Large-scale data management in real-world graphs

Imen Ben Dhia

ParisTech
INSTITUT DES SCIENCES ET TECHNOLOGIES
PARIS INSTITUTE OF TECHNOLOGY

TELECOM ParisTech

2013-ENST-0087

EDITE - ED 130

Doctorat ParisTech

T H È S E

pour obtenir le grade de docteur délivré par

TELECOM ParisTech

Spécialité « Informatique et Réseaux »

présentée et soutenue publiquement par

Imen BEN DHIA

le 16 Décembre 2013

Gestion des Grandes Masses de Données dans les Graphes Réels

Directeur de thèse : **Talel ABDESSALEM**
Co-encadrement de la thèse : **Mauro SOZIO**

Jury
Mme. Bénédicte LE GRAND, Professeur, Université Paris 1 Panthéon - Sorbonne
Mme. Amélie MARIAN, Associate Professor, Université de Rutgers, New Jersey
M. Dan VODISLAV, Professeur, Université de Cergy Pontoise
M. Cédric DU MOUZA, HDR, Conservatoire national des arts et métiers (CNAM)

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

T H È S E

Gestion des Grandes Masses de Données dans les Graphes Réels

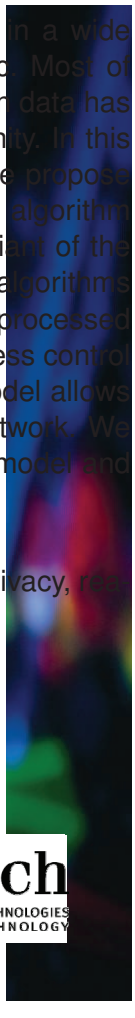
Imen BEN DHIA

RESUME : De nos jours, un grand nombre d'applications utilisent de grands graphes pour la modélisation de données du monde réel. Nous avons assisté, ces dernières années, à une très rapide croissance de ces graphes dans divers contextes ; à savoir, les réseaux sociaux, la bioinformatique, le web sémantique, les systèmes de gestion des données géographiques, etc. La gestion, l'analyse et l'interrogation de ces données constituent un enjeu très important et ont suscité un vaste intérêt dans la communauté des Bases de Données. Lóbjectif de cette thèse est de fournir des algorithmes efficaces pour líndexation et l'interrogation des données dans les grands graphes. Nous avons proposé *EUQLID*, une technique díndexation qui permet de répondre efficacement aux requêtes de calcul de distance dans les grands graphes orientés. L'efficacité de cette technique est dûee au fait qu'elle exploite des propriétés intéressantes des graphes du monde réel. En effet, nous proposons un algorithme basé sur une variante efficace du fameux algorithme 2-hop. Les résultats obtenus montrent que notre algorithme surpassent les approches existantes en terme de temps d'indexation, ainsi qu'en temps de réponse. En effet, il permet de calculer la distance entre deux noeuds en quelques centaines de millisecondes sur de très grands graphes. Nous proposons également un modèle de contrôle d'accès pour les réseaux sociaux qui permet aux utlisateurs de spécifier leurs poltiques de contrôle d'accès en se basant sur leurs relations sociales, et qui peut utiliser *EUQLID* pour passer à léchelle. Nous fournissons une étude de complexité détaillée du protocole de contrôle d'accès et décrivons *Primates* comme étant un prototype appliquant le modèle proposé.

MOTS-CLEFS : Contrôle d'accès, algorithmes dápproximation, requêtes de calcul de distance, indexation de graphes, sécurité, requêtes de test de connexion, échantillonnage, passage à l'échelle, réseaux sociaux.

ABSTRACT : In the last few years, we have been witnessing a rapid growth of networks in a wide range of applications such as social networking, bio-informatics, semantic web, road maps, etc. Most of these networks can be naturally modeled as large graphs. Managing, analyzing, and querying such data has become a very important issue, and, has inspired extensive interest within the database community. In this thesis, we address the problem of efficiently answering distance queries in very large graphs. We propose *EUQLID*, an efficient algorithm to answer distance queries on very large directed graphs. This algorithm exploits some interesting properties that real-world graphs exhibit. It is based on an efficient variant of the seminal 2-hop algorithm. We conducted an extensive set of experiments against state-of-the-art algorithms which show that our approach outperforms existing approaches and that distance queries can be processed within hundreds of milliseconds on very large real-world directed graphs. We also propose an access control model for social networks which can make use of *EUQLID* to scale on very large graphs. This model allows users to specify fine-grained privacy policies based on their relations with other users in the network. We describe and demonstrate *Primates* as a prototype which enforces the proposed access control model and allows users to specify their privacy preferences via a graphical user-friendly interface.

KEY-WORDS : Access control, approximation algorithms, distance queries, graph indexing, privacy, reachability queries, sampling, scalability, social networks.



TELECOM ParisTech

EDITE

ParisTech
INSTITUT DES SCIENCES ET TECHNOLOGIES
PARIS INSTITUTE OF TECHNOLOGY

With the exception of Appendix A which is a translation in french of Chapters 1 and 6 this thesis is written in English.

À l'exception de l'annexe A, qui propose une traduction en français des chapitres 1 et 6, cette thèse est rédigée en anglais.

This thesis is written with the help of the typesetting system LaTeX. The comic strips introducing each chapter were taken from `www.phdcomics.com`. They are reprinted here under fair use.

Cette thèse est rédigée à l'aide du système de composition de documents LaTeX. Les bandes dessinées introduisant chaque chapitre ont été prises de `www.phdcomics.com`. Elles sont reproduites ici en vertu du droit de citation.

"Life is all about timing... the unreachable becomes reachable, the unavailable become available, the unattainable... attainable. Have the patience, wait it out It's all about timing."

Stacey Charter

Dedications

This thesis is dedicated to:

My mother for her constant, unconditional love and support.

The memory of my beloved father who would have been happy to see me following these steps. May his memory forever be a comfort and a blessing.

My fiance and my future family Karim, in grateful thanks for his love, care and support.

Acknowledgments

This dissertation could not have been completed without the support and encouragement of many people. First, I would like to thank my advisor Pr. Talel ABESSALEM for offering me the opportunity to do a PhD and be part of the DBWeb team. I want to thank him for his trust, guidance, advice, understanding, and, also for the trips that I made during the past three years while offering me a lot of freedom and the opportunity to manage a number of responsibilities.

My special thanks goes to my co-advisor Dr. Mauro SOZIO, I have learned a great deal about research, academic writing and presentation skills from him. He always encourages me to think independently and argue with him about research ideas. He provided me with many interesting and relevant ideas and references. Working with him helped me to develop good reflexes that a good researcher should have and to ask the right questions. I want to thank him for his help, support, and, the time that we spent working together despite his busy schedule. I am very lucky to have worked with him for the past two years.

I would also like to thank my dissertation committee: Pr. Bénédicte LE GRAND, Dr. Amélie MARIAN, Pr. Dan VODISLAV, and, Dr. Cédric DU MOUZA for accepting to evaluate my work, and, for their time and effort to help improve and refine my thesis.

I would like to thank all the former and current permanents of DBWeb team, for whom I due a lot of respect for their professionalism and their passion for their work. In particular, Pr. Pierre SENELLART for his willingness to help whenever I solicited him, for reviewing my very first research works, and, for his valuable feedback.

Spending three years in graduate school could very well have been unbearable without many colleagues and friends to make life fun. I would like to thank all the research slaves who made my time in Télécom so enjoyable: Asma, Nora, Faheem, Serafeim, Silviu, Lamine, Hind, Oana, Roxana, Ines, and the list is still so long. Thank You for your help and for being my lunch buddies. I would also like to thank the many other friends who have supported me throughout my graduate school life. Especially, I would like to thank Zeineb for being a very close friend and my best listener, and, also for the several wonderful home-made *moelleux au chocolat* that she made for me when I was writing my thesis.

Finally, my thanks goes to my family, who have always been extremely supportive

of my study. In particular, I would like to thank my sister, for her help and support. I consider myself very lucky to have such a wonderful sister like her. I would also like to thank my cousin Sana and her husband Lyess for their help especially during my first year in Paris. I would like to thank Dali and Ons for being there for me whenever I needed them.

Contents

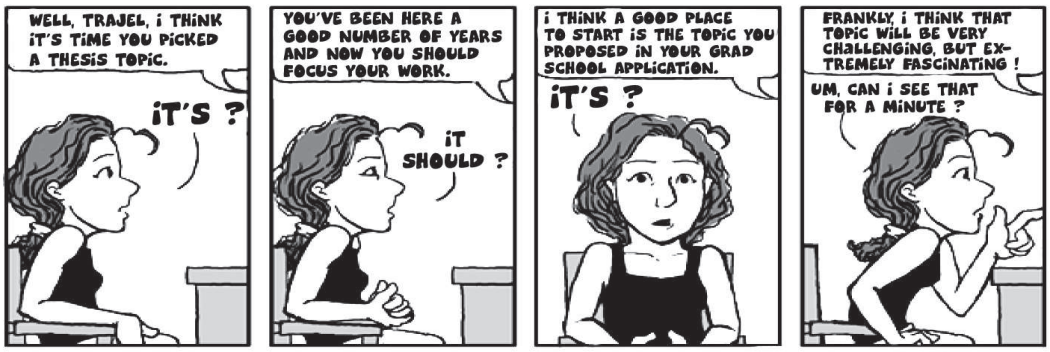
List of Algorithms

List of Figures

List of Tables

Chapter 1.

Introduction



Graphs are ubiquitous in computer science. They are widely used for modeling many real-world data in a large number of applications such as social networks, bio-informatics, the Internet, road networks, airline connections, to name but a few. Nodes usually represent real-world objects and edges denote relationships between them, e.g., users and their relationships in social networks, proteins and their interactions in biological networks, cities and their connections in city road networks. Nowadays, graphs are becoming large, and, are growing rapidly in size. For instance, the social networking site Facebook contains a large network of registered users and their friendships. The number of Facebook users has grown from almost 50 million in September 2007 to more than 1.15 billion monthly active users in 2013 [facb, faca].

Querying graphs has become a very important task in many applications. For example, in a social network, one would like to determine his relationship closeness to a given user (i.e., number hops between him and this user). Additionally, the newly introduced Facebook Graph Search feature (for more details about this feature, see Section 1.1.2), which basically performs a search of the Facebook graph to answer a user input natural language query, could also be considered as a form of querying graphs. For instance, some of the query examples (that could be specified by users) are the following: "Bars which have been visited by my friends who live in Paris, France" or "Photos of my friends taken in Hawaii". In a road network, one would like to know about the distance between two cities. In bio-informatics, people often need to compute the number of interactions needed to transform a given protein into another one. One solution could be to represent the input graph in a relational database, then query it using SQL queries. Unfortunately, this solution is often non-efficient (see Section 2.5). Several alternatives have been proposed in recent years which are often referred to as NoSQL (e.g. neo4j [neo]), however, despite all the efforts made a viable and satisfactory solution is still missing (see Section 2.5). The

issue of querying graphs is apparent even in large software companies such as Facebook, with Graph Search being often unable to deliver exact results [grab]. Therefore, there is an urgent need for developing powerful and scalable graph querying engines that allow effective and efficient processing of queries.

Given that there is not yet a viable and practical solution to deal with large arbitrary graphs, we devise an efficient graph querying system which exploits the properties that modern real networks exhibit. In particular, we focus on answering distance queries on such graphs, and, we consider one application of it, which is enforcing privacy constraints in online social networks. We propose two variants of our approach allowing both main memory and disk-based index storage and querying. Our extensive experimental study on real-world graphs shows that we are able to run distance queries in several hundreds of milliseconds over very large publicly available datasets. To deal with the latter problem, we also propose an access control model for social networks, which is the application domain that, in our case, raised the need to work on scaling reachability queries in large graphs.

This chapter is organized as follows. In Section 1.1, we define the problems to address in this thesis and give some motivating scenarios for them. Section 1.2 highlights our contributions, and, Section 1.3 describes the structure of the dissertation.

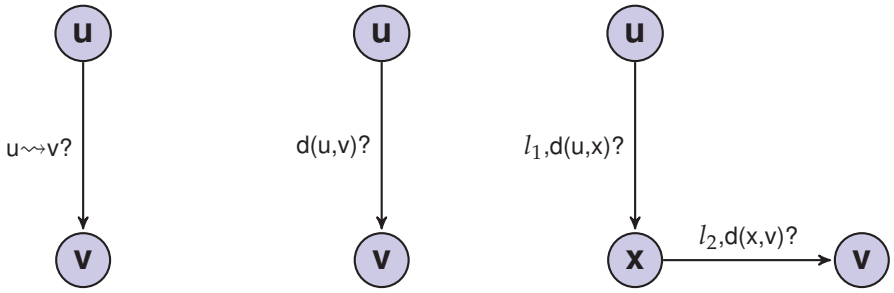
1.1. Context and Problem Definition

In this section, we discuss the general reachability problem, then we present and describe the problem of privacy in social networks as an application domain of it.

1.1.1. Scaling reachability and distance queries on large graphs

A reachability query basically seeks to answer the following simple question: *can u reach v in a graph G?*, where u and v are two nodes given in input. Depending on the application scenario, we distinguish three main types of reachability queries: (i) *Simple reachability queries* asking whether two nodes are connected in the graph without any constraint on the path connecting them. An example of a simple reachability query is depicted in Figure 1.1(a) where $u \rightsquigarrow v$ denotes that u can reach v. (ii) *Distance queries* are more specific than simple reachability queries; it does not only ask whether two nodes are reachable, but it also asks for the distance of the path linking them. An example of a distance query is depicted in Figure 1.1(b), where $d(u, v)$ denotes the distance of the shortest path between u and v. Note that there are two types of distance queries: point to Point (p2p) queries , and, single source shortest path (sssp) queries. The former type returns the distance of the shortest path between two nodes, while the latter type returns the distance of shortest path between a given source node and the rest of the nodes in the graph. And, (iii) *distance and reachability queries with constraints* are even more specific and consider constraints on edge labels, label order, distance, edge direction, etc. Figure 1.1(c) shows an example of such a query, which seeks to find out whether there is a path between u and v with $d(u, x)$ edges labeled l_1 followed by $d(x, v)$ edges labeled l_2.

Here, we highlight several applications domains where reachability queries are crucial:
Social Networks. Social networks are commonly modeled as graphs, where nodes and edges, respectively, denote users and connections between them. Edge labels denote



(a) Simple reachability (b) Distance query (c) Label-constraint reachability

Figure 1.1.: Different types of reachability queries

relation types (i.e., friend, colleague, etc.) between users. Many queries in social networks seek to discover how one node u relates to another node v. These queries in general ask if there is a path from u to v where the labels of all edges in the path are either of a specific type or follow a predefined sequence of labels. For instance, if we want to know whether u is a colleague of a friend of v, we ask if there is a path from u to v with an edge labeled *friend* followed by an edge labeled *colleague*. And, as described in Section 1.1.2, finding such information may help social network users to have more control on their shared information by allowing the desired audience to access it [AD11, DAS12]. Facebook graph search [fbG] (see Section 1.1.2) can also be considered as an application of distance queries in social networks as, for instance, users may search for their friends of friends and their friends for job opportunity purposes (i.e., friends that are 2 and 3 hops away from a given user). Moreover, in order to measure closeness between the corresponding users in a social network, we need to compute the distance between them, e.g., when one asks "How are Germany's chancellor Angela Merkel, the mathematician Richard Courant, Turing-Award winner Jim Gray, and the Dalai Lama related?", saying that all four have a doctoral degree from a German university requires to compute reachability between each one of them and the degrees obtained by the others [KRS+09]. Search ranking in social networks [VFD+07], which is modeled as a function that depends on the distances between users in the friendship graph is an additional application where distance queries are needed.

Bio-informatics. Biological data, such as protein-protein interaction networks and metabolic networks (chemical reactions of the metabolism), can be modeled as labeled graphs, where vertices denote cellular entities (e.g. proteins, genes, etc.). An edge between two entities denotes a chemical interaction that transforms an entity into another one. Edge labels record enzymes that are responsible for entity transformation. One of the basic questions is whether an entity can be transformed into another one under some constraints. These constraints are described as the availability of a set of enzymes in a given order within a given distance between the two entities. More generally, a reachability query in bio-informatics can be formulated as follows: *Is there a pathway between two cellular entities consisting of a given number of interactions with the presence of an ordered set of labels (enzymes)?* Here again, our problem can be described as a constraint reachability query with constraints on label order and distance of the edges along the path. For instance, biological scientists need to compute shortest paths in Protein-Protein Interaction Networks to identify which genes are related to a colorectal cancer [LHL+12]. Reachability queries are also needed to compute k-hop reachability to compare bio-molecular networks and

study their properties [ADH+08].

Semantic Web. The RDF data format, designed to represent data for the Semantic Web, has become a common format for building large data collections. Finding the shortest path between two nodes in an RDF graph is a fundamental operation that allows to discover complex relationships between entities. Despite its expressive capabilities, the standard RDF query language SPARQL does not support the discovery of complex relations between RDF objects (i.e., checking whether a given object X is reachable from object Y in the RDF graph). For instance, finding all scientists who were born in France in a knowledge base can be modeled as reachability queries asking whether there is a path between a given scientist and France. In addition to that, distance queries are used for computing closeness between entities in a knowledge base [KRS+09].

There are two basic approaches to answer reachability queries, which are two extremes in terms of index construction. The first extreme is to precompute and store the full transitive closure. This allows answering reachability queries in constant time by a single look-up. However, it requires a quadratic index space, making it practically unfeasible for large graphs. The other extreme consists in performing a depth-first (DFS) or breadth-first (BFS) traversal of the graph (see Section 2.4) starting from node u, until either the target v is reached or it is determined that no such path exists. This approach doesn't require any index, but requires $O(\mid V \mid + \mid E \mid)$ time for each query, which is computationally expensive in large graphs. Existing approaches lie in-between these two extremes. The challenge consists in attaining high online query efficiency with a low off-line index construction cost.

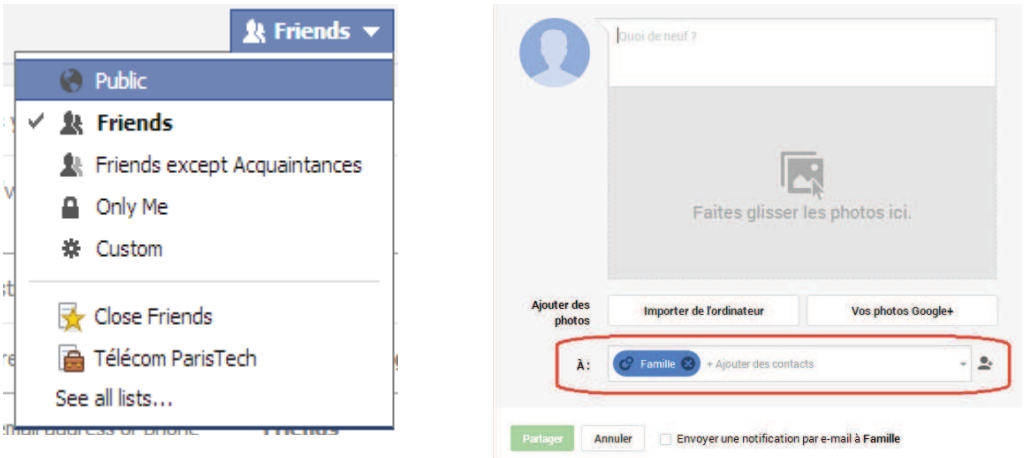
In this thesis, we focus on the problem of answering distance queries in large directed graphs which is a more specific problem than simple reachability on one hand, and, on the other hand, a solution for answering distance queries can be used to handle distance and reachability queries with constraints as explained in Section 6.2.

The classical Dijkstra's algorithm (see Section 2.4.3) fails at efficiently processing distance queries in large graphs. Hence, more sophisticated indexing algorithms were proposed. One of the most successful indexing algorithms for processing distance queries is the seminal 2-hop cover approach [CHKZ02] (see Section 2.6), which comes with guarantees on the size of the index as well as fast query response time. Unfortunately, there is no efficient algorithm for computing a 2-hop cover. While several variants of this approach have been proposed in recent years [STW04, CYL+08, CY09], a viable and satisfactory solution is still missing with the largest graphs being prohibitive to be indexed with this approach. In fact, any indexing algorithm for processing distance queries has to face the inherent problem of computing a compact representation of all distance information in the input graph. Such distance information might increase quadratically with the number of nodes making any index potentially very large in size as well as expensive to compute.

1.1.2. Privacy management in social networks

In an OSN (Online Social Network), each user can easily share information and multimedia content (e.g., personal data, photos, videos, contacts, etc.) with other users in the network, as well as organize different kind of events (e.g., business, entertainment, religion, dating etc.). While this presents unprecedented opportunities, it also gives rise to major privacy issues, as users could often access personal or confidential data of other users. The availability of such information obviously raises privacy and confidentiality

issues. For example, many employers search for their candidates on social networking sites before hiring them [Wor09]. In a tight job market, information that people share (e.g., political views, status updates, funny pictures, etc.) might be a deal breaker. The private information that is available on OSNs could endanger the future employment chances of job candidates, even before being invited for an interview. A survey commissioned by Microsoft discovered that 79% of recruiters and hiring managers in the United States have reviewed online information posted to social networking sites and blogs to screen job candidates, and 70% have rejected an applicant based on information they found [cnn]. Other potential dangers of private information disclosure could be identity theft, sexual harassment, and, stalking to name but a few.



(a) (b)

Figure 1.2.: Privacy Setting Tool of Facebook and Google+

Most social networks provide some basic access control policies, e.g., a user can specify whether a piece of information shall be publicly available, private (no one can see it) or accessible to friends only. For illustration, we describe the privacy management tool of Facebook as an example of one of the most popular OSNs these days, and, among the five top photo-sharing applications on the Internet [top]. As shown in Figure 1.2(a), Facebook allows two extreme privacy policy options: (i) a loose one, by sharing information with everyone in the network (*public* option), and, (ii) a too restrictive by limiting too much information sharing (i.e., *only me* option), which contradicts with the communication and sharing existential purposes of OSNs. Facebook also allows users to create friend lists, and then specify whether a given profile information should be visible or invisible to all friends in a particular list. This somehow forces the user to manually assign friends to lists. However, as the average Facebook user has 144 friends [faca], such task can be very time-consuming and tedious. Even worse, numerous lists might be required as user privacy preferences can be different for different pieces of data. As shown in Figure 1.2(b), Google+ also suffers from the same problem, where lists are denoted as circles.

Moreover, Graph Search [fbG] is a semantic search engine that was introduced by Facebook in March 2013. This new feature allows users to enter natural language queries to find friends, and, friends of friends, who share certain interests. It combines the large amount of data acquired from its over one billion users and external data into a

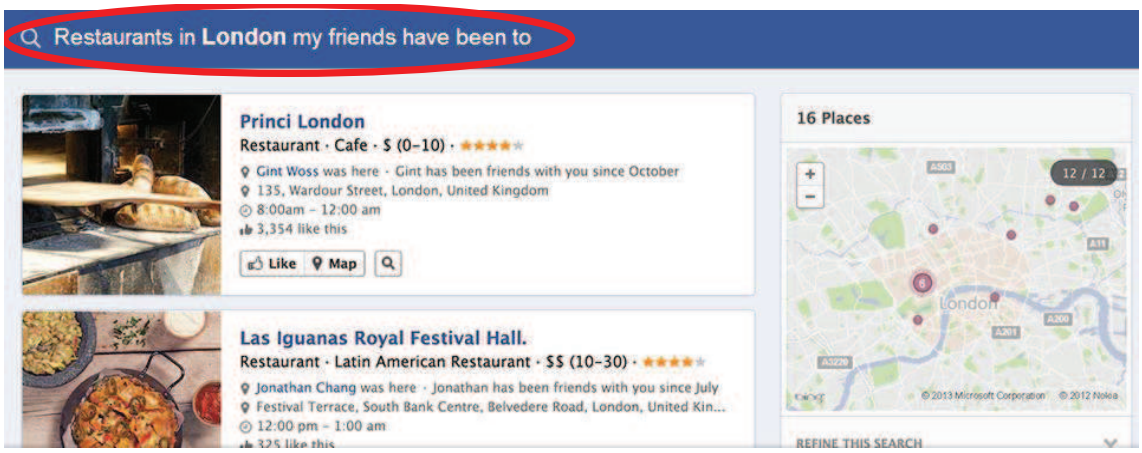


Figure 1.3.: Facebook graph search example

search engine providing user-specific search results. For example, if a user is visiting London and would like to know which restaurants did his friends visit in that city, he can specify a query to get such information as depicted in Figure 1.3. For setting up a potential date, a user could also specify the following query "Single men who live in San Francisco, California and are from Paris, France". Such tool could be used to uncover potentially embarrassing information (e.g., companies employing people who like racism) or illegal interests (e.g., Chinese residents who like the banned group Falun Gong [prib]). In addition to that, it applies the pre-existing privacy settings (i.e., users can access only the information already available to them) [pria], which raises privacy concerns as explained earlier in this section.

Thus, as the set of relationships represented in an OSN nowadays is quite rich and diverse, with relative relationships as well as the possibility of distinguishing between acquaintances and close friends becoming increasingly common, there is an increasing need to providing more sophisticated access control policies. For instance, one would like to say "*invite all children of my colleagues to our child's birthday party*" or "*show this picture of myself wearing a funny costume to my friends and the friends of my friends while not to my colleagues*". For illustration, let us consider the example depicted in Figure 1.4 showing the different relations that Alice has. Let us also consider the following example scenarios:

- **Scenario 1.** Suppose that Alice wants to share an after-work party picture that she took with her colleagues and some of their friends, yet she is reluctant to let the rest of her contacts know about that. She would like to share it only with her colleagues and their friends. The authorized audience would then be Karine, Colin, Julie, and, Bill.

- **Scenario 2.** Suppose that Alice wants to organize a surprise party for her child, and, she would like to share an online invitation that should be visible to the children of her colleagues who live in Paris only. According to the example, the authorized audience involves only Manon.

The previously described scenarios highlight some basic user needs that cannot be specified using the privacy tools provided by existing OSNs. Consequently, we observe

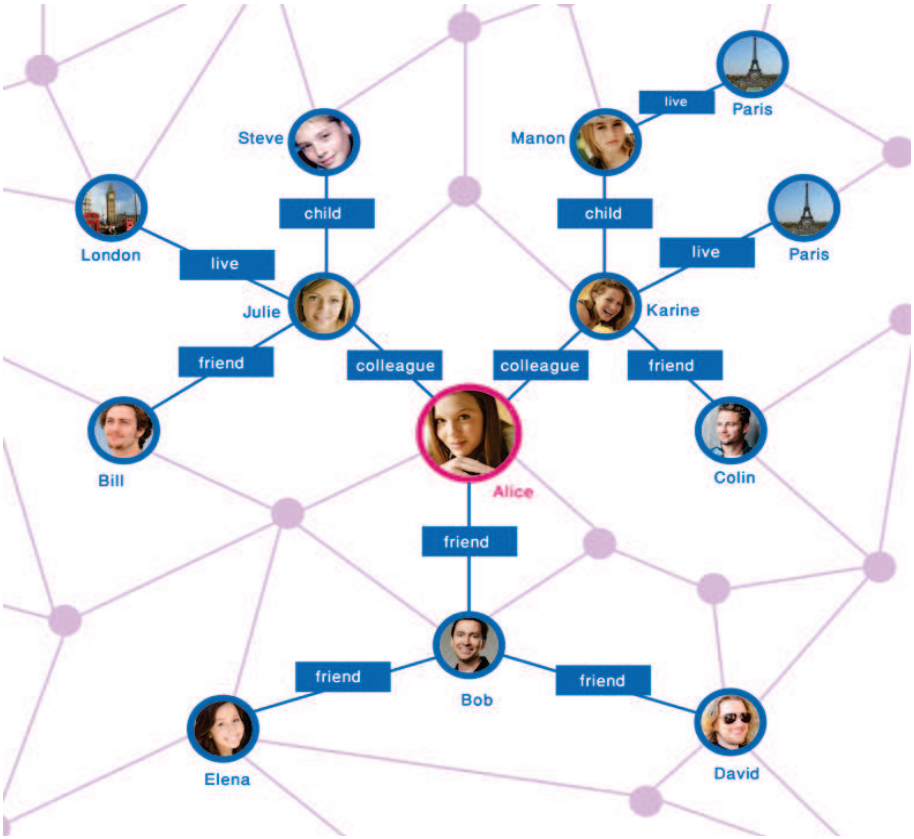


Figure 1.4.: Example of user relations in an OSN

that there is a clear need to devise a new access control model to allow users specify their privacy preferences as they would think about it in real-life scenarios (having a specific audience in mind) and have more control on the spread of their information within OSNs, without the threat of an unwanted hidden audience being possibly able to access it.

Access control can be considered as an application of the general reachability problem, as in order to enforce privacy preferences, we need to efficiently answer reachability queries in large social graphs. For more details about this problem, see Section 1.1.1.

1.2. Contributions

We highlight the contributions of our thesis as follows:

1.2.1. An indexing scheme for efficient distance querying: *EUQLID*

To deal with the problem of efficiently answering distance queries, we proposed *EUQLID* as an efficient indexing scheme. We define a variant of the 2-hop cover where we enforce an additional constraint to limit the size of the index while allowing partial coverage of distance information. The missing distance information can be retrieved efficiently at query time by means of a fast variant of the Dijkstra's algorithm whose search space is carefully pruned. Our goal during the indexing phase is then to cover as much distance

information as possible without violating the size constraint. We show that this variant of the 2-hop cover admits a $0.63-$approximation algorithm which inspired our efficient indexing algorithm.

In order to select which distance information should be stored in our index, we exploit an interesting property that we have discovered after performing some experiments on some real graphs, and, that real-world networks often exhibit. This property says that a few nodes connect all nodes through short paths (see Section 3.4). These nodes are usually the nodes with larges degrees in the network. This property can be observed empirically in our daily routine, as short roads connecting any two cities often go through large cities, long distance flights connect hubs in the corresponding countries, and researchers are connected through prominent scientists in their field in the corresponding co-citation networks. Our idea is to carefully select a set of large degree nodes and to store all distance information corresponding to paths traversing those nodes. As paths traversing large degree nodes are not necessarily shortest paths one needs to employ Dijkstra's algorithm to find exact distances at query time. However, the search space of Dijkstra's algorithm can be effectively pruned by avoiding traversing large degree nodes (as the corresponding distance information is covered by the index) and using the length of the paths traversing large degree nodes as maximum depth in the Dijkstra's algorithm.

We summarize our contributions on answering distance queries as follows:

- We define a new variant of the 2-hop cover where we enforce an additional constraint that limits the size of the index while improving query response time;
- we develop a 0.63-approximation algorithm for this problem;
- our main contribution is EUQLID, which is an efficient algorithm for indexing and processing distance queries on very large graphs. Our algorithm is based on an efficient variant of the approximation algorithm discussed above. Our extensive experimental evaluation against state-of-the-art algorithms shows that our approach outperforms existing approaches and that distance queries can be processed within hundreds of milliseconds on very large real-world directed graphs.

This work was submitted and it is still under review.

1.2.2. A reachability-based access control model for social networks

In order to deal with privacy issues in OSNs (described in Section 1.1.2), we propose a network-aware access control model that allows users to monitor the spread of their personal information. According to our model, users can explicitly specify fine-grained privacy preferences as they think about it in real life scenarios. Our model enables users, having a specific audience in mind, to associate to each piece of information a given audience, which is specified based on the relationship nature between the information owner and his contacts. Following this model, a given user can access an information if and only if a specific path between him and the owner exists. This path expresses constraints on relationship types (e.g., friend, colleague, etc.), edge direction, distance, trust, and, user attributes (e.g., live in Paris, age more than 20, etc.). Deciding whether access should be granted or not is done on the fly based on a the set of access policies assigned to the requested information. We studied the worst case running time of the

access control protocol, and, did experiments to study its performance on real social graphs, which have shown that our model is practical.

This access control model was presented in DBSocial 2011 and the PhD symposium joint to ICDT/EDBT 2012.

1.2.3. A privacy management system: *Primates*

We implemented a privacy management system for OSNs that we called *Primates*. This system is an enforcement of the proposed access control model. It allows users to select some personal information on their profile and assign the desired privacy policies to them. Privacy preference assignment is a user-friendly process which can be done via a graphical interface (i.e., using a user-friendly graphical tool). *Primates* also allows users to preview and visualize the authorized audience as a graph and change their policy accordingly, in case they realize that some unwanted users are part of the audience.

Primates was demonstrated at CIKM 2012.

1.3. Dissertation Organization

The remainder of the dissertation is organized as follows:

- **Chapter 2** provides a collection of concepts and algorithms as background material. Firstly, it introduces boolean expressions as they are used to specify privacy settings, provides some basic graph definitions, and, introduces well-known graph traversal algorithms. Secondly, it introduces the seminal 2-hop labeling scheme, and, describes the *Set Cover* and *Max Cover* algorithms. Then, it explains a fast algorithm for computing the *Set Cover* for large datasets. Finally, it proposes a sampling strategy for estimating the number of elements satisfying a given property in a very large set of elements.

- **Chapter 3** It first introduces two well-known models of complex networks (random and scale-free), and, describes some prominent real-world graph properties. Then, it reports some experiments that were conducted to analyze shortest paths in social networks, and, studies the impact of some discovered properties on improving the 2-hop algorithm described in Chapter 2.

- **Chapter 4** discusses related work about reachability, then defines formally the problem of answering distance queries and introduces notations and definitions. It also describes both the indexing algorithm and the query processing algorithm, then reports an extensive experimental evaluation of our approach against the state of the art.

- **Chapter 5** first discusses existing work on privacy in OSNs and highlights their limitations. Secondly, it formally defines some notions related to social networks, and, highlights the requirements to design appropriate access control models for social networks. Then, it describes the proposed access control model and how it can be enforced. It also studies the worst case running time of the access control protocol, and, shows some experimental results. Finally, it presents *Primates* as an implementation of the proposed access control model.

- **Chapter 6** concludes this thesis by reminding the most important questions that we studied, the proposed solutions, and, the obtained results. It also presents some research perspectives to address other issues related to the ones we studied.

Chapter 2.

Background and Preliminaries

2.1. Introduction

In this chapter, we introduce a collection of concepts and algorithms as background material for the rest of the dissertation. We define and explain the use of boolean expressions in our access control model in Section 2.2. We recall some basic graph definitions and describe graph traversal algorithms in Sections 2.3 and 2.4, respectively. In Section 2.5, we introduce graph database systems, and, describe some experimental studies for comparing these systems to their relational counterparts in terms of storing and traversing graphs. We introduce the 2-hop labeling scheme for computing reachablity and distance queries in Section 2.6. In Sections 2.7 and 2.8, we respectively describe the *Set Cover* and *Max Cover* algorithms, and, give examples of application. In Section 2.10, we explain a fast algorithm for computing the *Set Cover* for large datasets. In Section 2.11, we propose a sampling strategy for estimating the number of elements satisfying a given property in a very large set of elements. Section 2.12 concludes the chapter.

2.2. Boolean Expressions

A Boolean expression is a logical statement that is either `TRUE` or `FALSE`. Boolean expressions can compare data of any type as long as both parts of the expression have the same basic data type. A Boolean expression is a three-part clause that consists of two items to be compared separated by a comparison operator. A more complex Boolean expression can be created by joining any of these three-part expressions with the `AND` ($\wedge$) and `OR` ($\vee$) logical operators.

Boolean expressions could be used to express real-life facts. For instance, the fact that a person X is a scientist who lives in Paris could be expressed as follows:

$$P_1 \wedge P_2 \tag{2.1}$$

where P_1 and P_2 are two predicates ($P_1 = X$ *is a scientist* and $P_1 = X$ *lives in Paris*).

A slightly more complicated example could be the following: *The subway will be crowded when the soccer game finishes, unless our team wins*. These facts can be expressed as follows:

$$(P_3 \rightarrow P_4) \vee P_5 \tag{2.2}$$

where $P_3 =$ *the soccer game finishes*, $P_4 =$ *the subway will be crowded*, and, $P_5 =$ *our team wins*.

We use boolean expressions to formalize access rules (i.e., user specified privacy settings) in our access control model as it is illustrated in Section 5.5.1.

2.3. Basic Graph Definitions

In this section, we introduce some of the basic notations and definitions that are commonly used in graph theory.

Graph. Graphs (i.e., networks) are useful structures in science and mathematics, which help modeling data that is being generated by many real-world applications. More formally, a graph $G(V, E)$ consists of two sets of elements of different type, namely, a set of nodes V and a set of edges E (i.e. a set of ordered pairs (u, v) where u and v are nodes in G). $V(G)$ and $E(G)$, respectively, denote the set of vertices and the set of edges of G, and, $\mid V \mid$ and $\mid E \mid$ denote the number of nodes and edges, respectively. There are several possibilities to represent a graph G in memory (i.e., adjacency matrix, edge list, adjacency list [grac]).

Undirected, directed. An undirected graph is a graph where edges have no orientation (i.e., all edges are symmetric), while edges in a directed graph are directed. An edge $e = (u, v)$ is considered to be directed from u to v; u is called the head and v is called the tail of e; v is a direct successor of u, and u is a direct predecessor of v.

Unweighted, weighted. A graph is weighted if each edge $e \in E$ is assigned a weight $w(e)$. Depending on the problem, weights might represent, for example, costs, lengths or capacities. When no weights are associated to edges, the graph is always assumed to be unweighted.

Path. A directed path $\mathcal{P} = u \rightsquigarrow v$ is defined as a set of nodes $u_0, u_1, \ldots, u_k$ where $u_0 = u$,$u_k = v$ and (u_i, u_{i+1}) are edges of G, $i = 0, \ldots, k-1$. If there is a directed path $u \rightsquigarrow v$ in G, we say that u is a *predecessor* of v and v is a *successor* of u, while denoting with $S(v)$ and $P(v)$ the set of successors and predecessors of v, respectively. The length of a path $\mathcal{P}$ is defined as the sum of the weights of the edges in $\mathcal{P}$. $d(u, v)$ is the distance between u and v in G, that is, the length of the shortest path connecting u and v in G.

Degree. The degree of a given node v denotes the number of edges connected to v. The *in-degree* of a node v is defined as the number of predecessors u of v such that there is an edge (u, v) in G, while the *out-degree* of v is defined as the number of successors u of v such that there is an edge (v, u).

Strongly Connected Component (SCC). A strongly connected component is a maximal subgraph of a directed graph such that for every pair of nodes u and v in the subgraph, there is a directed path from u to v and a directed path from v to u.

Directed Acyclic Graph (DAG). A DAG is a directed graph with no directed cycles. It consists of a set of nodes and directed edges. Each edge connects a node to another in such a way that makes it impossible to start at a node v and follow a sequence of edges that eventually loops back to v again. A directed graph can be transformed into a DAG by collapsing each of its strongly connected components into a single node.

Density. The density of an undirected graph G is defined to be as $|E(G)|/|V(G)|$, which also denotes the average node degree in G.

Reachable, unreachable. A node v is reachable from another node u, denoted $u \rightsquigarrow v$, if there is a path of any length from u to v.

Figure 2.1 shows a directed and weighted graph, which we will use as a running example in the remaining sections of this chapter.

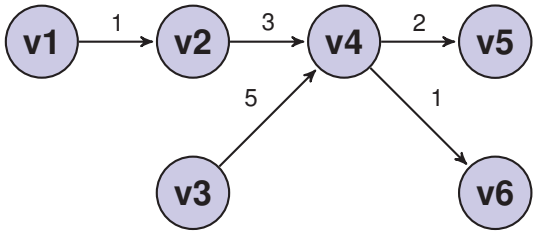


Figure 2.1.: Graph example

2.4. Graph Traversal

In this section, we discuss several graph traversal algorithms which form the basis for some of the proposed graph algorithms in Chapters 4 and 5. Traversal algorithms are differentiated according to the way they traverse graphs. In the following, we describe some of the most well-known ones.

2.4.1. Breadth-first search (BFS)

BFS is a search algorithm which exhaustively searches the entire graph (or a part of it) until it reaches a predefined goal (e.g., reaching a target node, visiting all nodes). According to the BFS algorithm, all the child nodes that are obtained by expanding a node are added to a FIFO (i.e., First In, First Out) queue. Whenever a node is visited during the traversal, it is marked as visited and removed from further exploration. The flow of the BFS algorithm is described in Algorithm 1. It takes as input a source node s and iteratively explores the graph level by level.

Let us consider the graph example depicted in Figure 2.1. Let v_1 be the source node. BFS visits the nodes in the following order: v_1, v_2, v_4, v_5, v_6.

The time and space complexity of BFS is $O(\mid V \mid + \mid E \mid)$. This algorithm can be used to solve many problems in graph theory, for instance, finding the shortest path from a node

Algorithm 1: Breadth-First Search (BFS)

Input: A graph $G(V, E)$ and a source node s.

```
1 Initialize a FIFO queue Q to s and mark s as visited;
2 Enqueue s in Q;
3 Mark all nodes in G as unvisited except s;
4 while Q ≠ ∅ do
5 |   Dequeue a node v from Q;
6 |   foreach immediate successor x of v do
7 |   |   if x is not marked as visited then
8 |   |   |   Enqueue x in Q and mark it as visited;
```

to another, exploring all the reachable nodes starting from a given node, finding all nodes within one connected component, testing a graph for bipartiteness, etc.

Bi-directional search is a variant of BFS, which runs two simultaneous BFS searches: one forward from a source node, and one backward from a target node, stopping when the two meet in the middle. While this approach can be faster in some cases, it can also perform worse than simple BFS when there are many high-degree nodes on the path(s) between the source and target nodes.

2.4.2. Depth-first search (DFS)

Just like BFS, DFS is also an algorithm for traversing graphs. The only difference is that, it traverses the depth of any particular sub-graph before exploring its breadth. That is, child nodes are traversed before visiting sibling nodes. A stack can be used to implement the algorithm.

As described in Algorithm 2, it takes a source node s as input. Then, it iteratively goes from the current node to an adjacent unvisited one until it can no longer find unexplored nodes. The algorithm then backtracks along previously visited nodes, until it finds a node connected to not yet visited nodes. It will then proceed down the new path as it had before, backtracking as it encounters dead-ends and ending only when the algorithm has backtracked the input source node s.

Let us consider again the graph example depicted in Figure 2.1 and v_1 as the input source node. DFS explores the graph in the following order: v_1, v_2, v_4, v_5, v_6.

The time and space complexity of DFS is $O(\mid V \mid + \mid E \mid)$. This algorithm is the basis for many graph algorithms, including topological sorts, planarity and reachability testing, etc.

2.4.3. Dijkstra's algorithm

Dijkstra's Algorithm is a graph search algorithm for the single-source shortest path problem. It takes in input a source node s and outputs the nodes in increasing order of their distance from s. It can be applied on graphs with non-negative edge weights.

For a given source vertex s, the algorithm finds the shortest path between s and every other node in the graph. It can also be used for finding costs of shortest paths from a single vertex to a single destination vertex by stopping the algorithm once the shortest

Algorithm 2: Depth-First Search (DFS)

Input: A graph $G(V, E)$ and a source node s.

```
1 Initialize a stack S to s and mark s as visited;
2 Push s to S Mark all nodes in G as visited except s;
4 while S ≠ ∅ do
5 |   Pop a node v from S;
6 |   foreach immediate successor x of v do
7 |   |   if x is not marked as visited then
8 |   |   |   Push x to S and mark it as visited;
```

path to the destination vertex has been determined. For example, if the nodes of the graph represent cities, and, edge weights represent driving distances between pairs of cities connected by a direct road, Dijkstra's algorithm can be used to find the shortest route between one city and all other cities.

As described in Algorithm 3, Dijkstra's algorithm recursively traverses the graph starting from s while maintaining the length of the shortest path so far found between s and the set of traversed nodes. Its worst case running time is $O(\mid V \mid^2)$. However, the min-priority queue-based implementation runs in $O(\mid E \mid + \mid V \mid \log \mid V \mid)$ [FT87].

Algorithm 3: Dijkstra's algorithm

Input : A graph $G(V, E)$ and a source node s.
Output: Shortest path distances between s and all the nodes in G

1 Initialize a priority PQ to $(s, d(s))$ (where $d(s) = 0$);
2 Associate the value ∞ to the rest of the nodes ($d(v) = \infty$);
3 Repeatedly choose an unexplored node $v \in PQ$ which minimizes

$$\pi(v) = \min_{e=(u,v):u \in S} d(u) + \omega(e)$$

4 add v to PQ and set $d(v)$ to $\pi(v)$;

The output of Dijkstra's algorithm when the graph depicted in Figure 2.1 and node v_1 are given in input is the following: $(v_1, 0)$, $(v_2, 1)$, (v_3, ∞), $(v_4, 4)$, $(v_5, 6)$, and, $(v_6, 5)$.

We used Dijkstra's algorithm as a benchmark to compare with in Chapter 4.

2.4.4. Topological sort

The Topological sorting problem consists in finding a linear ordering of nodes in an input directed graph G such that:

$$\forall (u, v) \in E, \tau(u) < \tau(v) \tag{2.3}$$

where $\tau(v)$ denotes the topological order of a given node v. For instance, if the nodes of the graph represent tasks to be performed and the edges represent constraints that one task must be performed before another, a topological ordering is a valid sequence for the

tasks. Note that, a topological ordering is possible if and only if the graph has no directed cycles (i.e., if it is a DAG).

As described in Algorithm 4, the topological sort algorithm first identifies nodes with no incoming edges (the in-degree of these nodes is equal to 0). Then, it iteratively picks such a node, incrementally assign it with its topological order and delete it along with all its outgoing edges from the graph. The algorithm stops when all nodes in the graph are assigned a topological order.

Algorithm 4: Topological Sort Algorithm

Input : A DAG $G(V, E)$.
Output: Topological order of all nodes in G

```
1 Initialize tp_counter to 0;
2 roots ← nodes with no incoming edges in G;
3 while roots ≠ ∅ do
4 |   Dequeue a node v from roots;
5 |   Delete v and all its outgoing edges from G;
6 |   tp(v) ← tp_counter++;
```

As an example, the topological order of the nodes in the graph depicted in Figure 2.1 is the following: $\tau(v_1) = 0$, $\tau(v_2) = 2$, $\tau(v_3) = 1$, $\tau(v_4) = 3$, $\tau(v_5) = 4$, and, $\tau(v_6) = 5$. This algorithm runs in polynomial time. We used it to compute the topological order of nodes in our graph datasets to further improve the performance of the query algorithm described in Section 4.4.2.

2.5. Graph Database Systems

One of the main purposes behind storing data using a database management system is the possibility to efficiently query this data. Several different database systems (e.g., relational and graph databases) are available for use, but, each variant has its own pros and cons depending on the structure of the data and the way it should be queried. Our goal, in this section, is to compare the performance and scalability of relational and graph database systems in order to choose the most suitable datastore for our graph datasets.

Before describing the settings and reporting some experimental results, let us quickly define what is a graph database.

Definition 2.1. Graph Databases. A graph database is a type of NoSQL* database. It basically stores data as a collection of nodes and edges. Each node represents an entity and each edge represents a connection or relationship between two nodes. Every node in a graph database is defined by a unique identifier, a set of outgoing edges and/or incoming edges and a set of properties expressed as key/value pairs. Each edge is defined by a unique identifier, a source node and/or ending node, and, a set of properties.

Figure 2.2 depicts an example of a graph database.

*an approach to data management and database design, which is useful for very large sets of distributed data. NoSQL systems doesn't not follow a relational structure and allow both unstructured and structured (SQL) query languages, e.g., column stores and graph databases.

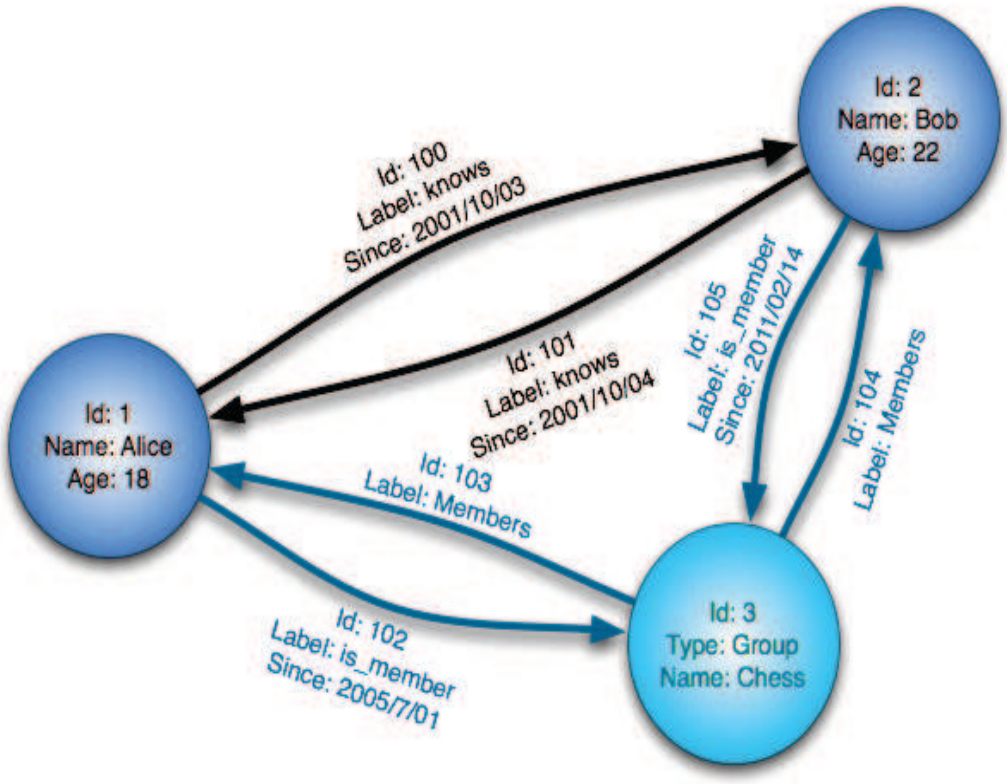


Figure 2.2.: A Graph Database example [Graa]

We performed our experiments on MySQL as a relational database, and, Neo4j as a graph database (see Section 5.7 for more details about Neo4j). Both implementations were tested using a sample dataset from the Facebook social network. We considered a sample of 984K unique users that represents the *groundtruth* of the Facebook OSN, i.e., a truly uniform sample of Facebook anonymized user IDs crawled by Gjoka et al. [GKBM10]. It consists of $\sim$ 1 million nodes and $\sim$ 9 million edges. For fair comparison, we created the relational and graph versions the dataset. The relational version was stored in table `graph(user1, user2)`, where each row stores an edge going from one user/node to another. We created an index on the first column `user1`. The traversal was evaluated on each database starting from 200 random root nodes.

The results are presented in Table 2.1, which reports the average running time for both the relational and graph systems (MySQL and Neo4j, respectively). Note that only Neo4j has the running time for a traversal of length 5. MySQL did not finish after waiting 1 hour to complete. In comparison, Neo4j took $\sim$ 1.2 seconds to complete a 5-hop traversal, which is quite fast. It is clear from the results that Neo4j is more suitable for graph traversal than its relational counterpart. However, this is not really very good news, as the size of the considered graph is not that large, and, the running time might be far longer over real graphs (which are much larger).

Our results were confirmed in [VMZ+10], as according to their comparison between Neo4j and MySQL, Neo4j was performing better on graph traversals. The largest graph

that they considered consisted of 100K nodes. They performed 4-hop and 128-hop queries on MySQL and Neo4j versions of the same graph. However, when it comes to executing queries on node attribute values or making statistics (such as node degree distribution, etc.), MySQL performs better.

#hops	MySQL traversal time (ms)	Neo4j traversal time (ms)
1	124	312
2	922	512
3	8 851	758
4	112 930	956
5	-	1243

Table 2.1.: MySQL and Neo4j traversal performance

As far as we are aware, no comparison study between NoSQL and relational databases on very large graphs ($>$ 1 million nodes) was published. To evaluate the performance of MySQL over large datasets, we created the relational version of a twitter dataset (with 40 million nodes and 1.4 billion edges) with an index on the first column. We performed a 2-hop query (which retrieves and counts all nodes at distance 2 from a given node) for a randomly chosen sample of 10 root nodes. The average running time was $\sim$ 20 minutes, which is already very high and inefficient. Note that these experiments were not possible with Neo4j, as it did not allow us to load and create a graph of such size in order to evaluate the traversal performance.

We can say that Neo4j is promising but it is still premature to deal with very large graphs. For instance, importing and indexing large-scale data may take months and there is no support to full graph scans*. Neo4j traversal performance depends significantly on the number of nodes to traverse. The higher this number is, the worse the traversal performance becomes. To analyze the graph and make statistics over it, it may be better to use other storage system like an RDMS (Relational Database Management System).

2.6. 2-hop Labeling

The 2-hop labeling is an indexing scheme which allows to compute reachability and distance queries. It was first introduced by Cohen et al. in [CHKZ02]. Here is a brief introduction to the 2-hop labeling: Let $G = (V, E)$ be a graph, a 2-hop labeling assigns to each vertex $v \in V$ two sets $L_{in}(v)$ and $L_{out}(v)$ such that for each vertex $x \in L_{in}(v)$ and $y \in L_{out}(v)$, there is a path between x and y which goes through v. A node v is reachable from a node u, denoted $u \rightsquigarrow v$, if and only if $L_{out}(u) \cap L_{in}(v) \neq \varnothing$.

The size of the labeling is defined to be:

$$\sum_{v \in V} |L_{\text{in}}(v)| + |L_{\text{out}}(v)| \tag{2.4}$$

Cohen et al. [CHKZ02] proposed an approximation algorithm to compute the 2-hop cover of a given graph with minimal size, so to obtain a compact version of the transitive

*http://neo4j.org/nabble/#nabble-td3351599

closure. This algorithm allows to compute a 2-hop cover whose size is at most by a factor of $O(log \mid V \mid)$ larger than the optimal size. We illustrate Cohen et al.'s algorithm in Algorithm 5.

Algorithm 5: Two-Hop Cover Algorithm

Input : A graph $G = (V, E)$.
Output: A 2-Hop Labeling H of G

1 $H \leftarrow \varnothing$;
2 $T \leftarrow$ Compute all reachable pairs in G (transitive closure);
▷ $T = \{(u, v) \mid u \rightsquigarrow v\}$
3 $T^{'} \leftarrow T$;
4 **while** $(T^{'} \neq \varnothing)$ **do**
5 **foreach** $v \in V$ **do**
6 $B_v \leftarrow$ Compute the bipartite graph corresponding to v;
▷ $B_v = \{(V, E) \mid V = V_{in} \cup V_{out}\}$
7 $denB_v \leftarrow$ Compute the densest subgraph of B_v;
8 $d(v) \leftarrow$ the density of $denB_v$;
9 $denB_w \leftarrow$ the densest subgraph with the highest density value;
10 $w \leftarrow$ the center node of $denB_w$;
11 **foreach** $u \in V_{in}$ *of* $denB_w$ **do**
$H \leftarrow H \cup (u, (u, w))$
12 **foreach** $u \in V_{out}$ *of* $denB_w$ **do**
$H \leftarrow H \cup (u, (w, u))$
13 Remove from $T^{'}$ the shortest paths covered by w;
▷ $T^{'} \leftarrow T^{'} \backslash E(denB_w)$
14 return H;

Initially, we compute the transitive closure of the input graph G, and, initialize T and T' with the resulting transitive closure (line 2). At each iteration and for each node $v \in V$, we compute the corresponding bipartite graph $B_v = (V, E)$ where $V = V_{in} \cup V_{out}$ and $E = \{(x, y) \mid x \in V_{in} \wedge y \in V_{out} \wedge (x, y)$ is uncovered $\}$ (line 7). V_{in} and V_{out}, respectively, denote the set of predecessors and successors of v. Then, we compute the density value $d(v)$ (line 7), where:

$$d(v) = \mid (V_{in}(v) \times V_{out}(v)) \cap T^{'} \mid / \mid V_{in}(v) + V_{out}(v) \mid$$

$d(w)$ denotes the ratio of the number of connections going through w which are not yet covered by the total number of nodes involved on such connections. The problem of finding the two sets $V_{in}(v)$ and $V_{out}(v)$ with the highest density value boils into finding the densest bipartite graph. This can be done as follows: we iteratively eliminate the node with the minimum degree from $B_v(V, E)$, then compute and store the density of the resulting graph until V is empty. Once densest subgraphs of all the nodes $v \in V$ are computed, Algorithm 5 picks the one with the highest density (denoted $denB_w$) along with its corresponding center (denoted w) as illustrated in lines 9 and 10.

Algorithm 5 updates the 2-Hop Cover H by adding new hops (lines 11 and 12) and T' by eliminating node pairs that correspond to added hops (line 13), and, stops when T' is empty (i.e., all shortest paths are covered in H).

Note that distances $d(x, v)$ and $d(v, y)$ are pre-computed and stored. Given a distance query, u and v, the index ensures that the returned distance $d(u, v)$ is the distance of the shortest path between u and v by computing it as follows:

$$d(u, v) = min_{w_i \in (L_{out}(u) \cap L_{in}(v))} d(v, w_i) + d(w_i, v) \tag{2.5}$$

A graph example and its corresponding 2-hop distance labeling are depicted in Figures 2.1 and 2.2, respectively.

Node v	$L_{out}(v)$	$L_{in}(v)$
v_1	$(v_4, 2), (v_2, 1)$	$\varnothing$
v_2	$(v_4, 1), (v_2, 0)$	$(v_2, 0)$
v_3	$(v_4, 1)$	$\varnothing$
v_4	$(v_4, 0)$	$(v_4, 0)$
v_5	$\varnothing$	$(v_4, 1)$
v_6	$\varnothing$	$(v_4, 1)$

Table 2.2.: 2-hop labeling

2.7. Set Cover

The problem of *Set Cover* aims to find the smallest sub-collection of sets that covers some universe of elements. In the *Set Cover* problem, we are given: (i) a universe U of n elements (i.e., $\mid U \mid = n$), and, (ii) a collection $S = \{S_1, S_2, ..., S_m\}$ of sets of the universe U that may be overlapping (i.e., $S_1, S_2, ..., S_m \subseteq U$). Each subset $S_i \in S$ has an associated cost c_i. A *Set Cover* is a collection of some of the sets in S, whose union includes all the elements in the universe U. More formally, C is a *Set Cover* if $\bigcup_{S_i \in C} S_i = U$. The goal is to find a collection C of sets S_i whose union is equal to U, and, such that $\sum_{S_i \in C} c_i$ is minimized. Let us consider the example shown in Figure 2.3. The example shows an input of $m = 10$ sets S_i over the universe $U = \{A, B, C, D, E, F, G, H, I\}$ of size $n = 9$.

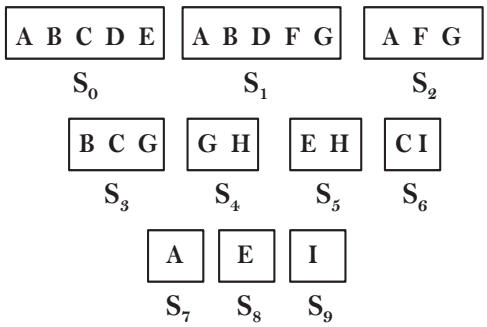


Figure 2.3.: Example of input $S = \{S_0, ..., S_9\}$

Computing the *Set Cover* is an NP-Hard problem. The best-known algorithm for computing the *Set Cover* is based on a greedy heuristic [Joh73]. The greedy algorithm for computing the *Set Cover* of a given universe is sketched in Algorithm 6.

Initially, all the elements in U are marked as uncovered (line 1). Then, at each step, the set having the minimum associated cost value c_i is picked and all its elements are marked as covered. The algorithm terminates when all the elements in U are covered.

Algorithm 6: Greedy Set Cover Algorithm

Input : A set U of n elements.
A collection $S = \{S_1, S_2, ..., S_m\}$ of subsets of U with costs c_i.
Output: A *Set Cover SC*

```
1 Mark all elements in U as uncovered;
2 SC ← ∅;
3 while (Some elements remain uncovered) do
    ▷ U ⊈ C
4 Pick S_i with the minimum cost c_i from S;
5 SC ← SC ∪ S_i;
6 Mark all elements in S_i as covered;
7 Remove S_i from S;
8 return SC;
```

Figure 2.4 shows an example of running the greedy set cover algorithm on the input sets in Figure 2.3. For simplicity, we consider in this example the cost c_i of a set S_i as $\frac{1}{|S_i|}$ where $\mid S_i \mid$ is the cardinality of S_i. Initially, the sets with the largest number of uncovered elements are S_0 and S_1 of size 5. The algorithm picks, then, randomly $S_0 = \{A, B, C, D, E\}$. After this step, all elements in S_0 are covered (these are shown in lowercase in Figure 2.4). Now, the set with the largest number of uncovered pairs is of size 2. The greedy algorithm arbitrarily picks S_1 and marks additional items (F and G) as covered. In the next step, S_4 is picked to cover H. Then, S_6 is picked in order to cover the remaining item I and the algorithm terminates. The resulting set cover is then $\{S_0, S_1, S_4, S_6\}$ which is not the optimal solution as picking the sets S_1, S_5, and, S_6 is sufficient to cover all the items in U. Thus, the greedy algorithm is an approximation algorithm of the *Set Cover*.

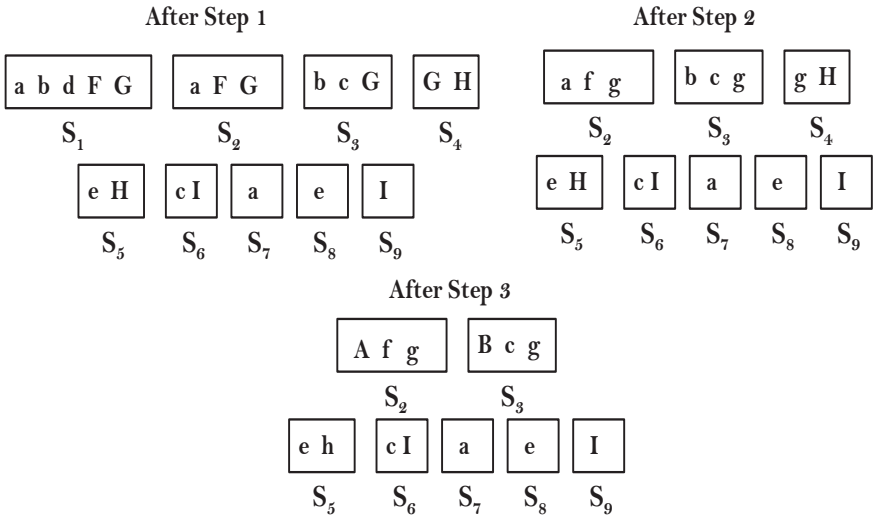


Figure 2.4.: Greedy algorithm execution example

2.8. Max Cover

The maximum coverage problem is a classical question in computer science. Given a universe U of n elements, a collection $S = \{S_1, S_2, ..., S_m\}$ of sets of the universe U, and, an integer value K. For the general weighted max cover problem, the goal is to select k sets from S such that the sum of costs is minimized. For the unweighted variant of the max cover problem, the goal consists in picking k sets from S such that the number of covered elements $| \bigcup_{S_i \in S} | S_i | |$ is maximized.

The maximum coverage problem is NP-Hard. It admits a greedy approximation algorithm (see Algorithm 7), which selects at each step the set with the maximum number of uncovered pairs (like the *Set Cover* greedy algorithm), but, it terminates when k are already picked. The approximation ratio of this algorithm is $(1 - \frac{1}{e})$ [Hoc97b].

Algorithm 7: Greedy Max Cover Algorithm

Input : A set U of n elements.
A collection $S = \{S_1, S_2, ..., S_m\}$ of subsets of U with costs c_i.
An integer value k.
Output: A max cover MC

```
Mark all elements in U as uncovered;
MC ← ∅;
while (number of picked sets < k) do
    Pick S_i with the minimum cost c_i from S;
    MC ← MC ∪ S_i;
    Mark all elements in S_i as covered;
    Remove S_i from S;
return MC;
```

2.9. Submodular Function Maximization

Given a finite set E, a function $f : 2^E \rightarrow \mathrm{R}^+$ is submodular if and only if the following holds:

$$\forall A, B \subseteq E, f(A) + f(B) \geqslant f(A \cup B) + f(A \cap B) \tag{2.6}$$

The problem of maximizing a sub-modular function can be described as follows: Given a finite set E such that $| E | = n$, a function $f : 2^E \rightarrow \mathrm{R}^+$ and an integer $k \leqslant n$, our goal is to pick at most k sets from E such that $\sum_i f(S_i)$ is maximized. f is specified via an oracle, which given a set $A \subseteq E$ will return $f(A)$.

The submodular function maximization generalizes the max cover problem, and, it is an NP-Hard problem. The solution to this problem is based on the greedy approximation algorithm with a $1 - \frac{1}{e} \approx 0.632$ guarantee.

2.10. *disk-friendly* Set Cover

The explained above greedy algorithm (see Algorithm 6) for computing the *Set Cover* typically finds solutions that are close to optimal. However, a direct implementation of the greedy approach does not behave well when the input is very large and/or disk-resident. One of the bottlenecks of the greedy approach is computing the number of uncovered elements for each set S_i in order to pick the set with the largest number of uncovered items at each step. To deal with this issue, Cormode et al. [CKW10] proposed a disk-friendly algorithm to find a *Set Cover* which is close to that of the greedy algorithm, but, which can be computed more efficiently. In this section, we describe the fast set cover algorithm proposed in [CKW10].

As sketched in Algorithm 8, the initial step consists in partitioning the input sets into sub-collections according to their size (i.e., number of items in each set) for the unweighted variant of the *Set Cover*, and, according to the associated cost c_i for the weighted case (line 1). More in detail, a set S_i is assigned to a sub-collection k if and only if the following holds:

$$p^k \leqslant \mid S_i \mid \leqslant p^{k+1} \tag{2.7}$$

where p is a parameter such that $p \in \mathrm{R}$ and $p > 1$, which governs both the approximation ratio of the algorithm and its running time.

Starting with the non-empty sub-collection $S^{(k)}$ with the largest k value. The algorithm proceeds as follows: it picks the first set S_i in $S^{(k)}$ and computes the number of uncovered items in it. If according to Inequality 2.7, S_i still belongs to the current sub-collection $S^{(K)}$, then S_i is added to the *Set Cover SC* and omitted from further consideration. Otherwise, S_i is removed from its current sub-collection $S^{(k)}$ and placed in the right sub-collection with respect to Inequality 2.7. The algorithm continues iterating until all items in sub-collections are processed.

Unlike Algorithm 6 which requires many passes over a given set S_i to compute the number of items along the iterations, Algorithm 8 allows to compute a *Set Cover* with a good approximation factor with a few passes over the input sets. More clearly, the benefit of this algorithm is that it has a good behavior, especially when sets are stored on disk, as it doesn't need to compute the number of uncovered items for all the sets at each iteration. Whenever a set is removed to a lower sub-collection, it becomes smaller in size, thus, future computation of its number of uncovered items is more efficient.

The worst case running time of Algorithm 8 is:

$$[1 + \frac{1}{p-1}] \times \sum_i \mid S_i \mid \tag{2.8}$$

which is at most $1 + \frac{1}{p-1}$ as large as the time needed to scan all the sets. In practice, such worst case examples is not expected to happen. If a set is not added to the *Set Cover* in a given iteration, it will be most likely moved down multiple levels, rather than just one. In addition to that, it is easy to see that disk-resident sub-collections can be accessed sequentially when the elements of each sub-collection are stored in a separate file.

Figure 2.5 shows an example of executing Algorithm 8 on the input list of sets shown in Figure 2.3 with $p = 2$. Initially, the input sets are sorted into three sub-collections containing sets whose sizes are in the following ranges $[4, 7]$, $[2, 3]$, and, $[1, 1]$, respectively. At step 1, the algorithm considers the first set in the highest sub-collection (note that

Algorithm 8: Disk-friendly Set Cover Algorithm

Input : A set U of n elements.
A collection $S = \{S_1, S_2, ..., S_m\}$ of subsets of U with costs c_i.
Output: A set cover SC

```
1  Sort the subsets S_i of S into K sub-collections according to their cost values c_i;
2  foreach k ← K down to 1 do
3      foreach set S_i in S^(k) do
4          if (| S_i\SC |⩾ p^k) then
5              SC ← SC ∪ S_i;
6              Remove S_i from S^(k) and from further consideration;
           else
7              S_i ← S_i\C;
8              Add the updated set S_i to sub-collection S^(k') such that p^k' ⩽| S_i |⩽ p^(k'+1);
9      foreach set S_i in S^(0) do
10         if (| S_i\SC |= 1) then
11             SC ← SC ∪ S_i;
12             Remove S_i from S^(0) and from further consideration;
13 return SC;
```

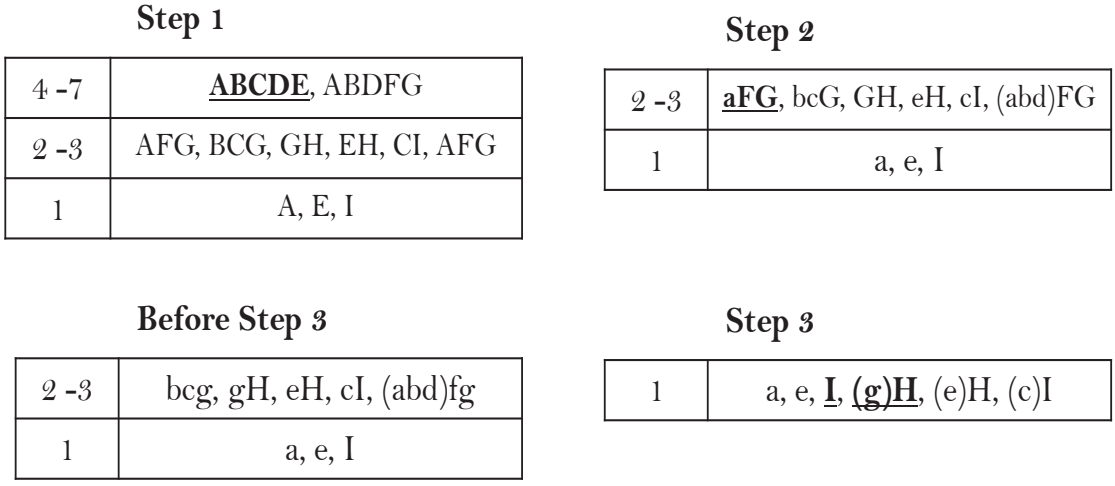


Figure 2.5.: Greedy algorithm execution example

the picked set at each iteration is bold and underlined, and, covered items are shown in lowercase in Figure 2.5). The next set in the same sub-collection now has only 2 uncovered items which are F and G, so the new set is moved to a lower sub-collection. At step 2, the first set aFG has two uncovered elements, so the algorithm adds it to the solution. Consequently, none of the sets in the current sub-collection are correctly placed as all of them cover now less than 2 elements, due to items being previously covered. At step 3, each of the sets I and gH has one uncovered item, thus they are consecutively picked

and added to the solution. At this point, the remaining sets in the current and last sub-collection have no uncovered items. Thus, the resulting *Set Cover* is $\{ABCDE\}$, $\{AFG\}$, $\{GH\}$, $\{I\}$, which is different from the resulting *Set Cover* of the greedy algorithm, but it has the same number of sets which is 4.

2.11. Wilson Score-based Sampling

In this section, we present our sampling strategy that we need in Chapter 4. Studying a given property in a very large set of items is an expensive task to perform. For instance, let us suppose that we have a very large set S of node pairs and an index I which covers some node pairs. Our goal is to determine the number of pairs in S that are not covered in I. Given that the total number of pairs in our case is $O(\mid V \mid^2)$ (see Chapter 4), checking the pairs exhaustively is a very expensive and it prevents us from computing exact statistics. To deal with this, we propose to estimate the number of uncovered pairs by sampling. More clearly, we propose to select an informative sample of node pairs from S based on which we determine the number of uncovered pairs. In order to decide the size of the

Algorithm 9: Estimating number of uncovered node-pair distances

Input : A center node w, partial 2-hop index $\mathcal{L}_G$.
Output: Approximate num. of uncovered shortest paths traversing w.

$N \leftarrow \frac{(Z_{\frac{\alpha}{2}})^2 \times p \times q}{E^2}$;
while *(true)* **do**
  sample uniformly at random a set S of N node pairs (u, v) where u and v are predecessors and successors of w in G, respectively;
  for $(u, v) \in S$ **do**
    if $(d_G(u, w) + d_G(w, v) < d_{\mathcal{L}}(u, v))$ **then**
      declare (u, v) uncovered;
  Using Wilson formula [Wil27] compute a 0.95 confidence interval $\mathcal{I}$;
  if $\mathcal{I}$ is not contained in any interval $[(1+\varepsilon)^{j-1}, (1+\varepsilon)^j]$ corresponding to the sub-collection S_j (that is we cannot determine the right sub-collection for H_w) add 1000 to N else break;
  if ($N > 10000$) break;
return the approximate num. of uncovered node-pair distances;

sample we first initialize the sample size using the following formula [BKH01]:

$$N = \frac{(Z_{\frac{\alpha}{2}})^2 \times p \times q}{E^2} \tag{2.9}$$

$(\frac{Z_{\alpha}}{2})^2$ is the z score separating an area of in the right tail of the standard normal distribution, p and q respectively denote the probability that a node pairs in a given star is covered or not, and, E denotes the error margin. Since we don't have information about the probability that a given node pair is covered or not, we set a probability value of 0.5 to both p and q.

We aim at a confidence of 0.95. After initializing N, we estimate the ratio of covered node-pairs distances by computing the corresponding Wilson Confidence Interval Score [Wil27]. Wilson returns an interval so it might be the case, then that such an interval is not contained in any of the intervals corresponding to the sub-collections see Algorithm 11. If this is the case we cannot find the right sub-collection for the bipartite graph at hand. Hence, we increase the size of the sample by 1000 until we are able to determine with a confidence of 0.95 the right sub-collection for the bipartite graph at hand (that is one interval is contained into the other interval). Algorithm 9 shows a pseudo-code for our sampling strategy.

2.12. Conclusion

In this chapter, we introduced some basic definitions and state-of-the art algorithms as background for our indexing scheme for answering distance queries in very large graphs *EUQLID*, our access control model, and, its corresponding privacy management system *Primates*, which we will respectively detail in Chapters 4 and 5. These algorithms have been used in the literature for a wide range of problems, especially for graph indexing problems. In the next chapter, we discuss and study some properties that real graphs exhibit in order to help devising efficient algorithms on such graphs.

Chapter 3.

Interesting Properties of Real Graphs

3.1. Introduction

Recently, there has been considerable interest in studying complex real-world networks (such as social networks, biological networks, road networks, the Internet, etc.), and, attempting to understand their properties. Several studies have shown that real graphs are not actually random, but, they share some peculiar properties like the small-world diameter, the power-law node degree distribution, and, the community structure properties, to name a few. These properties are interesting as they help to not only understand the nature of real graphs, but also, to devise efficient algorithms to run on such graphs.

In this chapter, we highlight and explain some prominent real graph properties that have been discovered in the literature. We also report some experiments that we have conducted to understand such networks. In particular, we discuss an interesting property of real graphs that we discovered. This property is very interesting, in our case, as it was useful to devise an efficient algorithm for computing distance queries in large directed graphs as explained in Chapter 4.

This chapter is organized as follows: In Section 3.2, we introduce and describe two different models of complex networks (i.e., random and scale-free models). In Section 3.3, we briefly highlight some well-known real-world graph properties. In Section 3.4 and 3.5, we respectively analyze shortest paths in several real graph datasets, and, study the impact of some discovered properties on improving the 2-hop algorithm described in Chapter 2. Section 3.6 concludes the chapter.

3.2. Complex Network Models: Random versus Scale-Free

Analyzing the structural and topological properties of complex networks (i.e., graphs) gives information on how nodes are connected to each other. A noteworthy property among these properties is the connectivity distribution $P(k)$, which denotes the probability that a node chosen at random has k links. $P(k)$ characterizes the architecture of a given network [Bar02, New03]. In the following, we discuss two well-known models of complex networks: *random networks* with a Poisson topology, and, *scale-free networks* with a scale-free topology [Bar02, New03].

Random networks were first described by the Hungarian mathematicians Paul Erdos and Alfréd Rényi. They could be defined as follows.

Definition 3.1. Erdös Réyni Random graphs. Let n be a positive integer, and, let p be a given probability such that $p \in [0, 1]$. The random graph $G(n, p)$ is a graph having n nodes, where there is an edge between two nodes u and v in G with probability p ($P((u, v) \in E(G)) = p$, with these events mutually independent) [ER60].

In such networks, although some nodes may have more connections than others, they all have the same connectivity on average.

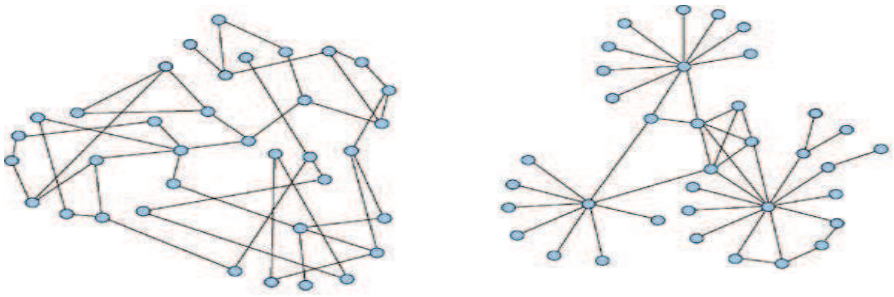
Scale-free networks were introduced by Barabasi et al. in [Bar02]. Such networks could be defined as follows:

Definition 3.2. Scale-free networks. A scale-free network is a graph whose degree distribution follows a power law [BA99]. This can be, mathematically, expressed as follows:

$$P(k) \sim k^{-\gamma} \tag{3.1}$$

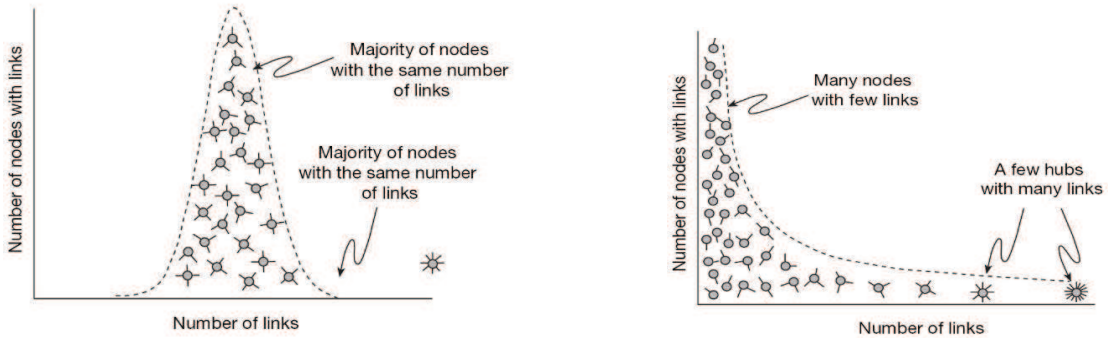
where k is an integer denoting a node degree, $P(k)$ is the fraction of nodes with degree k, and, γ is a parameter whose value is most probably in the range $]2, 3[$.

In contrast to random networks, the defining characteristic of scale-free networks is their high heterogeneity, i.e., some nodes have few connections, some have an average number of connections, and, some have many connections. In other words, in scale-free networks, the mean connectivity of the nodes is not representative of the actual connectivity of the whole network.



(a) Random network (with a probability x) (b) Scale-free network

Figure 3.1.: Random versus scale-free network example



(a) Random network (bell curve)

(b) Scale-free network (power-law distribution)

Figure 3.2.: Degree distributions

Figure 3.1(a) depicts an example of a random network. In such graphs, a plot of the node degree distribution follows a bell-shaped curve as illustrated in Figure 3.2(a) where most of the nodes have approximately the same degree value. Figure 3.1(b) shows an example of a scale-free network with some nodes having very high degrees and the rest having relatively small degree values. In contrast to random networks, in scale-free networks, the node degree distribution follows a power law. This distribution is depicted in Figure 3.2(b), and, it results in a straight line, if plotted in a double logarithmic scale.

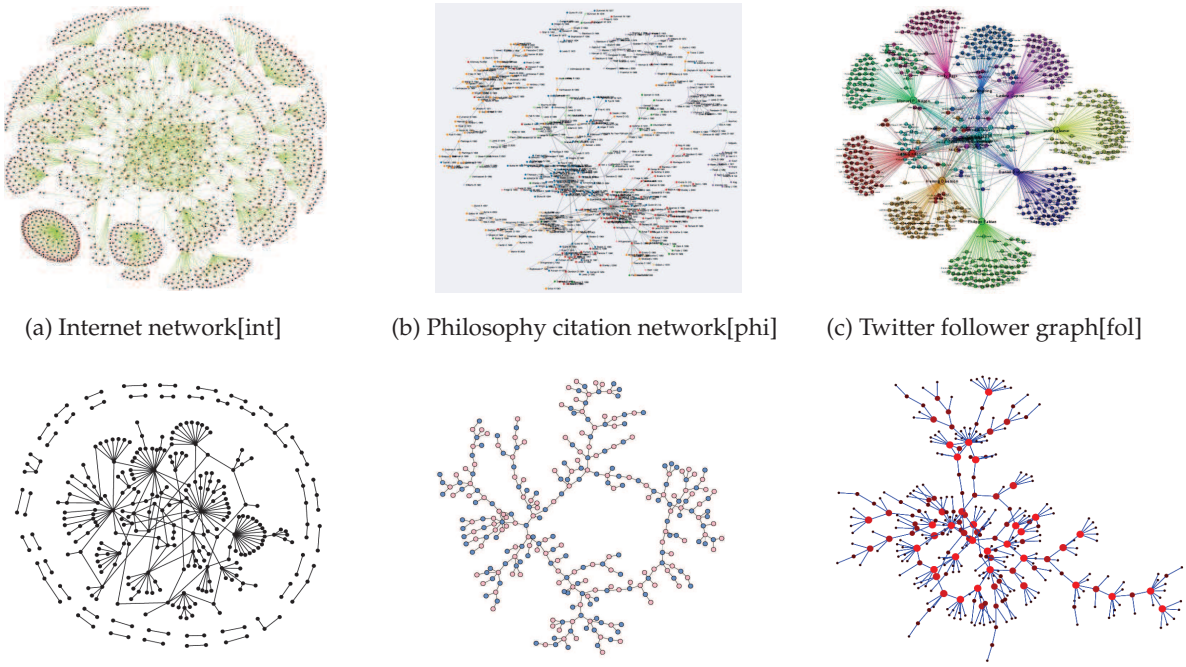
According to several comparative studies on network theory [BA99, FFF99], a large number of real networks adopt a scale-free architecture. This feature was found to be a consequence of two generic mechanisms: (i) networks expand continuously by adding new nodes, and, (ii) new nodes attach preferentially to nodes that are already well connected. Examples of these networks include social networks (e.g., friendships, sexual contacts, scientific collaborations and authors of publications, disease propagation), the Internet network, and, biological networks (e.g., gene regulation networks, protein networks, metabolic networks). Figure 3.3 depicts the network structure of some real graphs.

3.3. Well-known Properties

In this section, we report some of the most important properties that appear in real graphs. We describe two main classes of properties: (i) static properties, which describe the structure of snapshots of graphs, and, (ii) dynamic properties, which describe how the structure evolves over time.

3.3.1. Static properties

Heavy-tailed Degree Distribution. The degree distribution of many real graphs follow a power-law of the form $f(d) = d^{-\alpha}$, where α is a strictly positive parameter whose value is typically in the range $]2, 3[$ and $f(d)$ is the fraction of nodes with degree d. Intuitively, such distribution implies the existence of many low-degree nodes in the graph and a few number of high-degree nodes [LLDM08, FFF99, KKR+99]. Power laws have been found



(a) Internet network[int] (b) Philosophy citation network[phi] (c) Twitter follower graph[fol]

(d) Yeast protein interactions[MS02] (e) High school dating network[BMS02] (e) Sexual network[New03]

Figure 3.3.: Examples of real networks

in the Internet (Faloutsos et al., [FFF99]), the Web (Kleinberg et al., [KKR+99]; Broder et al., [BKM+00]), citation graphs (Redner [Red98]), online social networks (Chakrabarti et al. [CZF04]), etc.

Small Diameter. Most real-world graphs were found to exhibit relatively small diameter (the small- world phenomenon, or "six degrees of separation" by Milgram[TMTM69]). For instance, Boldi et al. [BV12] have recently observed that the diameter of Facebook is 4, which shows that the world is even smaller than what Milgram expected. The diameter of a graph denotes the maximum shortest path distance between any two nodes, which indicates how quickly we can get from one end of the graph to the other end [Bar03].

Community Structure. Real-world graphs exhibit a modular structure, where nodes form communities (i.e., groups of people having common interest(s)), and possibly communities within communities [GN02, SW92, FLGC02]. Generally, members within a community have few relations with people outside that community. Some people, however, are connected to a large number of communities (e.g., celebrities, politicians). Those people may be considered the key nodes which are responsible for the small-world phenomenon described in the previous paragraph. As the number of such key nodes cannot be so large compared to the rest of the nodes in the graph, we can confirm the resulting fact of the power-law degree distribution (which says that in real graphs, we have a small number of high-degree nodes connecting the rest of low-degree nodes).

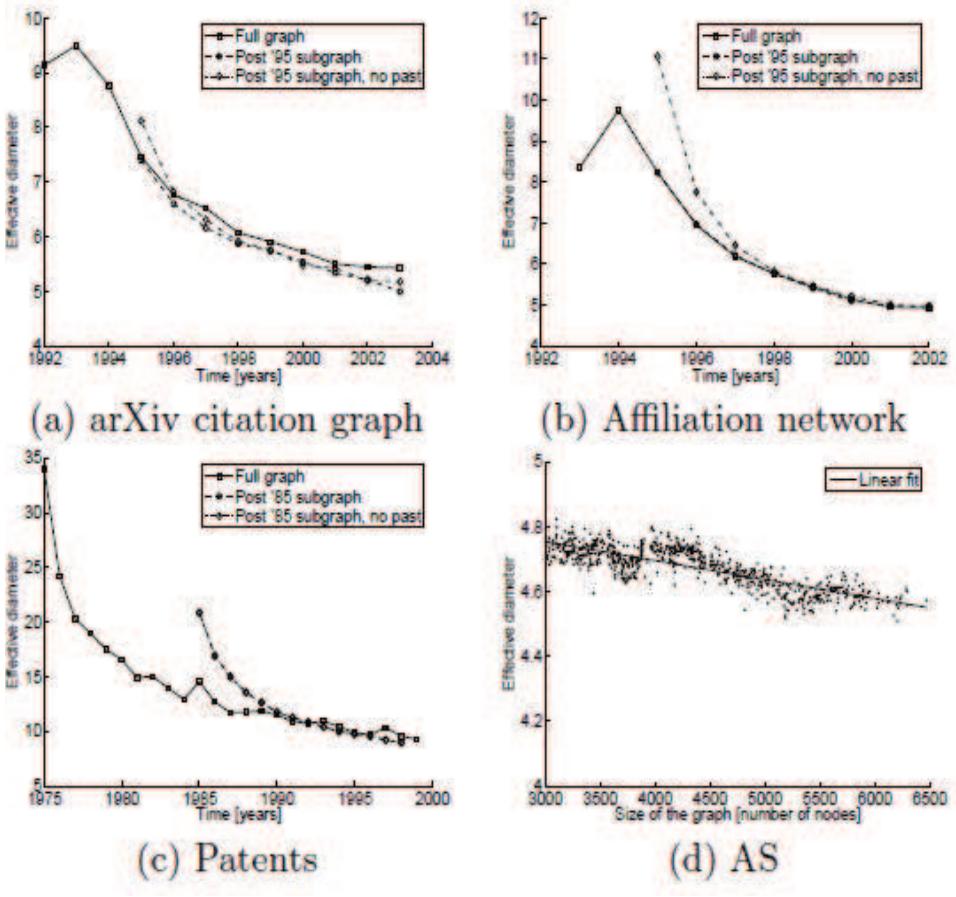


Figure 3.4.: Evolution of diameter over time [LKF05]

3.3.2. Dynamic properties

Shrinking Diameter. Leskovec et al. [LKF05, LKF07] showed that the diameter of real graphs is not only small, but it also shrinks then stabilizes over time. The plots shown in Figure 3.4 illustrate the evolution (i.e., decrease) of the diameter over time of some real graph datasets.

Densification Power Law (DPL). Leskovec et al. [LKF05] have also shown that, in real-world graphs, when the number of nodes doubles, the number of edges doubles even more, hence the densification (i.e., the graphs gets denser). This also explains the shrinking diameter phenomenon observed in real graphs, which was described earlier.

3.4. Analyzing Shortest Paths

Node degree distributions are important tools for studying and understanding the structure of real networks. They give insight into their structures and help designing generative models that capture their growth and dynamics. They are also key tools in the design and analysis of efficient algorithms for a number of challenging graph problems. Based

on the fact that node degrees in social graphs follows a power-law distribution, meaning that in the graph there is small number of high-degree nodes and the rest of nodes have small degree, we proposed to study the way high-degree nodes and low-degree nodes are connected to each other within the graph. In other words, we wanted to measure the importance of high-degree nodes to graph connectivity.

In this section, we take a closer look at shortest paths and check if they exhibit some additional interesting patterns, which could possibly help us to solve our reachability problem (described in Chapter 1). To do this, we sampled a large number of shortest paths for a selection of small and large social graphs, and, studied the degree of nodes that are involved in such paths.

Datasets and Settings. To perform our experiments, we used a set of datasets, which were compressed using the *WebGraph* framework [BRSV11]. This framework provides simple ways to manage very large graphs while exploiting modern compression techniques. We considered graphs of different sizes with a number of edges ranging from several hundreds of thousands to almost one billion and a half.

Sampling strategy. To do this, we sampled 300 seed nodes independently at random for each dataset, and, for each seed node we computed the set of shortest paths of lengths 3 and 4 starting from such nodes. We analyzed the obtained shortest paths by computing the degrees of nodes that they involve.

Results. After studying node degrees that are involved in the very large number of sampled shortest paths, we observed that in each path there is always at least one node with a degree far higher then the other node degrees involved in the same path. Based on this observation, we can say that, in social networks, shortest paths go through high-degree nodes. We denote such nodes as *hubs*. We say that a hub w covers a given path p if and only if p goes through w.

This property is very interesting to improve the 2-hop algorithm described in Chapter2. In fact, instead of considering all the nodes of the graph as candidate centers, we can only consider a subset of nodes S (i.e., the set of high-degree nodes involved in shortest paths). This can make the 2-hop cover indexing process more time-efficient.

In order to determine the set S of high-degree nodes involved in all possible shortest path in a given graph, we suggested to define a degree threshold θ. All the nodes with a degree larger than θ should be part of S. Given a degree threshold θ, S could be formally defined as:

$$S = \{w \mid w \in V \wedge degree(w) \geqslant \theta\} \tag{3.2}$$

θ is determined as follows. For each sampled node pair, we store the maximum node degree involved in its shortest paths. If there is more than a shortest path between two nodes, then, we select the maximum degree among all the degrees in all shortest paths between these nodes. Then, we select the minimum degree among all stored maximum degrees as the degree threshold θ. The algorithm for determining shortest paths is based on a breadth first search which traverses the graph starting from a seed node and computes the shortest path by maintaining the encountered nodes. Then, degrees of intermediate nodes that are involved in shortest paths are efficiently computed.

Table 3.1 reports the set of datasets that we considered along with their sizes, density values, and, the degree threshold θ. Based on θ, we computed the number of hubs (i.e., $\mid S \mid$), and, reported the percentage of these nodes in the corresponding graphs. As shown

Datasets	#Nodes	#Edges	Density	Diameter	θ	% of hubs	Source
Small datasets							
Enron	69 244	276 143	3.98	34.28 ($\pm$ 0.295)	70	2.30	[web]
Amazon 2008	735 323	5 158 388	7.01	13.42 ($\pm$ 0.098)	8	67.87	[web]
DBLP 2010	326 186	1 615 400	4.95	14.64 ($\pm$ 0.135)	18	11.54	[web]
DBLP 2011	986 324	6 707 236	6.8	8.88 ($\pm$ 0.076)	14	22.19	[web]
Large datasets							
Hollywood 2009	1 139 905	113 891 327	99.91	4.14 ($\pm$ 0.027)	1084	37.13	[web]
Hollywood 2011	2 180 759	228 985 632	105	4.92 ($\pm$ 0.045)	494	8.38	[web]
LiveJournal	5 363 260	79 023 142	14.73	7.36 ($\pm$ 0.068)	50	14.50	[web]
Twitter	41 652 230	1 468 365 182	35.25	5.29 ($\pm$ 0.016)	873	1.06	[web]

Table 3.1.: Degree threshold in Shortest Paths

in Table 3.1, the percentage of hubs in the datasets is small and gets even smaller when the graph gets larger and denser. For instance, the percentage of hubs in the *Twitter* dataset is 1.06% which is very small despite the very large size of the graph.

Note that the set of hubs, and, consequently the degree threshold change according to the length of paths that we would like to cover. In fact, the number of hubs that we need to cover paths of length k is the same that we need to cover paths of length $k^{'}$, where $k, k^{'} \in \mathrm{N}$ and $k \leqslant k^{'}$. More clearly, a path of length $k^{'}$ necessarily contains a sub-path of length k. According to our experiments, all paths of length 3 go through hubs, then, all paths of length larger than 3 go also through hubs. Based on this, we can say that the minimum number of hubs that we need to cover paths of length 3 is the same that we would need to handle paths of length n where $n \geqslant 3$. In Table 3.1, we report results to cover paths of length 3, as paths of length 2 are not expensive because traversing the 2-hop neighborhood of a given node in real time is not expensive.

The above experiments as well as those reported in Section 4.5 have shown that real graphs exhibit the two following properties:

- **Property** 1**.** If two nodes u and v are connected in the graph, then, there is at least one short path connecting u and v, which traverses at least one high-degree node.

- **Property** 2**.** The reachability backbone (see Section 3.5) of a real graph forms a strongly connected component.

We define a short path to be a path whose length does not exceed a given threshold. This threshold could be the average distance in the graph. In what follows, we explain these two properties, and, discuss their relation with other well-known properties. The first property is somehow related to the small-world phenomenon, which says that the diameter (longest shortest path) in real graphs is relatively small. It could be seen as a result of it, as the distance between two nodes could not exceed the diameter of the graph in any case. Moreover, this property confirms the densification power law saying that the number of edges increases over time much more than the number of nodes, and, makes graphs get denser over time (causing an increase of the average node degree). The same property also confirms the fact that the node degree distribution follows a power-law meaning that new added nodes to the graph tend to connect to high degree nodes.

The small world phenomenon property can be seen as a result of the second property, as paths going through high degree nodes may consist in shortcut paths between node pairs.

As a result of the community structure property of real graphs, nodes in a given community have few relations with nodes outside that community. However, some nodes (high-degree nodes) are connected to other communities, which confirms the second property (the reachability backbone forms a strongly connected component). Such high-degree nodes can be considered as the key nodes which are responsible for the small-world phenomenon.

3.5. Studying Stars

The subset of high-degree nodes determined based the threshold θ form a graph $G^{'}(V^{'}, E')$, where $V^{'} = S$ and $E^{'} = \{(u,v) \mid u,v \in V^{'} \wedge (u,v) \in E\}$. We define G' as the *Reachability Backbone* of G, as it captures the necessary information to deduce reachability between nodes. A *Star* is the set of all predecessors and successors of a given node v, denoted $S^*(v)$:

$$S^*(v) = P(v) \cup S(v) \tag{3.3}$$

A *k-hop star* of a node v, denoted $S^*_k(v)$, includes the set of predecessors and successors within k hops from v in backward and forward directions, respectively. To compute the 2-hop cover of a given graph, we need to compute for each candidate center v, its star $S^*(v)$, which can be very expensive when the input graph is very large and dense like real-world graphs are. A possible direction to deal with this is to consider k-hop stars (e.g., 2-hop or 4-hop stars) instead of full stars depending on the length of the paths that we are interested in. In this section, we study the impact of varying k on the size of stars and the time needed to get them.

Datasets	Reachability Backbone Size	Avg. 2-hop Star Size	2-hop Star Time	Avg. 4-hop Star Size	4-hop Star Time	Avg. Star Size	Star Time
Small datasets							
Enron	1592	761	16ms	2787	40ms	2976	75ms
Amazon 2008	499094	3	2ms	12	4ms	1747	1sec
DBLP 2010	37646	52	4ms	2500	46ms	7767	500ms
DBLP 2011	218881	527	9ms	120000	500ms	433362	1sec
Large datasets							
Hollywood 2009	423 310	2680	600ms	84618	30sec	84618	6sec
Hollywood 2011	182 851	1662	311ms	365700	30sec	365700	30sec
LiveJournal	778 148	1107	45ms	744391	11sec	1.5M	18sec
Twitter	440 338	206790	90sec	876565	6mins	876565	6mins

Table 3.2.: Statistics on Stars for all datasets

Table 3.2 reports the size of the reachability backbone for all the considered datasets. Note that the size of the reachablity backbone denotes the number of nodes that it contains. We do not report the number of edges as $G^{'}$ is a strongly connected component, where all the nodes $v \in V^{'}$ are connected to each other. This means that all stars of nodes within the reachability backbone overlap too much. We compared the average size of 2-hop stars, 4-hop stars and full stars, as well as the necessary time to get them from the input graph G. It is clear from the table, that considering larger k results in larger stars and takes longer to get them. For some datasets and especially the largest ones where the diameters are small (4 or 5), we observed that considering the distance (the 4-hop neighborhood) or not has no impact on the size of stars and the time needed to retrieve them (i.e., most of the nodes could be reached within a small number of hops).

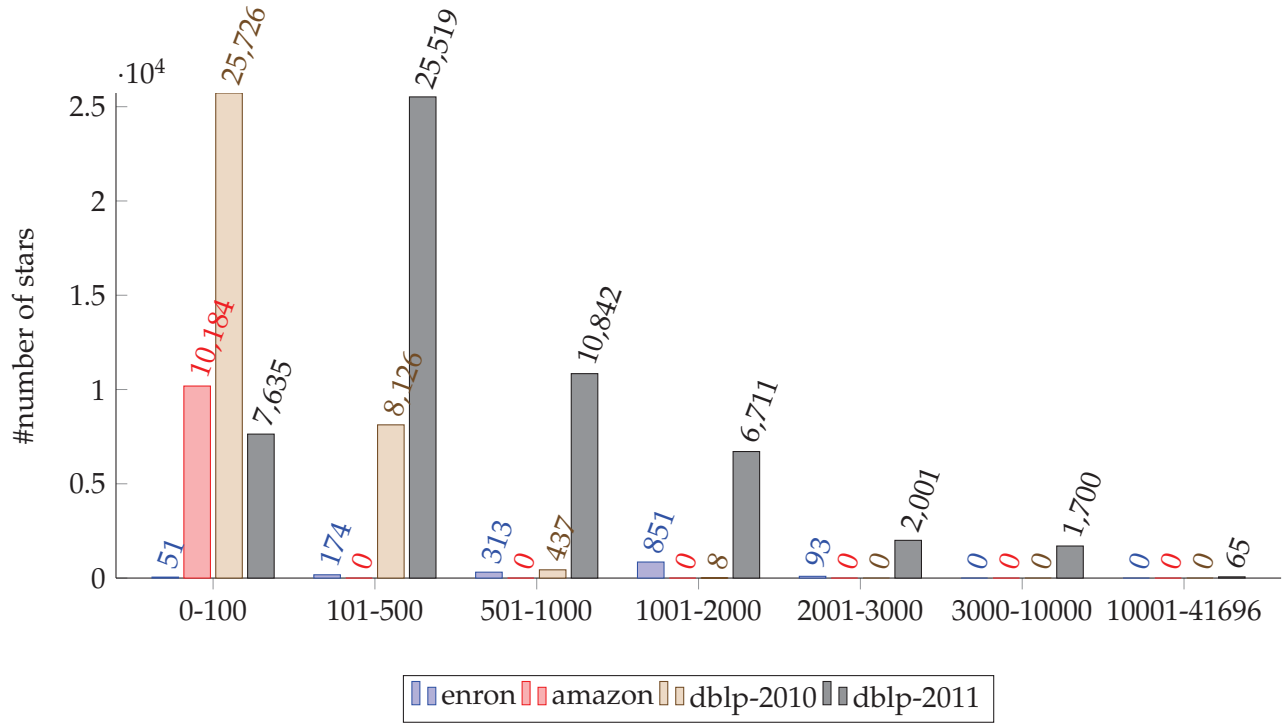


Figure 3.5.: Number of stars wrt to different sizes on small datasets

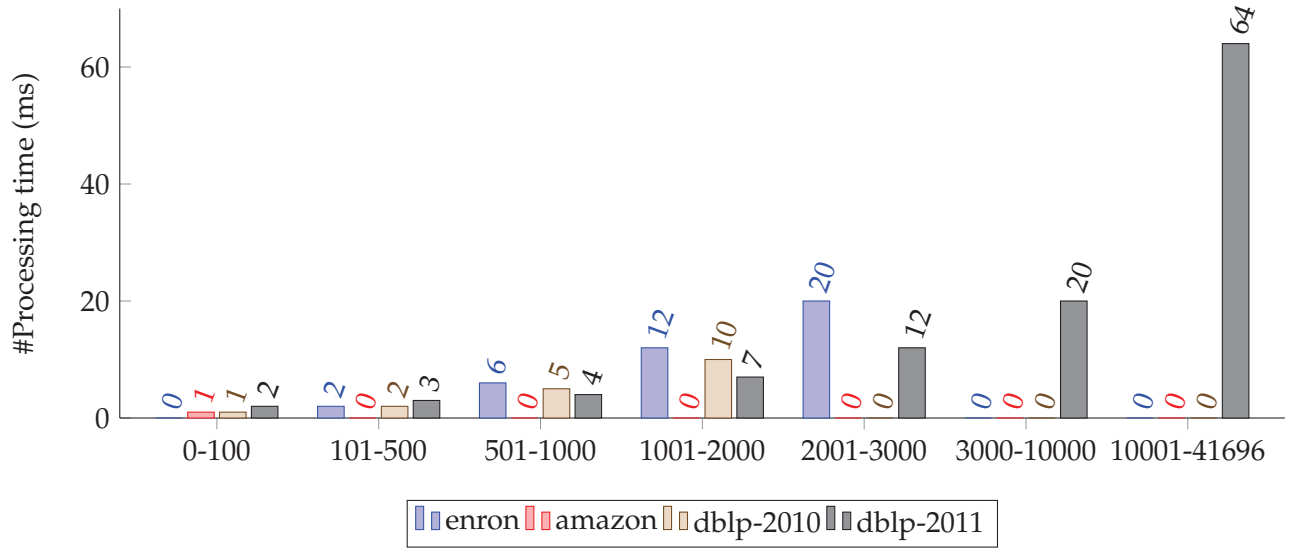


Figure 3.6.: Average processing time get 2-hop stars on small datasets

Figures 3.5, 3.6, 3.7 and 3.8 report histograms illustrating to the number of stars per size category for all the datasets and the time needed to get them.

Experiments on studying 2-hop stars size and processing time on small datasets are reported in Figure 3.5 and Figure 3.6. These experiments were run on a sample of hubs

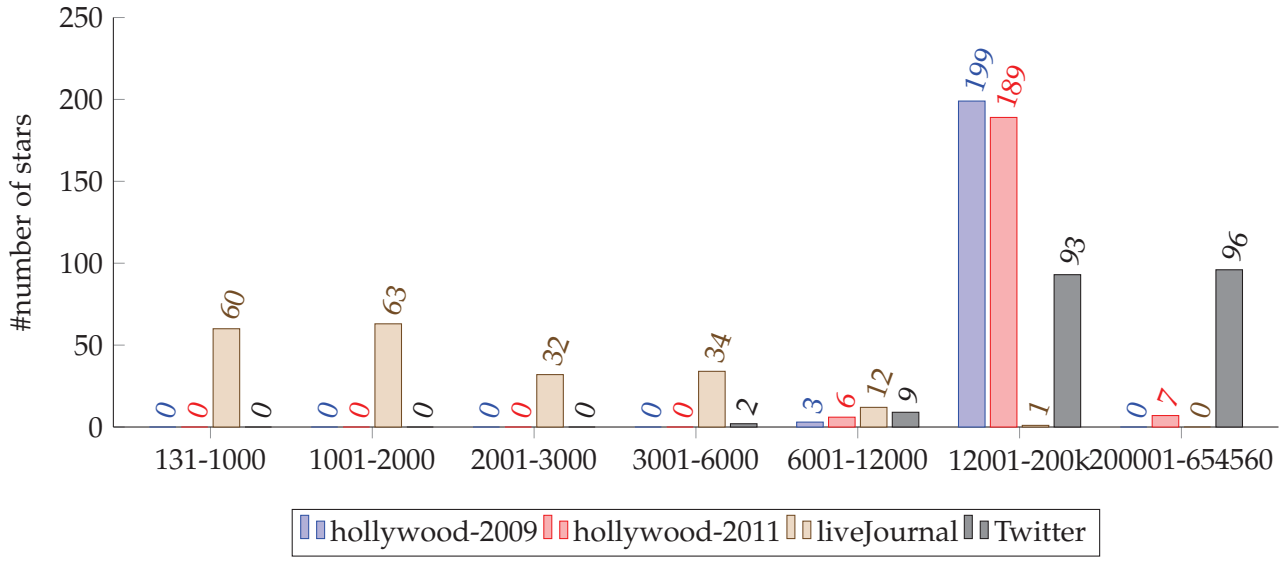


Figure 3.7.: Number of stars wrt to different sizes on large datasets

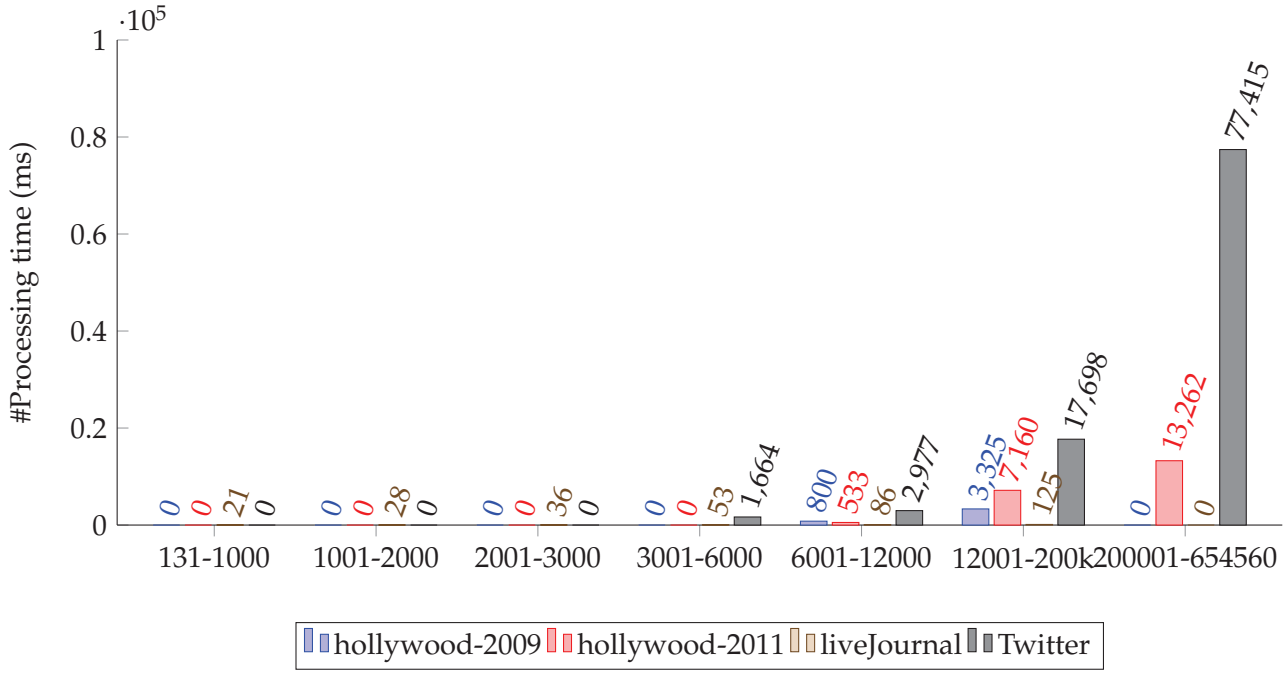


Figure 3.8.: Average processing time to get 2-hop stars on large datasets

from each of the datasets (a sample of 1482 hubs from enron, a sample of 10184 hubs from amazon-2008, a sample of 34276 hubs from dblp-2010, and, a sample of 54473 hubs from dblp-2011).

Experiments on studying 2-hop star sizes and processing time on large datasets are reported in Figure 3.7 and Figure 3.8. These experiments were run on a sample of 200 hubs of each of the datasets.

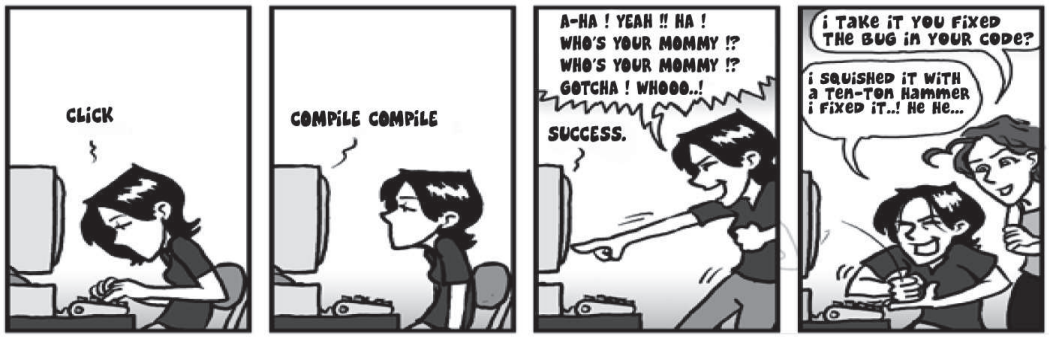
3.6. Conclusion

In this chapter, we studied the structure of real-world graphs in order to understand the properties that they exhibit. We briefly presented some of the well-known properties that were already discovered in the literature. Then, we reported and described our own experiments to study the way high and low-degree nodes are connected in the graph through our shortest path analysis. We also studied the impact of considering the distance when computing the stars to compute the 2-hop cover.

Going back to our original motivation of scaling reachability queries to large graphs, we have now revealed and understood some interesting real graph properties that can be used to devise an efficient algorithm to deal with the reachability problem described in Chapter 1. We now proceed, in the following chapter, to the issue of answering distance queries in large directed graphs.

Chapter 4.

Answering Distance Queries in Large Directed Graphs



4.1. Introduction

Computing the distance between any two nodes in a directed graph is a fundamental operation required in several applications encompassing social networks, road networks, semantic web and bio-informatics. State-of-the-art algorithms fail at coping with large real-world graphs which may be dense and may contain billions of links. In this chapter, we present *EUQLID* a method for indexing large directed graphs so to efficiently answer distance queries. *EUQLID* is based on a fast algorithm for a variant of the 2-hop cover problem where an additional constraint on the size of the index is enforced. It also exploits the property of modern real-world graphs that a few nodes connect all nodes through short paths. Our evaluation on directed social graphs and web graphs shows that *EUQLID* outperforms recent approaches and that distance queries can be processed within hundreds of milliseconds on very large publicly available graphs.

The rest of this chapter is organized as follows. Section 4.2 discusses existing work about the general reachability problem, Section 4.3 defines formally the problem of answering distance queries and introduces notations and definitions. Section 4.4 describes both the indexing algorithm and the query processing algorithm, while Section 4.5 contains an extensive experimental evaluation of our approach against the state of the art. Finally, we include in Section 4.6 our conclusions.

4.2. Related Work

As described in Section 1.1.1, we distinguish three main types of reachability queries: *simple reachability queries*, *distance queries*, and, *distance and reachability queries with constraints*

which are even more general and consider constraints on edge labels, label order, distance, edge direction, etc. We present here related work about answering these three types of reachability queries.

4.2.1. Simple reachability

Reachability queries consist in checking whether there is a path (of any distance) connecting two given nodes in a graph. The topic of answering reachability queries has been intensively studied, see for instance [ABJ89, BCN08, CP10, CC08, CSC+12, CYL+06, CYL+08, CHKZ02, JRDX12, JRXW11, JXRF09, STW04, vSdM11], and, [YCZ10, SABW13, CHWF13] for more recent works. A recent book surveying a number of existing indexing scheme can be found in [AW10].

Existing approaches for answering simple reachability queries can be classified in two main categories: (i) off-line indexing approaches, and, (ii) online guided search approaches. The first category proposes to pre-compute indexes which capture all reachability information stored in a compact way. However, the second category of approaches proposes to pre-compute an index holding some information which helps improving online search.

Several approaches have proposed new techniques [BCN08, CP10, CSC+12, CYL+06, CYL+08, JXRF09, STW04] to improve the original 2-hop labeling [CHKZ02]. However, they are very costly in constructing the index and they cannot handle large graphs. As pointed out in [JRDX12], most of the existing methods can handle relatively small graphs (i.e., with tens to hundreds of thousands vertices and edges). To process larger graphs, these methods are either too costly in indexing or in query processing which limits their application to real-world graphs. For graphs with millions of vertices and edges, only a few methods [YCZ10, JRDX12, SABW13, CHWF13] were proposed with reasonably good efficiency. For instance, Jin et al. [JRDX12] proposed a backbone structure as a general framework, called *SCARAB*, on top of which existing indexing methods can be applied. A reachability query (u, v) can be answered by first finding all backbone vertices that can be reached from u (denoted B_u) and all backbone vertices that can reach v (denoted B_v). Then, they check whether any vertex in B_u can reach any vertex in B_v. Any existing method can be applied to the backbone graph to process the second step, and, querying is generally faster since the backbone might be significantly smaller than the original graph. Although *SCARAB* can be used as a general framework to further improve the scalability of any reachability index, an efficient and scalable method itself is still crucial for query performance. In fact, *SCARAB* itself may not be scalable to large graphs, and, the backbone of a large graph may also still be too large for existing methods.

Yildirim et al. [YCZ10] proposed the *GRAIL* index which uses k random trees to cover a DAG, generating as many intervals as a label for each node. Their query processing algorithm uses these labels to quickly determine non-reachability, otherwise it recursively queries the nodes underneath in the DAG. In the same perspective, Seufert et al. [SABW13] proposed *Ferrari* which trades off query performance for reduced index size and indexing cost.

4.2.2. Distance queries

Given the sheer size of modern graphs in a large number of applications, computing distance in graph-structured data has become an important problem in the database research

community. In this section, we describe the most recent algorithms developed in this area of research. For an exhaustive and detailed survey about processing distance queries see [AW10].

Shortest Path Computation. One of the most well-known methods for shortest path computation is Dijkstra's algorithm [Dij59], which recursively traverses the graph on the fly, while maintaining the length of the shortest path so far found between a given source node s and the set of traversed nodes. Although, this algorithm is elegant and simple, but it is unfortunately non-efficient for large-scale graphs.

Recently, Wei [Wei10] proposed *TEDI* as a tree decomposition-based index for answering shortest path queries online on unweighted undirected graphs. They utilize the tree-width decomposition to help reduce the search space. This method is does not scale well on large graphs, because, it precomputes pair-wise shortest path distance between all the nodes that are stored in each tree, which is prohibitively expensive in large graphs and requires a huge storage space. In addition to that, since directed tree decomposition tends to be more elaborate and difficult, it is not clear whether it can be equally effective on weighted and/or directed graphs. Xiao et al. [XWP$^+$09] also proposed a tree decomposition-based index to answer shortest path queries online. Despite the modest size of the input graphs, the overall size of all the compressed BFS trees is still very large. Both [Wei10] and [XWP$^+$09] assume that the index fits into main memory which is not the case when the input graph is large or when there are some memory constraints.

Cheng et al. [CKCC12] proposed a disk-based vertex cover approach to answer SSSPs (Single Source Shortest Paths) in large undirected graphs, which can also be used to answer point to point distance queries. However, when applying this technique to point to point distance queries, many irrelevant nodes could be traversed before reaching the desired target node. Another vertex cover-based approach [CSC$^+$12] was proposed by the same author to find the *k-reachable* nodes starting from a given source node for applications like finding the small world of a user (i.e., closest people) in a social graph.

Landmark Encoding. Several works proposed to use *landmarks* to approximate the shortest path distance [PBCG09, GBSW10]. The main idea is to precompute the distance of the shortest path between all the nodes in the graph and these landmark nodes, then apply the triangle inequality to estimate the shortest path distance. This method uses a set of heuristics to select the landmarks. These heuristics are based on node properties such as the degree and the centrality. More clearly, they introduced the Landmark-Cover problem which tries to find the minimum number of landmarks such that for any pair of vertices u and v, there exist at least one landmark in the shortest path from u to v. However, their objective is to efficiently compute estimates of the actual distance while our goal is to develop an exact algorithm for computing the distance between any two nodes in the graph.

2-HOP Labeling. Cohen et al. [CHKZ02] proposed the *2-hop* labeling, which is a compact representation of the transitive closure of the graph. See next section and Section 2.6 for a more detailed explanation of this approach.

Shortest path distance computation. A theoretical analysis of several practical point to point shortest path methods (which was based on modeling road networks as graphs with

low highway dimension) was conducted by Abraham et al. [AFGW10]. According to this study, the labeling algorithm had the best time bounds among all the studied methods. However, the existence of a practical implementation is still an open issue. Based on this study, Abraham et al. [AFGW10] recently developed a fast and practical algorithm to heuristically construct the distance labeling on large road networks. However, it is not clear how this technique which was specifically designed for road networks can be broadened to larger class of graphs. Jin et al. [JRXL12] proposed a highway-centric labeling approach to answer distance queries in large sparse directed graphs. This approach is based on the 2-hop labeling approach. It proposes to select a spanning tree in the graph as a highway. Then, it considers the spanning tree nodes as centers in the 2-hop index. The distance between two given nodes u and v can be computed as the length of the shortest path from u to some vertex w_i in the highway, then from w_i via the highway (i.e., a path in the spanning tree) to some vertex w_j, and finally from w_j to v. This approach cannot scale to large and dense graphs because it requires pre-computing all pairs of shortest paths in the graph, which is prohibitive. *IS-label* is a very recent approach that was proposed by Cheng. et al [AHCR13] to answer point to point distance queries in large directed and undirected graphs. To the best of our knowledge, it is the most scalable approach that was so far proposed to answer distance queries. Their indexing algorithm has two main steps : (i) they first compute a vertex hierarchy (i.e., they consider different levels of nodes), then, (ii) they assign labels to vertices $v \in V$ based the the previously computed vertex hierarchy. A vertex belonging to a given hierarchy can be assigned only vertices from the same hierarchy as its label. To answer a distance query, they preform a bi-directional Dijkstra's algorithm guided by the computed index. In this chapter, we conducted an extensive evaluation on large real world graphs which shows that our technique outperforms *IS-label*.

4.2.3. Labeled queries

Existing work has mostly focused on simple reachability and distance queries. Less attention has been paid to distance and reachability queries with constraints, despite its importance in many application scenarios (i.e., social networks, protein-protein interaction networks, RDF graphs, etc.).

Fan et al. [FLM$^+$11] have addressed the problem of adding regular expressions and patterns to reachability queries. They have evaluated their algorithm on synthetic graphs of up to 1M nodes and 4M edges. Gubichev and Neumann [GN11] have implemented a technique of evaluating path queries over RDF graphs using purely database style indexing and efficient join processing techniques. Although Gubichev and Neumann have performed experiments on very large RDF graphs, their queries use more join-like expressions than rich path-patterns.

Jin et al. [JHW$^+$10] proposed a tree-structured index for evaluating label-constraint reachability (LCR) queries. Given a source node u, a target node v, and, a set of edge labels S, their goal was to determine whether there is any path between from u to v such that each edge label on that path is in S. No labels other than those in S can appear on that path and no order is imposed on edge labels. The distance between nodes is not taken into account. Jin et al. have evaluated their algorithm on synthetic and real graphs of up to 100K nodes and up to 150K edges. One of the major drawbacks of this approach is the fact that it requires materializing the transitive closure (i.e., which is to compute and store all

Notation	Description
$G(V, E)$	is a weighted directed graph
$w(e)$	is the weight of a given edge e
$u \rightsquigarrow v$	is a directed path from u to v
$P(v)$	is the set of nodes that can reach v
$S(v)$	is the set of nodes v can reach
H_w	is an undirected bipartite graph where $V(H_w) \subseteq P(w) \cup S(w)$ and there is an edge going from $u \in V(H_w)$ to $v \in V(H_w)$ iif $d(u, v)$ is not yet covered by the current $\mathcal{L}_G$
S_w	denotes a shortcut for $E(H_w)$
$L(v)$	is the label of v ($L_{in}(v) \cup L_{out}(v)$)
$\mathcal{L}_G$	is the two-hop distance labeling ($L_{in} \cup L_{out}$)
k	is the maximum allowed number of center nodes in $\mathcal{L}_G$
C	is the initial set of K nodes with largest out-degree in G (candidate centers)
$C_{\mathcal{L}_\mathcal{G}}$	is the set of centers in $\mathcal{L}_\mathcal{G}$
$d(u, v)$	is the distance between node u and node v
$d(u, v)?$	is a query asking for the distance between node u and node v

Table 4.1.: Frequently used notations I

possible path-labels between all node pairs in the graph). This is obviously too expensive and impossible to perform for graphs with millions of vertices. In addition to that, to evaluate a query between two given nodes u and v, they get the set successors of u and predecessors of v, then they perform a join between these two sets. In real large graphs having many high-degree nodes, the join process may result in a very large number of joins which can drastically affect query performance.

Atre et al. [ACZ12] have recently proposed *BitPath* which is an index for answering label order constraint queries. Given a source node u, a target node v, and, a sequence of labels $S = (A, B, C)$, their goal is to determine whether u can reach v through a path, where all labels in S appear in the specified order. No constraints on the distance are taken into account. Their approach can be described as follows. Their index is stored as a set of bit vectors of length $\mid E \mid$ and compressed using the Run Length Encoding scheme. The process of evaluating such a query starts by first checking the reachability between u and v. Then, they recursively split the query according to the high selectivity of labels in S. The algorithm stops whenever all the leaf nodes of the query tree are satisfied (yes answer) or when all node of the query tree are exhausted. While this approach aims to answer constraint reachability, it is not clear how they deal with simple reachability (i.e., they just consider it to be trivial and the solution for it boils into computing the intersection between two bit vectors).

4.3. Preliminaries and Formal Settings

We introduce the following notations and definitions for directed and undirected graphs. Let $G = (V, E)$ be a weighted directed graph where V is a set of nodes and E is a set of

edges, i.e. a set of ordered pairs (u, v) where u and v are nodes in G. Each edge comes with a positive weight $w(e)$. A directed path $\mathcal{P} = u \rightsquigarrow v$ is defined as a set of nodes $u_0, u_1, \ldots, u_k$ where $u_0 = u$,$u_k = v$ and (u_i, u_{i+1}) are edges of G, $i = 0, \ldots, k-1$. If there is a directed path $u \rightsquigarrow v$ in G, we say that u is a *predecessor* of v and v is a *successor* of u, while denoting with $S(v)$ and $P(v)$ the set of successors and predecessors of v, respectively. The length of a path $\mathcal{P}$ is defined as the sum of the weights of the edges in $\mathcal{P}$. We let $d(u, v)$ to be the distance between u and v in G, that is, the length of the shortest path connecting u and v in G. The *in-degree* of a node v is defined as the number of predecessors u of v such that there is an edge (u, v) in G, while the *out-degree* of v is defined as the number of successors u of v such that there is an edge (v, u). We denote by $V(G)$ and $E(G)$ the set of vertices and the set of edges of G, respectively. The density of an undirected graph G is defined as $|E(G)|/|V(G)|$. Table 4.1 summarizes the main notations that are used in this chapter.

The classic algorithm for computing distances between nodes in a graph is the well-known Dijkstra's algorithm [Dij59], whose running time in the worst case is $O(|E| + |V| \log |V|)$ when a min-priority queue is employed. If the input graph is unweighted then one could use the simple breadth first search algorithm whose worst-case running time is $O(|V| + |E|)$. Unfortunately, none of these algorithms are efficient when the input graph is large as shown by our experiments.

Our approach is based on a variant of the so-called 2-hop distance labeling which is defined as follows.

Definition 4.1. 2-hop Distance Labeling. Let $G = (V, E)$ be a weighted directed graph. A 2-hop distance labeling $\mathcal{L}_G$ of G assigns to each vertex $v \in V$ a label $L(v) = (L_{\text{in}}(v), L_{\text{out}}(v))$, such that $L_{in}(v)$ is a collection of pairs $(w, d(w, v))$ where w is a predecessor of v and similarly $L_{\text{out}}(v)$ is a collection of pairs $(w, d(v, w))$ where w is a successor of v and the following holds: for any two vertices u, v in V:

a) if u is a predecessor of v in G then there is a node $w \in V$ called *center* such that $(w, d(u, w)) \in L_{\text{out}}(u)$, $(w, d(w, v)) \in L_{\text{in}}(v)$, and $d(u, v) = d(u, w) + d(w, v)$.

b) if u is not a predecessor of v then there is no node w such that $(w, d(u, w)) \in L_{\text{out}}(u)$ and $(w, d(w, v)) \in L_{\text{in}}(v)$.

The size of the labeling is defined to be:

$$\sum_{v \in V} |L_{\text{in}}(v)| + |L_{\text{out}}(v)|$$

We denote with $C_{\mathcal{L}}$ the set of centers of a 2-hop distance labeling $\mathcal{L}_G$. Definition 4.1 suggests how to compute distance queries given a 2-hop distance labeling (see Section 4.4.2).

To the best of our knowledge, no efficient algorithm for computing a 2-hop distance labeling is known, while computing a compact representation of all-pairs distances suffers from the inherent problem that the number of node pairs is quadratic. For this reason, we define a variant of the 2-hop cover problem where we enforce an additional constraint on the number of centers in the labeling. Such a variant was inspired by the so-called *max cover* problem, for which we include a definition for completeness and self-containment.

The (unweighted) *max cover problem* [Hoc97a] is defined as follows. We are given an integer $k > 0$ as well as a collection of sets $\mathcal{S} = \{S_1, \ldots, S_m\}$ defined over a domain of

elements $X = \{1, \ldots, n\}$. We seek to find a sub-collection $\bar{\mathcal{S}} \subseteq \mathcal{S}$ containing at most k sets covering the largest number of elements, i.e., we wish that $|\bigcup_{S \in \bar{\mathcal{S}}} S|$ will be maximized.

There is a simple greedy algorithm for computing the max cover with approximation guarantee $(1 - \frac{1}{e}) \approx 0.63$ which proceeds as follows. In the first step, we include in the solution a set S_{i_1} with maximum cardinality among all S_i's. At step $t \geqslant 2$, let X_t be the current set of covered elements, i.e. $X_t = \bigcup_{j=1}^{t} S_{i_j}$; we include in the solution the set S containing a largest number of uncovered elements that is a set maximizing $|S_i \setminus X_t|$ among all S_i's. We iterate until all elements in X are covered or exactly k sets have been included in the solution. For a proof of the approximation guarantee of this algorithm see [Hoc97a]. Max cover is an instance of the general problem of maximizing monotone submodular functions subject to cardinality constraints [GLNF78]

The algorithm for max cover is quite simple and straightforward to implement. However, when the input is very large, such an algorithm turns out to be non-efficient, the main bottleneck being that we need to compute $S_i \setminus X_t$ for each S_i in $\mathcal{S}$ at each step. As a result, the running time of the algorithm is $O(k \sum_i |S_i|)$ which is also the average running time of the algorithm according to our experiments. In Section 4.4.1, we present a more efficient algorithm for this problem which is an adaptation of the algorithm for set cover presented in [CKW10]. The worst-case running time of this algorithm is bounded by $O((1 + \frac{1}{\varepsilon}) \sum_i |S_i|)$, where $\varepsilon > 0$ is a parameter given in input to determine its approximation guarantee. According to our experiments, such an algorithm works much better in practice.

We formalize the budgeted version of the 2-hop distance labeling problem as follows. We relax the constraint a) of Definition 4.1 and we call *partial 2-hop distance labeling* a labeling where such a constraint might not hold for every u, v where u is a predecessor of v. Moreover, we say that the distance between u and v is *covered* in $\mathcal{L}_G$ if such a constraint holds for u and v and *uncovered* otherwise. We then define the following variant of the 2-hop cover problem.

Problem definition: Max Budgeted 2-hop Distance Labeling (MAX BDL). Given a weighted directed graph $G = (V, E)$ and a positive integer $k > 0$, we seek to find a partial 2-hop distance labeling $\mathcal{L}_G$ containing at most k centers and covering the largest number of node-pair distances.

By a reduction from max cover, we can show that Max BDL is also NP-Hard. The reason for enforcing an upper bound on the number of centers in the index is motivated by the fact that such a parameter affects both the size of the index and the query time. In the next section, we develop an approximation algorithm for Max BDL as well as an efficient heuristic.

4.4. EUQLID

In this section we present the algorithm for building an index (see Section 4.4.1) and the corresponding query processing algorithm (see Section 4.4.2). Our goal during the indexing phase is to cover as much distance information as possible without violating the constraint on the number of centers allowed in the index. We show that this variant of the 2-hop cover admits a 0.63-approximation algorithm which inspired our efficient indexing algorithm. In order to select which centers should be included in our index, we exploit one property that real-world networks often exhibit, that is, a few nodes connect

all nodes through short paths. These nodes are usually the nodes having largest degree in the network.

Our idea is to carefully select a set of large degree nodes and to store all distance information corresponding to paths traversing those nodes. As paths traversing large degree nodes are not necessarily shortest paths one needs to employ Dijkstra's algorithm to find exact distances at query time. However, the search space of Dijkstra's algorithm can be effectively pruned by avoiding traversing large degree nodes (as the corresponding distance information is covered by the index) and using the length of the paths traversing large degree nodes as maximum depth in the Dijkstra's algorithm.

One of the technical issues that we need to cope with is to avoid storing redundant distance information in the index and to do it efficiently. To achieve this task we employ several techniques from statistics as well as recent results for efficiently optimizing submodular functions subject to cardinality constraints. This is discussed in the rest of this section.

4.4.1. Indexing Algorithm

We start by devising a greedy approximation algorithm for Max BDL, which turns out to be non-practical as it computes all-pairs shortest paths. Later on in this section, we shall see how to adapt our algorithm so to deal with massive datasets.

Our greedy approximation algorithm is inspired by the algorithms for the 2-hop cover [CHKZ02] and for maximum cover (see [Hoc97a] and Section 2.6), respectively. The first step is to compute all-pairs shortest paths and declare all node-pairs distances uncovered. Then, we greedily cover a large number of node-pair distances as follows. At each step t, for each node w we let $H_w = (P(w), S(w), E)$ be an undirected bipartite graph where there is an edge between u and v if the distance between u and v is uncovered at step t, with u and v being predecessor and successor of w on the shortest path connecting u and v, respectively. At each step, we select the node w whose corresponding H_w contains the largest number of edges and we include it in the labeling $\mathcal{L}_G$, that is, we add $(w, d(u, w))$ to $L_{\text{out}}(u)$ for every u in $P(w)$ and $(w, d(w, v))$ to $L_{\text{in}}(v)$ for every v in $S(w)$. We then update the set of covered node-pair distances, accordingly. We iterate until all node-pair distances are covered or exactly k centers are included in our index. See Algorithm 10 for a pseudo-code. Theorem 4.2 follows from the results in [CHKZ02] and [Hoc97a].

Theorem 4.2. *Algorithm 10 is a* $(1 - \frac{1}{e}) \approx 0.63$*-approximation algorithm for Max BDL.*

Proof. **(Sketch)**. The proof proceeds by turning any instance of Max BDL to an instance of max cover (with the same value for an optimum solution) as follows. For every bipartite graph H_w introduce a set S_w containing the set $E(H_w)$ of edges of H_w. Then, the approximation guarantee of Max BDL follows directly from the approximation guarantee of the algorithm for max cover. Alternatively, one could show that the objective function of Max BDL is monotone and submodular. □

Computing all-pairs shortest paths is prohibitive for large graphs. In order to alleviate the computational cost of our greedy algorithm we make use of sampling techniques and we relax our definition of uncovered distances as follows. At any step t of our algorithm let $d^t(u, v)$ denote the distance between u ad v in the index at step t, (i.e., at step $t = 0$ let $d^0(u, v) = \infty$). Given a node w we include an edge in the corresponding bipartite graph

Algorithm 10: A (slow) 0.63-approx. algorithm for BDL

Input : A directed graph $G = (V, E)$, a positive k.
Output: An approximate BDL $\mathcal{L}_G$ with at most k centers.

```
1 L_G ← ∅;
2 size ← 0;
3 compute all-pairs distances T in G;
4 while (size < k and T ≠ ∅) do
5 |   for each w ∈ C let H_w = (P(w), S(w), E) be a bipartite graph where (u, v) ∈ E if
  |   the distance between u and v is uncovered;
6 |   let w be a node such that H_w contains the largest number of edges;
7 |   add w to L_G, remove the covered distances from T;
8 |   size ← size +1 ;
9 return L_G;
```

H_w if there is a path $u \rightsquigarrow w \rightsquigarrow v$ whose length is smaller than $d^t(u, v)$. In other words, we include an edge (u, v) in H_w if by including w the distance between u and v in the index decreases. We then estimate the number of edges of H_w (i.e. the number of uncovered node-pair distances) by sampling the edges of H_w. The size of the sample is determined by the Wilson interval formula [Wil27]. According to our experiments, samples containing 2000 edges of the H_w's suffice in order to obtain accurate estimates with confidence 0.95 (for the largest datasets). This is much smaller than the quadratic number of shortest-path computations required by the greedy algorithm in the worst case. Notice that at step $t = 1$ the number of edges in H_w can be computed efficiently as $|P(w)| \times |S(w)|$ (as initially all node-pairs distances are uncovered).

Another bottleneck of the greedy algorithm is that at each step we need to either keep track of all node-pair distances that are covered at that step or to store/update all bipartite graphs. None of these approaches is feasible. Our solution is to devise an efficient algorithm for max cover (see Section 2.8) which will be used in our indexing algorithm. For presentation issues, we first present our efficient algorithm for maximum cover and we then show how it can be used to solve Max BDL.

Our algorithm was inspired by the "disk friendly" algorithm for a different problem (set cover) which was presented in [CKW10]. Given $\varepsilon > 0$, our algorithm first partitions the S_i's into sub-collections $\mathcal{S}_1, \ldots, \mathcal{S}_l$, where $l = O(\frac{\log n}{\varepsilon})$ as follows. At step t, let X_t be the set of covered elements; we say that a sub-collection $\mathcal{S}_j$ is the *right* sub-collection of a set S if the following holds:

$$(1 + \varepsilon)^{j-1} \leqslant |S \setminus X_t| \leqslant (1 + \varepsilon)^j.$$

Then, we first assign each of the S_i's to its right sub-collection. At step t, when looking for the set with best cardinality we pick (arbitrarily) a set S in $\mathcal{S}_{\bar{l}}$, where $\bar{l}$ is largest so that $\mathcal{S}_{\bar{l}}$ is non-empty. We then check whether $\mathcal{S}_{\bar{l}}$ is still the right sub-collection for S (this might not be the case as X_t is updated throughout the algorithm). If this is the case then we include S in the solution and we update X_t. If $\mathcal{S}_{\bar{l}}$ is not the right sub-collection anymore, we move S into its right sub-collection or remove it if $S \setminus X_t$ is empty. We stop when exactly k sets have been included in the solution or all sub-collections are empty.

Algorithm 11: Fast algorithm for Max BDL

Input : A directed graph $G = (V, E)$, $k > 0$, $\varepsilon > 0$
Output: An approximate solution $\mathcal{L}_G$ to Max BDL

$C \leftarrow l$ nodes with largest out-degree in G;
$\mathcal{L}_G \leftarrow \varnothing$;
$size \leftarrow 0$;
for each $w \in C$ let $|S_w| = |P(w)| \times |S(w)|$;
partition the S_w's into $\mathcal{S}_1, \ldots, \mathcal{S}_l$ where $l = \log n/\varepsilon$ so that $S_w \in \mathcal{S}_j$ if

$$(1+\varepsilon)^{j-1} \leqslant \frac{|S_w|}{c_w} \leqslant (1+\varepsilon)^j,\ j = 2, \ldots, l+1$$

while *(size $< k$ and there is $\mathcal{S}_j \neq \varnothing$)* **do**
    choose S_w arbitrarily from the first non-empty sub-collection $\mathcal{S}_j$;
    estimate the cardinality of S_w i.e. the number of edges in H_w by sampling N node pairs from $P(w) \times S(w)$. N is determined using Wilson score interval formula so that we can determine the right sub-collection of S_w with confidence of 0.95;
    if $\mathcal{S}_j$ is still the right sub-collections of S_w then add w to the index, add 1 to size, remove S_w;
    else place S_w into the right sub-collection;
return $\mathcal{L}_G$;

It can be shown that for any $\varepsilon > 0$ our efficient algorithm for maximum cover always computes a solution that is a factor of $(1-\varepsilon)(1-\frac{1}{e})$ from the optimum solution, while we can bound the worst-case running time of our algorithm by $O((1+\frac{1}{\varepsilon})\sum_i |S_i|)$. From a practical point of view, our algorithm is very efficient as often "good" solutions can be computed by processing solely the first few sub-collections. Our algorithm for maximum cover can be used in our algorithm for Max BDL by replacing each of the S_i's with the set of edges in H_w's (or an estimator of their cardinality).

Finally, to further improve the running time of the greedy algorithm we restrict the set of candidate nodes that might be included in the index to be the set of K nodes with largest out-degree in G, where K is a parameter specified in input. The reason for doing this is that the nodes with large out-degree increase significantly the search space of the Dijkstra's algorithm having significant impact on the running time of the shortest path computation. If nodes with large out-degree are included in the index, one could safely avoid traversing large out-degree nodes when executing Dijkstra's algorithm as shortest paths traversing those nodes are covered by our index. This is illustrated in Figure 4.2. Algorithm 11 shows a pseudo-code for our fast algorithm for Max BDL. As observed in the previous sections, a few large-degree nodes suffice for real-world graphs in order to deliver good results.

Example 4.3. *Let us consider the graph example depicted in Figure 4.1. For simplicity, we consider an unweighted graph (all edges have the same unit). If we set k to 2, then, it is clear that the top-k nodes are v_2 and v_4 with out-degree 7 for both nodes. As shown in Figure 4.2, the resulting partial 2-hop index $\mathcal{L}_G$ of the graph G would contain both v_2 and v_4 as centers and all the*

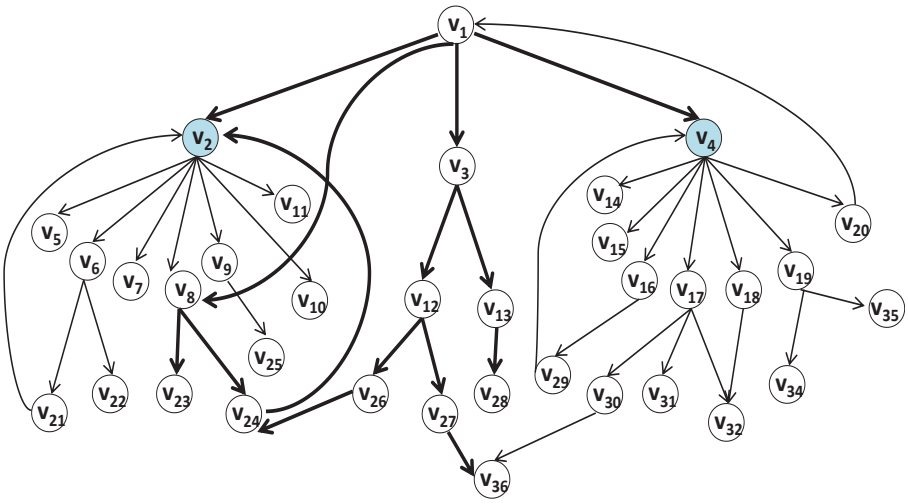


Figure 4.1.: Running Example

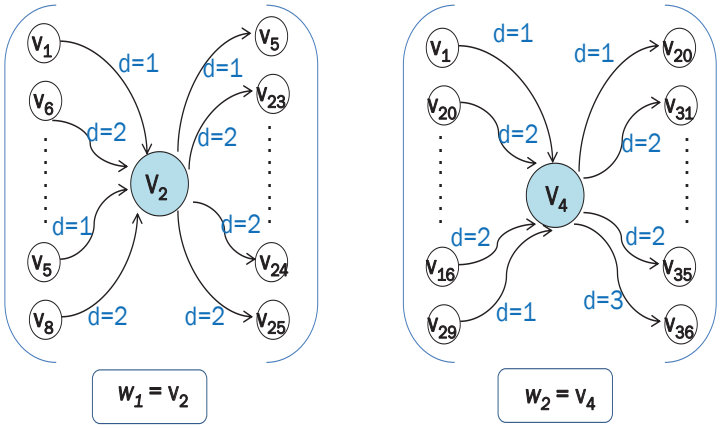


Figure 4.2.: Partial 2-hop Cover Example

paths going through these two nodes would be indexed. In order to get the distance of the shortest path between nodes v_1 and v_{36}, it is clear from Figure 4.1 that applying Dijkstra's algorithm results in traversing the whole graph before reaching the target node v_{36}. However using our algorithm, only thick edges are traversed.

4.4.2. Query Processing

In this section, we present our algorithm for computing the distance between two nodes u and v in the input graph. We remind that our index might not cover all node-pairs distances, therefore after querying the index we employ an efficient variant of the Dijkstra's algorithm where we do not traverse large out-degree nodes. See Figure 4.1 in Section 4.2 for an example illustrating the effectiveness of our approach. We proceed as follows. First, we check whether there is a node w such that $(w, d(u,w)) \in L_{out}(u)$ and $(w, d(w,v)) \in L_{in}(v)$

in $\mathcal{L}_G$. If this is the case we compute $d_{\mathcal{L}_G}(u,v)$ as:

$$d_{\mathcal{L}_G}(u,v) = \min_{\substack{w:(w,d(u,w))\in L_{out}(u)\wedge \\ (w,d(w,v))\in L_{in}(v)}} d(u,w) + d(w,v)$$

otherwise we set $d_{\mathcal{L}_G}(u,v)$ to a very large value. We then run an efficient variant of Dijkstra's algorithm where we limit our search space to nodes being at distance at most $d_{\mathcal{L}_G}(u,v)$ from u, while avoiding traversing the center nodes in $C_{\mathcal{L}_G}$ (as shortest paths traversing center nodes are already covered by our index). The latter one can be achieved by running Dijkstra's starting with all center nodes being marked as visited. We then return the minimum value between the distance computed according to our index and the one computed using our efficient variant of Dijkstra. To further improve the running time of our algorithm, we employ the Ferrari algorithm presented in [SABW13] or GRAIL [YCZ10] to decide quickly whether two nodes are not reachable. A pseudo-code of the algorithm for processing distance queries is shown in Algorithm 12. Next lemma shows the correctness of our algorithms.

Algorithm 12: Distance query processing

Input : A directed graph $G = (V, E)$, a partial 2-hop index $\mathcal{L}_G$, nodes u, v.
Output : The distance $d(u,v)$ between u and v or ∞ if they are not reachable.

1 Use Ferrari [SABW13] or GRAIL [YCZ10] to determine if v is reachable from u. If they are not reachable return ∞.;
2 Compute the distance $d_{\mathcal{L}_G}(u,v)$ between u and v in our index as follows. If there is a node w such that $(w, d(u,w)) \in L_{out}(u)$ and $(w, d(w,v)) \in L_{in}(v)$ then

$$d_{\mathcal{L}_G}(u,v) = \min_{\substack{w:(w,d(u,w))\in L_{out}(u)\wedge \\ (w,d(w,v))\in L_{in}(v)}} d(u,w) + d(w,v).$$

3 Otherwise $d_{\mathcal{L}_G}(u,v) \leftarrow \infty$;
4 Run Dijkstra(u,v) with maximum depth $d_{\mathcal{L}_G}(u,v)$ with all center nodes in $C_{\mathcal{L}_G}$ being marked as 'visited'. Denote such a distance with $d_D(u,v)$;
5 return $min(d_D(u,v), d_{\mathcal{L}_G}(u,v))$;

Lemma 4.4. *Let $\mathcal{L}_G$ be a partial 2-hop distance labeling computed by Algorithm 11. Let u, v be any two nodes in the input graph G and let $\hat{d}(u,v)$ be the distance between u and v computed by Algorithm 12. The following holds:*

$$d(u,v) = \hat{d}(u,v) \quad \forall u, v \in V(G).$$

Proof. If all shortest paths between u and v do not traverse any center node in $C_{\mathcal{L}_G}$ then $d_{\mathcal{L}_G}(u,v) = \infty$ and our efficient variant of Dijkstra in Algorithm 12 finds at least one such a path P. If there is at least one shortest path $\hat{P}$ between u and v traversing a center node w in $C_{\mathcal{L}_G}$ then u belongs to the set of predecessors of w while v belongs to the set of successors of v. Moreover, as Algorithm 11 computes the distance $d(u,w)$ between u and w as well as $d(w,v)$ and store them in our index we have that $l(\hat{P}) = d_{\mathcal{L}_G}(u,w) + d_{\mathcal{L}_G}(w,v)$. We

conclude the proof by recalling that Algorithm 12 always outputs the minimum between the $l(\hat{P})$ and $l(P)$. □

Example 4.5. *Let us consider the following distance query $d(v_1, v_{32})$. To answer this query, we first query our index $\mathcal{L}_G$ according to which $d_{\mathcal{L}_G}(v_1, v_{32}) = 3$. This value serves as an upper-bound on the distance between v_1 and the nodes to traverse using our algorithm. Then, starting from v_1, the graph is traversed up to distance 3. As you can see in Figure 4.1, only edges in bold are traversed up to distance 3 starting from v_1. Such traversal does not lead to node v_{32}. This means that the shortest path between v_1 and v_{32} has been covered in $\mathcal{L}_G$, and, $d(v_1, v_{32}) = 3$.*

4.4.3. Further optimization

In the following, we introduce some heuristics that can further speed up query processing.

4.4.3.1. Label Ordering

As we explained in the previous section, determining $d_{\mathcal{L}_G}(u, v)$ boils down to computing the intersection $L_{out}(u) \cap L_{in}(v)$, which requires a quadratic number of look-ups in the labels $L_{out}(u)$ and $L_{in}(v)$ (i.e., check for each element in the first label whether it is contained in the second one). For further optimization, we order the labels and perform a binary search for each element of the label having smaller cardinality. The benefit of this ordering consists in decreasing the time needed to query the index by avoiding useless look-ups over labels.

4.4.3.2. Topological Order Pruning

We enhance *EUQLID* with another criteria that allows additional pruning of the search space. We maintain for every node $v \in V$ its topological order number $\tau(v)$. While this simple variant of node labeling is obviously not sufficient to answer a distance query, a graph search procedure can benefit from the node labels: For a given query $d(u, v)$, the online search rooted at u can terminate the expansion of a branch of the graph whenever for the currently considered node x the following holds $\tau(x) \geqslant \tau(v)$.

Although we focus on point-to-point distance queries on directed graphs, our algorithm can be easily modified to deal with undirected graphs (which is a special case) and point-to-many distance queries.We postpone an extensive experimental evaluation to future work.

4.5. Experimental Evaluation

We evaluate our algorithms in terms of query processing time, indexing construction time as well as index size and we compare it against the *IS-Label* approach [AHCR13], which is to the best of our knowledge the most efficient approach for processing point-to-point distance queries [AHCR13] on large graphs. We also compare our query processing algorithm against the classical Dijkstra's algorithm. We implement two versions of EUQLID, a memory-based version M-EUQLID and a disk-based version D-EUQLID where our labeling index $\mathcal{L}_G$ is stored in main memory and on disk, respectively. We compare both

M-EUQLID and D-EUQLID against the memory-based version of *IS-Label* which gives faster query response time than the corresponding disk-based version.

4.5.1. D-EUQLID: Disk-based version

In addition to the memory-based version of *EUQLID* (denoted *M-EUQLID*), we devised a disk-based variant of our indexing scheme (denoted *D-EUQLID*), which can be applied when the index is too large to fit into main memory. In this paragraph, we give some implementation details about *D-EUQLID*. We store our index in relational database tables and run SQL queries against these tables to retrieve node labels. This implementation is based on MySQL 5.6.12, but could be easily carried over to other database platforms. Note that this approach automatically takes advantage from all the dependability and manageability benefits of modern database systems (indexing, query optimization, etc.). For storing the index, we need two tables `LIN` and `LOUT` that capture the L_{in} and L_{out} labels for each node $v \in V$. For a given distance query $d(u, v)$?, we can efficiently retrieve the labels $L_{out}(u)$ and $L_{in}(v)$ using two simple `SELECT` queries. Then, we compute the intersection between these two labels in main memory. Note that we could achieve this by means of a `JOIN` query on the labels of u and v. However, the first option gives best results when the number of centers is small enough which is the case in all our experiments.

4.5.2. Settings

We obtained the original source code of IS-Label from the authors and set the parameters as suggested in [AHCR13]. All algorithms have been implemented in C++ and compiled with the same compiler. All experiments were run on a machine equipped with 6 Intel Xeon CPUs at 2.93 GHz, 64 gigabytes of main memory with a 64-bit installation of Linux Ubuntu operating system(kernel 2.6.32). Since none of the algorithms are parallel all processes were in fact using one single CPU.

4.5.3. Methodology

We evaluate all algorithms in terms of:
Index construction time. For *EUQLID*, we set a k value (maximum number of centers in the index) so to keep a reasonable indexing time for each dataset while delivering fast response time, and ε to 1. As for *IS-LABEL* we set all parameters, such as hierarchy levels, depending on the size of the dataset at hand as suggested by the authors.
Query time. To assess query response time, we randomly generated $100K$ queries for small datasets and 2000 queries for the large ones. We then computed the average query response time.
Index size. We report the corresponding index size for each approach over all datasets.

4.5.4. Datasets

To validate our approach on real-world datasets, we considered a selection of benchmark datasets that were used in the literature over the last few years (see Table 4.2). A brief description of these datasets is the following: (1) *Wiki-vote* is a who-votes-on-whom directed graph which was extracted from Wikipedia. It contains all the Wikipedia voting

Datasets	#Nodes	#Edges	Density	Max out-deg.	Avg. dist.	Med. dist.	D.	% Reach. pairs	Source
Wiki-vote	7K	104K	14.57	893	4	4	9	23.83 (±1.19)	[les]
Cit-HepPh	35K	421K	12.20	411	12	11	41	40.49 (±1.49)	[les]
p2p	63K	148K	2.36	78	10	10	26	22.71 (±1.15)	[les]
Enron	70K	276K	4	1392	4.25	5	34.28	11.74 (±0.11)	[web]
Soc-Epinions	76K	509K	6.71	1801	5	6	11	45.95 (±1.39)	[les]
Soc-sign-slashdot	77K	517K	6.68	426	5	6	10	38.47 (±1.35)	[les]
Email-EuAll	265K	419K	1.58	930	5	5	10	13.51 (±0.93)	[les]

Table 4.2.: Benchmark Dataset characteristics

Datasets	#Nodes	#Edges	Density	Max out-deg.	Avg. dist.	Med. dist.	D.	% Reach. pairs	Source
Wiki-talk	2.4M	5M	2.10	100022	5	5	9	5.24 (±0.59)	[les]
LiveJournal	5.5M	80M	15	2469	6	6	7.36	78.62 (±0.78)	[web]
Hollywood 2009	1.14M	113.9M	100	11468	3.87	4	4.14	89.90 (±0.63)	[web]
Hollywood 2011	2.18M	229M	105	13107	4	4	5	76.56 (±0.75)	[web]
uk-2005	39.5M	936.4M	24	5213	15	18	23	64.30 (±0.62)	[web]

Table 4.3.: Large Dataset characteristics

data from its creation till January 2008. Nodes in the network represent users, and, there is a directed edge from node u to node v denotes that user u has voted on user v, (2) *Cit-HepPh* is a High Energy Physics paper citation network, (3) *p2p* is the graph of connections between hosts in the Gnutella network topology, (4) *Enron* is an an exchanging e-mail messages graph between some Enron employees, (5) *Soc-Epinions* is a who-trust-whom online social network of the Epinions.com website. A directed edges from a node u to a node v denotes that u trusts v, (6) *Soc-sign-slashdot* is a network where users can tag each other as friends or foes. There is a directed edge between two users if one of them has tagged the other, and, (7) *Email-EuAll* is an exchanging email network which was generated using email data from a large European research institution.

In addition to that, we performed experiments on much larger datasets which are, to the best of our knowledge, the largest real-world graphs publicly available. In the following, we briefly describe each of them, and, we summarize their characteristics in Table 4.3. (1) *Wiki-talk* is a directed graph where there is an edge from node u to node v if node u has edited at least one Wikipedia article created by node v, (2) *LiveJournal* is a social network where friend relationships are not necessarily symmetric (i.e., there is an edge from u to v if u considers v as friend), (3) *Hollywood 2009*, and, Hollywood 2011 are social networks of Hollywood actors where there is an edge between actor u and actor v if u declares that he appeared in a movie with v, and, (4) *uk-2005* is a directed graph, obtained from a 2005 crawl of the *.uk* web graph.

For a better understanding of the datasets we considered, we computed some statistics on the features that are most relevant for our algorithm. As shown in Tables 4.2 and 4.3, we report for each dataset the density, which is two times the average node degree as defined in Section 2.3 and the maximum out-degree. We also estimated the average distance (denoted *Avg. dist.*), the median distance (denoted *Med. dist.*), and, the diameter (the maximum shortest path distance denoted *D.*) of the graphs with a confidence of 0.95 based on [BKH01]. To estimate the percentage of reachable pairs in the graph, we used the Wilson score interval to determine an informative sample size [Wil27] with a confidence of 0.95.

4.5.5. Results

In the following, we present the results of our evaluation over all the above described datasets. We first describe our results for distance queries on a selection of graphs ranging from small to very large size. We provide the index construction time values in milliseconds for the small benchmark graphs and in seconds for large graphs . The index size and the query time are respectively given in KBytes and milliseconds for all datasets and $\mid V \mid_{vis}$ denotes the average number of visited nodes for the set of selected queries. Missing values are marked as '-' whenever a dataset could not be indexed due to long index construction time (we set the timeout to 48 hours).

Datasets	k	Index Size (KB)			Indexing Time (ms)		
		M-EUQLID	D-EUQLID	IS-LABEL	M-EUQLID	D-EUQLID	IS-LABEL
Wiki-vote	70	986.504	700	1300	1988	4531	3546
Cit-HepPh	60	5827.47	2300	8000	5804	21013	22409
p2p	100	9947.720	7350	4200	6316	47043	3010
Enron	95	7616.20	6200	7400	8495	24633	17000
Soc-Epinions	110	13745	8400	10500	8472	31839	15933
Soc-sign-slashdot	110	13170	8050	9700	8021	82812	6510
Email-EuAll	165	31327	20040	13000	26943	85016	108803

Table 4.4.: Index construction results on benchmark datasets

Datasets	M-EUQLID		D-EUQLID		IS-LABEL	Dijkstra	
	querying (ms)	$\mid V \mid_{vis}$	querying (ms)	$\mid V \mid_{vis}$	querying (ms)	querying (ms)	$\mid V \mid_{vis}$
Wiki-vote	0.043	3	0.73	3	0.5	0.378	275
Cit-HepPh	0.139	5	4.49	5	1.2	6.27	10375
p2p	0.21	10	1.4	10	0.3	9.71	6875
Enron	0.24	196	1.1	196	0.203	6.012	2222
Soc-Epinions	0.381	15	3.31	15	1.1	6.87	11074
Soc-sign-slashdot	0.275	12	4.97	12	4	9.7	31399
Email-EuAll	0.97	21	3.1	21	2.4	5.77	3452

Table 4.5.: Query processing on benchmark datasets (10K random queries)

Datasets	k	Index Size (KB)			Indexing Time (sec)		
		M-EUQLID	D-EUQLID	IS-LABEL	M-EUQLID	D-EUQLID	IS-LABEL
Wiki-talk	180	328029	200390	114000	141.5	662.595	28701
LiveJournal	200	1250801	1197000	1700000	4697	5581	3630
Hollywood 2009	210	282294.20	312180	1800000	3616.67	6318	21476
Hollywood 2011	210	506106.41	493000	3300000	3947.77	6525	46196
uk-2005	1110	8243634	8637000	-	8804	10673	-

Table 4.6.: Index construction results on web-scale datasets

4.5.5.1. Results over benchmark graphs

Table 4.4 and Figures 4.3 and 4.4 summarize the indexing performance of M-EUQLID, D-EUQLID and IS-LABEL over the small benchmark datasets. We can see that M-EUQLID outperform IS-label in terms of index construction time as well as index size for some of these datasets. Both our memory-based and disk-based versions of EUQLID outperform IS-label in terms of index construction time for the last dataset (i.e., Email-EuAll). This

Datasets	M-EUQLID		D-EUQLID		IS-LABEL	Dijkstra	
	querying (ms)	$\mid V \mid_{vis}$	querying (ms)	$\mid V \mid_{vis}$	querying (ms)	querying (ms)	$\mid V \mid_{vis}$
Wiki-talk	22.27	114	25	114	0.8	10	60197
LiveJournal	34.3	242	54	242	66	2360	1953712
Hollywood 2009	62.6	140	83.2	140	96	1315	459401
Hollywood 2011	82.91	128	98.4	128	74	1602	734014
uk-2005	174.11	1080	328	1080	-	7083	1415663

Table 4.7.: Query processing on web-scale datasets (2000 random queries)

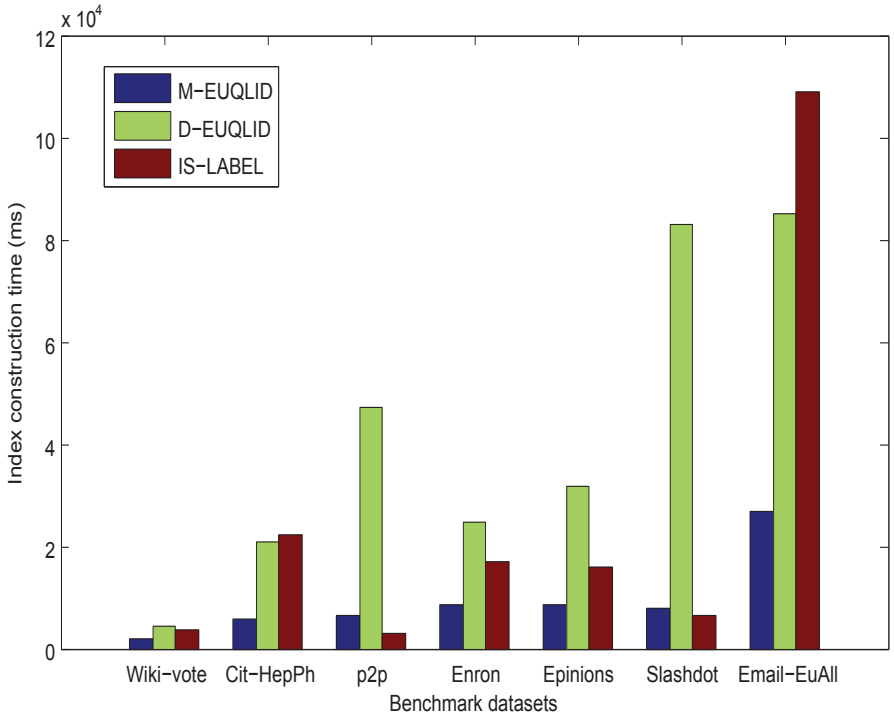


Figure 4.3.: Indexing time for benchmark datasets

might depend on the fact that *IS-LABEL* needs to perform a relatively expensive pre-processing step (i.e., computing a layered structure of vertex hierarchy of the input graph) before it starts assigning node labels, while *EUQLID* directly starts computing the labels of the nodes. Table 4.5 and Figure 4.5 shows query response time over the same datasets. We also include a comparison with the classical Dijkstra's algorithm that does not build any index. Table 4.5 shows that query response time for EUQLID is comparable to IS-LABEL for the small benchmark datasets with both algorithms delivering results within a few milliseconds. Moreover, both algorithms outperform Dijkstra's algorithm on these small datasets. We can already see from this table that the difference in terms of query response time between Dijkstra's algorithm and the other two approaches becomes wider as the size of the input graph increases. This will be even more apparent on larger datasets.

4.5.5.2. Evaluation over web-scale datasets

Tables 4.6 and 4.7 as well as Figures 4.6, 4.7, and, 4.8 summarize our results on very large graphs. These figures make the advantage of using our approach even more apparent.

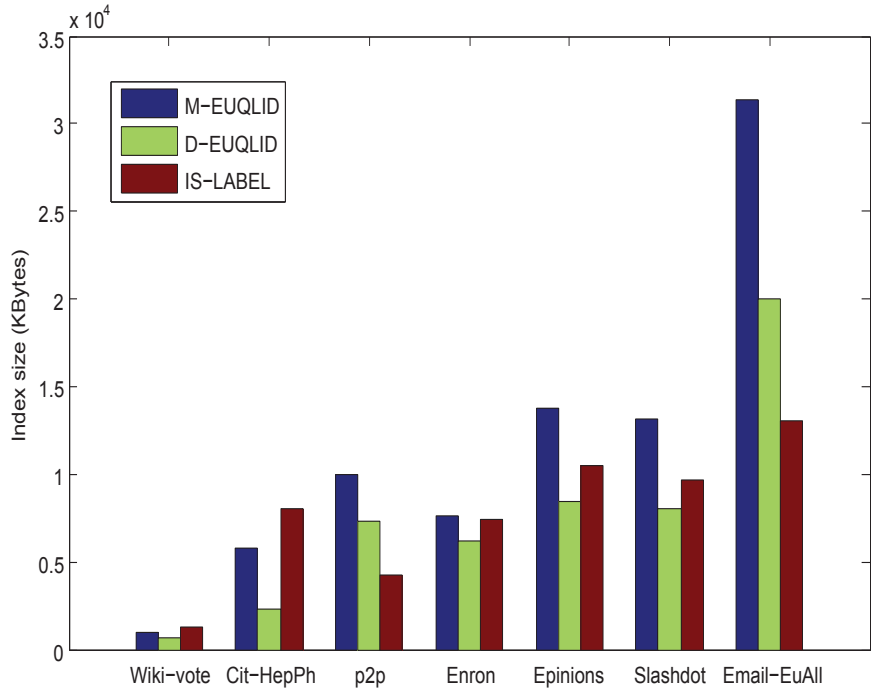


Figure 4.4.: Index size for benchmark datasets

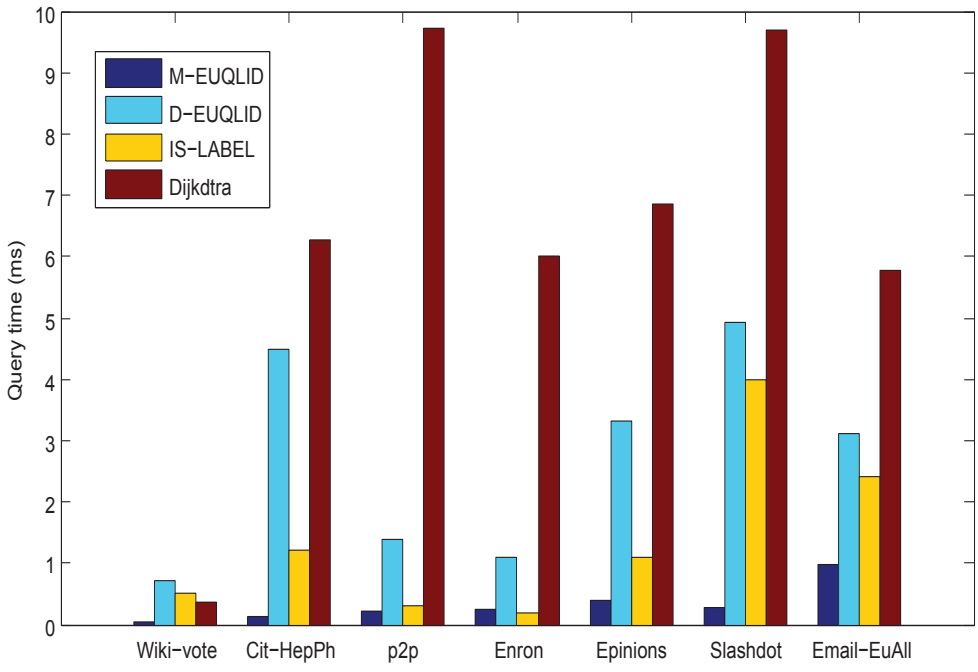


Figure 4.5.: Query processing time for benchmark datasets

Our indexing algorithm produces indices being significantly smaller than the indices produced by IS-LABEL while being much faster (Table 4.6) for the first four datasets.

For uk2005, IS-Label was not able to build an index after 48 hours, while our approach was able to index the largest dataset (with almost one billion edges) within a few hours. Table 4.7 confirms fast response time for our algorithm which is slightly better than the response time of IS-LABEL for the first four large datasets as IS-LABEL does necessarily not avoid traversing nodes with large out-degree which slows down the search process. In particular, the memory-based version of our algorithm always delivers results within 174*ms* while the disk-based version of our algorithm requires 328*ms* on the largest dataset with almost one billion edges. For uk, we were not able to make any comparison as IS-Label could not build any index within 48 hours. Both EUQLID and IS-Label significantly outperforms Dijkstra's algorithm with the difference between Dijkstra's response time and the other approaches being more apparent for largest graphs. The main reason for obtaining such good results lies on the fact that we were able to exploit one interesting property of real-world networks where a few nodes (hubs) connect all nodes through short paths (not necessarily shortest paths).

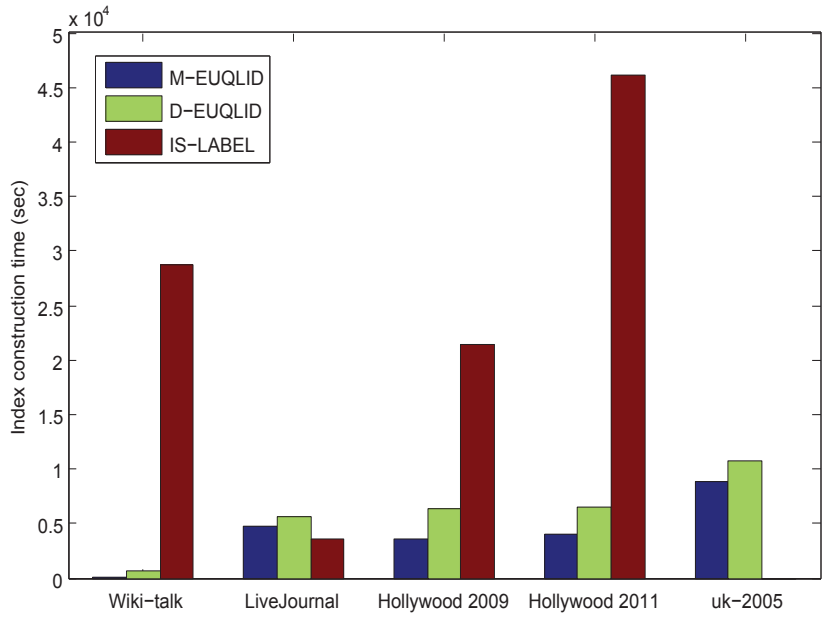


Figure 4.6.: Indexing time for large datasets

4.5.6. Effect on varying k

We studied the effect of varying k which is the maximum number of centers that are allowed to be included in our index. We have varied the values of k from 5 to 2000. We have chosen the values of k reported in Tables 4.5 and 4.7 so to keep a reasonable indexing time for each dataset while delivering fast response time. In general, the indexing time increases as k increases for all our datasets. We observed that increasing k helps improving query response time until k reaches the reported value for each dataset. After that there is no significant improvement in the query response time. This means that the marginal distance information of the other centers is very small, so that adding them to the index will not cover a large amount of "new" distance information.

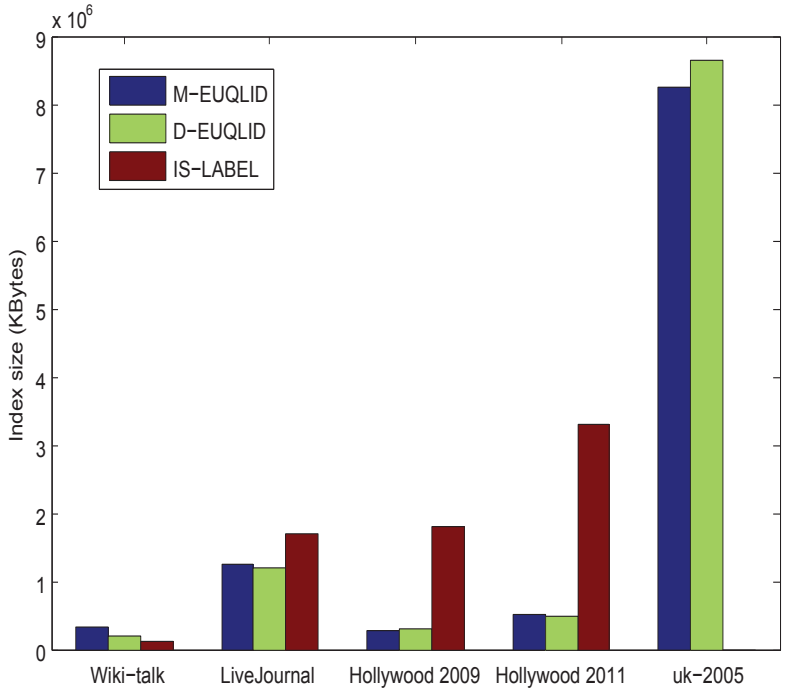


Figure 4.7.: Index size for large datasets

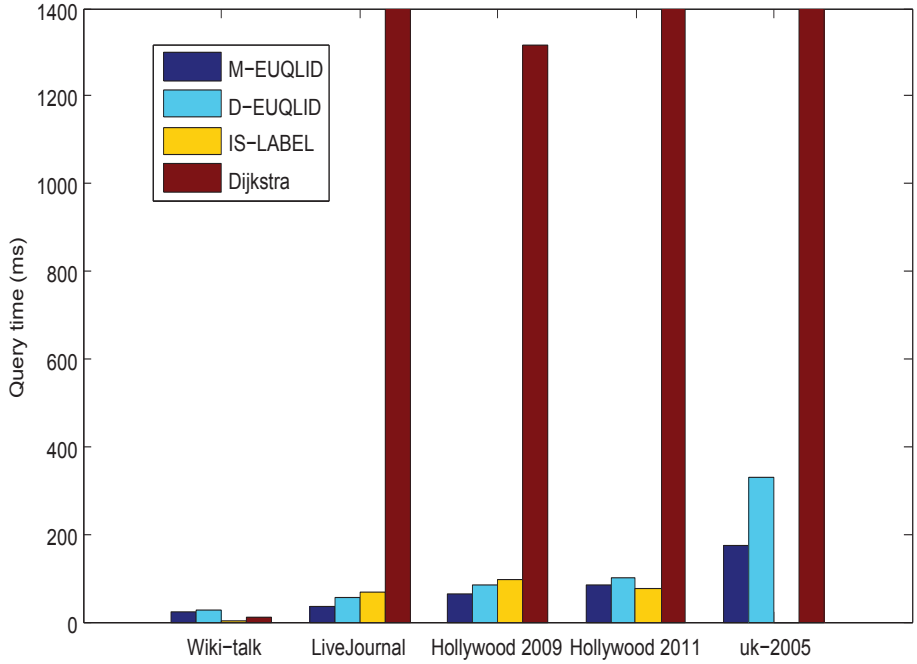


Figure 4.8.: Query processing time for large datasets

4.6. Conclusion

In this chapter, we presented EUQLID an approach for answering distance queries in large directed real-world graph. Our algorithm is based on a fast algorithm for a variant of the 2-hop cover where we enforce an additional constraint limiting the size of the index. We also exploit one key property of real-world graphs where a few nodes connect most nodes through short paths. Our extensive evaluation (for both a memory-based and a disk-based version of EUQLID) shows the effectiveness of our approach against state-of-the-art algorithms and that distance queries can be processed within a few hundred milliseconds on very large real-world graphs.

Chapter 5.

Access Control in Social Networks

5.1. Introduction

With the development of Web 2.0 technologies in the last few years, social networks (e.g., Facebook, LinkedIn, Flickr, Twitter, etc.) have become among the most successful services on the Web, used by a constantly increasing number of users. Actually, Facebook now reports over 900 million active users [facb] and Twitter has 200 million users [twi]. These so-called Online Social Networks (OSNs) are online communities whose main goal is to make available an information space, where each social network participant can publish and share information (e.g., personal data, photos, videos, opinions, contacts, etc.), as well as meet other people for a variety of purposes (e.g., business, entertainment, religion, dating, etc.).

The availability of this information obviously raises privacy and confidentiality issues. Users typically do not want to share all of their information with everyone. Consequently, OSNs must provide the right mechanisms in order to give users more control on the distribution of their resources which may be accessed by a community far wider than they can imagine.

It is clear that users should be provided with more flexible mechanisms to control access to their own information. As in the real world, OSN members will have in mind a specific audience for their resources. OSNs should then enable them to specify the desired audience and enforce their access policies. In this chapter, we propose a network-aware access control model for OSNs where access control rules are expressed as reachability constraints. These constraints express an encoding of the type of the path that should exist between the seeker of a resource and its owner. Thus, each user can specify the target audience for his resources. Access rules enforcement can be done on the fly when seekers try to access some shared resources. Our work is a new approach, which generalizes access constraints by taking into account the properties of the users, the indirect connections

between these users and enables expressing complex relationships (i.e, sequence of direct relationships of different types). The main idea is the specification of the target audience of each access rule in terms of a reachability constraint, which is expressed as a path expression over the social network graph. Thus, the enforcement of an access rule consists in the evaluation of a path, which can be computed on the fly when the resource is requested by the seeker.

This chapter is organized as follows. We first discuss existing work and explain its limitations in Section 5.2. In Section 5.3 and 5.4, we ,respectively, present some preliminary material which will be used through-out this chapter, and, highlight the requirements to design appropriate access control models for social networks. Section 5.5 describes the proposed access control model and how it can be enforced. In Section 5.6, we study the worst case running time of our privacy policy enforcement algorithm, and, show our experimental results in Section 5.7. Finally, in Section 5.8, we demonstrate `Primates`, which is a privacy management system for social networks implementing the proposed access control model. Section 5.9 concludes the chapter.

5.2. Related Work

Access control in social networks is a new research area. It has emerged with the growing popularity of OSNs, which have become an important part of our daily digital life. In this section, we give an insight about research work that has been done so far to mitigate privacy issues and tailor to user privacy needs. We classify existing work into different categories based on the way privacy policies are expressed.

Automatic Approaches. Several works proposed to automatically infer privacy policies in an implicit way based on the underlying graph structure and/or the already defined privacy policies. In this category, we can find approaches which propose automatic extraction of communities from social graphs as a way to simplify privacy preferences specification. Danezis [Dan09] proposed to classify users contacts into non-overlapping lists, so that contacts of the same list can have only access to information that is shared by their list members. Fang and LeFevre [FL10] proposed a privacy wizard that considers explicit user privacy preferences as well as automatically extracted communities to build a privacy preference model. When a user specifies a privacy policy for one of his contacts, then this policy is automatically applied to the rest of his contacts belonging to the same community. This privacy preference model can be automatically applied and adapted whenever the social network graph evolves. Xiao et al. [XAT12] proposed a model, which automatically recommends ad-hoc circles (i.e., lists of contacts) based on the history of circles that were already defined by the user. Additionally, Shehab et al. [SCT+10] proposed a supervised learning mechanism that generates access control policies based on user provided policy settings example, in a collaborative way. Squicciarini et al. [SSP09] considered an additional problem: co-ownership. They proposed an automated collective privacy management solution where data may have multiple owners, and, where owners might have different and possibly contradictory privacy preferences. This model uses a game-theoretical algorithm to control access to resources that are owned by more that one OSN member.

User-Guided Approaches. Unlike automatic approaches, user-guided approaches allow users to explicitly specify their privacy settings and control the audience of the

information that they share. In this category and more related to our work, there is the rule-based access control model proposed by Carminati et al. [CFP09]. Following this model, an access control policy is specified through constraints on the relationship type the requester of information must have in order to get access to the information that he requested for. This can can be specified as conditions on the type, the maximum depth and the minimum trust value that the required relationship must satisfy in order to obtain the access. Additionally, Akcora et al. [ACF12] proposed to associate an estimate risk level to social network users in order to provide them with a measure that shows how much is it risky to have interactions with other users, in terms of private information disclosure. Wang et al. [WZL+12] proposed *iSac*, an intimacy-based access control mechanism, where an OSN user can specify his access policy according to intimacy degrees with respect to other users. Intimacy degrees are computed based on atomic social activities between users. Only users whose intimacy degrees are within the range specified by the owner are allowed to access this owner's information.

Encryption-based Approaches. Some other research work has relied on cryptography techniques to protect user information and interactions on OSNs. For instance, the NOYB model [GTF08] encrypts personal information using a pseudo-random substitution technique which replaces a personal information with a pseudo-randomly selected information from a public dictionary. Another approach, called FlyByNight [LB09], presents a Facebook application that stores sensitive data in an encrypted form. Jahid et al. have recently proposed *EASiER* [JMB11], *DECENT* [JNM+12], and, *Cachet* [NJM+12] architectures for providing security and privacy guarantees, which leverage cryptographic techniques such as attribute-based encryption [SW05] to protect data confidentiality within social networks.

Anonymization Approaches. Selling OSN information is often a major source of revenue for service providers and publishing social graphs create opportunities for building new services and expend our understanding about social structures and their dynamics. However, selling or publishing such data without any further consideration for privacy may violate user rights for private information sharing in social networks. To alleviate this problem, several anonymization methods have been proposed to reduce the risk of a privacy breach on the provided social data, while allowing to analyze them and draw relevant conclusions. Zhou et al. [ZPL08] gives a survey of this field (i.e., anonymization for privacy preserving publishing of social network data) and its problems. Recently, Hay et al. [HMJ+08] proposed an approach that models aggregate network structure, then allows samples to be drawn from that model. Boldi et al. [BBGT12] also proposed another anonymization approach, which is based on injecting uncertainty in social graphs and providing the resulting uncertain graphs.

Limitations. While using automatic approaches users don't need to specify the desired privacy policy for some shared information, it is not clear to which extent the automatically inferred policy is reliable and accurate. As user contact networks grow constantly and the communities to which user contacts belong to maybe overlapping, it becomes difficult to make sure that such approaches do not grant access to some undesired users.

User-guided approaches that were so far proposed in the literature are not general enough to fit to user needs. For instance, the access control model proposed in [CFP09] does not allow specifying multiple relation types within a given authorization (e.g., one cannot grant access to the friends of his colleagues). Specifying constraints on relationship directions is also not supported (e.g., one cannot grant access to people that he considers

as a friend and not to those who consider him as a friend). Following this model, an OSN user cannot express the fact that he wants to grant access to contacts at a specific distance (i.e., only maximum distance is allowed).

While using cryptographic approaches, one can protect data from other users as well as the OSN provider, key management in such techniques remains a big issue, especially in large OSNs where scalability can become an obstacle.

The drawback of anonymization techniques is the fact that it is hard to prove to which extent these methods are secure, as opposed to explicit privacy policies specification or cryptographic operations. As graph information is partially hidden or obfuscated, other parts remain intact to be interesting for future use. This makes it impossible to predict which information maybe available to attackers with the complexity of the current OSNs.

5.3. Preliminaries

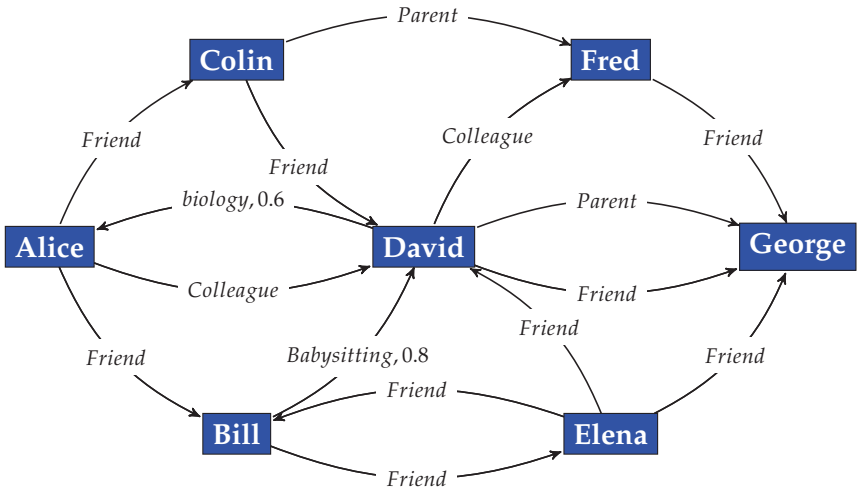


Figure 5.1.: An Online Social Network Subgraph

As defined by Boyd and Ellison in [BE07], an online social network is a web-based service that allows individuals to:

- create their own profiles,
- share connections with a list of other users,
- view and traverse their list of connections and those made by others within the service, and, see which information they shared.

As depicted in Figure 5.1, a social network is a dynamic structure made of nodes, which are connected to each other through various relations. The nodes and the edges of the graph denote, respectively, the social network users and the relationships that exist between them. Labels describe the relationship type associated to each edge, i.e., *Alice* considers *Bill* her friend, *Colin* considers *David* his friend, and so on. In this work, relationships are not supposed to be symmetric, i.e., if *Alice* considers *Bill* as a friend, that

does not mean that *Bill* considers *Alice* a friend too. Thus, we consider directed social network graphs, where each edge have an initial node and a terminal node.

Two types of relationships can be distinguished: *direct* and *indirect* relationships. A direct relationship involves only two nodes: the initial one and the terminal one. However, an indirect relationship has intermediary nodes, and consists of a finite number of direct relationships. For instance, *Alice* has a direct friend-typed relationship with *Bill*, and an indirect colleague-typed relationship with *Fred*. The *depth* of a given relationship corresponds to the number of direct relationships (of the same label) it is composed of.

In the real world, some interpersonal relationships are based on trust or reliability estimations. This can be denoted as weights on the corresponding relationships in the social network graph. In this case, the relationship type (for instance, babysitting) is called the utility of the trust, and the weight is the value, which is assigned to the trust relationship. As depicted in Figure 5.1, *Bill* trusts *David* for taking care of children up to 80%, and *David* trusts *Alice* up to 60% in the field of biology. When the trust relationship is direct, it corresponds to an explicit trust (i.e., given by the initial user). When it is an indirect relationship, its value has to be inferred based on available explicit trust values. Different trust propagation approaches are proposed in the literature [LLL+08, GKRT04, ACS09].

5.4. Access Control Requirements

The design and implementation of a suitable access control model for online social networks present a number of challenges. We consider that the following requirements are keys to developing such a model:

Dynamicity. In the real world, interpersonal relationships are varied, numerous, and changing over time. Consequently, as a first requirement, an access control model for OSNs should take into account relationships diversity and dynamics.
Suppose that *Bill* is authorized to access *Alice*'s holiday album because she considers him her friend. If *Alice* does not consider *Bill* her friend anymore, he should no longer be able to access her holiday album. In this case, *Alice* does not need to change access rules that she has set for her holiday album, she just has to update her relationship with *Bill* in the OSN.

Precision. Access control should allow targeting the audience of a given resource with the granularity that a user might need. Thus, in an OSN context, access control models should be able to take into account user properties (age, gender, location, status, hobbies, etc.), relationships between users in an extended sense (i.e., not limited to direct relationships), and trust measures when they can be inferred based on user input or their previous interactions.

In our example, *Alice* should be able to share her resources with her friends, her colleagues, her colleagues' friends, etc. *David* should be able to share his jokes with those who consider him as a friend (*Elena* and *Colin*), and he should be able to extend the audience to their friends (*George* and *Bill*, for *Elena*), and so on. Suppose now that *Elena* is looking for a baby-sitter for her kids, she may want to publish an advertisement and make it accessible to only some trusted baby-sitters. For instance, those she personally trusts for baby-sitting and those trusted by her friends (let us say with a trust level > 0.5). In this case, the advertisement could be accessible to *David*.

Scalability. Access control protocols have to be able to decide on the fly whether to allow or deny an access request sent by a user. This is important in the context of online social networks, where the number of active users can reach several hundreds of millions, and much more relationships between them. The response time in this case is a critical issue.

5.5. The Access Control Model

Devising an access control model implies the specification of both the access control policy, and the access control enforcement mechanism. The former corresponds to the desired rules according to high level requirements, and the latter denotes the access policy implementation.

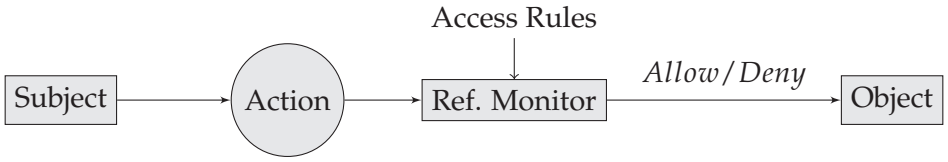


Figure 5.2.: Access control model

As shown in Figure 5.2, the fundamental components of our access control model are:

- The *Object* is the target resource, to which access may need to be controlled.
- The *Subject*, also called *Principal*, is a user who tries to get access to a particular resource.
- The *Action* is the operation that the *subject* wants to execute over the *object*.
- *AccessRules* specify the access control policy, i.e., user privacy preferences.
- The *ReferenceMonitor* is the component that implements user privacy preferences. It takes as input a request (an action sent by a subject) and a set of access rules according to which it will allow or deny access to the targeted *object*.

Access control policies are presented in section 5.5.1 and the reference monitor is described in section 5.5.2.

5.5.1. Access control policy

We propose a reachability-based access control model that enables targeting the audience of the shared resources in a network-aware manner. The intuition behind the design of this model comes from the observation that, in the real world, we generally conceive our privacy preferences based on relationships that bind us to each other.

In our model, an OSN is represented as a labeled directed graph, where nodes represent users and edges denote social relationships between users. User properties (age, gender, etc.) are expressed as attributes of the graph nodes. Privacy preferences are expressed by a set of access rules, each one being associated with a given resource (i.e, a shared resource)

Notation	Description
OSN	Online Social Network
Σ	the set of labels in a graph G (e.g., friend, colleague, etc.)
t	a trust computation function
u	an owner of a resource r within the OSN
r	a requester of a resource r within the OSN
AR	an access rule
ARS	the set of access rules specified for a resource r
P	a set of constraints on the paths connecting two nodes in an AR
p_j	a path pattern expressing constraints in terms of an edge label $l \in \Sigma$, direction dir and distance I

Table 5.1.: Frequently used notations II

and specifies, through a reachability constraint, the set of users who can access such a resource. Each reachability constraint is represented as a path expression over the OSN graph.

Definition 1. Online Social Network (OSN)
We formally represent an OSN as a directed graph:

$$G = (V, E)$$

where V is a set of nodes denoting social network users and E is a set of directed edges representing social relationships between them. A labeling function $l : E \rightarrow 2^{\Sigma}$ specifies the set of labels (where Σ is a set of labels such as friend, colleague, etc.) associated to each edge, while a function $t : E \rightarrow [0, 1]$ measures trust between users. In case a trust value is not specified, a default value (e.g, 0.5) can be used. Each node is associated with a set of pairs (attr,value) specifying a set of attributes and their values for the corresponding user.

Definition 2. Access Rule (AR)
An access rule expresses a set of constraints that should be met in order to access a given resource. Formally, an access rule is defined as a tuple:

$$AR = (u, r, P, C)$$

where u denotes the owner of a resource r. As it is very common and natural that access control policies of a given information in an OSN are expressed by their owners, we assume in our model that only the owner can specify access policies for his resources. C is a set of constraints on the attributes of the requester (such as location='Paris' or trust $\geqslant 0.8$) and $P = p_1, \ldots, p_k$ expresses a set of constraints on the paths connecting the requester to the owner of the resource; each p_j is defined as a triple $p_j = (l, dir, I)$ where l is a label in Σ, $I = (min, max)$ is a pair of integers specifying the minimum and maximum length of p_j and $dir \in \{\leftarrow, \rightarrow, \Leftrightarrow\}$ indicates the direction of p_j. Given a requester v for resource r, an access rule (u, r, P, C) is satisfied if all constraints C are satisfied and there is a path between u and v satisfying P; v is granted access to r if there is an access rule at least that is satisfied.

For instance $P = (\prime friend\prime, \rightarrow, (1, 2)), (\prime colleague\prime, \rightarrow, (1, 1))$ expresses the constraint that there must be a path between the owner and a requester that first traverses a path of length in $(1, 2)$ whose edges are labeled 'friend' and then an edge labeled 'colleague'.

Suppose that *Elena* wants to make her baby-sitting advertisement (identified as *ad*) accessible to her direct friends. She can specify an access rule as follows:

$$AR_1 = (Elena, ad, ('friend', \rightarrow, (1,1)), -)$$

If *Elena* wants to extend access to her indirect friends (i.e, the friends of her friends) the path that should be specified would be $('friend', \rightarrow, (1,2))$. Again, if she wants to extend it to the users that consider her as a friend, she should specify an other access rule which is the following:

$$AR_2 = (Elena, ad, ('friend', \leftarrow, (1,2)), -)$$

If *Elena* wants to change the access rules associated to her baby-sitting advertisement and make it accessible to the trustworthy (trust threshold = 0.8) baby-sitters of her friends within 2 hops then she should specify the following access rule:

$$AR_3 = (Elena, ad, \{('friend', \rightarrow, (1,2)), \\ ('babysitter', \rightarrow, (1,1))\}, [trust = 0.8])$$

Finally, the authorization can be limited to the baby-sitters living in Paris by adding an additional condition to the access rule as follows:

$$AR_4 = (Elena, ad, \{('friend', \rightarrow, (1,2)), \\ ('babysitter', \rightarrow, (1,1))\}, [location = Paris])$$

Note that the proposed model considers authorization access rules only. Thus, we avoid access authorization-denial conflicts.

5.5.2. Access control enforcement

The access control enforcement mechanism is performed by the *reference monitor*, which is a trusted software module that intercepts each access request submitted by a requester to access a resource, and, based on the specified access policy, determines whether access should be granted or denied to the requester. The decision module of the reference monitor is detailed in Algorithm 13.

Algorithm 13: Access Control Protocol

Precondition : A requester r wants to get access to a resource *res* of u.
Input : A requester r and a resource *res*.
Output : *Allow* or *Deny* access.

1 $ARS \leftarrow$ get the set of access rules of the requested resource *res*;
2 If ARS is empty then ARS $\leftarrow$ get the default access rule;
3 **foreach** $AR \in ARS$ **do**
  ▷ $AR = (u, r, P, C)$
4 If there is a path from u to r which satisfies P and r satisfies the trust and attribute constraints in C then return *Allow*;
5 If P is not satisfied then skip to next AR;
6 return *Deny*;

The enforcement of an access rule consists in evaluating the path P that is associated to it. The problem of evaluating these paths boils into a reachability problem in graph databases, which is well-known in the database community. Evaluating a reachability

query, in our case, consists in determining whether two nodes u and r (for instance, the owner and the requester) in the graph are connected through a path with constraints on labels and distance.

Suppose that a user r is requesting for a resource *res* where *res* is the resource identifier, and, u is a user owning such a resource. When r submits his/her access request to access the resource *res*, the system retrieves the set of access rules *ARS* associated to that resource (Line 1). If there are no access rules related to the requested resource, then, the system will apply the *default access rule* (Lines 2), which is defined by the user and applied whenever there are no access rules associated to the requested resource. This prevents the access control strategy from being too loose (by setting resources having no associated rules to public) or too restrictive (by setting resources having no associated rules to private).

Then, the reference monitor evaluates the set of retrieved access rules and stops, either when the requester satisfies one of these rules, or when all the rules were evaluated and the requester satisfies no rule (Lines 3-6). In the former case, the requester is authorized to access the resource that he asked for (Line 4). In the latter case, the requester is denied access because his profile is not consistent with the target audience (Line 6).

Algorithm 13 is an adapted version of the breadth-first-search (BFS) algorithm applied to the graph together with the specified constraints to reduce the search space. The evaluation of access rules is an iterative process. For each access rule, a path P (i.e., a sequence of subpaths p_j) is evaluated online using a breadth first search starting from u to check if r could be reached via P. If r cannot be reached from u with respect to the path P, then, the current access rule is not satisfied and the system goes directly to the next rule (Line 5). If r is reached during the search, we determine if r satisfies the set of attribute-constraints in C. The trust computation process between u and r is done at the same time, when the graph is explored. However, since our focus is on reachability, we implemented a simple trust propagation function and consider transitivity as the only way of propagation. The inferred trust value between two nodes u and r, connected through a path P, is computed by multiplying the explicit trust values associated with each edge in p. More sophisticated trust propagation functions can be found in the literature [LLL+08, GKRT04, ACS09].

5.6. Complexity Analysis

In this section, we focus on estimating the worst case running time of Algorithm 13, in order to measure its efficiency. The worst case occurs when there are no constraints neither on edge labels, nor on distances. More precisely, in such a case, we are called to discover the social graph without any constraints. Since we use a breadth-first search algorithm, exploring the network graph requires $(\mid V \mid + \mid E \mid)$ time complexity, where $\mid V \mid$ and $\mid E \mid$ denote, respectively, the OSN nodes and edges. After evaluating the first path p_j in P, the algorithm considers a new list of nodes satisfying p_j. In the worst case, this list will contain $\mid V \mid$ nodes. The same process is repeated $\mid P \mid$ times in order to evaluate all the paths $p_j \in P$. Consequently, the time complexity required to evaluate a path P is of the order of:

$$\mid P \mid \times \mid V \mid \times (\mid V \mid + \mid E \mid) \tag{5.1}$$

$\mid P \mid$ is the maximum number of paths p_j that P could contain. In the worst case, the evaluation of an access policy is iterated $\mid ARS \mid$ times, where $\mid ARS \mid$ is the maximum

number of access rules that may be associated to a resource. Thus, the required time complexity to evaluate the proposed algorithm is of the order of:

$$\mid ARS \mid \times \mid P \mid \times \mid V \mid \times(\mid V \mid + \mid E \mid) \tag{5.2}$$

Then, we can conclude that the problem for which we implemented Algorithm 13 can be solved in polynomial time depending on the number of access rules, access paths p_j and the social graph size.

Time costs given in the previous formulas are determined in the worst case. When users specify their access control rules, constraints they set on the type, direction and relationship depth levels, considerably reduce the part of the graph to be browsed to evaluate each access rule. Consequently, the cost of evaluating a path p_j is far lower then the worst case cost (i.e., $\mid V \mid + \mid E \mid$). However, the number of access rules $\mid ARS \mid$ and the number of paths $\mid P \mid$ have a clear impact on the response time of our algorithm.

5.7. Experiments

In this section, we perform experimental studies on a real social graph datasets to evaluate the performance of the proposed algorithm in terms of response time to the access requests. As shown in Table 5.2, we considered 5 datasets. The first 4 datasets are subgraphs of a sample of 984K unique users that represents the *groundtruth* of the Facebook OSN, i.e., a truly uniform sample of Facebook anonymized user IDs crawled by Gjoka et al. [GKBM10]. This Facebook dataset provides information about relations between users. It also provides the total number of friends each user has, his privacy settings, and his network membership. We implemented these sub-graphs in *Neo4j 1.2* [neo], which is becoming one of the foremost graph database systems. Instead of static and rigid tables, rows and columns, it manipulates a flexible graph network consisting of nodes, relationships and properties. Its high-speed traversal framework is able to traverse one million nodes per second [tra]. We have also considered a compressed version of a snapshot of the *liveJournal* graph consisting of 5M nodes and 80M edges, which is publicly available at [web]. The graph was compressed using the *WebGraph* framework [BRSV11] which provides algorithms for accessing compressed graphs without any decompressing. The compressed version of the *liveJournal* graph fits into main memory and can be accessed efficiently. User relationships have only a single type, which is *Friend*. For this reason, we artificially introduced other relationship types and associated them to edges on the *liveJournal* graph. The introduction of these types was done with respect to natural characteristics of human relations (e.g., a person can have on average 3 children and at most 2 close friends, etc.). We conducted our testing on a PC with a 2.34 GHz Intel processor, and 2 GB memory running Windows 7. We associated access rules to randomly selected user information.

As reported in Table 5.2, we have considered subgraphs consisting of a number of nodes ranging from 8K to 5M. We measured the response time of access requests on different-sized graphs. For each graph, we tested a large number of access rules and varied the distances in access paths p_j from 1 to the graph diameter. We fixed the owner and the requester in a targeted manner, i.e. two nodes connected by a path length corresponding to the desired path depth, and we measured for each test the average response time.

Datasets	| V |	| E |	Source
subgraph 1	8K	8K	[GKBM10]
subgraph 2	88K	88K	[GKBM10]
subgraph 3	310K	309K	[GKBM10]
subgraph 4	984K	9M	[GKBM10]
liveJournal	5M	80M	[web]

Table 5.2.: Sample Datasets

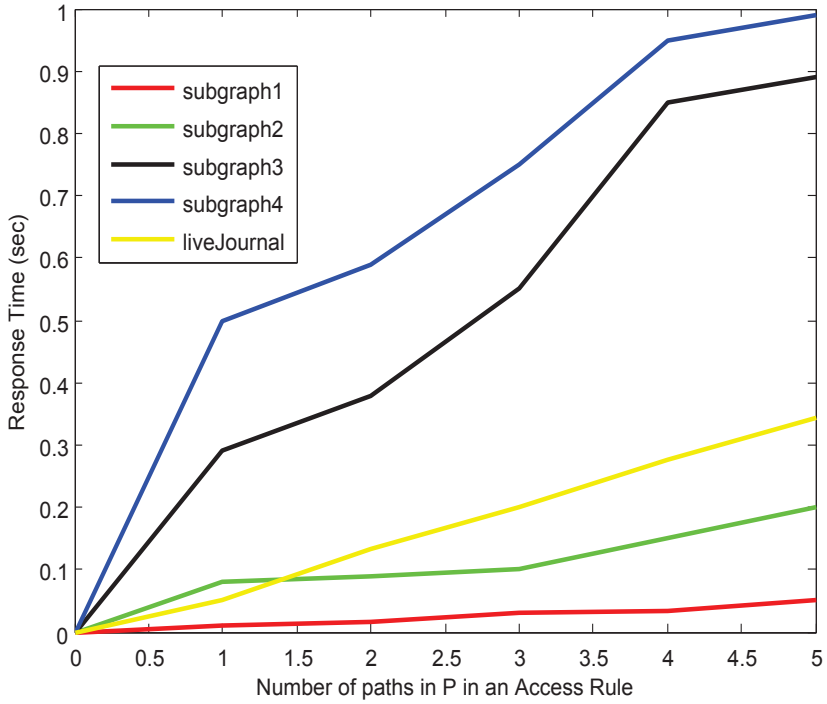


Figure 5.3.: Reference monitor performance depending on $\mid P \mid$

According to the obtained results (see Figure 5.3, the response time of the reference monitor depends on the social graph size (i.e., number of nodes and edges within the graph): it ranges from 0.001 sec for a graph consisting of 8K nodes to 1 sec for a graph consisting of 5M nodes. We recall that the size is not referred to the order of the whole social network graph, rather, given a relationship type, it represents the number of nodes having at least a relationship of such type. When a path depth gets larger, the time required to traverse the social graph and get the query result increases. As shown in our complexity study, in the previous section, the response time depends on the number of access rules to evaluate and the depth of the access paths. This was confirmed by our experiments. Experimental results have also shown that the number of nodes in the social graph as well as the number of path $p_j \in P$ (i.e., $\mid P \mid$) has also a clear impact on the performance of the reference monitor.

Note that the obtained response time on the *liveJournal* dataset was better then smaller

datasets. The reason behind this is the fact that it was implemented using the *WebGraph* framework instead of *Neo4j* as it is the case for the rest of the datasets.

5.8. `Primates`: A Privacy Management System for OSNs

In this section, we present `Primates` a privacy management system for social networks. `Primates` allows users to specify access control rules for their resources and enforces access control over all shared resources. The set of users who are allowed to access a given resource is defined by a set of constraints on the paths connecting the owner of a resource to its requester in the social graph. We demonstrate the accuracy of our access control model and the scalability of our system.

5.8.1. Global architecture

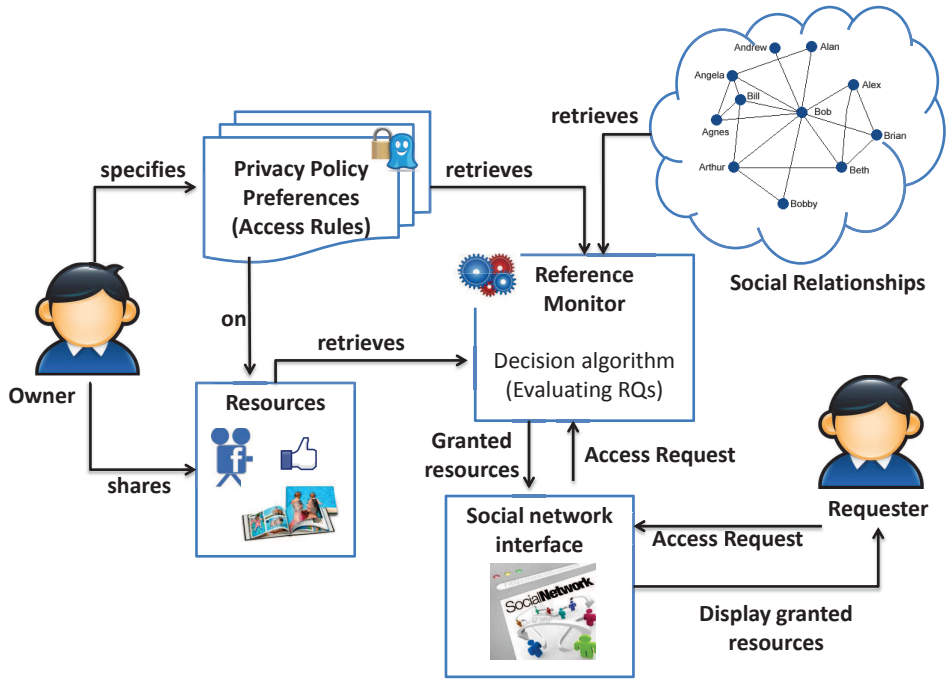


Figure 5.4.: System Architecture

The global architecture of our system is depicted in Figure 5.4. Each user can share multiple resources. Resources are information (photos, videos, comments, etc.) to which access may need to be controlled. A user (the owner) can express his privacy policy preferences for his resources by specifying one or several access rules to them. The requester, also called subject, is a user who is trying to access some resources of another user (the owner). The subject can send a request to access resources via the social network interface. This request is sent to the reference monitor, which is the component that implements user privacy preferences. It takes as input the access request and based on the access rules that are associated to the requested resource and social connections in

the social network graph, it will authorize rendering only the resources for which the requester is part of its authorized profiles.

5.8.2. Features

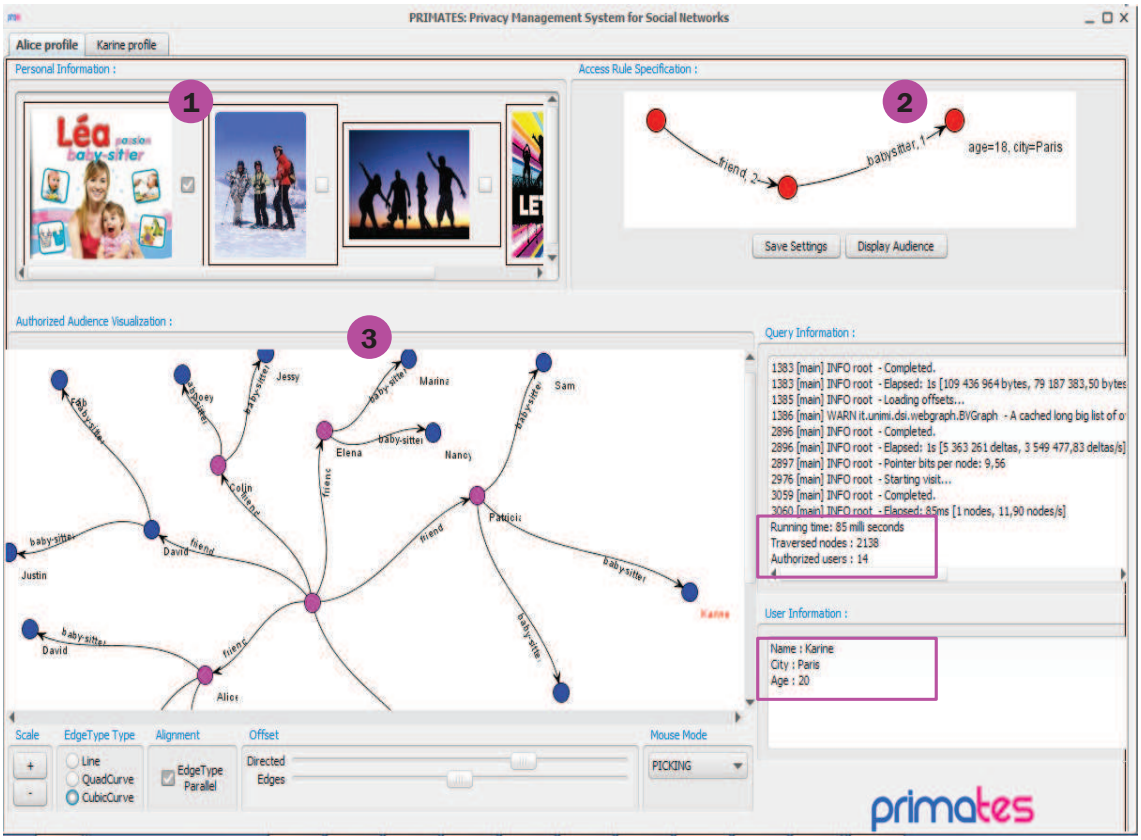


Figure 5.5.: User interface for specifying an access rule

Primates shows two contributions of our work. The first one is the design of an access control model that allows users to specify more sophisticated privacy policies which fit their privacy needs. The second one is the enforcement of this model in such a way that allows users to intuitively specify their privacy settings and efficiently visualize the set of users that are allowed to get access to their information.

As shown in Figure 5.5, using Primates, OSN users can select some information and express the desired privacy settings for it either from a set of access rules predefined by the system or by expressing new access rules in user-friendly graphical way. They can express constraints on edge label and distance, and, node properties. The motivation behind providing users with a graphical interface to specify access rules is that a parser can convert user actions into path patterns (i.e. access rules) that would be evaluated as reachability queries behind the scenes. Such suitably comprehensive front-end would minimize the burden on the user to remember the details of defining access rules.

Once access rules are specified, users will have the possibility to browse and visualize the graph of authorized users according to the specified privacy preferences. They can then

navigate this graph and see explicitly who are the authorized users and can eventually refine access rules based on that. Displayed users on the authorized audience graph can be clicked on to to display their profile information such as the name, age, city, etc.

Users can also see how each user in the social network sees only information to which he is authorized to get access. They can see the profile (all shared personal information) of a user as an information owner on one side and try to get access to that profile from the perspective of another user on the other side. They will clearly see that not all the information in the owner's profile is displayed, but only a subset of that information to which he/she is allowed to access.

An example scenario run on `Primates` is available in the following url `url_video`.

5.9. Conclusion

In this chapter, we presented a network-aware access control model for online social networks that enables a fine-grained description of privacy policies. User privacy preferences are specified in terms of reachability constraints, combined with user properties and trust considerations. Reachability constraints are expressed as paths in the social network graph. Thus, an access rule enforcement consists in the evaluation of a path, which can be made on the fly when the resource is requested by a seeker. Our experimental studies confirmed this intuition, and showed that the reference monitor of our access control system can decide on the fly whether a seeker is part of the audience of a given resource or not.

Chapter 6.

Conclusions and Research perspectives

The first main goal of this thesis is to devise an efficient algorithm to answer distance queries in large directed graphs. The second goal is to devise an access control model to help social network users having more control on the information that they share. The most important questions that we studied along this thesis are the following:

1. Can we efficiently answer distance queries in large directed and dense graphs?
2. Are there any properties of real graphs that could be used to devise efficient algorithms for answering distance queries?
3. What are the requirements that an access control model for social networks should take into account to fit to user needs?
4. Is it possible to enforce an access control model which considers these requirements and which is easy to use by social network users?

In this chapter, we summarize the findings and contributions that were detailed along the chapters of this thesis by providing answers to the above questions. After that, we bring forward some directions for future research.

6.1. Put It All Together

The first contribution of the dissertation is *EUQLID* which is an efficient indexing scheme for computing distance queries. As described in Chapter 4, *EUQLID* takes advantage from an interesting property that real-world graphs exhibit. We proposed two variants of *EUQLID*: a main memory and disk-based variants (*M-EUQLID* and *D-EUQLID* respectively).

The former variant can be used when the index can fit into main memory. The latter one should be used when there is not enough space to store the index into main memory. Our experimental studies has shown that *EUQLID* outperforms existing approaches, and, that distance queries can be processed within hundreds of milliseconds on the largest real-world directed graphs publicly available.

The second contribution is a fine-grained access control model described in Chapter 5. This model allows users to explicitly specify their privacy policy as they may think about it in real scenarios (based on the nature of relations with the others). Based on the user specified preferences, the decision of granting access to other users is made on real-time.

The third contribution is *Primates* (see Chapter 5), which is a privacy management system that demonstrates the accuracy and feasibility of the proposed model and allows users to specify their privacy preferences in a user-friendly way.

6.2. Research Perspectives

To conclude this thesis, we would like to highlight some of the most promising research directions stemming from the research conducted for the two main problems studied in this thesis (the reachability problem and privacy management in OSNs). We classify these directions into: (i) the ones that we are already addressing, (ii) those that, we believe, they require effort, but, should be tractable without any intrinsic difficulties, and (iii) those that involve intense research.

Let us start with work in progress.

- **Disk-resident graphs.** A useful extension of *EUQLID* is to develop I/O-efficient algorithms to index graphs that cannot fit in main memory. Some graphs are disk-resident and they don't fit into main memory because of their large size. This requires algorithms to access the disk whenever they need information about the input graph (e.g, list of neighbors of a given node, etc.). However, a large number of disk accesses may drastically affect the performance of algorithms when the graph is very large. Methods developed in [CKCC12, WC12, CC12] may be applied to achieve this task. More clearly, we can adopt the Block nested loop join algorithm as described in [CKCC12].

The following problems seem reasonably solvable, possibly with some effort:

- **Label-constraint reachability.** In order to handle reachability queries with constraints on labels and distance, we can partition an input labeled graph G into many subgraphs according to edge labels (i.e., each subgraph contains edges of the same label only). Then, we apply *EUQLID* on each of the subgraphs separately. For instance, let us consider the following label-constraint reachability query $u \rightsquigarrow l_1, d_1 \backslash l_2, d_2 \rightsquigarrow v$ which seeks to find out whether there is a path between u and v consisting of d_1 edges labeled l_1 followed by d_2 edges labeled l_2. To answer this query, we can use *EUQLID* to compute the distance between u and all nodes in the subgraph related to label l_1, and, store a set of nodes $S_1 = \{x \mid d(u, x) = d_1\}$. Then, we retrieve the set of nodes S_2 within the subgraph of label l_2 such that $S_2 = \{y \mid d(y, v) = d_2\}$. If the intersection between S_1 and S_2 is not empty then u can reach v with respect to the input reachability query, otherwise it is not.

Note that in typical real life graphs, the total number of distinct edge labels is very small compared to the total number of edges in the graph. For instance, the UniProt RDF graph of 22M edges has only 91 distinct edge labels [ACZ12]. This implies a relatively small number of subgraphs to consider. Moreover, the distribution of these edge labels is not uniform meaning that a large number of edges have few distinct labels. In the UniProt dataset, the edge label "rdf:type" appears on around 5M edges, about 10 edge labels appear on 1M edges each, and, about 20 edge labels occupy around 100K to 200M edges each. This skewed label distribution can help to prune the potentially large search space of edges within some subgraphs which results in smaller sets of nodes as input sets of nodes for which we need to compute the intersection.

- **Adopting the parallel set cover algorithm.** We can adopt the algorithm proposed by Berger et al. [BRS94] for designing an efficient parallel version of the greedy algorithm for Set Cover and integrate it to *EUQLID*.

- **Scaling *Primates*.** In *Primates*, we need to compute the distance between two nodes in order to decide whether to grant access to a requester or not. In the current implementation of *Primates*, this is done at the same time when traversing the graphs. In order to scale *Primates*, we would like to integrate *EUQLID* to it so to compute distance between nodes (when needed) in a much more efficient way.

- **Point-to-many distance queries.** A point-to-many query takes a given a source node as input and returns the distance of the shortest path between this node and the rest of the nodes on the graph. *EUQLID* can be adapted to handle this kind of queries by simply computing the distance between the given node and the rest of the nodes in the graph.

Finally, let us present some open issues that we believe to be important to extend the two addressed problem, and for which we do not know of existing solutions.

- **2-hop index update and maintenance.** Instead of rebuilding the labels in response to each single update in the graph (nodes/edges creation and/or deletion), it is desirable to devise an efficient 2-hop label maintenance algorithm. This problem was studied by Bramandia et al. [BCN08]. One direction to deal with index maintenance in *EUQLID* is to adapt the solution proposed in [BCN08].

- **Managing privacy policy conflicts.** OSN users share their information having a specific audience in mind with whom they would like to share. In real-life scenarios, they could also have specific unwanted audience for some information they want to share. Allowing users to specify the desired and unwanted audience at the same time may create authorization conflicts when some contacts belong to both audiences. To deal with this issue and manage conflicts, some game-theoretical algorithms may need to be applied as proposed in [SSP09].

- **Automatic inference of access policies.** One of the advantages of the access control model that we proposed is the fact that it allows users to explicitly specify their own privacy settings for each type of information that they would like to share within a social network. This user-guided approach could be combined with an automatic

one so to propose to users some inferred privacy policies based on the ones that they have already specified. This problem is not trivial and should probably be attacked using some natural language processing techniques combined with some machine learning algorithms.

The above list of open or partially solved problems has no pretension to be exhaustive.

Appendix A.

Résumé en français*

Les graphes sont omniprésents en informatique. En effet, ils sont largement utilisés pour modéliser des données du monde réel dans divers contextes, citons comme exemples les réseaux sociaux, la bioinformatique, Internet, les cartes routières et le réseau des aéroports. Les noeuds représentent généralement des objets du monde réels et les arcs désignent les relations entre ces différents objets, par exemple, les utilisateurs et leurs relations personnelles dans les réseaux sociaux, les protéines et leurs interactions dans les réseaux biologiques, les routeurs et leurs connections sur Internet, les villes et leurs connexions dans les cartes routières, les aéroports et les vols entre eux dans les réseaux aériens. De nos jours, les graphes deviennent de plus en plus larges et augmentent très rapidement en taille. Par exemple, le réseau social Facebook est constitué d'un grand nombre d'utilisateurs avec leurs relations d'amitié. Le nombre d'utilisateurs Facebook a augmenté de 50 millions en Septembre 2007 à plus de 1.15 milliard d'utilisateurs actifs chaque mois en 2013 [facb, faca].

L'interrogation des graphes est devenu une tâche très importante dans plusieurs applications. Par exemple, dans un réseau social, les utilisateurs s'intéressent éventuellement à déterminer leurs degré de proximité à d'autres utilisateurs (i.e., distance entre les utilisateurs). De plus, la nouvelle fonctionnalité de parcours de graphe de Facebook (Ang. *Facebook Graph Search*) (pour plus de détails sur cette fonctionnalité, voir Section A.1.2) permettant de parcourir le graphe de Facebook pour répondre à des requêtes formulées en langage naturel, peut être aussi considérée comme une forme d'interrogation des graphes. Considérons les exemples de requêtes suivants pouvant être spécifiés par les utilisateurs : "les Bars visités par mes amis qui vivent à Paris, France" ou "les Photos de mes amis pris à Hawaii". Dans un réseau routier, les utilisateurs ont besoin de connaître la distance séparant deux villes données. En bioinformatique, on a généralement besoin de calculer le nombre d'interactions nécessaires pour transformer une protéine en une autre. Pour résoudre ces problèmes, une solution possible consiste à stocker le graphe dans une base de données relationnelle, puis, l'interroger en utilisant des requêtes SQL. Cependant, cette solution n'est pas toujours efficace (voir Section 2.5) et plus particulièrement sur les grands graphes. D'autres alternatives NoSQL (ex : neo4j [neo]) ont été également proposées pour le stockage et l'interrogation des données graphe. Toutefois, une solution viable et satisfaisante reste manquante (voir Section 2.5). Le problème d'interrogation des graphes apparaît même dans Facebook, où l'outil de parcours du graphe est parfois incapable

* This appendix is a translation, in French, of Chapters 1 and 6; it does not contain any additional content, and may safely be skipped. An English-to-French lexicon of technical terms is also provided at the end of this chapter.
L'annexe A est une traduction en français de l'introduction, des chapitres 1 et 6. Elle inclut un lexique anglais-français des termes et expressions techniques utilisés dans cette thèse.

de fournir des résultats exacts. Par conséquence, le besoin de développer des systèmes performants et efficaces pour interroger les graphes devient de plus en plus urgent.

Etant donné qu'il n'est pas trivial de trouver une solution pour l'interrogation des grands graphes aléatoires, nous concevons une technique efficace d'interrogation des graphes tout en exploitant quelques propriétés intéressantes présentes dans les grands graphes réels. En particulier, nous nous focalisons sur les requêtes de calcul de distance dans ces graphes. Nous proposons également un modèle de contrôle d'accès dans les réseaux sociaux. Ce modèle peut être considéré comme une application du problème de calcul de distance dans les graphes où il est nécessaire d'appliquer et d'évaluer les contraintes d'accès formalisées par le modèle proposé.

Dans la suite de cette annexe, nous présentons les problèmes traités dans cette thèse dans la section 1.1 et nous donnons quelques exemples de motivation. Dans la section 1.2, nous détaillons la liste des contributions, et décrivons la structure du manuscrit dans la Section 1.3.

A.1. Contexte et Problématiques

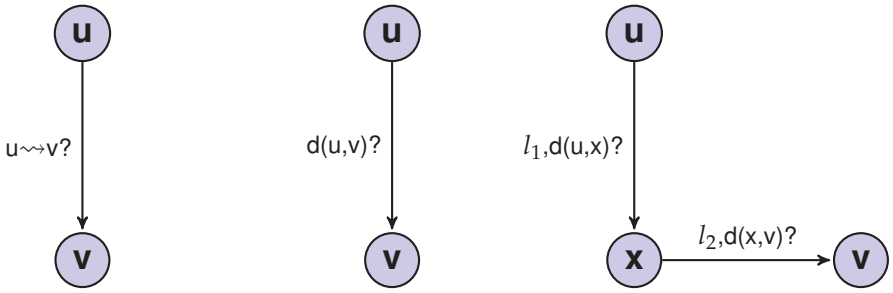
Dans cette section, nous discutons le problème général d'accessibilité dans les graphes, puis, nous procédons à décrire le problème de sécurité dans les réseau sociaux.

A.1.1. Passage à l'échelle des requêtes d'accessibilité et de calcul de distance dans les grands graphes

Etant donné deux noeuds u et v dans un graphe G, une requête d'accessibilité consiste à répondre à la question suivante : *v est-il accessible à partir de u dans G ?* Nous distinguons trois principales catégories de requêtes d'accessibilité en fonction des domaines d'application : (i) *Les requêtes d'accessibilité simple* déterminent si deux noeuds donnés sont connectés dans le graphe sans la prise en compte de contraintes sur le chemin liant ces deux derniers. Un exemple de requête d'accessibilité simple est illustré dans la figure A.1(a), où $u \rightsquigarrow v$ signifie que u peut atteindre v. (ii) *Les requêtes de calcul de distance* sont plus spécifiques que les requêtes d'accessibilité simple; elles déterminent non seulement si deux noeuds donnés sont connectés dans le graphe, mais elles donnent aussi la longueur du plus court chemin liant ces deux noeuds là. Un exemple de requête de calcul de distance est décrit dans la figure A.1(b), où $d(u, v)$ désigne la distance entre u et v. Il existe deux types de requêtes de calcul de distance : requête point à point (Ang. p2p query) dont le but est de déterminer la distance entre deux noeuds donnés, et, les requêtes qui cherchent à déterminer la distance entre un noeud source et les reste des noeuds dans le graphe (Ang. Single Source Shortest Path query). Et, (iii) *Les requêtes d'accessibilité sous contraintes* sont encore plus spécifiques et considèrent des contraintes sur les étiquettes ainsi que leur ordre d'apparition sur le chemin liant deux noeuds, sur la distance, l'orientation des arcs, etc. Figure A.1(c) montre un exemple de ce type de requêtes, qui cherche à déterminer s'il existe un chemin entre u et v composé de $d(u, x)$ arc(s) étiqueté(s) l_1 suivi par $d(x, v)$ arc(s) étiqueté(s) l_2.

Dans ce qui suit, nous mettons en évidence quelques domaines d'applications où les requêtes d'accessibilité sont nécessaires :

Les réseaux sociaux. Les réseaux sociaux sont souvent modélisés sous forme de graphes,



(a) Requête d'accessibilité simple (b) Requête de calcul de distance (c) Requête d'accessibilité sous contraintes

Figure A.1.: Les différents types de requêtes d'accessibilité

où les noeuds et les arcs représentent, respectivement, les utilisateurs et les relations entre eux. Les étiquettes sur les arcs désignent les différents types de relations entre les utilisateurs (ex : ami, collègue, ...). Plusieurs requêtes dans les réseaux sociaux cherchent à savoir comment deux noeuds donnés u et v sont connectés. Ces requêtes cherchent généralement à déterminer s'il existe un chemin allant de u à v ayant des étiquettes de certain type ou respectant une séquence prédéfinie de types d'étiquette. Par exemple, pour savoir si u est l'ami d'un collègue de v, il faut vérifier s'il y a un chemin allant de u à v composé d'un arc étiqueté *ami* suivi d'un arc étiqueté $collgue$. Comme expliqué dans la Section A.1.2, l'obtention de ce type d'information constitue un moyen qui aide les utilisateurs des réseaux sociaux à avoir plus de contrôle sur les informations qu'ils partagent tout en leur permettant de spécifier le public autorisé à voir ces informations [AD11, DAS12]. L'outil de parcours de graphes de Facebook [fbG] (voir Section A.1.2) peut aussi être considéré comme une application qui nécessite le calcul des distance entre les noeuds, comme par exemple, les utilisateurs peuvent avoir besoin de chercher les amis de leurs amis ou leurs amis pour des opportunités de travail (i.e., les amis à une distance 2 ou 3 d'un utilisateur donné). De plus, afin de pouvoir mesurer le degré de proximité entre deux utilisateurs dans un réseau social, nous aurons besoin de calculer la distance entre ces deux derniers. Considérons comme exemple la requête suivante : "Quel est le point commun entre la chancelière allemande Angela Merkel, le mathématicien Richard Courant, le gagnant du prix de Turing Jim Gray et Dalai Lama ? ". Dire que ces quatre personnes ont obtenu une thèse de doctorat d'une université allemande nécessite la vérification de l'existence de chemins entre chaque personne et les diplômes obtenus par les autres [KRS+09]. Le search ranking dans les réseaux sociaux [VFD+07], modélisé comme une fonction qui dépend des distances entre les utilisateurs dans un réseau d'amitié est une autre application où le calcul des requêtes de distance est nécessaire.

La bioinformatique. Les données biologiques tels que les réseaux d'interactions entre proteines et les réseaux métaboliques (réactions chimiques du métabolisme) peuvent être modélisés comme des graphes étiquetés, où les noeuds désignent les entités cellulaires (ex : protéine, gènes, ...). Un arc entre deux entités désigne une interaction chimique transformant une entité donnée en une autre. Les étiquettes sur les arcs désignent les enzymes responsables de la transformation. L'une des opérations les plus basiques consiste à déterminer si une entité peut être transformée en une autre sous certaines contraintes. Ces contraintes exigent l'existence d'un ensemble d'enzymes dans un ordre prédéfini et

une distance donnée entre deux entités. Plus généralement, une requête d'accessibilité en bioinformatique peut être formulée comme suit : *Y a-t-il une voie moléculaire entre deux entités cellulaires composée d'un certain nombre d'interactions avec la présence d'un ensemble d'étiquettes (enzymes) respectant un ordre bien défini ?* Ici encore, le problème peut être réduit au problème de répondre aux requêtes d'accessibilité sous contraintes. Dans cet exemple, les contraintes sont définies sur l'ordre des étiquettes et la distance du chemin liant deux noeuds. En biologie, les scientifiques ont besoin de calculer le chemin le plus court entre des paires de noeuds dans les réseaux d'interactions protéiques afin d'identifier les gènes lié à un cancer du côlon [LHL+12]. Les requêtes d'accessibilité sont aussi nécessaires pour le calcul du voisinage k-hop pour comparer les réseaux biomoléculaires et étudier leurs propriétés [ADH+08].

Le Web Sémantique. Le format RDF pour la représentation des données du Web sémantique, est devenu un format commun pour construire de large collections de données. Trouver le chemin le plus court entre deux noeuds dans un graphe RDF est une opération fondamentale permettant d'explorer les relations complexes entre les entités. Malgré son expressivité, une requête RDF standard, appelée requête SPARQL, ne permet pas d'explorer la nature des relations entre objets RDF (i.e., vérifier si un objet X peut atteindre un autre objet Y dans un graphe RDF). Par exemple, trouver tous les chercheur nés en France dans une base de connaissances peut être traduit par plusieurs requêtes d'accessibilité permettant de vérifier s'il existe un chemin entre un chercheur donné est l'entité France. En plus, les requêtes de calcul de distance sont utilisées pour calculer le degré de proximité entre les entités dans une base de connaissances [KRS+09].

Les approches classiques permettant l'évaluation des chemins les plus court soit elles prennent beaucoup de temps pour fournir la réponse quand la taille du graphe est assez grande (cas des parcours en largeur et en profondeur, voir Section 2.4), soit elles nécessitent un pré-calcul conséquent et beaucoup trop d'espace mémoire (cas de la fermeture transitive). Le défi qui se présente est de trouver un bon compromis en terme de temps et d'espace pour pouvoir répondre aux demandes d'accès des utilisateurs en un temps raisonnable.

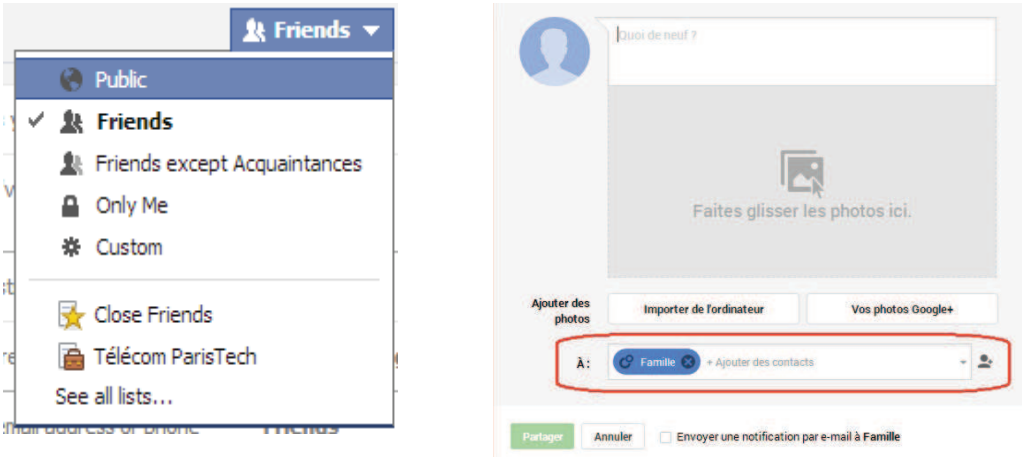
Dans cette thèse, nous nous intéressons au problème de calcul de distance entre les noeuds dans les grands graphes orientés qui est plus spécifique que le problème d'accessibilité simple, d'une part, et d'autre part, une solution à ce problème peut être utilisée ultérieurement pour résoudre le problème d'accessibilité sous contraintes comme expliqué dans la Section 6.2.

L'algorithme classique de Dijkstra (voir Section 2.4.3) ne parvient pas à répondre efficacement aux requêtes de distance dans les grands graphes. Pour cela, des techniques plus sophistiquées on été proposées. L'une des approches les plus répandues pour le calcul de distance entre les noeuds d'un graphe est la fameuse approche de couverture à deux sauts (Ang. 2-hop cover approach) [CHKZ02] (voir Section 2.6), qui donne des garanties sur la taille de l'index ainsi que sur le temps de réponse. Toutefois, il n'y a aucun algorithme efficace qui permet de calculer la couverture à deux sauts (ang. 2-hop cover). Malgré que quelques variantes de cette approches ont été proposées ces dernières années [STW04, CYL+08, CY09], une solution plus efficace est nécessaire, étant donné qu'il est impossible d'utiliser cette approche pour indxer de très grands graphes comme les graphes réels. En fait, tout algorithme d'indexation pour le calcul des distances dans un graphe doit faire face au problème de calcul d'une représentation compacte de toutes les informations sur la distance dans le graphe initial. Ces informations sur la distance

peuvent augmenter de manière quadratique en fonction du nombre de noeuds et rendent la construction des index sur les grands graphes extrêmement coûteuses.

A.1.2. Sécurité dans les réseaux sociaux

Dans un réseau social en ligne, chaque utilisateur peut partager des informations et du contenu multimédia (ex : informations personnelles, photos, vidéos, contacts, etc.) avec d'autres utilisateurs du réseau. Il peut également organiser différents types d'événements pour des raisons professionnelles, de loisir, de religion, etc. Tandis que ceci crée des opportunités, il relèvent des problèmes majeurs de sécurité, comme les utilisateurs peuvent souvent avoir accès à des informations personnelles, et parfois confidentielles sur d'autres utilisateurs. La disponibilité de telles informations soulève des problèmes de sécurité et de confidentialité. Par exemple, beaucoup de recruteurs cherchent leurs candidats sur les réseaux sociaux avant de les recruter [Wor09]. Sur un marché d'emploi compétitif, les informations que les gens partagent (ex : points de vue politique, statuts, images drôles, etc.) peuvent avoir des conséquences non-souhaitables. Les informations privées disponibles sur les réseaux sociaux peuvent mettre en danger les chances des candidats pour être accepter pour une offre d'emploi, même bien avant d'avoir l'occasion de décrocher un entretien. Une enquête fournie par Microsoft a montré que 79% des recruteurs et managers aux Etats-Unis ont consulté les informations publiées en ligne sur les réseaux sociaux et les blogs pour filtrer les candidats, et 70% ont rejeté des candidats en se basant sur des informations qu'ils ont trouvées [cnn]. D'autres exemples de dangers potentiels peuvent être les suivants : vol d'identité, harcèlement sexuel, etc.



(a) (b)

Figure A.2.: Outils de spécification des politiques d'accès sur Facebook and Google+

La plupart des réseaux sociaux fournissent des outils de contrôle d'accès très basiques, par exemple, un utilisateur peut spécifier si une information doit être publique, privée (personne ne peut la voir) ou accessible uniquement par les amis directs. Pour illustrer, nous décrivons l'outil de gestion de la confidentialité de Facebook comme il est un des réseaux sociaux les plus utilisés ces dernières années d'une part, et d'autre part, il est parmi le top 5 des applications de partage de photos sur Internet [top]. Comme le montre la Figure A.2(a), Facebook permet deux options extrêmes de contrôle d'accès : (i) la première

permet de partager les informations avec tous les utilisateurs du réseau social (option *publique*), et (ii) la deuxième permet une politique restrictive qui limite beaucoup le partage de l'information (i.e., option *moi uniquement*). La première option met donc en danger la vie privée des utilisateurs, tandis que la deuxième se contredit avec les finalités ultimes des réseaux sociaux qui sont la communication et le partage des informations. Facebook permet aussi aux utilisateurs de classer leurs contacts dans des listes différentes, puis, spécifier les paramètres de confidentialité pour chaque liste (c-à-d., spécifier pour chaque liste les informations que ses membres sont autorisés à voir). Ceci oblige les utilisateurs à affecter manuellement leurs contacts à des listes. Etant donné que le nombre moyen d'amis d'un utilisateur Facebook est de 144 [faca], cette tâche peut être fastidieuse et peut prendre beaucoup de temps. De plus, les utilisateurs peuvent être amenés à spécifier un très grands nombre de listes pour s'adapter à leur besoins qui sont souvent complexes et varient selon l'information à partager. Comme illustré dans la Figure A.2(b), Google+ souffre aussi du même problème, où les listes de contacts sont désignées par des cercles.

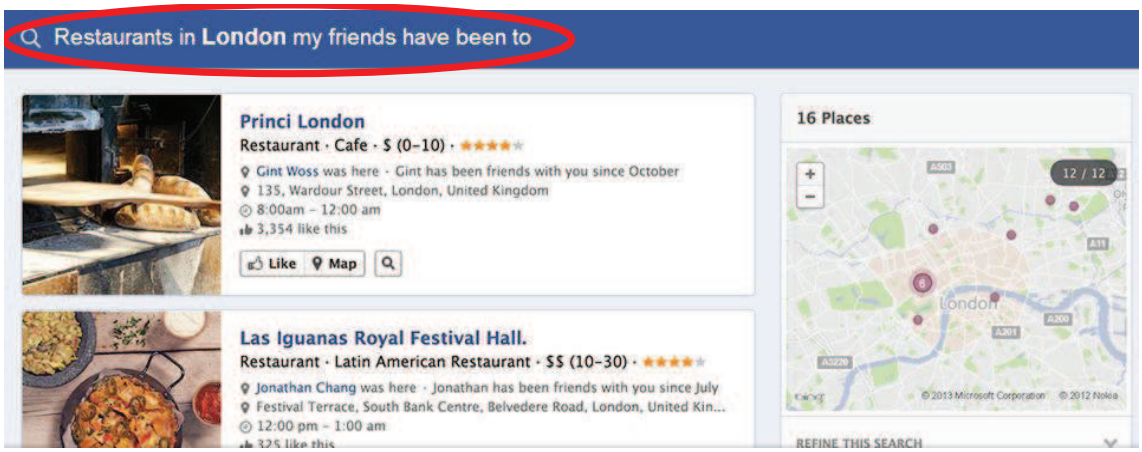


Figure A.3.: Outil de parcours de graphes de Facebook

Par ailleurs, l'outil de parcours de graphe de Facebook [fbG] peut être défini comme un moteur de recherche sémantique qui a été introduit par Facebook en Mars 2013. Cette nouvelle fonctionnalité permet aux utilisateurs de saisir des requêtes en langage naturel pour chercher leurs amis et les amis de leurs amis ayant un ou plusieurs centre(s) d'intéret commun(s). Ce moteur utilise les données stockées dans le graphe social de Facebook (constitué de plus d'un milliard d'utilisateurs), ainsi que les données du web pour fournir des résultats de recherche spécifiques. Par exemple, si un utilisateur compte visiter Londres et voudrait avoir la liste des restaurants londoniens visités par ses amis, il pourra saisir une requête pour obtenir de telles informations comme le montre la Figure A.3. Pour faire des rencontres, un utilisateur peut spécifier la requête suivante "Homme célibataires d'origine parisienne et vivant à San Francisco, Californie". Un tel outil peu être utilisé pour découvrir des informations potentiellement embarrassantes (ex : les entreprises recrutant des personnes qui aiment le racisme) ou pour des intérêts illégaux (ex : Résidents chinois qui aiment le groupe Falun Gong censuré par la Chine [prib]). De plus, cet outil applique les mêmes paramètres de confidentialité pour le partage des informations (c-à-d., les utilisateurs ne peuvent accéder qu'aux informations qu'ils sont

déjà autorisés à voir) [pria]. Ceci pose les mêmes problèmes de confidentialité mentionnés précédemment dans cette section.

Etant donné que l'ensemble des relations sociales existantes dans les réseaux sociaux est très riche et diverse, avec des liens familiaux ainsi que la possibilité de distinguer entre les connaissances et les amis proches devenant de plus en plus courante, le besoin de fournir des politiques de contrôle d'accès plus sophistiqués devient de plus en plus urgent. Par exemple, un utilisateur voudrait "*inviter les enfants de ses collègues pour l'anniversaire de son enfant*" ou "*partager ses photos portant une tenue amusante avec les amis et les amis de ses amis et non pas à ses collègues*". Pour illustrer, nous considérons l'exemple de la Figure A.4 qui montrent les différents types de relations sociales qu'Alice entretient avec ses contacts. Nous considérons également les scénarios suivants :

- **Scenario 1.** Supposons qu'Alice voudrait partager une photo qu'elle a prise dans un after-work avec ses collègues et quelques uns de leurs amis. En même temps, elle ne veut pas que le reste de ses contacts voient cette photos. Elle veut donc la partager uniquement avec ses collègues et les amis de ses collègues. Les utilisateurs autorisés à accéder à cette photos seront donc Karine, Colin, Julie, et Bill.

- **Scenario 2.** Supposons qu'Alice voudrait organiser une fête surprise pour son enfant, et elle voudrait partager une invitation en ligne avec les enfants de ses collègues qui habitent à paris uniquement. Selon l'exemple de la Figure A.4, le public autorisé serait donc uniquement Manon.

Les scénarios décrits précédemment mettent en évidence des exemples de besoins que les utilisateurs ne peuvent pas spécifier en utilisant les outils de gestion de confidentialité dans les réseaux sociaux existants. Par conséquent, nous constatons qu'il y a un besoin urgent de concevoir un modèle de contrôle d'accès pour permettre aux utilisateurs de spécifier leurs préférences de confidentialité comme ils y auraient penser dans les scénarios de la vie réelle (ayant un public préci en tête), et d'avoir plus de contrôle sur la propagation de leurs informations dans les réseaux sociaux tout en évitant les publics non-souhaités à accéder à certaines informations.

Le contrôle d'accès dans les réseaux sociaux, comme décrit précédemment, peut être considéré comme un domaine d'application du problème général d'accessibilité dans les graphes. En effet, pour évaluer les paramètres de confidentialité spécifiés par les utilisateurs, nous aurons besoin de répondre efficacement aux requêtes d'accessibilité dans les grands graphes. Pour plus de détails sur le problème d'accessibilité dans les graphes, voir la section A.1.1.

A.2. Contributions

Dans cette section, nous présentons les contributions de cette thèse comme suit :

A.2.1. *EUQLID* : un schéma d'indexation efficace pour le calcul des distances dans les graphes

Pour résoudre le problème de répondre efficacement aux requêtes de calcul de distance dans les graphes, nous proposons *EUQLID* comme une technique d'indexation efficace.

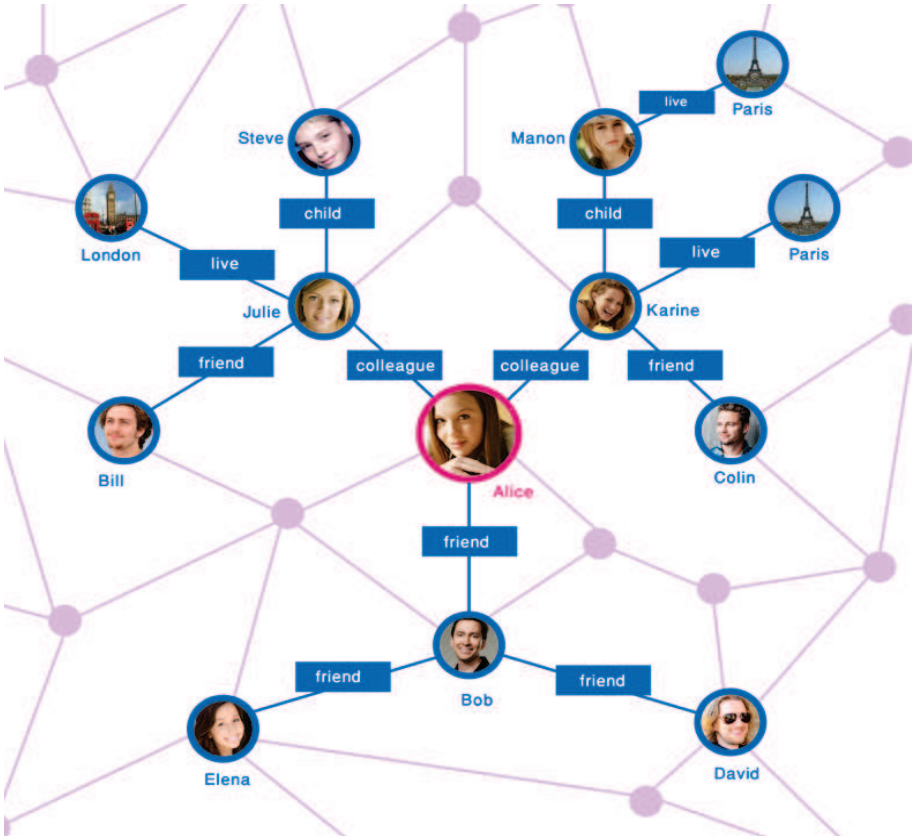


Figure A.4.: Exemple of user relations in an OSN

Nous définissons une variante de la couverture 2-hop et nous appliquons une contrainte additionnelle pour limiter la taille de l'index tout en assurant une couverture partielle des informations concernant les distances entre les noeuds du graphe. Les informations de distance manquantes (non couvertes) peuvent être déterminées de manière efficaces au moment de traiter la requête en utilisant une variante rapide de l'algorithme de Dijkstra sur un espace de recherche soigneusement élagué. Notre but durant la phase d'indexation consiste, donc, à stocker le maximum possible d'information sur la distance tout en respectant la contrainte sur la taille de l'index. Nous montrons que cette variante de la couverture 2-hop admet an algorithme 0.63-approché (ang. $0.63-$approximation algorithm), qui a inspiré notre algorithme d'indexation efficace.

Pour choisir les informations de distance qui doivent être stockées dans note index, nous exploitons une propriété intéressante qu'on a découverte après avoir effectuer des expériences sur des graphes réels. Cette propriété dit qu'un ensemble de noeuds relativement petit connecte le reste des noeuds du graphe à travers un chemin relativement court (voir Section 3.4). Les noeuds faisant partie de cet ensemble sont généralement les noeuds ayant les plus hauts degrés dans le graphe. Cette propriété peut être également observée dans notre quotidien, comme pour aller d'une petite ville à une autre il faut généralement passer par une grande ville, les correspondances dans les vols assez longs se font à travers les aéroports internationaux, et les chercheurs sont souvent connectés à travers les chercheurs les plus connus dans leurs domaines dans un graphe modélisant les

citations. Notre idée est de choisir de manière prudente un ensemble de noeuds ayant des degrés élevés et stocker toutes les informations de distance concernant les chemins qui passent par ces noeuds là. Comme les chemins traversant de tels noeuds ne sont pas forcément les chemins les plus courts entre deux noeuds donnés, nous avons besoin de faire appel à l'algorithme de Dijkstra afin de trouver la longueur exacte du chemin le plus court en temps réel. Cependant, l'espace de recherche de l'algorithme de Dijkstra peut être drastiquement élagué en évitant la traversée des noeuds ayant des degrés élevés (car les informations de distance couvertes par ces noeuds là existe déjà dans l'index) et en utilisant les distances stockés dans l'index comme limite supérieure sur la longueur des chemins à traverser par l'algorithme de Dijkstra.

Nous résumons les contributions concernant le calcul des distance dans les grands graphes comme suit :

- Nous définissons une nouvelle variante de la couverture 2-hop tout en appliquant une contrainte qui limite la taille de l'index et améliorant le temps de réponse;
- Nous développons un algorithme 0.63-approché pour ce problème;
- Notre principale contribution est *EUQLID*, qui est un algorithme efficace pour l'indexation et le traitement des requêtes de calcul de distance dans les grands graphes. Notre algorithme est basé sur une variante efficace de l'algorithme d'approximation précédemment discuté. Nous avons apporté une étude expérimentale approfondie tout en comparant notre algorithme aux algorithmes existants dans la littérature. Cette étude montre que notre approche est meilleure en temps de calcul et d'indexation d'une part, et d'autre part, que les requêtes de distance peuvent être traitées en quelques centaines de millisecondes sur le plus grands graphes réel disponible.

Ce travail a été soumis et il est encore sous révision.

A.2.2. Un modèle de contrôle d'accès dans les réseau sociaux basé sur l'accessibilité

Afin de résoudre le problème de confidentialité dans les réseau sociaux (décrit dans la Section A.1.2), nous proposons un modèle de contrôle d'accès à grain fin permettant aux utilisateurs de contrôler la distribution de leurs informations dans les réseaux sociaux. Selon ce modèle, les utilisateurs peuvent explicitement spécifier leurs paramètres de confidentialité en fonction de leurs besoins réels. Notre modèle permet aux utilisateurs, ayant un public visé dans leurs têtes, d'associer à chaque information le public approprié. Le public visé peut être spécifié en se basant sur les relations sociales entre le propriétaire de l'information et ses contacts. Un utilisateur donné pourra accéder à des informations, si et seulement si, il existe un chemin satisfaisant un patron bien défini entre lui et le publieur de cette information. Le patron exprime des contraintes sur les types de relation (ex : ami, collègue, etc.), l'orientation des arcs, la distance, la confiance, et les attributs (ex : habite à Paris, âgé plus de 20 ans, etc.). La décision d'autoriser l'accès ou non à des utilisateurs est faite en temps réel en se basant sur la politique d'accès spécifiée par les utilisateurs pour chacune de leurs informations. Nous avons étudié la complexité en temps de réponse su protocole de contrôle d'accès, et effectué des expériences pour évaluer les performances de

ce protocole sur des réseaux sociaux réels. Cette étude montre la faisabilité et la pratique du modèle de contrôle d'accès proposé.

Le modèle de contrôle d'accès proposé a été présenté dans DBSocial 2011, ainsi que dans le PhD symposium de la conférence ICDT/EDBT 2012.

A.2.3. *Primates* : un système de gestion de la confidentialité dans les réseaux sociaux

Nous avons implanté un système de gestion de la confidentialité dans les réseaux sociaux que nous avons appelé *Primates*. Ce système applique le modèle de contrôle d'accès qu'on a proposé, et permet aux utilisateurs de sélectionner les informations personnelles souhaitées et leur assigner les paramètres de confidentialité qu'ils désirent. Le processus de spécification est très pratique et convivial. Il est effectué à travers une interface graphique spécialement conçue pour faciliter cette tâche aux utilisateurs. *Primates* permet aussi la prévisualisation du public visé sous forme de graphe, et donne la main aux utilisateurs de changer leur paramètres au cas où ils se redent compte qu'il y a des utilisateurs non-désirés dans le public visé.

Primates a été présenté et démontré à CIKM 2012.

A.3. Organisation du Manuscrit

Ce manuscrit est organisée comme suit.

- Dans le **chapitre 2**, nous introduisons les concepts et algorithmes utilisés tout au long de ce manuscrit. Tout d'abord, nous introduisons les expressions booléennes comme elles sont utilisées pour spécifier les paramètres de sécurité. Nous présentons également des définitions de base dans la théorie des graphes, ainsi que quelques algorithmes de parcours de graphes. Ensuite, nous introduisons le fameux algorithme 2-hop, et décrivons les algorithmes de couverture d'ensemble (ang. *Set Cover*) et de couverture maximale (ang. *Max Cover*). Puis, nous expliquons un algorithme efficace pour calculer la couverture d'ensemble pour de grands jeux de données. Enfin, nous décrivons une technique d'échantillonnage pour estimer le nombre d'éléments respectant une propriété donnée dans un ensemble d'éléments qui est très grands.

- Dans le **chapitre 3**, nous introduisons, tout d'abord, deux modèles connus de graphes de terrain (ang. complex networks) : graphes aléatoires et graphes scale-free. Ensuite, nous décrivons certaines propriétés importantes des graphes du monde réel. Puis, nous présentons des expériences que nous avons effectuées pour analyser les chemins les plus courts dans les graphes réels, et étudions l'impact de quelques propriétés sur l'amélioration l'algorithme du 2-hop décrit dans le Chapitre 2.

- Dans le **chapitre 4**, nous présentons l'état de l'art sur les travaux connexes aux problématiques de calcul des requêtes d'accessibilité dans les grands graphes, ensuite, nous définissons, d'une manière formelle, le problème de calcul des requêtes de distance, et présentons quelques notations et définitions. Puis, nous décrivons les algorithmes d'indexation et de requêtage. Enfin, nous reportons une étude expérimentale approfondie de notre approche tout en la comparant avec les approches existantes.

- Dans le **chapitre 5**, nous présentons l'état de l'art sur les travaux connexes à la problématique de la gestion de la confidentialité dans les réseaux sociaux et montrons leurs limites. Ensuite, nous introduisons quelques notions formelles liés aux réseaux sociaux, et mettons en évidence les conditions requises afin de concevoir un modèle de contrôle d'accès appropriés aux réseaux sociaux. Ensuite, nous décrivons le modèle de contrôle d'accès proposé et expliquons comment ceci peut être appliqué et implanté. Puis, nous étudions la complexité en temps de réponse du protocole de contrôle d'accès, et, décrivons les expériences effectuées pour étudier ses performance. Enfin, nous présentons *Primates* comme étant un système de gestion de la confidentialité dans les réseaux sociaux appliquant le modèle proposé.
- Dans le **chapitre 6**, nous concluons ce rapport en rappelant les problématiques traitées, les solutions proposées et les résultats obtenus. Nous présentons également quelques perspectives de recherche pour adresser d'autres problèmes liés à aux problématiques traités dans cette thèse.

A.4. Conclusion Globale

Nous nous sommes intéressés dans cette thèse, en premier lieu, au problème de calcul des requêtes d'accessibilité d'une manière générale. Plus précisément, notre but consistait à concevoir et implanter une technique d'indexation permettant le calcul efficace des requêtes de distance dans les grands graphes réels. En deuxième lieu, nous avons traité le problème de gestion de la confidentialité dans les réseaux sociaux, notre objectif étant d'aider les utilisateurs à avoir plus de contrôle sur diffusion et la distribution des informations qu'ils partagent. Les questions les plus importantes qu'on a étudiées dans cette thèse sont les suivantes :

1. Est-il possible de répondre aux requêtes de distance de manière efficace dans les grands graphes denses et orientés ?
2. Y a-t-il des propriétés spécifiques au grands graphes réels qui peuvent exploitées afin de concevoir un algorithme efficace pour les calcul des requêtes de distance ?
3. Quelles sont les exigences qu'un modèle de contrôle d'accès dans les réseaux sociaux doit prendre en compte pour répondre aux besoins des utilisateurs ?
4. Est-il possible de réaliser un modèle de contrôle d'accès qui prend en compte ces exigences et qui est facile à utiliser par les utilisateurs ?

Dans ce chapitre, nous résumons les principales contributions qui ont été détaillées tout au long de cette thèse et apportons des réponses aux questions ci-dessus. Ensuite, nous présentons quelques perspectives de recherche.

A.4.1. Synthèse

La première contribution de cette thèse est *EUQLID*, qui est un schéma d'indexation efficace pour le calcul des requêtes de distance. Comme décrit dans le chapitre 4 et après une étude approfondie de l'état de l'art et des problèmes fondamentaux que soulève ce domaine de recherche, nous avons proposé une nouvelle technique d'indexation qui

exploite une propriété spécifique aux grands graphes réels qu'on a découverte. Cette technique nous semble la plus pratique et la plus efficace pour l'interrogation des grands graphes réels. Nous proposons deux variantes de notre approche : (i) la première variante, appelée *M-EUQLID*, permet de stocker l'index en mémoire vive quand ceci est possible, et (ii) la deuxième variante, appelée *D-EUQLID*, stocke l'index sur le disque dur : elle est utilisée quand il n'y a pas assez d'espace dans la mémoire vive pour stocker l'index. Notre étude expérimentale a montré que *EUQLID* surpassent les approches existantes en termes de temps d'indexation et de réponse, et que les requêtes de calcul de distance peuvent être traitées en quelques centaines de millisecondes sur le plus grand graphe disponible.

La deuxième contribution est un modèle de contrôle d'accès dans les réseaux sociaux (voir Chapitre 5). Ce modèle grain fin permet aux utilisateur de spécifier, de manière explicite, leurs politiques de contrôle d'accès en fonction de leurs besoins (en fonction de la nature des relations avec les autres utilisateurs). En se basant sur les paramètres de confidentialité spécifiés, la décision d'autoriser l'accès aux données partagées est prise à la volée.

La troisième contribution est *Primates* (voir Chapitre 5), qui est un système de gestion de la confidentialité dans les réseaux sociaux démontrant la fiabilité du modèle de contrôle d'accès proposé. Il permet aux utilisateurs de fixés leurs paramètres de confidentialité de manière simple et conviviale.

A.4.2. Perspectives

En conclusion de cette thèse, nous aimerions relever certains problèmes importants dans les contextes de l'interrogation des grands graphes réels et de la gestion de la confidentialité dans les réseaux sociaux, respectivement. Nous classifions ces problèmes en : (i) ceux que nous traitons déjà, (ii) ceux qui demandent, à notre avis, un certain effort mais devraient pouvoir être résolus sans difficulté intrinsèque, et (iii) ceux qui nécessitent des travaux de recherche conséquents.

Commençons par les travaux en cours :

- **Gestion des graphes stockés sur le disque dur.** Une extension utile de *EUQLID* est de développer un algorithme efficace pour indexer les graphes qui ne peuvent pas être entièrement représentés dans la mémoire principale. En effet, certains graphes ne peuvent pas être contenu dans la mémoire principale à cause de leur grande taille et ne peuvent être entièrement stockés que sur un disque dur. L'indexation de ce type de graphes nécessite un accès au disque (Entreés/Sorties ou E/S) à chaque fois que des informations sur le graphe initial sont demandées (ex : liste des successeurs directs d'un noeud donné). Cependant, un grand nombre d'E/S peut affecter considérablement les performances du processus d'indexation surtout quand il s'agit de très grands graphes. Pour ce faire, les approches proposées dans [CKCC12, WC12, CC12] peuvent éventuellement être adaptées et appliquées. Plus précisément, nous pouvons adopter l'algorithme du *Block nested loop* expliqué dans [CKCC12].

Les problèmes suivants semblent être raisonnablement résolubles, avec éventuellement un certain effort :

- **Requêtes d'accessibilité avec des contraintes.** Afin de traiter les requêtes d'accessibilité avec des contraintes sur les étiquettes et la distance, nous pouvons

partionner le graphe initial G (graphe orienté et étiqueté) en plusieurs sous-graphes en fonction des étiquettes sur les arcs (c-à-d., chaque sous-graphe ne comporte que des arcs avec une même étiquette). Ensuite, on applique *EUQLID* sur chaque sous-graphe de manière séparée. Considérons l'exemple de requête d'accessibilité sous contraintes suivant : $Q = u \rightsquigarrow l_1, d_1 \backslash l_2, d_2 \rightsquigarrow v$ cherchant à déterminer s'il existe un chemin entre u et v qui commence par d_1 arcs étiquetés l_1 suivis par d_2 arcs étiquetés l_2. Pour répondre à cette requête, nous pouvons utiliser *EUQLID* pour calculer la distance entre u et le reste des noeuds contenu dans le sous-graphe de l'étiquette l_1, et, on stocke l'ensemble des noeuds $S_1 = \{x \mid d(u, x) = d_1\}$. Ensuite, nous déterminons l'ensemble S_2 dans le sous-graphe de l'étiquette l_2 tel que $S_2 = \{y \mid d(y, v) = d_2\}$. Si l'intersection de S_1 et S_2 n'est pas égale à l'ensemble vide alors u peut atteindre v tout respectant la requête Q, sinon est le réponse à la requête est négative (c-à-d., il n'existe aucun chemin entre u et v dans le graphe initial G respectant la requête Q).

Il est à noter que dans les graphes réels, le nombre total des différentes étiquettes est relativement petit par rapport au nombre total d'arcs dans le graphe. En effet, le graphe RDF de la base de connaissance UniProt est constitué de 22 millions d'arcs et de 91 différentes étiquettes uniquement [ACZ12]. Ceci un implique un nombre relativement petit de sous-graphes résultant du partionnement du graphe. De plus, la distribution de ces étiquettes n'est pas uniforme ce qui signifie qu'il y a un grand nombre d'arcs ayant quelques étiquettes distinctes. Dans le graphe UniProt, l'étiquette "rdf:type" apparaît sur environ 5 millions d'arcs, environ 10 étiquettes apparaissent sur 1 millions d'arcs chacune, et environ 20 étiquettes occupent de 100 milles à 200 millions arcs chacune. Cette distribution biaisée sert à élaguer l'espace potentiellement grand des arcs à explorer lors de traverser les sous-graphes en question. Ceci résulte en des ensembles de noeuds relativement petits pour le calcul de l'intersection.

- **Parallélisation du calcul de la couverture maximale.** Nous pouvons adopter l'algorithme proposé par Berger et al. [BRS94] afin de concevoir une variante parallèle et efficace de l'algorithme incrémental pour le calcul de la couverture maximale d'un ensemble d'éléments donnés, ensuite l'intégrer à *EUQLID*.

- **Passage à l'échelle de *Primates*.** Dans *Primates*, nous avons besoin de calculer la distance entre deux noeuds donnés afin de décider si l'accès doit être autorisé ou non. Dans l'implantation actuelle de *Primates*, ceci est fait au moment que l'on traverse le graphe. Ainsi, pour le passage à l'échelle de *Primates*, nous désirons y intégrer *EUQLID* pour le calcul des distance entre les noeuds (si nécessaire) de façon beaucoup plus efficace.

- **Requête de distance un à plusieurs.** Une requête de distance un à plusieurs calcule la distance entre un noeud source donné et le reste des noeuds dans le graphe. *EUQLID* peut être adapté pour répondre à ce type de requêtes de distance en calculant la distance entre le noeud source et le reste des noeuds un par un.

Enfin, nous présentons quelques problèmes ouverts dont nous pensons qu'ils sont importants pour les deux problématiques traités dans cette thèse, et pour lesquels nous ne connaissons pas de solution.

- **Maintenance de l'index 2-hop.** Au lieu de re-caluler l'index à chaque fois que le graphe change suite à la création/suppression de noeuds/arcs, il vaut mieux concevoir un algorithme permettant la mise à jour de manière efficace de l'index 2-hop déjà crée. Le problème de mise à jour et de maintenance de la couverture 2-hop a été étudié par Bramandia et al. [BCN08]. Une direction possible pour la maintenance de *EUQLID* est d'adapter la solution proposée dans [BCN08].

- **Gestion des conflits.** Les utilisateurs des réseaux sociaux partagent leurs informations ayant un public visé en tête. Dans les scénarios de la vie réelle, ils peuvent aussi avoir un public indésirable pour certaines informations partagées. Permettre aux utilisateurs de spécifier des règles d'autorisation et de refus d'accès en même temps peut engendrer des conflits quand certains utilisateurs font partie des public visé et non désiré. Pour résoudre ce type de conflits, des algorithmes de la théorie des jeux peuvent être éventuellement appliqués [SSP09].

- **Génération automatique des paramètres de confidentialité.** L'un des avantages du modèle de contrôle d'accès qu'on a proposé est le fait que ce dernier permet aux utilisateurs de spécifier de manière explicite leurs propres paramètres de confidentialité. Ce processus guidé par l'utilisateur peut également être combiné à un autre processus automatique permettant aux utilisateurs de proposer des paramètres de confidentialité automatiquement déduites en se basant sur les paramètres déjà spécifiés. Ceci est un problème difficile, qui devrait probablement être attaqué à l'aide d'une combinaison de techniques de traitement du langage naturel et d'apprentissage artificiel.

Cette liste de problèmes ouverts ou partiellement résolus n'est bien sûr pas exhaustive.

Lexique anglais-français

online social network (OSN) réseau social

access control contôle d'accès

p-**approximation algorithm** algorithm *p*-approché

access rules règles d'accès

privacy confidentialité

reachability query requête d'accessibilité

distance query requête de calcul distance

indexing scheme schéma d'indexation

2-hop cover couverture à 2 sauts

set cover couverture d'ensemble

max cover couverture maximale

graph search parcours de graphes

high-degree node noeud à degré élevé

trust confiance

sampling échantillonnage

shortest path chemin le plus court

submodular function fonction sous-modulaire

greedy incrémental

graph traversal parcours de graphe

breadth-first search parcours en largeur

depth-first search parcours en profondeur

spanning couvrant

strongly connected component Composante fortement connexe

scalability passage à l'échelle

user-guided guidée par l'utilisateur

fine-grained grain fin

Self References

[1] Talel Abdssalem and Imen Ben Dhia. A Reachability-Based Access Control Model for Online Social Networks. In First ACM SIGMOD Workshop on Databases and Social Networks (DBSocial), Athenes, Greece, June 2011.

[2] Imen Ben Dhia. Access Control in Social Networks : A reachability-Based Approach. In Joint *EDBT/ICDT Ph.D. Workshop*, Berlin, Germany, March 2012.

[3] Imen Ben Dhia, Talel Abdssalem and Mauro Sozio. Primates: A Privacy Management System for Social Networks. In *Proc CIKM*, Maui, Hawaii, US, October 2012.

[4] Imen Ben Dhia, Talel Abdssalem and Mauro Sozio. Primates: A Privacy Management System for Social Networks. In *FGG-EGC Workshop*, Toulouse, France, January 2013.

[5] Imen Ben Dhia, Talel Abdssalem and Mauro Sozio. EUQLID: Efficient Distance Queries in Large Directed Graphs. Under review.

External References

[ABJ89] R. Agrawal, A. Borgida, and H. V. Jagadish. Efficient management of transitive relationships in large data and knowledge bases. *SIGMOD Rec.*, pages 253–262, 1989.

[ACF12] Cuneyt Akcora, Barbara Carminati, and Elena Ferrari. Privacy in social networks: How risky is your social graph? In *ICDE*, pages 9–19, 2012.

[ACS09] Talel Abdessalem, Bogdan Cautis, and Asma Souihli. Trust management in social networks. Technical report, Telecom ParisTech, 2009.

[ACZ12] Medha Atre, Vineet Chaoji, and Mohammed J. Zaki. Bitpath – label order constrained reachability queries over large graphs. *CoRR*, abs/1203.2886, 2012.

[AD11] Talel Abdessalem and Imen Ben Dhia. A reachability-based access control model for online social networks. In *DBSocial*, pages 31–36, 2011.

[ADH+08] Noga Alon, Phuong Dao, Iman Hajirasouliha, Fereydoun Hormozdiari, and S. Cenk Sahinalp. Biomolecular network motif counting and discovery by color coding. *Bioinformatics*, pages i241–i249, 2008.

[AFGW10] Ittai Abraham, Amos Fiat, Andrew V. Goldberg, and Renato F. Werneck. Highway dimension, shortest paths, and provably efficient algorithms. SODA, pages 782–793, 2010.

[AHCR13] Fu Ada, Wu Huanhuan, Cheng, and Wong Raymond. Is-label: an independent-set based labeling scheme for point-to-point distance querying. *PVLDB*, 2013.

[AW10] Charu C. Aggarwal and Haixun Wang, editors. *Managing and Mining Graph Data*, volume 40 of *Advances in Database Systems*. Springer, 2010.

[BA99] Albert-László Barabási and Réka Albert. *Science*, pages 509–512, 1999.

[Bar02] Albert-Laszlo Barabasi. Linked the new science of networks, 2002.

[Bar03] Albert-Laszlo Barabasi. *Linked: How Everything Is Connected to Everything Else and What It Means for Business, Science, and Everyday Life*. Plume Books, 2003.

[BBGT12] Paolo Boldi, Francesco Bonchi, Aristides Gionis, and Tamir Tassa. Injecting uncertainty in graphs for identity obfuscation. *Proc. VLDB Endow.*, pages 1376–1387, 2012.

[BCN08] Ramadhana Bramandia, Byron Choi, and Wee Keong Ng. On incremental maintenance of 2-hop labeling of graphs. pages 845–854, 2008.

[BE07] Danah M. Boyd and Nicole B. Ellison. Social network sites: Definition, history, and scholarship. *Journal of Computer-Mediated Communication*, page article 11, 2007.

[BKH01] James E. Bartlett, Joe W. Kotrlik, and Chadwick C. Higgins. Organizational research: Determining appropriate sample size in survey research. *Information Technology, Learning, and Performance Journal*, 19, 2001.

[BKM+00] Andrei Broder, Ravi Kumar, Farzin Maghoul, Prabhakar Raghavan, Sridhar Rajagopalan, Raymie Stata, Andrew Tomkins, and Janet Wiener. Graph structure in the web. *Comput. Netw.*, pages 309–320, 2000.

[BMS02] Peter S. Bearman, James Moody, and Katherine Stovel. Chains of affection: The structure of adolescent romantic and sexual networks. *American Journal of Sociology*, pages 44–91, 2002.

[BRS94] Bonnie Berger, John Rompel, and Peter W. Shor. Efficient nc algorithms for set cover with applications to learning and geometry. *J. Comput. Syst. Sci.*, pages 454–477, 1994.

[BRSV11] Paolo Boldi, Marco Rosa, Massimo Santini, and Sebastiano Vigna. Layered label propagation: A multiresolution coordinate-free ordering for compressing social networks. WWW, 2011.

[BV12] Paolo Boldi and Sebastiano Vigna. Four degrees of separation, really. In *ASONAM*, pages 1222–1227, 2012.

[CC08] Yangjun Chen and Yibin Chen. An efficient algorithm for answering graph reachability queries. In *ICDE*, pages 893–902, 2008.

[CC12] Shumo Chu and James Cheng. Triangle listing in massive networks. *ACM Trans. Knowl. Discov. Data*, pages 17:1–17:32, 2012.

[CFP09] Barbara Carminati, Elena Ferrari, and Andrea Perego. Enforcing access control in web-based social networks. *ACM Trans. Inf. Syst. Secur.*, pages 6:1–6:38, 2009.

[CHKZ02] Edith Cohen, Eran Halperin, Haim Kaplan, and Uri Zwick. Reachability and distance queries via 2-hop labels. SODA, pages 937–946, 2002.

[CHWF13] James Cheng, Silu Huang, Huanhuan Wu, and Ada Wai-Chee Fu. Tf-label: a topological-folding labeling scheme for reachability querying in a large graph. SIGMOD, pages 193–204, 2013.

[CKCC12] James Cheng, Yiping Ke, Shumo Chu, and Carter Cheng. Efficient processing of distance queries in large graphs: a vertex cover approach. SIGMOD, pages 457–468, 2012.

[CKW10] Graham Cormode, Howard Karloff, and Anthony Wirth. Set cover algorithms for very large datasets. CIKM '10, pages 479–488, 2010.

[cnn] http://edition.cnn.com/2010/tech/03/29/facebook.job-seekers/index.html.

[CP10] Jing Cai and Chung Keung Poon. Path-hop: efficiently indexing large graphs for reachability queries. CIKM, pages 119–128, 2010.

[CSC+12] James Cheng, Zechao Shang, Hong Cheng, Haixun Wang, and Jeffrey Xu Yu. K-reach: who is in your small world. *Proc. VLDB Endow.*, pages 1292–1303, 2012.

[CY09] Jiefeng Cheng and Jeffrey Xu Yu. On-line exact shortest distance query processing. EDBT, pages 481–492, 2009.

[CYL+06] Jiefeng Cheng, Jeffrey Xu Yu, Xuemin Lin, Haixun Wang, and Philip S. Yu. Fast computation of reachability labeling for large graphs. EDBT, pages 961–979, 2006.

[CYL+08] Jiefeng Cheng, Jeffrey Xu Yu, Xuemin Lin, Haixun Wang, and Philip S. Yu. Fast computing reachability labelings for large graphs with high compression rate. EDBT, 2008.

[CZF04] D. Chakrabarti, Y. Zhan, and C. Faloutsos. R-mat: A recursive model for graph mining. SIAM, 2004.

[Dan09] George Danezis. Inferring privacy policies for social networking services. AISec, 2009.

[DAS12] Imen Ben Dhia, Talel Abdessalem, and Mauro Sozio. Primates: a privacy management system for social networks. In *CIKM*, pages 2746–2748, 2012.

[Dij59] Edsger. W. Dijkstra. A note on two problems in connexion with graphs. *Numerische Mathematik*, 1:269–271, 1959.

[ER60] P. Erdos and A Rényi. On the evolution of random graphs. In *PUBLICATION OF THE MATHEMATICAL INSTITUTE OF THE HUNGARIAN ACADEMY OF SCIENCES*, pages 17–61, 1960.

[faca] http://newsroom.fb.com/key-facts.

[facb] http://www.facebook.com/press/info.php?statistics.

[fbG] https://www.facebook.com/about/graphsearch.

[FFF99] Michalis Faloutsos, Petros Faloutsos, and Christos Faloutsos. On power-law relationships of the internet topology. In *SIGCOMM*, pages 251–262, 1999.

[FL10] Lujun Fang and Kristen LeFevre. Privacy wizards for social networking sites. WWW, 2010.

[FLGC02] Gary William Flake, Steve Lawrence, C. Lee Giles, and Frans M. Coetzee. Self-organization and identification of web communities. *Computer*, pages 66–71, 2002.

[FLM+11] Wenfei Fan, Jianzhong Li, Shuai Ma, Nan Tang, and Yinghui Wu. Adding regular expressions to graph reachability and pattern queries. ICDE, pages 39–50, 2011.

[fol] http://www.fernfachhochschule.ch/ffhs/afe/lws/forschung/research.

[FT87] Michael L. Fredman and Robert Endre Tarjan. Fibonacci heaps and their uses in improved network optimization algorithms. *J. ACM*, pages 596–615, 1987.

[GBSW10] Andrey Gubichev, Srikanta J. Bedathur, Stephan Seufert, and Gerhard Weikum. Fast and accurate estimation of shortest paths in large graphs. In *CIKM*, 2010.

[GKBM10] Minas Gjoka, Maciej Kurant, Carter T. Butts, and Athina Markopoulou. Walking in facebook: A case study of unbiased sampling of osns. In *Proceedings of IEEE INFOCOM*, 2010.

[GKRT04] R. Guha, Ravi Kumar, Prabhakar Raghavan, and Andrew Tomkins. Propagation of trust and distrust. WWW, 2004.

[GLNF78] L. A. Wolsey G. L. Nemhauser and M. L. Fisher. An analysis of approximations for maximizing submodular set functions. *Mathematical Programming 14*, pages 265–294, 1978.

[GN02] M. Girvan and M. E. J. Newman. Community structure in social and biological networks. *Proceedings of the National Academy of Sciences*, pages 7821–7826, 2002.

[GN11] Andrey Gubichev and Thomas Neumann. Path query processing on very large rdf graphs. In *WebDB*, 2011.

[Graa] http://en.wikipedia.org/wiki/graphdatabase.

[grab] http://newsroom.fb.com/news/562/introducing-graph-search-beta.

[grac] http://www.sommer.jp/aa10/aa8.pdf.

[GTF08] Saikat Guha, Kevin Tang, and Paul Francis. Noyb: privacy in online social networks. WOSP, 2008.

[HMJ+08] Michael Hay, Gerome Miklau, David Jensen, Don Towsley, and Philipp Weis. Resisting structural re-identification in anonymized social networks. *Proc. VLDB Endow.*, pages 102–114, 2008.

[Hoc97a] D. S. Hochbaum. Approximation algorithms for np-hard problems. *PWS Publishing Co., Boston, MA, USA*, 1997.

[Hoc97b] Dorit S. Hochbaum. Approximation algorithms for np-hard problems. chapter Approximating covering and packing problems: set cover, vertex cover, independent set, and related problems, pages 94–143. 1997.

[int] http://www.sciencedaily.com/releases/2007/08/070831144233.htm.

[JHW+10] Ruoming Jin, Hui Hong, Haixun Wang, Ning Ruan, and Yang Xiang. Computing label-constraint reachability in graph databases. SIGMOD, pages 123–134, 2010.

[JMB11] Sonia Jahid, Prateek Mittal, and Nikita Borisov. Easier: encryption-based access control in social networks with efficient revocation. In *ASIACCS*, pages 411–415, 2011.

[JNM+12] Sonia Jahid, Shirin Nilizadeh, Prateek Mittal, Nikita Borisov, and Apu Kapadia. Decent: A decentralized architecture for enforcing privacy in online social networks. In *PerCom Workshops*, pages 326–332, 2012.

[Joh73] David S. Johnson. Approximation algorithms for combinatorial problems. STOC '73, pages 38–49, 1973.

[JRDX12] Ruoming Jin, Ning Ruan, Saikat Dey, and Jeffrey Yu Xu. Scarab: scaling reachability computation on large graphs. SIGMOD, pages 169–180, 2012.

[JRXL12] Ruoming Jin, Ning Ruan, Yang Xiang, and Victor Lee. A highway-centric labeling approach for answering distance queries on large sparse graphs. In *Proceedings of the 2012 ACM SIGMOD International Conference on Management of Data*, SIGMOD, pages 445–456, 2012.

[JRXW11] Ruoming Jin, Ning Ruan, Yang Xiang, and Haixun Wang. Path-tree: An efficient reachability indexing scheme for large directed graphs. *ACM Trans. Database Syst.*, pages 7:1–7:44, 2011.

[JXRF09] Ruoming Jin, Yang Xiang, Ning Ruan, and David Fuhry. 3-hop: a high-compression indexing scheme for reachability query. SIGMOD, pages 813–826, 2009.

[KKR+99] Jon M. Kleinberg, Ravi Kumar, Prabhakar Raghavan, Sridhar Rajagopalan, and Andrew S. Tomkins. The web as a graph: measurements, models, and methods. COCOON, pages 1–17, 1999.

[KRS+09] Gjergji Kasneci, Maya Ramanath, Mauro Sozio, Fabian M. Suchanek, and Gerhard Weikum. Star: Steiner-tree approximation in relationship graphs. In *ICDE*, pages 868–879, 2009.

[LB09] Matthew Lucas and Nikita Borisov. Flybynight: mitigating the privacy risks of social networking. SOUPS, 2009.

[les] http://snap.stanford.edu/data/.

[LHL+12] B. Li, T. Huang, L. Liu, Y. Cai, and K. Chou. Identification of colorectal cancer related genes with mrmr and shortest path in protein-protein interaction network. *PLoS One*, 7(4):e33393, 2012.

[LKF05] Jure Leskovec, Jon Kleinberg, and Christos Faloutsos. Graphs over time: densification laws, shrinking diameters and possible explanations. SIGKDD, pages 177–187, 2005.

[LKF07] Jure Leskovec, Jon Kleinberg, and Christos Faloutsos. Graph evolution: Densification and shrinking diameters. *ACM Trans. Knowl. Discov. Data*, 2007.

[LLDM08] Jure Leskovec, Kevin J. Lang, Anirban Dasgupta, and Michael W. Mahoney. Statistical properties of community structure in large social and information networks. WWW, pages 695–704, 2008.

[LLL+08] Haifeng Liu, Ee-Peng Lim, Hady W. Lauw, Minh-Tam Le, Aixin Sun, Jaideep Srivastava, and Young Ae Kim. Predicting trusts among users of online communities: an epinions case study. In *EC*, 2008.

[MS02] Sergei Maslov and Kim Sneppen. Specificity and stability in topology of protein networks. *Science*, pages 910–913, 2002.

[neo] http://www.neo4j.org/.

[New03] M. E. J. Newman. The structure and function of complex networks. *SIAM REVIEW*, pages 167–256, 2003.

[NJM+12] Shirin Nilizadeh, Sonia Jahid, Prateek Mittal, Nikita Borisov, and Apu Kapadia. Cachet: a decentralized architecture for privacy preserving social networking with caching. In *CoNEXT*, pages 337–348, 2012.

[PBCG09] Michalis Potamias, Francesco Bonchi, Carlos Castillo, and Aristides Gionis. Fast shortest path distance estimation in large networks. CIKM, pages 867–876, 2009.

[phi] http://kieranhealy.org/blog/archives/2013/06/18/a-co-citation-network-for-philosophy/.

[pria] https://www.facebook.com/about/graphsearch/privacy.

[prib] http://www.pcmag.com/article2/0,2817,2414599,00.asp.

[Red98] S. Redner. *European Physical Journal B*, pages 131–134, 1998.

[SABW13] Stephan Seufert, Avishek Anand, Srikanta Bedathur, and Gerhard Weikum. Ferrari: Flexible and efficient reachability range assignment for graph indexing. ICDE, 2013.

[SCT+10] Mohamed Shehab, Gorrell Cheek, Hakim Touati, Anna C. Squicciarini, and Pau-Chen Cheng. Learning based access control in online social networks. WWW, 2010.

[SSP09] Anna Cinzia Squicciarini, Mohamed Shehab, and Federica Paci. Collective privacy management in social networks. In *WWW*, 2009.

[STW04] Ralf Schenkel, Anja Theobald, and Gerhard Weikum. Hopi: An efficient connection index for complex xml document collections. In *EDBT*, 2004.

[SW92] Michael Schwartz and David C.M. Wood. Discovering shared interests among people using graph analysis of global electronic mail traffic. *Communications of the ACM*, 36:78–89, 1992.

[SW05] Amit Sahai and Brent Waters. Fuzzy identity-based encryption. EUROCRYPT, pages 457–473, 2005.

[TMTM69] Jeffrey Travers, Stanley Milgram, Jeffrey Travers, and Stanley Milgram. An experimental study of the small world problem. *Sociometry*, 32:425–443, 1969.

[top] http://lifehacker.com/5808625/five-best-web-sites-for-image-hosting-and-photo-sharing/.

[tra] http://www.slideshare.net/thobe/nosqleu-graph-databases-and-neo4j.

[twi] http://en.wikipedia.org/wiki/twitter.

[VFD+07] Monique V. Vieira, Bruno M. Fonseca, Rodrigo Damazio, Paulo B. Golgher, Davi de Castro Reis, and Berthier Ribeiro-Neto. Efficient search ranking in social networks. CIKM, pages 563–572, 2007.

[VMZ+10] Chad Vicknair, Michael Macias, Zhendong Zhao, Xiaofei Nan, Yixin Chen, and Dawn Wilkins. A comparison of a graph database and a relational database: a data provenance perspective. In *Proceedings of the 48th Annual Southeast Regional Conference*, pages 42:1–42:6, 2010.

[vSdM11] Sebastiaan J. van Schaik and Oege de Moor. A memory efficient reachability data structure through bit vector compression. SIGMOD, pages 913–924, 2011.

[WC12] Jia Wang and James Cheng. Truss decomposition in massive networks. *Proc. VLDB Endow.*, pages 812–823, 2012.

[web] http://law.di.unimi.it/datasets.php.

[Wei10] Fang Wei. Tedi: efficient shortest path query answering on graphs. SIGMOD, pages 99–110, 2010.

[Wil27] Edwin B. Wilson. Probable inference, the law of succession, and statistical inference. *Journal of the American Statistical Association*, pages 209–212, 1927.

[Wor09] JENNA Wortham. More employers use social networks to check out applicants. *The New York Times*, 2009.

[WZL+12] Yonggang Wang, Ennan Zhai, Eng Keong Lua, Jianbin Hu, and Zhong Chen. isac: Intimacy based access control for social network sites. In *Proceedings of the 2012 9th International Conference on Ubiquitous Intelligence and Computing and 9th International Conference on Autonomic and Trusted Computing*, pages 517–524, 2012.

[XAT12] Qian Xiao, Htoo Htet Aung, and Kian-Lee Tan. Towards ad-hoc circles in social networking sites. DBSocial, pages 19–24, 2012.

[XWP+09] Yanghua Xiao, Wentao Wu, Jian Pei, Wei Wang, and Zhenying He. Efficiently indexing shortest paths by exploiting symmetry in graphs. EDBT, pages 493–504, 2009.

[YCZ10] Hilmi Yildirim, Vineet Chaoji, and Mohammed J. Zaki. Grail: scalable reachability index for large graphs. *Proc. VLDB Endow.*, 2010.

[ZPL08] Bin Zhou, Jian Pei, and WoShun Luk. A brief survey on anonymization techniques for privacy preserving publishing of social network data. *SIGKDD Explor. Newsl.*, pages 12–22, 2008.