HAL
open science

Keyword Search and Summarization Approaches for RDF Dataset Exploration

Mohamad Rihany

To cite this version:

Mohamad Rihany. Keyword Search and Summarization Approaches for RDF Dataset Exploration. Data Structures and Algorithms [cs.DS]. Université Paris-Saclay, 2022. English. NNT : 2022UPASG030 . tel-03679017

HAL Id: tel-03679017

https://theses.hal.science/tel-03679017v1

Submitted on 25 May 2022

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

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

Approches de recherche par mot clé et de résumé pour l'exploration de données RDF

Keyword Search and Summarization Approaches for RDF Data Exploration

Thèse de doctorat de l'Université Paris-Saclay

École doctorale n° 580 Sciences et Technologies de l'information et de la communication (STIC) Spécialité de doctorat : Informatique
Graduate School : Informatique et sciences du numérique
Référent : Université de Versailles saint-Quentin en Yvelines

Thèse préparée dans l'unité de recherche **DAVID** (Université Paris-Saclay, UVSQ),
sous la direction de **Zoubida KEDAD**, Maîtresse de conférences
le co-encadrement de **Stéphane LOPES**, Maître de conférences

Thèse soutenue à Versailles, le 30 mars 2022, par

Mohamad RIHANY

Composition du jury

Yolaine Bourda Professeure, Centrale Supélec Université Paris-Saclay	Présidente
Amel Bouzeghoub Professeur, Telecom SudParis	Rapporteur & Examinatrice
Elisabeth Métais Professeur, Conservatoire National des Arts et Métiers	Rapporteur & Examinatrice
Jose Antonio Fernandes de Macedo Proffesseur, Université Fédérale de Ceará (Brésil)	Examinateur
Zoubida Kedad Maîtresse de conférences, Université Paris-Saclay	Directrice de thèse

THESE DE DOCTORAT

NNT : 2022UPASG030

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

Titre: Approches de recherche par mot-clé et de résumé pour l'exploration de données RDF

Mots clés: Exploration de données, Ressource Description Framework, recherche par mot-clé, résumé d'un graphe RDF, identification de thèmes

Résumé: Un nombre croissant de sources de données sont publiées sur le web, exprimées dans les langages proposés par le W3C comme RDF, RDFS et OWL. Ces sources représentent un volume de données sans précédent disponible pour les utilisateurs et les applications. Afin d'identifier les sources les plus pertinentes et de les utiliser, il est nécessaire d'en connaître le contenu, par exemple au moyen de requêtes écrites en Sparql, le langage d'interrogation proposé par le W3C pour les sources de données RDF. Mais cela nécessite, en plus de la maîtrise du langage Sparql, de disposer de connaissances sur le contenu de la source en termes de ressources, classes ou propriétés qu'elle contient. L'objectif de ma thèse est d'étudier des approches permettant de fournir un support à l'exploration d'une source de données RDF. Nous avons proposé deux approches complémentaires, la recherche mots-clés et le résumé d'un graphe RDF.

La recherche mots-clés dans un graphe RDF renvoie un ou plusieurs sous-graphes en réponse à une requête exprimée comme un ensemble de termes à rechercher. Chaque sous-graphe est l'agrégation d'éléments extraits du graphe initial, et représente une réponse possible à la requête constituée par un ensemble de mots-clés. Les sous-graphes retournés peuvent être classés en fonction de leur pertinence. La recherche par mot-clé dans des sources de données RDF soulève les problèmes suivants : (i) l'identification pour chaque mot-clé de la requête des éléments correspondants dans le graphe considéré, en prenant en compte les différences de terminologies existant entre les mots-clés et les termes utilisés dans le graphe RDF, (ii) la combinaison des éléments de graphes retournés pour construire un sous-graphe résultat en utilisant des algorithmes d'agrégation capable de déterminer la meilleure façon de relier les éléments du graphe correspondant à des mots-clés, et enfin (iii), comme il peut exister plusieurs éléments du graphe qui correspondent à un même mot-clé, et par conséquent plusieurs sous-graphes résultat, il s'agit d'évaluer la pertinence de ces sous-graphes par l'utilisation de métriques appropriées. Dans notre travail, nous avons proposé une approche de recherche par mot-clé qui apporte des solutions aux problèmes ci-dessus.

Fournir une vue résumée d'un graphe RDF peut être utile afin de déterminer si ce graphe correspond aux besoins d'un utilisateur particulier en mettant en évidence ses éléments les plus importants ; une telle vue résumée peut faciliter l'exploration du graphe. Dans notre travail, nous avons proposé une approche de résumé originale fondée sur l'identification des thèmes sous-jacents dans un graphe RDF. Notre approche de résumé consiste à extraire ces thèmes, puis à construire le résumé en garantissant que tous les thèmes sont représentés dans le résultat. Cela pose les questions suivantes : (i) comment identifier les thèmes dans un graphe RDF ? (ii) quels sont les critères adaptés pour identifier les éléments les plus pertinents dans les sous-graphes correspondants à un thème ? (iii) comment connecter les éléments les plus pertinents pour créer le résumé d'un thème ? et enfin (iv) comment générer un résumé pour le graphe initial à partir des résumés de thèmes ? Dans notre travail, nous avons proposé une approche qui fournit des réponses à ces questions et qui produit une représentation résumée d'un graphe RDF garantissant que chaque thème y est représenté proportionnellement à son importance dans le graphe initial.

Title: Keyword Search and Summarization Approaches for RDF Data Exploration

Keywords: Data Exploration, Resource Description Framework, Keyword Search, RDF Graph Summarization, Theme Identification

Abstract: An increasing number of datasets are published on the Web, expressed in the standard languages proposed by the W3C such as RDF, RDF (S), and OWL. These datasets represent an unprecedented amount of data available for users and applications. In order to identify and use the relevant datasets, users and applications need to explore them using queries written in SPARQL, a query language proposed by the W3C. But in order to write a SPARQL query, a user should not only be familiar with the query language, but also have some knowledge about the content of the RDF dataset in terms of the resources, classes or properties it contains. The goal of this thesis is to provide approaches to support the exploration of these RDF datasets. We have studied two alternative and complementary exploration techniques, keyword search and summarization of an RDF dataset.

Keyword search returns RDF graphs in response to a query expressed as a set of keywords, where each resulting graph is the aggregation of elements extracted from the source dataset. These graphs represent possible answers to the keyword query, and they can be ranked according to their relevance. Keyword search in RDF datasets raises the following issues: (i) identifying, for each keyword in the query, the matching elements in the considered dataset, taking into account the differences of terminology between the keywords and the terms used in the RDF dataset, (ii) combining the matching elements to build the result by defining aggregation algorithms that find the best way of linking the matching elements, and finally (iii), finding appropriate metrics to rank the results, as several matching elements may exist for each keyword and consequently several graphs may be returned. In our work, we propose a keyword search approach that addresses these issues.

Providing a summarized view of an RDF dataset can help a user in identifying if this dataset is relevant to his needs, and in highlighting its most relevant elements. This could be useful for the exploration of a given dataset. In our work, we propose a novel summarization approach based on the underlying themes of a dataset. Our theme-based summarization approach consists of extracting the themes in a data source and building the summarized view so as to ensure that all these discovered themes are represented. This raises the following questions: (i) how to identify the underlying themes in an RDF dataset? (ii) what are the suitable criteria to identify the relevant elements in the themes extracted from the RDF graph? (iii) how to aggregate and connect the relevant elements to create a theme summary? and finally, (iv) how to create the summary for the whole RDF graph from the generated theme summaries? In our work, we propose a theme-based summarization approach for RDF datasets which answers these questions and provides a summarized representation ensuring that each theme is represented proportionally to its importance in the initial dataset.

Acknowledgements

First of all, I would like to express my sincere gratitude to my thesis supervisors, Prof. Zoubida Kedad and Prof. Stéphane Lopes for their continuous support throughout this thesis, their insightful comments, patience, motivation, and immense knowledge. . I learned a lot from them and I address them with my gratitude for all this, it was a real pleasure to work together. I could not have imagined having better advisors and mentors for my Ph.D study.

Besides my advisors, I would like to thank the rest of my thesis committee. Specifically, I value the effort of Prof. Amel Bouzeghoub and Prof. Elisabeth Métais in reviewing my dissertation. I truly appreciate their constructive feedback and for the time they have kindly devoted to it. I also deliver my sincere gratitude to Prof. Yolaine Bourda and Prof. Jose Antonio Fernandes de Macedo, it is an honor having you as my jury member and having my work validated by you.

My sincere thanks also go to all members of DAVID laboratory and in particular the members of ADAM team and especially, Prof. Karine Zeitouni, Dr. Béatrice Finance, and Dr. Yehia Taher. I had the chance to meet them and get their feedback on several occasions. In addition, I would like to thank my friends and colleagues from the DAVID laboratory. We were not only able to support each other by deliberating over our problems and findings but also happily by talking about things other than just our papers.

Last but not the least, I would like to thank my family: my parents Afif and Zeinab, my sisters, and my brother for supporting me spiritually throughout writing this thesis and my life in general.

My deepest gratitude goes to my lovely wife Malak, you will always be my best friend my psychologist, and my life. Everything I have reached is because of your continuous support and love.

Table of contents

1 Introduction **1**
1.1 Context and Motivation 1
1.2 Challenges 3
1.3 Contribution 4
1.4 Organization of the Manuscript 5

2 State of the Art **7**
2.1 Introduction 8
2.2 Preliminaries 10
2.2.1 The Resource Description Framework 10
2.2.2 RDFS and OWL 12
2.2.3 The SPARQL Query Language 12
2.2.4 Linked data and the Semantic Web 13
2.3 Keyword Search 13
2.3.1 Identifying the matching elements 14
2.3.1.1 Node Matching 15
2.3.1.2 Node and Edge Matching 16
2.3.1.3 Pattern Matching 17
2.3.2 Aggregating the matching elements 20
2.3.2.1 Aggregating Matching Elements by using the Shortest Path 20
2.3.2.2 Aggregating Matching Elements by using Backward Propagation 21
2.3.2.3 Aggregating Matching Elements by using a Summarized Graph 22
2.3.2.4 Ad-hoc aggregation methods 24
2.3.3 Ranking Answers Subgraphs 25
2.3.4 Discussion and open issues 26
2.4 Summarization of an RDF Graph 29
2.4.1 Summarization based on Node Level Metrics 29
2.4.1.1 Quotient Based Summarization 30

2.4.1.2 Centrality based Summarization 36
2.4.1.3 Statistical Summarization 40
2.4.1.4 Hybrid Summarization 42
2.4.2 Summarization Based on Graph Analysis Techniques 44
2.4.2.1 Community structure 44
2.4.2.2 Label propagation 45
2.4.2.3 Theme Identification in RDF Graphs 47
2.4.2.4 InWalk [14] 49
2.4.2.5 Spectral Partitioning [5, 2, 71] 51
2.4.2.6 Kernighan-lin(KL) [48] 52
2.4.3 Discussion and open issues 54
2.5 Conclusion 57

3 Semantic-Based Matching for Keyword Search in RDF Data 59
3.1 Introduction 60
3.2 Approach Overview 62
3.3 Problem statement 64
3.4 Indexing an RDF Data Graph 66
3.4.1 Structure of the Index 67
3.4.2 Building the Index 68
3.5 External Knowledge used to Enhance the Matching Process 69
3.5.1 Semantic relations to bridge the terminological gap 70
3.5.2 Equivalence between properties and paths 71
3.5.2.1 SameResult Pattern 73
3.5.2.2 SameProperty Pattern 74
3.5.2.3 Property Pattern 74
3.6 Enhancing the Matching Process with Semantic Relations 75
3.6.1 Finding Equivalent Matching Elements 76
3.6.1.1 Exact Matching 76
3.6.1.2 Synonymy Relations 77
3.6.1.3 Antonymy Relations 77
3.6.2 Finding Close Matching Elements 79
3.7 Using Patterns in the Matching Process 80
3.8 Semantic relations and pattern in the Matching Process 82
3.9 Experimental evaluation 86
3.10 Conclusion 90

4 Keyword Search for RDF data: an Aggregation Approach 91
4.1 Introduction 92
4.2 Problem Statement 94
4.3 Aggregation as Steiner tree problem 96

4.3.1 Preliminaries 97
4.3.2 Building the answer Tree 101
4.3.2.1 Adaptation for DNH 102
4.3.2.2 Adaptation of Kruskal's Algorithm 104
4.3.2.3 Using Adapted DNH and Kruskal's algorithm in Building the final sub-graph Result 105
4.3.3 Building the Answer Sub-graph 108
4.4 Ranking the Results 109
4.4.1 Using the Type of Matching Element in the Result 110
4.4.2 Using the Proportion of the Nodes Corresponding to Matching Elements in the Result 112
4.4.3 Aggregating Different Rankings 112
4.5 Experimental Evaluation 113
4.5.1 Prototype Environment 113
4.5.2 Datasets 114
4.5.3 Results 114
4.6 Conclusion 115

5 Theme-Based Summarization for RDF Datasets 117
5.1 Introduction 117
5.2 Motivating Example 119
5.3 Problem Statement 121
5.4 Overview of the Approach 122
5.5 Theme Identification in an RDF Graph 124
5.6 Theme Summarization 125
5.6.1 Evaluating Node Relevance in an RDF Graph 126
5.6.2 Identifying the Relevant Nodes 128
5.6.3 Building a Theme Summary 130
5.7 Building the final summary 134
5.8 Evaluation 137
5.9 Conclusion 140

6 Conclusion 141
6.1 Summary of the contributions 141
6.2 Future Works 142

A Résumé en Français 145

List of Figures

2.1 An Example of RDF Dataset about Movies 11
2.2 Graph representation for the RDF data in figure 2.1 11
2.3 An example of RDF graph describing universities and scientists[38] 17
2.4 Predicate and Predicate Paths that can be mapped to Relation Phrases 18
2.5 Example of sub-graph which matches the Brother keyword 19
2.6 Example of an RDF data graph 23
2.7 An Augmented summary graph 23
2.8 query pattern graph 24
2.9 An RDF graph describing movies 31
2.10 An Example of Graph-Structured Data [19] 32
2.11 Example of an RDF Graph describing information about Students, Courses, Articles and Professors 33
2.12 Weak Summary of the RDF graph in Figure 2.11 33
2.13 Typed Weak Summary of the RDF graph in Figure 2.11 34
2.14 Strong Summary of the RDF graph in Figure 2.11 34
2.15 Typed Strong Summary of the RDF graph in Figure 2.11 35
2.16 Summary of the RDF graph in Figure 2.11 38
2.17 Knowledge graph [80] 39
2.18 patterns extracted from the graph in figure 3.9 [80] 39
2.19 Summary of the RDF graph in Figure 2.11 after applying the approach presented in [12] 40
2.20 Summary of the RDF graph in Figure 2.11 after applying the ExpLOD approach [49] 41
2.21 Summary of the RDF graph in Figure 2.11 after applying the LODSight approach [61] 42
2.22 Summarized graph after applying the utility algorithm on the RDF graph in Figure 2.11 43
2.23 Summarized graph after applying the above algorithm on the RDF graph in Figure 2.11 44
2.24 Undirected Weighted Graph 45

2.25 Appying LPA on the graph of Zacharys Karate club network [74] . . 46
2.26 Initial step of label propagation [33] 46
2.27 Detecting overlapping community by using label propagation [33] . 47
2.28 Example of k-cores in a graph with the MCODE algorithm 48
2.29 Clustering the graph with MCODE algorithm 49
2.30 Finding overlapping communities with MCODE algorithm 49
2.31 Clique Percolation Method [67] 50
2.32 Practical example for Clique Percolation Method [67] 51
2.33 Graph G with adjacency diagonal and Laplacian matrix of this graph. 51
2.34 Eigenvector and Eigenvalue for the Laplacian matrix in figure 2.33 . 52
2.35 Extracting communities with spectral partitioning algorithm on graph of figure 2.33 52
2.36 Graph partitioned into two equal subset of nodes 53
2.37 Graph partitioned into two equal subset of nodes with minimum number of cut edges after applying KL algorithm 54

3.1 An Example of RDF Dataset describing Movies 61
3.2 Approach Overview 63
3.3 An RDF graph 65
3.4 Graphical representation for the index for some words in the graph of figure 3.1 67
3.5 An RDF graph about movie 71
3.6 Path equivalent to the "Brother" property 73
3.7 Two Nodes are equivalent by using SameResult Pattern 73
3.8 Two properties are equivalent by using SameProperty Pattern . . . 74
3.9 Property "co-starring" equivalent to a path 75
3.10 Matching Elements for the keyword "co-starring" in Q11 75
3.11 Matching Element for the keyword "Brother" in Q12 81
3.12 Matching Elements for the keyword "co-starring" in query Q12 . . . 82
3.13 Overview of the Matching Elements Process 83
3.14 Example of Equivalent Path for the Keyword Brother 85
3.15 Matching Elements for each Keyword in Query Q13 85
3.16 Average Execution Time According to the Size of the Query 88
3.17 Execution Time for Each Query over DBpedia 88
3.18 Execution Time for Each Query over AIFB 89

4.1 RDF graph Example 92
4.2 RDF graph about Movies 94
4.3 RDF graph showing different types of matching elements 95
4.4 Sub-graph results for the keyword query Q3 96
4.5 Undirected weighted graph 97

4.6 Distance Graph for the graph in figure 4.5 98
4.7 Minimum Spanning tree for the graph in figure 4.6 98
4.8 The average and standard deviation of critical parameters 99
4.9 From the Minimum Spanning tree to the Steiner Tree 100
4.10 101
4.11 Matching Elements for each Keyword in the Query Q9 101
4.12 Matching Elements for each Keyword in the Query Q9 102
4.13 RDF graph of figure4.2 after executing the translation rules 103
4.14 Distance graph constructed from the graph of figure 4 and C_4 combination 107
4.15 Building the Minimum Spanning Tree on the Distance Graph of figure 4.14 107
4.16 Replacing the Nodes and the Edges in the MST 108
4.17 Solution for the Keyword Query Q9 after using the combination C4 from the list of combination in the figure 4.12 109
4.18 Second possible Solution for the Keyword Query Q9 after using the combination C2 from the list of combination in the figure 4.12 . . . 110
4.19 An RDF graph describing movies 111
4.20 Two possible answers for the query Q10 111
4.21 Two Possible sub-graph Results for the Query Q10 112
4.22 Average Execution Time According to the Size of the Query 114

5.1 Example of RDF graph with set of themes 120
5.2 Graph summary for the graph in figure5.1 120
5.3 A Theme-Based Summary for the graph in figure 5.1 121
5.4 Thematic-Based Summarization Framework 123
5.5 Example of an RDF Graph Describing Movies 127
5.6 From Distance Graph to summarized graph 133
5.7 Final Summarized Graph 133
5.8 Final Summarized Graph 134
5.9 Path Selection for Building the Final Summary 135
5.10 Example of a RDF graph with three different themes 135
5.11 Summarized Themes and the Most Central Nodes in these Themes 136
5.12 Final Summary 137
5.13 Average Execution Time 138
5.14 Number of Extracted Themes 138

List of Tables

2.1 Relation Phrase and Supporting Entity Pairs 18
2.2 Paraphrase Dictionary 19
2.3 Summary of some Keyword Search Approaches 28
2.4 External and Internal costs for all the nodes in A 53
2.5 External and Internal costs for all the nodes in B 53
2.6 The gain score of all the possible combinations of exchanging a node in set A to another node in set B 54
2.7 Comparative Table of Summarization 56

3.1 Example of Keyword Queries 87
3.2 Number of Results for each Keyword Query(DBpedia) 88
3.3 Number of Results for each Keyword Query(AIFB) 89
3.4 Top-K precision of the basic approach and the semantic based matching 89

4.1 The Centrality Degree for the Edges in the Distance Graph of figure4.14 108
4.2 Number of Results for each Keyword Query(DBpedia) 115
4.3 Top-K Precision 115

5.1 Number of extracted themes and the execution time for samples from Olympics dataset 139
5.2 Top-k precision 140

Chapter 1

Introduction

Contents

1.1 Context and Motivation 1
1.2 Challenges 3
1.3 Contribution 4
1.4 Organization of the Manuscript 5

1.1 Context and Motivation

An increasing number of interlinked datasets are published on the web. They are described in languages proposed by the W3C, such as the Resource Description Framework (RDF). These interlinked datasets represent a massive amount of knowledge available for numerous applications. However, to use these data sources meaningfully, the user first needs to understand them and have some knowledge about their content. In our work, we are interested in providing the users with tools to explore the data sources in order to get an insight about their content. Indeed, the most important issue in this setting is enabling the users to discover which data are relevant to their needs and ranges of interest.

The natural way of interacting with an RDF dataset is to use a query language such as SPARQL[72]. However, to write the SPARQL query, the user should have some knowledge about the data source such as, the classes and resources it contains, the list of their properties, the vocabulary used if any, etc. To acquire this knowledge, the user should manually browse and explore the content of the RDF and check the properties used in the data set. The manual browsing process consists of finding a seed node in the graph, issuing queries to get the properties of this seed, and then repeating the process with neighboring nodes until the

relevant information is found. But this task is very complex and takes a lot of time. Moreover, the user should also be familiar with the SPARQL query language in order to write the different queries.

Some approaches aim to propose alternative ways of querying an RDF dataset, such as keyword search approaches and natural language querying approaches. These approaches require some knowledge about the content of a dataset in order to choose the appropriate keywords.

Some approaches have been proposed in order to provide a description of the content of an RDF dataset, which could be very useful to support query formulation. These approaches have different goals, and deal with different problems. We can find among them schema discovery approaches, summarization approaches, profiling and visualization approaches, or topic identification approaches. Some of these approaches are based on quantitative information on the data, and they generate some statistics on the different elements contained in the data source, such as the number of classes, or the number of triples described by a given property. Thus, the user will have a global vision of the source, but he must use the SPARQL query language to query the data and get the information he needs. There are also works proposing exploring RDF data sources in cases where the user has no information and knowledge about the underlying data set. Since the RDF data can be represented as a graph, thus the approaches propose creating a summarized RDF graph to summarize the main RDF data set. The proposed approaches aim to provide the user with a concise representation containing the most relevant information in the graph to ease its exploitation; its purpose is to extract meaningful information from the RDF graph representing their content as faithfully as possible.

In our work, we have focused on two complementary ways of supporting the user during the exploration of an RDF dataset. The first one is to provide an alternative to querying the dataset using the well-known dedicated query languages. One possible approach is keyword search, which has been addressed by several works [3, 23, 10, 85, 90, 41]. The second way of supporting the user during his exploration of the source is to provide him with a structural summarized description of the dataset representing its most important elements. Some research works have proposed such summarization techniques [18, 58, 94]. Both families of approaches can be used to help a user or an application willing to use the dataset without having a detailed knowledge about the resources or properties in the dataset. Although several solutions have been proposed for these two problems, some issues are still open. In the next section, I will present the most important challenges that face both keyword search and summarization.

1.2 Challenges

Our goal consists in providing support for exploring and querying the RDF dataset to allow the user to retrieve useful information and understand the content of the RDF graph. For this purpose, we have explored two distinct but complementary paths. The first one is keyword search, used to query an RDF dataset without knowing the properties and the resources it contains.The second one is summarization, which aims at creating a summarized graph to help the user in the process of finding out whether the dataset contains information which is relevant to his needs.

Keyword search helps the user to remove the obstacle of the complexity of the formulation of a query in a SPARQL query language. This allows the exploration and retrieval of relevant information for the user. The main key issues that face keyword search can be summarized as follow: (1) Identifying the relevant elements: how to identify the elements in the dataset that can be matched to the given query keyword? (2) Aggregating the relevant elements: after identifying the relevant elements we need to connect them to create a sub-graph. How do we aggregate the relevant elements and construct the results to be returned for the user? What is the best way to connect two relevant elements related to a query? Can we measure the importance of the paths in the final results? and finally, (3) Ranking the final results: since we might have more that one sub-graph result for a given keyword query, then we need to find a method in order to rank these different results. What is the best way to rank the set of final results to the query?

We might have a query keyword with no relevant elements for this keyword in the dataset. This is because there is no such concept in the dataset which can be matched with this keyword or there is an equivalent concept in the graph but this concept expressed using different terminology. The challenge is how to bridge the terminological gap between the keyword query and the concept used in the graph? In particular, this is one of the challenges that we will address. Once we have identified relevant elements, we need to aggregate them to create the sub-graph result and this is the second challenge. The third challenge we are addressing is to identify the best metric to be used to rank the different results.

Summarization provides a concise representation for the content of an RDF dataset. This representation reflects the most relevant elements in the graph. This work revolves around two main challenges linked to the problem of summarizing RDF graphs: (1) Identifying the relevant elements: we need to identify the elements in the graph that will be used to create the summarized graph. What criteria are used to identify these elements? Do we use the same criteria for all the typed of nodes? (2) Aggregating the relevant elements: after selecting the relevant elements we need to aggregate and connect them to create the summarized graph. What is the method used to aggregate and connect the relevant elements to create

a summarized graph?

1.3 Contribution

Our work has studied some of the possible ways to provide support to RDF datasets exploration and querying. More precisely, we are interested in two problems: keyword search and summarization of RDF graph. Keyword search could be useful when the user has very little knowledge about the dataset, its topic or the different concepts used to describe the content of the dataset, and therefore he can not write a sparql query corresponding to his needs. Moreover, summarization could be useful when the user needs to know the different concepts that are described in the dataset and how these concepts are connected and presented. The rest of this section details our contributions.

We have proposed an approach for keyword search in RDF data [75]. We target the problem of solving the terminological gap that exists between the keywords of the queries provided by the user and the different elements and terms of the graph representing the RDF dataset. The proposed approach uses an external source of knowledge providing semantic relations in an online linguistic resource. These relations are integrated into the matching part of the approach to bridge the terminological gap between keyword query and RDF terms in order to improve the quality of the results. We have defined two different types of matching according to a hierarchy of semantic relations. This knowledge also helps the user explore the RDF data by using his own terms without being restricted to the terms used in the RDF graph.

Each keyword query can be matched with one or more elements from the RDF dataset; these elements can be a node, an edge, or a subgraph. These elements should be connected to form the subgraph result to the query. Consequently, we have proposed a new approach to improve the way of connecting the matching elements to build a meaningful final result [45]. In this approach, a novel method is presented to aggregate the matching elements and find the best paths between them to extract the sub-graph corresponding to the keyword query. Our solution is an adaptation of two algorithms. The first one is used to solve the Steiner tree problem, and the second one is used to extract the minimum spanning tree. We have used a score to calculate the importance of the path connecting two elements in a graph. This score is used to determine the best path when selecting the one to be chosen in the process of building the final sub-graph result. Finally, we have proposed a ranking method to rank the set of possible answers based on the semantic relations used during the matching process.

We have also proposed an approach for summarizing RDF graphs [76]. Unlike the existing approaches, our approach relies on the detection of the underlying

themes or topics in the considered dataset. It ensures that the most relevant nodes of each theme are present and that their representativity is reflected in the summary. The underlying themes are first identified in the graph, then each of them is summarized. The global summary is then built from the set of theme summaries. In order to detect a theme in the graph, we rely on previous works presented in [65], where a theme is represented by a dense area in the graph, i.e., an area where nodes are highly connected. The summary of a given theme is built by selecting the most relevant nodes in the graph. We have introduced an extended definition of node centrality. We have proposed an aggregation method to find the best paths between theme summaries. Moreover, we have stated theme-based summarization as a Steiner tree problem. To this end, we have adapted existing algorithms which have been proposed to solve the Steiner tree problem and finding the minimum spanning tree, to connect the different summarized themes to build the final summary for the RDF graph.

1.4 Organization of the Manuscript

This manuscript consists of five chapters apart from this introduction.

Chapter 2 presents a state of the art related to the problems we have addressed. We survey the recent works on both keyword search in RDF graphs and RDF graph summarization. We present each existing keyword search approach focusing on the way it has dealt with the different challenges posed by keyword search. The challenges can be summarized as follows: (i)**Matching keywords** where we need to find the best matching elements in the dataset that matches with the query keywords (ii)**Aggregating Graph Elements** The challenge is how to connect and aggregate the matching elements (iii)**Result Ranking**. As each keyword may have more than one matching element in the dataset, there may be several possible results to the query. The problem is to rank the different results and to find a ranking method capable of determining if there are better results than others. We present existing summarization approaches and we categorize them according to the technique used to build the summary. For each family of existing approaches, we compare and analyse the different works and we discuss their limitations.

Chapter 3 deals with the matching process in keyword search. This problem consists of finding the elements in the RDF graph that can be matched to the query keyword. First, an overview of the keyword search approach is presented. Second, the dataset indexation technique is presented, as well as the external knowledge source used in the matching process. Third, our matching process is discussed. Finally, an experimental evaluation is provided to compare the basic matching process with the enhanced one using an external source of knowledge.

Chapter 4 deals with another challenge of keyword search in RDF graphs, which

is aggregating the graph elements matching the keywords of the query. We state our problem as a Steiner tree problem; then we describe the algorithms used to solve this problem. We define the notion of centrality score associated to a path, which is used to reflect the importance of the path connecting two different elements in the graph. We also detail the proposed algorithms and their adaptations to solve our problem. The proposed approach is evaluated by comparing different aggregation techniques according to their execution time and the total number of results created.

In chapter 5, we present a novel summarization approach for an RDF graph. We first present our method for identifying the different themes in the graph, and we describe the identification of the relevant elements in each theme. We then present a method to aggregate the relevant elements in the theme. We also describe the algorithm for connecting the different themes summaries. The proposed approach is tested by using another method of summarization which is the baseline approach where summarization is performed solely on the basis of centrality, without considering the underlying themes. This test is done to show the difference in the respective execution times; we also present the precision of the computed result.

Finally, we provide a conclusion in Chapter 6, where we sum up our contributions and show how our proposal can help querying RDF graphs. We discuss the open problems and present some future works.

Chapter 2

State of the Art

Contents

2.1 Introduction **8**

2.2 Preliminaries **10**

2.2.1 The Resource Description Framework 10

2.2.2 RDFS and OWL 12

2.2.3 The SPARQL Query Language 12

2.2.4 Linked data and the Semantic Web 13

2.3 Keyword Search **13**

2.3.1 Identifying the matching elements 14

2.3.1.1 Node Matching 15

2.3.1.2 Node and Edge Matching 16

2.3.1.3 Pattern Matching 17

2.3.2 Aggregating the matching elements 20

2.3.2.1 Aggregating Matching Elements by using the Shortest Path 20

2.3.2.2 Aggregating Matching Elements by using Backward Propagation 21

2.3.2.3 Aggregating Matching Elements by using a Summarized Graph 22

2.3.2.4 Ad-hoc aggregation methods 24

2.3.3 Ranking Answers Subgraphs 25

2.3.4 Discussion and open issues 26

2.4 Summarization of an RDF Graph **29**

2.4.1 Summarization based on Node Level Metrics 29
2.4.1.1 Quotient Based Summarization 30
2.4.1.2 Centrality based Summarization 36
2.4.1.3 Statistical Summarization 40
2.4.1.4 Hybrid Summarization 42
2.4.2 Summarization Based on Graph Analysis Techniques . . 44
2.4.2.1 Community structure 44
2.4.2.2 Label propagation 45
2.4.2.3 Theme Identification in RDF Graphs 47
2.4.2.4 InWalk [14] 49
2.4.2.5 Spectral Partitioning [5, 2, 71] 51
2.4.2.6 Kernighan-lin(KL) [48] 52
2.4.3 Discussion and open issues 54
2.5 Conclusion 57

2.1 Introduction

An increasing number of data sources are published on the web, expressed in languages proposed by the W3C, such as RDF, RDFS, and OWL. These data sources are available for users and applications, but using them is not an easy task, since the users need to have some knowledge about the content of the data source before querying them. They can be queried by using dedicated languages proposed by W3C such as SPARQL. However, to write a SPARQL query, the user should first have some information about the content of the data source, such as the schema, the vocabularies, or the list of the properties used in this data source. To do so, the user must first browse the RDF data source and look for the labels, properties and instances that are of interest to him. Then the user could formulate the queries that will provide the relevant information by using the collected information from the previous step. Moreover, the user should be familiar with the SPARQL syntax; a query in this language consists of triple patterns with variables, properties and resources that should be matched to the RDF data source.

In this work, we are interested in providing support to exploit meaningfully the RDF datasets. In other words, our goal is to enable and guide the interrogation of RDF datasets, considering that a user or an application does not necessarily have a detailed knowledge about the content of this dataset, which makes the use of the SPARQL query a complex process. If the user do not have information about

the types, properties, and resources contained in the dataset, he will not be able to write the SPARQL query. We have explored two distinct ways to support the process of exploiting the dataset. The first one is keyword search, which enables answering queries formulated as a set of keywords, regardless of the types and properties of the dataset. Keyword search raises several issues, one of these issues is the mismatch between the terminology of the dataset and the one of the queries.

When we use keyword search, sometimes we can find the same label for what we are looking for and sometimes it is not straightforward but we can find something in the dataset which is closed in meaning or even equivalent to the keyword query. The challenge consists of determining all the possibilities of matching between the keyword query and the dataset. Another challenge consists of finding the match between number expressed in character. For example how can we match the keyword query "one" with a node labelled with "1".

Keyword search is a technique used to retrieve information from large amount of data. Keyword search approaches have been proposed for various type of data, such as databases or Web pages. It consists of entering a set of keywords into the system, which will then match these keywords with the data and connect the keywords to provide a set of sub-graphs as an answer.

In the context of an RDF dataset, answering a query composed of a set of keywords consist firstly in retrieving from the dataset the elements which are the best matches for each keyword, and secondly in connecting the matching elements in order to provide the sub-graph which represents a possible answer to the initial query. As several matching elements can be retrieved for each keyword, processing a single query may lead to several results. This leads to an important issue which is identifying the best answer, this requires us to be able to find a ranking method to compare the set of possible answers and identify the best ones.

Providing a summary of an RDF dataset is another way of supporting the users and the applications willing to use the dataset. Summarizing RDF datasets consists in extracting a concise representation that gives insight into the dataset's content. The challenge here is to define what are the relevant elements and how they will be selected. The evaluated element can be a node, an edge or set of connected nodes and edges, in some cases we need to define an appropriate metrics to evaluate the relevance of the different elements in the dataset; in other cases a method is needed to determine which paths or sub-graphs need to be selected as relevant elements. In some of the summarization approaches, the elements of the dataset that are present in the summary, are selected based on their importance in the dataset; in other approaches, the elements are selected based on similarity between the elements or on the most similarity of the paths. Another challenge consists in building the summary from the selected relevant elements.

The summarization of the dataset consists of first selecting the most relevant

elements, then connecting those elements to create the summarized graph. The relevant elements differ from one approach to another, some of them use the most central nodes, and others use different metrics to identify the important classes, but the main idea is to select the elements that represent important information in the dataset. The main concept of summarization is to reduce the number of elements as much as possible but without loosing the important information described by the dataset.

In this chapter we will study existing approaches for both keyword search and summarization. The preliminaries are presented in the section 2.2. The second part, presented in section 2.3, is devoted to keyword search. We will present the challenges raised by this problem and the related works, as well as a comparative analysis of these works. The third part, presented in section 2.4, deals with summarization approaches. We will present the different summarization techniques, as well as the different ways of evaluating the relevance of elements in an RDF dataset. For both sections, we will provide a discussion and the open research issues for the considered problem. Finally, we will provide a conclusion in section 2.5 highlighting the open problems.

2.2 Preliminaries

This section introduces preliminaries and concepts used in our work. First, we present the Resource Description Framework (RDF), then a brief definition for the Web Ontology Language (OWL) is presented; We also introduce SPARQL which is the query language for RDF. Finally, we present the concepts related to linked open data cloud and the semantic web.

2.2.1 The Resource Description Framework

Resource Description Framework (RDF) [51] is a language proposed by W3C to describe the datasets published on the web. RDF triples are used to express descriptions of resources. The building block in RDF is a triple <subject, predicate, object>. The subject is a resource represented by URI or a blank node and the object is the description of the resource and it can be described by URI, or it can be a blank node or even a literal. For example, <The_God_Father_2, starring, Al_Pacino> is a triple where the subject is the resource "The_God_Father_2" and the object is "Al_Pacino".

The Uniform Resource Identifiers (URIs) are used to identify anything, including real-world objects, such as people and places, concepts, or information resources such as web pages and books. The URIs are shared across all Web documents existing worldwide; It allows different people to reuse the URI to refer to

the same resource.

The literal is a constant value of different types such as a string or a number. The predicate is the relation that connects the subject and the object, and it is identified by the a URI.

The Resource Description Framework (RDF) dataset can be represented as a graph model where the subjects and the objects are the nodes, and the predicates are the edges connecting the nodes.

Informally, sentence "The God Father 2 movie was released in 12-12-1974" can be represented as an RDF triple as shown in figure 2.1, where the subject is "The God Father 2", the predicate is "release date" and the object is "12-12-1974".

```
<?xml version="1.0" encoding="utf-8" ?>
<rdf:RDF xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
         xmlns:schema="http://schema.org/"
         xmlns:ns0="http://dbpedia.org/ontology/">

  <schema:Movie rdf:about="http://dbpedia.org/resource/The_God_Father_2">
    <ns0:starring rdf:resource="http://dbpedia.org/resource/Al-Pacino"/>
    <ns0:starring rdf:resource="http://dbpedia.org/resource/Robert_De_Niro"/>
    <ns0:releaseDate rdf:datatype="http://www.w3.org/2001/XMLSchema#date">1974-12-12</ns0:releaseDate>
    <ns0:director rdf:resource="http://dbpedia.org/resource/Francis_Ford_Coppola"/>
  </schema:Movie>

</rdf:RDF>
```

Figure 2.1: An Example of RDF Dataset about Movies

These triples can also be represented as a graph, as shown in Figure 2.2.

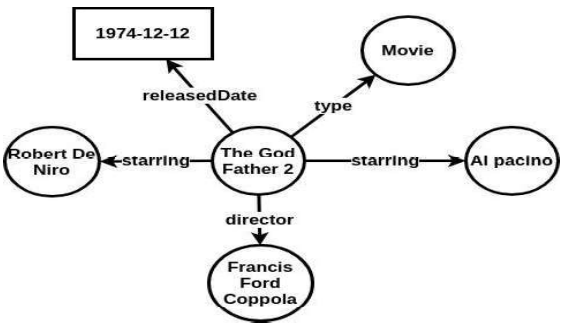


Figure 2.2: Graph representation for the RDF data in figure 2.1

2.2.2 RDFS and OWL

RDF is a data model that describes set of resources. Moreover, the RDF Schema (RDFS) provides a data-modelling vocabulary for RDF data. This vocabulary can be used to specify URIs as being of a specific type (classes, properties and instances), to denote special relationships between resources. The flexibility of the RDF data model allows the representation of both the schema and the instances information in the form of RDF triples.

For example in the graph of figure 2.2, "Movie" is a class, "release_Date" is a property and "12-12-1974" is literal.

Therefore, the vocabularies that are provided by RDF Schema to define classes and properties can be included as "rdfs:Class, rdf:Property (from the RDF namespace), rdfs:subClassOf, rdfs:subPropertyOf, rdfs:domain, and rdfs:range". It may also include properties for documentation, including rdfs:label and rdfs:comment. To go beyond these primitives, the W3c has proposed an Ontology Web Language (OWL).

The W3C Web Ontology Language (OWL) [22] is a Semantic Web language designed to represent rich and complex knowledge about things, groups of things, and relations between things. OWL is a computational logic-based language such that knowledge expressed in OWL can be exploited by computer programs, e.g., to verify the consistency of that knowledge or support some inferences to derive new knowledge from what we already have. OWL documents, known as ontologies, can be published in the World Wide Web and may refer to or be referred from other OWL ontologies [88]. For example, owl allows to express that some property has a cardinality.

2.2.3 The SPARQL Query Language

RDF datasets can be queried using languages such as the SPARQL language [79] where queries are specified as basic graph patterns evaluated against the dataset. A basic graph pattern is a set of triple patterns. A triple pattern is an RDF triple in which the resources and/or the property could be variables. The idea is to match the triples in the SPARQL query with the existing RDF triples and find solutions to the variables. For example, if the user wants to query the data represented in the graph of figure 2.2 to find the director and release date of the movie “The God Father 2”, he should write the following query:

```
SELECT ?x ?y
WHERE {< The_God_Father_2, director, ?x >.
< The_God_Father_2, Release_date, ?y>.}
```

The result of the query for the variable "x" will be "Francis Ford Coppola"

which is the director of the movie and "12-12-1974" is the value of the variable "y", which is the release date for "The God Father 2" movie.

A SPARQL query[86] can be of the following forms:

- SELECT; Returns all, or a subset of, the variables bound in a query pattern match.
- CONSTRUCT; Returns an RDF graph constructed by substituting variables in a set of triple templates.
- ASK; Returns a boolean indicating whether a query pattern matches or not.
- DESCRIBE; Returns an RDF graph that describes the resources found.

2.2.4 Linked data and the Semantic Web

The web of data or linked data was introduced by Tim Berners-Lee [8] to represent the multiple structured and interconnected data sources published on the Web. It builds upon standard Web technologies such as HTTP, RDF and URIs, but rather than using them to serve web pages only for human readers, it extends them to share information in a way that can be read automatically by computers.

The semantic web is a distributed collection of "linked data" on the Web. Just as Web pages are linked via hypertext, the goal of the semantic web is to ultimately link all the available data. In other words, each data source includes links to other data sources, thus forming a huge information space, which can be exploited by machines, and not only by humans.

According to Berners-Lee[7] “The step of putting data on the Web in a form that machines can naturally understand, or converting it to that form is call a Semantic Web – a web of data that can be processed directly or indirectly by machines”.

The Semantic Web is about two things. It is about common formats for integration and combination of data drawn from diverse sources, where on the original Web mainly concentrated on the interchange of documents. It is also about language for recording how the data relates to real-world objects. That allows a user, or a an application, to start off in one dataset and navigate to all the interconnected datasets as if it was a single virtual data space [87].

2.3 Keyword Search

Keyword search is a process that takes some keywords issued by a user and retrieves any document that has all or any of those terms used. Although mostly known

for searching collections of documents, some keyword search approaches have been proposed for relational databases [1, 6, 9, 24, 42, 44], other approaches have been proposed to enable keyword search on XML data [20, 34, 92, 34]. Finally, some approaches have addressed keyword search over RDF datasets [93, 38, 66, 91].

Keyword search is an alternative way used to query an RDF graph, it does not require any knowledge about the syntax of the query language from the user; the process consists of entering a set of keywords into the system. The system finds the best match for each keyword and returns them as a sub-graph containing all the keywords and representing an answer to the query. There are three main challenges for keyword search:

Identifying the matching elements: This consists in finding the elements of the graph that match each keyword in the query. In some cases, the user may enter a keyword for which an exact match can not be found in the dataset. The problem is to identify the equivalent concepts and the close concepts to some keywords in the dataset.

Aggregating the matching elements: After identifying the matching elements in the RDF graph, the problem is to build the final result from these elements and aggregate them into a connected sub-graph representing an answer to the query.

Ranking the final results: As each keyword may have more than one matching element in the dataset, there may be several possible results to the query. The problem is to rank the different results and to find a ranking method capable of determining if there are better results than others.

The above challenges are tackled by the existing keyword search approaches, which either propose a general solution covering all of them or focus on a subset of these challenges. In the rest of this section, I will present each challenge and the existing approaches with their techniques to overcome each of them.

2.3.1 Identifying the matching elements

Identifying the matching elements consists in identifying the elements in the dataset that can be matched with the query keywords. Some of the existing approaches find the exact matches between the query keywords and the dataset; exact matches means that the same query keyword was found in the dataset. Other approaches use some external knowledge to help in identifying the matching elements. In this section, I will present existing approaches that tackle this problem.

The approaches can be classified into three categories according to the type of matching they perform. The first one is node matching; the approaches in this type match the query keywords with the nodes of the RDF graph. The second one is edge matching; the approaches in this type match the query keywords with the nodes and the edges of the RDF graph. The last type is pattern matching, where the query keyword can be matched with a node, an edge or even a sub-graph from the RDF graph.

2.3.1.1 Node Matching

Node Matching is the first category of approaches that match the nodes in the RDF graph with the query keywords. In this section, we will present the different approaches under this category.

Spark [93] is a keyword search approach that takes a set of keywords as an input and returns a ranked list of SPARQL queries as an output. These queries are used to create the final answer to the query keyword. The matching between the query keywords and the elements of the RDF graph in this approach is done according to two ways. (i) The first one is morphological matching, and (ii) the second one is semantic matching. The first kind of matching is to use some string based functions and comparison techniques such as Sub-String and Edit-Distance. These techniques aim to find nodes and edges in the graph that are morphologically similar to the query keyword. Stemming sub-string is a process used to reduce the words into their root form and try to find the matching elements between the query keyword and the nodes and/or edges in the graph. For example, "browse" can be matched with "browsing". Another technique is the Edit-Distance which calculates the minimum number of operations required to transform one string into the other; these operations are deletion, insertion and substitution of a character. For example to convert "cat" into "cut" one operation is needed which is replacing 'a' with 'u'. The second method consists of finding semantically relevant words to map the terms in a graph with the keywords in the query by using some external knowledge. A pre-defined confidence value to determine the mapping quality is assigned to each matching element. This score is between 0 and 1 where the value for a direct matching is higher than that for a semantic matching. At the end of the matching process, each keyword in the query will be associated with a set of matching nodes from the RDF graph.

In the Q2semantic [91] approach, the matching is performed using an external knowledge base. In this approach, all the property names, URIs, resources and literals values are considered as words, and every single word is represented by a term t_i. A document is created for each term t_i in the dataset; this document contains the features that are matched to t_i. These features are extracted from Wikipedia by searching for t_i in it. They are consisted of the titles of the first

articles that appears in the search and the title of another article that redirects to the first article. The matching process starts by comparing the keywords in the query with the terms and the documents associated with each term in the dataset. The search in this approach is an exact match between one of the elements in the document related to the terms of the data set and the query keyword. The matching process provides for each query keyword a set of matching elements.

The previous approaches use an external knowledge base to enrich the matching process and find more keywords to match the query. Other approaches provide only the exact matching between the query keyword and the graph patterns as in [40, 46] but these approaches differ in the way they find the result for the query. One of these approaches is BLINKS (Bi-Level INdexing for Keyword Search)[40], it is an indexing and query processing scheme for ranked keyword search over node-labeled directed graphs. In this approach, the keywords are matched to the nodes in the RDF graph by exact matching, finding the identical keywords in the nodes of RDF graph that match the query keyword by using an index defined by the authors.

In the approach presented by Kargar et al. [46], a system for effective keyword search in a graph with weighted nodes is presented. This system returns a subgraph containing all the keywords in the query entered by the user. In the matching process, for each query keyword k_i the approach finds the set of matching nodes S_{k_i} in the RDF graph that are identical to k_i by using the inverted index, which is used in the search engine to optimize the speed of the query.

2.3.1.2 Node and Edge Matching

The second category of approaches not only consider the nodes but the edges as well. Each query keyword in these approaches will be matched with either a node or an edge of the RDF graph. The matching technique is different from one approach to another, for example the approach presented in [38] builds a bipartite graph from RDF data. A bipartite graph is a graph whose vertices can be divided into two disjoint and independent sets U and V such that every edge connects a vertex in U to one in V. This approach consists of creating an annotated query AQ from the query keyword Q={t1, t2, t3...}, by checking if each term in Q could be matched to an entity, a class, or a relation of the RDF dataset. The mapping is done by using the entity linking [39] and relation paraphrasing [11, 62].

Consider the example given in figure 2.3, representing an RDF graph extracted from DBpedia, and let Q = {scientist, university, locate, United _States} be a query keyword. The annotated query is AQ = {"scientist ": class, "graduate from": relation, "university": class, "locate": relation, "USA": entity }, where "scientist" is matched to the {Scientist} class, "university" is matched to the { University} class, and "USA" is matched to the entities {United_States}. In addition, the relation

"locate" is also matched to the candidate predicate { location}.

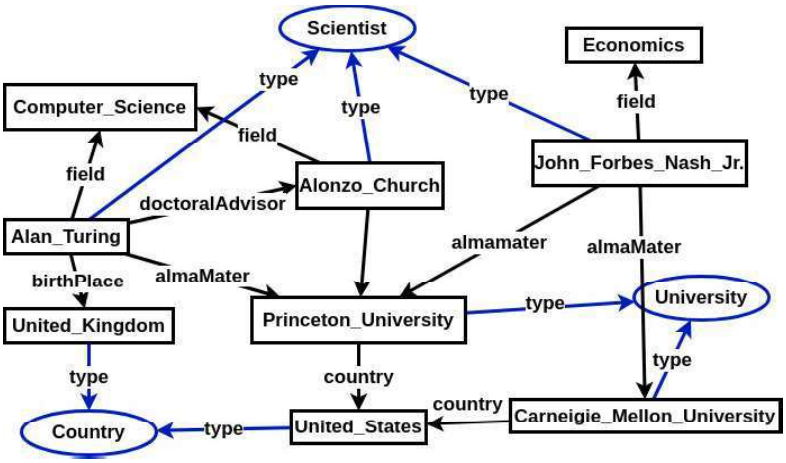


Figure 2.3: An example of RDF graph describing universities and scientists[38]

All the previous approaches provide a subgraph as a result of the query keywords. Another type of keyword search approach provides set of SPARQL queries from the query keyword and then executes this query to provide the final result. The approach presented in Gkirtzou et al. [31] takes a set of keywords as input and translates them into candidate SPARQL queries. In this approach, the matching elements are determined by finding the exact matching between the query keyword and the elements of the RDF graphs. The matching elements can be either nodes or edges.

2.3.1.3 Pattern Matching

The third category of approaches use not only matching between nodes and edges, but also use pattern matching. The keyword in this category will be matched with either a node, or an edge or even a path that is represented as a pattern in the RDF graph. The approaches in this category use external knowledge to find the patterns that match a path in the graph with a query keyword. Another approach presented by Zou et al. [95] proposes a graph data-driven solution to answer a natural language question over an RDF graph. It takes a natural language question as an input and returns a set of sub-graphs that matches the question as an output. The approach consists of two stages, an offline stage and an online one. In the offline stage, a dictionary is created to support the matching process between the natural language question and the terms in the RDF graph. In the online stage, the matching elements are identified, and the possible sub-graph solutions are created.

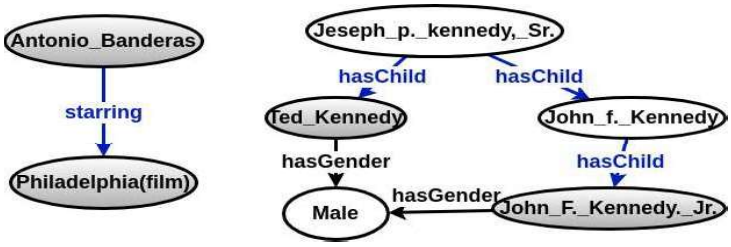


Figure 2.4: Predicate and Predicate Paths that can be mapped to Relation Phrases [95]

In the offline part, the approach relies on some external knowledge to create a paraphrase dictionary. This external knowledge is provided in the form of supporting entity pairs that consist of matching relation to entities; for example, table 2.1 presents the relation phrases and the supporting entity pairs. The first column contains the relation phrase, and the second one contains the supporting entity pairs where the pairs in column two can be connected using the relation in column one; for example, <Antonio_Banderas> is connected to <Philadelphia(film)> by the relation "Play in".

Relation Phrase	Supporting Entity Pairs
"play in"	$(\langle Antonio_Banderas\rangle, \langle Philadelphia\,(film)\rangle)$, $(\langle Julia_Roberts\rangle, \langle Runway_Bride\rangle)$ ...
"uncle of"	$(\langle Ted_Kennedy\rangle, \langle John_F._Kennedy_Jr.\rangle)$, $(\langle Peter_Corr\rangle, \langle Jim_Corr\rangle)$ ...

Table 2.1: Relation Phrase and Supporting Entity Pairs [95]

The paraphrase dictionary contains the relation phrase and the predicate or path equivalence to this phrase and a confidence probability to reflect the matching score between them. For example, Table 2.2 shows an example of paraphrase dictionary created by using the entity pairs in the table 2.1 and the graphs in figure 2.4. The dictionary shows that the relation "uncle of" is equivalent to the path highlighted in blue in the graph of figure 2.4.

In the online phase, each keyword in the natural language question will be matched to a term in the RDF graph exactly or by using the paraphrase dictionary created in the offline phase. For example, if "play in" is one of the keywords in the natural language question, it will be matched to the predicate "director" according to the paraphrase dictionary in table 2.2. As an output of the online phase, for each keyword in the question could be matched with more than one matching element.

In the approach presented by Ouksili et al. [66], the equivalence between

Relation Phrase	predicates or Predicate Paths	confidence probability
“play in”	<starring>	0.9
“play in”	<director>	0.5
“uncle of”	<hasChild> <hasChild> <hasChild>	0.8

Table 2.2: Paraphrase Dictionary [95]

keywords and paths is also used. This approach takes a set of keywords as an input query and returns a set of sub-graphs as a result to the query. External knowledge expressed as patterns are used to match the query keyword with patterns from the RDF graph. The pattern represents the equivalences between a property and a path. For example, the property “brother” is equivalent to the graph shown in figure 2.5, which can therefore be selected as a matching element for the keyword "brother".

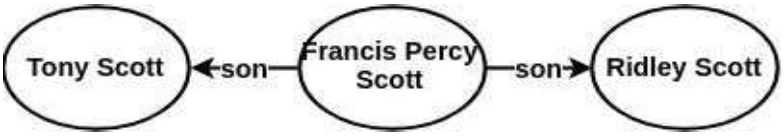


Figure 2.5: Example of sub-graph which matches the Brother keyword

In this approach, the matching process is based on the exact matching between the keywords and the terms of the RDF graph; if the keyword is not found in some graph elements, then the patterns are used to find a sub-graph that is equivalent to the keyword. Finally, each query keyword can be matched to more than one candidate sub-graph.

In all the presented approaches the matching elements between the keyword query and the sub-graphs are defined in two ways. The first one is using the exact match, which consists of making comparisons over the letters of the keywords. The second one consists of finding keywords in the RDF graph that are close in meaning to the keywords in the query using some external knowledge. Most of the approaches that uses the exact matches use the indexation to optimize the search and find the keywords in the fastest way. But using exact matching alone is not enough; The problem is that the user could use a keyword which can not be found exactly as it is in the graph but we can still find some elements that are close in meaning to this keyword. To overcome this problem, some approaches have been

proposed based on using some external knowledge to bridge the terminological gap between the query keywords and the terms in the graph. The limitations of these approaches is that most of them rely on some external knowledge which is domain specific which makes these approaches unsuitable for any RDF dataset.

2.3.2 Aggregating the matching elements

After finding the matching elements, the goal is to aggregate these elements and connect them to create a sub-graph corresponding to the query keyword. This is done by finding the best paths that connect the matching elements. Determining what is the best path is a challenge. The problem is to find a way to assess the path according to different criteria that reflect its meaningfulness according to the user.

In keyword search approaches there has been several ways to aggregate a given set of matching elements in order to produce the smallest sub-graph containing them all, and this sub-graph represents an answer to the query. We can divide these approaches into four categories according to the graph algorithm they use. The first one is using the shortest path, the approaches in this category find the shortest path between the matching elements to create the final result. The second category is summarization; in this category, the approaches build a summarized graph starting from the main graph in order to find the best result to connect all the matching elements. The third category is backpropagation; this algorithm starts from different nodes and propagates backward to converge in one node to find the best solution. The final category is the ad-hoc aggregation method where each approach in this category has its own method in creating the result. In the next section, the approaches are presented according to the four categories.

2.3.2.1 Aggregating Matching Elements by using the Shortest Path

The first category is to build the sub-graph result based on the shortest path between two nodes. The metric to compute the shortest path differs from one approach to another; it can be based on the number of edges that connected the two nodes or on the sum of a weight associated to the edge. The approach presented in [46] weighted the edges and find the shortest weighted path between the matching elements. The weight of the edge depends on the degree of the two nodes that are connected by this edge. The process consists of selecting a node in the graph and finding the shortest path from this node to all the other nodes containing one of the query keywords. Since each node in the graph can be a potential root for an answer tree, the system starts with a random node n_i in the graph. Finally, it creates a tree which root is the node n_i and all its leafs containing the matching elements. The previous steps are repeated by choosing another root

node n_j to create another possible result. After finding all the possible result trees, a score is calculated by computing the sum of the weight of the edges in the tree. The tree having the highest score is the best answer to the query keyword.

This method is also used by the approach presented in [66]. Each query keyword corresponds to a set of matching elements; The cartesian product is performed to extract all the possible combinations between the matching elements. For each combination, a sub-graph is created by randomly selecting one of the marching elements and connecting all the shortest paths from this element to all the other elements in the combination. Finally, a set of sub-graph results is returned to the user.

The approach presented in [93] uses the minimum spanning tree algorithm that is based on the shortest path. This approach constructs SPARQL queries from the list of matching elements of the query keywords. Since there is a list of matching elements for each keyword in the query, all the possible combinations are derived. In each combination there is one matching element that corresponds to one keyword query. For each combination, the approach finds the sub-graph connecting all the terms in the combination to create the query graph by using an algorithm called the minimum spanning tree [55]. This algorithm constructs a tree from a graph by connecting the nodes without any cycle and with the minimum possible edge weight. In this approach, the authors consider that the weight of the edges equals one. After building the query graph, the SPARQL query is created. Since SPARQL is a graph pattern-based query language, it is straightforward to convert the query graph into a corresponding SPARQL query.

2.3.2.2 Aggregating Matching Elements by using Backward Propagation

Backward propagation starts form the matching nodes, which are the nodes that can be matched to the query keywords, and propagates backward until all the nodes converge to a root node and all the nodes are connected. Q2semantic [91] uses backward propagation and it creates a clustered graph summarizing the original graph, then uses the propagation method on the summarized graph. This graph is called the RACK graph; it consists of R-Edge, A-Edge, C-Node, and K-Node obtained from the original RDF graph through the following mappings:

Mapping instances: Every instance of the RDF graph is mapped to a C-Node labeled by the concept name that the instance belongs to.

Mapping attributes: Every attribute value is mapped to a K-Node labeled by the literal value.

Mapping relations: There are two types of relations, the first one is the relations connecting two instances that are mapped to an R-Edge and the second one is a relation between an instance and literal that is mapped to an A-Edge.

The clustered RACK graph is obtained by applying the following rules:

Rule 1: Two C-Nodes are merge into one if they have the same label.

Rule 2: Two R-Edges are merge into one if they have the same label and connect the same pair of C-Nodes.

Rule 3: Two A-Edges are merge into one if they have the same label and are connected to the same C-Node.

Rule 4: Two K-Nodes are merge into one if they are connected to the same A-Edge. The resulting node inherits the labels of both K-Nodes.

The final query is constructed from the clustered RACK graph by creating a thread for each keyword and do a traversal from the K-nodes until all threads converge to the same node. When all the keyword threads converge to the same node, then a final solution is found.

Another approach that also uses backward propagation is presented in [40], but in this approach, the authors first decompose the data graph into blocks. The decomposition of the graph is based on node-based since the number of the nodes that separate the different partitions, which are denoted by separators, are smaller in the node-based partition; since the nodes are the separators then only one element will be duplicated in the two separated graphs unlike the edge-based partitioning which requires the duplication of 3 elements, the edge separator and the two nodes connected to this edge. The authors also mentioned that the node-based partition eases the implementation. The process of creating the resulting sub-graph starts by traversing the RDF graph from the blocks containing the query keywords and starts expanding backwards until the propagation converges to the same node. The shortest distance score SumDist for a node is defined as the sum of the shortest distances from this node to all the nodes that contain the query keywords. This SumDist is calculated for all the nodes, and we check if this score is less than a given threshold, then the sub-graph created by connecting this node to all the nodes that contain the query keywords is a candidate answer. A list of ranked sub-graphs is returned as an output.

2.3.2.3 Aggregating Matching Elements by using a Summarized Graph

Similarly to the Q2semantic approach [91] that summarizes the graph before building the final sub-graph result, the approach proposed in [31] creates an augmented summary graph. This graph is a combination of an aggregated representation of the RDF dataset. All RDF entities from the RDF data graph that have the same type are represented by a vertex labeled with the name of this class. Let r_1 and r_2 be two resources in the RDF graph and let R_1 and R_2 be the aggregated vertex in the augmented summary graph that represent the resources r_1 and r_2 respectively. If there is an edge between r_1 and r_2 in the RDF graph, then this edge need to be added in the augmented summary graph.

After creating this graph, a query graph pattern is extracted by calculating the shortest paths between every pair of matching elements and then all paths are combined together into one connected query pattern graph. Finally, the query pattern graph is transated into a SPARQL query.

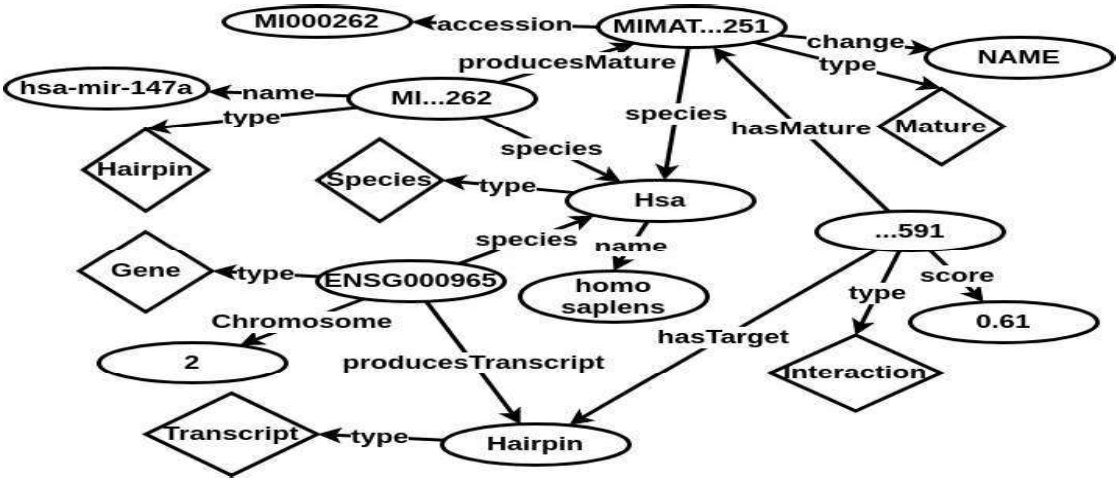


Figure 2.6: Example of an RDF data graph describing biological data [31]

Let us consider the graph in figure 2.6; and let k={ "MI000262", "name" , "hasTarget"} be the query keyword. There are three matching elements for the keyword "name", one for "MI000262" and one for "hasTarget". There are three possible combinations; let us take the combination having the keyword "name" as a literal. Figure 2.7 shows the augmented summary graph.

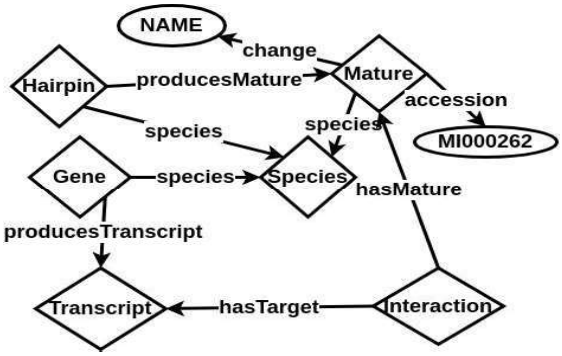


Figure 2.7: Augmented summary graph [31]

To extract the query pattern graph from the augmented summary graph, for each pair of matching elements the shortest path is calculated, then all the shortest paths are combined to create one connected subgraph. The final step in this process

is to translate the query pattern graph into a SPARQL query.

The final SPARQL query is obtained by translating the query pattern of graph

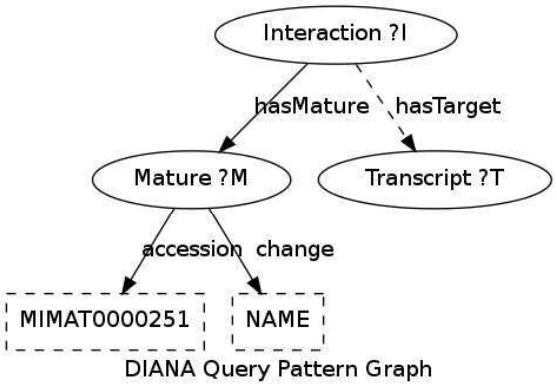


Figure 2.8: query pattern graph [31]

2.8, and the corresponding SPARQL query is given below:

```
SELECT ?I ?M ?T WHERE
 {?I a diana:Interaction. ?M a diana:Mature. ?T a diana:Transcript.
 ?I diana:hasMature ?M. ?I diana:hasTarget ?T.
 ?M diana:accession ‘‘MIMAT0000251’’. ?M diana:change ‘‘NAME’’.}
```

2.3.2.4 Ad-hoc aggregation methods

Beside summarization based and shortest path based, there are other methods presented by different approaches and can be used to connect the matching elements and compute the final result. One of these approaches is to use bipartite graph as in [38], the process consists of classifying the matching keywords into two sets. The first one contains vertex pairs for all the possible combinations of classes and literals, and the second one contains the possible relations (edges) between the items in the first set. Then all the possible combinations are derived. For each combination, a SPARQL query is created.

Another method that can be used consists of translating the natural language question into a graph as in the approach presented in [95]. The aggregation in this approach consists of two processes: the first one is question understanding; its goal is to translate the natural language question into a dependency tree. The second process is query evaluation, which finds a sub-graph in the RDF data graph that can be matched with the dependency tree created in the previous process. The translation of the natural language question into a tree is done by using the Stanford Parser [21], then the relations are extracted from this tree, and a

comparison is done with the paraphrase dictionary in order to match the relations in the tree with the phrases and the pairs in the dictionary. Ambiguity is allowed in this matching, i.e., one element in the dependency tree may be matched to more than one element in the RDF graph. Finally, all the sub-graphs that match the dependency tree are extracted from the RDF graph and ranked according to the confidence probability in the paraphrase dictionary. A ranked list of sub-graphs is returned to the user as an output.

2.3.3 Ranking Answers Subgraphs

Since most of the approaches provide more than one answer to the query keyword, there is a need of a ranking method to rank these results. There are different ranking methods proposed in the approaches of keyword query. Some of them realise the probability formulas, and others use a method based on the matching element that reflects how the matching element is close to the query keyword. In this section, I will present how the different approaches deal with calculating the rank for each answer in the resulting set of answers.

In SPARK [93] after finding all the sub-graphs for all the combinations, a ranking method is applied in order to rank the results. This method is based on the matching score assigned to the terms during the matching process. The ranking method consists in representing the probability of constructing formal query from the given query keyword; it is treated as a conditional probability event; i.e what is the probability of constricting sparql query Q given the query keywords.

In the approach presented in [95] the results are ranked based on the probability. In this approach, a paraphrase dictionary D is used to compute the equivalence paths between the relation phrases and the paths connecting the supporting entity pairs. Each relation phrase in the dictionary D is associated with a confidence probability, which is the probability of mapping relation phrase to predicate paths. During the ranking phase, this confidence probability is used to calculate the final rank for the created result by adding the confidence probability for each matching element.

Other approaches combine more than one method to rank the results. In Q2Semantic [91], three methods are used for ranking the sub-graph results. Q2Semantic consists of creating query graphs. The ranking is based on these graphs. The first method is the query length, which consists in calculating the number of edges in the query graph. The second method is calculated based on a matching relevance score; this score is associated with the matching element during the matching process. In this method, the query graph with the minimal number of nodes and with the higher matching relevance score is the higher-ranked query. The third one is a score that reflects the importance of edges and nodes. This is calculated by taking into account the number of nodes and edges that are not related to the matching

elements and are added to the summarized RACK graph.

Another approach that uses more than one method to rank the final results is presented in [66]. In this approach, all the results are ranked according to a ranking function that combines two criteria, the compactness relevance, and the mapping relevance. The compactness relevance is the relevance between the size of the matching elements and the final sub-graph result; it is used to check the size of the intermediate part of the final sub-graph that is added during the construction of the result, the rank of the final result increase as the size of the intermediate part decreases. The mapping relevance consists of giving a weight for each matching element; this weight represents the relevance of the matching element with the query keyword.

2.3.4 Discussion and open issues

We have presented some existing works addressing keyword search in RDF datasets. Some of these works deal with one of the challenges which are matching the keyword query with the elements of the RDF graph, aggregating the matching elements to build an answer to the query and finally ranking the final set of answers to the query; other approached deal with all of them. The approaches in this section are presented according to these challenges.

For matching the query keywords with the elements of the dataset, Spark [93] uses morphological mapping and semantic mapping, but the user might write a query keyword which is not in the ontology, or it is not equivalent to a predicate path. Another approach [38] matches the query keyword to entity, class, or relation and then deduces the clique with the largest coverage, but this study did not take into consideration that the user may enter a keyword different from the terms in the graph. Other approaches used external knowledge as in [91] and [95] but according to [91] a large-sized memory is needed to handle the documents for each keyword, and it is expensive to extract all the related elements, and the target element may be not founded in these documents. Moreover, the approach in [95] uses the supporting entity pairs but these pairs are domain specific and they are not available for all datasets according to [62].

For the second challenge, which is aggregating the matching elements, the approach in [93] extracts the relation from the ontology and proposes to build query graphs by uses minimum spanning tree and convert the graph into SPARQL queries. We can use the ontology to generate some knowledge from it after doing some kind of inferences. However not all datasets have their ontology, and not all relations can be extracted from it. There are approaches that transfer the data graph into a summarized graph and use them to answer the query keyword [91, 40, 57], but creating a summarized graph leads to a lack of information and loss lot of relations may be important to connect the matching elements. Other approaches

[31] use some ad-hoc methods to create query pattern graph and then generate the SPARQL query based on this graph, but this approach needs good knowledge about the initial graph, the ontology, and all the relations between the nodes. The last approach to discuss is [38], consists of creating a bipartite graph by classifying the keywords into two sets, classes and literals (vertices) and relations (edges); the problem with this approach is that the connected keywords may not be in the query keyword (use only query keyword to create the edges set).

There are not many studies dealing with the last challenge, which is ranking the final set of answers. In the approach presented in [95] the authors calculate the confidence probability of mapping relation phrase to predicate path. While [38] and [91] use the edge weight and the query length, they consider that the smallest the query the better the rank but this might not be true since the query with a smaller length is not necessary to be the best one. Other approaches use probability to compute the ranking, for example in [93], the probability of constructing a formal query from the given query keyword is calculated.

Table 2.3 summarizes the keyword search algorithms presented in this section. As we can observe from the table, some of them match the keywords with nodes and others match it with nodes and edges. We can also see that some approaches use exact matching and others use an external knowledge base consists of semantic similarity between concepts under a specific domain. Usually, the input to all the approaches is a set of keywords, but the output differs from one approach to another, some of them return a set of sub-graphs as an output, others return just one sub-graph and there is also the approaches that return a SPARQL query as an output.

One of the limitations of the existing approaches is that they do not consider the terminological gap between the query keyword and the RDF elements. Suppose the user enters a keyword that can not be matched to any component of the graph, but there exist one or more elements that are similar to this query keyword, none of the studied approaches can handle this problem. In that case, some of them are using specific domain external knowledge relations. Still, those are not enough since they are not general relations and do not reflect all the possible semantic relations for all the elements of the RDF graph. Another limitation of the existing approaches which rely on shortest path, is that they only consider the length of the path and they do not consider the semantics carried by the edges of the path.

Approaches	Matching elements	Type of Matching	Input	Output	Use of External knowledge
SPARK [93]	Nodes	Exact and approximate	Keyword Query	A Set of sub-graphs	Yes WordNet
Han et al. [38]	Nodes and Edges	Exact	Keyword Query	SPARQL query	No
Q2Senabtic [91]	Nodes	Exact and approximate	Keyword Query	Sub-graph	Yes Wikipedia
Zou et al. [95]	Nodes ,Edges and Patterns	Exact	Question	Set of sub-graphs	No
BLINKS [40]	Nodes	Exact	Keyword Query	Sub-graph	No
Gkirtzou et al. [31]	Nodes and Edges	Exact	Keyword Query	SPARQL query	No
Kargar et al. [46]	Nodes	Exact	Keyword Query	Sub-graph	No
Ouksili et al. [66]	Nodes , Edges and Patterns	Exact	Keyword Query	Set of sub-graphs	Yes Patterns

Table 2.3: Summary of some Keyword Search Approaches

2.4 Summarization of an RDF Graph

Summary can guide the user in exploring RDF dataset since it can provides us with a concise representation containing the most relevant information in the graph. Summarizing RDF graphs has been the topic of several research works [18, 58]. The main idea consists of selecting the most relevant elements and connect them to build the graph summary. These elements differ from one approach to another, it can be a node, an edge or even a patterns in a graph. One challenge of graph summarization consists in identifying the relevant elements in the graph; we need to find the best criteria to assess the nodes and patterns to identify the relevant elements that will represent the graph. Another challenge is to aggregate these relevant elements in order to build a summary; we need to find a method to connect the relevant elements without introducing to much new elements to the summary.

The problem of summarization has been tackled according to two different perspectives, the first one is based on the node level metric and the second one is based on graph analysis techniques. Summarization approaches which rely on node level metric are based on identifying the most relevant nodes to be extracted from the main graph and connect them to create the summarized graph. The relevant elements are identified based on different metrics calculated based on the structure of the graph. These metrics can be statistical information, node centrality calculation, or identifying equivalent nodes based on predefined equivalence relation. The second perspective of summarization is based on graph analysis such as identifying the different communities in a graph or decomposing the graph according to the regions with the highest density.

In this section, we will present the existing approaches to summarize RDF graphs based on the two different categories, the **Graph summarization based on node level metrics** and **Graph summarization based on graph analysis techniques**.

2.4.1 Summarization based on Node Level Metrics

Summarization based on node level metrics depends on assessing the nodes in the graph using different metrics. Building the summarized graph differs from one approach to another. According to the approaches, there are four different categories for summarizing RDF graph based on node level metrics; the first one is the quotient based graph, where an equivalence relation is defined over the nodes and an equivalence class of nodes is represented by one node in the summarized graph. The centrality is the second category in the node level metrics. The summarization in this category is based on identifying the most central elements in the RDF graph and building the summary starting from them. The third category is the statistical one, where the summarized graph is constructed based

on statistical information collected from the RDF graph. The last category is composed of hybrid approaches, which summarizes graphs by using two or more approaches from the previous three categories. In the next section, I will present each category by giving the definition and the approaches that use it.

2.4.1.1 Quotient Based Summarization

In this category, the summarized graphs are generated according to the concept of quotient graph. This concept is based on aggregating the nodes that have the same features into one single node, then connect the aggregated nodes together to create the summarized graph. Not all the approaches use the same technique to compare the nodes; some of them compare the nodes according to their labels, other according to the incoming and outgoing edges. In this section, I will present the definition of the quotient graph and the different approaches that use it.

Quotient Graph: Given a data graph G(V,E) with V nodes and E edges, let us consider an equivalence relation between the node of G denote $\equiv$, the quotient graph G by $\equiv$, denoted G $/\equiv$, is a graph such that:

1. G $/\equiv$ contains a node for each set of equivalent nodes based on the equivalence relation $\equiv$ (thus, for each set of equivalent G nodes)
2. for each edge $n_1 \xrightarrow{a} n_2$ in G, there is an edge $m_1 \xrightarrow{a} m_2$ in G $/\equiv$, where m_1 , m_2 are the quotient nodes corresponding to the equivalence nodes of n_1 , n_1 respectively.

This equivalence relation can be define differently according to the methodology, for example if two nodes have the same type, then these two nodes are equivalent under the type equivalence relation; then the quotient of G by $\equiv$, denoted G $/\equiv$, is the graph such that:

1. the nodes G $/\equiv$ are the equivalence classes of V produced by $\equiv$.
2. for each edge $n_1 \xrightarrow{a} n_2$ in G, there is an edge $m_1 \xrightarrow{a} m_2$ in G $/\equiv$, where m_1 , m_2 are the quotient nodes corresponding to the equivalence classes of n_1 , n_1 respectively.

The approaches in [47, 19] presented an adaptive structural summary for graph structured data. One example of a structured data is XML, where the data can be represented by a tree graph. This approach summarizes the tree graph into an index graph to reduce the execution time of the query and have the same result as if it is applied on the tree and on the index graph. The algorithm for building

the graph is based on the quotient graph. The equivalence relation between the nodes is based on the concepts of node path and label path. A definition for node path and label path is presented below:

Definition 1. A node path in the data graph G is a sequence of nodes, $n_1 n_2 \cdots n_p$, such that there is an edge between nodes n_i and n_{i+1} , for $1 \leq i \leq p$ -1.

Definition 2. A label path is a sequence of labels $l_1 l_2 \cdots l_p$.

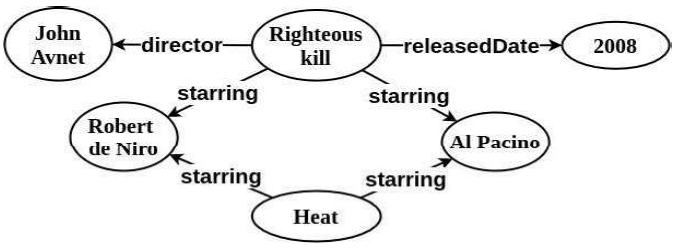


Figure 2.9: An RDF graph describing movies

For example let us consider the graph in figure 2.9, from this graph we can observe that "John Avnet - Righteous kill - 2008" is a node path since there is an edge between the node "John Avnet" and the node "Righteous kill" and also another edge between the node "Righteous kill" and the node "2008". From the same graph, we can also observe that "director - releasedDate" is a label path.

A node path matches a label path if label(n_i) = l_i, for $1 \leq i \leq p$. A label path, $l_1 l_2 \cdots l_p$ matches a node n if there is some node path ending in node n that matches $l_1 l_2 \cdots l_p$. Two nodes n_1 and n_2, are equivalent if and only if all their label paths are the same. For example, let us take the graph in figure 2.10, the nodes 7 and 10 are equivalent, while nodes 7 and 9 are not because node 7 has a parent labeled "actor", while node 9 does not have any parent labeled "actor". The construction of the index graph consists of aggregating the equivalent nodes into one single node and use the label path to connect the aggregated nodes.

Some of the quotient based summarization approaches are dedicated to non RDF data such as XML as in the approach presented in [47, 19]. Other quotient based summarization approaches dedicated to RDF data such as in the approaches presented in [15, 36, 16, 17], they summarize the RDF graph by using different equivalence relations over the nodes. These equivalent relations are presented below.

Let G be an RDF graph and let p1 and p2 be two data properties G:

Definition 3. p1, p2 $\in$ G are source-related if and only if either: (i) a data node in G is the subject of both p1 and p2, or (ii) G contains a data node that is the subject of p1 and a data property p3, and p3 and p2 being source-related.

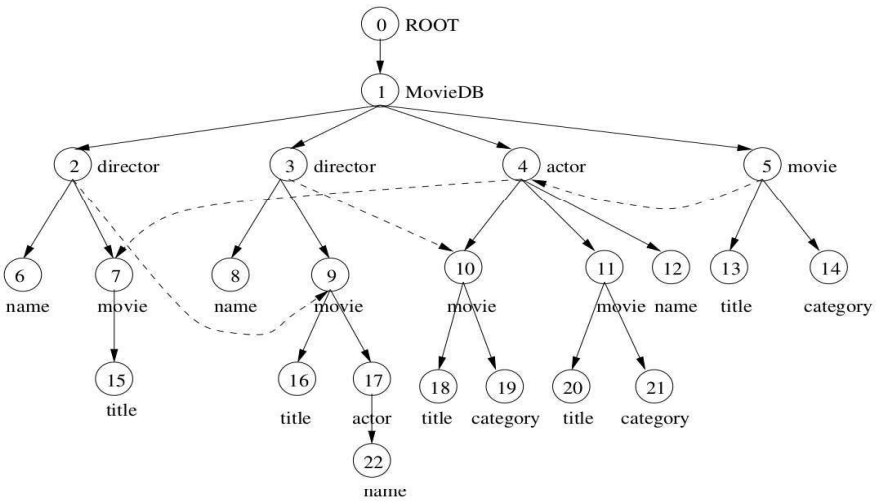


Figure 2.10: An Example of Graph-Structured Data [19]

Definition 4. p1 , p2 $\in$ G are target-related if and only if either: (i) a data node in G is the object of both p1 and p2, or (ii) G contains a data node that is the object of p1 and a data property p3, and p3 and p2 being target-related.

A maximal set of data properties in G which are pairwise source-related is called a source property clique. Similarly, a maximal set of data properties in G which are pairwise target-related is called a target property clique.

In these approaches, the authors define two equivalence relations, weak equivalence and strong equivalence. They are based on the subject which is also defined as the source of the data properties and the object which is the target of the data properties. Consider two data nodes of G n1 and n2: (i) n1 and n2 are **strongly equivalent**, denoted n1 $\equiv_S$ n2 , if they have the same source and target cliques; (ii) n1 and n2 are **weakly equivalent**, denoted n1 $\equiv_S$ n2 , if they have the same non-empty source or non-empty target clique.

The definitions of weak and strong equivalence are used to define four novel summaries (weak summary - strong summary- typed weak summary and typed strong summary) based on property clique which generalizes property co-occurrence.

The weak summary [36] of a data graph G, denoted G/W, is its quotient graph with respect to the weak equivalence relation $\equiv_W$. The process of weak summarization starts by creating a node for each set of nodes having the same source clique or target clique.

The second step is to add the edges to the summarized graph as follows; for each edge $n_1 \xrightarrow{a} n_2$ in G, there is an edge $m_1 \xrightarrow{a} m_2$, where m_1 and m_2 are the quotient nodes corresponding to the equivalence classes of n_1 and n_1 respectively.

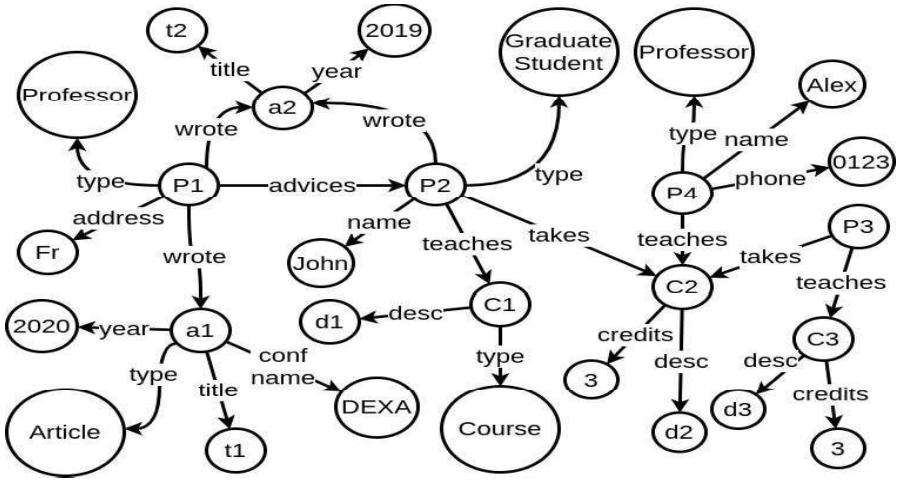


Figure 2.11: Example of an RDF Graph describing information about Students, Courses, Articles and Professors

Let us consider the RDF graph in figure 2.11, this graph describes information about the relations between students, courses, articles and professors. This graph will be used each time it is possible as an input for the different presented summarization approached to illustrate and compare their results. Figure 2.12 shows the summarized graph created by using the weak summary method. As we can observe from the resulting graph, the node "N1" has two different types, while in the graph presented in 2.11 there is no node with more than one type. We can deduce that some links have been added to the summary which were not in the initial graph, and such a links are not necessarily accurate .

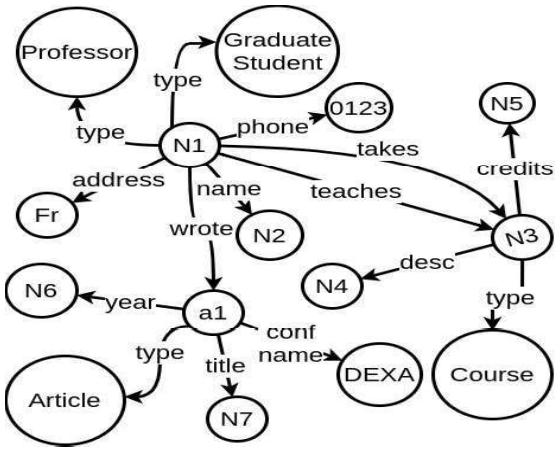


Figure 2.12: Weak Summary of the RDF graph in Figure 2.11

In RDF data some nodes has type and are denoted by typed nodes and other are

not and are denoted by data nodes. The typed weak summary [17] distinguishes between typed nodes and data nodes. For example node "P1" in the graph of figure 2.11 is a typed node since it has a type which is "Professor". In this method, the node types are more important when deciding whether nodes are equivalent or not. Let us consider the graph in figure 2.11 and build the typed weak summary of this graph. First, the nodes are aggregated according to their type. The type of node "p2" is graduate student, then node "p2" and its type will be added to the summarized graph. The other data nodes are aggregated according to the source clique and target clique similarly to the weak summary. The last step which is adding the edges to the summarized graph is based on the same method used in the weak summary definition. The resulting typed weak summary graph in shown in the figure 2.13.

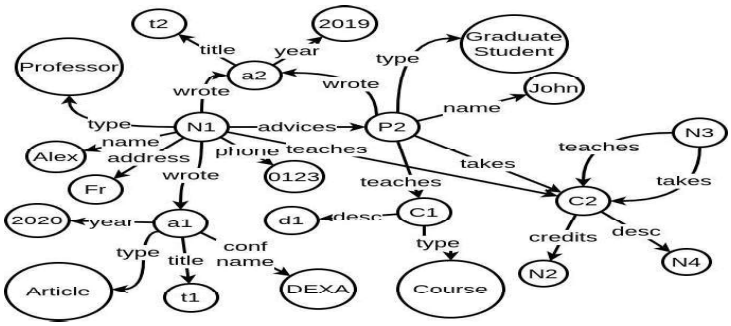


Figure 2.13: Typed Weak Summary of the RDF graph in Figure 2.11

The strong summary [36] of a data graph G, denoted G/S, is its quotient graph with respect to the strong equivalence relation $\equiv_S$. In the weak summary, the nodes aggregated if they either have the same source clique or the same target cliques. In the strong summary, the nodes are aggregated if they have both the source and the target cliques. The same procedure as for the weak summarization are used but the way of aggregating the nodes is different. The strong summary of the graph of figure 2.11 is presented in figure 2.14.

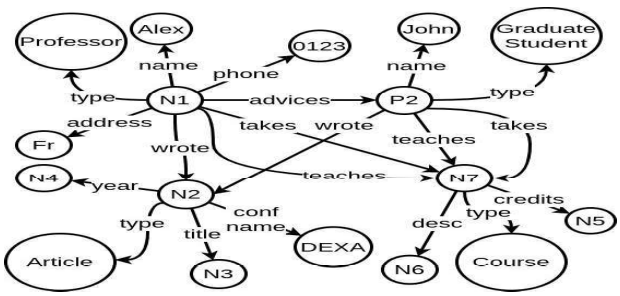


Figure 2.14: Strong Summary of the RDF graph in Figure 2.11

Type strong summary [17] consists of aggregating the nodes according to their type, as in the type weak summary. The type of node p1 is Professor and the type of node p2 is Graduated Student, then both nodes p1 and p2 and their types will be added to the summarized graph. In order to build the type strong summary, we aggregate the nodes based on the source and the target cliques this is done by using the same method as the one used to build the strong summary. The edges are added to the type strong summary according to the same method presented in the strong summary definition. The resulting type strong summary graph shown in figure 2.15.

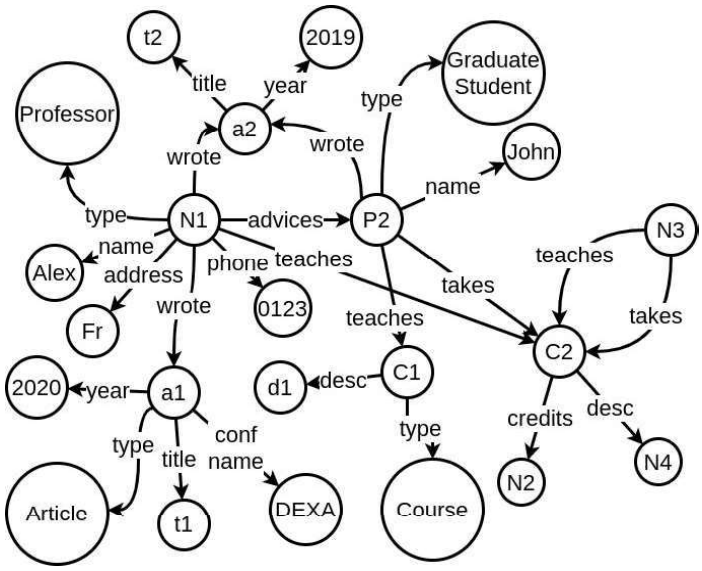


Figure 2.15: Typed Strong Summary of the RDF graph in Figure 2.11

In [32] the authors summarize a dynamic data graph. The approach is similar to [15, 36, 16, 17] but in an incremental way. In [15, 36, 16, 17] the algorithm traverses the graph, computes all the cliques and then performs a graph traverse to represent nodes according to their cliques, but in [32] the incremental algorithm the cliques are build as the graph is traversed; each time we traverse an RDF triple, the cliques will be updated. Simultaneously the summarization graph is build based on these cliques and it is continuously updated.

The authors in [35] summarize big graphs and build the RDFQuotient summaries in parallel. They present four algorithms for the parallel computation of strong, weak, type strong and type weak summaries. The main idea of the algorithm consists of distributing the nodes of the main graph into M machines to compute the summary on each machine alone, taking into consideration the classes and properties which must be preserved by summarization; this step is done by allowing the communication between the machines during the computation of the

summary. At the end, the summarized graph is created.

2.4.1.2 Centrality based Summarization

The second category in the structure-based summary is the centrality. The approaches in this category are based on identifying the most important nodes by using different measures of centrality. Centrality is define as the more central a node, the closer it is to all other nodes. Calculating centrality differs from one approach to another, but all approaches have the same intuition, which is assessing all the nodes and identify the most important ones. After finding the most important nodes, the summary is build by connecting these nodes, the method used for connecting the relevant nodes is different from one approach to another. In this category, not all the nodes are represented in the summarized graph; for example if the most central nodes represents one topic, then the summary will be consists of only this topic and the other topics will not be represented at all.

RDFDigest [82, 83] is defined for RDF graphs and uses the schema information such as the domain and range of a property to enrich the graph with the implicit data such as the type of the resources that use this property. After enriching the graph, the most important nodes are identified by calculating the relevance score for each node. This score depends on the centrality of the node and the centrality of their directly connected neighbors. The relevant nodes are then selected to build the summarized graph. The connection is done by using a score to assess the coverage of a path connecting two distinct relevant nodes. This score is based on the relevant score for each node in the path and also the length of this path.

There are other approaches [68, 84, 81] that use four different methods to calculate the importance of the nodes in the graph.

- **Betweenness centrality:** quantifies the number of times a node acts as a bridge along the shortest path between two other nodes. Assume we compute the shortest paths for every pair of nodes in a connected graph. The betweenness centrality for any node n_i is the number of the shortest paths that pass through the node for all the pairs in the graph.

- **Bridging centrality** is the product of betweenness centrality of a node and the bridging coefficient of the node. The bridging coefficient reflects how well the node is located between high degree nodes. The bridging coefficient of a node n is defined as follows:

  $BC(n) = \sum_{i \in N(n)} \frac{d(n)^{-1}}{1/d(i)}$

  where d(n) is the degree of node n, and N(n) is the set of neighbors of node n.

- **Degree centrality** is the number of edges connected to this node.

- **Harmonic Centrality** also known as valued centrality, it is a variant of closeness centrality of node u and it is equals to $\frac{1}{\sum_{i=1}^{n} d(u,v_i)}$ where n is the total number of nodes in the graph and v_i is a node where u $\neq v_i$ and d(u,v_i) is the distance from node u to node v_i.

After identifying the most important nodes based on these notations of centrality, the approach deal with the problem of connecting the important nodes as a Steiner tree problem [37]. In this approach, the user can parameterize the process by specifying the number of important nodes to be selected in the summarized graph.

Another way of building the summarized graph consists in calculating the importance of the node by combining more than one measure as in the approach presented in [73]. The authors first identify the number of important nodes to be selected by using some parameters which can be the either the proportion of the number of node in the initial graph or a threshold identified by the user. The most important nodes are selected based on the relevance score calculated according to each node's centrality. This score is based on the shortest paths and can be defined as the reciprocal of the sum of the length of the shortest paths between the node and all other nodes in the graph. This score is knows as closeness centrality of a node. The closeness of node v is equal to $\frac{N}{\sum d(u,v)}$ where d(u,v) is the distance between the nodes u and v and N is the total number of nodes in the graph. Finally the summarized graph is build by connecting the selected important nodes. The connection is based on an algorithm that finds the best path between two nodes according to the relevance and the degree score of the nodes in this path.

Another method of building the summarized graph can be done by using the graph patterns as in the approach presented in [94]. The approach consists of three main steps: the first one is to build a binary matrix mapper that creates a matrix for the graph based on each node and the properties that are connected to this node. The second one is the graph pattern identification which identifies the patterns according to the constructed matrix and the last one is the construction the graph summary based on the extracted patterns. The graph in this approach is not an RDF graph. The three steps are detailed below.

Binary Matrix Mapper In this first step, the graph is transformed into a binary matrix where the rows represent the nodes, and the columns represent the edges. For each property two columns are created, the first one captures the subject of the property and the second one (reverse-property) captures the object. Let us consider two nodes i and j belong to graph G, the matrix of graph G is defined as follows:

$$D(i,j) = \begin{cases} 1 & iff\ the\ node\ i\ (has\ j-type)\ or\ (is\ j-property's\ domain/range) \\ & or\ (j-th\ attribute's\ domin) \\ 0 & otherwise \end{cases}$$

Graph Pattern Identification

The binary matrix created in the previous step is used to create a set of k patterns that best describe the input data. The quality of the patterns is measured internally with some cost functions. The PaNDa+ algorithm [59] is used to extract relevant patterns from the binary dataset resulting from the transformation of the original RDF graph.

Constructing the summary graph

The final step is to construct the summary by using the extracted patterns. For each pattern in step 2, a node labeled by a URI is generated and an attribute representing the number of instances for this pattern is added. The graph of figure 2.16 shows the summarized graph after using the previous approach on the graph presented in figure 2.11 .

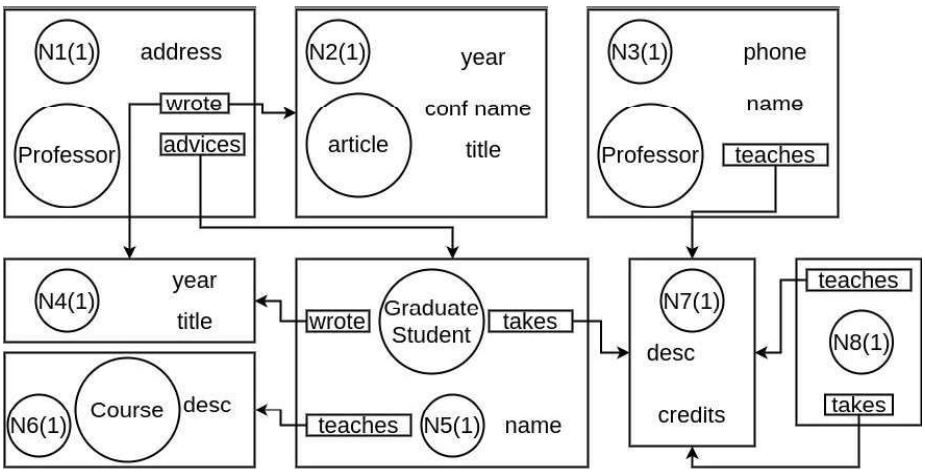


Figure 2.16: Summary of the RDF graph in Figure 2.11

Song et al. [80] also uses patterns to create the summarized graph. The authors present a novel summarization framework to compute diversified summaries and evaluate knowledge graph queries with their summaries. The approach takes a threshold "d" and the RDF graph as an input and provides a set of patterns that summarize and represent the graph as an output. The threshold "d" is used to capture the similarity between entities in terms of their label and neighborhood information up to distance "d". The approach provides summaries for schema-less knowledge graphs.

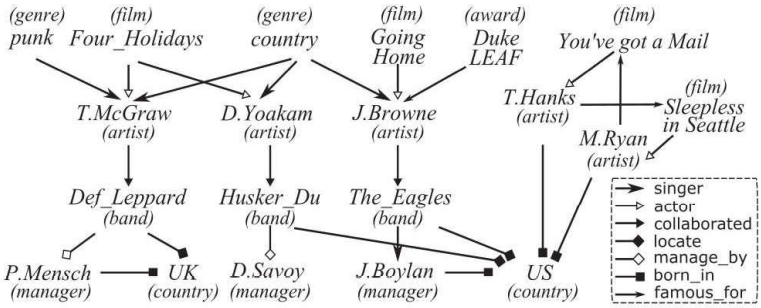


Figure 2.17: Knowledge graph [80]

Let us consider the graph in figure 2.17, and let "d" be equals to two; three different patterns can be extracted from this figure. The results are presented in figure 2.18.

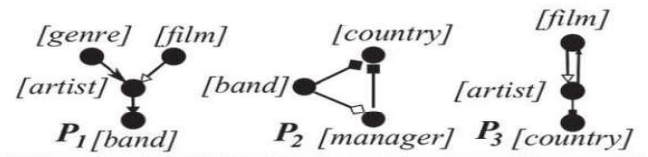


Figure 2.18: patterns extracted from the graph in figure 3.9 [80]

The authors introduce the notion of base graph for the patterns. The base graph consists of all the nodes in the original graph that match the patterns P. For example, the node "T.McGraw", "D.Yoakam" and "J.Browne" match the pattern P1, then these three nodes are in the base graph of P1. The base graph is used to facilitate query answering and to provide a general overview about the data in the graph.

Another approach which is VoG [53] also summarizes very large graphs using patterns. The proposed technique consists in decomposing and extracting patterns from the initial graph. First, the initial graph is decomposed into many overlapping sub-graphs using any decomposition algorithm. The sub-graphs are classified into four categories: cliques, bi-partite cores, stars or chains. This classification is based on the shapes that can represent them. The optimal summarization is created by using those sub-graphs. The method is to select the set of sub-graphs that together yield the best lossless compression of the graph, this is done by using the minimum description length (MDL)[77].

2.4.1.3 Statistical Summarization

The approaches in this category present summarization techniques based on statistical information. In this category, the content of the graph is summarized quantitatively. The main idea is to build the summarized graph based on statistical information that can be extracted from the graph; this information can be: (i) the occurrences of a class or property, (ii) the frequency of usage of a specific class with specific property (iii) the frequency of a path in the graph. In [13, 12] the summarized graph is created by extracting the type and predicate for each node in the RDF graph then grouping the nodes which share the same set of types into the same node summary. This node summary contains the type name and all its properties. The nodes that are not associated to a class are grouped according to their attributes. Each node in the summarized result will be associated with a number of its occurrences in the initial graph. The final result is not necessarily a graph. For example figure 2.19 shows the summarized representation of the graph of figure 2.11 after using the above approach.

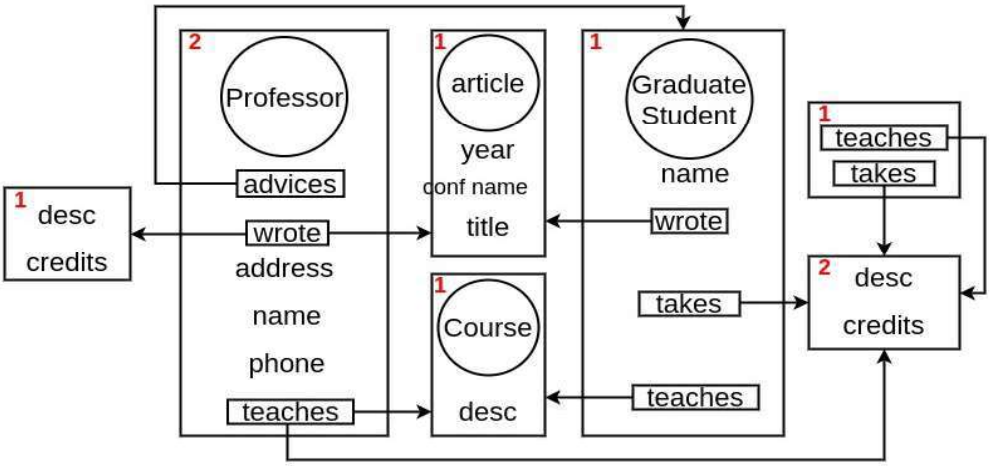


Figure 2.19: Summary of the RDF graph in Figure 2.11 after applying the approach presented in [12]

ExpLOD [49, 50] is another approach that presents a summarization technique based on statistical information extracted from the graph. The summarized graph in this approach is computed over the RDF graph based on the different sets that are extracted from the graph. These sets are created from the nodes based on classes and predicates. The authors use four different concepts to describe the RDF graph content and create the different sets. The concepts are: (i) class instantiation which is the number of instances of a class; (ii) predicate instantiation is the number of times a predicate is used to describe all instances; and (iii) predicate usage is the sets of predicates used to describe an instance. Then all

the nodes that share the same class usage are grouped together. The different sets are connected using the properties usage. The class instantiation, predicate instantiation and statistics are added to the final summarized graph. For example, the graph in figure 2.20 shows a summarized graph of the graph of figure 2.11 after applying the ExpLOD approach.

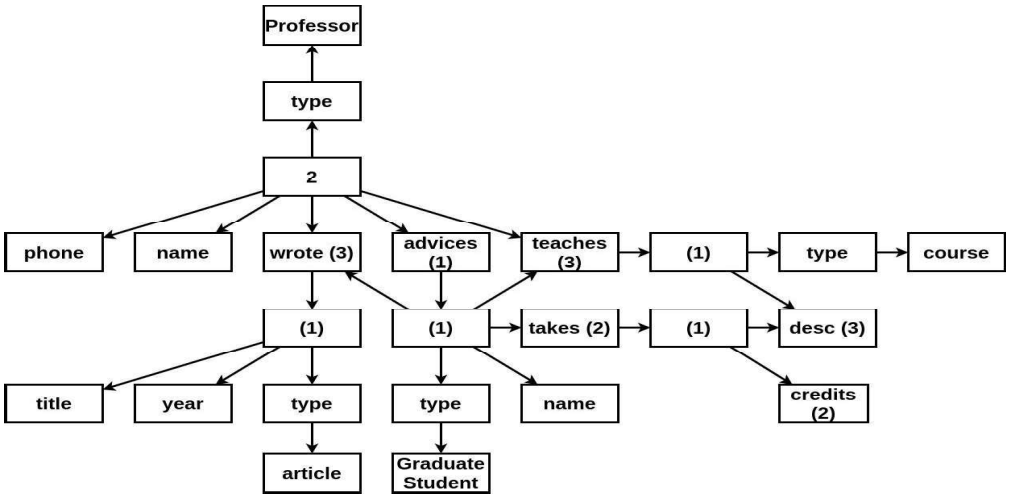


Figure 2.20: Summary of the RDF graph in Figure 2.11 after applying the ExpLOD approach [49]

Dudas et al.[25, 61] present the LODSight approach which finds the summarized graph based on properties and statistical information retrieved from the graph. The authors introduce two types of paths; the first one is the *type-property path* which is the path connecting two instances of different types (type1-property-type2) and the second path is the *data-type-property path* which is the path between an instance and a literal. LODsight starts by retrieving all the type-property paths in the graph and merges them together to create one graph, then finds all the data-type-property paths and add the one which subject type is present in the type-property paths. SPARQL queries are used to retrieve the above paths; through these SPARQL queries, it collects statistical information on the available combinations of types and properties. The statistical information are graphically visualized through the size of the nodes and edges in the summarized graph. The node with a high occurrence in the initial graph, will be visualized larger in the summarized graph. The summarized graph in figure 2.21 was created after applying LODSight approach on the graph of figure 2.11.

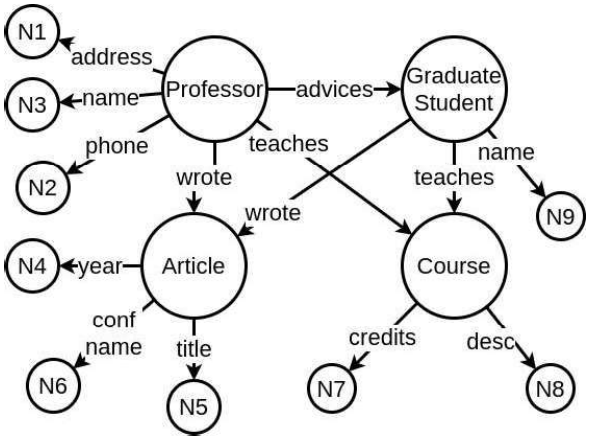


Figure 2.21: Summary of the RDF graph in Figure 2.11 after applying the LOD-Sight approach [61]

2.4.1.4 Hybrid Summarization

Summarization in this category is performed by combining two or more of the previous categories, the quotient summary, the central based summary and the statistical summary.

In [56] the authors propose an iterative utility-driven graph summarization approach. The summarized graph is build in an iterative way and keeps track of the utility of the graph summarization. The utility depends on calculating the difference between two scores, the first one is the centrality of edges in the summarized graph; the second score is the percentage of change (loss of gain nodes and edges) in the data graph after reconstructing it from its summarized one. The centrality of edges can be assigned by four properties which are the following:

- Edge Importance: Changes that create disconnected components or weaken the connectivity should be penalized more than the changes that maintain the connectivity properties of the graph.

- Spurious Edge Awareness: Additional edges for connecting the relevant elements must lead to a lower utility.

- Weight Awareness: In weighted graphs, the higher the weight of the removed edge or added spurious edge, the greater the impact on the utility measure should be.

- Edge Submodularity: A specific change is more important in a graph with fewer edges than in a much denser graph.

The summarization process consists of first creating a set of pairs of nodes from the graph as candidates to form supernodes. Each pair represents two nodes that are either directly connected by an edge or by a 2-hop connection via a common neighbors. An importance score is calculated and assigned to each pair; the score is based on combining the node centrality score for all the nodes in the path connecting the nodes of the pair. Creating a summarized graph starts by combining the two nodes related to the pairs with the lowest importance score into one supernode. After that, all the neighbors of the combined nodes are checked to see whether they can also be combined with the supernode or not.

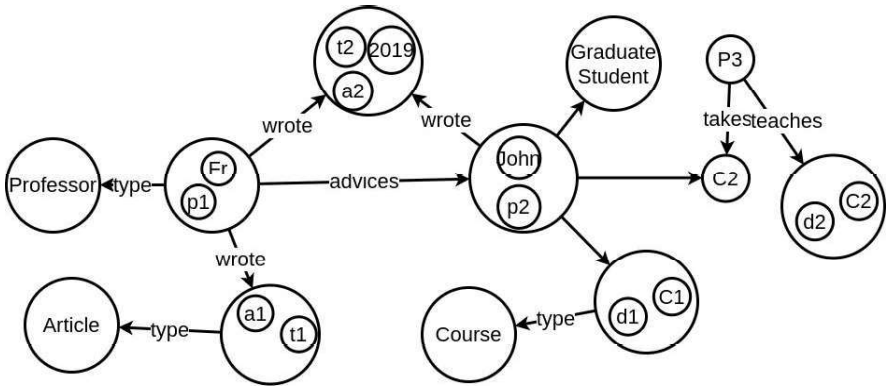


Figure 2.22: Summarized graph after applying the utility algorithm on the RDF graph in Figure 2.11

The decision is taken after calculating the penalty of connecting it and not connecting it to the supernode. After checking all the neighbors, a new pair of nodes is selected and all the previous steps are executed. The process ends when there are no more nodes that can be combined to a supernode. The final result is a summarized graph. Figure 2.22 shows the summarized graph obtained after applying the utility algorithm on the graph in figure 2.11.

In [69] the notation of centrality and frequency are used to summarize the RDF graph. First, the centrality (an adaptation of the degree centrality) and the frequency of all the nodes in the graph are calculated. Then relevant concepts are selected according to the defined threshold. Grouping the adjacent nodes in the initial graph and then in order to link non-adjacent groups, the first k-paths connecting them are examined to select the ones that have the best f-measure and average relevance. Figure 2.23 shows the summarized graph obtained after applying the above algorithm on the graph in figure 2.11.

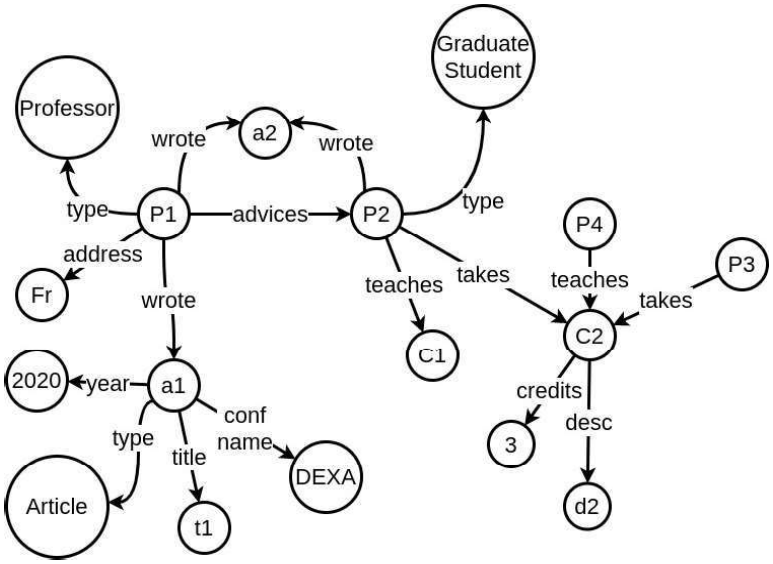


Figure 2.23: Summarized graph after applying the above algorithm on the RDF graph in Figure 2.11

2.4.2 Summarization Based on Graph Analysis Techniques

In the previous section we have studied the approaches that build the summarized graph based on the node metrics, in this section we will study another method of summarizing the RDF graph by using the graph analysis techniques. In the previous part the summary was build based on assessing each element alone in the initial graph. In this part the approaches build the summary by using another perspective that consists in using some graph analysis technique to assess the whole graph and build the summary. The graph analysis technique is different from one approach to another, but they all focus on decomposing the graph and identifying some sub-graphs which represents some logical unit, each sub-graph alone have some shared semantics which are related to the same topic that can be considered as a theme. The thematic view of an RDF dataset can be considered as a summary since this way provides information about the semantic of this graph. In this section, we will present the most important approaches that uses graph analysis techniques to summarized the RDF graph.

2.4.2.1 Community structure

The goal of this approach is to decompose the graph by using the betweenness centrality. This centrality was first defined by [27], the centrality betweenness of a vertex "i" is defined as the number of shortest paths between pairs of other vertices that pass through "i". Based on this definition, [30] defines an approach to detect

the different themes in the graph according to the edge betweenness centrality. The betweenness centrality of an edge is the number of the shortest paths that contains this edge in a graph. Let us consider the graph in figure 2.24 with the edge betweenness centrality on each edge of the graph. For example, the edge betweenness centrality for the edge connecting nodes "c" and "d" is calculated according to the number of shortest paths that pass through this edge. All the paths that connect the nodes {"a" , "b" or "c"} with the nodes {"d", "e" or "f"} pass through this edge then the edge betweenness centrality is equals to nine.

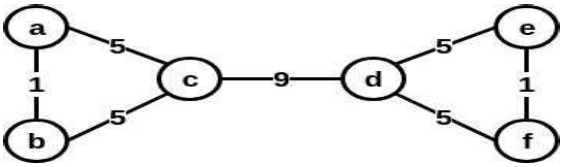


Figure 2.24: Undirected Weighted Graph

The algorithm used to identify the communities consists in calculating the betweenness centrality score for all the edges in the graph and then select the edge with the highest betweenness centrality score to be removed from the graph, recalculate the betweenness centrality for all the edges affected by the removal, this process is repeated recursively until the communities are detected.

2.4.2.2 Label propagation

The approach proposed by Raghavan et al. [74] use the Label propagation algorithm (LPA); this approach works by propagating labels throughout the network and forming communities based on these labels. The algorithm starts by giving each node a unique label. At each propagation iteration, each node is updated by replacing its label with the one used by the highest number of neighbors. At the end of the propagation, nodes that have the same labels are assigned to the same community. Figure 2.25 shows an example of detecting three different communities after applying the LPA algorithm on the graph of the Zacharys Karate club network [74].

Gregory et al. [33] improved the Label propagation algorithm [74] in order to detect the overlapping communities in very large graphs. LPA can not detect overlapping communities since each node should have only one label. It is therefore impossible for a node to belong to more than one community. The approach proposed in [33] extends LPA in order to allow a node to have more than one label, so the node label will be a set of pairs (c,p) where c is the community identifier and p is the probability of this node to be labeled by c. Initially, each node should be labeled by a unique identifier, after each propagation step, the label

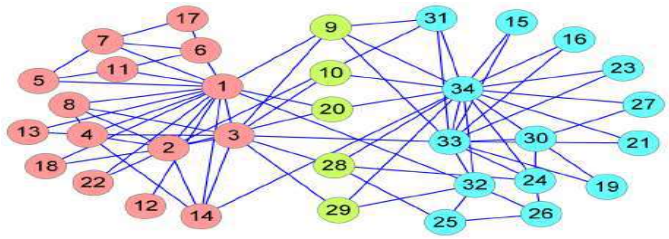


Figure 2.25: Appying LPA on the graph of Zacharys Karate club network [74]

of a node is the union of its neighbors' labels. This is done by normalizing the sum of the probabilities of the communities overall its neighbors. Let us consider the example in figure 2.26.

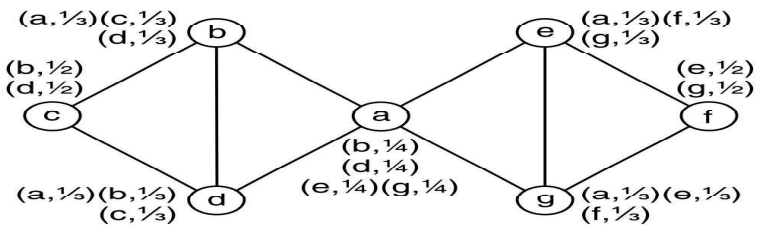


Figure 2.26: Initial step of label propagation [33]

Consider the node b having c, d and a as its neighbors. After the first propagation, the set of community identifiers for b will be (a,1/3),(b,1/3),(c,1/3). The same process is performed for all the other nodes.

A threshold is defined in order to detect the final overlapping communities. if v is the maximum number of communities any node can belong to then the threshold is the reciprocal of v ($1/v$). After the first propagation, each pair (c, p) having a probability p less than the threshold will be deleted. It is possible that all pairs in a node label have a probability less than the threshold. In this case, the pair that has the greatest probability is kept and all the others are deleted. If more than one pair has the same maximum probability below the threshold, a randomly one is selected to be kept. After deleting pairs from the node label, the probabilities are normalized by multiplying the probability of each remaining pair by a constant so that their sum equals 1.

Figure 2.27 shows the results after executing the algorithm with $v = 2$ on the graph of figure 2.26. In the first iteration, node c is labeled with community identifiers b and d, each with the probability (1/2), those two labels are kept because their probability is equal to the threshold (1/2). Similarly, f is labeled with e and g. The other five vertices have at least three neighbors, so their probabilities are below the threshold. For example, b is first labeled with (a, 1/3), (c, 1/3),

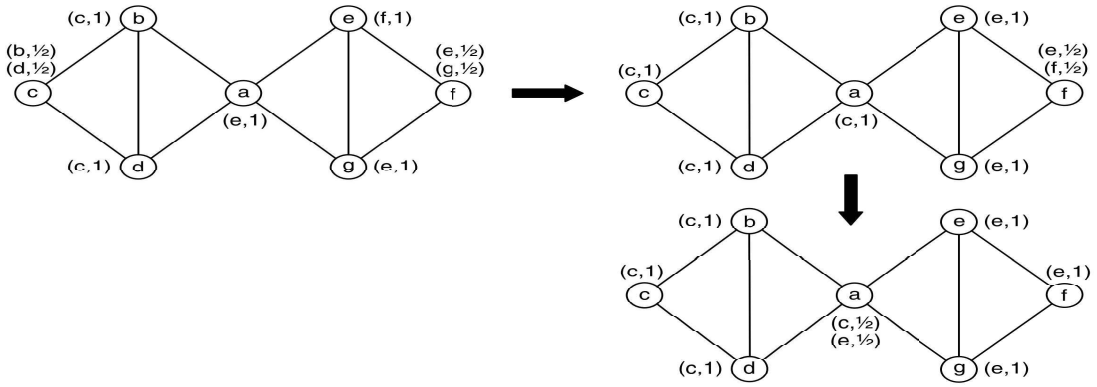


Figure 2.27: Detecting overlapping community by using label propagation [33]

(d, 1/3). We randomly choose c, delete a and d and we normalize the sum of probabilities so that the total is 1 and therefor c is associated to probability 1, (c,1). The labels for a, d, e and g are randomly chosen in the same way. Before the final iteration, node a has two neighbors labeled c, and two others labeled e; the node a will therefore have both community identifiers: {(c, 1/2), (e, 1/2)}. The final solution contains two overlapping communities: {a,b,c,d} and {a,e,f,g}.

2.4.2.3 Theme Identification in RDF Graphs

The idea of this approach is to use the MCODE algorithm [4] to detect the different themes in an RDF graph that are represented by dense area. MCODE was proposed first in the field of bioinformatics for the identification of the molecular complexes in large networks of interactions based on the extraction of dense regions in the network. MCODE is composed of three steps that are described below.

Vertex Weight:

This step consists in calculating the weight for all the nodes in the graph based on the notion of k-core defined as follows:

Definition (k-core): A k-core of a graph G is a maximal connected subgraph of G in which all vertices have a degree of at least k.

The weight of the node v_i is calculated based on the density of the largest k-core formed by this node and its directly connected neighbors. The weight of the node tends to be high in the highly connected areas in the graph. The weight of all the nodes is calculated according to algorithm 1.

Let us consider the example in figure 2.28 which consists of a graph with a weight associated to each node. The node weight represents the maximum k-core which can be created from this node. For example the weight of the nodes a, b and c is equal to 3 since the maximal connected sub-graph in which all vertices have

Algorithm 1 MCODE-Vertex-Weighting

procedure MCODE-VERTEX-WEIGHTING(graph G)
  $W = \emptyset$
  for $v \in G$ **do**
    $N \leftarrow findneighborsofvtodepth1$
    $K \leftarrow Gethighestk - coregraphfromN$
    $k \leftarrow Gethighestk - corenumberfromN$
    $d \leftarrow GetdensityofK$
    $w(v) \leftarrow kxd$
    $W \leftarrow W \cup (v, w(v))$
  end for
end procedure

a degree of at least 3 is the graph containing the nodes a, b, c and d, similarly to the nodes d, e, f and g but with k-core equal to 4.

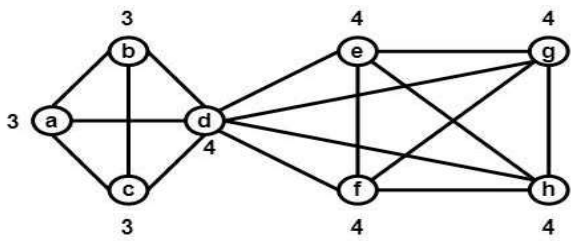


Figure 2.28: Example of k-cores in a graph with the MCODE algorithm

Identifying topics

The different topics in the graph are identified based on the weights of the nodes from the previous step. This step takes the weighted graph as an input and returns a set of clusters as an output where each cluster describes one topic. In addition, the algorithm requires a parameter t ranging from 0 to 1. To build the clusters, the algorithm starts with the node with the highest weight called seed node, to initiate a cluster. All nodes adjacent to the seed node are added to the cluster, provided that their weight is greater than the predefined threshold which is proportional to the weight of the initial node and defined as follows: w(n_j) > w(n_i) * (1 — t) where n_i is the seed node, and n_j one of its neighbor. Each time a node is included in a cluster, all of its neighbors are considered in the same way to check if they should be included in the current cluster. Each node added to a cluster is marked and it will not be considered when building the other clusters. Once there are no more nodes that can be added to the current cluster. We select a new seed node by choosing from the unmarked nodes the one with the highest

weight for initiating a new cluster. The same procedure is applied to build a new cluster. The process ends when there are no more unmarked nodes. It should be noted that t determines the level of connectivity as well as the size of the clusters. The closer it gets to 0, the denser and smaller the clusters.

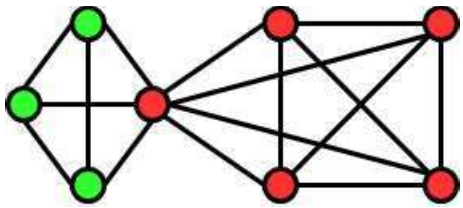


Figure 2.29: Clustering the graph with MCODE algorithm

Post Processing:

The final step of the MCODE algorithm is aimed to identify the overlap between the clusters of the previous step in order to produce a set of overlapping clusters. This step uses a threshold d ranging from 0 to 1 to identify the overlapping clusters. The algorithm traverses all the clusters and all the neighbors of each node n_i of a given cluster, if the density of the subgraph formed by n_i and its direct neighborhood exceeds the threshold 1-d, then this node is added to the current cluster. The nodes added to the clusters by this step are not marked, which will allow the same node to be inserted into multiple clusters.

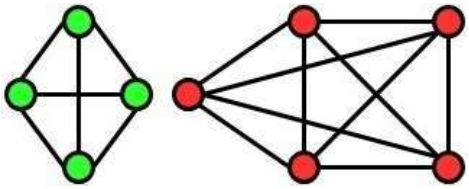


Figure 2.30: Finding overlapping communities with MCODE algorithm

2.4.2.4 InWalk [14]

This approach extracts the topics for a given RDF graph by using the Clique Percolation Method(CPM) which is a method introduced by [67] to determine the overlapping communities in the graph. The main idea of the method consists of finding the k-cliques and constructing the communities according to these cliques.

Definitions:

k-clique: a k-clique is a complete graph of k vertices with $k(k-1)/2$ edges.

Adjacent K-cliques: Two cliques C_1 and C_2 are adjacent if they share exactly k-1 nodes.

The algorithm starts by finding all the k-cliques in the RDF graph for a given size K. The second step is to create a clique graph, where all cliques are nodes and the cliques that are adjacent are connected with an edge. At the end, the communities are the connected components of this graph. The CPM method is presented in the algorithm 2.

Algorithm 2 Clique Percolation Method (CPM)

- **Parameter:** K.

1: return Overlapping communities
2: $Clique_k$ = find all the cliques of size K
3: Construct clique graph G(V,E), where |V| = $|Cliques_k|$
4: E = e_{ji}| clique i and clique j are adjacent
5: Return all connected components of G

For example, figure 2.31 (a) consists of a graph. First, all the k-cliques from this graph where k is equal to three. Figures 2.31 (b) 2.31 (c) and 2.31 (d) show the three cliques that are computed from the graph in figure 2.31 (a).

Two cliques C1 and C2 are adjacent if they share two nodes between them. this notation is used to construct the clique graph by representing each clique with one node, and if two nodes n1 and n2 are representing two different cliques C1 and C2 then n1 and n2 are connected by an edge.

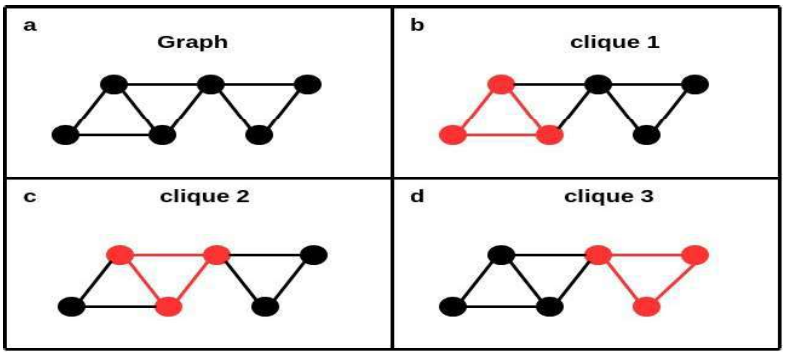


Figure 2.31: Clique Percolation Method [67]

For example clique 1 and clique 2 in figure 2.31 are adjacent since they have two share nodes; the other cliques are not adjacent. Figure 2.32 (a) represents the clique graph for the main graph in 2.31 (a).

The connected components represent the communities, figure 2.32 (b) shows the two communities that can be derived.

The last step of the inWalk approach consists in extracting for each topic a set of labels that describe its content; these labels are extracted according to some

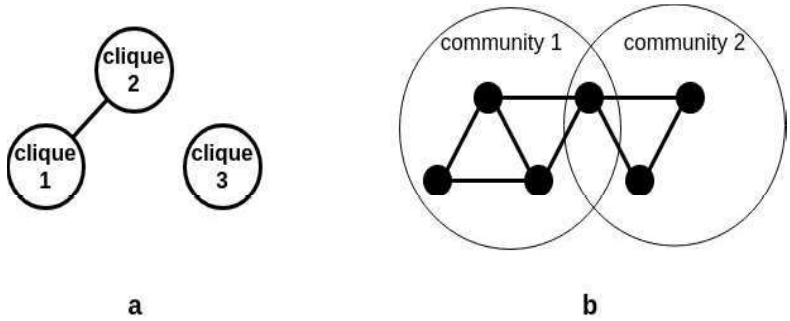


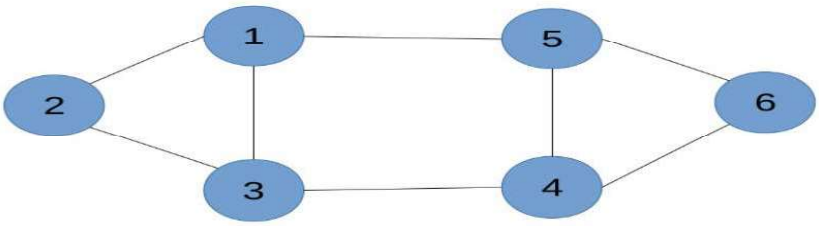
Figure 2.32: Practical example for Clique Percolation Method [67]

statistics related to the occurrences of the elements in the theme.

2.4.2.5 Spectral Partitioning [5, 2, 71]

This algorithm aims to detect the different communities in a graph using the eigenvectors and eigenvalues. Let G=(V, E) be a graph, the degree matrix (D) of G is a diagonal matrix which contains information about the degree of each node in G, and the adjacency matrix (A) is a square matrix used to represent G. The elements of the matrix indicate whether pairs of vertices are adjacent or not in G. The Laplacian matrix (L) is defined as L = D - A.

The spectral partitioning method consists of finding the second eigenvector and eigenvalue for the Laplacian matrix.



$$A = \begin{bmatrix} 0 & 1 & 1 & 0 & 1 & 0 \\ 1 & 0 & 1 & 0 & 0 & 0 \\ 1 & 1 & 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 & 1 & 1 \\ 1 & 0 & 0 & 1 & 0 & 1 \\ 0 & 0 & 0 & 1 & 1 & 0 \end{bmatrix} D = \begin{bmatrix} 3 & 0 & 0 & 0 & 0 & 0 \\ 0 & 2 & 0 & 0 & 0 & 0 \\ 0 & 0 & 3 & 0 & 0 & 0 \\ 0 & 0 & 0 & 3 & 0 & 0 \\ 0 & 0 & 0 & 0 & 3 & 0 \\ 0 & 0 & 0 & 0 & 0 & 2 \end{bmatrix} L = \begin{bmatrix} 3 & -1 & -1 & 0 & -1 & 0 \\ -1 & 2 & -1 & 0 & 0 & 0 \\ -1 & -1 & 3 & -1 & 0 & 0 \\ 0 & 0 & -1 & 3 & -1 & -1 \\ -1 & 0 & 0 & -1 & 3 & -1 \\ 0 & 0 & 0 & -1 & -1 & 2 \end{bmatrix}$$

Figure 2.33: Graph G with adjacency diagonal and Laplacian matrix of this graph.

Let us consider figure 2.33 which is composed of a graph with 6 nodes and 8 edges, the figure also shows the adjacency, diagonal and Laplacian matrices of this

graph.

In the figure 2.34, the eigenvectors denoted by X and the eigenvalues denoted λ, for the Laplacian matrix in figure 2.33 are presented. The communities are detected according to the sign of the values in the second eigenvector X_2. The values of nodes 1, 2, and 3 in the second eigenvector are positive while the values of nodes 4, 5 and 6 are negative, then the graph can be decomposed into two different communities. The first one consists of nodes 1, 2 and 3 while the second one consists of the nodes 4, 5 and 6. Figure 2.35 shows the two communities.

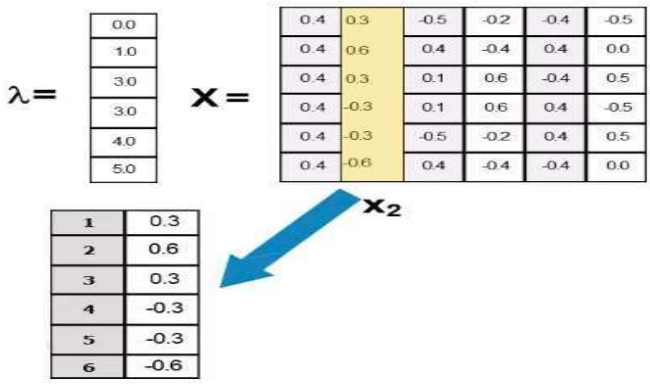


Figure 2.34: Eigenvector and Eigenvalue for the Laplacian matrix in figure 2.33

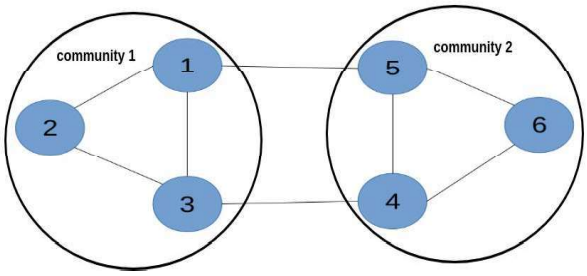


Figure 2.35: Extracting communities with spectral partitioning algorithm on graph of figure 2.33

2.4.2.6 Kernighan-lin(KL) [48]

Kernighan-lin [48] is a graph partitioning method used to partition the nodes of the graph G(V, E) into two disjoint subsets A and B of equal size in a way that minimizes the number of edges that cross from A to B, these edges are denoted by cut edges. The goal of the algorithm is to reduce the number of cut edges between

the two partitions. The algorithm starts with the initial partition for sets A and B and tries to find a sequence of node pair exchanges (exchange nodes of A with nodes of B) that can decrease the number of cut edges between sets A and B.

Let us Consider a graph G(V, E), and let A and B be two different set of nodes such that the size of A equals the size of B, and V = A ∪ B. Let a ∈ A and b ∈ B be two nodes in A and B. The cost denoted cons(A, B) equals the number of cut edges between the sets A and B. Suppose we want to check if the exchange between pairs (a,b) can decrease the number of cut edges between sets A and B. We define two concepts the external cost and the internal cost. Let I_a be the internal cost of node a, that is the sum of the costs of edges between a and other nodes in A, and let E_a be the external cost of node a, that is the sum of the costs of edges between a and nodes in B. Similarly, we define the two costs I_b, E_b for set B. Let $D_s = E_s$ - I_s be the difference between the internal and the external cost of s. The cost of exchanging node a and with node b can be defined as (cost (a,b)) = $D_a + D_b$ - $2c_{a,b}$ where c_{ab} equal one if there is an edge between a and b; and zero otherwise.

Figure 2.36 presents a graph with 6 nodes, which is randomly partitioned into two equal sized subsets of nodes. We can see from the figure that the graph is partitioned into two different sets A and B, where A represents the set inside the red box A={2, 3, 4} and B ={1, 5, 6} represents the set outside the box. The number of cut edges between the two sets A and B is equal to three since there are three different edges connecting nodes in the set A to other nodes in set B.

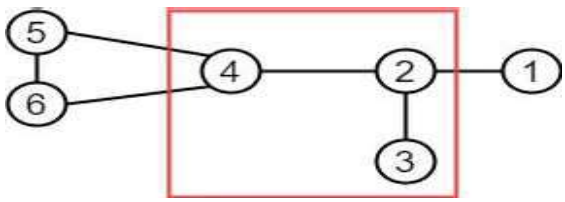


Figure 2.36: Graph partitioned into two equal subset of nodes

In the two tables 2.4 and 2.5 all the external and internal costs for all the nodes in sets A and B are presented.

partition A	Ea	Ia	D
2	1	2	-1
3	0	1	-1
4	2	1	1

Table 2.4: External and Internal costs for all the nodes in A

partition B	Ea	Ia	D
1	1	0	1
5	1	1	0
6	1	1	0

Table 2.5: External and Internal costs for all the nodes in B

Table 2.6 contains all the possible combinations of exchanging a node in set A to another node in set B. The combination with the highest positive gain score

	G2,1	G2,5	G2,6	G3,1	G3,5	G3,6	G4,1	G4,5	G4,6
c(a,b)	1	0	0	0	0	0	0	1	1
Gain	-2	-1	-1	0	-1	-1	2	-1	-1

Table 2.6: The gain score of all the possible combinations of exchanging a node in set A to another node in set B

is selected. From the table we can see that "G4,1" has the highest positive score which is equals to "2", then the nodes "4" and "1" should be exchanged in order to reduce the number of cut edges between sets A and B. After the exchange, the new sets are A= 1,2,3 and B=4,5,6 with one cut edge. The graph is partitioned into two equal sets A and B with the minimal number of cut edges and they are presented in the figure 2.37.

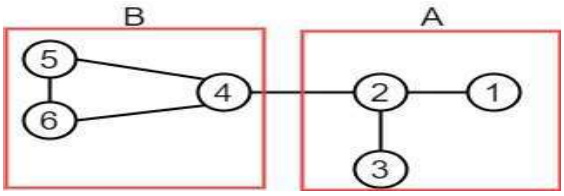


Figure 2.37: Graph partitioned into two equal subset of nodes with minimum number of cut edges after applying KL algorithm

2.4.3 Discussion and open issues

After presenting the existing summarization approaches, we can see that there are two different sets of approaches. The first set groups the approaches that are based on node level metrics. These approaches summarize the graph based on analyzing the nodes. These analysis can be the selection of the most important node according to its centrality or frequency, or can be based on the equivalent between the nodes based on equivalence relation. In this group, the approaches are divided into four different categories; the first one is the quotient graph [36, 32, 19] which groups approaches that use the equivalence relations to build the summarized graph. But this approach is not sufficient, since in some graphs it is possible to have the data without the schema and the nodes are not necessarily typed. In this case, this approach will not be able to provides a summarized graph.

The second category in the node based summarization groups the approaches that uses the centrality, such as in [82]. The approaches in this category determine

the importance of the nodes based on different centrality metrics. The most important nodes are selected to be presented in the summarized graph. The creation of the summarized graph is done by connecting the important nodes. Finding the connection between the important nodes is different from one approach to another; some of them consider the shortest path, others define a score to assess the path based on its length and the degree of the nodes in this path. Using this technique the summarized graph might be containing nodes from just one type (in the case where all the important nodes belong to the same class) and such a summary does not reflect all the information presented in the main graph.

The approaches that use the statistical methods is the third category in the node based summarization such as in [49, 49]. The approaches in this category build the summarized graph based on statistical information collected from the graph. This information can be the frequent number of the classes and/or predicates. The statistical calculations are also used by [13, 12], the summarized graph is created by extracting the type and predicate for each node in the RDF graph then group the nodes which share the same set of types into the same node summary. These approaches are important since they reflect the importance of the class in the graph by having information about its instances. The limitation of these approaches consists in presenting all the classes in the summary and ignoring the untyped nodes even if they are more important than the typed ones. Moreover, the summarized graph in this category might have a very large number of elements such as all the classes, but these elements are not reflecting the important elements in the RDF graph.

The last category in the node based summarization is the group of hybrid; the approaches in this category [56, 69] create the graph by combining two or more of the previous three categories. The limitation of the approaches in this category is that the user needs to use more than one metric to determine the relevant nodes; these metrics are based on some statistical information extracted from the graph, such as the number of instances for a specific type or the number of times a property is used or the number of accuracy for a specific type with a list of properties, but this is complicated, and time-consuming technique since the user needs to do a lot of calculations to determine the importance of the node. Moreover, since it is a hybrid category, then all the limitations of the three previous categories are the limitation of this one also.

The second group is the approaches that summarize the graph based on graph analysis techniques. One of the methods used in this group is the identification of the different underlying themes in the graph. The second method is to decompose the graph into different sub-graphs. The importance of this technique is in presenting all the topics of the graph, but the topics are not a summary.

Table 2.7 shows some summarized graph with a comparison between the ap-

Approaches	Technique or Algorithm	Summarization Unit	Require node Typing Information	Constraints on the Summary
[47, 19]	Quotient Graph	Nodes	No	Not identified
[16, 17]	Quotient Graph	Nodes	Yes	Not identified
[82]	Centrality metrics	Nodes	No	size constraint
[68]	Centrality metrics	Nodes	No	size constraint
[94]	Graph patterns	Patterns	No	Not identified
[53]	Graph patterns	Patterns	No	Not identified
[13, 12]	Statistical Information	Nodes and edges	No	Not identified
[49]	Statistical Information	Node and edges	Yes	Not identified
[25]	Statistical Information	Paths	No	Not identified
[14]	InWalk	Nodes	No	Not identified
[48]	Kernighan-lin	Nodes	No	Not identified
[69]	hybrid method	Nodes	No	size constraint

Table 2.7: Comparative Table of Summarization

proaches. The comparison is done on four aspects. The first one is to show the summary was build according to the node level metrics or graph analysis. The second aspect is to show the elements that are assessed and included in the summarized graph, from the table we can observe that some approaches use only the nodes, others use nodes and edges and there are also approaches that use the patterns. Checking if the type of the node was taken into consideration is the third aspect of comparison. We can observe that the approaches are divided into two parts, some of them takes the type into consideration and others are not. The last aspect in this comparison is the size of the summarized graph, some of the approaches did not identify the final size but others put constraints on the final size.

After presenting all the above approaches, we can observe that some of them prioritize the typed nodes and preserve all the classes in their summary as the approached presented in [15, 36, 16, 17, 94, 80, 13, 12, 13, 12, 25, 61]. It is difficult

to know in advance that the most important nodes are the typed nodes or the untyped nodes, we should be able to select relevant nodes regardless if all the types are presented in the summarized graph or not. For example, if we have a graph with more than 100 classes, then all those classes are represented in the summarized graph. Therefore the computed summary graph is still a big graph, and the user cannot understand it easily. Some of these classes are important, the others are not then we do not need to select them all. The second limitation of the existing approaches consists in preserving the classes along with their instances which is kind of redundancy. The instance is represented by the class then there is no need to have it also in the summary. The third limitation of the presented approaches is creating summary containing one topic of the graph. If we use some relevant metric to select the most important nodes, we discover that these nodes are related to the same topic which makes the summary less diverse. What could be interesting is building a more diverse summary. For example, if all the important nodes in the graph belong to the same class, then the summarized graph will be representing one class and its instance and ignoring the others. This is not good summarized graph.

2.5 Conclusion

In this chapter we have studied approaches related to keyword search and approaches related to summarization. In the first part, we discussed the different approaches that target keyword search over RDF data. We have seen that the approaches mainly focus on finding the matching elements between the keyword query and the RDF data, some of them match the query keyword with just nodes, but others match with nodes and edges. Some of them use external knowledge to enrich the matching elements, and others match the exact keyword between the query and the terms in the dataset. For the aggregation challenge, some approaches use information from the schema to connect the matching elements and build the final result, and others use a metric to identify the most important path to connect the matching elements. The main limitation of the existing approaches is how to fill the terminological gap between the keywords in the query and the terms used in the RDF data. The second limitation is the aggregation of the matching elements in order to find the best sub-graph results.

The second part of the chapter discussed different approaches about the graph summarization techniques. We have seen that most of the approaches find the most relevant elements in the graph and connect them to create the summarization graph. The main limitation in such approaches consists of creating graphs lacking from diversity in the content of the graph. Moreover, most of the summarized graphs are not taking into consideration all the underlying topics in the

RDF graph during the process of summarization. We also have seen that in some approaches, all the classes appear in the summarized graph. The problem is that the summarized graph might contain classes that are not important at the expense of important untyped nodes. Other approaches just focus on the importance of the elements regardless if they preserve all the classes in the summarized graph or not. The problem of such approaches is that the important elements might all belong to the same class, then the summarized graph will miss the diversity.

As a conclusion, this work mainly targets two different aspects of exploring RDF graphs. The first one is the keyword search and how to fill the terminological gap between the queries and the data-set and also find the best aggregation method to create a set of sub-graphs as a solution and finally provide a ranking method to rank the results. The second one is the summarization, which consists of creating a summarized graph by taking into account all the different topics in the RDF graph and selecting the most central nodes after using different metrics. Finally, create the summarized graph by aggregating the relevant nodes.

In the upcoming chapters, I will present my contributions for both keyword search and summarization.

Chapter 3

Semantic-Based Matching for Keyword Search in RDF Data

Contents

3.1 Introduction 60

3.2 Approach Overview 62

3.3 Problem statement 64

3.4 Indexing an RDF Data Graph 66

3.4.1 Structure of the Index 67

3.4.2 Building the Index 68

3.5 External Knowledge used to Enhance the Matching Process 69

3.5.1 Semantic relations to bridge the terminological gap . . . 70

3.5.2 Equivalence between properties and paths 71

3.5.2.1 SameResult Pattern 73

3.5.2.2 SameProperty Pattern 74

3.5.2.3 Property Pattern 74

3.6 Enhancing the Matching Process with Semantic Relations 75

3.6.1 Finding Equivalent Matching Elements 76

3.6.1.1 Exact Matching 76

3.6.1.2 Synonymy Relations 77

3.6.1.3 Antonymy Relations 77

3.6.2 Finding Close Matching Elements 79

3.7 Using Patterns in the Matching Process 80
3.8 Semantic relations and pattern in the Matching Process 82
3.9 Experimental evaluation 86
3.10 Conclusion 90

3.1 Introduction

Exploring RDF datasets is a very important task to understand the dataset and use it. One way of exploration consists of using one of the query languages (such as SPARQL) to retrieve the needed information for our usage. To use this language, the user should be familiar with its structure because it has a specific characteristic which consists of a set of triples where the subject, predicate and/or object can be variables. The idea is to match the triples in the SPARQL query with the existing RDF triples and find solutions to the variables. Let us take the graph in figure 3.1 as an example, if the user needs to retrieve the movies that are released in 1974 and Al Pacino is one of its actors, then he should write the following SPARQL query:

```
SELECT ?Movie
WHERE {?Movie Release_date 1974.
?Movie starring Al_Pacino.}
```

From the above query, we can observe that the SPARQL query consists of two parts: the SELECT clause identifies the variables to appear in the query results, and the WHERE clause provides the basic graph patterns to match against the data graph. The user should know the language and the different parts of it in order to write a query. The structure of the SPARQL query is not the only problem that faces the user, there is also another problem which consists of having knowledge about the content of the RDF dataset. The user needs to know the different classes and properties in the RDF data set, since this kind of information is necessary to write the triples of the SPARQL query. For example, to write the above SPARQL query, the user should know that "release_date" and "starring" are properties in the RDF dataset.

An alternative way of querying RDF datasets is keyword search, in which the user can write a query without having a knowledge base either about the structure of the query or about the content of the RDF dataset.

Keyword search has been used to retrieve information from a large amount of data, whether the data is databases or web pages. The idea of keyword search is the same as in web browser, where the user issues some keywords to find the

relevant documents that matches these keywords. RDF data can be represented as a graph and the keyword search over the RDF graph consists in finding the graph fragments that match the keywords and connect them to create a sub-graph which represents a result to the keyword query.

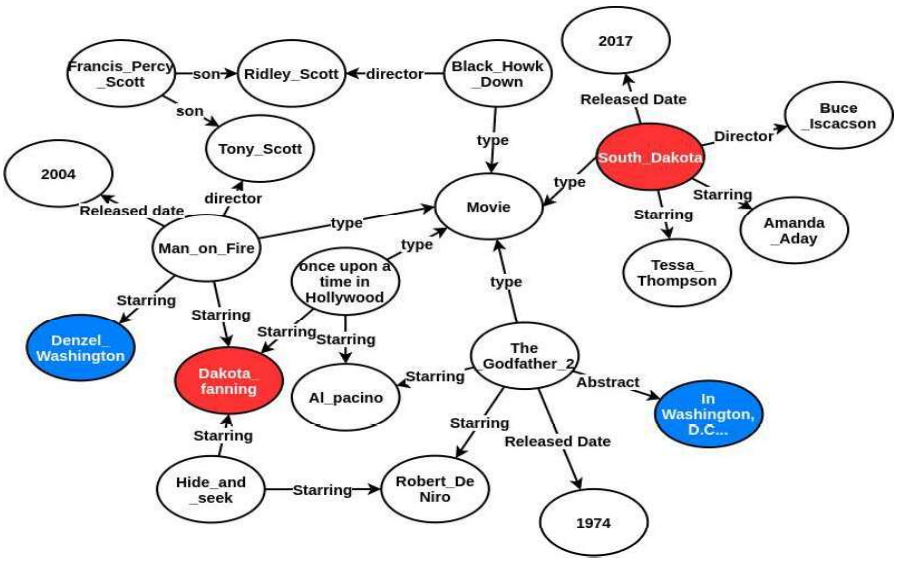


Figure 3.1: An Example of RDF Dataset describing Movies

During the matching process, each query keyword can be matched to more than one element in the graph. For example, let us consider the RDF graph in Fig. 3.1 and consider the query Q1 = {Dakota, Washington} to be the keyword query submitted to this graph. As we can observe, the keyword "Dakota" can be matched to two nodes in the graph highlighted in red, and the keyword "Washington" can be matched to the nodes highlighted in blue. The goal of the matching process is to identify all the possible elements that can be matched to one of the query keywords.

The user might issues a keyword that can not be found in the RDF graph, but we can find some elements that are close in meaning to this keyword. The problem is to identify the close concepts to a keyword in the graph. Let us consider the graph in figure 3.1 as an example and issue the following keyword query Q2=["film maker", "brothers","2004"] on this graph. We can observe that there is no element that contains "film maker", but we know that "film maker" is closed in meaning to "director", then we need to find a technique that helps us to select "director" as a matching element for "film maker". Also, we can observe that "brothers" can not be matched to any element in that graph but from the graph we know that "Ridley_Scott" and "Tony_Scott" are brothers since they are both the sons on "Francis_Percy_Scott". The problem consists in finding a technique that

helps us in selecting "Ridley_Scott" and "Tony_Scott" as matching elements to the keyword "Brothers". From this example we can deduce that the type of the matching elements can be a node as for the keyword "2017" or an edge as for the keyword "film maker" that can be matched with the edge labeled with "director", or even a sub-graph as for the keyword "brother" that is matched with the sub-graph that connects "Ridley_Scott", "Tony_Scott" and "Francis_Percy_Scott".

The challenge we target in this chapter is identifying the matching elements. It lies on the limitation of the existing techniques in detecting the close concepts in the graph to the query keywords. The user might provide a keyword query that contains concepts different from the one used in the graph. Our goal is to solve the heterogeneity of terminology used between the keyword query and the elements of the RDF graph.

The rest of this chapter is organized as follows. The overview of our keyword search approach is presented in section 3.2. The problem statement is provided in section 3.3. We present the indexation in section 3.4. We have used semantic knowledge provided either by domain experts or by online resources, this knowledge is provided in Section 3.5. The use of the external knowledge is presented in sections 3.6 and 3.7. Section 3.8 presents our marching process. Our experiments are presented in section 3.9. Finally, we conclude the chapter in section 3.10.

3.2 Approach Overview

In chapter 2, we have presented the existing approaches for keyword query and how the problem is targeted via multiple techniques. However, applying the existing approaches on RDF graphs will not produce results in some cases. Our goal is to find a solution that takes into account the similarity between the keywords in the query and the elements in the RDF graph. To do so, we propose a framework for keyword search that is presented in figure 3.2. This framework takes a set of keywords as an input and returns a set of subgraph results as an output. Each subgraph consists of a minimal sub-graph containing one matching element for each keyword query. The main idea is to match the keywords of the query with the elements of the graph and then connect them to have the final results. Our framework comprises three components, a matching component, which searches the graph elements corresponding to the keywords, the aggregation component which aggregates and connect the matching elements to create possible subgraph result, and ranking component to rank the multiple results for a given query. In the next section I will present each component, and what problem each one of them targets.

The matching component takes as input the keyword query and searches for the matching elements in the dataset. Each keyword is compared to the graph elements

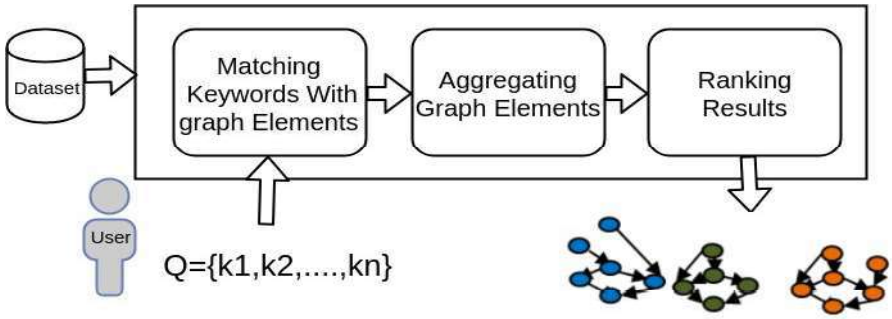


Figure 3.2: Approach Overview

such as nodes, edges or sub-graphs, and the matching elements are identified. In some cases, the user may enter a keyword for which an exact match can not be found in the dataset, but some graph elements could be close to the keyword. The problem is to identify the close concepts to a keyword in the dataset. Let us consider the graph in figure 3.1 as an example and issue the following keyword query Q3=["film maker", "abstract"] on this graph. We can observe that there is no element that contains "film maker", but we know that "film maker" is closed in meaning to "director", then we select "director" to be a matching element for film maker. Also, how can we select the edge "abstract" as a matching element to the keyword query "abstract".

Once the matching elements in the RDF graph are identified for each keyword, the final result from these elements needs to be built. The process is to aggregate them into a connected subgraph representing an answer to the query. The problem consists of finding the best way to connect these elements. The path that connects two elements might be the shortest path, longest path, or the most semantically meaningfull path. We need to determine which is the most suitable case for our challenge and how we can find it.

Each keyword can be associated with one or more elements in the RDF graph; this will lead to several possible results to the query. We consider that each combination of matching elements containing exactly one element for each keyword is a possible answer to the query. Therefore more than one possible answer can be derived. Since there are several results, it would be useful to provide a ranking method to assess these results. We need to find a ranking method capable of assessing each result based on defined criteria. The problem consists of defining these criteria to determine if there are better results than others. For example, let Q3 = {Dakota, 1974, Al_pacino } be a keyword query, from the graph of figure 3.1 we know that set of matching elements for the keyword "Dakota" denoted by Me_{Dakota} = {south_Dakota, Dakota_fanning}, Me_{1974} = {1974} and Me_{Al_Pacino} = {Al_Pacino}. From these three different sets we can derive two possible com-

binations C1 = {south_Dakota, 1974, Al_Pacino} and C2 = {Dakota_fanning, 1974, Al_Pacino} each one can create different sub-graph result. In order to assess the different results, then a ranking method is needed.

In the rest of this chapter we will target the first challenge in keyword search which is matching the keyword query with the different elements of the RDF graph.

3.3 Problem statement

One of the problems raised by keyword search in RDF datasets is matching the query keywords with the elements of the dataset. If I compare a keyword query to the graph elements, and I did not succeed to find the exact term in the graph this does not mean that there is no answer to the query. Since there might be a concept in the graph that is close in meaning to the keyword query then I can provide an approximate answer the the query.

For example let Q4={2008, film-maker} be the keyword query to be submitted to the graph in figure 3.3, the node having the value "2008" can be exactly matched to the keyword "2008" from Q4. Moreover, we can consider that "director" is matching element to "film-maker" since there is a semantic closeness between the the two concepts.

Our problem can be stated as follows, let us consider:

- G(V,E) an RDF graph data where V is the set of nodes and E is the set of edges
- Q ={$kw_1, kw_2, kw_3, ..., kw_n$} a keyword query with n keywords
- m: $kw_i \rightarrow Me_i$={me_{i1}, me_{i2}, me_{i3}, ...,me_{ij}} $where\ me_i \in G$ and m a mapping function from the query Q to the graph G such as $me_i = kw_i$ or $sim(kw_i, me_i) > threshold$ where sim(x, y) is a similarity function that checks the similarity between the two concepts x and y.

Our goal is to find the set ME={$Me_1, Me_2, Me_3, ..., Me_n$} where:

- Me_i={me_{i1}, me_{i2}, me_{i3}, ...,me_{ij}} is the set of matching elements for the keyword query kw_i and
- $me_i = kw_i\ or\ sim(kw_i, me_i) > threshold$ and
- type of me_i is a node, an edge or sub-graph

Finding the matching elements between the keyword query and the elements of the RDF graph consists of three different problems. The first one is finding the

exact keyword in the graph that can be matched with the query keyword ($me_i = kw_i$). This matching element can be either a node or an edge. For example, let us consider the RDF graph in figure 3.3 and let Q5 = {"2008" ,"Al pacino" and "director"} be a keyword query, the result to this query are the movies and the directors of the movies where "Al Pacino" is one the actors of those movies. As we can see from the graph, the keyword "Al pacino" can be matched to the resource "Al Pacino", the keyword "2008" can be matched to the literal "2008" and finally, the keyword "director" is matched to the property "director"; the problem consists in finding these exact matching elements in the RDF graph and this requires indexing techniques.

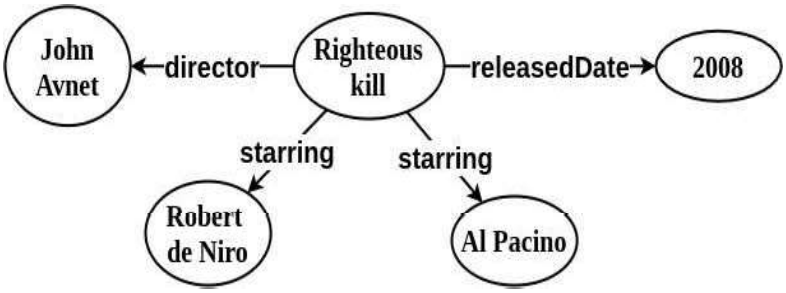


Figure 3.3: An RDF graph

The second problem is how to solve the terminological gap which may exist between the keywords and the data graph elements. Indeed, a keyword of the query itself may not be found in the dataset, but an equivalent element could be found ($sim(kw_i, me_i) > threshold$). For example, let Q6 = {"film-maker" and "John Avent"} be the keyword query, considering the graph in figure 3.3 we can see that there is no exact matching for both keywords, but we know that "film-maker" has a close meaning to "director", therefore the edge labeled "director" should be considered as matching element to "film-maker".

The third problem we might face is matching a subgraph from the graph with an element from the keyword query. The existing approached do not deal with this problem. For example, let Q7 = {"Al Pacino" and "co_Staring"} be the keyword query, we know from the graph of figure 3.3 that "co_Staring" has no matching element, but "Robert De Niro" and "Al Pacino" are co_Staring since they are both acting in the same movie "Righteous Kill". Then the sub-graph connecting the two actors can be considered as a matching element to the keyword "co_Staring". The problem consists in providing a technique that finds the relationship between a keyword query and a sub-graph.

In our work, we are going to deal with three main problems, the first one we address in our work is being able to match a keyword with a sub-graph. In the existing approaches a keyword is generally matched with a node, what we would

like to achieve is to find a matching elements which could be composed of more than a single node. Another problem we consider is how to find beside the graph elements that have exactly the same name as the keyword, the one that are not exactly the same as the keyword but they are close in meaning and this would allow as to overcome the terminological heterogeneity between the elements of the graph and the keyword query.

In this chapter, we introduce a matching process for the keyword query search approach that takes keywords as an input and returns the best matching elements from the graph as an output in order to create the final result of the query. In our matching approach, unlike existing approaches, we match the keywords with all the elements of the graph such as nodes, edges and sub-graphs. We also propose an exact matching between the keyword query and the graphs where we look for elements in the graph that are the same as the keyword we search. We have also proposed an approximate matching where we use an external source of knowledge providing semantic relations to bridge the terminological gap which exists between the keywords and the graph terminology.

For any type of matching we need to be able to retrieve efficiently the different types of graph elements such as nodes, edges or even sub-graph, according to some selection criteria. The approach generally used is indexation which consists in finding the elements in the graph that are exactly matched with the keyword query. In the next section, we will present the indexation used by our approach.

3.4 Indexing an RDF Data Graph

In order to have a fast access to the edges and nodes in the graph, the indexation is needed. Indexing is a process carried offline independently from the user's request. The index is created for the textual content of the document or file. There are many types of indexing, such as the citations index [29] used to store the citation or hyperlinks between documents, the N-gram index [43] is a contiguous sequence of n items from a given sample of text, it is used in text mining. The document-term matrix index is another index used in natural language processing, it consists in describing the frequency of the terms that occur in a collection of documents. Another index type is the inverted index which is an index storing a mapping from a content to its location in the document. This content can be the words in the document.

Since our approach consists in retrieving the matching elements from the RDF graph, then the inverted index is used. It allows fast full-text searches on datasets at scale, for example, in search engines. In our context, we need to search for the existence of the keyword query in the elements of the RDF graph. To create the index, we rely on previous work [63] proposed by Ouksili, H. which aims at

creating an inverted index to search the different RDF graph elements.

3.4.1 Structure of the Index

The inverted index structure consists of two elements: the first one is the keyword and the second one is the occurrence. Generally speaking, the keywords are the set of all distinct words in the text. The occurrence is a document containing information about the keyword, such as the index where this keyword appears or the number of occurrences for the keyword in the test.

In the case of RDF datasets, the keywords are all the distinct words appearing in the nodes and edges of the RDF graph, and the document is a table containing information about the different elements in the graph where the keywords appear. This information contains the type of the RDF graph element where the keyword appears. The graphical representation of the inverted index is described in figure 3.4. The first part is the keywords which represents a list of all the distinct words in the graph, these words are extracted from the different elements of the graph (classes, resources, properties, literals, etc.). Each keyword is linked to one or several tables which are the second part of the index. The table is composed of fields representing information about the elements of the data source. These fields are the part of the graph where the word appears, the type of the element in the graph (resource, literal, property).

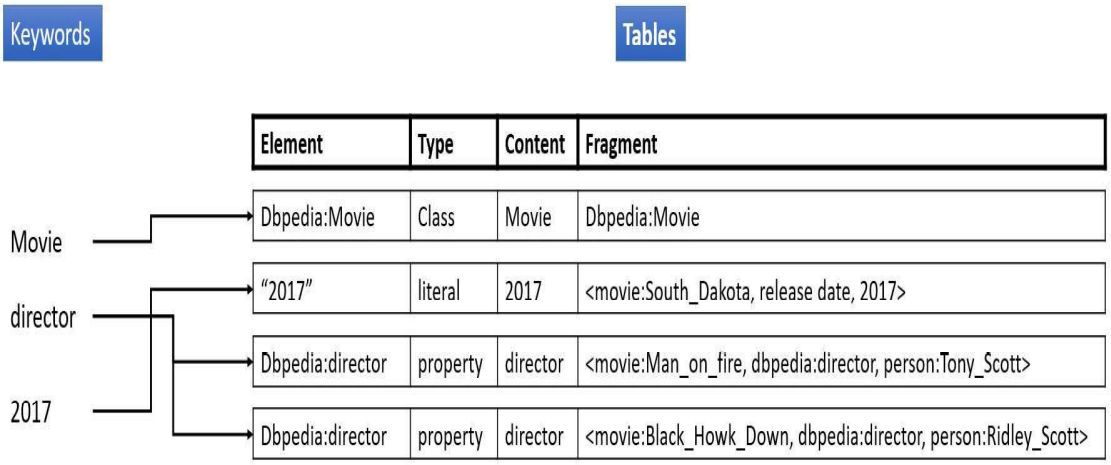


Figure 3.4: Graphical representation for the index for some words in the graph of figure 3.1

The field **Element** contains the word to be indexed from the graph. The **Element** is a URI or a literal depending on the type of the resource it describe, the second field is the **Type** which represents the type of the **Element** which can be a

class, instance, property or literal. The third field is the **Content** which contains the text that was extracted from the graph element, this extraction method differs from one element to another depending on its type. Finally, the last field, which is the **Fragment**, represents the part of the graph corresponding to the keyword, it is a sub-graph that will be used for the construction of the results. For example, the second keyword "2017" is pointing to the second table. From this table, we know that the type of element where "2017" appears is a literal, and also we know that this literal is a value of the properties "released date" that describes the resource "South_Dakota".

3.4.2 Building the Index

Before creating the index, we have used existing text analysis methods. This analysis is a common operation performed during keyword search and it consists of many functions. These functions are used to create a precise index by removing all the unnecessary elements and by extracting the useful texts to be indexed. We have mainly used three tasks, lemmatization, removing the stop words and information extraction functions. Each one of them is suited for a particular type of graph element. Lemmatization is used for the resources, properties and literals, remove stop words are used for the literals, and information extraction is used for the URIs. In the next Paragraph, I will provide an explanation for each one of them.

The first considered text analysis task is lemmatization [70], it consists of replacing each word by its canonical form. By lemmatization, we increase the number of occurrences of each word in the data source and reduce the number of distinct words in the vocabulary, but this process can also decrease the precision for the final results. For example, searching for a name will find all its forms in the graph such as the singular and the plural forms. The second text analysis task used is removing the stop words. Stop words do not add anything to the semantics and they are more used as a connectives, so they do not provide information to the content of the text, an example of stop word is (a, an, the, at, being...) those words increase the size of the index file without being informative. The last text analysis task is the information extraction function[78]. This function is carried out on the local names of the URIs in order to extract the text. Since there are no spaces between the words formulating the URIs, then we need to extract each word alone. For example, the URI <http://www.dbpedia.org/Starring/Denzel_Washington> has a local name Denzel_Washington which consists of two different words, then this function extracts each word alone and adds them to the content field of the indexation table.

The indexation of the elements of the RDF graph differs from one type to another, for example, the process of indexing the classes is different from the

indexation of the literal.

Let us index the literal "2017" in the RDF graph of figure 3.1, the indexation is in the table of figure 3.4. In the content field the extracted text is "2017", the type is literal that describes a resource. If this literal is selected as a matching element, this literal should be selected with the resource describing it. In other words, each literal l is a value for a property P that describes a resource r, then the triple <r,p,l> is added to the fragment field of the table and to be selected as a matching element l. For example, the literal "2017" describes the released date of the movie "South Dakota" (second record in the table of figure 3.4) then <movie: South_Dakota, property: released date, "2017"> is the fragment to be added to the table.

In the case of a property, which corresponds to the last two records in the table of figure 3.4, we can see in the content field the extracted text "director" that is used as a property to a resource. If this property is selected as a matching element, then it will be added to the fragment section of the table along with the resource and the object connected to it. For example if the property "director" is used to show that "Tony Scott" is the director of the movie "Man on fire", then <movie: Man__on_fire, property: director, person: Tony_Scott> is added to the table.

In the case of a resource (class or instance), we use the local name of the URI in the content field. The fragment field will contain the URI of this resource. For example, the first record of figure 3.4 is a class, then the local name from the URI, which is "Movie" is extracted to be added to the content field, and the URI "Dbpedia:Movie" is added to the fragment field.

Using the index will help us to retrieve firstly the elements in the RDF graph that are exactly matched with the keyword query. In the next section, we will present the external knowledge which will be used in our approach.

3.5 External Knowledge used to Enhance the Matching Process

In our matching algorithm, we will use external knowledge to enhance the matching elements and to solve the heterogeneity between the query keyword and the element of the RDF graph. There are two different kinds of knowledge that are used in our approach. The first kind of knowledge we can use are the semantic relationships provided by the digital resources, they help in finding the semantic relations between the elements in the graph and the query keyword. The second external knowledge is the patterns that match a path to an equivalent property in the RDF graph. They will help us to identify the approximate matches according to their semantic closeness with the considered keyword. In this section, we will

present each one of them.

3.5.1 Semantic relations to bridge the terminological gap

The first source of knowledge considered in our approach is the use of some external knowledge stored in online linguistic resources such as WordNet [1], which is a large lexical database of English providing numerous semantic relations among concepts. WordNet's structure makes it a useful tool for computational linguistics and natural language processing. This lexical database can help in solving the terminological gap which may exist between the concept and the data graph elements.

There are many semantic relations between two concepts, some of them are very useful to help us find the best matching elements between the query and the dataset. In our work, we have considered the following relations to search for matching elements.

- Synonymy: a concept c is a Synonym of another concept c' if c means exactly or nearly the same as c'. For example, the concept Couch is the synonym of the concept sofa.
- Antonymy: a concept c is a Antonym of another concept c' if c means opposite in meaning to c'. Increase is an antonym of decrease.
- Hyponymy: a concept c is a Hyponym of another concept c' if c is denotes a subcategory of a more general concept c'. For example, the concept meal is the Hyponym of the concept lunch.
- Hypernymy: a concept c is a Hypernym of another concept c' if c is superordinate to c'. For example, the concept fly is the Hypernym of the concept travel.
- Substance meronym: a concept c is a Substance meronym of another concept c' if c' is Substance to c. For example, the concept water is the substance meronym of the concept Oxygen.
- Part meronym: a concept c is a part meronym of another concept c' if c' is part of c. For example, the concept table is the part meronym of the concept leg.
- Member meronym: a concept c is a Member meronym of another concept c' if c' is Member of c. For example, the concept faculty is the member meronym of the concept professor.

[1] https://wordnet.princeton.edu/

- Substance of holonym: a concept c is a Substance holonym of another concept c' if c is Substance of c'. For example, the concept gin is the substance holonym of the concept Martini.

- Part of holonym: a concept c is a Part holonym of another concept c' if c is part of c'. For example, the concept window is the part holonym of the concept building.

- Member of holonym: a concept c is a Member holonym of another concept c' if c is Member of c'. For example, the concept copilot is the member holonym of the concept crew.

- Troponym: A verb v is Troponym to verb v' is verb v expressing a specific manner elaboration of v'. For example, the verb walk is the troponym of the verb stroll.

Let us consider an example of query keyword Q9 ={"board", "2008"} and submit it to the graph of figure 3.5.

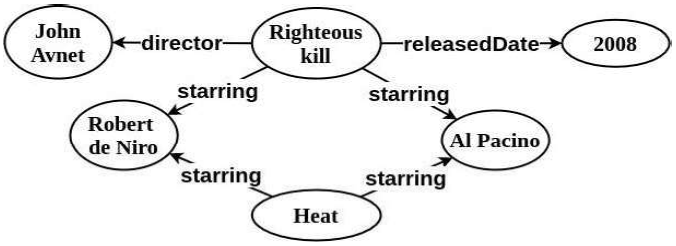


Figure 3.5: An RDF graph about movie

We can observe that the keyword "board" is not in the graph, which means that there is no exact matching element for "board". Assume that an external resource stating that "board" is the meronym of "director" is provided, then the property "director" will be matched to the keyword "board". We can see that having these semantic relations will enhance the matching between the query keyword and the RDF graph elements.

3.5.2 Equivalence between properties and paths

Let us consider the query keyword Q10 ={"co-starring", "Al Pacino"} issued on the graph of figure 3.5. There is no element that can be matched with the keyword "co-starring", but from the graph, we know that "Al Pacino" and "Robert de Nero" are co-starring in two different movies "Heat" and "Righteous kill". We can then deduce that a property "co-starring" can be matched to the path connecting two

different actors through a common movie and by using the relation starring. Using this equivalence between a property and a path can enhance the matching process and help in bridging the terminological gap.

In our approach, we have used the patterns proposed by Ouksili, H. [63], these patterns express the equivalence between a property and a path in the RDF graph. For example, if "grandson" is one of the query keyword issued on an RDF graph, and there is no matching element for this keyword, but there is a path P1 = [(X son Y), (Y son Z)] in the graph, and we know that "grandson" is equivalent to this path then we can consider P1 is the matching element to the keyword "grandson". In this section, I will present the definition of patterns, their type, and how we can use them in our approach.

The definition of a patterns relies on property expression and path expression. I will first give the definition of property expression and path expression and then define the patterns.

Definition 5. Property Expression

A property expression is a triple (X, p, Y), where X and Y are either resources, literals or variables and p is a property.

For example, (South Dakota,releasedDate, 2017) is a property expression that represents the release date of the South Dakota movie.

Definition 6. Path Expression

A path expression describes a relation between resources. The relation is a path (a sequence of properties) between two resources. More formally, a path expression is defined as a triple (X, P, Y), where X and Y are either resources or literals or variables and P is a SPARQL path expression [89].

The SPARQL path expression can be expressed as a sequence of properties. This sequence is created by using the property path expression. It is similar to a string regular expression but over properties, not characters. For example, "^" express the inverse of the property and "|" express an alternative between two properties.

For example, (X, (^son|son) , Y) represents all the resources related to X represented by a sequence of properties the inverse of son and son. For example, from the graph of figure 3.1 we can see "Ridley Scott" and "Tony Scott" are related by both the property son, and it's inverse which means that they are brothers.

Definition 7. A pattern is a pair [propertyEx, pathEx] where propertyEx is a property expression and pathEx is a path expression. It represents the equivalence between one property expression and one path expression.

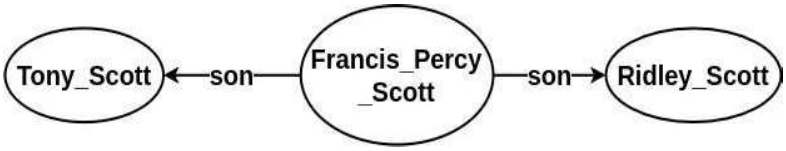


Figure 3.6: Path equivalent to the "Brother" property

For example, in order to express the equivalence between the property "brother" and the path expression presented in the figure 3.6, the following pattern is defined:

[(X, brother , Y) , (X, ^son / son , Y)]

In other words we can say that the property brother is equivalent to the path having the properties inverse of the property son and the property son.

Three different types of patterns are proposed in [63], (i) the sameResult patterns, (ii) SameProperty patterns and (iii) patterns using the domain properties. In the following, we will provide a definition for each one of them.

3.5.2.1 SameResult Pattern

The SameResult pattern is a pattern created to describe the semantic closeness between two nodes of the graph. Consider that we have two different nodes n1 and n2, and assume they are related by a sequence of owl:sameAs properties. We can say that these nodes contain equivalent information as they represent the same object. We can define generic patterns by using the properties from subClassOf, type and sameAs from the RDFS, RDF and OWL, respectively. The general form of these patterns can be one of the two following:

[(X, pattern : sameResult , Y) , (X, (owl : sameAs | ^ owl : sameAs) $^{+}$, Y)]

[(X, pattern : sameResult , Y) , (X, rdf:type ?/ rdfs:subClassOf * , Y)]

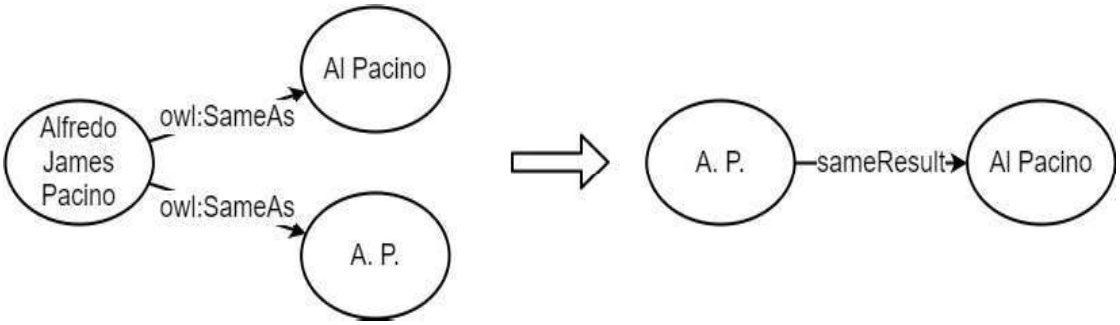


Figure 3.7: Two Nodes are equivalent by using SameResult Pattern

Let us consider the graphs of figure 3.7 as an example, the graph on the right expresses that "Alfredo James Pacino" is connected to "Al Pacino" and "A. P." by "owl:SameAs" property. We can observe that "A. P." has the same result as "Al Pacino" since they are connected by a path composed of owl:SameAs properties.

3.5.2.2 SameProperty Pattern

SameProperty pattern is used to describe the equivalence between edges (properties). The properties owl:inverseOf, owl:equivalentProperty and rdfs:subPropertyOf are used in this type of patterns and its general form is shown below:

[(X, pattern : sameProperty , Y) , (X, (owl:equivalentProperty | ^ owl:equivalentProperty | rdfs:subPropertyOf) $^{+}$, Y)]

Figure 3.8: Two properties are equivalent by using SameProperty Pattern

Let us consider the graphs of figure 3.8 as an example, from this graph and from the general form of the sameProperty pattern we can deduce that "starring" is the same property as "Acted In".

3.5.2.3 Property Pattern

The third type of pattern is the patterns created by using specific domain properties. Let us consider the following pattern:

[(X, p , Y) , (X, P , Y)]

For example if p is "co-starring" and P is starring /^ starring then the pattern [(X, co-starring , Y) , (X, starring /^ starring , Y)] expresses that if there are two resources representing two different actors starring in the same movie, then they are co-starring. More details on the definition of the patterns can be founded on [63].

Let Q11 ={"John Avent", "co-starring"} be the query keyword to be issued on the graph of figure 3.5 and let the path in figure 3.9-a be equivalent to the property on the figure 3.9-b. We can observe that the keyword "co-starring" is not in the graph, which means that there is no exact matching element for the keyword

"co-starring", but we can see that "Robert De Niro" and "Al Pacino" are acting in the same movies "Righteous kill" and "Heat" and therefore they should return as a matching element to the keyword "co-starring". We can deduce that the path in figure 3.10 a possible matching element for the query keyword "co-starring".

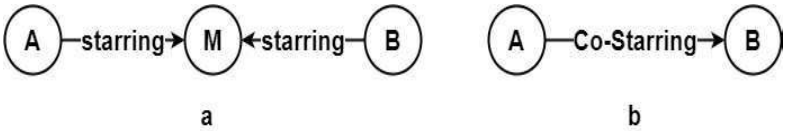


Figure 3.9: Property "co-starring" equivalent to a path

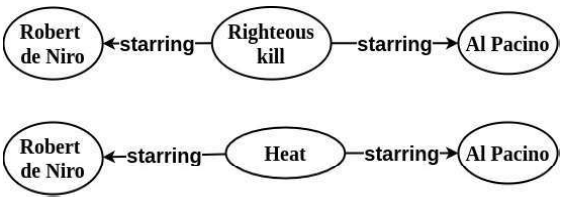


Figure 3.10: Matching Elements for the keyword "co-starring" in Q11

This kind of information can help in finding the matching elements for the query Q11. In the next two sections, we will discuss the use of semantic relations and the patterns in the matching process.

3.6 Enhancing the Matching Process with Semantic Relations

In this section, we will present the use of some external knowledge to enhance the matching process. In our work, we have used WordNet [2] as a source for the semantic relation. This lexical database will be used to help in finding the matching elements between the keyword queries and the graph elements using the provided semantic relations presented in section 3.5.1. Using the semantic relations will bridge the gap between the query keyword and the elements of the RDF graph. We have first divided the semantic relations provided by WordNet into two sets:

- Relationships Indicating Equivalence, used to find whether two concepts are equivalent; these semantic relations are: synonym and antonym.
- Relationship Indicating Closeness, which express other relations than equivalence or generalization; the considered concepts are not synonyms, but are

[2] https://wordnet.princeton.edu/

linked by some other semantic relation, which could be hypernym, hyponym, meronym or holonym.

In the next subsections, we will present the two sets of relations which are *Set of Relationships Indicating Equivalence* and the *Set of Relationship Indicating Closeness*, and the use of each one of them in our matching process.

3.6.1 Finding Equivalent Matching Elements

In this section we will search for concepts in the RDF graph that are equivalent to another concepts in the RDF graph by using equivalence relationships. The equivalence relations are derived by using three different types: exact, synonym, and antonym. We will present each one here after.

3.6.1.1 Exact Matching

An exact match is a graph element, such as a node or an edge, that contains one of the query keyword in their content; these elements can be extracted by using the index. Let us first illustrate with an example, suppose that Q4= {2017, director} is a query keyword. If a user issues this query keyword on the data graph of figure 3.1, then the matching elements for the keyword "2017" will be the elements of the graph that have "2017" in there content. From the table in figure 3.4 we can observe that the literal describing the release date of the movie "South_Dakota" has "2017" in its content. Since the object is the value of the property that describes a specific subject, then the triple <subject, property, object> will be considered as a matching element. In our example, the object "2017" is the value of the property "released date" that describes the subject "South_Dakota", then the triple <"South_Dakota", "released date", "2017"> will be considered as a matching element for the keyword "2017". As for the second query keyword, "director", it can be matched to the property "director", and from the index table in figure 3.4 the matching element of the keyword "director" are the two triples having "director" as their property.

The different cases of exact matching depend on the types of the graph elements and they are presented as follows: Let kw_i be the query keyword and let me_i be one of the elements in the graph that can be exactly matched to Kw_i then:

- If me_i is of type resource (class or instance) then the matching element to be matched with Kw_i is the resource itself (me_i).

- If me_i is of type literal, then the matching element that will be matched to Kw_i is the triple <subject, predicate, me_i>

- If me_i is of type property, then the matching element that will be matched to Kw_i is the triple <subject, me_i, object>

3.6.1.2 Synonymy Relations

The second type is a synonym; it is to find a synonym relation between the query keyword and the graph elements. Let kw_i be the query keyword and let me_i be the element in the graph that can be matched to Kw_i such that me_i is synonym for Kw_i. For example, lets us consider the query keyword Q5={performing, film maker} be a query keyword. If a user issues this query keyword on the data graph of figure 3.1, then the answer will be empty. However, knowing that "performing" is a synonym for "starring" and "film-maker" is a synonym to "director", we can see that there are in the graph some elements that can be matched to the query.

3.6.1.3 Antonymy Relations

The last type is the antonym, where one of the query keyword and one of the graph elements are both antonyms to the same concept. For example, the search for matching elements using the antonymy relation is done by issuing the following query to Wordnet.

```
SELECT ?y
WHERE {$kw_i$ Antonym\_to ?x.
?x Antonym\_to ?y.}
And $kw_i$ different from ?y.
```

Finding equivalent matching elements consists of searching for elements in the dataset by using the equivalent relations. Consider a query keyword Q=$\{kw_1, kw_2, kw_3, ...kw_n\}$. For each keyword kw_i, we perform the task of searching the knowledge base for the equivalent of the considered keyword. This consists of querying WordNet to extract the semantic relations (synonyms and antonyms) involving kw_i and find the exact matching of these semantic relations and the keyword kw_i in the data graph.

The first goal behind using the equivalent relations is to bridge the terminological gap between the query keyword kw_i and the RDF graph G. Second goal is to enrich the matching elements for kw_i by extracting semantically related elements even if there exists an exact matching for the keyword kw_i.

Algorithm 3 describes the identification of equivalent matching elements. Let us start by presenting the notations used in the algorithm. Let K=$\{kw_1, kw_2, kw_3...kw_n\}$ be the query keyword, EM(kw_i) a function to extract the exact matching elements for kw_i in the dataset (these elements can be literals, instances, classes, properties, etc.).

The semantic relations between the query keyword kw_i and WordNet are extracted by using the following functions; Synonym(kw_i) is used to extract the set of synonyms for the query keyword kw_i and Antonym(kw_i) is used to extract the set of antonym for the query keyword kw_i.

The exact matching between kw_i and the graph elements is first extracted (line 4), then WordNet is queried to extract the sets S and A of synonyms and antonyms respectively(lines 6-10) for each keyword kw_i in the query. The dataset is accessed to check if there is a matching element for the words in the sets S and A in the edges and nodes of the graph (lines 8-12). At the end, for each query keyword, a set of matching elements is returned.

Algorithm 3 Finding the equivalent matching elements

1: **procedure** SEARCHEQUIVALENTMATCHES(kw_i)
2: $MatchingElements = EM(kw_i)$
3: $S \leftarrow synonym(kw_i)$ ▷ S is set of synonyms to the keyword kw_i
4: **for** each s_j in S **do**
5: $MatchingElements \leftarrow MatchingElements \cup EM(s_j)$ ▷ extract the matching elements for the synonym s_j and add them as matching element to the query keyword kw_i
6: **end for**
7: $A \leftarrow Antonym(kw_i)$ ▷ function Antonym take kw_i as input and return keyword similar to it as output after executing the following query {kw_i Antonym to ?x, ?x antonym to ?y.} and kw_i different from ?y
8: **for** each a_j in A **do**
9: $MatchingElements \leftarrow MatchingElements \cup EM(a_j)$
10: **end for**
11: Return $MatchingElements$
12: **end procedure**

Consider the example of the keyword query Q6={film maker, Robert_De_Niro} to be submitted to the graph of figure 3.1, the resource "Rober_De_Niro" will be considered as an exact matching to the keyword "Rober_De_Niro". The keyword "film maker" has no exact matching element. The search for equivalent matches is processed on the query Q6, and we query WordNet using the following SPARQL query

```
Select ?Synonym where
{<http://www.w3.org/instances/"film maker">
<http://www.w3.org/schema/SynonymOf> ?Synonym.}
```

As a result S = {author, director, ...} is the set of synonyms for "film maker", then an exact matching is performed between the elements in the set S and the

RDF graph. We can observe that the word "director" from the set S has an exact matching with the property "director" in the RDF graph. Consequently all the "director" properties are selected to be the matching elements to the keyword query "film maker".

3.6.2 Finding Close Matching Elements

To find the equivalent matching elements, we have used some semantic relations from section 3.5.1. In some cases we could not find the equivalent matching elements, but there are other semantics that can be used to infer that there exists some kind of closeness between the RDF graph elements and the keyword query. These elements can be derived by using semantic relations such as hypernymy.

The idea of finding the elements that are closed in meaning to the keyword query is similar to search for equivalent matching elements by using an external knowledge source. In our case, we have considered semantic relations in Wordnet. The semantic relations we are interested in for finding the elements that are closed in meaning to keyword query are hypernyms, hyponyms, holonyms and meronyms. For each keyword kw_i, we query WordNet to search for one of the semantic relations.

The relations of hypernym, hyponymy, holonymy and meronymy do not express equivalence but express some sort of closeness between two concepts. If a meronymy relation is found between kw_i and a concept c, and if c has an exact match in the dataset, i.e., a graph element labeled c, then this latter is a close concept to kw_i. Indeed, if c does not represent an equivalent concept, it still represents a close concept as c is part of kw_i because the two are linked by a meronymy relation, according to Wordnet.

The algorithm for finding the matching elements that are close in meaning to keyword query is presented in 4. WordNet is queried to find the semantic relations by searching for the hypernym, hyponymy, holonymy and meronymy for query keyword kw_i. For each concept c related to kw_i by one of these relations, we search for elements labeled c in the dataset; these elements are added to the set of matching elements for the keyword kw_i.

Let us consider the query keyword Q4={1974, board}, if we issue this query on the graph of figure 3.1, the keyword "1974" will be matched to the RDF triple <The_Godfather_2, released date, "1974"> since "1974" is a literal. There is neither Exact match nor equivalent match for the keyword "board", then searching for close matches is executed to query WordNet to search for close semantic relation for the keyword "board". The SPARQL query below is one of the queries that can be used to search for close elements to the keyword "board".

```
Select ?Meronym where
```

Algorithm 4 Finding the Elements that are Close in Meaning

1: **procedure** SEARCHCLOSEMATCHES(kw_i)
2: $MatchingElements = \phi$
3: $semanticKeywords \leftarrow Hypernym(kw_i) \cup Hyponym(kw_i) \cup Meronym(kw_i) \cup Holonym(kw_i)$ ▷ $semanticKeywords$ is set of Hypernyms, Hyponyms, Myronyms and Holonyms to the keyword kw_i
4: **for** each h_j in semanticKeywords **do**
5: $MatchingElements \leftarrow MatchingElements \cup EM(h_j)$
6: **end for**
7: Return $MatchingElements$
8: **end procedure**

```
{<http://www.w3.org/instances/"board">
<http://www.w3.org/schema/MeronymOf> ?Meronym.}
```

The result of the meronym for the word "board" are shown in the set M = {director, board member, etc.}. We need to check if the elements in the set M can be matched to the elements of the RDF graph. We can observe that the word "director" from the set M can be matched to the "director" property in the RDF graph, consequently all the "director" properties are selected to be the matching elements for the keyword "board".

3.7 Using Patterns in the Matching Process

Besides the use of the semantic relations provided by an online resource to improve the matching process, we also use the notion of pattern defined in [63] and presented in section 3.5.2. The patterns are used to enrich the set of matching elements for the keyword query kw_i by extracting paths from the RDF graph that can be matched to kw_i. Suppose that we have a property p_i that is equivalent to a path P, and suppose that one of the keyword query kw_i is p_i, then during the matching process, we search for P in the RDF graph and consider it as a matching element to the keyword kw_i. As defined in section 3.5.2, a pattern is a pair having the following structure: [propertyEx, pathEx] where propertyEx is a property expression and pathEx is a path expression.

The use of the patterns requires first an evaluation stage which is done before querying the RDF graph. It consists of retrieving for each pair [propertyEx, pathEx] all the paths from the RDF graph that are equivalent to the property (propertyEx) using the pattern (pathEx). This is done by the using the mean of a SPARQL query. For each property propertyEx there exists a list of paths = $\{path_1,$

$path_2$,.. $path_n$} such that $path_i$ is equivalent to propertyEx. The results are stored in an external knowledge base to be used in the matching process. During the matching process and after sending the keyword query, for each query keyword kw_i, we check if it is matches one of the properties in the external knowledge (propertyEx). If it is matched, then we select the list of paths that are equivalent to the property as matching elements for the keyword query kw_i.

In section 3.5.2, we have three different types of patterns; SameResult Pattern, SameProperty Pattern and Property Pattern. The type of matching element for each type of pattern is presented as follows:

- If the pattern is a SameResult Pattern, then the type of matching element is one node.
- If the pattern is a SameProperty Pattern, then the type of matching element is one edge.
- If the pattern is a Property Pattern then the type of matching element is a subgraph.

Let Q12 ={"Brother", "co-starring"} be the keyword query to be applied on the graph of figure 3.1, and let p1 = [(X, brother , Y) , (X, son /^ son , Y)] and p2 = [(X, co-starring , Y) , (X, starring /^ starring , Y)] be the external knowledge that represents two patterns; the first one expresses the property "brother" is equivalent to a path consisting of property "son" followed by its inverse, and the second pattern represents an equivalence between the property "co-starring" and a path which consists of the property "starring" followed by its inverse.

We can observe that the keyword "brother" is not in the graph, which means that there is no exact matching element to this keyword "brother". But from the graph we know that "Ridley Scott" and "Tony Scott" are the sons of "Francis Percy Scott", then the path in figure 3.11 is found in the graph and is selected to be the matching element for the keyword "brother". There is no exact matching element for the keyword "co-starring", but also from the graph we have that "Dakota fanning" and "Al Pacino" are acting in the same movie "Once upon a time in Hollywood" so they are acting together. We can deduce that the paths in figure 3.12 are equivalent to the keyword "co-starring". Hence these paths are selected as matching elements for this keyword.

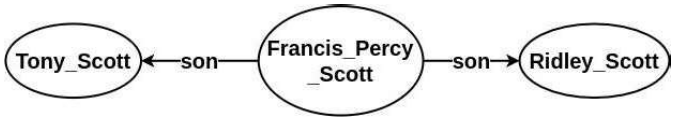


Figure 3.11: Matching Element for the keyword "Brother" in Q12

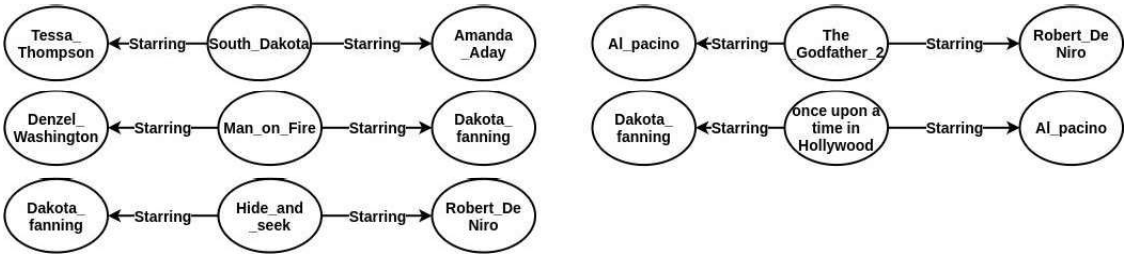


Figure 3.12: Matching Elements for the keyword "co-starring" in query Q12

3.8 Semantic relations and pattern in the Matching Process

After presenting the use of the semantic relations and the patterns in the matching process, we will present in this section the overview and algorithm of the matching process used in our keyword search approach.

The general principle of the matching process is to find for each query keyword kw_i the elements that can be matched to it in the graph. These matching elements can be found by using the exact matching, using one of the equivalent relations, or using the patterns. If using the equivalent relations and equivalent patterns failed to find matching elements for the query keyword kw_i, then the searching for matching elements using close relations is executed to find the graph elements that are closed in meaning to kw_i. The overview of this process is presenting in figure 3.13.

The process takes the keyword query Q = $\{kw_1, kw_2, ..., kw_n\}$ as input and returns a set of lists Me = $\{Me_1, ..., Me_n\}$ where $Me_i = \{me_{i1}, me_{i2}, ..., me_{im}\}$ is the list of matching elements me_{ij} extracted from the graph G which corresponds to the query keyword kw_i. Our matching process consists of three phases: (i) searching using equivalent relations, (ii) searching using equivalent patterns and (iii) searching using close relations. For each keyword query kw_i, the equivalent relations and equivalent patterns are search for in the RDF graph; if no matching element is found then we search for matching elements by using the close relations.

The first phase, which is **searching using equivalent relations** consists of two parallel tasks. The first one is comparing the query keyword with the graph elements (resources, classes or properties) to identify the matching elements. We refer to this task as exact matching, and it uses the index as defined in section 3.4. The second task is searching for synonyms and antonyms, which consists of querying WordNet to find the equivalent semantic elements that are matched to the keywords of the query.

The second phase is **searching for patterns**. This task is executed after the first one, and it uses the patterns and the paths equivalent to these patterns, and

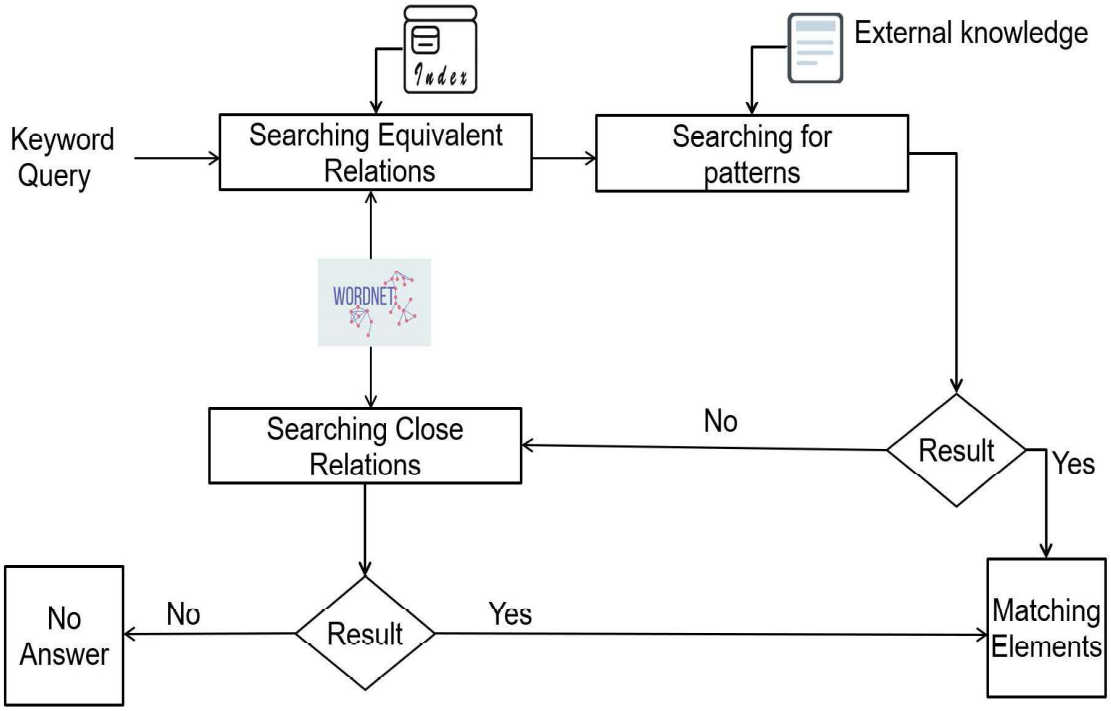


Figure 3.13: Overview of the Matching Elements Process

they are expressed as external knowledge. This part consists of finding the paths in the graph that matches to one of the query keyword kw_i and adding this path to the matching element of kw_i.

The last phase is **searching using close relations** which is executed if searching using equivalent relations and searching for patterns failed to find a matching element for the keyword query kw_i. This task also uses WordNet to search for close semantic elements that are matched to the query keyword kw_i.

The matching process is presented in algorithm 5, with the following notations: EM(kw_i) is a function which extracts the exact matching elements for kw_i from the RDF graph (these elements can be literals, instances, classes or properties). The semantic relations between the keyword query kw_i and WordNet are extracted using two functions, (i) SearchEquivalentMatches(kw_i) (algorithm 3) which returns the equivalent elements to kw_i by using the relation synonymy and antonymy, and (ii) SearchCloseMatches(kw_i) (algorithm 4) which returns elements that are closed to the keyword kw_i by using the relations hypernymy, hyponymy, meronymy and holonymy.

For each keyword kw_i in the query, all the exact matching elements are retrieved by searching the index file (line5). Then WordNet is queried to extract the equivalent semantic elements involving kw_i; as a result, we obtain a set ES_i

such that: $ES_i = \{c_{ij} | \exists\, equ - rel(kw_i, c_{ij})\}$ where equ-rel(kw_i,c_{ij}) is one of the two equivalent relations: synonymy or antonymy between kw_i and c_{ij}. For each concept c_{ij} in ES_i, we check the index table to find exact matching elements in G and add them to the set of matching elements $Me_i(kw_i)$ (line 6-9).

The next step consists in finding the patterns in the graph that matches kw_i. This is done by using the PatternMatching(kw_i) function that searches if there exists a pattern that is equivalent to kw_i in the external knowledge base, then extract this pattern from the graph and added it as a matching element for each keyword kw_i (line 10). If no matching element has been found in the two previous steps, then a search for close matching elements is performed (line 11). In this phase, WordNet is queried to extract elements that are closed to kw_i by using the close relations; as a result, we obtain a set CS_i such that: $CS_i = \{c_{ij} | \exists\, clo - rel(kw_i, c_{ij})\}$ where clo-rel(kw_i,c_{ij}) is one of the following close relations: hypernymy, hyponymy, meronymy or holonymy between kw_i and c_{ij}. For each concept c_{ij} in CS_i, we check the index table to find exact matching elements in G; these elements are added to the set of matching elements for the keyword kw_i (line 12-15). At the end of the execution, for each keyword, the algorithm returns a set of matching elements.

Algorithm 5 Matching the Query Keywords with the RDF Graph

```
1:  keywordQuery = {kw_1, kw_2, kw_3, ...kw_i}
2:  procedure MATCHING(keywordQuery)
3:      for each keyword kw_i in keywordQuery do
4:          MatchingElements = ∅
5:          MatchingElements = MatchingElements ∪ EM(kw_i)
6:          SemanticRelationsEquivalent ← SearchEquivalentMatches(kw_i)
7:          for each s_j in SemanticRelationsEquivalent do
8:              MatchingElements = MatchingElements ∪ EM(s_j)
9:          end for
10:         MatchingElements = MatchingElements ∪ PatterMatching(kw_i)
11:         if MatchingElements is empty then
12:             SemanticRelationsClose ← SeachCloseMatches(kw_i)
13:             for each s_k in SemanticRelationsClose do
14:                 MatchingElements = MatchingElements ∪ EM(s_k)
15:             end for
16:         end if
17:         hashmap.add(kw_i, MatchingElements)
18:     end for
19:     Return hashmap
20: end procedure
```

Let us consider Q13= {brother, Washington, outline, board}, a keyword query issued on the data graph of figure 3.1. For each keyword we will get a set of matching elements.

As we can observe from the data graph, the keyword "Washington" can be matched exactly with two nodes, the first is the actor "Denzel Washington" and the second one is a literal describing the movie "The God Father 2". These nodes are extracted by using the index created in section 3.4.

There is no exact matching for the keyword "outline" but from WordNet, we can find that "outline" is a synonym to "abstract". The property abstract connecting the movie "The God Father 2" and the literal describing this movie is then considered to be a matching element to the "outline" keyword.

The second phase is searching for pattern. As we can observe from the data graph, there is no graph element corresponding to the keyword "brother"; searching for pattern enables us to infer that "brother" is equivalent to the graph shown in figure 3.14, which can therefore be selected as a matching element for the keyword "brother". All the matching elements are presented in figure 3.15.

Figure 3.14: Example of Equivalent Path for the Keyword Brother

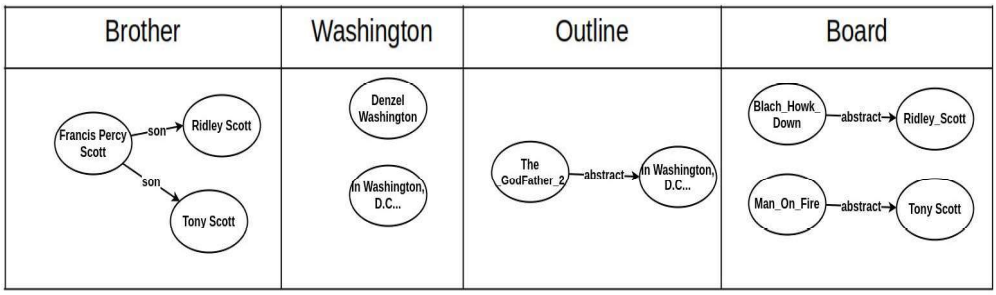


Figure 3.15: Matching Elements for each Keyword in Query Q13

After processing the first two phases to extract the matching elements for the keywords in Q13, we deduce that it fails to find a matching element for the keyword "board"; for this reason, the third phase (searching for closed relations) is processed. After searching WordNet for elements that are closed to the keyword "board" by using the close semantic relations, we can find that the keyword "director" is a meronym of "board". All properties "director" are then selected as matching elements of the keyword "board".

In the next section, we are going to present some experiments to show the effectiveness and the efficiency of our approach.

3.9 Experimental evaluation

This section describes our experiments to validate the performances of our approach. Our goal is to observe the performance of using WordNet as an external knowledge to fill the gap between the keywords and the dataset terminologies with various keyword queries

All the experiments have been done on Intel Core i7 with 32GB RAM. Our approach is implemented in Java. We have used the Jena API for the manipulation of RDF data. For indexing and searching the keyword query, we have used the Lucene API. The Jung API is used for graph manipulation and visualization.

We have used two datasets: AIFB and DBpedia. AIFB is a dataset containing data taken from the AIFB Institute at Karlsruhe University. It describes entities of research community such as persons, organizations, publications (bibliographic metadata) and their relationships. The dataset contains 8281 entities and 29 233 triples. DBpedia is a project aiming to extract structured content from the information created in the Wikipedia project. The extracted data is related to movies, titles, actors, directors, released data, and other properties. This dataset contains 30 793 triples.

We have tested and compared our keyword search approach both with and without the use of external knowledge. We will refer to the approach with external knowledge as semantic based matching since we use semantic relations. We will refer to the approach without external knowledge as the basic approach.

The query size was between 3 and 8 keywords. Some of the keywords have exact matching with graph elements, and others are closed in meaning to some elements in the dataset. We have also used some keywords that can be matched to a path according to the pattern we have defined in section 3.5.2. The total number of queries was 20 queries (10 for each dataset) in order to cover all the different WordNet semantic relations during the matching stage. Table 3.1 shows some examples of queries keyword, the number of nodes and edges containing these keywords in the data graph, and the type of semantic relations between the keywords and the data graph after querying WordNet. For example, Query 1 (carole_lombard, 5940, theoretical and edwin) consists of 4 keywords. These keywords appear in the data graph (nodes and edges) 14, 18, 0 and 10 times respectively. The keyword "theoretical" is not in the dataset, but it is closed to the concept "academic" since there is a hypernymy relation between "academic" and "theoretical" in WordNet.

As we can observe from figure 4.22, the execution time increases when the

Query Number	keyword Query	number of appear Nodes/edges	WordNet Relations
Q1	carole_lombard 5940 theoretical edwin	14 18 0 10	hypernym
Q2	swedishfilms lagercrantz 1997 plosion	2 4 97 0	hyponym
Q3	psychiatric hampshire ziskin system	2 1 4 0	hyponym
Q4	bring cut mind	0 1 0	antonym hypernym
Q5	solar_system secondary vocabulary	0 0 0	holonym synonym meronym

Table 3.1: Example of Keyword Queries

number of keywords increases for both datasets. We can also see that the execution time for AIFB is greater than the execution time for DBpedia because the size of data in AIFB is greater than the size in DBpedia.

The number of matching elements also affect the execution time; for example Q1 {carol_lombard, 5940, theoritical and edwin} in table I takes 4.91 sec (fig. 3.17) and contains 4 keywords while Q6 {poor, 1990, mind and Ellen_Burstyn} needs 31.57 sec (fig. 3.17) to be executed: the two queries contain the same number of keywords (4), but the difference in the execution time is due to the number of matching elements. In Q1, the four keywords appear 14, 18, 0 and 10 times respectively as we can see from table 3.1, but the keywords in Q6 appear 20, 258, 0 and 2 times respectively. We can deduce that as the number of matching elements for the query keyword increase, the execution time increase.

The differences in terminology between the query keyword and the dataset also affect the execution time. Consider the queries Q3 and Q4 in table 3.1; Q3 consists of 4 keywords, and these keywords appear 2, 1, 4 and 0 times respectively in the dataset while Q4 has 3 keywords and these keywords appear 0, 1 and 0 times respectively in the dataset. But the execution time for Q4 (7.76 sec) is greater

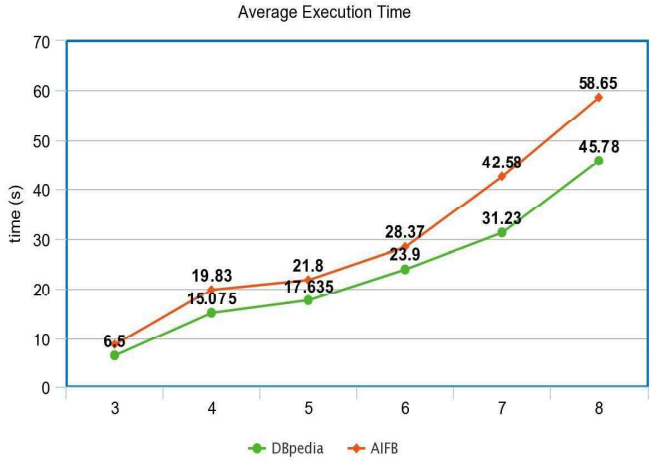


Figure 3.16: Average Execution Time According to the Size of the Query

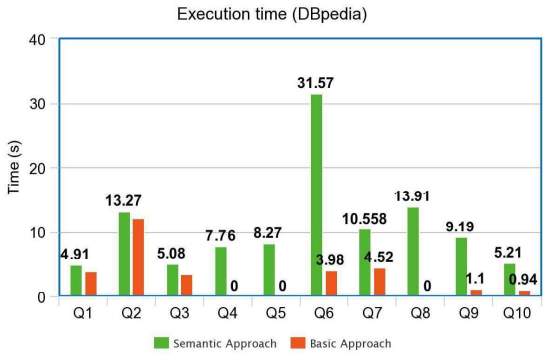


Figure 3.17: Execution Time for Each Query over DBpedia

than the execution time of Q3 (5.08 sec); this is because Q4 requires access to WordNet two times to search for semantic relations involving the keywords, while Q3 requires only one access.

query	Q1	Q2	Q3	Q4	Q5	Q6	Q7	Q8	Q9	Q10
№ of results (semantic based matching)	2	173	7	8	50	250	110	132	78	34
№ of results (basic approach)	1	150	1	0	0	54	25	0	2	2

Table 3.2: Number of Results for each Keyword Query(DBpedia)

As we can observe from tables 3.2 and 3.3, the number of results increases when WordNet is used during keyword search because using WordNet increases

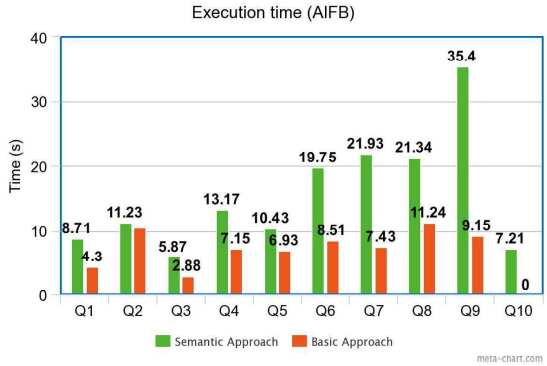


Figure 3.18: Execution Time for Each Query over AIFB

query	Q1	Q2	Q3	Q4	Q5	Q6	Q7	Q8	Q9	Q10
№ of results (semantic based matching)	9	10	21	25	32	38	20	18	25	5
№ of results (basic approach)	3	4	10	7	12	22	14	7	13	0

Table 3.3: Number of Results for each Keyword Query(AIFB)

the number of matching elements. This means that the number of combinations and, therefore the number of results both increase.

To check the effectiveness of our approach, we have used 10 queries from the tables 3.2 and 3.3 and asked three users to check the top-k results for each query and give the number of relevant results to calculate the precision at k as follows:

$$P@K = \frac{NumberOfRelevantResults}{K} \tag{3.1}$$

According to table 3.4, we can see that P@K varies between 0.82 and 0.96 for the semantic based matching. These results are better than those obtained using the basic approach, where P@K varies between 0.62 and 0.8; this means that the results achieved with the semantic based matching were more accurate according to the users.

Data	AIFB			DBpedia		
K	3	5	8	3	5	8
semantic based matching	0.93	0.88	0.82	0.96	0.9	0.87
basic approach	0.73	0.68	0.62	0.8	0.76	0.68

Table 3.4: Top-K precision of the basic approach and the semantic based matching

3.10 Conclusion

In this chapter, we have presented our matching process for the keyword search over RDF graphs. We have presented an indexation approach for an RDF graph. We have extended the existing matching approaches by introducing semantics into it. The semantics help in enriching the matching elements and try to find approximate solution for the query keyword if we could not find the exact keywords in the dataset. To this end we have used the patterns that are equivalent to properties based on previous work. We have proposed the use of the semantic relationships provided by available online linguistic resources that are used to search for the semantic relations between the keyword query and the elements of the RDF graphs. Our contribution consists of using these semantic relations to enrich the matching elements of the keyword query and find a solution even if the keyword query is not the same as a graph element. The second contribution in our approach is the type of the matching elements, where the matching elements are nodes, edges and sub-graphs. We have conducted some experiments showing that using some external knowledge give more results for some queries and sometimes returns an answer where other keyword search approaches fail to.

The future works will focus on other types of matching, such as matching the numbers with the character. For example, match "1" with "one". We will also use other methods of external knowledge to enrich the matching elements, such as using machine learning and deep learning in the matching process. Another thing to be considered is to capture the user intent in the keywords. This can be done by finding relationships between them before matching them to the graph. We will also study scalability issues and enable efficient keyword search for massive datasets.

Chapter 4

Keyword Search for RDF data: an Aggregation Approach

Contents

4.1 Introduction 92

4.2 Problem Statement 94

4.3 Aggregation as Steiner tree problem 96

4.3.1 Preliminaries 97

4.3.2 Building the answer Tree 101

4.3.2.1 Adaptation for DNH 102

4.3.2.2 Adaptation of Kruskal's Algorithm 104

4.3.2.3 Using Adapted DNH and Kruskal's algorithm in Building the final sub-graph Result 105

4.3.3 Building the Answer Sub-graph 108

4.4 Ranking the Results 109

4.4.1 Using the Type of Matching Element in the Result . . . 110

4.4.2 Using the Proportion of the Nodes Corresponding to Matching Elements in the Result 112

4.4.3 Aggregating Different Rankings 112

4.5 Experimental Evaluation 113

4.5.1 Prototype Environment 113

4.5.2 Datasets 114

4.5.3 Results 114

4.6 Conclusion 115

4.1 Introduction

In the previous chapter, we have extended the process of matching one keyword with some graph elements using some external knowledge to enhance the process with more semantics. In this chapter we are tackling another problem related to keyword search which is aggregating the matching elements to form a sub-graph representing an answer to the query. Aggregation takes the set of matching elements for each query keyword as an input and find the best way to combine them and produce set of sub-graph results as an output.

For each keyword k_i in a query Q = $\{k_1, k_2, k_3,...k_n\}$ a set $S_i = \{me_1, me_2,...me_j\}$ of matching elements corresponding to k_i was retrieved. In order to provide a sub-graph that is an answer result to Q, we need a find a way to aggregate and combine these matching elements. Our task is to have one element from each set S_i, and we need to aggregate these elements and connect them to create the subgraph result.

Consider G(V,E) to be an RDF graph and Q = $\{k_1, k_2,...,k_n\}$ to be a keyword query and assume that at least one matching element for each keyword query has been identified. The problem is to combine these elements in order to provide an answer to the query Q.

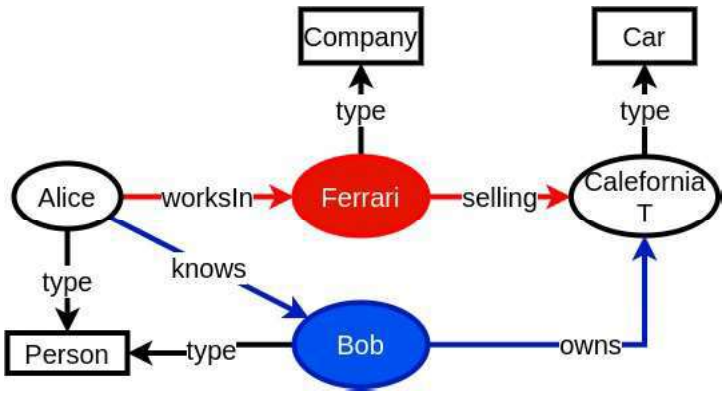


Figure 4.1: RDF graph Example

For example, suppose the two nodes "Alice" and "Car" in the graph of figure 4.1 are matching elements for a keyword query, and the user needs to connect them in order to build an answer to this query. The problem how we can combine them to create the sub-graph result. In our context, we refer to this problem as an aggregation problem. The goal is to find a way that allows us to combine a given set of graph elements to create one single sub-graph; those graph elements can be an edges, a nodes or even a sub-graphs.

Aggregation is not an easy task since there are some main challenges facing this process. The straightforward idea if I want to find the best paths that connects

two or more graph elements is to enumerate all the possible paths between these elements. But this task is a time-consuming and costly process. Another problem consists of creating the final sub-graph with a minimal number of elements. The aim is to focus on the matching elements and not to introduce new unnecessary elements. Creating the minimal sub-graph is not always the best answer, we need also to check the semantic meaning of the final sub-graph and this is another challenge for aggregation. For example, in the previous example we can see that there are two different paths connecting the two nodes "Alice" and "Car" from the graph of figure 4.1. The first one is in red and reflects that Alice works in a company that sales cars. The second path is in blue and represents that Alice knows Bob that owns a car. We can observe that each path reflects a different semantic, the question is which one should be selected in order to connect the two nodes? Checking the semantic of the paths helps us to select the most suitable one in order to build a meaningful final result for the keyword query. Our problem is to define a method to assess a path and determine which one is more meaningful than the others.

Many research works dedicated to keyword search target these problems, some of them use the shortest path as in the approach presented in [66] that combines the shortest paths between all pairs of matching elements. Other approaches use the backpropagation as the approach in [91] that starts from the matching elements and propagates to a common node to build the final graph, but this will not reflect the importance of the semantic of the graph and will introduce unnecessary elements to the final result. Others use the schema to enrich the aggregation process and discovering the paths between the matching elements, the limitation of such an approach consists in having a dataset without a schema.

Our contribution is related to detect the best way of connecting the different matching elements. In our approach, we need to find an assessment method to assess the paths that connect the matching elements during the building of the sub-graph result. Moreover, we need to reduce the number of elements in the sub-graph as minimum as possible without losing the semantic meaning of the sub-graph result.

For the same keyword query, it is possible to have several possible solutions. This is due to the fact that each query keyword can have more than one matching element. In creating the sub-graph result, we need to take into account all the possible matching elements, i.e., all the possible solutions that can be created from the different sets of matching elements. Finally, we will end with a set of sub-graph results. The problem consists in finding a ranking method to rank this set and determine what is the best answer.

As an example, let us consider the RDF graph in figure 4.2 and let K= {Ridley _scott, Denzel_washington, Abstract} be the set of elements to be connected.

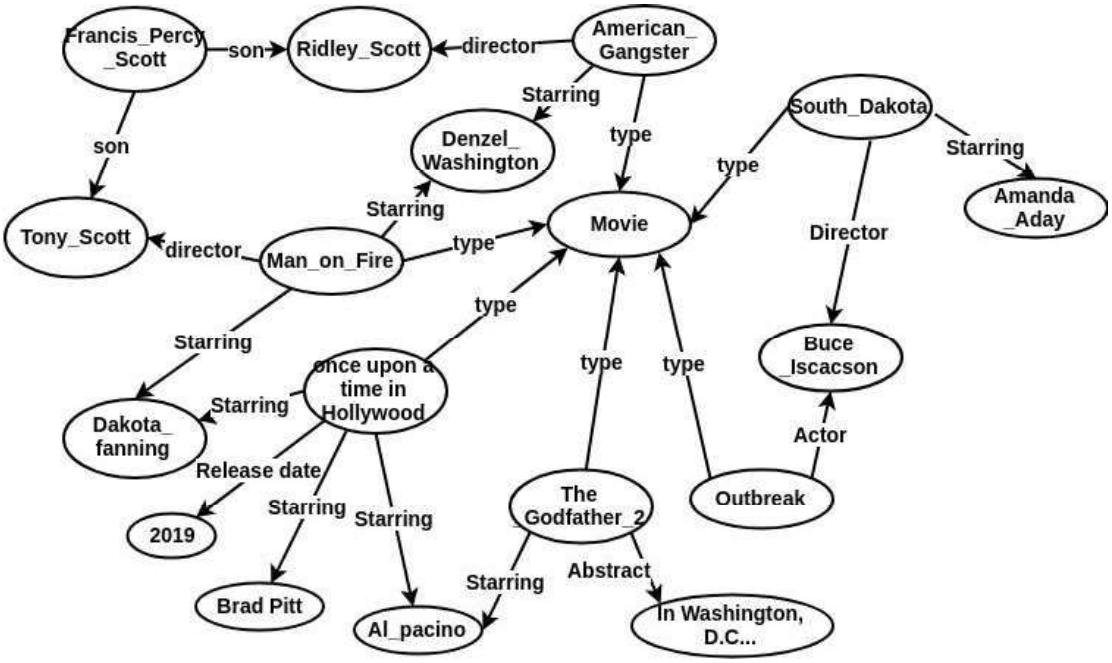


Figure 4.2: RDF graph about Movies

Aggregating the elements in the set K consists in finding a sub-graph containing all the elements in K. We can observe from the graph that we have more than one path connecting the node having the keyword "Ridley_scott" with the edge having the property "abstract". The problem consists in deriving all the possible paths and determining the best path to be used in the aggregation process.

In this chapter, we will present the aggregation method, as well as the ranking method we have proposed to rank the sub-graph results. The rest of this chapter is organized as follows. A problem statement is presented in section 4.2. Section 4.3 presents an overview of our aggregation approach. The ranking method is discussed in section 4.4. Section 4.5 presents our experiments. Finally, we conclude the chapter in Section 4.6.

4.2 Problem Statement

Let us consider G(V,E) to be an RDF graph and Q = $\{k_1, k_2,...,k_n\}$ a keyword query, for each keyword query k_i let $S_i = \{s_{i1}, s_{i2}, ..., s_{in}\}$ be its set of matching elements. Then $S = \{S_1, S_2, ..., S_n\}$ will be the sets of matching elements for the keyword query Q. The set of matching elements S_i represents the elements that can be found in G and matched with the keyword query k_i. The type of these matching elements can be either a node an edge or even a subgraph. Having

a set of matching elements corresponding to each keyword raises the problem of identifying all the possible combinations of matching elements which represents the candidate to create the resulting sub-graph.

Let us consider a combination C = $\{(k_1, s_1), (k_2, s_2), ..., (k_n, s_n)\}$ where each s_i is a matching element for the keyword k_i will be an input. Our goal is to create a sub-graph result G_r(V',E') for the keyword query Q by aggregating the elements in the combination C. Building this sub-graph is what we call aggregating the matching elements. This sub-graph result G_r should be a minimal sub-graph such that for each keyword query k_i in C one matching element s_i is found in G_r, for any pair of matching elements (s_i,s_j), there is only one path connecting them.

For example, let us take the graph of figure 4.3 and let Q={brother, director, Man} be the keyword query. From the graph of the figure 4.3 we can see that the sub-graph colored in blue can be matched with the keyword "brother", the edge colored in green can be matched with the keyword "director" and the node colored in red can be matched with the keyword "Man". Therefore, not all the matching elements are of the same type and they can be nodes, edges and sub-graphs. The problem consists of finding a way to connect the different types of matching elements.

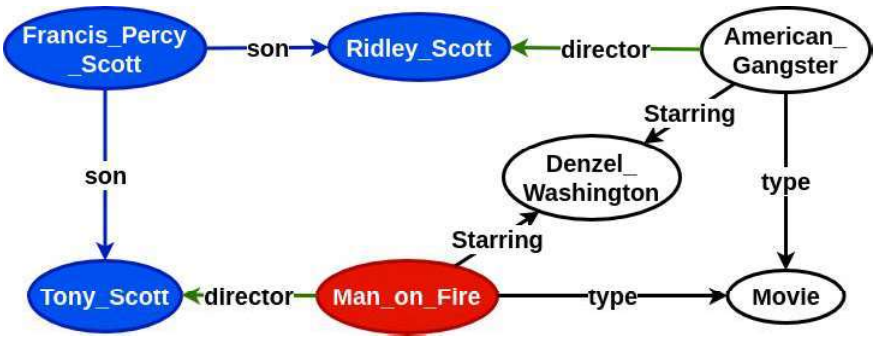


Figure 4.3: RDF graph showing different types of matching elements

There are three main challenges in the aggregation process, the first one consists of building a minimal connected sub-graph containing the matching elements of the considered combination. This sub-graph represents a possible result of the query; it is built by introducing the minimum number of nodes and edges which do not correspond to matching elements. Connecting two elements in a graph is done through a path, but this introduces new elements in the result. Our goal is to minimize the elements in the resulting sub-graph by focusing on the matching elements in C and not introduce new unnecessary elements.

During the building of the sub-graph result, we can find several possible paths between a given pair of matching elements. The problem is which path should we select to build the sub-graph. The second challenge consists in finding a metric to

rank the relevance of the path connecting two matching elements. Let p_i and p_j be two possible paths connecting the elements s_1 and s_2. We need to know which path should be selected in the process of aggregation. What are the criteria used to determine the importance of one path on another? State-of-the-art aggregation tools rely on finding the shortest path between the matching elements. But the shortest is not always the best answer; what if all the paths have the same length. The limitation consists in determining which one will be selected. For this reason, we need to find a way to assess the importance of the paths in the RDF graph.

Since for a given keyword query k_i we can have a set of matching elements $S_i = \{s_{i1}, s_{i2}, ..., s_{in}\}$, then several combinations are created. If we have several combinations, then we will have several sub-graph results for the keyword query. The third challenge is how to rank the different possible answers. For example, let us consider the graph of figure 4.3 and let Q3 = brother, director, Man be the keyword query. The figure 4.4 consists of two differed possible sub-graph results for the keyword query Q3. The problem consists in determining which one is better than the other. Another problem is finding the criteria that affect the rank of a given sub-graph and finding a method to calculate a score of ranking for each sub-graph result.

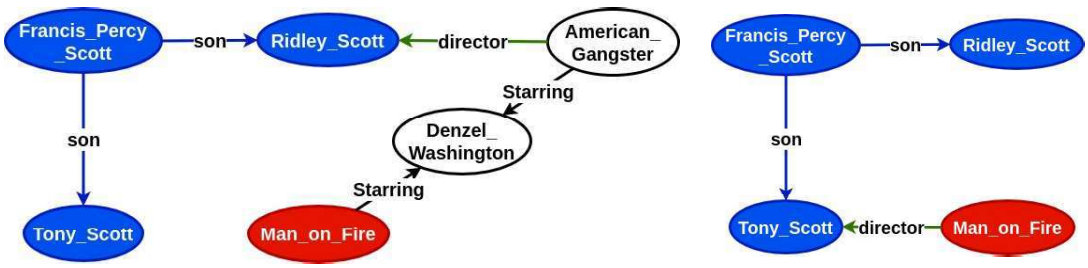


Figure 4.4: Sub-graph results for the keyword query Q3

In this chapter, we present a novel method to aggregate the matching elements and find the best paths between them to extract the sub-graph corresponding to the keyword query. In our method we introduce new way of measure in the addition to the shortest path, this way consists of finding the importance of the path by checking its centrality in the graph. Deriving the possible combination and aggregating the matching elements are presented in the next section. In the second part, a method to calculate the score of ranking for the results is given.

4.3 Aggregation as Steiner tree problem

Our goal is to design a method that aggregates the matching elements and finds the minimal sub-graph connecting them, in addition to have only one path between

any pairs of matching elements.

In order to find the minimal sub-graph, then all the paths between all the matching elements should be retrieved, and all the combinations should be derived to find the minimal sub-graph. This problem has been addressed in the field of algorithmics, and it is a combinatorial optimization problem which is the Steiner tree problem [28]. In the rest of this section, a definition for the Steiner tree problem will be given and we will also present why it could be used in our context and why our aggregation problem can be stated as Steiner tree problem. In addition, we will provide the algorithm we used in our approach and the adaptations we made to better solve the problem. Finally, we will show how we build the answer sub-graph result.

4.3.1 Preliminaries

In this section we define the concepts of the weighted graph, the complete graph and the distance Graph. We have also introduced definition of the Steiner tree problem and the algorithm that solves it.

Definition 8. Weighted Graph: A weighted graph is a graph in which its elements are given a numerical weight. The elements can be node or edge. The weight of the graph is calculated by adding the weights that are assigned to its elements. For example, the graph of figure 4.5 is a weighted graph and the weights are assigned to its edges, the weight of this graph is 23, which is the sum of weight for all its edges.

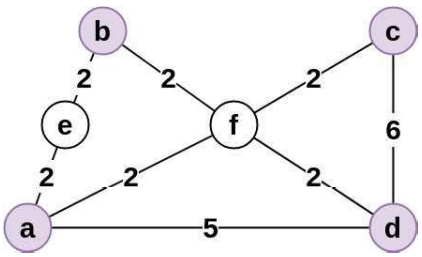


Figure 4.5: Undirected weighted graph

Definition 9. Complete Graph: The complete graph is a graph in which every pair of distinct vertices is connected by a unique edge. For example the graph of figure 4.6 is a complete graph.

Definition 10. Distance Graph: Let graph G (V, E) be a graph, and let $T \subseteq V$ be a subset of nodes called terminals, and a weighted function $d : E \to \mathbb{R}$ on the

edges. The Distance graph DG (T,E') is a complete graph that can be derived from G and its nodes the set of terminal T. Each pair nodes (n_i,n_j) is connected with an edge e_{ij}' $\in$ E' labeled by a weight a_{ij} representing the shortest path weight connecting the nodes n_i and n_j. Let us consider the graph of figure 4.5 and let T = {"a","b", "c", "d"} be the set of terminal nodes, then the distance graph that can be derived from the graph in 4.5 and from the set T is represented in the graph of figure 4.6.

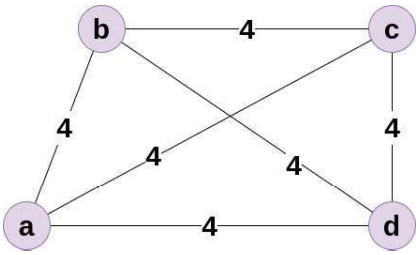


Figure 4.6: Distance Graph for the graph in figure 4.5

Definition 11. Minimum Spanning Tree: The minimum spanning tree is a subset of edges of connected weighted graph that connects all the vertices together without any cycle and with the minimal possible total edge weight. For example the graph of figure 4.7 represents the minimum spanning tree for the distance graph in figure 4.6.

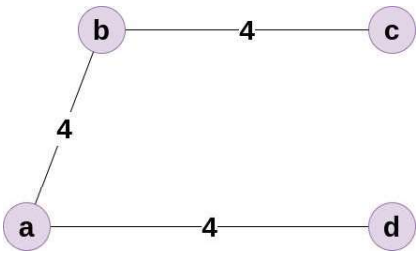


Figure 4.7: Minimum Spanning tree for the graph in figure 4.6

Kruskal's algorithm[54]: It is an algorithm used to find the minimum spanning tree of the weighted graph. The algorithm consists of creating a forest F which is the set of trees where each vertex in the graph is a separated tree in T, and create set S containing all the edges in the graph. The algorithm starts by selecting an edge from S with the minimum weight. If the selected edge is connecting two different trees, then we add it to the forest F and combining two trees into one tree. If it is not, then we select another edge from S. The algorithm terminates

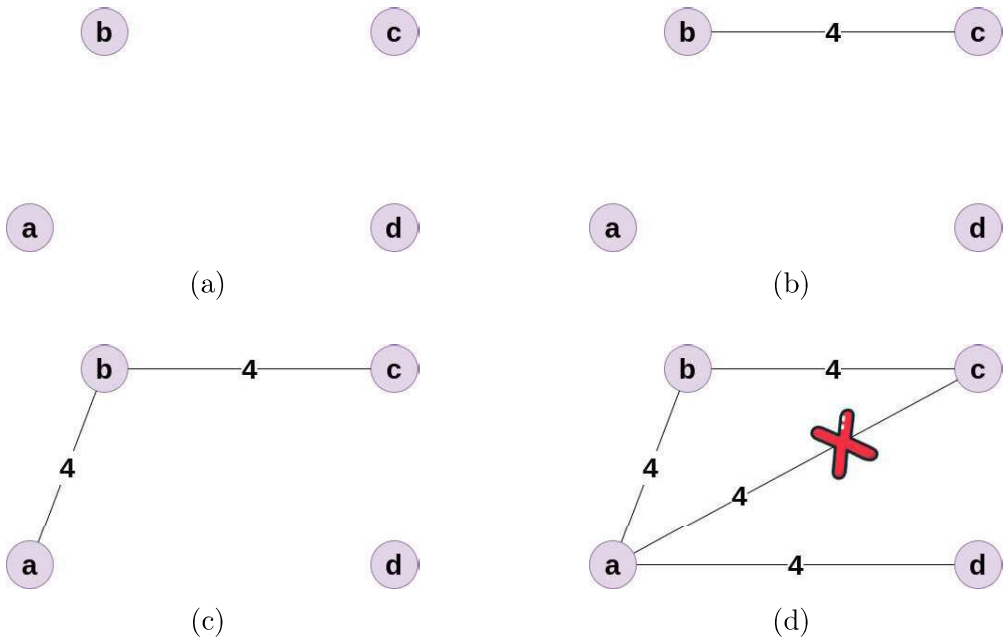


Figure 4.8: Kruskal's algorithms steps to build the minimum spanning tree of the graph in figure 4.6

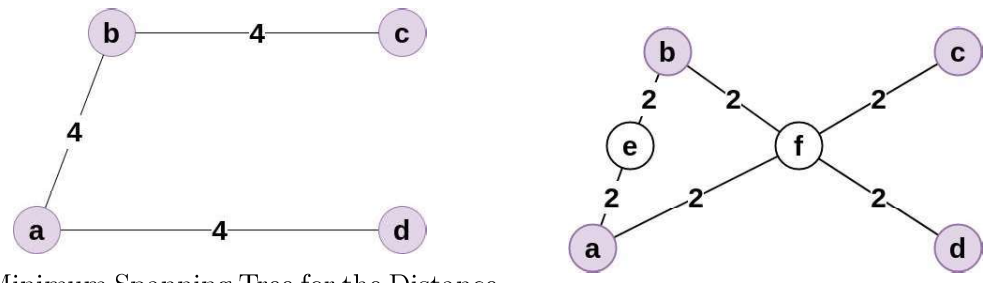
when all the trees in F are connected into one single tree, which is the minimum spanning tree. For example, the steps of building the minimum spanning tree for the distance graph of figure 4.6 are presented in figure 4.8.

Definition 12. Steiner tree problem [28]: Let graph G = (V, E) be a graph, and let $T \subseteq V$ be a subset of nodes called terminals, and a weighted function $d : E \rightarrow \mathbb{R}$ on the edges, the goal is to find a sub-graph S of minimal weight in G containing all the terminals. Other nodes than the terminals can be added to S; they are called Steiner nodes. S should be a tree, which means that from every node s and t $\in V$ there should be exactly one path between s and t. The Steiner tree problem is NP-hard, and there are many research works which aim to find approximate solutions to this problem. The quality of the approximation algorithm is measured by calculating the ratio between the weight of the resulting tree and the optimal Steiner tree.

Distance Network Heuristic(DNH) [52]: It is an algorithm used to solve the Steiner tree problem. Given graph G and set of terminals T=$\{t_1, t_2,..., t_n\}$, a Steiner tree is created by using the distance network heuristic algorithm, which consists of several steps. First, a distance graph DG is created; this graph contains only the elements in T, and every two elements t_i and t_j are connected by a labeled edge. The label is the weight of the shortest path connecting t_i and t_j in G. The

next step is to create the minimum spanning tree MST from DG. The minimum spanning tree is a subset of the edges of a DG without any cycles and with the minimum possible total edge weight. Therefore the sum of edge weights in this tree is as small as possible. The result it MST containing n nodes and n-1 edges, and each edge connecting t_i and t_j is labeled by the length of the shortest path connecting t_i with t_j. The third step of the algorithm consists of creating a sub-graph G' by replacing each edge in MST with a corresponding minimum cost path in G; if several paths are found, one of them is randomly selected. Due to the replacement, G' might not be a tree, Therefore the minimum spanning tree MST' is computed for G'. The last step of the DNH algorithm consists of deleting from MST' all non-terminals of degree 1. Since DNH is an heuristic algorithm, then the final result is an approximate solution for the Steiner tree problem.

Let us take the graph in figure 4.5 as an example, and let T = {"a", "b", "c", "d"} be the set of terminal nodes. We need to find a minimal weight sub-graph S connecting all the nodes in T. Figure 4.6 shows the distance graph created by using the graph in figure 4.5 and the elements in T. As we have seen before, the figure 4.7 shows the minimum spanning tree created from the distance graph in figure 4.6. The next step is to replace the edges in the graph of figure 4.7 with the corresponding path in the graph of figure 4.5.

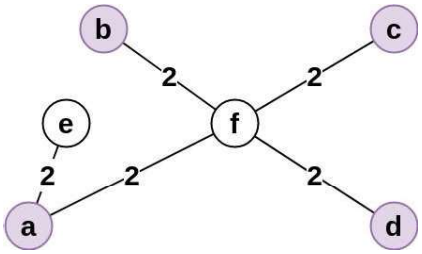


(a) Minimum Spanning Tree for the Distance graph in figure 4.6

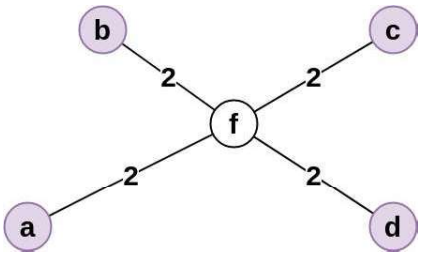
(b) Steiner Tree

Figure 4.9: From the Minimum Spanning tree to the Steiner Tree

In figure 4.9b we have the created graph G' after replacing the paths in figure 4.7 by the corresponding paths in figure 4.5. As we can observe, G' is not a tree and it contains a cycle in it. The next step is to create the MST' which is the MST derived from G'. Figure 4.10a shows the minimum spanning tree that is computed from the graph of figure 4.9b. The last step of DNH is to remove the non-terminal nodes with degrees equals to one. The nodes that are not belong to T denoted by non-terminal nodes. In the graph of figure 4.10a the node labeled by "e" having a degree equal to one and it is not in the set T, then this node and the edge connecting this node to the node "b" are removed. The final result is presented in figure 4.10b.



(a) Minimum Spanning Tree for the graph in figure 4.9b



(b) Final Steiner Tree result

Figure 4.10

4.3.2 Building the answer Tree

We state the aggregation problem as Steiner tree problem. In our context, the set of terminals represents a combination of matching elements. For example, let Q9 ={brother, Dakota, Washington, Actor} be the keyword query to be applied to the graph of figure 4.2. First we search for the matching elements in the RDF graph that can be matched with the keyword in Q9 by using the matching approach presented in section 3.1. The results are presented in the table of figure 4.11 where each keyword is associated with a set of matching elements corresponding to it.

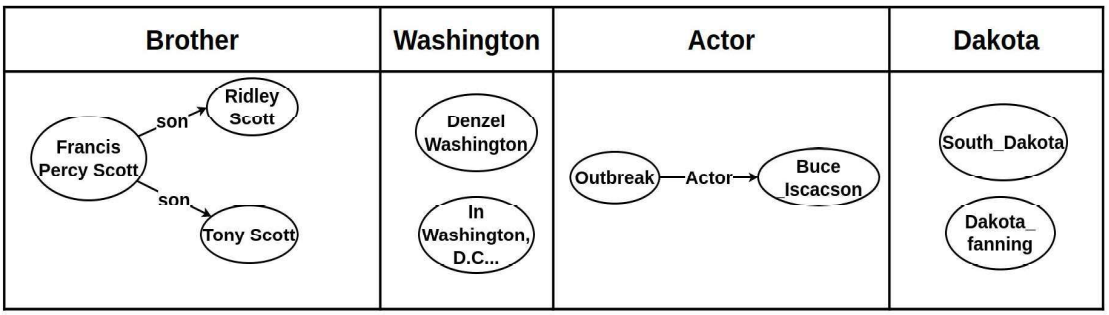


Figure 4.11: Matching Elements for each Keyword in the Query Q9

After obtaining the set of matching elements for each keyword, the possible combinations are constructed. In figure 4.12 we can see all the possible combinations, obtained by performing the cartesian product on the sets of matching elements shown in figure 4.11.

Let us take the combination "c4" from figure 4.12. We can see from this combination that the keyword "brother" is matched with sub-graph while the keyword "Actor" is matched with an edge and the keywords "Washington" and "Dakota" are matched with nodes. We can deduce that the three different types of matching elements are presented in this combination. In the DNH algorithm, all

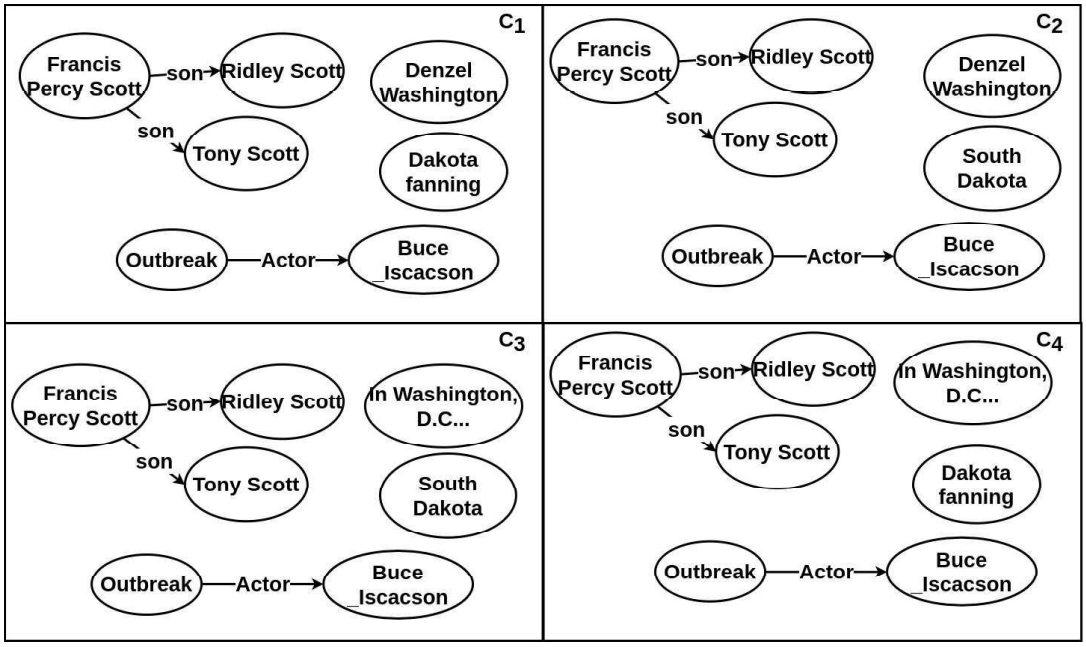


Figure 4.12: Matching Elements for each Keyword in the Query Q9

the terminals are nodes, so in order to use the DNH in our context, an adaptation is needed.

4.3.2.1 Adaptation for DNH

In DNH algorithm the terminals are a set of nodes, but in our context, the terminals can be either nodes, edges, or sub-graphs. An adaptation is needed in order to make the DNH algorithm fits our needs. We need to translate all the matching elements of different types (nodes, edges and sub-graphs) into terminal nodes. Let us consider a matching element m_i for a query keyword k_i, this matching element is translated into a node according to the following rules:

- **Rule1:** If m_i is a node then the resulting node is m_i
- **Rule2:** If m_i is a edge then the edge and the 2 connected nodes are replaced by a single node
- **Rule3:** If m_i is a sub-graph then the sub-graph is replaced by one node

After applying the above translation rules we will end with the graph of figure 4.13. The sub-graph in figure 4.12 representing the keyword brother is replaced by one node labeled brother according to rule3, and the edge "Actor" with its

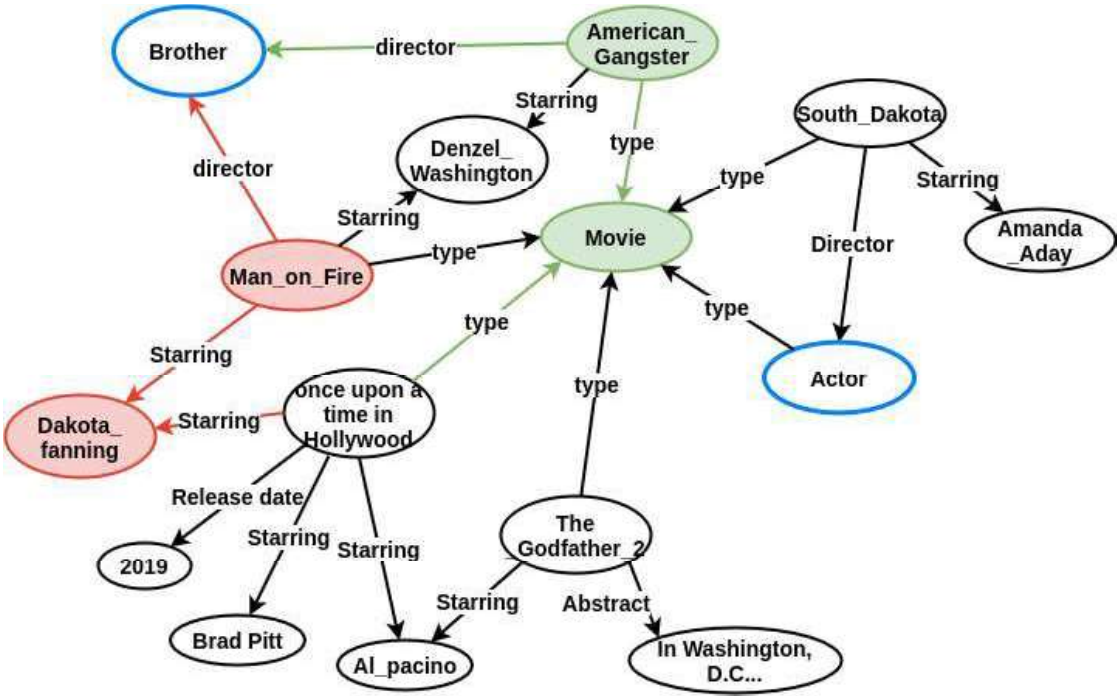


Figure 4.13: RDF graph of figure4.2 after executing the translation rules

corresponding nodes "Outbreak" and "Buce_Iscacson" are replaces by one node labeled "Actor" according to rule2.

Another adaptation of DNH algorithm is related to the selection of the most relevant path. In DNH, while replacing the edges in minimum spanning tree with the original paths from the graph, if two edges have the same path, one of them is selected randomly. In our approach, we need to select one of the paths with the same length according to some criteria to be defined. To this end, we have introduced the notion of centrality degree weight of a path. Instead of selecting an arbitrary path if there are several paths between a pair of terminals, we will select according to the centrality degree weight. The centrality degree weight for a node is defined as the number of both incoming and outgoing edges to this node.

Definition 13. Path Centrality Score

The centrality degree weight of a path is calculated as follows. Let p=[$(v_1, e_1, v_2)(v_2, e_2, v_3)....(v_{n-1}, e_{n-1}, v_n)$] be the path connecting the two terminal nodes v_1 to v_n; the centrality degree weight of p $C(p) = \frac{\sum_n^{i=1} deg(v_i)}{n}$ is the average degree of the nodes in the path.

We can also limit the computation of $C(p)$ to the top k nodes having the highest centrality. This score allows us to know which path is more centralized

in the graph. For example let us take the graph of figure 4.13, and take two different paths with the same length as an example. There is the first path P1 that connects the node "Brother" with the node "once upon a time in Hollywood" and highlighted in red; another path P2 is also connected the same nodes but it is highlighted in green. Both paths P1 and P2 have the same length which is equals to three. If we want to check which one is more important than the other then the $C(p)$ is calculated. In P1 there is 4 nodes, the degree of these nodes are 2, 4, 2 and 5 then $C(P1) = 13/4$, while $C(P2) = 16/4$. Then we can conclude that P2 is more centralized than P1 in the graph of figure 4.13. The adapted distance network heuristic is described by algorithm 6.

Algorithm 6 Finding steiner tree

1: **procedure** AggregatingMatchingElements(G, C_i) /*G is the RDF graph and C_i is one of the combinations*/
2: $DG \leftarrow DistanceGraph(G)$ /*Compute the complete distance graph*/
3: $MST \leftarrow MinimumSpanningTree(DG)$ /*Compute the Minimum Spanning Tree*/
4: $G' \leftarrow Replace(G, MST)$ /*replace each edge in MST by a corresponding minimum cost path in G; if several paths are found, select the path having the highest centrality degree weight*/
5: $MST' \leftarrow MinimumSpanningTree(G')$
6: $F_G \leftarrow delete(MST')$ /*Delete all non-terminals of degree 1*/
7: **return** F_G
8: **end procedure**

4.3.2.2 Adaptation of Kruskal's Algorithm

One of the steps of DNH algorithm is to build the minimum spanning tree. This is done by using the Kruskal's Algorithm that is defined in 4.3.1. This tree is build by selecting the edges with the minimum weight in the graph. If we have more than one edge with the same minimum weight then one of them is selected arbitrarily. In our context, we need to select the edge based on its meaning. This can be done by using another assessment method on the edges and select the most relevant one. For example while building the minimum spanning tree of the graph in figure 4.7, all the edges have a weight equals to four, then any edge will be selected randomly. In order to solve this problem, we have adapted the Kruskal's algorithm that computed the minimum spanning tree. The adaptation consists of using additional metric during the assessment of the path instead of just using the length of the path. This allows us to create more meaningful sub-graphs that connects the different matching elements.

Algorithm 7 Adapted Kruskal's algorithm

```
1:  A = φ
2:  procedure KRUSKAL(G)
3:      Make − set(V)
4:      for each v ∈ G.V do
5:          Make-set(v)
6:      end for
7:      for each (u, v) in G.E ordered by weight(u, v) and CDW, increasing do
8:          if FIND − SET(u) ≠ FIND − SET(v) then
9:              A = A ∪ {(u, v)}
10:             UNION(u, v)
11:         end if
12:     end for
13:     return A
14: end procedure
```

Let p_{nk} and p_{kl} be the two edges in the distance graph that connect the nodes n_i with n_j and n_k with n_l respectively. The weight w(p_{nk}) is the weight of the shortest path that connects the nodes n_i with n_j. If w(n_i,n_j) and w(n_k,n_l) are equal, then instead of selecting arbitrarily one of them another method is used. This method consists in calculating the centrality degree for each edge and to select the relevant edge according to it. This centrality degree value is based on the original path that the edge in distance graph represents and it is calculated according to the formula in definition 13. The edges are first sorted according to w and then if we have two or more edges having the same w, we sort them according to centrality degree and select the relevant edge. The adapted Kruskal's algorithm is presented in algorithm 7.

4.3.2.3 Using Adapted DNH and Kruskal's algorithm in Building the final sub-graph Result

To build the final sub-graph result for the keyword query, all the combinations are derived. For each combination we apply the adapted DNH and the adapted Kruskal's algorithms to create the sub-graph result. First all the matching elements are translated into nodes according to the translation rules and the new graph G_{new} with the translated matching elements is created. The adapted DNH algorithm is applied on G_{new}, by first extracting the distance graph DG from the new graph G_{new}. To extract the minimum spanning tree MST from the graph DG, the adapted Kruskal's algorithm is extracted. Each edge in MST will be replaced with the corresponding paths in the graph G_{new}. In this phase, if there are more

Algorithm 8 Finding results sub-graph

1: **procedure** BUILDINGRESULTS(G, $hashmap(k_i, S_i)$)
2: C = $Combinations(S_i)$
3: **for** each combination c_i in C **do**
4: $G_{tran} \leftarrow ReplaceFromMatchingElements(G)$ /*Replace matching elements with nodes by using translation rules*/
5: $DG \leftarrow DistanceGraph(G_{tran})$ /*Compute the complete distance graph*/
6: $MST \leftarrow MinimumSpanningTree(DG)$ /*Compute the Minimum Spanning Tree*/
7: $G' \leftarrow Replace(G, MST)$ /*replace each edge in MST by a corresponding minimum cost path in G; if several paths are found, select the path having the highest centrality degree weight*/
8: $MST' \leftarrow MinimumSpanningTree(G')$
9: $R_G \leftarrow deleteNonTerminals(MST')$ /*Delete all non-terminals of degree 1*/
10: $F_G \leftarrow ReplaceToMatchingElements$ /*Replace the nodes with the corresponding matching elements */
11: $ResultsSubGraphs = ResultsSubGraphs \cup F_G$
12: **end for**
13: **return** $ResultsSubGraphs$
14: **end procedure**

than one path candidate to replace an edge in the minimum spanning tree, then the path centrality score is used. After replacing all the edges in the minimum spanning tree, G' is created. In order to compute the tree, the minimum spanning tree of G' denoted MST' is extracted. All the non-terminal nodes that are of degree equals to one are deleted to create the result to the keyword query. The last step is to translate the nodes that represents the matching elements to their original type to build the final result. The algorithm of creating the resulting sub-graphs is presented in algorithm 8.

For example let us take the combination C_4 in figure 4.12. The graph of figure 4.13 is the result of the graph in figure 4.2 after executing the translation rules to translate the matching elements into nodes. The next step is to extract the distance graph DG from this graph by using the shortest path between all the matching elements. Let me_i and me_j be two matching elements, then w(me_i,me_j) is the number of edges in the shortest path connecting me_i and me_j. For example the shortest path between "brother" and "dakota" is equal to two and the shortest path between "Brother" and "Washington" is equal to four. Figure 4.14 shows the distance graph induced by C_4.

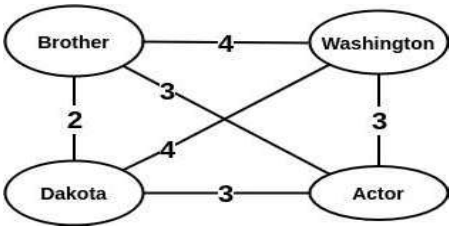


Figure 4.14: Distance graph constructed from the graph of figure 4 and C_4 combination

We compute the minimum spanning tree from the distance graph by using the modified Kruskal's algorithm presented in Algorithm 7.

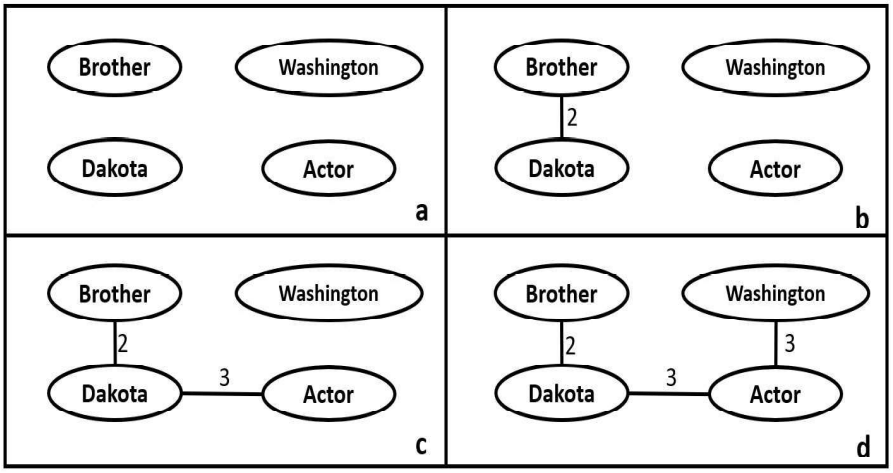


Figure 4.15: Building the Minimum Spanning Tree on the Distance Graph of figure 4.14

Figure 4.15-d shows the Minimum spanning tree constructed using the adapted Kruskal's algorithm. The steps are shown in figure 4.15. After selecting the first edge which connects "Brother" with "Dakota" in figure 4.15-b, we select the edge with the minimal weight. But as we can see in figure 4.14, there are four edges all having the same weight which is equal to three. We then compute the centrality degree for these edges; the results are shown in the table below.

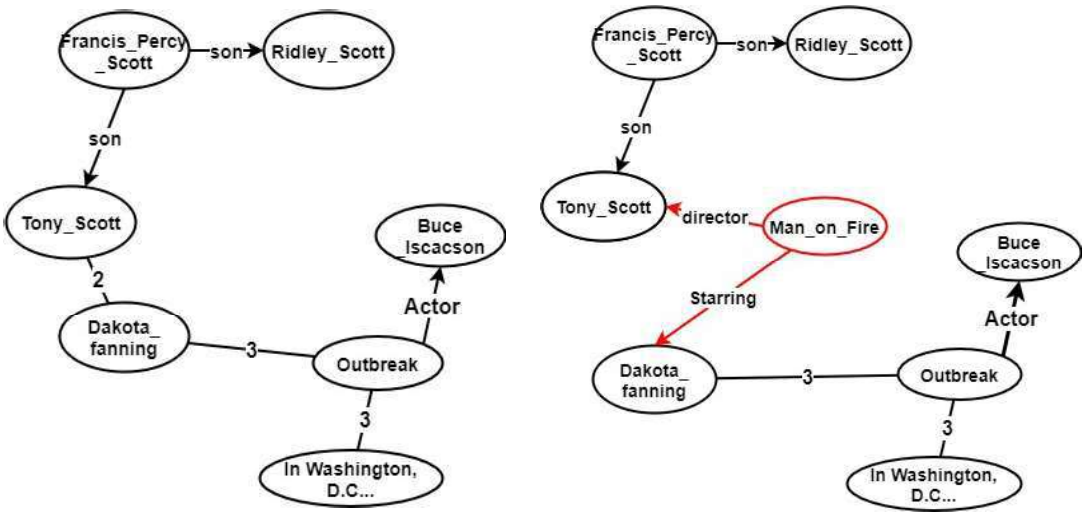
We select the edge with the highest centrality degree as shown in figure 4.15-c and then select the edge connecting "Washington" and "Actor" as presented in figure 4.15-d. If the number of edges equals |V|-1, then we can deduce that this is the minimum spanning tree for the distance graph in figure 4.14.

path	C(p)
Washington-Actor	3
Brother-Actor	4.66
Dakota-Actor	5

Table 4.1: The Centrality Degree for the Edges in the Distance Graph of figure4.14

4.3.3 Building the Answer Sub-graph

The result we obtain from the DNH algorithm is an undirected tree with edges labeled with weight and nodes labeled with the query keywords. The next step consists of replacing the edges with the corresponding paths from the graph G. We have done adaptations to replace the matching elements with nodes before executing the DNH. This means that some of the nodes in the result of DNH are not actual nodes so we need to rewrite the resulting graph by replacing these nodes with the corresponding matching elements. For example the node "Brother" will be replaced by the sub-graph which represents that "Tony_Scott" and "Ridley_Scott" and brothers. After replacing all the matching elements, we will get the graph of figure 4.16a.



(a) Replace nodes in MST with Matching Elements

(b) Replace Edges in MST with Exact Paths from the Initial Graph

Figure 4.16: Replacing the Nodes and the Edges in the MST

To create the final resulting sub-graph we need to replace the edges that are connecting the matching elements in the graph of figure 4.16a with the corresponding paths. For example the edge connecting the two nodes "Tony_Scott"

and "Dakota_fanning" will be replaced by the path colored in red in the figure 4.16b. After replacing the nodes and the edges in the minimum spanning tree by the matching elements and the paths in G. We obtain the sub-graph of figure 4.17 as a final solution.

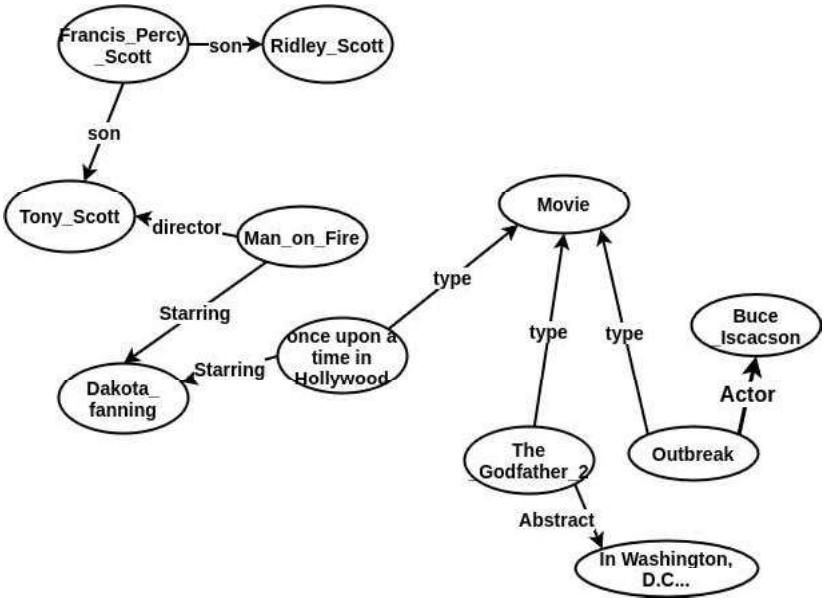


Figure 4.17: Solution for the Keyword Query Q9 after using the combination C4 from the list of combination in the figure 4.12

4.4 Ranking the Results

For each keyword query, we might have more than one candidate matching element. In order to take all the candidates into account while creating the sub-graph result, the cartesian product is used to provide the set of combinations. The elicitation of all the combinations of graph elements and the aggregation of graph elements for each combination lead to several sub-graphs, each one representing a possible answer for the query. For example, let us take the combination C_2 from the set of combinations for the keyword query Q9 that are presented in figure 4.12 and build the sub-graph result. We will end with the graph of figure 4.18 as a solution.

We can observe that the graph of figure 4.17 is different from the graph of figure 4.18 this difference raises a question about the relevance of the different sub-graph results and if there is a way to rank them. The problem consists in ranking these sub-graph results and determining if there are better results than others. Let K=$\{k_1, k_2,.. ,k_n\}$ be a keyword query and let c_i and c_j be two combinations for

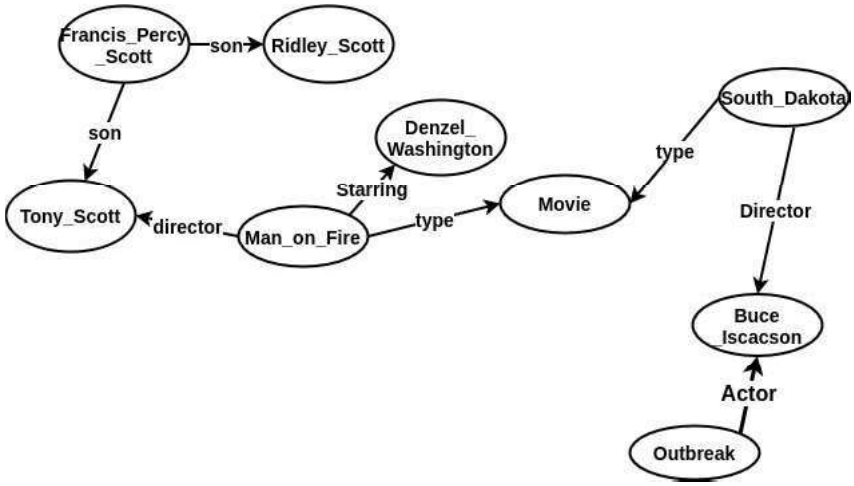


Figure 4.18: Second possible Solution for the Keyword Query Q9 after using the combination C2 from the list of combination in the figure 4.12

the matching elements of K. Suppose we have two sub-graph G'_1 and G'_2 created by applying the aggregation process described above on both c_1 and c_2 respectively. We need to define a function rank(G) that takes a sub-graph as input and returns a score between 0 and 1 as an output, this score represents the ranking of the sub-graph result G. In our approach, we have ranked the results according to three different strategies, the first one is by using the type of matching element in the Result extracted during the matching process. The second one is based on using of the Proportion of the Nodes Corresponding to Matching Elements in the Result according to the total number of nodes and edges in the resulting sub-graph. The third one is an aggregation method for the previous two strategies. The three different types of ranking are expressed in the next part.

4.4.1 Using the Type of Matching Element in the Result

According to the matching process, the matching elements can be classified into two types the matching elements retrieved by using one of the equivalent relations and are denoted by "equivalent matching elements" and the ones that are retrieved by using one of the closed relations between the query keyword and the RDF graph elements and are denoted by "approximate matching elements". The matching elements in the resulting sub-graph can be equivalent matching element or approximate matching element.

Our ranking strategy is based on considering the type of matching elements, we consider that a answer with more equivalent matching elements should have higher rank than an answer with a lower number of equivalent elements. The answer that

contains only equivalent matching elements reflects that we have found exactly what we are looking for. Unlike the answer with approximate matching elements, it reflects that we did not find the elements we are interested in but instead of not providing an answer, we have provided an approximation of it. Therefore in the final result we prefer to have an equivalent matching elements. We can conclude that the more approximate matching elements we have in the answer, the lower the quality of the answer. For example let us take the graph of figure 4.19 as an example, and let Q10={"director", "Starring", "Release-date"} be the keyword query.

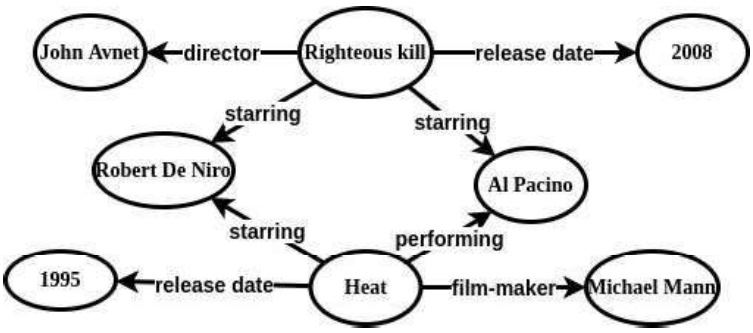


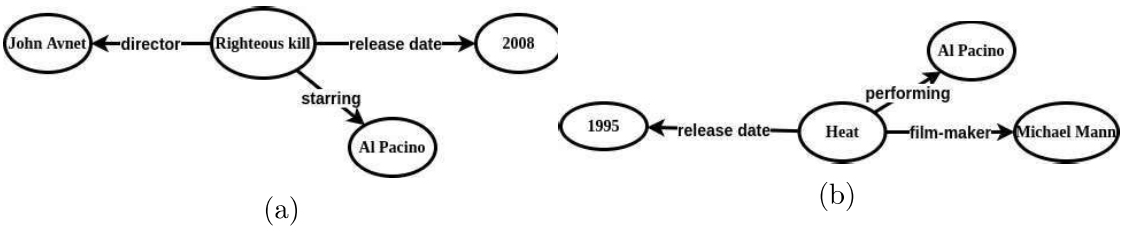
Figure 4.19: An RDF graph describing movies

In figure 4.20 there are two possible sub-graph results. As we can observe from the figure, the two graphs have the same number of nodes and edges but the difference is in the types of the matching elements. In figure 4.20a all the matching elements are of type exact matching while the graph of figure 4.20b consists of one exact matching element and two approximate matching elements.

Let G be a resulting sub-graph, the formula of ranking G according to this strategy is as follows:

$$\text{Score(G)} = 1 - \frac{C}{N}$$

where C is the number of approximate matching elements and N is the total number of nodes and edges in the graph G.



(a) (b)

Figure 4.20: Two possible answers for the query Q10

4.4.2 Using the Proportion of the Nodes Corresponding to Matching Elements in the Result

Considering the type of matching elements in the resulting sub-graph is one way of ranking the solution. Another way could be considered is the proportion of nodes and edges that have been introduced to link the matching elements and denoted as linking elements. For example, in figure 4.21 there are two possible solutions for the query Q10. As we can observe from the two graphs, both have three equivalent matching elements, while the difference is in the number of linking elements used to connect the matching elements and create the final sub-graph result. The graph of figure 4.21a contains less linking elements (4 linking elements) than the graph of figure 4.21b (6 linking elements).

Let G be a resulting sub-graph, the formula of ringing G according to this strategy is as follows:

$$\text{Score(G)} = 1 - \frac{L}{N}$$

where L is the number of Linking elements and N is the total number of nodes and edges in the graph G.

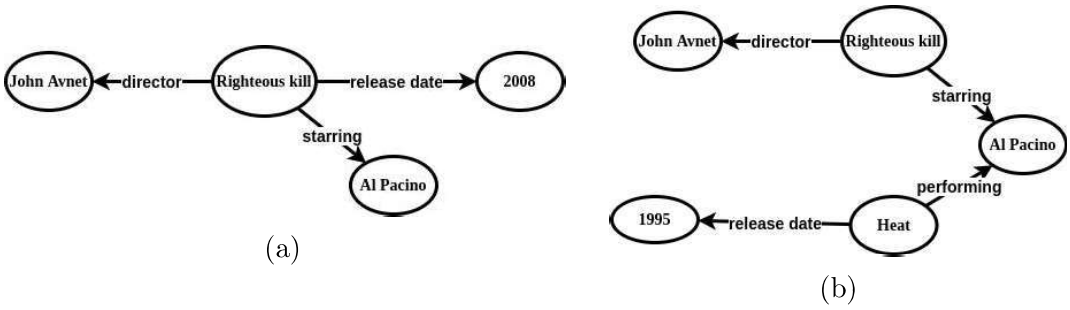


Figure 4.21: Two Possible sub-graph Results for the Query Q10

4.4.3 Aggregating Different Rankings

In the previous two sections we have seen two different strategies of ranking the RDF graph results. In this section we will present an aggregation method that aggregates both strategies. A ranking method that we will propose should reflects that the graphs in figure 4.20a is more important than the graphs of figures 4.20b and 4.21b.

We need to present a ranking method expressing that the less linking elements in a sub-graph, the better the solution. Also this formula should Express that the more equivalent matching elements in a sub-graph, the better the solution. We calculate the ranking score as follows:

$$\text{Score} = 1 - \frac{[w*C+(1-w)*L]}{N}$$

where C is the number of close matching elements, L is the number of linking elements, N is the total number of nodes and edges in the sub-graph and $0 \leq w \leq 1$ is the weight for C.

In this formula, both the approximate and the linking elements are presented by C and L, then both of them affects the ranking. The weight w is used to express the importance of one criteria on another. If the user considers that having approximate elements in the final result is better than having linked elements, then he will increase the w to be closed to zero. If this weight is closed to one this expresses that the approximate elements are much more important than the linking element. If we have a sub-graph result and all its matching elements are exact matching and there is no linking elements in it, then its score will be equals to one according to the formula.

Let us take the graphs of figures 4.18 and 4.17 as examples. Both of these sub-graphs are results for the keyword query Q9. Suppose that w is equal to 0.5, then the ranking score for the graph of figure 4.17 is equal to 0.69 since there are 13 linking elements and 21 is the total number of nodes and edges while the score of the graph is figure 4.18 is equal to 0.73 since the number of linking elements is equal to 9 and the total number of elements equals to 17.

4.5 Experimental Evaluation

This section describes our experiments to validate the performances of our approach. Our goal is to observe the performance of using an adapted DNH algorithm during the building of the subgraph result.

4.5.1 Prototype Environment

Our approach is implemented in Java, we have used the Jena API for the manipulation of RDF data. For indexing and searching the keyword query, we have used the Lucene API. The Jung API is used for graph manipulation and visualization.

In the rest of this section, we describe our experiments to validate the performances of our approach. Our goal is to study the impact of using an adapted DNH algorithm in building the final sub-graph results, as well as to evaluate the ranking model with various keyword queries. All the experiments have been performed on Intel Core i7 with 32GB RAM.

4.5.2 Datasets

We have used two datasets: AIFB and DBpedia. AIFB contains 8281 entities and 29 233 triples. The subset of DBpedia we have used is related to movies, their title, stares, director, released data and other properties. This dataset contains 30 793 triples. The size of the keyword queries was between 3 and 7 keywords. The total number of queries was 30 queries (15 for each dataset).

We have tested and compared our keyword search approach both with and without the use of the adapted distance network heuristic algorithm to solve the Steiner tree problem. We have compared our approach (referred to as ST) to an approach consisting of picking a random matching element and computing the shortest path between this element and all the other ones in a combination. We refer to this approach as the basic approach.

4.5.3 Results

In this section, we need to present three different experiments that have been performed. The first experiment is to check the average execution time of our approach with respect to the number of keywords in the query. The second experiment is to check the number of results sub-graphs before and after using the adapted distance network heuristic algorithm. The last experiment is to check the effectiveness of our approach by calculating the precision at K and comparing it with the basic approach.

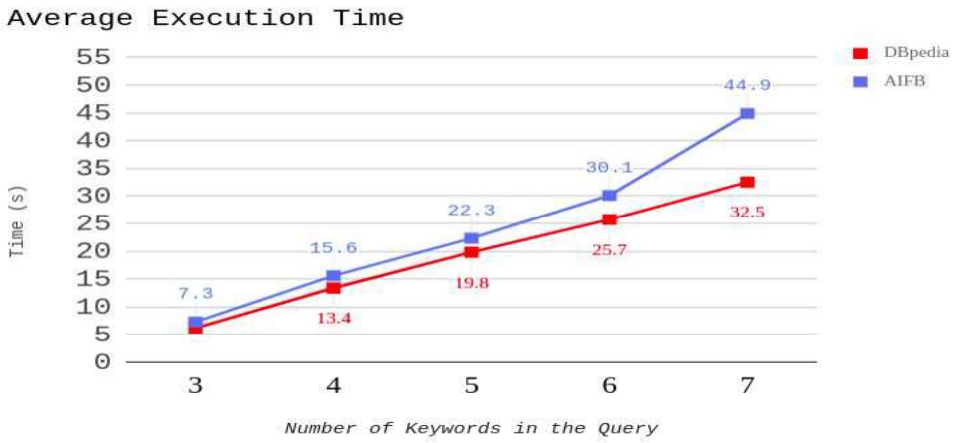


Figure 4.22: Average Execution Time According to the Size of the Query

Figure 4.22 shows the execution time with respect to the size of the query. The graph shows that the execution time increases when the number of keywords

increases for both datasets. We can also see that the execution time for AIFB is greater than the execution time of DBpedia because the size of data in AIFB is greater than the size of the dataset extracted from DBpedia.

query	Q1	Q2	Q3	Q4	Q5	Q6	Q7	Q8	Q9	Q10
№ of results (ST approach)	5	17	7	5	32	48	4	10	2	9
№ of results (basic approach)	7	32	9	5	53	62	6	17	3	11

Table 4.2: Number of Results for each Keyword Query(DBpedia)

As we can observe from table 4.2, the number of results decreases when the adapted distance network heuristic algorithm is used during the aggregation process because using this method decreases the number of inaccurate results as shown in table 4.2.

To check the effectiveness of our approach, we have used 10 queries from table 2 and asked five users to check the top-k results for each query and give the number of relevant results. We have computed the precision at k as follows:

$$P@K = \frac{NumberOfRelevantResults}{K} \tag{4.1}$$

.

According to table 4.3, we can see that P@K varies between 0.98 and 0.94 for our approach. These results are better than the ones obtained using the basic approach, where P@K varies between 0.89 and 0.87. We can deduce that the results achieved with our approach were more accurate according to the users.

Data	AIFB	
K	5	10
Top-K ST approach	0.98	0.94
Top-K basic approach	0.89	0.87

Table 4.3: Top-K Precision

4.6 Conclusion

In this chapter, we have provided an approach of the aggregating method for the matching elements returns by the matching elements process in the keyword query approach. The goal of this chapter was to characterize the semantic of the

different paths during the aggregation process. This process consists in first using the cartesian product to extract all the possible combinations for the matching element. Then for each combination, we need to create a sub-graph connecting all the elements in this combination, we stated our problem as a Steiner tree problem. We have adapted the distance network heuristic algorithm, which is a well known algorithm used to solve the Steiner tree problem. Our adaptation was to introduce a score for calculating the average degree in the path, and this score reflects the importance of the paths during the construction of the sub-graph result. We have also presented three different methods of ranking in order to assess the final results. These methods are based on two different criteria, the first one is the number of the linking elements and the total number of nodes and edges is the final result. The second criteria that affects the ranking score is the type of the matching elements; if the matching elements are extracted by equivalent relations of closed relations.

In future works, we will study the scalability issue and enable efficient keyword search for massive datasets. We will also study the importance of the path that connects two matching elements not only by using the centrality but also by using the semantics that is reflected by this path. This can be achieved by using the schema or by using some external knowledge to express the most meaningful relation that should be between two concepts. In the next chapter, we will present another summarization graph techniques that be also used as support to explore RDF graphs..

Chapter 5

Theme-Based Summarization for RDF Datasets

Contents

5.1 Introduction **117**

5.2 Motivating Example **119**

5.3 Problem Statement **121**

5.4 Overview of the Approach **122**

5.5 Theme Identification in an RDF Graph **124**

5.6 Theme Summarization **125**

5.6.1 Evaluating Node Relevance in an RDF Graph 126

5.6.2 Identifying the Relevant Nodes 128

5.6.3 Building a Theme Summary 130

5.7 Building the final summary **134**

5.8 Evaluation **137**

5.9 Conclusion **140**

5.1 Introduction

Summarization is a technique that consists in providing a short, clear and brief description of the main facts or ideas of something. The purpose of the summary is to give the reader or the user condensed and objective points of the main idea of a text. The summary is used in a variety of situations. For example, the abstract

of this thesis is a type of summary to provide the reader with the main idea of this thesis and the main contribution.

Summarization can be used in RDF data to support the user in querying, exploring and understanding what this dataset describes. RDF graph summarization has been the topic of several research works [18]. The goal of the proposed approaches is to provide the user with a concise representation that captures the semantics of a graph to ease its exploitation.

According to the related works described in chapter 2, the summarization of an RDF graph can be classified into 3 distinct categories. The first category of approaches summarize the RDF graph based on the structure of the graph as in [68, 84, 81] that use different methods to assess the importance of the nodes in the graph by calculating their centrality and build the summary based on the top-k most central nodes. Others like in the approaches presented in [47, 36] provide the summarization by considering the typed elements as the most important elements in the graph and all the typed elements and their classes should be presented in the summary. Focusing on the typed elements will lead to having a summary missing important elements that are untyped.

The second category of approaches summarize the RDF graph based on statistical information collected from the graph itself as in [49, 61]. This information can be the number of occurrences for a class with a set of properties or the number of times a property appears in the graph. The summarized graph will be based on the most important nodes and/or edges based on the information collected. The summarized graph in this category will contain the statistical information in addition to the important elements. In this category, the approaches take into account the importance of the elements according to statistical information, but selecting the most relevant elements can lead to the fact that some parts of the graph will be over-represented.

The third category of approaches consists of semantic-based summary where the summary is based on the semantics of the information described by the graph as in [14, 71]. The RDF graph can describe more than one topic, for example, the graph of figure 5.2 describes three different subjects, the first one is determined by a red circle and describes the monuments in France. The second topic is determined by a blue circle and presents information about the university, professors, and courses. The last topic about the publications and the articles is presented in the green circle. We refer to each topic by a theme, the summarized graph in three category is based on the themes extracted from the RDF graph. The extracted themes can provide the users with a summary of what the graph describes. This way of summarization might need a huge number of elements to describe the different themes in the graph.

If we consider all these categories we can see that they are based on either the

type of the node or on the topology of the graph. Some of them provide a summary without taking into account all the underlying topics in the RDF graph, and others build the summary based on the typed nodes. In our view, there is another way of considering the summarization problem. It is to consider the underlying topics of the graph and to guide the summary based on these topics.

The main goal we target in this chapter is to provide an approach that not only describes all the underlying themes in an RDF graph but also to provide a summarization graph taking into account all these themes. This raises several issues, the first one is to identify the different themes in the graph. As they are not specified explicitly in the graph, we need to find a technique to retrieve them. After identifying the themes, what are the relevant nodes to be selected from it? These nodes will be part of the theme summary. The problem consists in finding an assessment tool to assess the elements of the theme. The last problem, consists in creating the final summary for whole dataset from a given set of summary for each theme. The problem is how can we aggregate the individual theme summaries to create the final summarized graph.

In this chapter, we present a theme based summarization approach, which extracts a meaningful summary from an RDF graph. This summary takes into consideration all the underlying themes in the RDF graph by extracting them before creating the summary. Each theme will be summarized based on set of relevant metrics that are capable of identifying the most important nodes in a theme. The final graph summary is created by aggregating all the summarized themes.

The rest of this chapter is organized as follows. A problem statement is given in section 5.3. An overview of the approach is provided in section 5.4. We present theme identification in Section 5.5. Section 5.6 presents the process of summarizing the extracted themes. The generation of the graph summary is presented in section 5.7. Section 5.8 presents our experiments. Finally, we conclude the chapter in Section 5.9.

5.2 Motivating Example

Consider the example of an RDF graph given in figure 5.1, which describes professors, their courses, their publications, the students they advise and the monuments of Paris. Assume that the summarization algorithm consists in selecting the top-k most central nodes, considering that the centrality of a node is equal to its degree. For k=4, the nodes "p1", "c2", "s1" and "s2" are the most central nodes as their degree is 5, 5, 4 and 4 respectively.

Assume that the summary is built by finding the shortest paths between the relevant elements. Figure 5.2 shows an example of graph summary considering the

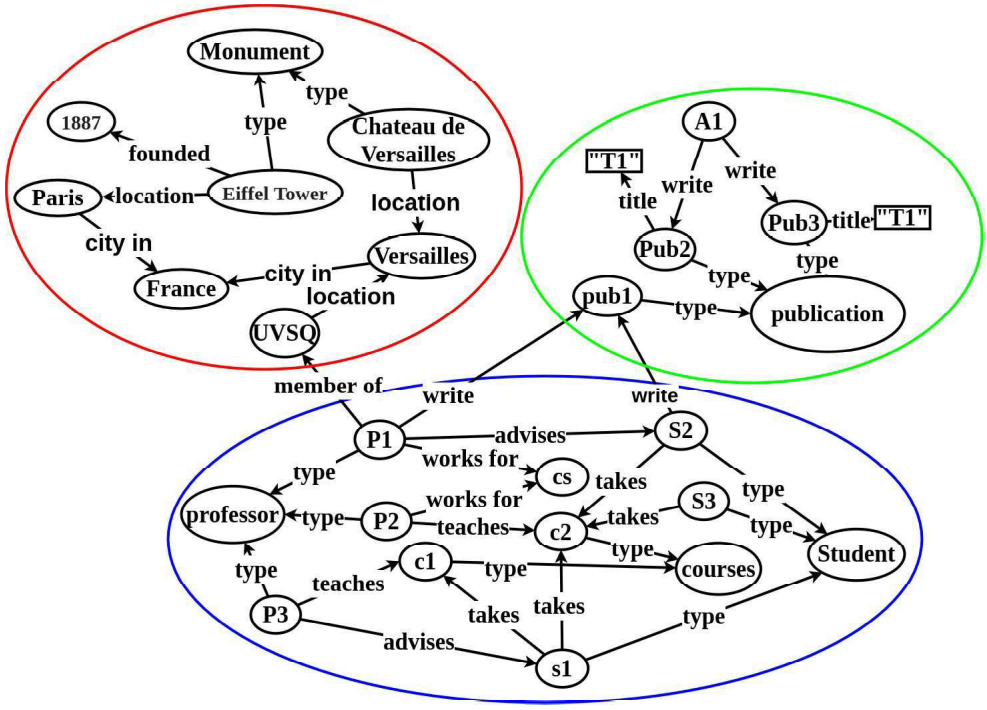


Figure 5.1: Example of RDF graph with set of themes

set of relevant nodes {"p1", "c2", "s1" , "s2"}

We can notice that the graph summary in figure 5.2 contains information about professors, students and courses only. Information about the publications and the monuments of Paris are missing. If we consider that there are three underlying themes in the initial graph, one describing publications, the second describing professors, their courses and their students, and the last one describing the monuments of Paris, we can see that only one theme is represented in the resulting summary, since the summarized graph is giving information about professors, students and courses.

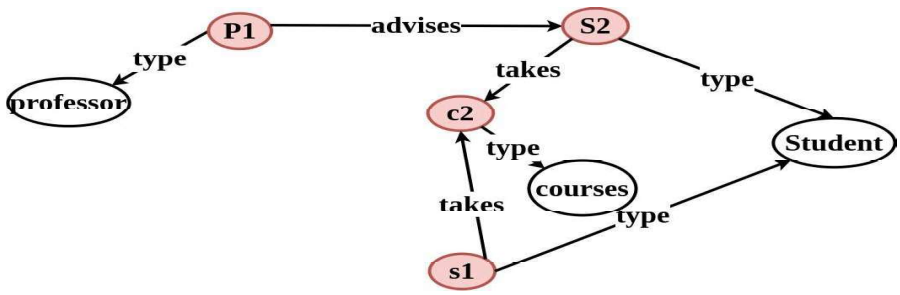


Figure 5.2: Graph summary for the graph in figure5.1

An alternative way of building this summary would be to ensure that each theme, represented by a circle in figure 5.1, is present in the result. For example, the graph of figure 5.3 presents a summary for the graph in figure 5.1 by taking into account the three themes in the graph of figure 5.1. In this summarized graph, we can find information about professors, courses, publications, and monuments, unlike the summarization in figure 5.2 that presents information about professors, courses and students only.

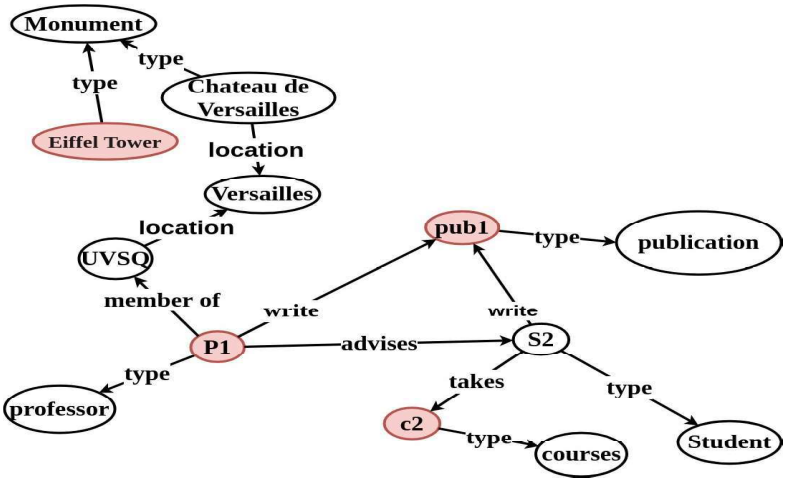


Figure 5.3: A Theme-Based Summary for the graph in figure 5.1

To create the summarized graph of figure 5.3, the three different themes in the graph of figure 5.2 should be identified, and this is the first challenge. The second challenge consists in building the theme summary based on identifying the most important nodes in each theme. After building the summary for the three different themes, we need to aggregate them and create the graph in figure 5.3 and this is the last challenge. In the next section we will present all these challenges we are facing during the building of the RDF summary.

5.3 Problem Statement

Each RDF graph can be related to more than one theme. By theme, we mean a set of elements that belongs to the same domain and describes the same topic. A theme in the RDF graph is a sub-graph containing resources and properties which are semantically related to a specific topic.

Let $G(V, E)$ be an RDF graph, we need to build a summary S_G for it such that: (i) all the themes in the graph G are represented in S_G,
(ii) each theme is represented proportionally to its size, and
(iii) the number of other nodes is minimal.

The size of the theme is calculated by computing the number of nodes in this theme, size(t_i) = number of nodes in the theme t_i. For example, the size of the theme in the blue circle of figure 5.1 is equal to 12.

From the definition of the graph summary, we can deduce that building this summary raises four main challenges. The first one is extracting the set of all the themes that are presented in the RDF data graph denoted by $T = \{t_1, t_2, t_3, ..., t_n\}$ where $t_i(V_i, E_i)$ is a sub-graph of G and $V_i \subseteq V$, $E_i \subseteq E$ and V_i and E_i related to the same topic. As we have discussed in the motivating example that identifying the relevant elements will not create a clear summary describing all the topics in the graph, then we need another way to build the graph by representing all the underlying themes of a given graph in constructing its summary. It is not obvious to identify the themes in an RDF graph, the problem is how we can extract the different themes. For example, what is the best way to determine and extract the three different themes described by the graph of figure 5.1.

To create S_G we should start from the set of themes T, and build the summarized theme S_{t_i} for each theme t_i and this is the second problem. Building the summarized theme consists in identifying the relevant elements and aggregate them. Identifying the relevant elements is not an easy task, we need to define a set of metrics that are capable to assess all the nodes in the theme based on a relevance score and quantify the important ones. For example, in the graph of figure 5.1, what are the most important nodes and how can we assess them? After selecting the relevant elements, we will face the third challenge, which is aggregating the relevant nodes corresponding to one theme and build the summarized graph for the theme t_i. For example, from the graph of figure 5.1 if we select the nodes "c2", "p1", "pub2" and "Eiffel Tower" the most important nodes, what is the best way to be used in order to connect these nodes and build the summarized graph.

To build the final summarized graph, all the summarized themes should be connected, and this is the last challenge. The problem here is to find the best paths that connect the different summarized themes and not introduce new nodes to the final summarized graph.

In this chapter, we have provided an approach to theme-based summarization. An overview of this approach and the way of building the summary is presented in the next section.

5.4 Overview of the Approach

Our approach summarizes an RDF graph based on the different topic in this graph. This is done by identifying the different themes described in the graph. There is a bunch of similar work that identifies the different themes in a graph and it is a well known problem in RDF graphs. Once the themes are identified, we need to

build a summary for each one of them. Finally we need to build a global summary by aggregating the summarized themes. Our framework, presented in figure 5.4, comprises three main components: *theme identification*, *theme summarization* and *building the final summary*.

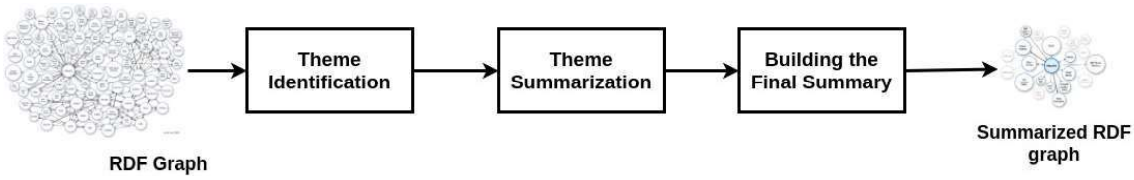


Figure 5.4: Thematic-Based Summarization Framework

The first component in our framework is the *Theme identification*. It is used to solve the challenge of extracting the underlying themes in the input RDF graph. To this end, we have used an approach proposed in [64], which identifies the different topics in the graph. The main idea of this approach is to identify highly connected areas in the graph, each of these areas representing a theme. The algorithm used in the approach is MCODE [4], a density-based algorithm producing possibly overlapping clusters. Each of the clusters represents a theme. This work is suitable to our context, but we would use any other approach such as the one presented in [14].

The second component, *Theme summarization*, was used to solve the challenge of building the summarized graph for a theme. To this end we have defined assessment tools to quantify the most relevant elements in the graph. In our approach we want to make the difference between schema-level nodes (classes) and instance-level nodes (untyped entities). Since the instance is just an individual element in the graph, while the class represents a set of entities, so we need to distinguish between the two types of elements during the identification of the relevant nodes. This component also solved the problem of connecting the relevant elements and built the theme summary. This summary is built by connecting the set of most relevant nodes in the theme while minimizing the number of non-relevant nodes.

Finally, the third component of our framework was used to solve the challenge of building the global summary from the theme summaries. once we have a summary for each theme, we need to aggregate them and build the global summary. For each theme summary, the most central node is selected, and all of them are connected to build the final graph summary. This problem can be formulated as a Steiner tree problem [28] similar to what we have presented in chapter 4. During this process, we ensure that all the themes are represented in the summary proportionally to their size. In the following sections, we will describe each of these three components.

5.5 Theme Identification in an RDF Graph

The proposed approach is to build a theme-based summary, then we need first to extract the different underlying themes in the RDF graph. Some approaches have been proposed in order to identify the underlying themes in an RDF data graph such as the approach presented in [26]. The purpose of theme identification is to group graph elements such that each group consists of semantically related elements representing a given topic.

A theme is represented in the graph by a dense area, where nodes are highly connected. To identify these dense areas, we rely on the approach proposed in [64], which proposes the use of a graph clustering algorithm to identify highly connected areas in the graph. The intuition behind this approach is that the more connected a set of nodes in the graph, the more likely they are related to the same theme. Graph clusters are generated using the MCODE graph clustering algorithm [4]. This algorithm exploits graph density to build the clusters and does not require the number of clusters as a parameter. Each generated cluster corresponds to a theme.

MCODE has initially been used in bioinformatics to identify molecular complexes in large networks of interactions based on the extraction of dense regions in the network. One of the features of MCODE is that it produces overlapping themes, which could be interested in our context. Since a given node can be related to several topics. MCODE is composed of three tasks, described in the remainder of this section, *computing vertex weights*, *building clusters* and *post-processing*.

In the first task which is *computing vertex weights*, we need to calculate a weight for all the nodes in the graph. The computation of the weights relies on the notion of *k-core*.

Definition 14. k-core A *k-core* of a graph G is a maximal connected sub-graph of G in which all vertices have a degree of at least k.

The weight of the node v is computed from the density of the largest k-core formed by this node and its directly connected neighbours. The weight of the node tends to be high in the highly connected areas in the graph. The weight of all the nodes is calculated according to the algorithm 9.

After calculating the weight for all the nodes, we will end with weighted graph. The goal of *building clusters* is to build a clustered graph based on the weighted graph we have created. The algorithm requires a parameter t ranging from 0 to 1, which expresses the level of connectivity: the higher the number, the higher the clusters density. To generate the clusters, the algorithm initiates a cluster with the node v_i with the highest weight, called a *seed node*. For each adjacent node v_j such that the difference between the weights of v_i and v_j is less than the threshold t, v_j is assigned to the same cluster as v_i. Each time a node is included in a cluster,

Algorithm 9 MCODE-Vertex-Weighting

```
procedure MCODE-VERTEX-WEIGHTING(graph G)
    W = ∅
    for v ∈ G do
        N ← find neighbors of v to depth 1
        K ← Get highest k − core graph from N
        k ← Get highest k − core number from N
        d ← Get density of K
        w(v) ← k * d
        W ← W ∪ (v, w(v))
    end for
end procedure
```

all its neighbours are processed in the same way to see if they should be included in the current cluster. Once no more nodes can be added to the current cluster, the process is performed again by initiating a new cluster considering the next seed node, i.e. the node having the highest weight that has not been assigned to a cluster.

Finally, the *post-processing* identifies the nodes that could be members of different clusters, thus producing a set of overlapping clusters. The algorithm considers all the clusters, and for each node v, if the sub-graph formed by v and its adjacent nodes is highly connected, then all the adjacent nodes are added to the current cluster.

We will rely on the themes identified in the graph for our summarization approach. In the next section we will present the different metrics we have proposed to assess the important nodes in each theme as well as the technique to build the summarized theme.

5.6 Theme Summarization

In this section, we will present the concepts and algorithms of our summarization approach. Consider the sub-graph G_i representing a theme t_i in the RDF data graph G. Summarizing G_i consists in building a graph S_{G_i} such that: (i) S_{G_i} contains the top-k most relevant nodes of G_i, and (ii) the number of other nodes is minimal. This requires the computation of a relevance score for each node in G_i.

In section 5.6.1, we describe the proposed node relevance metrics. In section 5.6.2, we present our algorithm to identify the most relevant nodes in the graph G_i. Finally, in section 5.6.3, we describe our theme summarization algorithm which

builds the summary S_{Gi} from the relevant nodes.

5.6.1 Evaluating Node Relevance in an RDF Graph

An RDF graph comprises schema-related information, such as classes or type definitions, and instance-related information, such as entities and literals. In the example given in figure 5.5, "Movie" is a type, "G" is an entity which type is "Generation", and "2012" is a literal representing the value of the property "releaseDate". From this example we can see that the instance nodes describe one single entity, while the class can represent the class it self and its entities, so they can not be assessed in the same way. In our context, we propose two different level of metrics, the first one is on the schema-level nodes and it is specific for classes and typed nodes. The second one is on the instance-level nodes such as entities and untyped nodes. In this way, we can assess the different graph nodes according to their importance and what they represent. This technique also preserves that both the schema-level nodes and the instance-level nodes are presented in the summary.

Let $G_i = (V, E)$ be the graph representing the theme t_i in a data graph G, where V is the set of vertices and E the set of edges. A common way of assessing the relevance of a node in a graph is node centrality, which is calculated as the degree of this node.

In our work, we have proposed a new method to calculate the centrality score of a class. This score is based on the class itself and the entities that are belonging to this class. For example, the class "Sci-Fi Movie" in the graph of figure 5.5 has two instances with degree one while the class "Comedy Movie" has one instance but this instance is connected to more other elements in the graph, then we can consider that this class is more central than the class "Sci-Fi Movie". The centrality is evaluated as the weighted average between the degree of the node representing this class and the average degree of the nodes representing the entities belonging to this class. Since the edges with label "type" are the common edges that connect the classes with their instances, then they should be considered once during the calculation of the centrality. For this reason, we are not considering them at the level of the classes.

Definition 15. Class Centrality. Consider the class C_i in the RDF graph G, and its set of entities E_i. The centrality score of C_i is:

$$CS(C_i) = \frac{\sum_{e_i \in E_i} degree(e_i)}{|E_i|} \times w + D(C_i) \times (1 - w)$$

where $D(C_i)$ is the degree of the node corresponding to C_i in the graph G excluding all the edges labelled "type", and w a value ranging from 0 to 1.

The average degree of the entities of a class is calculated in order to reflect the importance of this class in the graph. If the score of the classes was calculated according to the degree of the class only, then the node "Sci-Fi Movie" which has a degree of three in the graph of figure 5.5 would be more relevant than the node "Comedy Movie" which has a degree of two. But we can see that the node "Comedy Movie" has an instance with a degree of four while the two instances of the class "Sci-Fi Movie" have a degree of two; this means that according to their instances, the class "Comedy Movie" has more links with the other nodes of the graph than the class "Sci-Fi".

When computing the centrality score of a class, the weight w expresses the level of confidence in the schema compared to the instances: a high value will give more weight to the average degree of the entities, which could represent the fact that the description of the classes in the graph is incomplete; a low value will give more weight to the class degree, which could express the fact that the description of the class is complete and accurate.

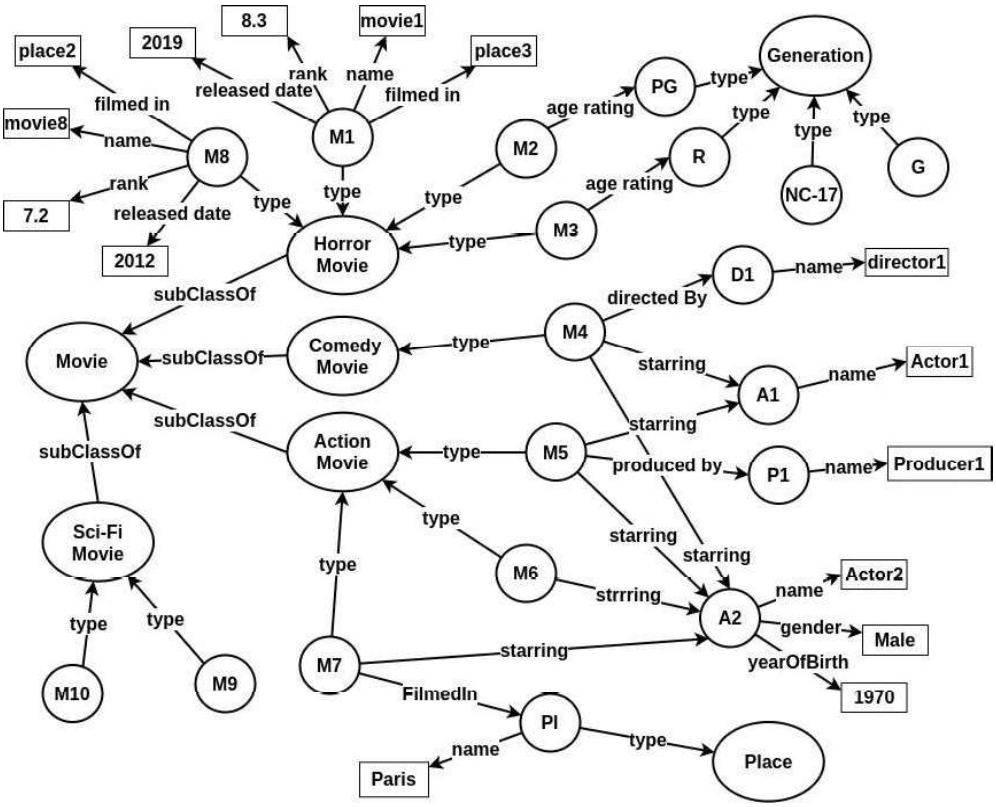


Figure 5.5: Example of an RDF Graph Describing Movies

The set of classes in the graph of figure 5.5 is C ={Horror Movie, Comedy Movie, Action Movie,Sci-Fi Movie, Generation and place}. A centrality score will

be calculated for each of these classes. For the "Horror Movie" class, the set of entities is $E_i = \{M1, M2, M3, M8\}$. The degree of each of these nodes is 2, 2, 5 and 5 respectively. Therefore, the average degree of the entities in the class "Horror Movie" is 2.75. The degree of the class "Horror Movie" is computed considering all the edges except those labelled with the "type" property and this degree is equal 1. If we assume that w is equal to 0.7, then: $CS("HorrorMovie") = \frac{14}{4} \times 0.7 + 1 \times 0.3 = 2.75$.

Centrality is not the only score that reflects the relevance of a class. Indeed, a class could have a low centrality score but a high number of instances. For example, the centrality score of the "Generation" class is equal to 0.46, which is one of the lowest in the example. However, this class has the highest number of instances. To this end we have defined the representativity of a class as follows.

Definition 16. Class Representativity. The *representativity* of a class C_i in a graph G is the number of entities of this class, i.e., the number of nodes connected to C_i by an edge labelled with the "type" property.

During the selection of the most relevant classes, we also take into account the centrality as well as the representativity score.

Besides the nodes corresponding to classes, some nodes in the graph may correspond to entities for which no type declaration has been given. The centrality of these nodes is defined hereafter.

Definition 17. Untyped Entity Centrality. The centrality of an untyped entity in a graph G is the degree of the node corresponding to the entity in G, i.e., the number of edges incoming and outgoing edges connected to this node.

During the selection of the relevance and the computation of the importance score, we do not take into account the typed nodes since they are included in the calculation of the centrality of the class.

Unlike the existing approaches, we have defined three different metrics that take into account the importance of the typed and untyped nodes in the graph and use two different ways to assess the importance of the node.

5.6.2 Identifying the Relevant Nodes

Consider the RDF graph G and the set of underlying themes T=$\{t_1, t_2, t_3, ..., t_n\}$, and assume we want to build a summary for G by identifying the top-k most relevant nodes. One of our requirements is to maximize the number of themes present in the summary, and to ensure that the themes are represented according to their respective size in G. All the themes have different sizes, we want to identify the different number of relevant nodes that have to be extracted from each one of them. To this end, we defined a formula presented below.

Definition 18. The number k_i of relevant nodes for each theme t_i in T is determined proportionally to its size in G as follows: $k_i = \frac{|t_i|}{|G|} \times k$, where $|t_i|$ is the number of nodes in the theme t_i and $|G|$ is the total number of nodes in the graph G.

The number of relevant nodes of the theme t_i is the sum of both the number of relevant classes and the number of relevant untyped entities that should be included in the summary. Hence we need to calculate these numbers. This is done so as to reflect the proportion of classes and untyped entities in t_i.

Definition 19. Let p_i be the proportion of classes and their entities in t_i.

- The number of classes in the summary of t_i is defined as follows: $nc_i = p_i \times k_i$.
- The number of untyped entities in the summary of t_i is defined as follows: $ne_i = (1 - p_i) \times k_i$.

The relevance of an untyped entity is evaluated using the centrality metric defined in the previous section. The selection process will extract the top-ne_i most relevant untyped entities. The relevance of a class can be considered according to two facets, its centrality or its representativity. In the graph of figure 5.5 the top-3 most central classes are "Comedy Movie", "Action Movie" and "Horror Movie", while the top-3 most representative ones are "Horror Movie", "Action Movie" and "Generation". In our approach, we take into consideration both facets. It is up to the user to determine if he wants to give more importance to the connection between the classes and the other nodes in the graph or to give more weight to the number of instances represented by the class. To this end, a weight w specifies the importance of centrality with respect to representativity is defined. According to w, we calculate two different number, the first one is the number of relevant classes according to centrality metric and the second one is the number of relevant classes according to representativity metric.

Definition 20. If nc_i classes are to be selected, and w_{cr} is a weight between zero and one then $ncc = w_{cr} \times nc_i$ is the number of classes having the highest centrality to be selected from the theme and $ncr = (1 - w_{cr}) \times nc_i$ is the number of classes having the highest representativity to be selected.

After defining the metrics used to assess the importance of the different elements in the theme and presenting the way of calculating the number of k relevant classes and untyped nodes, the principle of our approach by describing the identification of the relevant nodes in a given theme is presented. After identifying the themes, we summarize them by calculating the importance of the nodes according to our definition of the centrality and representative for the classes and the centrality for the untyped nodes. Our summarization process is divided into tasks; the

first one is on the schema information level (classes and their instances). This stage task consists in calculating the classes' importance based on the class's representative and centrality and then selecting the most important classes according to the calculated scores. The second task consists in calculating the importance for the untyped nodes, and the most important nodes are selected to be in the summarized graph. At the end, all the selected nodes from both stages are connected to have the final summarized theme. By this approach, we ensure that the schema-related information and the instance-related information are evaluated with appropriate metrics and the schema-related information we consider both the properties of the class and the instances of the class. Our summary reflects the representation of both information in the theme.

Algorithm 10 describes our approach, it takes as an input the theme to be summarized, the number of relevant nodes k_i, the number of relevant classes nc_i, and the weight w_{cr} that allows to specify if we want to give a higher weight to the centrality or to representativity. First, for each class, we evaluate both the representativity and the centrality and for each untyped nodes we evaluated the the centrality. From the list of classes we create two different sorted sets, the first one is sorted according to the centrality score and the second one according to the representativity score (lines 3 and 4). The untyped nodes are also sorted according to the centrality score (line 5). The number of relevant classes according to centrality and representativity scores is calculated using the weight w_{cr} (lines 6 and 7). Then the number of relevant nodes is deduced from k_i and nc_i (line 8). The most central and the most representative classes from the two sorted lists of classes are selected (lines 9). After choosing the relevant classes, the relevant untyped nodes are selected; this is done by choosing the most central untyped nodes after sorting them according to their centrality (line 10). All the relevant nodes for the theme t_i are returned as an output.

5.6.3 Building a Theme Summary

In the previous section, we have our relevance metrics and define the different components of the summary. After selecting the relevant nodes, the summarized theme is built by connecting them. The problem consists of aggregating and connecting the relevant nodes to have one sub-graph. The relevant nodes should be connected in a way we can introduce them as minimal as possible to the non-relevant elements.

In chapter 4 we have presented an approach to aggregate the relevant elements in RDF graph during keyword search. The relevant elements could be of three different types: a node, an edge and a sub-graph. In this section our goal is to build a theme summary after selecting a set of relevant nodes. The problem is similar, the only difference is that we consider here only nodes as relevant elements.

Algorithm 10 Identifying Relevant Nodes

Input: Graph G_i, set of classes C, set of untyped entities E, number of relevant nodes k_i, number of relevant classes nc_i and weight w

Output: Set of relevant nodes R_i

1: **procedure** RELEVANT NODES(G_i, C, E, k_i, nc_i, w)
2: $R_i = \phi$ /*set of relevant nodes*/
3: $CCS \leftarrow sort(Classes)$ /*sort classes according to the centrality score*/
4: $CRS \leftarrow sort(Classes)$ /*sort classes according to the representativity score*/
5: $ECS \leftarrow sort(entities)$ /*sort untyped entities according to the centrality score*/
6: $l = w * nc_i$ /*number of classes according to centrality*/
7: $m = (1 - w)nc_i$ /*number of classes according to representativity*/
8: $n = (k_i - nc_i)$ /*number of relevant untyped entities*/
9: $R_i \leftarrow top{-}l\ classes\ of\ CCS \cup top{-}m\ classes\ of\ CRS$ /*Identify relevant classes*/
10: $R_i \leftarrow R_i \cup top - n\ entities\ of\ ECS$ /*Identify relevant untyped entities*/
11: **return** R_i
12: **end procedure**

We have proposed the same technique used to solve the aggregation problem by stating it as a Steiner tree problem [28]. We have adapted the distance network heuristic(DNH) [52], which has an approximation ratio equal to $2-\frac{2}{p}$ where p is the total number of the terminal nodes. The weight of the path equals the length of this path in the non-weighted graphs. Instead of having the length, we proposed a score that reflects the importance of the nodes in this path. The intuition behind this definition is to have as many central nodes as possible in the summarized graph. The path containing the most central nodes will better reflect the RDF graph. In this way, we can have most central paths in our summarized graph. For example, if we have two different paths, P1 and P2 have the same length, which equals 4. This means that there are 5 nodes in each path. Suppose that the nodes in P1 having a higher degree than the nodes in path P2; this means that the nodes in P1 are important and are connected to more elements in the RDF graph. This score is called the centrality score and it is based on the degree of the nodes in the path.

Definition 21. The centrality score of a path is calculated as follows. Let p=[(v_1, e_1, v_2)(v_2, e_2, v_3)....(v_{n-1}, e_{n-1}, v_n)] be the path connecting the two terminal nodes v_1 to v_n; the centrality score of p $CS(p) = \frac{\sum_{n}^{i=1} deg(v_i)}{n}$ is the average degree of the nodes in the path.

Building the Steiner tree by using DNH consists first of computing the distance graph DG_i from G_i by using the shortest path between all the relevant nodes. Let i and j be two relevant nodes, n_{ij} the number of edges in the shortest path connecting i and j. The computation of DG_i will be based on n_{ij}. The next step is to compute the minimum spanning tree starting from DG_i. This is done by using the Kruskal algorithm, this algorithm consists in creating a forest F (a set of trees) where each node in the graph is a separate tree, and creating a set S containing all the edges in the graph. The algorithm finds an edge with minimal weight connecting any pair of trees in the forest without forming a cycle. If several edges match these criteria, then one is chosen arbitrarily. The edge is added to the spanning tree, and this step is repeated until there are |V|-1 edges in the spanning tree (where |V|is the number of vertices).

We have also used the same adaptation of the Kruskal's Algorithm we have proposed in the chapter 4. The adaptation consists of using additional metrics during the assessment of the path instead of just using the length of the path. In our approach, if two edges have the same weight, we will then check the centrality of these paths to decide which path will be selected. The path with the highest centrality degree will be selected.

After creating the minimum spanning tree, we need to replace the edges with the actual paths to create the summarized theme.

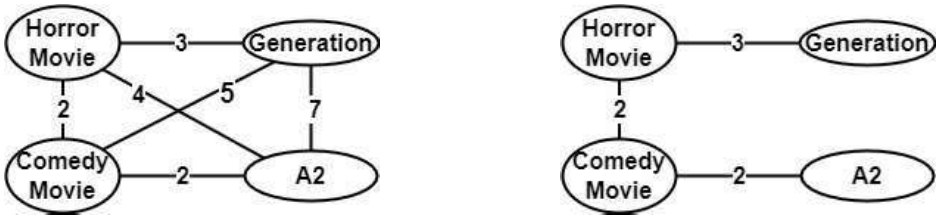
The adapted distance network heuristic is described by algorithm 11. The distance graph is first computed (line 2), then the minimum spanning tree is deduced by using the distance graph (line 3). The edges in the minimum spanning tree will be replaced by the actual paths from the theme (line 4). The final summarized theme will be returned as an output to this algorithm (line 5).

Algorithm 11 Building Theme Summary

1: **procedure** BUILDINGTHEMESUMMARY(G_i, R_i)
2: $DG_i \leftarrow DistanceGraph(G_i)$ /*Compute the complete distance graph*/
3: $MST_i \leftarrow MinimumSpanningTree(DG_i)$ /*Compute the Minimum Spanning Tree*/
4: $G'_i \leftarrow Replace(G_i, MST_i)$ /*replace each edge in MST_i by a corresponding minimum cost path in G_i; if several paths are found, select the path having the highest centrality degree weight*/
5: **return** G'_i
6: **end procedure**

Assume that R={"Horror Movie", "Comedy Movie", "Generation", "A2"} is the set of relevant nodes of the graph in figure 5.5. First we generate the distance graph by connecting each node in R with all the other relevant nodes in R by

an edge labelled with the shortest distance between them. For example, from the graph of figure 5.5 we know that the shortest distance between the two nodes "Horror Movie" and "Comedy Movie" is two, then in the distance graph we create an edge connecting "Horror Movie" with "Comedy Movie" and labelled with two. The distance graph is shown in figure 5.6a.



(a) Distance Graph (b) Steiner Tree

Figure 5.6: From Distance Graph to summarized graph

From this distance graph we need to build the Steiner tree by using the adapted Kruskal's algorithm. Figure 5.6b shows the generated Steiner tree, each edge in this graph need to be replaced by the corresponding path from the initial graph. For example the edge connecting "Horror Movie" with "Comedy Movie" and labelled by two should be replaced by the corresponding path from the graph of figure 5.5. After replacing the edges in the Steiner tree with the paths from G, we obtain the sub-graph of figure 5.7.

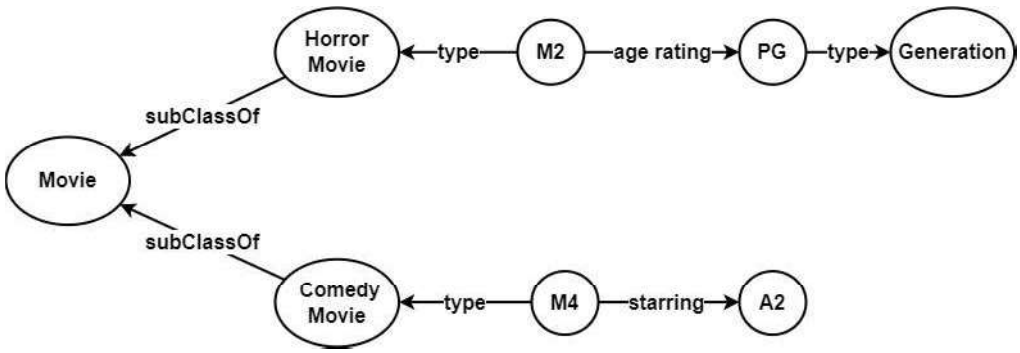


Figure 5.7: Final Summarized Graph

After creating the summarized graph we can observe that the classes and their instances are presented in the summary, for example in the graph of figure 5.7 the class "Horror Movie" and its instance "M2" are presented and it is some kind of redundancy. Our goal is to remove the instances and just keep the classes and the untyped nodes.

Suppose in a graph G we have class C_1 with three instances i_1, i_2 and i_3, in the new graph this class should be replace with one node labelled C_1. Assume there exists an edge e between tow nodes n_i and n_j: If n_i is of type C_1 and n_j is of type C_2 then in the new graph we need to add an edge e between the nodes C_1 and C_2. If n_i is of type C_1 and n_j is untyped node, then we need to add the edge e connecting the node C_1 with the node n_j in the new generated graph.

The summarized graph after removing the typed instances is presented in figure 5.8. Finally, we enrich the resulting summary by adding some edges corresponding to properties that describe the selected nodes, such as "label" and "name". For example the edge labelled "name" has been added for the node "A2".

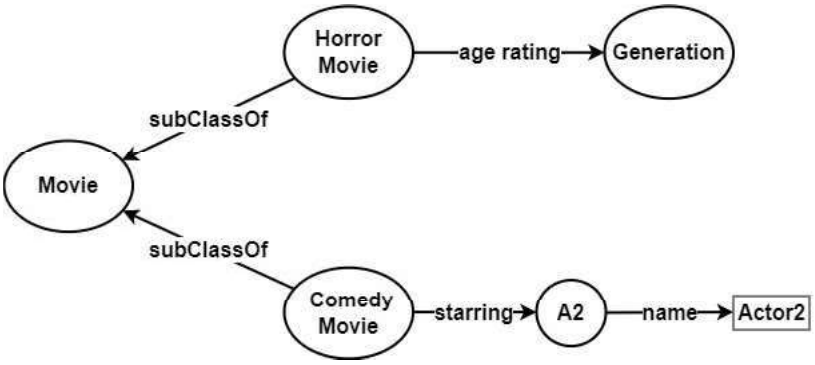


Figure 5.8: Final Summarized Graph

5.7 Building the final summary

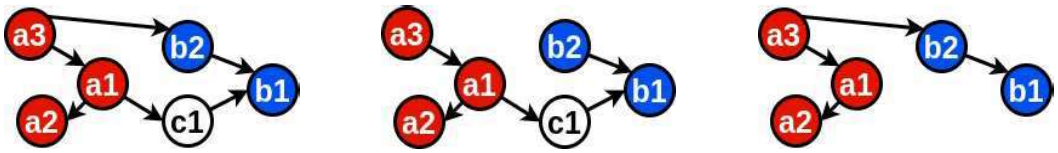
In the previous sections, we have seen how we can identify the different themes in the graph and for each one we have produced a summary. Our aim now is to build the final summary from all the summaries of the extracted themes. Let S ={ S_{G1}, S_{G2} ... S_{Gn}} be the set of summaries corresponding to the themes in a graph G such that each S_{Gi} is the summary of a sub-graph corresponding to the theme t_i.

Our goal is to connect all the summarized themes in S in order to create the final summary. To this end, we have proposed to select from each summarized theme S_{Gi} the most central node c_i and connect these nodes to build the final summary. Moreover, we need to find that minimal sub-graph in which all the nodes c_1, c_2,..., c_n are connected. This problem is similar to connecting the relevant nodes in the theme and build the theme summary.

Let C = $\{c_1, c_2, ..., c_n\}$ be the set of most central nodes in each theme where c_i is the central node in the theme t_i and let G be the initial RDF graph. The problem is to find the minimal sub-graph in G and containing all the elements in

set C. This problem can be stated as the Steiner tree problem, where the terminals are the elements in set C.

We have also used the DNH algorithm to solve this problem; the algorithm starts by creating the distance graph based on the weights of the edges in the graph. The default weight of each edge equals one. Let us consider the graph in figure 5.9a where two theme summaries are represented in red and blue respectively; consider "a1" and "b1" the most central nodes of the two themes.

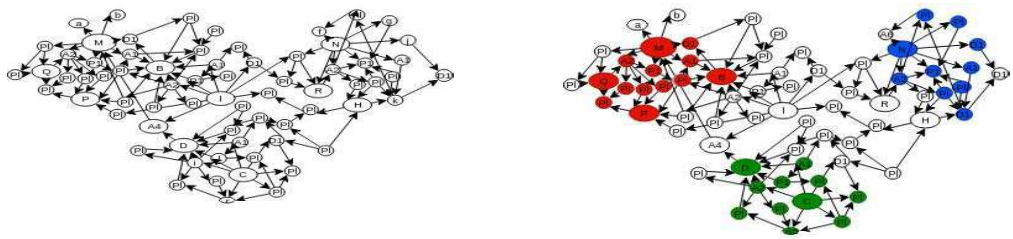


(a) Initial graph (b) Summary Using Path P1 (c) Summary Using Path P2

Figure 5.9: Path Selection for Building the Final Summary

The shortest path connecting nodes "a1" and "b1" is P1=[a1, c1, b1], but we can observe that choosing this path will introduce the node "c1", while the path P2=[a1, a3, b2, b1] connects the two themes without introducing any additional node. We then modify the weights such that the weight of an edge equals to zero if its origin and destination nodes are in one of the theme summaries and equals to one otherwise. This leads to the graph depicted in figure 5.9c. Path P2 will be selected as its weight is equal to one while the weight of P1 is equal to two.

After creating the distance graph based on the weight we have proposed, the next step is to extract the minimum spanning tree from the distance graph by using the Kruskal's algorithm. All the edges in the minimum spanning tree will be replaces by the path from the RDF graph to connect the summarized themes. At the end, all the central nodes will be connected forming a sub-graph. This sub-graph is the final summarized graph. Algorithm 12 presents all the steps of building the global summary.



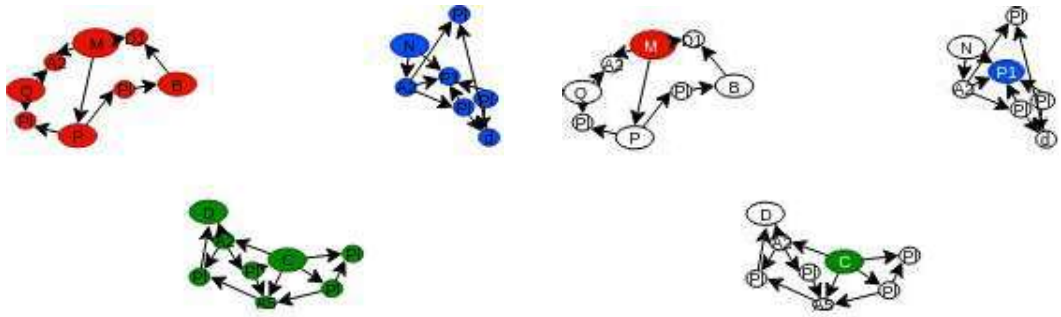
(a) RDF graph (b) Identified themes

Figure 5.10: Example of a RDF graph with three different themes

Algorithm 12 Building Global Summary

```
procedure BuildingGlobalSummary(S ={ S_G1, S_G2 ... S_Gn}, G)
    G_weight ← weight(G)
    for S_Gi ∈ S do
        c_i ← Central(S_Gi)
        C.add(c_i)
    end for
    DG ← DistanceGraph(G_weight, C) /*Compute the complete distance graph*/
    MST ← MinimumSpanningTree(DG) /*Compute the Minimum Spanning Tree*/
    G_Global ← Replace(G, MST)
    return G_Global
end procedure
```

Let us consider the graph in figure 5.10a, the themes are identified and colored in red green and blue in the graph in figure 5.10b.



(a) summarized themes (b) Select most central nodes

Figure 5.11: Summarized Themes and the Most Central Nodes in these Themes

Graph in figure 5.11a have the summarized graphs of the themes identified in the figure 5.10b. Figure 5.11b shows the most central nodes in the summarized themes of figure 5.11a. After applying the DNH on the selected nodes in in figure 5.11b, we get the graph in figure 5.12.

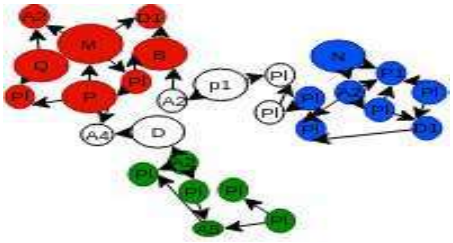


Figure 5.12: Final Summary

5.8 Evaluation

In this section, we describe our experiments to validate the performances of our approach and to compare the resulting summary with the one of a baseline approach where summarization is performed solely on the basis of centrality, without considering themes.

Our approach is implemented in Java, we have used the Jena API for the manipulation of RDF data. The Jung API is used for graph manipulation and visualization. All the experiments have been done on Intel Core i7 with 32GB RAM.

We have used three datasets: AIFB, DBpedia and Olympics. AIFB is a dataset containing data taken from the AIFB institute, at Karlsruhe University. It describes entities of research communities such as persons, organizations, publications (bibliographic metadata) and their relationships. The dataset contains 8281 entities and 29 233 triples. DBpedia is a project aiming to extract structured content from the information created in the Wikipedia project. The extracted data is related to movies their title, actors, director, released date among other properties. This dataset contains 30 793 triples. The Olympics dataset contains information on 120 years of Olympic history, among which information about athletes and their medal results from Athens 1896 to Rio 2016. This dataset contains 1 781 625 triples.

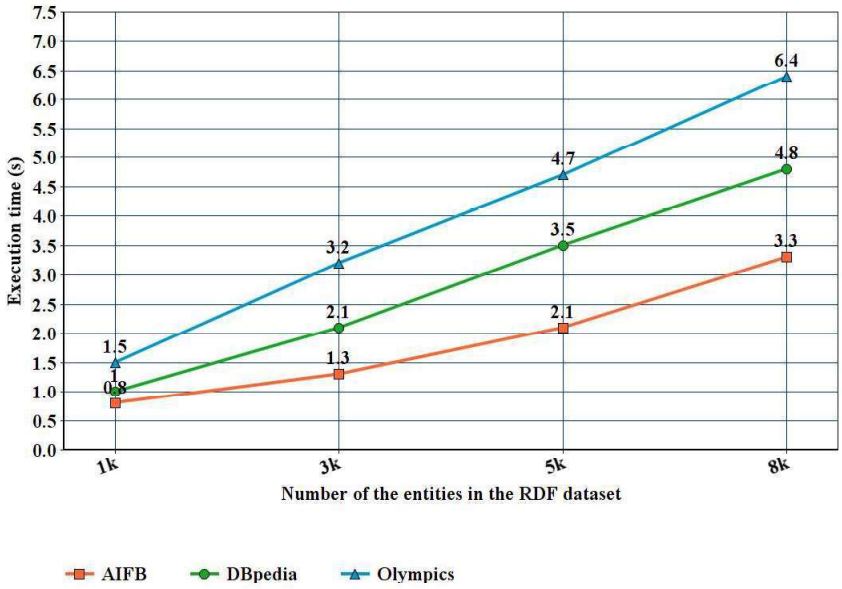


Figure 5.13: Average Execution Time

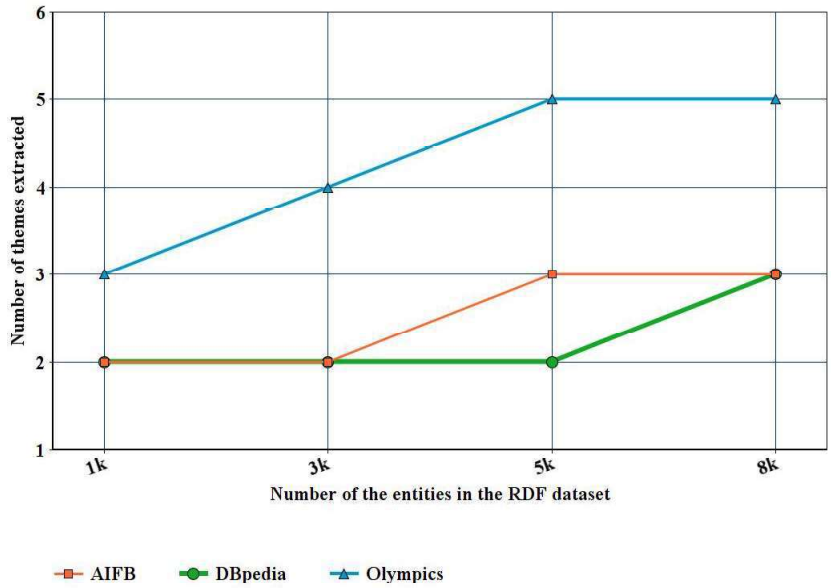


Figure 5.14: Number of Extracted Themes

Figure 5.13 shows the execution time with respect to the size of the dataset.

The graph shows that the execution time increases when the number of triples increases for all the datasets. We can also see that the execution time for the Olympics is greater than the execution time of AIFB and DBpedia. The number of themes extracted from Olympics is greater than the number of themes extracted from AIFB and DBpedia according to the graph in figure 5.14, which explains the increase in the execution time for the Olympics dataset.

Number of entities	Top-k	Number of extracted themes	Size of summarized graph (entities)	Execution time (sec)
1k	10	3	16	1.5
10k	20	7	35	9.5
100k	40	8	53	35.8
200k	80	8	113	76.1

Table 5.1: Number of extracted themes and the execution time for samples from Olympics dataset

We have extracted 4 samples from the Olympics dataset with a respective size of 1k, 10k, 100k and 200k and executed our summarization process on each of them. The results are presented in table 5.1. We can see that the number of extracted themes and the execution time increase as the number of entities increases. As described in section 5.6, we distinguish between typed and untyped entities.

To check the effectiveness of our approach, we have used 8 simple graphs (the number of entities in those graphs varies from 24 to 34) and asked five users to select the relevant elements in those graphs. We have then compared their answers to the results obtained by our summarization algorithms(TBA). We have also compared our results with the application of a baseline summarization approach(BA), where the top-k nodes are selected based on their degree. We have computed the precision at k as follows:

$$P@K = \frac{NumberOfrelevantelements}{K}$$

According to table 5.2, we can see that P@K ranges between 0.84 and 0.98 for our thematic-based approach. These results are better than those obtained using the baseline approach, which range between 0.62 and 0.88; on these graphs, the results achieved with the thematic-based approach were more accurate according to the users.

Data	G1		G2		G3		G4		G5		G6		G7		G8	
K	5	10	5	10	5	10	5	10	5	10	5	10	5	10	5	10
TBA	0.88	0.84	0.84	0.88	0.88	0.98	0.84	0.86	0.96	0.94	0.92	0.94	0.92	0.94	0.96	0.96
BA	0.84	0.88	0.72	0.80	0.72	0.84	0.88	0.82	0.8	0.72	0.68	0.62	0.68	0.72	0.64	0.62

Table 5.2: Top-k precision

5.9 Conclusion

In this chapter, we have provided an approach for summarizing RDF graphs based on theme extraction. Unlike the existing approaches, our proposed idea is to take into account all the underlying themes in the RDF graph when building the summary. In our approach, we proposed different metrics to assess the different types of nodes in the graph. In this case we ensure that both the schema-level information as well as the instance-level information are assessed correctly and presented in the summarized graph. We stated the problem of connecting the relevant nodes as the Steiner tree problem, and we have introduced a novel way to assess the paths while building the summarized theme. This assessment is based on the degree of the nodes in the path, and it reflects how this path is centralized in the RDF graph. We have used it to build the theme summary. For building of the global summary, we have proposed an approach based on connecting the summarized themes. We have also conducted some experiments to evaluate our approach.

In the present work, the resulting summary contains nodes and edges which exist in the initial graph. A possible extension would be to insert in the resulting summary properties of nodes that do not exist in the graph but have some semantic relationships with existing nodes; for instance, this could lead to replacing a path in the summary with a single edge conveying the same meaning, or replacing two classes by their super-type if they are close in meaning. This could be possible through the use of external knowledge sources such as domain ontologies or exploiting the vocabularies used in the considered dataset.

Chapter 6

Conclusion

This thesis explored differed ways of providing a support for the meaningful exploration of RDF datasets. To this end we have addressed two distinct problems keyword search over RDF dataset and summarization of an RDF graphs. The keyword search approach consists of having a set of keywords as an input and returns a set of sub-graphs as an output. Each sub-graph will contain elements related to the query keywords. The summarization allows the user to have a brief description of the data; this is done by extracting relevant elements from the graph. The solution turned quickly into other problems; in the keyword search we have faced the problem of matching the query keywords with the different concepts used in the RDF graph elements. Moreover, in the summarization, we had the problem of creating a summarized graph representing all the different topics relies in the RDF graph.

6.1 Summary of the contributions

In chapter 3, we presented an approach for keyword search over RDF dataset. The contribution of this work was to introduce a way to bridge the terminological gap between the keyword query and the terms used in the RDF dataset. In our approach, we integrated the use of a knowledge base to ease the exploration and querying of the RDF dataset in terms that are not necessarily used in the elements of the dataset. We have also used the patterns [63] as an external knowledge base to enrich the matching elements for the keywords in the query. The results of different experiments carried out on several data sources have shown that the integration of external knowledge improves the number of matching elements and obtains a result better meets the needs of the user.

After finding the matching elements for each keyword in the query, we focused on the aggregation of those elements to create the final meaningful sub-graph

result. Our second contribution was in introducing the centrality score to measure the importance of the path connecting two different elements in a graph, we also have adapted an algorithm used to solve the Steiner tree problem after stating our problem as a Steiner tree problem. After creating all the possible solutions, a ranking method is presented based on the importance of the matching elements and the size of the graph compared with the number of keywords in the query.

Summing up, we handled three problems related to keyword search: matching elements, aggregating these matching elements, and ranking the final set of subgraph results. The experiments showed that the solutions gave an interesting improvement in the quality of the results created.

In this thesis, we also presented thematic-based summarization. This approach aims to provide the user with a brief and obvious description of the information carried by the RDF graph in a summarized way. As the RDF graph can have more than one theme, each theme describes a topic; we need our summarization to reflect the different themes in the RDF graph. Our contribution was to introduce thematic-based summarization in the approach. The summarized graph in this approach is build based on the extracted themes from the RDF graph. After extracted the themes, we need to build a summary for each one of them. To this end, we have presented the method to evaluate the importance of the classes based on their centrality score and representativity score. Our contribution was to take into account the relevant untyped nodes based on their centrality score in our summarization. In other words, our summary will have information about schema level and instance level. The summarized theme is built by connecting and aggregating the relevant nodes selected according to the different relevance metrics. At the end, a summarized graph is built by aggregating all the summarized themes extracted from the graph. In this process, we ensure that all the themes, hence all the topics, are represented in the summarized graph. In this work, we handled two problems related to the summarization: representing all the themes and choosing the relevant nodes to be presented in the graph.

6.2 Future Works

Our work still has a high potential for future improvements. For a keyword search, we can improve the extraction of the semantic relations between the keyword query and the terms in RDF data by using text mining and machine learning techniques. For example we can use word2vec[60] which is a technique for natural language processing used to produce word embeddings. It uses a neural network model to learn word associations from a large corpus of text. Once trained, such a model can detect synonymous words. These words can be used to bridge the terminological gap between the keyword query and the RDF dataset.

Another improvement can be on the level of numbers, such as the ability to match the numbers represented by integers by the number represented by characters, for example, matching the keyword query "one" with a node having the value "1". Moreover, we can work on using greater than or less than in the keyword query, for example "greater than five" will be matched with all the elements in the RDF graph that have a number greater than five.

As the number of data on the web increases, we are facing a scalability issue problem. To this end we can study this problem and how we can query a big RDF dataset; what is the best way can be done for the matching process, can we decompose the RDF graph into different themes and lunch the query keyword on each theme in a parallel way is this solution efficient? What about the aggregation problem? How can the relevant nodes be connected in massive datasets?

On the level of summarization, in the present work, the resulting summary contains nodes and edges which exist in the initial graph. A possible extension would be to insert in the resulting summary properties of nodes that do not exist in the graph but have some semantic relationships with existing nodes; for instance, this could lead to replacing a path in the summary with a single edge conveying the same meaning, or replacing two classes by their super-type if they are close in meaning. This could be possible through the use of external knowledge sources such as domain ontologies or exploiting the vocabularies used in the considered dataset.

Finally, we can combine the two different methods of exploration into one single approach. It can be decomposes into two different processes, the summarization process and the keyword search process. After generating the summarization graph from the first process, the user can utilize this summary to have an idea about the dataset and what vocabulary it contains. This will helps him to issue the query keywords. These keywords will be an input to the second process which is keyword search that generates a ranked set of sub-graphs representing an answer to the query.

Appendix A

Résumé en Français

Un nombre croissant de sources de données sont publiées sur le web, exprimées dans les langages proposés par le W3C comme RDF, RDFS et OWL. Ces sources représentent un volume de données sans précédent, disponible pour les utilisateurs et les applications. Afin d'identifier les sources les plus pertinentes et de les utiliser, il est nécessaire d'en découvrir le contenu, par exemple au moyen de requêtes écrites en Sparql, le langage d'interrogation proposé par le W3C pour les sources de données RDF. Mais cela nécessite, en plus de la maîtrise du langage Sparql, de disposer de connaissances sur le contenu de la source en termes de ressources, classes ou propriétés qu'elle contient. L'objectif de ma thèse est d'étudier des approches permettant de fournir un support à l'exploration d'une source de données RDF. Pour cela, nous avons proposé deux approches complémentaires, la recherche mots clés et le résumé d'un graphe RDF. La recherche mots clés dans une source de données RDF permet d'interroger cette source sans avoir de connaissances préalables sur son contenu en termes de ressources ou de propriétés. Elle renvoie un ou plusieurs sous-graphes en réponse à une requête exprimée comme un ensemble de termes à rechercher. Chaque sous-graphe est une réponse possible à la requête, et est obtenu par l'agrégation d'éléments extraits du graphe initial, chacun correspondant à un mot clé de la requête. Les sous-graphes retournés sont classés en fonction de leur pertinence. La recherche par mot clé dans des sources de données RDF soulève les problèmes suivants : (i) l'identification pour chaque mot clé de la requête des éléments qui lui correspondent dans le graphe considéré, en prenant en compte les différences de terminologies existant entre les mots clés et les termes utilisés dans le graphe RDF, (ii) la combinaison des éléments du graphe qui correspondent aux mots clés pour construire un sous-graphe résultat en utilisant des algorithmes d'agrégation capable de déterminer la meilleure façon de relier ces différents éléments, et enfin (iii), comme il peut exister plusieurs éléments du graphe qui correspondent à un même mot clé, et par conséquentplusieurs sous-graphes résultats, il s'agit d'évaluer la pertinence de ces sous-graphes par l'utilisation de métriques appropriées. Dans notre travail, nous avons proposé une approche de recherche par mot clé qui apporte des solutions aux problèmes ci-dessus. Pour cela, nous utilisons une source de connaissances externe qui fournit un ensemble de relations sémantiques entre concepts, et nous avons proposé un processus de matching qui exploite ces relations pour résoudre l'hétérogénéité terminologique entre le graphe RDF considéré et les mots clés de la requête. Le matching renvoiedétermine pour chaque mot clé un ou plusieurs éléments du graphe RDF, un élément pouvant être un nœud, un arc ou encore un sous-graphe. Nous avons proposé une approche permettant d'agréger ces éléments en exploitant les chemins existants entre eux et en sélectionnant les meilleurs, afin de construire les sous-graphes résultats. La construction d'une vue résumée d'un graphe RDF, qui mettrait en évidence ses éléments les plus représentatifs, pourrait être utile afin de déterminer si ce graphe correspond aux besoins d'un utilisateur particulier ; une telle vue résumée peut être utilisée comme support pour explorer le graphe, en ciblant par exemple ses éléments les plus représentatifs. Dans notre travail, nous avons proposé une approche de résumé originale fondée sur l'identification des thèmes sous-jacents dans un graphe RDF. Notre approche de résumé consiste à extraire ces thèmes, puis à construire le résumé en garantissant que tous les thèmes sont représentés dans le

résultat. La construction d'un tel résumé soulève les questions suivantes : (i) comment identifier les thèmes dans un graphe RDF ? (ii) quels sont les critères adaptés pour identifier les éléments les plus pertinents dans les sous-graphes correspondants à un thème ? (iii) comment connecter les éléments les plus pertinents pour créer le résumé d'un thème ? et enfin (iv) comment générer un résumé pour le graphe initial à partir de l'ensemble des résumés de thèmes ? Dans notre travail, nous avons proposé une approche qui fournit des réponses à ces questions. Elle permet d'identifier les thèmes, qui correspondent aux régions denses du graphe RDF. Pour chaque thème, un résumé est construit en identifiant d'abord les sommets les plus représentatifs, et nous avons pour cela introduit une extension de la notion de centralité d'un nœud dans le graphe. Puis les résumés de thèmes sont connectés en utilisant une mesure de pertinence des chemins entre les sommets représentatifs. Le résumé final est construit par l'agrégation des résumés de thèmes de telle sorte que tous les thèmes soient représentés proportionnellement à leur importance dans le graphe initial.

Bibliography

[1] Sanjay Agrawal, Surajit Chaudhuri, and Gautam Das. "DBXplorer: enabling keyword search over relational databases". In: *Proceedings of the 2002 ACM SIGMOD international conference on Management of data*. 2002, pp. 627–627.

[2] Charles J Alpert, Andrew B Kahng, and So-Zen Yao. "Spectral partitioning with multiple eigenvectors". In: *Discrete Applied Mathematics* 90.1-3 (1999), pp. 3–26.

[3] Samur Araujo and Daniel Schwabe. "Explorator: a tool for exploring RDF data through direct manipulation". In: *LDOW*. 2009.

[4] Gary D Bader and Christopher WV Hogue. "An automated method for finding molecular complexes in large protein interaction networks". In: *BMC bioinformatics* 4.1 (2003), p. 2.

[5] Stephen T Barnard and Horst D Simon. "Fast multilevel implementation of recursive spectral bisection for partitioning unstructured problems". In: *Concurrency: Practice and experience* 6.2 (1994), pp. 101–117.

[6] Sonia Bergamaschi et al. "Keyword search over relational databases: a metadata approach". In: *Proceedings of the 2011 ACM SIGMOD International Conference on Management of data*. 2011, pp. 565–576.

[7] Tim Berners-Lee. *Weaving the Web: The original design and ultimate destiny of the World Wide Web by its inventor*. Harper San Francisco, 1999.

[8] Tim Berners-Lee, James Hendler, and Ora Lassila. "The semantic web". In: *Scientific american* 284.5 (2001), pp. 34–43.

[9] Gaurav Bhalotia et al. "Keyword searching and browsing in databases using BANKS". In: *Proceedings 18th International Conference on Data Engineering*. IEEE. 2002, pp. 431–440.

[10] Nikos Bikakis and Timos Sellis. "Exploration and visualization in the web of big linked data: A survey of the state of the art". In: *arXiv preprint arXiv:1601.08059* (2016).

[11] Qingqing Cai and Alexander Yates. "Large-scale semantic parsing via schema matching and lexicon extension". In: *Proceedings of the 51st Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*. 2013, pp. 423–433.

[12] Stéphane Campinas, Renaud Delbru, and Giovanni Tummarello. "Efficiency and precision trade-offs in graph summary algorithms". In: *Proceedings of the 17th International Database Engineering & Applications Symposium*. ACM. 2013, pp. 38–47.

[13] Stephane Campinas et al. "Introducing RDF graph summary with application to assisted SPARQL formulation". In: *2012 23rd International Workshop on Database and Expert Systems Applications*. IEEE. 2012, pp. 261–266.

[14] Silvana Castano, Alfio Ferrara, and Stefano Montanelli. "inWalk: Interactive and Thematic Walks inside the Web of Data." In: *EDBT*. Citeseer. 2014, pp. 628–631.

[15] Šejla Čebirić, François Goasdoué, and Ioana Manolescu. "A framework for efficient representative summarization of RDF graphs". PhD thesis. Inria Saclay Ile de France; Ecole Polytechnique,; Université de Rennes 1 [UR1], 2017.

[16] Šejla Čebirić, François Goasdoué, and Ioana Manolescu. "Query-oriented summarization of RDF graphs". PhD thesis. INRIA Saclay; Université Rennes 1, 2017.

[17] Šejla Čebirić et al. *Compact Summaries of Rich Heterogeneous Graphs*. 2018.

[18] Šejla Čebirić et al. "Summarizing semantic graphs: a survey". In: *The VLDB Journal* 28.3 (2019), pp. 295–327.

[19] Qun Chen, Andrew Lim, and Kian Win Ong. "D (k)-index: An adaptive structural summary for graph-structured data". In: *Proceedings of the 2003 ACM SIGMOD international conference on Management of data*. 2003, pp. 134–144.

[20] Sara Cohen et al. "XSEarch: A semantic search engine for XML". In: *Proceedings 2003 VLDB Conference*. Elsevier. 2003, pp. 45–56.

[21] Marie-Catherine De Marneffe and Christopher D Manning. *Stanford typed dependencies manual*. Tech. rep. Technical report, Stanford University, 2008.

[22] Mike Dean and Guus Schreiber. "OWL Web Ontology Language Reference: http://www. w3. org". In: *Retrieved from* (2004).

[23] Leonidas Deligiannidis, Krys J Kochut, and Amit P Sheth. "RDF data exploration and visualization". In: *Proceedings of the ACM first workshop on CyberInfrastructure: information management in eScience*. 2007, pp. 39–46.

[24] Bolin Ding et al. "Finding top-k min-cost connected trees in databases". In: *2007 IEEE 23rd International Conference on Data Engineering*. IEEE. 2007, pp. 836–845.

[25] Marek Dudáš, Vojtěch Svátek, and Jindřich Mynarz. "Dataset summary visualization with LODSight". In: *European Semantic Web Conference*. Springer. 2015, pp. 36–40.

[26] Siham Eddamiri, Elmoukhtar Zemmouri, and Asmaa Benghabrit. "Theme Identification for RDF Graphs Based on LSTM Neural Reccurent Network". In: *The International Conference on Artificial Intelligence and Computer Vision*. Springer. 2021, pp. 711–720.

[27] Linton C Freeman. "A set of measures of centrality based on betweenness". In: *Sociometry* (1977), pp. 35–41.

[28] Michael R Garey and David S. Johnson. "The rectilinear Steiner tree problem is NP-complete". In: *SIAM Journal on Applied Mathematics* 32.4 (1977), pp. 826–834.

[29] Eugene Garfield. "" Science Citation Index"-A New Dimension in Indexing". In: *Science* 144.3619 (1964), pp. 649–654.

[30] Michelle Girvan and Mark EJ Newman. "Community structure in social and biological networks". In: *Proceedings of the national academy of sciences* 99.12 (2002), pp. 7821–7826.

[31] Katerina Gkirtzou, George Papastefanatos, and Theodore Dalamagas. "RDF keyword search based on keywords-to-sparql translation". In: *Proceedings of the First International Workshop on Novel Web Search Interfaces and Systems*. ACM. 2015, pp. 3–5.

[32] François Goasdoué, Paweł Guzewicz, and Ioana Manolescu. "Incremental structural summarization of RDF graphs". In: *EDBT 2019-22nd International Conference on Extending Database Technology*. 2019.

[33] Steve Gregory. "Finding overlapping communities in networks by label propagation". In: *New Journal of Physics* 12.10 (2010), p. 103018.

[34] Lin Guo et al. "XRANK: Ranked keyword search over XML documents". In: *Proceedings of the 2003 ACM SIGMOD international conference on Management of data*. 2003, pp. 16–27.

[35] Paweł Guzewicz and Ioana Manolescu. "Parallel quotient summarization of RDF graphs". In: 2019.

[36] Pawel Guzewicz and Ioana Manolescu. "Quotient RDF Summaries Based on Type Hierarchies". In: *2018 IEEE 34th International Conference on Data Engineering Workshops (ICDEW)*. IEEE. 2018, pp. 66–71.

[37] S Louis Hakimi. "Steiner's problem in graphs and its implications". In: *Networks* 1.2 (1971), pp. 113–133.

[38] Shuo Han et al. "Keyword Search on RDF Graphs—A Query Graph Assembly Approach". In: *arXiv preprint arXiv: 1704 .00205* (2017).

[39] Xianpei Han and Le Sun. "A generative entity-mention model for linking entities with knowledge base". In: *Proceedings of the 49th Annual Meeting of the Association for Computational Linguistics: Human Language Technologies*. 2011, pp. 945–954.

[40] Hao He et al. "BLINKS: ranked keyword searches on graphs". In: *Proceedings of the 2007 ACM SIGMOD international conference on Management of data*. ACM. 2007, pp. 305–316.

[41] Konrad Höffner et al. "Survey on challenges of question answering in the semantic web". In: *Semantic Web* 8.6 (2017), pp. 895–920.

[42] Vagelis Hristidis and Yannis Papakonstantinou. "Discover: Keyword search in relational databases". In: *VLDB'02: Proceedings of the 28th International Conference on Very Large Databases*. Elsevier. 2002, pp. 670–681.

[43] Xuedong Huang et al. "The SPHINX-II speech recognition system: an overview". In: *Computer Speech & Language* 7.2 (1993), pp. 137–148.

[44] Heasoo Hwang, Vagelis Hristidis, and Yannis Papakonstantinou. "Objectrank: a system for authority-based search on databases". In: *Proceedings of the 2006 ACM SIGMOD international conference on Management of data*. 2006, pp. 796–798.

[45] Yenier Torres Izquierdo. "Keyword Search Algorithm over Large RDF Datasets". In: *International Conference on Conceptual Modeling*. Springer. 2019, pp. 230–238.

[46] Mehdi Kargar, Lukasz Golab, and Jaroslaw Szlichta. "Effective keyword search in graphs". In: *arXiv preprint arXiv:1512.06395* (2015).

[47] Raghav Kaushik et al. "Exploiting local similarity for indexing paths in graph-structured data". In: *Proceedings 18th International Conference on Data Engineering*. IEEE. 2002, pp. 129–140.

[48] Brian W Kernighan and Shen Lin. "An efficient heuristic procedure for partitioning graphs". In: *Bell system technical journal* 49.2 (1970), pp. 291–307.

[49] Shahan Khatchadourian and Mariano P Consens. "ExpLOD: Summary-based exploration of interlinking and RDF usage in the linked open data cloud". In: *Extended semantic web conference*. Springer. 2010, pp. 272–287.

[50] Shahan Khatchadourian and Mariano P Consens. "Exploring RDF Usage and Interlinking in the Linked Open Data Cloud using ExpLOD." In: *LDOW*. 2010.

[51] Graham Klyne. "Resource description framework (RDF): Concepts and abstract syntax". In: *http://www. w3. org/TR/2004/REC-rdf-concepts-20040210/* (2004).

[52] Lawrence Kou, George Markowsky, and Leonard Berman. "A fast algorithm for Steiner trees". In: *Acta informatica* 15.2 (1981), pp. 141–145.

[53] Danai Koutra et al. "Summarizing and understanding large graphs". In: *Statistical Analysis and Data Mining: The ASA Data Science Journal* 8.3 (2015), pp. 183–202.

[54] JB Kruksal. "On the shortest spanning subtree of a graph and the traveling salesman problem". In: *Proceedings of the American Mathematical Society* 7.02 (1956).

[55] Joseph B Kruskal. "On the shortest spanning subtree of a graph and the traveling salesman problem". In: *Proceedings of the American Mathematical society* 7.1 (1956), pp. 48–50.

[56] K Ashwin Kumar and Petros Efstathopoulos. "Utility-driven graph summarization". In: *Proceedings of the VLDB Endowment* 12.4 (2018), pp. 335–347.

[57] Wangchao Le et al. "Scalable keyword search on large RDF data". In: *IEEE Transactions on knowledge and data engineering* 26.11 (2014), pp. 2774–2788.

[58] Yike Liu et al. "Graph summarization methods and applications: A survey". In: *ACM Computing Surveys (CSUR)* 51.3 (2018), pp. 1–34.

[59] Claudio Lucchese, Salvatore Orlando, and Raffaele Perego. "A Unifying Framework for Mining Approximate Top-k Binary Patterns". In: *IEEE Transactions on Knowledge and Data Engineering* 26.12 (2013), pp. 2900–2913.

[60] Tomas Mikolov et al. "Efficient estimation of word representations in vector space". In: *arXiv preprint arXiv:1301.3781* (2013).

[61] Jindřich Mynarz et al. "Generating examples of paths summarizing RDF datasets". In: *In praci ucastniku vedecke konference doktorskeho studia Fakulta informatiky a statistiky.* 2016, p. 34.

[62] Ndapandula Nakashole, Gerhard Weikum, and Fabian Suchanek. "PATTY: A taxonomy of relational patterns with semantic types". In: *Proceedings of the 2012 Joint Conference on Empirical Methods in Natural Language Processing and Computational Natural Language Learning.* Association for Computational Linguistics. 2012, pp. 1135–1145.

[63] Hanane Ouksili. "Exploration et interrogation de données RDF intégrant de la connaissance métier". PhD thesis. Paris Saclay, 2016.

[64] Hanane Ouksili, Zoubida Kedad, and Stéphane Lopes. "Theme identification in RDF graphs". In: *International Conference on Model and Data Engineering.* Springer. 2014, pp. 321–329.

[65] Hanane Ouksili, Zoubida Kedad, and Stéphane Lopes. "Theme Identification in RDF Graphs". In: *Model and Data Engineering - 4th International Conference, MEDI 2014, Larnaca, Cyprus, September 24-26, 2014. Proceedings.* Ed. by Yamine Aït Ameur, Ladjel Bellatreche, and George A. Papadopoulos. Vol. 8748. Lecture Notes in Computer Science. Springer, 2014, pp. 321–329. DOI: `10.1007/978-3-319-11587-0\_30`. URL: `https://doi.org/10.1007/978-3-319-11587-0\_30`.

[66] Hanane Ouksili et al. "Using Patterns for Keyword Search in RDF Graphs." In: *EDBT/ICDT Workshops.* 2017.

[67] Gergely Palla et al. "Uncovering the overlapping community structure of complex networks in nature and society". In: *nature* 435.7043 (2005), p. 814.

[68] Alexandros Pappas et al. "Exploring importance measures for summarizing RDF/S KBs". In: *European Semantic Web Conference.* Springer. 2017, pp. 387–403.

[69] Carlos Eduardo Pires et al. "Summarizing ontology-based schemas in PDMS". In: *2010 IEEE 26th International Conference on Data Engineering Workshops (ICDEW 2010).* IEEE. 2010, pp. 239–244.

[70] Joël Plisson, Nada Lavrac, Dunja Mladenic, et al. "A rule based approach to word lemmatization". In: *Proceedings of IS.* Vol. 3. 2004, pp. 83–86.

[71] Alex Pothen, Horst D Simon, and Kang-Pu Liou. "Partitioning sparse matrices with eigenvectors of graphs". In: *SIAM journal on matrix analysis and applications* 11.3 (1990), pp. 430–452.

[72] Eric Prud'hommeaux. "SPARQL query language for RDF, W3C recommendation". In: *http://www. w3. org/TR/rdf-sparql-query/* (2008).

[73] Paulo Orlando Queiroz-Sousa, Ana Carolina Salgado, and Carlos Eduardo Pires. "A method for building personalized ontology summaries". In: *Journal of Information and Data Management* 4.3 (2013), p. 236.

[74] Usha Nandini Raghavan, Réka Albert, and Soundar Kumara. "Near linear time algorithm to detect community structures in large-scale networks". In: *Physical review E* 76.3 (2007), p. 036106.

[75] Mohamad Rihany, Zoubida Kedad, and Stéphane Lopes. "Keyword Search Over RDF Graphs Using WordNet." In: *BDCSIntell.* 2018, pp. 75–82.

[76] Mohamad Rihany, Zoubida Kedad, and Stéphane Lopes. "Theme-Based Summarization for RDF Datasets". In: *International Conference on Database and Expert Systems Applications.* Springer. 2020, pp. 312–321.

[77] Jorma Rissanen. "Modeling by shortest data description". In: *Automatica* 14.5 (1978), pp. 465–471.

[78] Sunita Sarawagi. *Information extraction.* Now Publishers Inc, 2008.

[79] Evren Sirin and Bijan Parsia. "SPARQL-DL: SPARQL Query for OWL-DL." In: *OWLED.* Vol. 258. Citeseer. 2007.

[80] Qi Song et al. "Mining summaries for knowledge graph search". In: *IEEE Transactions on Knowledge and Data Engineering* 30.10 (2018), pp. 1887–1900.

[81] Georgia Troullinou et al. "Exploring RDFS kbs using summaries". In: *International Semantic Web Conference.* Springer. 2018, pp. 268–284.

[82] Georgia Troullinou et al. "Ontology understanding without tears: The summarization approach". In: *Semantic Web* 8.6 (2017), pp. 797–815.

[83] Georgia Troullinou et al. "RDF Digest: Ontology Exploration using Summaries." In: *International Semantic Web Conference (Posters & Demos).* 2015.

[84] Georgia Troullinou et al. "RDFDigest+: A Summary-driven System for KBs Exploration." In: *International Semantic Web Conference (P&D/ Industry/BlueSky).* 2018.

[85] Yannis Tzitzikas, Nikos Manolis, and Panagiotis Papadakos. "Faceted exploration of RDF/S datasets: a survey". In: *Journal of Intelligent Information Systems* 48.2 (2017), pp. 329–364.

[86] W3C. *SPARQL 1.1 Query Language.* `https://www.w3.org/TR/sparql11-query/`. Online; accessed 30 May 2020. 2013.

[87] W3C. *w3c semantic web activity homepage.* `https://www.w3.org/2001/sw/`. Online; accessed 30 May 2020. 2013.

[88] W3C. *Web Ontology Language (OWL).* `https://www.w3.org/OWL/`. Online; accessed 30 May 2020. 2012.

[89] W3.org. *Property Paths Definition.* URL: `https://www.w3.org/2009/sparql/docs/property-paths/Overview.xml`. (accessed: 2009).

[90] Haixun Wang and Charu C Aggarwal. "A survey of algorithms for keyword search on graph data". In: *Managing and Mining Graph Data.* Springer, 2010, pp. 249–273.

[91] Haofen Wang et al. "Q2semantic: A lightweight keyword interface to semantic search". In: *The Semantic Web: Research and Applications* (2008), pp. 584–598.

[92] Yu Xu and Yannis Papakonstantinou. "Efficient keyword search for smallest LCAs in XML databases". In: *Proceedings of the 2005 ACM SIGMOD international conference on Management of data.* 2005, pp. 527–538.

[93] Qi Zhou et al. "SPARK: adapting keyword query to semantic search". In: *The Semantic Web.* Springer, 2007, pp. 694–707.

[94] Mussab Zneika et al. "RDF graph summarization based on approximate patterns". In: *International Workshop on Information Search, Integration, and Personalization.* Springer. 2015, pp. 69–87.

[95] Lei Zou et al. "Natural language question answering over RDF: a graph data driven approach". In: *Proceedings of the 2014 ACM SIGMOD international conference on Management of data*. ACM. 2014, pp. 313–324.