dans la construction du wrapper et dans l'extraction des candidats. à partir des résultats de la comparaison, nous concluons que notre approche peut extraire davantage de candidats pertinents pour les semences. Ceci, à son tour, contribue à la performance de notre approche en optimisant le rappel tout en offrant une précision comparable à celle de l'état de la technique.

## B.2 Vérité dans la série d'expansion de n-uplets

Auparavant, nous avions présenté le problème de l'extension des ensembles de n-uplets et proposé une approche permettant de résoudre ce problème en généralisant le problème de l'expansion de l'ensemble. Nous présentons et détaillons notre approche à chaque phase, c.-à-d. l'exploration, la génération d'enveloppes et l'extraction potentielle, ainsi que la méthode de classement. Nous étendons l'idée proposée à l'origine par Wang et al. dans [24] et a appliqué une marche aléatoire sur un graphique hétérogène de pages Web, de wrappers, de graines et de candidats afin de classer les candidats en fonction de leur pertinence pour les graines. Cependant, ce mécanisme de classement n'est pas en mesure de déterminer de manière fiable les vraies valeurs parmi les candidats extraits. Faire la distinction entre les valeurs vraies et fausses de certains faits de manière automatisée, généralement connue sous le nom de recherche de la vérité, est un problème complexe qui a été étudié en détail dans plusieurs contextes tels que l'intégration des données et la vérification des faits. Nous proposons d'intégrer un algorithme de recherche de la vérité dans notre processus d'expansion. L'objectif est de pouvoir présenter les vrais candidats à l'utilisateur et d'obtenir de meilleures performances en termes de précision, de rappel et de F-mesure.

### B.2.1 Définition du problème

Nous abordons ici le problème de la recherche des vraies valeurs parmi les candidats extraits par le système d'extension des ensembles de n-uplets. Avant de présenter notre approche consistant à utiliser l'algorithme de recherche de la vérité dans le processus d'expansion, nous commençons par la définition du problème.

Nous fixons l'ensemble des n-uplets candidats $\mathcal{T}$ extraits par STEP. Ces tuiles extraites représentent des exemples d'objets du monde réel caractérisés par certains attributs (ou propriétés). Nous nous limitons dans ce travail au cadre commun à une vérité où tout attribut de chaque objet n'a qu'une valeur correcte et de nombreuses valeurs possibles. Nous considérons des ensembles finis fixes d'étiquettes d'attribut $\mathcal{A}$ et de valeurs $\mathcal{V}$, ainsi qu'un ensemble fini d'identificateurs d'objet $\mathcal{I}$. Ensuite, nous avons formellement:

**Definition 1.** *Un n-uplet $t$ est un couple $(i, v)$ où $i$ est un identifiant d'objet donné dans $\mathcal{I}$ et $v$ une correspondance $v : \mathcal{A} \to \mathcal{V}$.*

L'algorithme d'expansion de jeu de n-uplets inspecte les documents en utilisant des wrappers pour extraire une collection de n-uplets, c'est-à-dire ses déclarations sur les valeurs de certains attributs d'objets: voir Example 1. Dans ce paramètre, nous décrivons une source de données modélisée par un document donné ( où la valeur a été extraite) et un wrapper (l'extracteur de la valeur). Au niveau des attributs, nous définissons en effet une source de données comme suit:

**Definition 2.** *Une source est une fonction partielle $S : \mathcal{I} \times \mathcal{A} \to \mathcal{V}$ a avec un domaine non vide. Une vérité de terrain (candidate) est une fonction totale $G : \mathcal{I} \times \mathcal{A} \to \mathcal{V}$.*

Étant donné une source de données, un n-uplet particulier $t$ est représenté par l'ensemble des couples d'identificateur d'objet et de valeur d'attribut qui partagent le même identificateur d'objet. C'est-à-dire que, étant donné une source $S$ et deux attributs distincts $a$, un $a' \in \mathcal{A}$, nous déclarons que $S(i_1, a)$ et $S(i_2, a')$ font référence à des déclarations concernant les attributs $a$ et $a'$ du même n-uplet $i_1 = i_2$ où $i_1$ et $i_2$ sont des identificateurs d'objet dans $\mathcal{I}$.

**Example 1.** *En réexaminant les n-uplets dans l'exemple 1 de la section 2.3, on peut observer qu'il s'agit d'objet du monde réel «Pays» avec leurs propriétés (attributs) «Capitale» et «Monnaie»: nous avons trois n-uplets candidats <Algérie, Alger, Dinar Algerian>, <Algérie, Alger, Leka Algérienne> et <Algérie, Alger, Dinar Algérienne> à partir de trois sources de données différentes Source_1, Source_2 et Source_3. Notez que dans l'exemple 1, pour des raisons de simplicité, l'identificateur d'objet, c'est-à-dire «Algérie», a été omis, tandis que les valeurs d'attribut fournies sont «Alger», «Dinar algérien» et «Leka algérienne», respectivement.*

L'objectif principal du problème de recherche de la vérité est de déterminer la vérité sur le terrain à partir des déclarations d'un certain nombre de sources. Les sources sont spécifiquement définies comme éventuellement incomplètes; en particulier, ils peuvent ne pas couvrir tous les attributs:

**Definition 3.** *La couverture d'attribut d'une source $S$ par rapport à $X \subseteq \mathcal{A}$ est la proportion d'attributs $a \in \mathcal{A}$ pour lesquels il existe au moins un objet $o \in \mathcal{O}$ tel que $S(o, a)$ est défini: $Cov(S, X) := \frac{|\{a \in X | \exists o \in \mathcal{O}\ S(o,a)\ defined\}|}{|X|}$.*

Deux sources $S$ et $S'$ 'se contredisent, c'est-à-dire en désaccord, chaque fois qu'il existe $i \in \mathcal{I}$, $a \in \mathcal{A}$ tel que $S(i, a)$ et $S'(i, a)$ soient tous deux définis et $S(i, a) \neq S'(i, a)$. Une source est correcte par rapport à la vérité sur le sol $G$ sur $i \in \mathcal{I}$, $a \in \mathcal{A}$ lorsque $S(i, a) = G(i, a)$ et fausse lorsque $S(i, a)$ est défini, mais $S(i, a) \neq G(i, a)$. Un processus de recherche de la vérité, tel que ceux examinés à la section 2.3, est formellement défini comme suit:

**Definition 4.** *Un algorithme de recherche de vérité $F$ est un algorithme qui prend en entrée un ensemble de sources $\mathcal{S}$ et renvoie une vérité au sol candidate $F(\mathcal{S})$, ainsi qu'une précision estimée $Accuracy_F(S)$ pour chaque source $S \in \mathcal{S}$ et un score de confiance $Confidence_F(S(i, a))$ pour chaque déclaration $S(i, a)$ faite par une source $S$ sur tout attribut $a$ de l'objet $i$.*

Un grand nombre de processus de recherche de la vérité sont des algorithmes d'itération à virgule fixe qui calculent une vérité au sol candidate en fonction de valeurs de précision de la source (ou de fiabilité) et de scores de confiance. Si un algorithme de recherche de vérité $F$ n'utilise pas spécifiquement les valeurs de précision de la source (c'est le cas, par exemple, du vote à la majorité), on peut définir respectivement la précision de la source et la confiance pour $F$ simplement:

$$Accuracy_F(S) := \frac{|\{\, i \in \mathcal{I}, a \in \mathcal{A} \mid S(i, a) = F(\mathcal{S})(o, a)\}|}{|\{\, i \in \mathcal{I}, a \in \mathcal{A} \mid S(i, a) \text{ défini}\}|} \tag{B.1}$$

Pour un identifiant d'objet fixe $i \in \mathcal{I}$ et un attribut $a \in \mathcal{A}$, on a:

$$Confidence_F(S(i,a)) := \frac{\sum_{S' \in \mathcal{S} | S'(i,a)=S(i,a)} Accuracy_F(S')}{|\{ S' \in \mathcal{S} \mid S'(i,a) = S(i,a)\}|} \tag{B.2}$$

**Example 2.** *En appliquant l'équation B.1 sur les n-uplets candidats de l'exemple 1, nous obtenons les valeurs de précision de 1, 1/2 et 1 pour Source_1, Source_2 et Source_3 respectivement. En effet les valeurs vraies sont choisies en votant ainsi les valeurs pour les attributs "Capitale" et "Monnaie" seront "Alger" et "Dinar algérien". Selon l'équation B.2, les scores de confiance des affirmations «Alger», «Algérienne» et «Leka Algérienne» seront fixés à 5/6, 1 et 1/2. De toute évidence, «Dinar algérien» provient de sources de premier plan, c'est-à-dire Source_1 et Source_3, est donc choisi à la place de «Dinar algérien».*

C'est ainsi que l'approche la plus simple de recherche de la vérité, c'est-à-dire le vote, détermine les vraies valeurs de certains faits. Dans notre contexte, les faits sont des valeurs d'attributs de certains objets du monde réel. Par conséquent, nous exploitons l'algorithme de recherche de la vérité dans le processus d'extension des n-uplets pour trouver les valeurs vraies de ces attributs et choisir les candidats appropriés. Nous présentons officiellement la suite d'un tel processus.

### B.2.2 Approche proposée

Comme indiqué en détail à la section 3.2.3, nous construisons le graphique d'entités et appliquons Pagerank au graphique avant de classer les candidats en fonction du poids attribué par PageRank. Dans ce travail, nous proposons d'incorporer l'algorithme de recherche de la vérité dans le processus d'expansion afin de déterminer les vrais candidats parmi les candidats extraits et de présenter une liste classée des candidats comme résultat final. Pour pouvoir atteindre cet objectif, nous devons associer le graphique des entités au paramètre de recherche de la vérité.

**Algorithme 18 :** Générer_des_faits

**Entrées :** Un graphique d'entités $G = \{V, E\}$
**Sorties :** Un ensemble de faits n-uplets $B$

```
1  B = ∅;
2  C = {récupérer tous les noeuds de G où le type de noeud == «candidats»};
3  pour chaque c ∈ C faire
4      p =Trouver_le_parent_source_de(c);
5      k = c[1];
6      pour i ← 2 à n faire
7          b =< k, Attr_{i-1}, c[i], p >;
8          B = B ∪ {b};
9      fin
10 fin
11 return B;
```

À partir de la définition B.2.1, un algorithme de recherche de vérité prend en entrée un ensemble d'instructions faites par des sources sur tout attribut d'un objet. Nous considérons ici les déclarations en tant que candidats du n-uplet et les sources en tant que pages Web où nous dérivons le wrapper utilisé pour extraire les candidats correspondants. L'objectif général d'intégrer l'algorithme de recherche de la vérité dans notre approche est d'améliorer les candidats présentés comme résultat final. Cependant, dans le problème de l'extension des n-uplets, le concept d'objet et ses attributs ne sont pas explicitement modélisés. Cependant, les éléments dans les semences et dans les n-uplets extraits entretiennent des relations. Compte tenu du n-uplet <Angola, Luanda, Kwanza>, on peut facilement affirmer que Luanda et Kwanza sont les attributs de l'Angola. Pour être cohérent avec le paramètre de recherche de la vérité, nous décomposons ensuite techniquement l'objet et ses attributs dans les n-uplets extraits par notre système. Un objet est représenté par le premier élément du n-uplet, désigné par la clé d'objet, tandis que les autres éléments restants servent d'attributs. Nous utilisons en outre cet élément clé pour décomposer un n-uplet en n-uplets binaires attribut-clé. Le n-uplet précédent est décomposé en <Angola, Luanda> et <Angola, Kwanza>. En général, si un n-uplet est composé de n éléments, alors, après décomposition, nous obtenons n-1 n-uplets d'attributs clés.

Pour expérimenter des algorithmes de recherche de vérité à la pointe de la technologie présentés dans la section 2.3, nous nous appuyons sur la plateforme DAFNA [1]. DAFNA fournit AllegatorTrack [126], une application Web configurant une implémentation de la plupart des algorithmes de recherche de vérité à la pointe de la technologie dans le même contexte. AllegatorTrack accepte en entrée un ensemble d'entrées dont les éléments principaux sont une clé d'objet, un attribut d'objet, une valeur d'attribut et une source de données. Ainsi, dans chacun des n-uplets d'attributs-clés, nous devons également inclure l'attribut de l'objet et la source. La source est la page Web à partir de laquelle l'encapsuleur utilisé pour extraire le candidat est dérivé. Comme l'attribut de l'objet n'est pas connu a priori, nous donnons simplement «Attr» suivi d'un index comme libellé.

Nous utilisons l'algorithme 18 pour générer les faits à partir du graphe d'entités. Nous récupérons d'abord tous les nœuds candidats du graphe (ligne 2). Ensuite, pour chaque nœud $c$, nous trouvons la page Web $p$ à partir de laquelle le candidat est extrait (ligne 4). Pour ce faire, nous suivons le périmètre du nœud candidat au wrapper, puis à la page Web. La clé d'objet $k$ est le premier élément du n-uplet candidat de n éléments, $c[1]$, tandis que les autres éléments $(c[2], ..., c[n])$ sont les valeurs d'attribut. Ainsi, pour chacune des valeurs d'attribut, nous formons le n-uplet sous la forme $<k, Attr_{n-1}, c[i], p>$ et l'ajoutons au résultat (ligne 6-9).

Table B.2: Extrait du jeu de données en entrée dans AllegatorTrack

| **Objet** | **Attribut** | **Valeur** | **La source** |
|---|---|---|---|
| Angola | *Attr_1* | Luanda | `http://1min.in/content/international/currency-codes_1` |
| Angola | *Attr_2* | Kwanza | `http://1min.in/content/international/currency-codes_1` |
| Angola | *Attr_2* | Kuanza | `http://1min.in/content/international/currency-codes_2` |

[1] http://da.qcri.org/dafna/#/dafna/home_sections/home.html

STEP génère des wrappers pour une page Web en combinant les contextes gauche, droit et moyen (voir section 3.2.2)). Cela donne la possibilité d'extraire les n-uplets identiques ou distincts d'une même page Web au moyen de wrappers différents. Il est à noter qu'une source donnée ne peut pas fournir plus d'une déclaration pour le même attribut d'objet dans notre paramètre de recherche de la v'erit'e. En conséquence, nous nous référons à la source d'un n-uplet candidat donné en indexant le nom de la page Web inspectée avec le numéro du wrapper utilisé. Le tableau B.2 montre un extrait d'un jeu de données en entrée dans AllegatorTrack dans lequel chaque ligne est composée respectivement de la clé d'objet, suivie de l'attribut de l'objet, puis de la valeur de l'attribut et enfin de la source.

### B.2.3 Évaluation des performances

Nous avons effectué notre évaluation de manière intensive sur sept sujets constitués d'un ensemble de n-uplets extraits par notre système STEP pour des domaines distincts; Nous avons détaillé dans la section 3.2 comment notre système retourne un ensemble de n-uplets dans un domaine donné, à partir de certaines graines. La figure B.1 compare la F-mesure moyenne des algorithmes de recherche de la vérité et de PageRank en fonction des scores F-mesure de chaque approche sur nos sept sujets. De toute évidence, les algorithmes de recherche de la vérité surpassent PageRank avec une amélioration allant de 10% à 26% en termes de précision et de rappel. Accu (AC) est le meilleur modèle de recherche de la vérité pour classer w.r.t. scores de F-mesure calculés. La seule exception est T4 où Accu reçoit le score minimal pour les trois métriques. Nous étudions davantage ce cas particulier en appliquant d'abord le vote à la majorité sur les candidats extraits, puis en calculant la précision, le rappel et la F-mesure du résultat. Nous observons que le score est très faible, ce qui indique que les candidats extraits de la plupart des sources ne sont pas corrects, ce qui engendre un processus de recherche de la vérité biaisé. Les candidats extraits contiennent également beaucoup de bruits, par exemple, deux sources extraites «Dollar de Singapour» et «Dollar» en tant que valeur d'un des attributs. Alors qu'une personne considérerait que ces deux valeurs font référence au même nom de devise pour le pays Singapour, il serait difficile pour un système d'inférer cela sans aucune interaction humaine. Néanmoins, Accu améliore la précision, le rappel et la F-mesure de notre approche de 0,54%, 0,29% et 0,51% respectivement. Sur cette base, nous concluons que l'utilisation d'un algorithme de recherche de la vérité peut effectivement améliorer les performances de notre approche.

## B.3 Reconstruction de n-uplets

Un ensemble de systèmes d'extension de n-uplets, tel que STEP [35], extrait des informations du World Wide Web sous la forme de n-uplets. Particulièrement, étant donné un ensemble d'exemples de n-uplets <Indonésie, Jakarta, Roupie indonésienne>, <Singapour, Singapour, Singapour, Dolor>, STEP renvoie une liste comprenant le nom d'un pays, sa capitale, ainsi que le nom de la devise. STEP

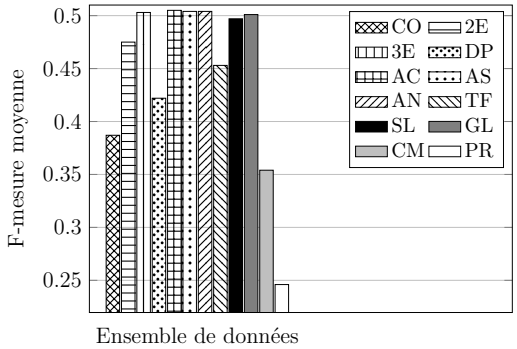


Figure B.1: Scores moyens de la F-mesure

applique un mécanisme de classement des votes basé sur la valeur Pagerank des n-uplets candidats.

PageRank [77] est principalement utilisé par Google pour mesurer l'importance ou l'importance des pages Web. Il s'agit d'une requête donnée par l'utilisateur. Chaque lien d'autres pages Web vers une page Web cible est traité comme un vote pour la page Web cible. Les pages Web sont ensuite classées en fonction du nombre de votes correspondant. Dans le cas particulier de STEP [35], un graphe est construit en définissant un ensemble d'entités (pages Web, wrappers, semences, domaines et n-uplet candidats) en tant que nœuds et les relations entre entités en tant que liens . Un n-uplet candidat est uniquement lié à des nœuds wrapper. Cela signifie que le rang d'un n-uplet candidat dans la liste finale dépend uniquement du nombre de wrappers utilisés pour extraire ce candidat particulier.

Les wrappers dans STEP sont des wrappers basés sur des expressions régulières. Ils sont générés en comparant les contextes d'une paire d'occurrences de graines dans une page Web. Cette méthode peut produire davantage de candidats-n-uplets, mais d'un autre côté, elle est très susceptible d'en extraire de faux, c'est-à-dire des candidats dont les éléments ne font pas partie de la même classe sémantique que les semences. Ces faux candidats peuvent recevoir autant de votes que les vrais candidats en raison de la nature du graphique d'entités. Ils peuvent avoir plus de votes s'ils sont extraits par de nombreux wrappers de la même page ou par d'autres wrappers d'autres pages Web. La situation inverse où les vrais candidats ont plus de votes que les faux peut également se produire. Néanmoins, il est de la plus haute importance d'extraire de vrais candidats plutôt que de faux. Telle est la motivation de cette recherche pour laquelle nous proposons une solution visant à garantir que les vrais candidats auront toujours plus de voix que les faux et se classeront plus haut sur la liste. Notre solution proposée incorpore une phase de reconstruction de n-uplets en combinant des n-uplets binaires. Nous expliquons brièvement notre méthode proposée ensuite.

Les n-uplets tels que ceux donnés à titre d'exemple plus haut sont constitués de $n$ éléments, c'est-à-dire le n-uplet n-éléments. Chaque élément est une valeur pour un attribut d'une entité du monde réel (objet). Nous utilisons le terme élément et attribut de manière interchangeable tout au long de ce chapitre. Parmi ces $n$ éléments, l'un d'entre eux peut être considéré comme l'attribut clé, tandis que tous

les éléments restants sont les valeurs des propriétés de l'attribut clé. Considérons le n-uplet <Indonésie, Jakarta, Rupiah Indonésienne> où $n = 3$, on peut certainement identifier que l'attribut clé dans le n-uplet est le pays «Indonésie», tandis que «Jakarta» et «Rupiah Indonésienne» sont les valeurs pour les attributs «capitale» et «monnaie» de l'Indonésie, respectivement. En remarquant cette notion d'attribut de clé, le n-uplet précédent peut être décomposé en deux n-uplets à 2 éléments (n-uplets binaires). Nous y parvenons en associant les valeurs d'attribut à l'attribut clé: <Indonesia, Jakarta> et <Indonesia, Roupie indonésienne>. Supposons que nous ayons la connaissance des deux n-uplets binaires a priori (en interrogeant une ontologie ou en extrayant des n-uplets binaires avec la même relation), les ajoutons au graphe d'entités en tant que nouveaux nœuds et créons des liens vers le n-uplet candidat < Indonésie, Jakarta, roupie indonésienne>, le n-uplet candidat devrait prendre plus de poids après l'application de l'algorithme PageRank. Par la suite, le n-uplet candidat devrait également se classer plus haut dans la liste finale présentée à l'utilisateur.

Sur la base de cette intuition, nous proposons le problème de la reconstruction des n-uplets. à partir des n-uplets binaires, nous reconstruisons des n-uplets de $n$-éléments ($n$> 2). Les n-uplets reconstruits peuvent ensuite être ajoutés au graphe d'entités dans le but d'enrichir le graphe. Nous montrons que l'ajout de ces n-uplets reconstruits peut améliorer la confiance dans les n-uplets candidats extraits.

### B.3.1 Définition du problème

Nous considérons des ensembles finis fixes d'étiquettes d'attribut $\mathcal{A}$ et de valeurs $\mathcal{V}$. Soit n le nombre d'attributs (éléments). Nous avons formellement défini ce qui suit.

**Definition 5.** *Un n-uplet t consiste en un ensemble de {v} où v est un mappage* $v : \mathcal{A} \rightarrow \mathcal{V}$.

Pour former un n-uplet, nous devons d'abord sélectionner un ensemble d'attributs pour une entité du monde réel. Par exemple, nous sélectionnons l'entité comme étant un pays et pour cette entité, nous choisissons $n = 3$ attributs, qui incluent la capitale, le code de la devise et le nom de la devise. Chacun de ces attributs peut prendre exactement une valeur du domaine $\mathcal{V}$. Supposons que nous prenions «Indonésie», «Jakarta», «IDR» et «Roupie indonésienne» comme valeurs des attributs, puis que nous formions le n-uplet <Indonésie, Jakarta, IDR, Roupie indonésienne>. Nous définissons ensuite l'attribut clé d'un n-uplet. Soit i, l'indice de chaque élément d'un n-uplet, respectivement.

**Definition 6.** *L'attribut clef est l'identifiant d'un n-uplet et est composé des n-1 premiers éléments du n-élément où n ≥2.*

L'attribut clef agit comme l'identifiant du n-uplet. Par exemple, dans le n-uplet <Indonesia, Jakarta>, l'attribut clé est «Indonesia» où il représente le nom d'un pays.

En suivant les définitions ci-dessus, nous pouvons définir le problème de la reconstruction des n-uplets. Le problème de la reconstruction des n-uplets concerne la

tâche de construire des n-uplets à $n$ éléments à partir de binaires. Nous proposons ici deux méthodes de reconstruction des n-uplets, à savoir stricte et lâche. Comme nous l'avons décrit brièvement plus tôt, un n-uplet à $n$ éléments peut être construit à partir d'une paire d'éléments ($n$-1)-éléments pour $n \geq 2$. Ces deux ($n$-1)-éléments n-uplets doivent avoir les mêmes éléments ordonnés de l'indice $i = 1$ à $n$-2.

**Definition 7.** *Une paire de n-uplets à n éléments peut être fusionnée pour construire un n-uplet d'éléments (n+1) s'ils ont tous deux le même attribut de clé n-1 avec n ≥2 (strict).*

Considérons la figure 5.1 où, au premier niveau du graphique, nous avons trois n-uplets binaires <Indonésie, Jakarta>, <Indonésie, IDR> et <Indonésie, roupie indonésienne>. En appliquant la définition 7 sur les n-uplets binaires, nous générons les nœuds du deuxième niveau, à savoir <Indonésie, Jakarta, IDR>, <Indonésie, Jakarta, Roupie indonésienne> et <Indonésie, IDR, Roupie indonésienne>. Deux des trois n-uplets du deuxième niveau ont le même attribut de clé selon la définition 6. Ainsi, en appliquant à nouveau la définition 7 au deuxième niveau, nous obtenons au troisième niveau le n-uplet <Indonésie, Jakarta, IDR, roupie indonésienne> en combinant les nœuds le plus à gauche et le milieu.

**Definition 8.** *Deux n-uplets de n-éléments (n ≥ 2) peuvent être fusionnés pour former (n+1)-éléments des n-uplets s'ils ont les mêmes valeurs pour les n-2 premiers éléments et diffèrent par une valeur des éléments restants (en vrac).*

La méthode stricte de génération de n-uplets peut entraîner la possibilité d'ignorer certains candidats potentiels. Nous ne pouvons pas générer le n-uplet précédent en combinant le nœud le plus à gauche ou le nœud du milieu avec le nœud le plus à droite au deuxiéme niveau de l'arborescence, car il enfreint les définitions 6 et 7. Si nous examinons de plus près les trois nœuds, nous pouvons voir qu'ils partagent le même attribut de clé $n$-2 et ne diffèrent que par l'une des valeurs d'attribut dans les $n$-1 attributs restants. Ainsi, nous définissons une autre méthode de génération des n-uplets, comme dans la définition 8. En utilisant la définition 8, nous sommes en mesure de générer le n-uplet au troisième niveau en combinant le nœud le plus à droite du deuxième niveau avec l'un des deux autres nœuds. Nous nous référons aux n-uplets générés à partir de la définition 7 et de la définition 8 en tant que n-uplets reconstruits.

### B.3.2 Approche proposée

Nous expliquons ensuite comment nous intégrons directement le processus de reconstruction des n-uplets pour construire le graphe des entités de STEP. STEP prend en entrée un ensemble de graines de n-uplet, chacune d'elles étant composée de $n$-éléments. La méthode stricte de reconstruction des n-uplets nécessite que $n \geq 2$ alors que, d'après la définition de la méthode souple, l'exigence est au moins $n$=3. Reprenons l'ensemble des graines de n-uplet de notre exemple en cours: <Indonesia, Jakarta, Rupiah indonésienne, IDR >, <Singapour, Singapour, dollar de Singapour, SGD> et <Malaisie, Kuala Lumpur, Ringgit malaisien, MYR> où $n$=4.

**Algorithme 19 :** Graphique_avec_n-uplets_reconstruits

**Entrées :** Un graphique d'entités $G = (V, E)$, un ensemble de $n$ éléments de n-uplet $R = \{T_1, T_2, ..., T_{Ns}\}$

**Sorties :** Un graphique d'entités avec des n-uplets reconstruits $G' = (V', E')$

1 $G', B$ = Récupérer_des_n-uplets_binaires($G$);
2 $B'$ ={Reconstruire_n-uplet($B$, $n$)};
3 $V' = V' \cup B' \backslash B$;
4 **pour** $i \leftarrow 3$ **à** $n$ **faire**
5 $\quad B_{new}$ = {sélectionnez des n-uplets de longueur $i$ dans $B'$ };
6 $\quad$ **pour chaque** $t \in B_{new}$ **faire**
7 $\quad\quad Anc$ = {Trouver des ancêtres de $t$};
8 $\quad\quad$ **pour chaque** $a \in Anc$ **faire**
9 $\quad\quad\quad E' = E' \cup \{< a, CONSTRUCT, t >\}$;
10 $\quad\quad$ **fin**
11 $\quad\quad Des$ = {Trouver des descendants de $t$};
12 $\quad\quad$ **pour chaque** $a \in Des$ **faire**
13 $\quad\quad\quad E' = E' \cup \{< a, INV_CONSTRUCT, t >\}$;
14 $\quad\quad$ **fin**
15 $\quad$ **fin**
16 **fin**
17 **return** $G'$;

Notre objectif est de reconstruire des n-uplets en extrayant d'abord des n-uplets binaires des graines données et en combinant récursivement les n-uplets binaires. générer n éléments n-uplets. Nous modifions l'algorithme 7 de la section 3 pour incorporer la reconstruction des n-uplets dans le processus de construction du graphe d'entités, comme indiqué dans l'algorithme 11 de la section 5 et l'algorithme 19. Dans l'algorithme 11 de la section 5, nous trouvons des n-uplets binaires à partir de l'ensemble donné de germes. Nous décomposons chaque grappe de graines en groupes de graines binaires construites en appariant en alternance l'élément clé et l'un des éléments restants (ligne 4-6). Ensuite, nous récupérons un ensemble de pages Web contenant les semences binaires. Pour chaque page Web, nous générons des wrappers et extrayons les n-uplets binaires. Nous ajoutons les entités au graphique et créons les arêtes entre elles, comme indiqué dans le tableau 3.2 (lignes 8-27). Le résultat de l'algorithme est un graphe avec l'ajout de n-uplets binaires ainsi que les relations entre les nœuds en tant qu'arêtes et l'ensemble de n-uplets binaires. Nous reconstruisons ensuite les n-uplets à partir de l'ensemble de n-uplets binaires en utilisant l'algorithme 19. Nous ajoutons tous les n-uplets reconstruits en tant que nouveaux noeuds sur le graphe, mais en laissant de côté les binaires tels qu'ils ont été ajoutés au graphe plus tôt (ligne 3). Nous commenǫns à ajouter les n-uplets reconstruits au graphe à partir de n-uplets de longueur 3 à n (ligne 3-4). Pour chaque n-uplet de l'ensemble sélectionné des n-uplets reconstruits, nous trouvons ses ancêtres et ses descendants. Les ancêtres, dans ce cas, sont l'ensemble des n-uplets ($n$-1)-éléments qui peuvent être utilisés pour construire le n-uplet de $n$ éléments correspondant. Sup-

Table B.3: Comparaison de la précision, du rappel et de la F-mesure de STEP avec et sans reconstruction du n-uplet

| **Sujet** | **Nombre de candidats** | | **Précision** | | **Rappel** | | **F-mesure** | |
|---|---|---|---|---|---|---|---|---|
| | **sans TR** | **avec TR** | **sans TR** | **avec TR** | **sans TR** | **avec TR** | **sans TR** | **avec TR** |
| DT1 | 193 (187) | 304 (207) | 0.968 | 0.68 | 0.766 | 0.848 | 0.855 | 0.754 |
| DT2 | 53 (44) | 53 (44) | 0.83 | 0.83 | 0.423 | 0.423 | 0.526 | 0.526 |
| DT3 | 451 (326) | 1235 (968) | 0.035 | 0.104 | 0.722 | 0.783 | 0.066 | 0.183 |

posons que <Royaume-Uni, Londres, Poundsterling> soit un n-uplet reconstruit, les ancêtres de ce n-uplet sont <Royaume-Uni, Londres>, <Royaume-Uni, Poundsterling>. Nous pouvons également ajouter <London, Poundsterling> à l'ensemble des ancêtres si nous générons le n-uplet correspondant à l'aide de la définition 8. Pour chacun des ancêtres, nous créons un bord pour le n-uplet correspondant et nous le désignons comme CONSTRUCT (ligne 8-10). D'autre part, les descendants sont l'ensemble des $n$+1 n-uplets reconstruits générés à partir du n-uplet correspondant. Pour le n-uplet reconstruit <Royaume-Uni, Londres, Poundsterling>, un exemple de son descendant est <Royaume-Uni, Londres, Poundsterling, GBP>. Nous créons ensuite un bord de chacun des descendants au n-uplet correspondant et affectez INV_CONSTRUCT comme étiquette (ligne 12-14).

### B.3.3 Évaluation des performances

Le tableau B.3 montre que l'intégration de la reconstruction des n-uplets dans le système d'extension des n-uplets améliore le nombre de candidats extraits et le rappel du système, en particulier dans les thèmes DT1 et DT3. Dans la rubrique DT2, les performances de STEP avec et sans la reconstruction du n-uplet sont les mêmes. Nous étudions plus avant l'impact de la reconstruction du n-uplet pour chaque sujet et détaillons ensuite le résultat.

Nous commenǫns par le sujet DT2 car il s'agit d'un cas intéressant où l'utilisation de la reconstruction des n-uplets n'améliore pas les performances de STEP en termes de candidats extraits ainsi que des trois métriques que nous utilisons. Lorsque nous donnons l'ensemble des graines <Alfa Romeo, Italie, Fiat SpA, alfaromeo.com>, <Chevrolet, états-Unis, Maîtres, chevrolet.com> en tant qu'entrée dans STEP, nous remarquons que le moteur de recherche renvoie uniquement plusieurs informations Web pages. `http://www.autosaur.com/car-brands-complete-list/` est une des URL renvoyées à partir de laquelle STEP parvient à extraire 53 candidats. Sur ces candidats, 9 sont rejetés avec référence à la vérité du sol pour diverses raisons, telles que le bruit des candidats (balises HTML), une différence mineure dans l'écriture («& amp» au lieu de «&») ou aucune référence dans le sol. la vérité («les voitures de sport adicales» ne sont pas contenues dans notre vérité au sol). Lors de la deuxième exécution de STEP (avec reconstruction des n-uplets), le nombre de candidats extraits est exactement le même. Cela signifie que la reconstruction de n-uplet ne parvient pas à ajouter même un seul candidat. Nous examinons le processus pour déterminer la raison de cet échec. Nous décomposons chacune des graines du n-uplet en trois groupes de graines binaires, à savoir le premier groupe:

<Alfa Romeo, Italie>, <Chevrolet, USA>, le deuxième groupe: <Alfa Romeo, Fiat SpA>, <Chevrolet, General Motors> et < Alfa Romeo, alfaromeo.com>, <Chevrolet, chevrolet.com>. à partir du premier groupe de graines binaires, nous essayons de trouver des relations contenant le nom d'une marque automobile et son pays d'origine. Nous avons réussi à extraire 84 candidats. En utilisant le deuxième groupe de semences binaires, à partir de l'ensemble des URL renvoyées par le moteur de recherche, nous ne parvenons qu'à extraire 22 candidats. Si nous combinons les deux groupes de candidats, nous prévoyons avoir 22 n-uplets reconstruits de longueur $n = 3$ avec les relations des marques de voitures automobiles, du pays d'origine et de la société mère. Cependant, lorsque nous utilisons le dernier groupe de semences binaires, nous ne parvenons pas à extraire les candidats de l'ensemble des URL. Cela nous empêche de reconstituer un n-uplet de longueur $n = 3$ avec le site officiel de la marque automobile comme l'un de ses éléments. De plus, cet échec nous empêche également de construire des n-uplets de longueur $n = 4$. Nous trouvons cela comme la raison pour laquelle la reconstruction du n-uplet ne contribue à aucun candidat dans le groupe final de candidats.

La reconstruction des n-uplets montre son potentiel d'amélioration du nombre de candidats sur les sujets DT1 et DT3. Dans le premier cas, STEP sans reconstruction de n-uplet extrait 193 candidats. Ce nombre est amélioré de 57% à 304 candidats par reconstruction de n-uplets. Parmi les trois groupes de semences binaires, à savoir <Indonésie, Jakarta>, <Singapour, Singapour>, <France, Paris>; <Indonésie, Rupiah Indonésienne>, <Singapour, Dollar de Singapour>, <France, Euro>; <Indonésie, IDR>, <Singapour, SGD>, <France, EUR>; nous parvenons à extraire respectivement 297, 276 et 164 candidats. à partir de ces trois groupes de candidats binaires, nous sommes alors en mesure de reconstruire 111 n-uplets de longueur $n = 4$, dont 20 sont acceptés par la vérité du sol. Les 20 nouveaux candidats contribuent à améliorer le rappel de STEP de 0,766 à 0,848. Cependant, les 91 n-uplets reconstruits contribuent à la diminution de la précision de 0,968 à 0,68. L'amélioration la plus significative de la reconstruction des n-uplets est présentée dans la rubrique DT3. STEP ne recueille que 451 candidats sans reconstitution de masse. Ce nombre est plus que triplé, passant à 1235 candidats lorsque nous intégrons la reconstruction des n-uplets dans le processus d'expansion. Les n-uplets reconstruits contribuent également à l'amélioration du rappel de 0,722 à 0,783. Cependant, les scores de précision des deux exécutions de STEP (avec et sans reconstruction de n-uplet) sont vraiment faibles. Dans la rubrique DT3, nous essayons d'extraire les relations de tous les codes d'aéroports avec leurs noms, localisations et pays respectifs avec `http://www.nationsonline.org/oneworld/IATA_Codes` comme site Web de référence. Le site Web de référence lui-même comprend 9220 entrées. Ce nombre total d'entrées dans la vérité au sol est d'une magnitude supérieure à celle des candidats obtenus de 451 et 1235 pour STEP sans et avec reconstruction du n-uplet, respectivement. Nous étudions plus en détail les candidats extraits par STEP sans reconstruction du n-uplet et nous constatons qu'ils ne sont constitués que des grands aéroports du monde. La reconstruction de n-uplet aide ensuite à ajouter 784 nouveaux candidats. Nous constatons que ces nouveaux candidats sont en réalité tous les aéroports des états-Unis. En effet, lorsque nous vérifions les URL ren-

voyées par le moteur de recherche à l'aide des graines binaires décomposées <CGK, aéroport international Soekarno-Hatta>, <CDG, aéroport international Charles De Gaulle>, <JFK, John F. Kennedy International Aéroport>; <CGK, Jakarta>, <CDG, Paris>, <JFK, New York>; <CGK, Indonésie>, <CDG, France>, <JFK, USA>; nous parvenons à extraire les relations binaires des codes d'aéroport et leurs noms, les codes d'aéroport et les emplacements correspondants de la ville et du pays à partir de la même URL `http://www.airportcodes.us/airports-by-name.htm`. Nous sommes certains que si les URL renvoyées par le moteur de recherche pointent vers des pages Web pertinentes, c'est-à-dire des pages Web contenant les relations binaires ciblées, la reconstruction du n-uplet ajoutera encore plus de nouveaux candidats. Néanmoins, d'après les résultats de l'évaluation des performances, nous concluons que la reconstruction des n-uplets a le potentiel de trouver des n-uplets candidats qui ne sont pas extraits par l'ensemble du système d'expansion des n-uplets.

## B.4 Étiquetage de sujet avec recherche de vérité

Dans la deuxième partie de la thèse, nous étudions la possibilité d'étendre sémantiquement les graines de n-uplet, c'est-à-dire en recherchant d'autres éléments correspondants des différents groupes d'éléments dans les graines. Considérez les graines de n-uplet suivantes: <Indonésie, Jakarta, Rupiah Indonésienne>, <Singapour, Singapour, Dollar de Singapour>, <France, Paris, Euro>. En examinant les premier, deuxième et troisième éléments de chaque graine, on peut supposer que ces groupes appartiennent respectivement à la classe sémantique «pays», «capitale» et «nom de la devise». Cependant, ces étiquettes de classe ne sont pas définies a priori, mais peuvent être déduites à partir des exemples d'éléments fournis. Une fois que l'étiquette correcte de chaque groupe d'éléments est identifiée, nous pouvons ensuite développer l'ensemble d'exemples en utilisant les autres membres de la classe. Par exemple, «Malaisie», «Royaume-Uni», «Allemagne» appartiennent tous à la classe sémantique «pays» et peuvent donc être utilisés pour développer le premier groupe d'éléments. Ce type de tâche consistant à rechercher automatiquement l'étiquette pour un ensemble d'exemples est étroitement lié à l'étiquetage de rubrique.

Une approche de modélisation de sujet telle que Latent Dirichlet Allocation [1] (LDA) apprend des groupes de mots décrivant le mieux les sujets latents sous-jacents à une collection de documents. Après avoir appris les mots décrivant chaque sujet, il faut ensuite trouver l'étiquette qui convient pour ce sujet. Bien que réalisable, l'étiquetage manuel exige un travail difficile et peut prendre beaucoup de temps dans certains scénarios. Cela motive la tâche de l'étiquetage de sujet. L'étiquetage de rubrique traite de la recherche automatique d'étiquettes pour les modèles de rubrique. Généralement, la tâche est principalement axée sur deux sous-tâches: la génération d'étiquettes candidates et le classement. La première consiste à récupérer ou à générer un ensemble d'étiquettes candidates, tandis que la dernière applique un mécanisme de classement sur les étiquettes candidates pour sélectionner l'étiquette la plus pertinente pour le sujet. De nombreuses méthodes ont été proposées pour abor-

der la tâche, y compris celles qui exploitent les bases de connaissances[111, 128, 105] et utilisent des algorithmes d'exploration de données ou d'apprentissage automatique [107, 110, 115]. Dans [105], les auteurs génèrent un ensemble d'étiquettes candidates principales et secondaires à partir de Wikipedia, tandis que le processus de classement est effectué sur les caractéristiques des étiquettes candidates à l'aide d'un modèle de régression à vecteur de support. Bhatia et al. [107] utilise également Wikipedia comme source pour les libellés candidats, mais ceux-ci utilisent des documents et des mots incorporés pour représenter les libellés dans le processus de classement.

Dans cette recherche, nous proposons de tirer parti de l'étiquetage par sujet pour étiqueter automatiquement l'ensemble des éléments dans les graines de n-uplet données. Nous traitons la classe sémantique comme le sujet tandis que les éléments de la classe sont l'ensemble de mots décrivant le sujet correspondant. Nous transformons le problème de la proposition de l'étiquette d'un sujet en domaine de recherche de la vérité (découverte de la vérité) où nous trouvons l'étiquette de chacun des $N$ premiers mots du sujet. L'étiquette d'un sujet est ensuite sélectionnée parmi les étiquettes des $N$ premiers mots correspondants.

### B.4.1 Approche proposée

Nous proposons d'étiqueter automatiquement les sujets générés via des modèles de sujets en analysant un graphe hétérogène reliant les différents objets et concepts caractérisant le problème: sujets, mots, articles Wikipedia, informations sémantiques auxiliaires et étiquettes candidates. Le graphique est traité à l'aide d'algorithmes de découverte de vérité afin de produire une liste classée d'étiquettes candidates pour chaque sujet. Nous décrivons d'abord la manière dont nous générons les étiquettes de candidats, puis détaillons le processus de classement.

Générer des étiquettes de candidat Notre méthode de génération d'étiquettes candidates s'inspire de Lau et al. [105]. Nous exploitons une base de connaissances. Dans notre cas, il s'agit de Wikipedia[2], en tant que source des libellés candidats. Chaque article de Wikipedia est associé à une liste de catégories. Les catégories sont placées au bas de la page vers la fin de chaque article. Nous utilisons ces catégories comme étiquettes de candidats. Nous appliquons les deux méthodes suivantes pour extraire les étiquettes candidates pour le modèle de sujet. Pour détailler chaque méthode, considérons les mots suivants (charles, prince, roi, diana, royale, reine, famille, parker, britannique, bowles) qui sont triés par ordre décroissant en fonction du poids attribué par LDA. Tout d'abord, nous utilisons chaque mot pour interroger Wikipedia et obtenir le top 10 des articles. De chacun des articles, nous extrayons les catégories en tant qu'étiquettes candidates. Ensuite, nous construisons des n-grammes à partir de l'ensemble de mots et répétons le processus.

Classement des étiquettes Après avoir généré un ensemble d'étiquettes candidates, l'étape suivante consiste à classer les étiquettes et à sélectionner l'étiquette la plus pertinente pour chaque sujet. Pour réaliser cette tâche, nous construisons

[2]http://en.wikipedia.org

d'abord un graphique d'entités avec des rubriques, des mots, des articles Wikipedia et des libellés candidats en tant que nœuds, tandis que les relations entre eux en sont les contours. Nous générons ensuite des faits à partir du graphe d'entités en tant qu'entrée dans l'algorithme de découverte de la vérité. Enfin, nous sélectionnons l'étiquette de chaque sujet en fonction du résultat de l'algorithme de découverte de vérité.

Le processus de construction du graphe d'entités commence par les $K$ sujets que nous désignons dans le graphe par Topic-1, Topic-2, ..., Topic-$K$. Chacun des sujets a un ensemble de mots, nous ajoutons donc tous les mots au graphique en tant que nœuds. Nous établissons un lien entre chaque sujet et son ensemble de mots en traçant des contours entre eux. Nous proposons deux méthodes pour récupérer les étiquettes candidates, c'est-à-dire utiliser chaque mot ou n-grammes pour interroger Wikipedia. Dans cette dernière approche, nous générons d'abord les n-grammes à partir de l'ensemble des mots $W_i$, ajoutons les n-grammes au graphique et relions chaque mot utilisé pour composer les n-grammes. Nous remplaçons ensuite $W_i$ par l'ensemble des n-grammes. à partir de chaque élément (mot / bigramme) de $W_i$, nous récupérons un ensemble d'articles de Wikipedia. Ainsi, nous relions chaque mot / bigramme et les articles Wikipedia résultants dans le graphique. Nous extrayons ensuite des catégories sur chaque article en tant qu'étiquettes candidates, nous les ajoutons au graphique et nous créons des liens entre les articles et les étiquettes candidates.

L'algorithme de découverte de la vérité prend en entrée un ensemble de faits proposés par certaines sources concernant les valeurs de propriétés de certains objets. L'objectif de l'algorithme de découverte de la vérité est de déterminer la valeur réelle des propriétés ainsi que le niveau de fiabilité de chaque source. Conformément à cette définition, dans cette recherche, nous définissons les objets, les sources et les faits comme des mots décrivant les sujets, des documents (articles Wikipedia) et des étiquettes de candidats extraites des articles, respectivement. Nous générons l'entrée dans les algorithmes de découverte de la vérité à partir du graphique d'entités construit dans notre processus de génération d'étiquettes candidates. Nous commençons par récupérer un ensemble de nœuds $W$ avec le type de «mot». Pour chaque mot $w$ dans $W$, nous récupérons ses nœuds enfants $D$ où le type de nœuds est «document». Dans le graphe construit, un nœud de type «mot» peut être directement lié à un «document» ou via les n-grammes. Dans ce dernier cas, nous trouvons d'abord les nœuds «n-grammes» dérivés du nœud «mot», puis nous collectons les nœuds «document» dans les n-grammes. Les algorithmes de découverte de la vérité ne reconnaissent pas les relations entre les objets et les sources (mots et documents), nous ne tenons donc pas compte des n-grammes lors de la génération des faits. Pour chacun des documents, nous récupérons les étiquettes et générons les faits sous forme de n-uplets comprenant le mot, l'étiquette et le document.

### B.4.2 Évaluation des performances

Nous commençons l'expérience en explorant le processus de génération d'étiquettes candidates. Nous appliquons l'approche suivante aux trois jeux de données que nous

utilisons dans l'évaluation de la performance, chaque jeu de données comprenant dix sujets. Les étiquettes de candidats sont générées en extrayant les catégories de chaque article renvoyé par Wikipedia. Nous appliquons les deux méthodes décrites précédemment pour récupérer les articles de Wikipedia. Au début, nous utilisons chaque mot comme une requête distincte de Wikipedia. Ensuite, nous générons des n-grammes à partir de l'ensemble de mots. Après chaque recherche, nous construisons notre graphique d'entités. Pour chacun des graphiques générés à l'étape précédente, nous appliquons des algorithmes de découverte de la vérité, notamment le vote à la majorité, Cosinus, Truth Finder [81], 3-Estimates [92], Depen [89] et LTM [83]. Nous mesurons ensuite la précision de chaque algorithme. Nous calculons ensuite la précision moyenne pour chaque méthode de génération d'étiquettes candidates pour chaque sujet des jeux de données via différents algorithmes de découverte de vérité. Sur la base de l'examen des résultats de l'expérience, nous concluons qu'il est préférable d'utiliser 2 grammes comme interrogation sur Wikipedia plutôt que d'interroger chaque mot.

Ensuite, nous étudions l'impact de l'utilisation de n-grammes différents pour $n$=2, 3, 4, 5 pour interroger Wikipedia. Notre intuition est que l'utilisation de plus de mots en tant que requête donnerait des articles plus spécifiques sur Wikipedia. En fin de compte, cela conduirait à des étiquettes plus spécifiques. Nous évaluons l'impact de l'utilisation de différents n-grammes sur les algorithmes suivants: vote à la majorité (MV), cosinus (CO), TruthFinder (TF), 3-estimations (3E), Depen (DP) et LTM (LT). En général, former 2 grammes de l'ensemble des $N$ premiers mots en tant que requête sur Wikipedia donne une meilleure précision que d'utiliser $n$=3, 4 ou 5. Dans cette expérience, nous nous intéressons à l'analyse du meilleur algorithme de découverte de la vérité pour notre tâche de étiquetage de sujet. Pour centrer notre examen, nous menons les expériences en utilisant la meilleure méthode de génération d'étiquettes candidates. La sélection de la méthode est basée sur l'expérience précédente. De cet ensemble d'expériences, nous concluons que le meilleur algorithme de découverte de vérité pour la tâche d'étiquetage de sujet est Depen. Lors de notre dernière expérience, nous étudions la possibilité d'améliorer les performances de la méthode proposée en faisant varier la sélection des étiquettes les plus importantes de la liste des étiquettes candidates pour le sujet. Pour cette expérience, nous utilisons uniquement le meilleur algorithme de découverte de vérité en utilisant la meilleure méthode pour générer les étiquettes candidates qui sont toutes deux basées sur les résultats des expériences précédentes. Nous constatons que le plus d'étiquettes sélectionné pour chaque sujet, plus la précision de la méthode proposée sera précise.

## B.5 Définir l'étiquetage à l'aide de la classification multi-étiquettes

Dans cette recherche, nous étudions une approche différente d'étiquetage de l'ensemble des graines de n-uplet. Au lieu de considérer chaque groupe de graines de n-uplet comme des mots décrivant les modèles de sujet décrits précédemment, nous les

traitons comme des éléments constituant un ensemble, puis nous essayons d'inférer l'étiquette appropriée pour cet ensemble en utilisant certaines caractéristiques des éléments. Pour atteindre l'objectif de déduire l'étiquette de l'ensemble, nous utilisons un classificateur multi-étiquettes. Nous illustrons la tâche que nous abordons dans cette recherche par le scénario simple et réel suivant. Considérons Alice qui recherche des produits cosmétiques. Elle navigue sur le site Web de Kiko et s'intéresse à deux produits qu'elle trouve sur le site Web: «Bright Lift Night» et «Bright Lift Mask». Malheureusement, les deux produits semblent être en rupture de stock et ne peuvent être achetés en ligne. Alice décide de rechercher des produits similaires sur les sites Web d'autres marques. Elle visite le site Web d'Yves Rocher. Elle pourrait utiliser le champ de recherche du site Web ou parcourir les catégories, mais non seulement les noms des produits, mais également leurs catégories, et Alice n'est pas sûre des mots-clés à utiliser. Le problème auquel Alice est confrontée dans le scénario décrit précédemment est de savoir comment inférer une étiquette (catégorie) pour un ensemble (l'ensemble de produits) à partir de quelques exemples de ses membres (produits). Bien entendu, dans le cas d'Alice, déduire l'étiquette nécessite plus que le nom des produits. Il est donc impératif de rechercher d'autres caractéristiques (caractéristiques) des produits pour en déduire l'étiquette. Nous définissons correctement le probléme dans la section suivante et proposons une solution possible. Dans le cas d'Alice, nous appliquons la solution proposée qui permet à Alice de découvrir les catégories de produits qui l'intéressent. La solution, appliquée sous la forme d'un outil de recherche, permet de retrouver les catégories correspondantes dans le site Web d'Yves Rocher aux catégories des produits du site Web de Kiko.

### B.5.1 Définition du problème

Nous définissons correctement la tâche d'étiquetage des ensembles que nous abordons dans ce chapitre. L'étiquetage des ensembles consiste â trouver l'étiquette la plus appropriée pour un ensemble, â partir de quelques exemples de ses membres. Les exemples de membres sont appelés des «semences», tandis que l'ensemble de ces semences constitue la requête. La tâche suppose que l'ensemble est homogène, c'est-â-dire que chaque membre de l'ensemble appartient â la même classe d'objets ou de concepts. Set labelling essaie de trouver une caractérisation appropriée, un nom ou une étiquette, pour la classe. Pour ce travail, nous considérons des ensembles de termes (mots). Par exemple, étant donné les graines «Indonésie», «Singapour» et «Malaisie», l'étiquetage des ensembles peut en déduire que ces trois objets appartiennent â un ensemble de pays, qui peuvent donc être étiquetés comme «pays».

### B.5.2 Approche proposée

Nous proposons une solution à la tâche consistant à étiqueter les ensembles en utilisant un classificateur multi-étiquettes. Dans le problème classique multi-classe, le classifieur doit choisir pour une entrée donnée une seule étiquette considérée comme la plus pertinente. Cependant, dans la tâche d'étiquetage d'ensemble, un ensem-

ble donné de semences peut être affecté à plusieurs étiquettes pertinentes. Dans ce contexte, il est plus approprié d'adopter la classification multi-étiquettes pour notre tâche, car un classifieur multi-étiquettes renvoie un ensemble d'étiquettes pertinentes pour l'entrée donnée. Le classificateur classe également les étiquettes en fonction de leur niveau de confiance, ce qui correspond également à la définition de la tâche d'étiquetage définie.

Le processus commence par l'apprentissage du classifieur multi-étiquettes à l'aide d'un jeu de données étiqueté. Chaque entrée du jeu de données est composée d'un objet, des étiquettes attribuées et des caractéristiques de l'objet. Dans cette mise en œuvre, nous choisissons les caractéristiques comme un ensemble de mots décrivant l'objet correspondant, c'est-à-dire des descriptions des produits cosmétiques. Nous transformons ensuite les entités en une dimension inférieure sous forme de sac de mots vectorisé. Pour acquérir les vecteurs sac-de-mots, nous utilisons Scikit-learn [131] en utilisant les modules CountVector et TFIDFVectorizer [132]. Nous menons également des expériences en utilisant le doc2vec [133] de gensim à titre de comparaison.

Une fois que le classificateur a été formé, l'utilisateur peut interagir avec le système en fournissant soit des mots-clés, soit une description des objets inconnus. étant donné que le classifieur multi-étiquettes est formé à l'aide de descriptions des objets et des étiquettes qui lui sont affectées, il est préférable, lors de l'interaction avec le système, de donner également une forme de description des objets inconnus sous forme de requêtes. Néanmoins, si l'utilisateur ne fournit que des mots-clés pour chaque objet, nous devons être en mesure de développer les mots-clés donnés et de les utiliser plutôt comme caractéristiques de l'objet inconnu. Nous y parvenons en intégrant LDA (Latent Dirichlet Allocation) [1] dans le processus d'expansion.

LDA est un modèle probabiliste génératif d'un corpus. LDA suppose qu'un document est composé de mélanges aléatoires sur des sujets latents, chaque sujet étant lui-même une distribution sur des mots. Dans le scénario relatif aux produits cosmétiques et aux hôtels, nous considérons respectivement les étiquettes de produits et les catégories d'hôtels. Nous utilisons les descriptions de tous les produits ou les critiques d'hôtels comme entrée et demandons à LDA d'apprendre des sujets $K$ (nombre d'étiquettes de produits ou de catégories d'hôtels). La sortie de LDA est $K$ sujets oùchaque sujet est une distribution de mots. Nous utilisons la distribution de mots dans chaque sujet dans le processus d'extension de la requête, qui est détaillé ci-après.

Pour développer la requête, nous utilisons l'algorithme 20 et donnons en entrée un ensemble de mots-clés $O$, une distribution de mots-sujets $K$ et un nombre $C$. Pour chaque mot $o$ de l'ensemble de mots-clés $O$, nous sélectionnons le sujet $z$ parmi les $K$ sujets où o a la probabilité la plus élevée (ligne 3). Word $o$ peut apparaître dans toutes les distributions de $K$ topic-words mais a des probabilités différentes. Notre intuition est de sélectionner le sujet où o a la probabilité la plus élevée, c'est-à-dire où appartient probablement à ce sujet. Du sujet $z$, nous prenons les mots du top-to pour ajouter $o$ (ligne 4). Cela signifie que les mots $T$ sont les meilleurs mots pour décrire le sujet $z$. Nous faisons la même procédure pour chaque mot $o$ et retournons l'union de tous les mots en tant que résultat d'expansion (ligne 7).

**Algorithme 20 :** Extension_de_requête

**Entrées :** Un ensemble de mots $I$, distribution du sujet $K$, entier $T$
**Sorties :** Un ensemble de mots $O$

```
1 O = I;
2 pour chaque w ∈ I faire
3 |   z ← {sélectionnez parmi K où w a la probabilité la plus élevée};
4 |   m ← {sélectionnez top-T mots de z};
5 |   O = O ∪ m;
6 fin
7 return O;
```

La même procédure est répétée autant de fois que nécessaire en fonction du nombre d'objets inconnus. Chaque requête adressée au classifieur multi-étiquettes renvoie une liste classée d'étiquettes jugées pertinentes par le classifieur avec une référence à l'entrée donnée. Les étiquettes proposées pour l'ensemble des objets inconnus sont ensuite récupérées en croisant toutes les listes d'étiquettes classées de chaque exécution de la requête. Parmi l'ensemble des étiquettes proposées, nous récupérons d'autres objets sous ces étiquettes et les présentons à l'utilisateur.

### B.5.3 Évaluation des performances

D'après les résultats de l'expérience, nous constatons que, pour le jeu de données Yves Rocher, CountVectorizer fonctionne mieux pour un petit nombre d'entités. D'autre part, les performances de TFIDFVectorizer augmentent en fonction du nombre de fonctionnalités. L'augmentation des performances de CountVectorizer et de TFIDFVectorizer est perceptible lorsque nous modifions le nombre de fonctionnalités de 100 à 300. Toutefois, un nombre supérieur à 300 n'améliore pas les performances. Dans la deuxième expérience, nous avons choisi TFIDFVectorizer pour le comparer à doc2vec. Pour Yves Rocher, nous pouvons constater que l'utilisation de doc2vec pour vectoriser la description produit des résultats faibles pour toutes les métriques. L'augmentation du nombre de fonctionnalités à 500 n'améliore pas non plus les performances. Pour évaluer les performances de notre méthode proposée d'étiquetage des ensembles, nous créons n produits inconnus à partir de chaque catégorie dans le jeu de données Yves Rocher. Pour chaque produit, nous concaténons de manière aléatoire deux descriptions de produit dans la même catégorie. Nous demandons au classificateur d'attribuer les étiquettes à chacun des produits. Nous prenons ensuite l'intersection des étiquettes et comparons le résultat avec l'étiquette d'origine pour calculer la précision. D'après les résultats de l'expérience, nous constatons que notre approche d'étiquetage d'ensemble atteint une précision moyenne de plus de 90%.

## B.6 Conclusion

Dans cette thèse, nous avons abordé plusieurs défis dans la recherche d'informations par des exemples, en particulier des informations sous la forme de relations $n$-aires (n-uplet). Nous avons présenté un certain nombre de contributions qui aident à extraire et à classer les n-uplets candidats extraits. Nous avons abordé les problèmes importants suivants: extension du jeu de n-uplets, véracité des candidats extraits, reconstruction du n-uplet et extension sémantique de la série d'exemples, qui constituent une étape prometteuse dans la réalisation d'une extraction d'information axée sur les exemples.

Dans la première partie de la thèse, nous avons généralisé le problème de l'extension set à l'extension des ensembles de n-uplets [35]. La méthode proposée prend en entrée un ensemble de n-éléments n-éléments comme exemples, tandis que les approches classiques d'expansion d'ensembles ne prennent en compte que les valeurs atomiques. Le processus de généralisation soulève des difficultés dans les phases d'exploration, de génération d'emballages et d'extraction. Notre méthode proposée applique plusieurs stratégies pour récupérer des documents pertinents basés sur les semences données, y compris la permutation des semences en tant que mots-clés du moteur de recherche. La localisation des occurrences des semences dans les documents prend également en compte le fait que chaque élément des semences peut apparaître de manière arbitraire. Nous avons mis en œuvre notre approche proposée d'un système (STEP) et démontré son efficacité en utilisant onze sujets, chaque sujet définissant un ensemble de n-uplets ou de tables. Nous avons également comparé notre système à des systèmes d'expansion de jeux à la pointe de la technologie (DIPRE [16] et Extended SEAL [30]) et montré qu'il offrait de meilleures performances en termes de précision et de rappel dans plusieurs domaines.

Dans [35], nous avons adopté le mécanisme de classement de SEAL [24] qui est basé sur un graphe d'entités où les nœuds représentent des entités alors que les arêtes entre elles dénotent les relations. Nous avons ensuite ajouté une nouvelle entité, c'est-à-dire un domaine, au graphe et à ses relations. Les candidats sont ensuite classés en fonction du poids attribué aux nœuds une fois la marche terminée. Cependant, ce mécanisme de classement ne garantit pas que les candidats ayant le poids le plus élevé sont de véritables candidats, STEP pouvant également extraire de faux candidats. Nous avons proposé de tirer parti de l'approche de recherche de la vérité pour trouver les vrais candidats parmi tous les candidats extraits par STEP [36]. Nous avons généré des faits en tant qu'entrée dans les algorithmes de recherche de la vérité à partir du graphe d'entités. Nos expériences utilisant plusieurs algorithmes de recherche de vérité de pointe sur sept sujets ont montré que Accu [89] est le meilleur algorithme. Accu apporte également une amélioration significative en termes de précision, de rappel et de mesure F par rapport au mécanisme de classement basé sur des graphiques. Dans [37], nous avons également comparé le classement des sources produit par la méthode des graphes et Accu. Nous avons constaté que l'utilisation de l'algorithme de recherche de la vérité, en particulier de Accu, peut effectivement améliorer la confiance dans le classement établi sur la base des scores de Tau de Kendall.

Nous avons proposé la tâche de reconstruction de n-uplets [38] avec l'objectif de reconstituer des uplets à $n$ éléments à partir d'éléments binaires. Cette tâche est motivée par le fait que les documents (par exemple, les pages Web) contenant des informations sous la forme de n-uplets à $n$ éléments sont rares, alors qu'au contraire, les n-uplets binaires sont très courants. Nous avons présenté deux méthodes pour reconstruire les n-uplets de $n$ éléments. Pour évaluer les performances de l'approche proposée, nous avons intégré la reconstruction du n-uplet au processus de construction du graphe d'entités dans STEP. Les résultats des expériences sur trois sujets ont montré que la reconstruction du n-uplet peut proposer de nouveaux candidats qui ne sont pas extraits par STEP. En effet, cette intégration de la reconstruction de n-uplets dans STEP a augmenté son rappel. Cependant, cette amélioration peut également avoir un impact négatif sur la précision.

Dans la deuxième partie de la thèse, nous concentrons notre attention sur l'extension sémantique de la série d'exemples. Un ensemble de $n$-éléments de n-uplets peut être considéré comme $n$ groupes de mots décrivant $n$ sujets. Connaître l'étiquette de chaque sujet permettrait d'élargir sémantiquement l'ensemble des exemples. En tant que premier travail, nous avons tiré parti de l'étiquetage des sujets pour donner des étiquettes pertinentes pour les sujets. Nous avons transformé le problème de la proposition de l'étiquette d'un sujet en domaine de recherche de la vérité [40]. Nous avons utilisé les mots comme interrogation dans une base de connaissances et récupéré les catégories des documents retournés en tant qu'étiquettes candidates. Nous avons ensuite associé une étiquette à chaque mot du sujet en utilisant l'approche de recherche de la vérité. L'étiquette de sujet est ensuite sélectionnée en tant qu'étiquette des mots du sujet avec la plus grande confiance donnée par l'approche de recherche de la vérité.

Enfin, nous avons abordé le problème de l'étiquetage des ensembles, qui consiste à trouver une étiquette appropriée pour un ensemble à partir de quelques exemples de ses membres. Nous avons considéré les éléments d'un groupe dans l'ensemble d'exemples en tant que membres d'un ensemble. Ainsi, en déduisant l'étiquette de l'ensemble à partir des caractéristiques des éléments, nous pourrions développer les exemples sémantiquement. Nous avons proposé une solution au problème en formant d'abord un classifieur multi-étiquettes à l'aide d'un jeu de données étiqueté. Chaque entrée du jeu de données désigne un objet, les étiquettes attribuées et les caractéristiques de l'objet. Nous avons utilisé la description de l'objet comme caractéristique. Une fois que nous avons formé le classifieur multi-étiquettes pour un ensemble d'objets inconnus ou inconnus, le classificateur a ensuite déduit les étiquettes pertinentes pour chacun des objets. Les étiquettes de l'ensemble sont ensuite sélectionnées en tant qu'intersection de toutes les étiquettes renvoyées par le classificateur pour tous les objets inconnus. Nous avons appliqué notre solution du problème d'étiquetage défini dans un moteur de recherche [39] et avons expérimenté deux scénarios concrets, à savoir les produits cosmétiques et les hôtels. Nous avons montré comment le moteur de recherche peut déduire les étiquettes, puis présenter à l'utilisateur tous les objets situés sous les étiquettes correspondantes.

# Bibliography


[1] D. M. Blei, A. Y. Ng, and M. I. Jordan, "Latent dirichlet allocation," *J. Mach. Learn. Res.*, vol. 3, pp. 993–1022, Mar. 2003.

[2] D. Nadeau and S. Sekine, "A survey of named entity recognition and classification," *Linguisticae Investigationes*, vol. 30, pp. 3–26, January 2007. Publisher: John Benjamins Publishing Company.

[3] L. F. Rau, "Extracting company names from text," in *[1991] Proceedings. The Seventh IEEE Conference on Artificial Intelligence Application*, vol. i, pp. 29–32, Feb 1991.

[4] K. Humphreys, R. Gaizauskas, S. Azzam, C. Huyck, B. Mitchell, H. Cunningham, and Y. Wilks, "University of sheffield: Description of the lasie-ii system as used for muc-7," 06 2001.

[5] S. Sekine and C. Nobata, "Definition, dictionaries and tagger for extended named entity hierarchy.," in *LREC*, European Language Resources Association, 2004.

[6] O. Etzioni, M. Cafarella, D. Downey, A.-M. Popescu, T. Shaked, S. Soderland, D. S. Weld, and A. Yates, "Unsupervised named-entity extraction from the web: An experimental study," *Artif. Intell.*, vol. 165, pp. 91–134, June 2005.

[7] A. McCallum and W. Li, "Early results for named entity recognition with conditional random fields, feature induction and web-enhanced lexicons," in *Proceedings of the Seventh Conference on Natural Language Learning at HLT-NAACL 2003 - Volume 4*, CONLL '03, (Stroudsburg, PA, USA), pp. 188–191, Association for Computational Linguistics, 2003.

[8] G. Lample, M. Ballesteros, S. Subramanian, K. Kawakami, and C. Dyer, "Neural architectures for named entity recognition," *CoRR*, vol. abs/1603.01360, 2016.

[9] D. Klein, J. Smarr, H. Nguyen, and C. D. Manning, "Named entity recognition with character-level models," in *Proceedings of the Seventh Conference on Natural Language Learning at HLT-NAACL 2003 - Volume 4*, CONLL '03, (Stroudsburg, PA, USA), pp. 180–183, Association for Computational Linguistics, 2003.

[10] D. M. Bikel, R. Schwartz, and R. M. Weischedel, "An algorithm that learns what's in a name," *Machine Learning*, vol. 34, pp. 211–231, Feb 1999.

[11] J. P. C. Chiu and E. Nichols, "Named entity recognition with bidirectional lstm-cnns," *CoRR*, vol. abs/1511.08308, 2015.

[12] M. Habibi, L. Weber, M. L. Neves, D. L. Wiegandt, and U. Leser, "Deep learning with word embeddings improves biomedical named entity recognition," *Bioinformatics*, vol. 33, no. 14, pp. i37–i48, 2017.

[13] N. Kambhatla, "Combining lexical, syntactic, and semantic features with maximum entropy models for extracting relations," in *Proceedings of the ACL 2004 on Interactive Poster and Demonstration Sessions*, ACLdemo '04, (Stroudsburg, PA, USA), Association for Computational Linguistics, 2004.

[14] Z. GuoDong, S. Jian, Z. Jie, and Z. Min, "Exploring various knowledge in relation extraction," in *Proceedings of the 43rd Annual Meeting on Association for Computational Linguistics*, ACL '05, (Stroudsburg, PA, USA), pp. 427–434, Association for Computational Linguistics, 2005.

[15] D. P. T. Nguyen, Y. Matsuo, and M. Ishizuka, "Relation extraction from wikipedia using subtree mining," in *Proceedings of the 22Nd National Conference on Artificial Intelligence - Volume 2*, AAAI'07, pp. 1414–1420, AAAI Press, 2007.

[16] S. Brin, "Extracting patterns and relations from the world wide web," in *WebDB*, pp. 172–183, 1999.

[17] E. Agichtein and L. Gravano, "Snowball: Extracting relations from large plain-text collections," in *Proceedings of the Fifth ACM Conference on Digital Libraries*, DL '00, (New York, NY, USA), pp. 85–94, ACM, 2000.

[18] R. Gabbard, M. Freedman, and R. Weischedel, "Coreference for learning to extract relations: Yes, virginia, coreference matters," in *Proceedings of the 49th Annual Meeting of the Association for Computational Linguistics: Human Language Technologies: Short Papers - Volume 2*, HLT '11, (Stroudsburg, PA, USA), pp. 288–293, Association for Computational Linguistics, 2011.

[19] J. Chen, D. Ji, C. L. Tan, and Z. Niu, "Relation extraction using label propagation based semi-supervised learning," in *Proceedings of the 21st International Conference on Computational Linguistics and the 44th Annual Meeting of the Association for Computational Linguistics*, ACL-44, (Stroudsburg, PA, USA), pp. 129–136, Association for Computational Linguistics, 2006.

[20] T. Hasegawa, S. Sekine, and R. Grishman, "Discovering relations among named entities from large corpora," in *Proceedings of the 42Nd Annual Meeting on Association for Computational Linguistics*, ACL '04, (Stroudsburg, PA, USA), Association for Computational Linguistics, 2004.

[21] J. Chen, D.-H. Ji, C. L. Tan, and Z.-Y. Niu, "Unsupervised feature selection for relation extraction," in *IJCNLP*, 2005.

[22] M. Mintz, S. Bills, R. Snow, and D. Jurafsky, "Distant supervision for relation extraction without labeled data," in *Proceedings of the Joint Conference of the 47th Annual Meeting of the ACL and the 4th International Joint Conference on Natural Language Processing of the AFNLP: Volume 2 - Volume 2*, ACL '09, (Stroudsburg, PA, USA), pp. 1003–1011, Association for Computational Linguistics, 2009.

[23] O. Lopez de Lacalle and M. Lapata, "Unsupervised relation extraction with general domain knowledge," in *Proceedings of the 2013 Conference on Empirical Methods in Natural Language Processing*, pp. 415–425, Association for Computational Linguistics, 2013.

[24] R. C. Wang and W. W. Cohen, "Language-independent set expansion of named entities using the web," in *ICDM*, pp. 342–350, 2007.

[25] D. X. Yeye He, "SEISA: Set expansion by iterative similarity aggregation," in *WWW*, 2011.

[26] C. Wang, K. Chakrabarti, Y. He, K. Ganjam, Z. Chen, and P. A. Bernstein, "Concept expansion using web tables," in *WWW*, pp. 1198–1208, 2015.

[27] Z. Chen, M. Cafarella, and H. V. Jagadish, "Long-tail vocabulary dictionary extraction from the web," in *WSDM*, pp. 625–634, 2016.

[28] V. Crescenzi, "RoadRunner: Towards Automatic Data Extraction from Large Web Sites," *International Conference on Very Large Data Bases (VLDB)*, 2001.

[29] M. Banko, M. J. Cafarella, S. Soderland, M. Broadhead, and O. Etzioni, "Open information extraction from the web," in *Proceedings of the 20th International Joint Conference on Artifical Intelligence*, IJCAI'07, (San Francisco, CA, USA), pp. 2670–2676, Morgan Kaufmann Publishers Inc., 2007.

[30] R. C. Wang and W. W. Cohen, "Character-level analysis of semi-structured documents for set expansion," in *EMNLP*, pp. 1503–1512, 2009.

[31] R. McDonald, F. Pereira, S. Kulick, S. Winters, Y. Jin, and P. White, "Simple algorithms for complex relation extraction with applications to biomedical ie," in *Proceedings of the 43rd Annual Meeting on Association for Computational Linguistics*, ACL '05, (Stroudsburg, PA, USA), pp. 491–498, Association for Computational Linguistics, 2005.

[32] N. Jayaram, A. Khan, C. Li, X. Yan, and R. Elmasri, "Querying knowledge graphs by example entity tuples," in *ICDE*, pp. 1494–1495, May 2016.

[33] R. Gilleron, P. Marty, M. Tommasi, and F. Torre, "Interactive tuples extraction from semi-structured data," in *2006 IEEE/WIC/ACM International Conference on Web Intelligence (WI 2006 Main Conference Proceedings)(WI'06)*, pp. 997–1004, Dec 2006.

[34] C. Bădică and A. Bădică, "Rule learning for feature values extraction from html product information sheets," in *Rules and Rule Markup Languages for the Semantic Web* (G. Antoniou and H. Boley, eds.), (Berlin, Heidelberg), pp. 37–48, Springer Berlin Heidelberg, 2004.

[35] N. A. S. Er, T. Abdessalem, and S. Bressan, "Set of t-uples expansion by example," in *iiWAS*, pp. 221–230, 2016.

[36] N. A. S. Er, M. L. Ba, T. Abdessalem, and S. Bressan, "Truthfulness of candidates in set of t-uples expansion," in *Database and Expert Systems Applications* (D. Benslimane, E. Damiani, W. I. Grosky, A. Hameurlain, A. Sheth, and R. R. Wagner, eds.), (Cham), pp. 314–323, Springer International Publishing, 2017.

[37] N. A. S. Er, M. L. Ba, T. Abdessalem, and S. Bressan, "Set of tuples expansion by example with reliability," *International Journal of Web Information Systems*, vol. 13, no. 4, pp. 425–444, 2017.

[38] N. A. S. Er, M. L. Ba, T. Abdessalem, and S. Bressan, "Tuple reconstruction," in *DASFAA Workshops*, vol. 10829 of *Lecture Notes in Computer Science*, pp. 239–254, Springer, 2018.

[39] N. A. S. Er, J. Read, T. Abdessalem, and S. Bressan, "Set labelling using multi-label classification," in press.

[40] N. A. S. Er, M. L. Ba, T. Abdessalem, and S. Bressan, "Harnessing truth discovery algorithms on the topic labelling problem," in press.

[41] W. Zhang, A. Ahmed, J. Yang, V. Josifovski, and A. J. Smola, "Annotating needles in the haystack without looking: Product information extraction from emails," in *KDD*, pp. 2257–2266, 2015.

[42] Y. Sheng, S. Tata, J. B. Wendt, J. Xie, Q. Zhao, and M. Najork, "Anatomy of a privacy-safe large-scale information extraction system over email," in *24th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pp. 734–743, 2018.

[43] A. Q. Mahlawi and S. Sasi, "Structured data extraction from emails," in *2017 International Conference on Networks Advances in Computational Technologies (NetACT)*, pp. 323–328, July 2017.

[44] A. Arasu and H. Garcia-Molina, "Extracting structured data from web pages," Technical Report 2002-40, Stanford InfoLab, July 2002.

[45] T. Abdessalem, B. Cautis, and N. Derouiche, "Objectrunner: Lightweight, targeted extraction and querying of structured web data," *PVLDB*, vol. 3, no. 2, pp. 1585–1588, 2010.

[46] D. Qiu, L. Barbosa, X. L. Dong, Y. Shen, and D. Srivastava, "Dexter: Large-scale discovery and extraction of product specifications on the web," *Proc. VLDB Endow.*, vol. 8, pp. 2194–2205, Sept. 2015.

[47] T. Furche, G. Gottlob, G. Grasso, X. Guo, G. Orsi, C. Schallhart, and C. Wang, "Diadem: Thousands of websites to a single database," *Proc. VLDB Endow.*, 2014.

[48] M. Faheem and P. Senellart, "Adaptive web crawling through structure-based link classification," in *ICADL*, pp. 39–51, 2015.

[49] Z. Chen, M. Cafarella, J. Chen, D. Prevo, and J. Zhuang, "Senbazuru: A prototype spreadsheet database management system," *Proc. VLDB Endow.*, vol. 6, pp. 1202–1205, Aug. 2013.

[50] Z. Chen, M. J. Cafarella, and E. Adar, "Diagramflyer: A search engine for data-driven diagrams," in *WWW*, 2015.

[51] K. Bollacker, C. Evans, P. Paritosh, T. Sturge, and J. Taylor, "Freebase: A collaboratively created graph database for structuring human knowledge," in *Proceedings of the 2008 ACM SIGMOD International Conference on Management of Data*, SIGMOD '08, (New York, NY, USA), pp. 1247–1250, ACM, 2008.

[52] 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 '07, (New York, NY, USA), pp. 697–706, ACM, 2007.

[53] M. E. Califf and R. J. Mooney, "Relational learning of pattern-match rules for information extraction," in *Proceedings of the Sixteenth National Conference on Artificial Intelligence and the Eleventh Innovative Applications of Artificial Intelligence Conference Innovative Applications of Artificial Intelligence*, AAAI '99/IAAI '99, (Menlo Park, CA, USA), pp. 328–334, American Association for Artificial Intelligence, 1999.

[54] C. Fellbaum, *WordNet: An Electronic Lexical Database*. Bradford Books, 1998.

[55] S. Soderland, "Learning information extraction rules for semi-structured and free text," *Mach. Learn.*, vol. 34, pp. 233–272, Feb. 1999.

[56] B. Adelberg, "Nodose—a tool for semi-automatically extracting structured and semistructured data from text documents," in *Proceedings of the 1998 ACM SIGMOD International Conference on Management of Data*, SIGMOD '98, (New York, NY, USA), pp. 283–294, ACM, 1998.

[57] C.-H. Chang and S.-C. Lui, "Iepad: Information extraction based on pattern discovery," in *Proceedings of the 10th International Conference on World Wide Web*, WWW '01, (New York, NY, USA), pp. 681–688, ACM, 2001.

[58] C.-H. Chang and S.-C. Kuo, "Olera: semisupervised web-data extraction with visual support," *IEEE Intelligent Systems*, vol. 19, pp. 56–64, 2004.

[59] A. Hogue and D. Karger, "Thresher: Automating the unwrapping of semantic content from the world wide web," in *Proceedings of the 14th International Conference on World Wide Web*, WWW '05, (New York, NY, USA), pp. 86–95, ACM, 2005.

[60] J. Wang and F. H. Lochovsky, "Data extraction and label assignment for web databases," in *Proceedings of the 12th International Conference on World Wide Web*, WWW '03, (New York, NY, USA), pp. 187–196, ACM, 2003.

[61] P. P. Talukdar, T. Brants, M. Liberman, and F. Pereira, "A context pattern induction method for named entity extraction," in *Proceedings of the Tenth Conference on Computational Natural Language Learning*, CoNLL-X '06, (Stroudsburg, PA, USA), pp. 141–148, Association for Computational Linguistics, 2006.

[62] M. Paşca, "Weakly-supervised discovery of named entities using web search queries," in *Proceedings of the Sixteenth ACM Conference on Conference on Information and Knowledge Management*, CIKM '07, (New York, NY, USA), pp. 683–690, ACM, 2007.

[63] Y. Zhang, Y. Xiao, S.-w. Hwang, and W. Wang, "Long concept query on conceptual taxonomies," *arXiv preprint arXiv:1511.09009*, 2015.

[64] P. Pantel, E. Crestan, A. Borkovsky, A.-M. Popescu, and V. Vyas, "Web-scale distributional similarity and entity set expansion," in *Proceedings of the 2009 Conference on Empirical Methods in Natural Language Processing: Volume 2 - Volume 2*, EMNLP '09, (Stroudsburg, PA, USA), pp. 938–947, Association for Computational Linguistics, 2009.

[65] X. Zhang, Y. Chen, J. Chen, X. Du, K. Wang, and J.-R. Wen, "Entity set expansion via knowledge graphs," in *Proceedings of the 40th International ACM SIGIR Conference on Research and Development in Information Retrieval*, SIGIR '17, (New York, NY, USA), pp. 1101–1104, ACM, 2017.

[66] L. Sarmento, V. Jijkuon, M. de Rijke, and E. Oliveira, ""more like these": Growing entity classes from seeds," in *CIKM*, pp. 959–962, 2007.

[67] E. Riloff and J. Shepherd, "A corpus-based approach for building semantic lexicons," 06 1997.

[68] D. Widdows and B. Dorow, "A graph model for unsupervised lexical acquisition," in *Proceedings of the 19th International Conference on Computational Linguistics - Volume 1*, COLING '02, (Stroudsburg, PA, USA), pp. 1–7, Association for Computational Linguistics, 2002.

[69] M. A. Hearst, "Automatic acquisition of hyponyms from large text corpora," in *Proceedings of the 14th Conference on Computational Linguistics - Volume 2*, COLING '92, (Stroudsburg, PA, USA), pp. 539–545, Association for Computational Linguistics, 1992.

[70] Z. Kozareva, E. Riloff, and E. Hovy, "Semantic class learning from the web with hyponym pattern linkage graphs," in *Proceedings of ACL-08: HLT*, pp. 1048–1056, Association for Computational Linguistics, 2008.

[71] S. P. Igo and E. Riloff, "Corpus-based semantic lexicon induction with web-based corroboration," in *Proceedings of the Workshop on Unsupervised and Minimally Supervised Learning of Lexical Semantics*, UMSLLS '09, (Stroudsburg, PA, USA), pp. 18–26, Association for Computational Linguistics, 2009.

[72] X.-L. Li, L. Zhang, B. Liu, and S.-K. Ng, "Distributional similarity vs. pu learning for entity set expansion," in *Proceedings of the ACL 2010 Conference Short Papers*, ACLShort '10, (Stroudsburg, PA, USA), pp. 359–364, Association for Computational Linguistics, 2010.

[73] Z. Ghahramani and K. A. Heller, "Bayesian sets," in *Advances in Neural Information Processing Systems 18* (Y. Weiss, B. Schölkopf, and J. C. Platt, eds.), pp. 435–442, MIT Press, 2006.

[74] L. Zhang and B. Liu, "Entity set expansion in opinion documents," in *Proceedings of the 22Nd ACM Conference on Hypertext and Hypermedia*, HT '11, (New York, NY, USA), pp. 281–290, ACM, 2011.

[75] E. Fredkin, "Trie memory," *Commun. ACM*, vol. 3, pp. 490–499, Sept. 1960.

[76] B. G. Malkiel, *A Random Walk Down Wall Street.* Norton, New York, 1973.

[77] L. Page, S. Brin, R. Motwani, and T. Winograd, "The pagerank citation ranking: Bringing order to the web.," Technical Report 1999-66, Stanford InfoLab, November 1999.

[78] L. C. Freeman, "Centrality in social networks conceptual clarification," *Social Networks*, p. 215, 1978.

[79] S. Auer, C. Bizer, G. Kobilarov, J. Lehmann, R. Cyganiak, and Z. Ives, "Dbpedia: A nucleus for a web of open data," in *Proceedings of the 6th International The Semantic Web and 2Nd Asian Conference on Asian Semantic Web Conference*, ISWC'07/ASWC'07, (Berlin, Heidelberg), pp. 722–735, Springer-Verlag, 2007.

[80] W. Wu, H. Li, H. Wang, and K. Q. Zhu, "Probase: A probabilistic taxonomy for text understanding," in *Proceedings of the 2012 ACM SIGMOD International Conference on Management of Data*, SIGMOD '12, (New York, NY, USA), pp. 481–492, ACM, 2012.

[81] X. Yin, J. Han, and P. S. Yu, "Truth discovery with multiple conflicting information providers on the web," *IEEE TKDE*, June 2008.

[82] X. L. Dong, L. Berti-Equille, and D. Srivastava, "Truth discovery and copying detection in a dynamic world," *PVLDB*, vol. 2, August 2009.

[83] B. Zhao, B. I. P. Rubinstein, J. Gemmell, and J. Han, "A Bayesian approach to discovering truth from conflicting sources for data integration," *PVLDB*, vol. 5, February 2012.

[84] Z. Zhao, J. Cheng, and W. Ng, "Truth discovery in data streams: A single-pass probabilistic approach," in *CIKM*, (Shangai, China), November 2014.

[85] J. Bleiholder, K. Draba, and F. Naumann, "FuSem: exploring different semantics of data fusion," in *VLDB*, (Vienna, Austria), 2007.

[86] X. L. Dong and F. Naumann, "Data fusion: Resolving data conflicts for integration," *PVLDB*, vol. 2, August 2009.

[87] R. Pochampally, A. Das Sarma, X. L. Dong, A. Meliou, and D. Srivastava, "Fusing data with correlations," in *SIGMOD*, (Snowbird, Utah, USA), May 2014.

[88] Q. Li, Y. Li, J. Gao, B. Zhao, W. Fan, and J. Han, "Resolving conflicts in heterogeneous data by truth discovery and source reliability estimation," in *SIGMOD*, (Snowbird, Utah, USA), May 2014.

[89] X. L. Dong, L. Berti-Equille, and D. Srivastava, "Integrating conflicting data: the role of source dependence," *PVLDB*, vol. 2, August 2009.

[90] M. L. Ba, L. Berti-Equille, K. Shah, and H. M. Hammady, "VERA: A platform for veracity estimation over web data," in *WWW*, 2016.

[91] W. Liu, J. Liu, H. Duan, J. Zhang, W. Hu, and B. Wei, "TruthDiscover: Resolving object conflicts on massive linked data," in *WWW*, pp. 243–246, 2017.

[92] A. Galland, S. Abiteboul, A. Marian, and P. Senellart, "Corroborating information from disagreeing views," in *WSDM*, (New York, USA), February 2010.

[93] D. Wang, L. Kaplan, H. Le, and T. Abdelzaher, "On truth discovery in social sensing: A maximum likelihood estimation approach," in *IPSN*, (Beijing, China), April 2012.

[94] J. Pasternack and D. Roth, "Latent credibility analysis," in *WWW*, (Rio de Janeiro, Brazil), May 2013.

[95] X. L. Dong, L. Berti-Equille, Y. Hu, and D. Srivastava, "Global detection of complex copying relationships between sources," *VLDB Endow.*, vol. 3, September 2010.

[96] X. S. Fang, "Truth discovery from conflicting multi-valued objects," in *WWW*, pp. 711–715, 2017.

[97] X. S. Fang, Q. Z. Sheng, X. Wang, and A. H. Ngu, "Value veracity estimation for multi-truth objects via a graph-based approach," in *WWW*, pp. 777–778, 2017.

[98] M. L. Ba, R. Horincar, P. Senellart, and H. Wu, "Truth finding with attribute partitioning," in *WebDB SIGMOD Workshop*, (Melbourne, Australia), May 2015.

[99] L. Berti-Equille, "Data Veracity Estimation with Ensembling Truth Discovery Methods," in *IEEE Big Data Workshop*, 2015.

[100] T. Yamane, *Statistics an introductory analysis.* Harper & Row, 1969.

[101] T. L. Griffiths and M. Steyvers, "Finding scientific topics," *Proceedings of the National Academy of Sciences*, vol. 101, pp. 5228–5235, April 2004.

[102] T. Minka and J. Lafferty, "Expectation-Propagation for the Generative Aspect Model," in *Proceedings of the 18th Conference on Uncertainty in Artificial Intelligence*, pp. 352–359, 2002.

[103] Q. Mei, X. Shen, and C. Zhai, "Automatic labeling of multinomial topic models," in *Proceedings of the 13th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, KDD '07, (New York, NY, USA), pp. 490–499, ACM, 2007.

[104] S. Kullback and R. A. Leibler, "On information and sufficiency," *Ann. Math. Statist.*, vol. 22, no. 1, pp. 79–86, 1951.

[105] J. H. Lau, K. Grieser, D. Newman, and T. Baldwin, "Automatic labelling of topic models," in *Proceedings of the 49th Annual Meeting of the Association for Computational Linguistics: Human Language Technologies - Volume 1*, HLT '11, (Stroudsburg, PA, USA), pp. 1536–1545, Association for Computational Linguistics, 2011.

[106] K. Grieser, T. Baldwin, F. Bohnert, and L. Sonenberg, "Using ontological and document similarity to estimate museum exhibit relatedness," *J. Comput. Cult. Herit.*, vol. 3, pp. 10:1–10:20, Feb. 2011.

[107] S. Bhatia, J. H. Lau, and T. Baldwin, "Automatic labelling of topics with neural embeddings," *CoRR*, vol. abs/1612.05340, 2016.

[108] Q. V. Le and T. Mikolov, "Distributed representations of sentences and documents," *CoRR*, vol. abs/1405.4053, 2014.

[109] T. Mikolov, K. Chen, G. Corrado, and J. Dean, "Efficient estimation of word representations in vector space," *CoRR*, vol. abs/1301.3781, 2013.

[110] W. Kou, F. Li, and T. Baldwin, "Automatic labelling of topic models using word vectors and letter trigram vectors," in *Information Retrieval Technology - 11th Asia Information Retrieval Societies Conference, AIRS 2015, Brisbane, QLD, Australia, December 2-4, 2015. Proceedings*, pp. 253–264, 2015.

[111] I. Hulpus, C. Hayes, M. Karnstedt, and D. Greene, "Unsupervised graph-based topic labelling using dbpedia," in *Proceedings of the Sixth ACM International Conference on Web Search and Data Mining*, WSDM '13, (New York, NY, USA), pp. 465–474, ACM, 2013.

[112] M. Newman, *Networks: An Introduction*. New York, NY, USA: Oxford University Press, Inc., 2010.

[113] K. Stephenson and M. Zelen, "Rethinking centrality: Methods and examples," *Social Networks*, vol. 11, no. 1, pp. 1–37, 1989.

[114] M. E. J. Newman, "A measure of betweenness centrality based on random walks," *Social Networks*, vol. 27, pp. 39–54, 2005.

[115] A. E. C. Basave, Y. He, and R. Xu, "Automatic labelling of topic models learned from twitter by summarisation," in *Proceedings of the 52nd Annual Meeting of the Association for Computational Linguistics, ACL 2014, June 22-27, 2014, Baltimore, MD, USA, Volume 2: Short Papers*, pp. 618–624, 2014.

[116] A. K. McCallum, "Multi-label text classification with a mixture model trained by em," 1999.

[117] S. Godbole and S. Sarawagi, "Discriminative methods for multi-labeled classification," *Advances in Knowledge Discovery and Data Mining*, pp. 22–30, 2004.

[118] C. Vens, J. Struyf, L. Schietgat, S. Džeroski, and H. Blockeel, "Decision trees for hierarchical multi-label classification," *Mach. Learn.*, vol. 73, pp. 185–214, Nov. 2008.

[119] R. E. Schapire and Y. Singer, "Boostexter: A boosting-based systemfor text categorization," *Mach. Learn.*, vol. 39, pp. 135–168, May 2000.

[120] G. Tsoumakas and I. Katakis, "Multi-label classification: An overview," *Int J Data Warehousing and Mining*, vol. 2007, pp. 1–13, 2007.

[121] J. Read, B. Pfahringer, G. Holmes, and E. Frank, "Classifier chains for multi-label classification," *Machine Learning*, vol. 85, p. 333, Jun 2011.

[122] V. I. Levenshtein, "Binary codes capable of correcting deletions, insertions and reversals.," *Soviet Physics Doklady*, vol. 10, no. 8, pp. 707–710, 1966. Doklady Akademii Nauk SSSR, V163 No4 845-848 1965.

[123] R. C. Wang and W. W. Cohen, "Iterative set expansion of named entities using the web," in *Proceedings of the 2008 Eighth IEEE International Conference on Data Mining*, ICDM '08, (Washington, DC, USA), pp. 1091–1096, IEEE Computer Society, 2008.

[124] H. Tong, C. Faloutsos, and J.-Y. Pan, "Fast random walk with restart and its applications," in *Proceedings of the Sixth International Conference on Data Mining*, ICDM '06, (Washington, DC, USA), pp. 613–622, IEEE Computer Society, 2006.

[125] Z. Ghahramani and K. A. Heller, "Bayesian sets," in *Advances in Neural Information Processing Systems 18* (Y. Weiss, B. Schölkopf, and J. C. Platt, eds.), pp. 435–442, MIT Press, 2006.

[126] D. A. Waguih, N. Goel, H. M. Hammady, and L. Berti-Equille, "AllegatorTrack: Combining and reporting results of truth discovery from multi-source data," in *ICDE*, (Seoul, Korea), 2015.

[127] M. Kendall, *Rank correlation methods*. London: Griffin, 1948.

[128] M. Allahyari and K. J. Kochut, "Automatic topic labeling using ontology-based topic models," *2015 IEEE 14th International Conference on Machine Learning and Applications (ICMLA)*, pp. 259–264, 2015.

[129] T. Joachims, "Text categorization with suport vector machines: Learning with many relevant features," in *Proceedings of the 10th European Conference on Machine Learning*, ECML '98, (London, UK, UK), pp. 137–142, Springer-Verlag, 1998.

[130] S. Gardner and H. Nesi, "A classification of genre families in university student writing," *Applied Linguistics*, vol. 34, no. 1, pp. 25–52, 2013.

[131] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay, "Scikit-learn: Machine learning in python," *J. Mach. Learn. Res.*, vol. 12, pp. 2825–2830, Nov. 2011.

[132] L. Buitinck, G. Louppe, M. Blondel, F. Pedregosa, A. Mueller, O. Grisel, V. Niculae, P. Prettenhofer, A. Gramfort, J. Grobler, R. Layton, J. VanderPlas, A. Joly, B. Holt, and G. Varoquaux, "Api design for machine learning software: experiences from the scikit-learn project.," *CoRR*, vol. abs/1309.0238, 2013.

[133] R. Řehůřek and P. Sojka, "Software Framework for Topic Modelling with Large Corpora," in *Proceedings of the LREC 2010 Workshop on New Challenges for NLP Frameworks*, (Valletta, Malta), pp. 45–50, ELRA, May 2010.

[134] T. Fawcett, "An introduction to roc analysis," *Pattern Recogn. Lett.*, vol. 27, pp. 861–874, June 2006.

[135] J. H. Lau and T. Baldwin, "An empirical evaluation of doc2vec with practical insights into document embedding generation," *CoRR*, vol. abs/1607.05368, 2016.