

FINDING PATTERNS IN SEMANTIC GRAPH FORMALISMS

BY

GOKARNA SHARMA

A DISSERTATION
SUBMITTED TO THE FACULTY OF COMPUTER SCIENCE
IN CONFORMITY WITH THE REQUIREMENTS FOR
THE DEGREE OF MASTER OF SCIENCE

FREE UNIVERSITY OF BOZEN-BOLZANO
BOLZANO, ITALY
OCTOBER 2008

I certify that I have read this dissertation and that, in my opinion, it is fully adequate in scope and quality as a dissertation for the degree of Master of Science in Computer Science.

(Prof. Enrico Franconi) First Supervisor

I certify that I have read this dissertation and that, in my opinion, it is fully adequate in scope and quality as a dissertation for the degree of Master of Science in Computer Science.

(Dr. Peter F. Patel-Schneider) Second Supervisor

Abstract

When intelligence analysts are required to understand a complex uncertain situation, one of the techniques they use most often is to simply draw a diagram of the situation. The diagrams, also called attributed relational graphs or semantic graphs, generally capture the meaning about the situation in their nodes and edges, where the nodes represent concepts/entities and the edges represent the relations/connectivity between the nodes.

An important research problem in the area of semantic knowledge discovery and pattern analysis is to identify common/uncommon patterns and instances on these diagrams. Finding patterns and anomalies in data has important applications in intelligence analysis domains such as crime detection and homeland security. The intelligence community's focus over many years on improving intelligence collection has come at the cost of improving intelligence analysis. The problem today is often not a lack of information, but instead, information overload. Analysts lack tools to locate the relatively few bits of relevant information and tools to support reasoning over that information. Graph-based algorithms can help intelligence analysts solve this problem by sifting through a large amount of data to find the small subset that is indicative of suspicious or abnormal activity.

Till today, large amount of work related to analysis/prediction of threat to national security has been done "manually". This process is slow and much labor-intensive. Today's need is tools to analyze large semantic graphs automatically so that we can get the related results in the considerably short span of time. There are many challenges to do these things fast and effectively. The challenges, including how to represent the data effectively, representing temporal information, representing the use of ontology related to them, etc. Other significant challenges include scale and complexity of data and ontologies that are useful for the analysis. We formalize these graphs in this thesis along with provide the efficient way to represent them in logic formalisms.

While there are several existing supervised/unsupervised learning frameworks to identify patterns and anomalies from graph data, there has been little work aimed at discovering patterns and abnormal instances in very large semantic graphs whose nodes are richly connected with many different types of links from knowledge representation perspective. To address this problem, we design a novel, disjunctive logic programming framework that utilizes the information provided by different types of nodes and links to identify abnormal nodes and patterns. Our approach represents the dependencies between nodes and paths in the graph in first order logic predicates to capture what we call "semantic profiles" of nodes, and then applies disjunctive logic rules to find abnormal nodes and patterns that are significantly different from their closest neighbors. In a set of experiments on movies data, our system can almost perfectly identify the abnormal instances/patterns

and outperforms several other state-of-the-art machine learning methods that have been used to analyze the same data.

Last, as semantic graphs comprise of ontology with small non-trivial TBox of terminologies and very large ABox of assertions, we study the possibilities of analyzing semantic graphs using description logic (DL) reasoners. We also review the work that has been done which may help us in the analysis of semantic graphs and perform several experiments with various DL reasoners: KAON2, RACER, and Pellet using semantic graph knowledge bases.

Acknowledgments

At the outset, I would like to extol my thesis supervisor Prof. Enrico Franconi for his advice during my masters research endeavor during yester months. As my supervisor, he constantly forced me to remain focused on achieving my goal. His observation and comments helped me to establish the overall direction of the research and to move forward expeditiously with investigation in depth. I thank him for providing me the opportunity to work with numerous local and global papers.

I am grateful to my second supervisor Dr. Peter F. Patel-Schneider, Member of Technical Staff, Alcatel-Lucent Bell Laboratories for providing me the guidance in carrying out my research work during my stay at Bell Laboratories this summer. He cared me a lot ranging from day-to-day activities to research activities. He had always time to meet to answer lots of my emails, to listen to my ideas, to spot possible directions of research. His suggestions went from how to write a good paper to how to give a good talk to where to go for good food.

My gratitude is also due to the teaching and non teaching staff of my faculty for their constant help. Specially, I would like to thank our faculty secretary Ms. Federica Maria Cumer for taking care of all administrative matters.

I would like to acknowledge the EMCL consortium for the two year Erasmus Mundus grant which helped me start this all; the Alcatel-Lucent Bell Laboratories for providing me necessary facilities to conduct research as a summer consultant.

The joy I received from working on this thesis would have been meaningless without my relatives and friends. I would like to thank them all at this moment.

Gokarna Sharma
Bolzano, 2008

Contents

	i
Abstract	ii
Acknowledgments	iv
List of Tables	vii
List of Figures	viii
I Semantic Graph Formalisms	1
1 Introduction	2
1.1 Problem Definition	2
1.2 Research Objective and Goals	4
1.3 Current Situation	4
1.4 Contributions and Design Considerations	5
1.5 Thesis Outline	6
2 Semantic Graphs	7
2.1 Graph Terminologies	7
2.2 Semantic Graphs	9
2.3 Ontology Graph	14
2.4 Importance of the Ontology Graph	16
2.5 Scale in Semantic Graphs	17
2.6 Semantic Networks and Ontology Hierarchy	19
2.7 Some Issues in Ontology-Assisted Querying	20
II Reasoning on Semantic Graphs using DLP	22
3 Representing Semantic Graphs in DLP	23
3.1 Disjunctive Logic Programs and DLV System	23
3.1.1 Syntax	24
3.1.2 Semantics	25
3.2 Modeling Semantic Graphs in First Order Logic	27
3.3 Inductive Logic Programming	32

4	Finding Patterns in Semantic Graphs	35
4.1	Structure of a Pattern Query	35
4.2	The Importance of Abnormal Instances	36
4.3	Graph Matching	37
4.3.1	Partial Graph Matching	38
4.3.2	Graph Matching allowing more than one Correspondence per Vertex	39
4.3.3	Complexity of Graph Matching	39
4.3.4	Graph Edit Distance	39
4.4	Pattern Analysis in Semantic Graphs	40
4.5	Ontologies for Graph Matching	43
4.6	Pattern Matching	44
4.6.1	Exact Subgraph Matcher	45
4.6.2	Partial Subgraph Matcher	45
4.6.3	Hierarchy Matcher	46
4.6.4	Inexact Matcher	46
4.7	Result Filtering Mechanisms	46
5	Analysis on Movies Database	50
5.1	System Description	50
5.2	Experimental Setup	53
5.3	Analysis and Evaluation	54
5.4	Experience with DLV	56
5.5	Related Work	56
III	Reasoning on Semantic Graphs using Description Logics	59
6	Representing Semantic Graphs in DLs	60
6.1	Background	60
6.2	Description Logic <i>SHIQ</i>	62
6.3	Representing Semantic Graphs	64
7	Experiments on Semantic Graph Knowledge Bases	65
7.1	KAON2, RACER and Pellet Architecture	65
7.2	Data and Experiments	66
7.2.1	Test Knowledge Bases and Queries	67
7.2.2	Performance Measurement	69
7.3	Analysis	71
7.4	Related Work	74
IV	Summing Up	76
8	Conclusions and Further Research	77
8.1	Conclusions	77
8.2	Further Research	78
	Bibliography	80

List of Tables

5.1	Pseudo-code algorithm for pattern finding framework	52
6.1	Syntax and semantics of the Description Logic $\mathcal{SHIQ}$	63
7.1	Test data statistics	68
7.2	Performance table of queries over different knowledge bases	70

List of Figures

2.1	A semantic graph of bibliography domain and its corresponding ontology graph . .	10
2.2	A semantic graph with vertex and edge attributes	12
2.3	A coarser ontology graph with only one vertex type vehicle	18
2.4	A two-level ontology hierarchy for vehicle domain	18
2.5	A finer ontology graph with possible hierarchy in vertex types	19
4.1	Example query structure	35
4.2	A basic pattern represented as a graph with respective types of the nodes	41
4.3	A part of a semantic graph that match to the pattern given in figure 4.2	41
4.4	A part of a semantic graph that is similar to the pattern in figure 4.2	42
4.5	The result after matching figure 4.3 with pattern defined in figure 4.2	42
4.6	The result after matching figure 4.4 with pattern defined in figure 4.2	43
4.7	A basic pattern represented as a graph indicating their types respectively	43
4.8	A pattern in semantic graph that match to the query pattern defined in figure 4.7	44
4.9	Exact graph matching	45
4.10	Partial graph matching	46
4.11	Inexact graph matching	47
5.1	Flow graph for analyzing semantic graphs using Disjunctive Datalog	51
5.2	Ontology graph of Movies database	54
7.1	Movie query $M_1(x)$	71
7.2	Movie query $M_2(x, y)$	71
7.3	Movie query $M_3(x, y, z)$	71
7.4	Univ-Bench query $U_1(x)$	72
7.5	Univ-Bench query $U_2(x, y)$	72
7.6	Univ-Bench query $U_3(x, y, z)$	72
7.7	KAON2 performance over queries	73
7.8	RACER performance over queries	73
7.9	Pellet performance over queries	73

Part I

Semantic Graph Formalisms

Chapter 1

Introduction

‘Imagination is more important than knowledge. Knowledge is limited. Imagination encircles the world.’

Albert Einstein

Semantic graphs are used extensively for representing information about the data come from different sources. Before proceeding toward formalizing semantic graphs in next chapter, this chapter explains in brief the problem considered in this thesis and the method proposed for the solution of the problem. We start this chapter with first describing the problem considered for the thesis in detail and present our contributions to solve the problem. Next, we discuss in brief about the related work that has been done in several related areas which may help us to solve our problem and we present outline of the thesis at the end.

1.1 Problem Definition

When intelligence analysts are required to understand a complex uncertain situation, one of the techniques they use most often is to simply draw a diagram of the situation [CGM04]. These diagrams, also called *attributed relational graphs* or *semantic graphs*¹ [BERC05, KYL04], generally capture the meaning about the situation in terms of diagrams with nodes and edges, where the nodes represent concepts/entities and the edges represent the relations/connectivity between the nodes. In other words, nodes represent people, organization, objects, or events and edges represent relationships like interaction, ownership, or trust.

An important problem in the area of homeland security is to identify useful information for intelligence analysis in large data sets, which is represented naturally in the form of attributed relational graphs. The purpose is to extract small useful information from the very large set of data. The intelligence community’s focus over many years on improving intelligence collection has come at the cost of improving intelligence analysis. The problem today is often not lack of information, but instead, information overload [CGM04]. Analysts lack tools to locate the relatively few bits of relevant information and tools to support reasoning over that information. Graph-based

¹<http://dydan-research.blogspot.com/2007/05/analyzing-semantic-graphs.html>

algorithms can help intelligence analysts solve the first problem by sifting through a large amount of data to find the small subset that is indicative of suspicious or abnormal activity. These activities are suspicious not because of the characteristics of the single actor, but because of the dynamics between a group of actors. *Subgraph isomorphism* [GJ90] and *social network analysis* (SNA) [Sco00] are two important graph-based approaches that help analysts detect suspicious activities in large volumes of data [CGM04].

Graph-based techniques are quite popular in the field of intelligence analysis. Not only they allow us to represent the data but also they carry semantic information in their nodes and edges. The semantic information is again useful for the intelligence analysis as it carries the important information exhibits by the data. Various existing algorithms operate on graphs make us easier to analyze the graphs to extract useful information. For instance, subgraph isomorphism algorithms search through large graphs to find regions that are instances of specific pattern graph [Ull76].

Social network analysis studies the sequences of interaction between the actors to find out the suspicious activities from the data available for analysis [CM04]. *Data mining* [HSM01] learns from past experience and applies this knowledge to another situation to develop a predictive model of what will happen in the future. Although there are methods from data mining and social network analysis focusing on finding patterns which exhibit abnormal behavior from the large data sets, these methods aren't so powerful in terms of finding the relevant patterns indispensable for the intelligence analysis. These methods need much manual labor and resources to actually find the results needed for the intelligence analysis.

Till today, large amount of work related to analysis/prediction of threat to national security has been done "manually" [CGM04]. This process is slow and labor-intensive. Today's need is tools to analyze large semantic graphs automatically so that we can get the related results in the considerably short span of time. There are many challenges to do these things fast and effectively. One of the most significant challenge is the scale and complexity of data and ontologies that are useful for the analysis. Although semantic graphs has been using for many years for the analysis of large data set, the little work has been done in formal definitions of these graphs.

The other challenges are related to how to effectively store and maintain very large graphs in the databases, how to effectively query them, and perform logical inferences like deduction and abduction, how to find most interesting, relevant, abnormal nodes and connections, how to validate the associated domain ontologies with respect to the semantic graph data, and how to translated/map between the graphs based on different ontologies, etc. We have to consider for the scalable representation and reasoning mechanism because the mechanism of semantic knowledge discovery plays vital role in the effective analysis of semantic graphs.

Analysis of data represented in the form of attributed relational graphs from other aspects is necessary to find the suspicious and useful information from the large volume of data. As we can represent real world data in terms of attributed relational graphs (also called semantic graphs here after), we can transform these graphs in to logic programs, especially disjunctive logic programs (DLP). Using the datalog, especially disjunctive datalog we can really investigate all the possible patterns from the data. The representation of semantic graphs in the disjunctive datalog program is quite obvious: we can write the edge type as a predicate and the two ends of edge as arguments.

1.2 Research Objective and Goals

Our motivation comes from representing real world data in terms of meaningful diagrams such as attributed relational graphs because most of the real word data exhibits the properties of graphs [ERC05]. The research goal of this thesis is three fold. The first part, which we refer to as the semantic graph formalisms, focus on formalizing the semantic graphs and its corresponding ontology graph. The second part, which we refer to as the reasoning on semantic graphs using DLP, focus on designing a framework that is capable of identifying abnormal nodes and patterns in very large and complex semantic graphs from disjunctive logic programming perspective. The last part, which we refer to as the reasoning on semantic graphs using DLs, focus on reviewing the related work that has been done which may help us in the analysis of semantic graphs from the description logic perspective. As family of description logics have the rich syntax and semantics, it is worthy to find a way to analyze the semantic graphs using one of the families of description logics.

The main challenge of the second part of the thesis is to design an anomaly detection system for semantic graphs that can achieve all the requirements discussed previously. To our knowledge, there is no effective system based on logic programming has been proposed that can satisfy our requirements. While there are systems aimed at identifying useful patterns and suspicious instances in semantic graphs (in terms of MRN's) all of them are either supervised or unsupervised classification systems and are mostly indirect methods. In this thesis, we study semantic graphs from knowledge representation perspective using disjunctive logic programs for the intelligence analysis. At first, we introduce formal definitions of semantic graph and corresponding ontology graph. Then we formalize the pattern finding mechanisms for semantic graphs to extract important information. We describe our proposed system with implementation details to identify the patterns and suspicious (or abnormal) nodes in a semantic graph along with the analysis of results for pattern analysis. We use the Movies data available on UCI KDD archive [HB99] to evaluate our framework for the real world pattern analysis efficiency. We further investigate the effective use of ontology hierarchy to find the exact/inexact matches from the semantic graph.

At the third part of the thesis, we propose a way to analyze the semantic graphs from the description logics perspective. The goal is to highlight the benefits of using DL to formally analyzing semantics graphs. As semantic graphs have domain ontology hierarchy associated with their ontology graphs, the analysis of them using the richer syntax and semantics of ontology reasoning tools based on the family of description logics, we believe, can provide an effective way to deal with them. We also perform several experiments with different DL reasoners using knowledge bases of varying size.

1.3 Current Situation

Semantic graphs have been studied for the intelligence analysis from many years. Its formal semantics from the logical point of view, however is still an ongoing work. We can get some of the references for knowledge representation and path finding issues in semantic graphs for relationship detection in [BERC05, LC03a, ERC05, LC07]. [BERC05] presents some statistical measures for the

analysis of semantic graphs, as well as issues related to the scale (level of detail) of semantic graphs. They basically generalize complex network based techniques in order to apply them in analysis of semantic graphs. [ERC05] focus on finding the probabilistic heuristics that utilize semantic graph's ontological information to reduce the search space between a source vertex and a destination vertex of semantic graph. An unsupervised framework to identify abnormal or suspicious node in the semantic graph along with the understandable explanation for such type of findings has purposed in [LC07]. [LC03a] presents an unsupervised link discovery method aimed at discovering unusual, interestingly linked entities in multi-relational datasets and various notions of rarity are introduced to measure the “interestingness” of sets of paths and entities. [LC03b] developed a set of unsupervised link discovery methods that compute interestingness in Bibliographical dataset.

The semantic graphs have been studied from the view of link discovery [Cha07, ACM⁺04] using logic-based approaches too. The POWERLOOM² use logic-based knowledge representation and reasoning system and RDBMS to represent evidence, patterns, background knowledge, meta-knowledge, etc. The POWERLOOM enables effective link discovery on realistic datasets that are structurally rich, high volume and maintains high precision and recall. POWERLOOM logic model use KIF. While POWERLOOM is not based on description logics, it does have a description classifier which uses technology derived from the Loom classifier to classify descriptions expressed in full first order predicate calculus. This is just a classifier, specially a link discovery system, and no progress has been seen in ongoing work for finding the patterns and instances on data sets.

There is also a small community of machine learning and social network analysis researchers that have used semantic graphs with heterogenous types of vertices and edges [Get03, NAJ03, MJ03]. These algorithms are typically designed for learning probabilistic models on vertices and/or edges for subsequent inference. For example, [MJ03] learn models that identify predictive structures in semantic graphs. For social networks, [FMT04] has developed an algorithm for detecting connection subgraphs. This approach regards a social network with weighted undirected edges as an electric circuit with a network of resistors and a connection between two vertices as a path with the most units of electric current. Their algorithms can not be inducted to semantic graphs because semantic graphs are carrying much richer information than social networks. In summary, our work is related to the problems and solutions from a variety of fields including intelligence analysis, data mining, etc., which we discuss in detail at Section 5.5.

1.4 Contributions and Design Considerations

As we mentioned already, an important problem in the area of homeland security is to identify *useful patterns* and *abnormal (or suspicious) entities* in large datasets [Lin06, LC07], which can be represented naturally in the form of semantic graphs [XC05]. Our goal is to design a pattern analysis framework to mirror the processing that is currently done using semantic graph, its corresponding ontology graph and some graph matching algorithms (especially subgraph isomorphism algorithms) from knowledge representation perspective. First, we propose a disjunctive logic programming framework to analyze the semantic graphs. Then, as family of description logics has richer syntax

²<http://www.isi.edu/isd/LOOM/PowerLoom/index.html>

and semantics and the large semantic graph has relatively big domain ontology associated with it, we review the way to analyze the semantic graph using the richer family of description logics. As far as we know, nobody is developed the technique based on knowledge representation perspective - disjunctive logic programming or description logics - for the analysis of semantic graphs. The data mining and social network analysis techniques currently used for the analysis of semantic graphs for extraction of important information are not so powerful and they lack formal syntax and semantics. We are interested to formalize whole process of extraction of information from semantic graphs from logical point of view which would be more formal approach for intelligence analysis. If we can use optimization techniques embodied in DLV system³ – a particular system based on disjunctive logic programming and/or in KAON2⁴, RACER⁵ or Pellet⁶ - reasoners for the family of description logics for the analysis of semantic graphs, that would be more formal and they can push approach further for the formal analysis of large semantic graphs.

1.5 Thesis Outline

The remainder of this thesis is organized as follows: we proceed by introducing some graph terminologies, semantic graph, ontology graph and the importance of the ontology graph in Chapter 2 along with their formalisms. We give the formal syntax and semantics of the core language of DLV system - disjunctive logic programming in Chapter 3 and describe how to represent semantic graph in disjunctive logic programs, with some running examples. We discuss in detail about finding patterns in semantic graphs and the issues related to using ontologies for finding exact/inexact matches in Chapter 4. System description, implementation details, the analysis of the results we have achieved, and the ongoing work from other areas that is related to our work are presented in Chapter 5. Chapter 6 provides the overview of standard description logic *SHIQ* and the formalisms needed to analyze semantic graphs from DL perspective. We provide the overview of several experiments on semantic graph knowledge bases performed using different DL reasoners and related work in Chapter 7 and the Chapter 8 concludes and proposes future research directions.

³<http://www.dbai.tuwien.ac.at/proj/dlv/>

⁴<http://kaon2.semanticweb.org/>

⁵<http://www.racer-systems.com/index.phtml>

⁶<http://pellet.owldl.com/>

Chapter 2

Semantic Graphs

‘A discovery is said to be an accident meeting a prepared mind.’

Albert Szent-Gyorgyi

This chapter provides the formal definition of ontology graphs and semantic graphs in detail. We first present some graph terminologies and formalize the ontology graphs and semantic graphs. Then, we introduce the scalability issues in semantic graphs and the importance of ontology graph in semantic graph analysis. The amount of the work presented here provide the formalities for the rest of the thesis.

2.1 Graph Terminologies

A graph G is a couple (V, E) , where V is a set of *vertices* (also called *nodes* or *points*) and $E \subset V \times V$ (also defined as $E \subset [V]^2$ in the literature) is a set of *edges* (also known as *arcs* or *lines*). The difference between a graph G and its set of vertices V is not always made strictly, and commonly a vertex v is said to be in G when it should be said to be in V . An edge e is a pair of vertices $\{u, v\}$. The *order* (or *size*) of a graph G is defined as the number of vertices in G and it is represented as $|V|$ and the number of edges as $|E|$ ¹. Graphs are *finite*, *infinite*, *countable* and so on according to their order.

If two vertices in G , say $u, v \in V$, are connected by an edge $e \in E$, this is denoted by $e = (u, v)$ and the two vertices are said to be *adjacent* or *neighbors*. Edges are said to be *undirected* when they have no direction, and a graph G containing only such types of edges is called *undirected*. When all edges have directions and therefore (u, v) and (v, u) can be distinguished, the graph is said to be *directed*. Usually, the term *arc* is used when the graph is directed, and the term *edge* is used when it is undirected. In this report we will mainly use directed graphs, but graph operations like *matching* can also be applied to the undirected ones. In addition, a directed graph $G = (V, E)$ is called *complete* when there is always an edge $(u, u') \in E = V \times V$ between any two vertices u, u' in the graph. The *degree* (or *valency*), denoted as $d(v)$ of a vertex is the number $|E(v)|$ of

¹In some reference in the literature, the number of vertices and edges are also represented by $|G|$ and $||G||$ respectively.

edges at v ; by our definition of a graph², this is equal to the number of neighbors of v . The graph vertices and edges can also contain information. When this information is a simple *label* (i.e. a name or number) the graph is called *labeled graph*. Other times, vertices and edges contain some more information. These are called vertex and edge *attributes* and the graph is called *attributed graph*. More usually, this concept is further specified by distinguishing between *vertex-attributed* (or *weighted graphs*) and *edge-attributed graphs*³.

A *path* between any two vertices $u, u' \in V$ is a non-empty sequence of k different vertices $\langle v_0, v_1, \dots, v_k \rangle$ where $u = v_0, u' = v_k$ and $(v_{i-1}, v_i) \in E, i = 1, 2, \dots, k$. Finally, a graph G is said to be *acyclic* when there are no cycles between its edges, independently of whether the graph G is directed or not. An acyclic graph, one not containing any cycles, is called a *forest*. A connected forest is called a *tree*⁴. The vertices of degree 1 in a tree are its leaves⁵. A *multigraph* is a pair (V, E) of disjoint sets (of vertices and edges) together with a map $E \rightarrow V \cup [V]^2$ assigning to every edge either one or two vertices, its ends. Thus, multigraphs too can have loops and multiple edges: we may think of a multigraph as a directed graph whose edge directions have been ‘forgotten’. To express that u and v are the ends of an edge e we still write $e = uv$, though this no longer determines e uniquely.

Hypergraphs are generalization of graph concepts where an edge is incident with unspecified number of vertices. Formally, a hypergraph H is a pair $H = (V, E)$ where V is a set of elements, called nodes or vertices, and E is a set of non-empty subsets (of any cardinality) of V called *hyperedges* or links. Therefore, E is a subset of $P(V) \setminus \emptyset$, where $P(V)$ is the power set of V . While graph edges are pairs of nodes, hyperedges are arbitrary sets of nodes, and can therefore contain an arbitrary number of nodes. Thus, graphs are special hypergraphs. A hypergraph is also called a set system or a family of sets drawn from the universal set V . Hypergraphs can be viewed as incidence structures and vice versa. In particular, there is a Levi (or incidence) graph corresponding to every hypergraph, and vice versa. Unlike graphs, hypergraphs are difficult to draw on paper, so they tend to be studied using the nomenclature of set theory rather than the more pictorial descriptions (like ‘trees’, ‘forests’ and ‘cycles’) of graph theory.

In the mathematical field of graph theory, a *bipartite graph* is a graph whose vertices can be divided into two disjoint sets U and V such that every edge connects a vertex in U to one in V ; that is, U and V are independent sets. Equivalently, a bipartite graph is a graph that does not contain any odd-length cycle. The two sets U and V may be thought of as the colors of a coloring of the graph with two colors: if we color all nodes in U blue, and all nodes in V green, each edge has endpoints of differing colors, as is required in the graph coloring problem. In contrast, such a coloring is impossible in the case of a non-bipartite graph, such as a triangle: after one node is colored blue and another green, the third vertex of the triangle is connected to vertices of both colors, preventing it from being assigned either color. One often writes $G = (U, V, E)$ to denote a bipartite graph whose partition has the parts U and V . If $|U| = |V|$, that is, if the two subsets have equal cardinality, then G is called a *balanced bipartite graph*.

²But this is not true for multigraphs.

³Attributed graphs are also called labeled graphs in some references, therefore these definitions are also known as *vertex-labeled* and *edge-labeled* graphs.

⁴A forest is a graph whose components are trees.

⁵Except that the root of a tree is never called a leaf, even if it has degree 1.

2.2 Semantic Graphs

The data structure we will focus on is the semantic graphs. Semantic graphs are appropriate to represent the semantical information in their nodes, i.e., they carry semantic information on their nodes and edges [Isr07]. A semantic graph is a type of network where nodes represent objects of different types (e.g., persons, papers, organizations, etc.) and links (or edges) represent binary relationships between those objects (e.g., friend, citation, authorship, etc.). In contrast to the usual mathematical description of a graph, semantic graphs have different types of nodes, and in general, different types of links. Technically speaking, a semantic graph is a network of *heterogeneous* nodes and links [BERC05]. A semantic graph is a powerful representation structure which can encode semantic relationships between different types of objects. The edge relation information provide us the information of how the two different object nodes are connected to each other and their meaning. These graphs encode relationships as *typed* link between a pair of *typed* nodes. These semantically structured graphs are also called a *relational data graph* or an *attributed relational graph*. Indeed, semantic graphs are very similar to *semantic networks*⁶ and *multi-relational networks* (MRNs) [CSH⁺05, Rod07] used in artificial intelligence and knowledge representation. Moreover, the same nodes and edge relations do appear in the semantic graphs make them the replication of simple graph formed from the several instances of some common objects. For example, a bibliography network such as the one shown in Figure 2.1 is a semantic graph, where the edges represent multiple, different relationships nodes - for example authorship (a edge connecting a person and a paper) or citation (a edge connecting two paper nodes).

The node and link types in a semantic graph are related through an *ontology graph* also known as a *schema* [ERC05]. Furthermore, each node in a semantic graph might also have a set of attributes associated with it. For example, a person node might have *age* and *weight* as attributes. Though the examples we use throughout this thesis assume that there are no such attributes for simplicity reasons, but the methodology we describe can be easily adapted to semantic graphs that contain node-associated attributes. Sometime the nodes labeled with one or more *attributes* help us to identify the specific node (e.g., *whale*) or give additional information about that node (e.g., *average_age* of whale). We will discuss about the node and edge attributes in detail later.

Besides nodes and directed links, each node of the semantic graph have a *type* (e.g., movie). The set of types is usually small compared to the number of nodes. Links may also have types, for example, the (person $\rightarrow$ movie) link may be of type “acted-in” or “directed”. Multigraphs, or graphs that may have multiple links between the same pair of nodes, are thus possible. As a result, semantic graphs have been a popular method to capture relationship information. For example, a semantic network [Qui67, Bra79] or social network [Sco00] can be regarded as a semantic graph in that it has multiple different types of relations. A *kinship network* is a semantic graph that represents human beings as nodes and various kinship relationships between them as links. WordNet [Fel98] can be regarded as a semantic graph that captures the lexical relationships between concepts. Hence, the power of semantic graphs lies not only in their structure but also in the semantic information that resides in their nodes and links. Because the semantic graphs are relatively simple but powerful and intuitive to encode relationship between objects, they are

⁶<http://www.jfsowa.com/pubs/semnet.htm>

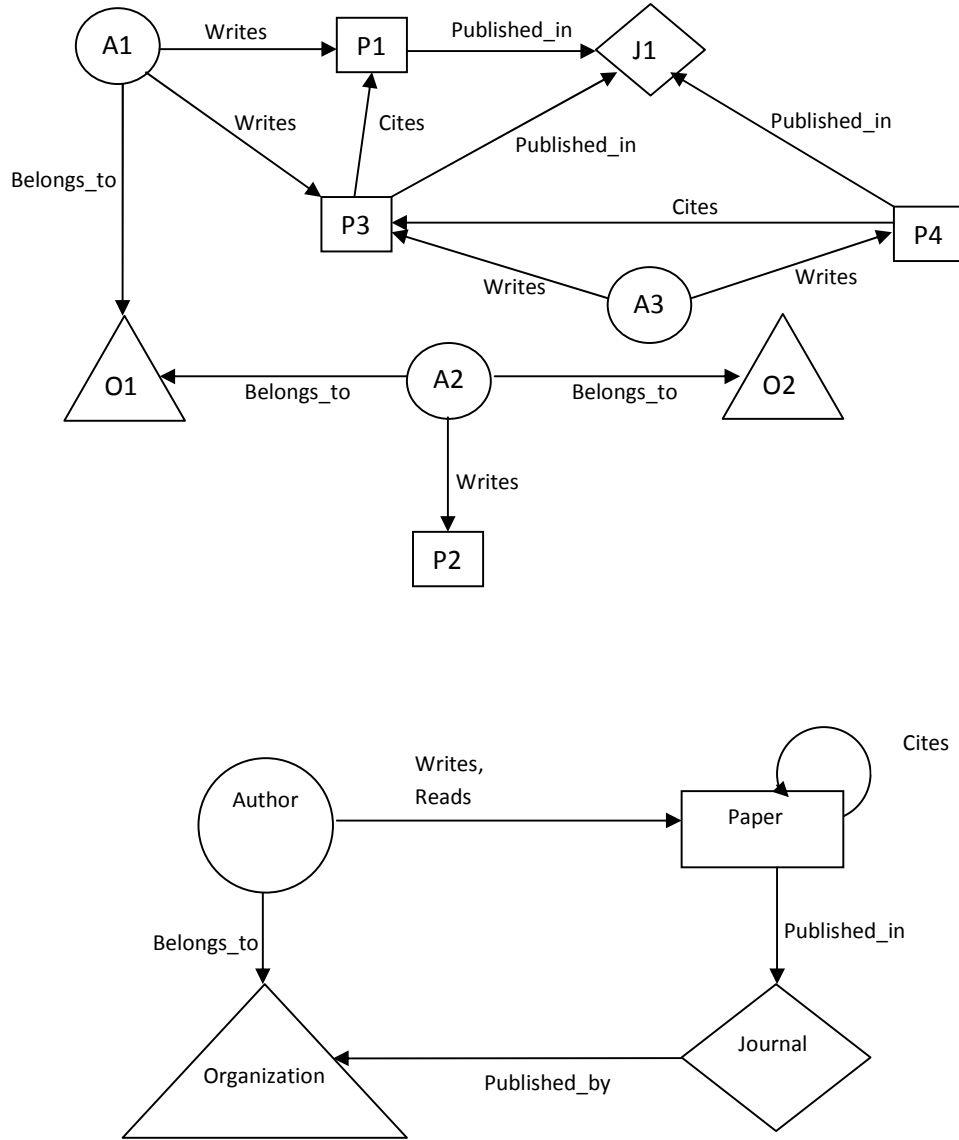


Figure 2.1: A semantic graph of bibliography domain and its corresponding ontology graph

becoming an important representation schema for analysts in the intelligence and law enforcement communities [Spa91, JRB03, SG02]. Having multiple relationship types in the data is crucial, since different relationship types carry different kinds of semantic information, allowing us to capture deeper meaning of the instances in the graph in order to compare and contrast them automatically. We can formally define semantic graph as follows:

Definition 3.1: [Semantic Graph] A semantic graph is a quintuple $G = (V, E, L, vt, et)$, where $V = \{v_1, \dots, v_n\}$ is a finite set of vertices, L is a finite set of edge labels in the semantic graph, $E \in \{(v_i, l, v_j) \subseteq V \times L \times V, \text{ where } v_i, v_j \in V, l \in L \text{ and } i, j = 1, \dots, n\}$ is a finite set of edges, vt is a mapping from V to T_V that associates a vertex type of the ontology graph with each vertex of semantic graph, and et denotes a mapping from E to T_E that associates an edge type of the

ontology graph to each edge of semantic graph.

For example, for the semantic graph shown in Figure 2.1⁷, A1, A2, A3, J1, P1, P2, P3, P4, O1 and, O2 are the finite set of vertices, Published_in, Writes, Cites, Belongs_to are finite set of edge labels, (A1, Writes, P1), (P1, Published_in, J1), $\dots$, (A2, Belongs_to, O2) are finite set of edges in semantic graph, vt associates the vertex types of the ontology graph Author with A1, A2, and A3, Paper with P1, P2, P3, and P4, Organization with O1, and O2 and Journal with J1 vertex of the semantic graph, and et associates the edge types of the ontology graph (Author, Writes, Paper) with (A1, Writes, P1), (Paper, Published_in, Journal) with (P1, Published_in, J1), $\dots$, (Author, Belongs_to, Organization) with (A1, Belongs_to, O1) edge of the semantic graph, respectively. We discuss in detail about the vertex and edge attributes later in this section.

If s is a vertex in a semantic graph, vt_s is the type for vertex s , and if k is an edge in a semantic graph, et_k is the type for edge k . It is important to note that a semantic graph does not have vertices and edges with types that are not present in its associated ontology graph. In other words, T_V and T_E (as given in the formal definition of ontology graph at Definition 3.7) are, respectively, supersets of the vertex and edge types that occur in the semantic graph G .

Generally, data for semantic graphs come from relations parsed from freeform text documents, data from web documents and/or data from relational databases [LGMF04]. The conversion of raw data spread on these different sources to the meaningful diagrams like semantic graphs is obligatory. As the manual conversion is not so effective in terms of labor and resources, automatic conversion tools must be necessary. As of [CGM04], natural language processing has matured to the point where the conversion of freeform reports to such diagrams can be largely automated and we can find such conversion tools which do their work effectively and automatically.

Unfortunately, the selection of types and attributes for both nodes and links largely depends on human expertise and is somewhat subjective and even arbitrary [BERC05]. This subjectiveness introduces biases in to any algorithm that operates on semantic graphs. For the consistency of the semantic graph with its corresponding ontology graph, the nodes and edge types should be carefully chosen.

With the meaningful attributes and types information on their vertices and edges, we can construct an *ontology graph* [ERC05] (also called a schema) whose vertices and edges are, respectively, the vertex types and edge types of one or more semantic graphs. In other words, a semantic graph contains an instantiation of the vertex and edge types that are defined in its ontology graph [ERC05]. We discuss in detail about ontology graph in Section 2.3.

Sometime the information provided by the vertex and edge relation only is not sufficient to efficiently extract the information from semantic graph. For example, as given in Figure 2.2, the time is important in the case of chasing and finally eating the rabbit by the fox. That means, in addition to the edge relation chase and eat, we need information about time of chasing and time of eating and the time time of chasing should be prior than time of eating.

Briefly, the vertex and edge relations that have the additional attributes describes the activities or the relations more clearly. As shown in the Figure 2.2, the *chase* and *eat* relations between the *fox* and *rabbit* nodes have the attribute *time*, which gives the exact time about the chasing and

⁷The nodes represented using different shapes indicate that they are of different types, i.e., nodes of different types are represented with different shapes and nodes with same types are represented with same shapes.

eating activity. Here this information is useful because the chasing operation should be preceded by the eating operation. This data can enrich semantic graph to handle the temporal (dynamic behavior) data. Again, the vertex attribute *age* gives the information about the age of particular fox and rabbit [Sil06]. By utilizing this information, we can find that the fox is capable of chasing the rabbit. In the assumption that, the fox of very small age may not be able to chase and eat the rabbit of 6/7 years old. The attribute data in particular node may also be useful to determine the exact match node in the collection of many similar nodes in the very large graphs.

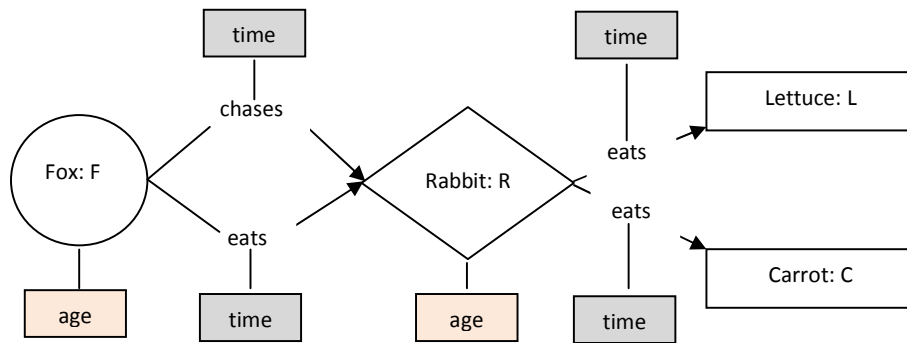


Figure 2.2: A semantic graph with vertex and edge attributes

If we consider the auxiliary information about the nodes and edge relations then our semantic graph definition looks like the one given below:

Definition 3.2: [Semantic Graph with Vertex and Edge Attributes] A semantic graph with vertex and edge attributes is a septuple $G = (V, E, L, A_V, A_E, vt, et)$, where $V = \{v_1, \dots, v_n\}$ is a finite set of vertices, L is a finite set of edge labels in the semantic graph, $E \in \{(v_i, l, v_j) \subseteq V \times L \times V, \text{ where } v_i, v_j \in V, l \in L \text{ and } i, j = 1, \dots, n\}$ is a finite set of edges, A_V is a finite set of a vertex (or node) attributes providing additional information about a particular node, A_E is a finite set of edge attributes providing auxiliary information about a particular edge relation, vt is a mapping from V to T_V that associates a vertex type of the ontology graph with each vertex of semantic graph, and et denotes a mapping from E to T_E that associates an edge type of the ontology graph to each edge of semantic graph.

We can look semantic graphs from other perspective too. Considering the edge relations, it comprises multiple different types of relations, which we call multi-relational networks (or MRNs). Having multiple relationship types in the data is important, since those carry different kinds of semantic information, which allow us to automatically compare and contrast entities connected by them [LC07]. MRNs are powerful yet simple representation mechanism to describe complex relationships and connections between individuals. For example, a bibliography network such as the one shown in Figure 2.1 is an MRN that represents authors, papers, journals, organizations, etc., as nodes and their various relationships such as authorship, affiliations, citations, etc., as links.

Definition 3.3: [Multi-relational Network] Formally, a multi-relational network is a directed labeled graph which constitutes a triple $M = (V, E, L)$ where V is a finite set of nodes, L is a finite

set of labels and $E \in \{(v_i, l, v_j) \subseteq V \times L \times V, \text{ where } v_i, v_j \in V \text{ and } l \in L\}$ is a finite set of edges. Given a triple representing an edge, the functions source, label and target map onto its start vertex, label and end vertex respectively. The function $\text{types}(V) \rightarrow \{\{l_1, \dots, l_k\}, l_i \in L, k \geq 1\}$ maps each vertex onto its set of type labels.

In our analysis of semantic graphs, we restrict the edges to be binary, but any n -ary relation can be represented by introducing an additional element reifying the relationship and n binary edges to represent each argument. The reification we will be using to represent n -ary relation in semantic graphs is similar to the reification of Resource Description Networks (RDFs) [RDFb, RDFa, GHM04].

Definition 3.4: [Inverse Edge] Let $G = (V, E, L, vt, et)$ be a semantic graph. The inverse edge set E^{-1} is the set of all edges (v_i, l^{-1}, v_j) such that $(v_i, l, v_j) \in E$.

When analyzing a semantic graph, we can consider both its forward and inverse edge sets but we have to remember that this is not the same as treating it as an undirected graph, since forward and inverse edges participate in different path types. One of the examples of an edge relation that is obviously bidirectional one is a friendship relation. If ‘A’ is friend of ‘B’, then ‘B’ is also a friend of ‘A’.

Definition 3.5: [Path] Let $G = (V, E, L, vt, et)$ be a semantic graph. A path p in G is a sequence of edges $(e_1, e_2, \dots, e_n), n \geq 1$, such that each $e_i \in E$ and $\text{target}(e_i) = \text{source}(e_{i+1})$.

For example, for the semantic graph shown in Figure 2.1, $\{(A1, \text{Writes}, P3), (P3, \text{Cites}, P1), (P1, \text{Published.in}, J1)\}$ is a path comprising nodes A1, P3, P1, and J1.

Definition 3.6: [Path Set] Let $G = (V, E, L, vt, et)$ be a semantic graph and P be set of paths in G . A set of path types $P_T(P)$ is a disjoint partition $\{pt_1, pt_2, \dots, pt_m\}, m \geq 1$, of P such that each pt_i is a set of paths $\{p_{i1}, p_{i2}, \dots, p_{in}\}, p_{ij} \in P$, that are considered to be equivalent.

Furthermore, semantic graphs are somehow related to *social graphs*. Brad Fitzpatrick⁸ defines social graph as “the global mapping of everybody and how they are related”. He went on to outline the problems with it, as well as a broad set of goals going forward.

The intuitive question arises here is: what makes a graph “semantic?” How is the semantic graph different from social networks like Facebook⁹ for example? Many people think that the difference between a social graph and a semantic graph is that a semantic graph contains more types of nodes and links. That’s potentially true, but not always the case. In fact, we can make a semantic social graph or a non-semantic social graph. The concept of whether a graph is semantic is orthogonal to whether it is social.

A graph is *semantic* if the meaning of the graph is defined and exposed in an open and machine-understandable fashion. In other words, a graph is semantic if the semantics of the graph are part of the graph or at least connected from the graph. This can be accomplished by representing a social graph using RDF and OWL¹⁰, the languages of the Semantic Web.

Today most social networks are non-semantic, but it is relatively easy to transform them into

⁸<http://bradfitz.com/social-graph-problem/>

⁹<http://www.facebook.com/>

¹⁰<http://www.w3.org/TR/owl-features/>

semantic graphs by using simple process. A simple way to make any non-semantic social graph into a semantic social graph is to use the FOAF¹¹ ontology to define the entities and links in the graph.

FOAF stands for “friend of a friend” and is a simple ontology of people and social relationships. If a social network links its data to the FOAF ontology, and exposes these linkages to other applications on the web, then other applications can understand the meaning of the data in the network in an unambiguous manner. In other words, it is now a semantic social graph because its semantics are visible to other applications.

A semantic graph is far more reusable than a non-semantic graph because it is a graph that carries its own meaning.

The semantic graph is not merely a graph with links to more kinds of things than the social graph. It is a graph of interconnected things that is machine-understandable — its meaning or semantics is explicitly represented on the Web, just like its data. This is the real way to make social networks open and reusable. Only when the semantics of data is defined and shared in an open way can any graph truly be said to be semantic. Once data around the web is defined in a machine-understandable way, a whole new world of easy, instant mashups becomes possible. Applications can start to freely and instantly mix and match each other’s data, including new data they were not programmed in advance to understand. This opens up the door to the web truly becoming a giant database and eventually an integrated operating system in which all applications are able to more easily interoperate and share data.

To untangle the criminal networks, both reliable data and sophisticated techniques are indispensable. However, intelligence and law enforcement agencies are often faced with the dilemma of having too much data, which results in an inability to find the relevant data, making the data of little value [XC05]. On the one hand, they have large volume of “raw data” collected from multiple resources: bank accounts, phone records, registration records, to name a few. On the other hand they lack sophisticated analysis tools and techniques to utilize the data effectively and efficiently [Kap06]. Today’s criminal network analysis is primarily a manual process that consumes much human time and efforts and thus has limited applicability [CGM04, New03]. We are optimistic that the efficient use of semantic graphs for the analysis of large amount of data sets from different sources help intelligence analysts solve the problems they faced for many years.

2.3 Ontology Graph

As we have already seen, semantic graphs contain instantiations of the vertex and edge types that are defined in the ontology. To precisely define semantic graphs, it is thus necessary to define what an ontology graph is and how they control the information in a semantic graph [BERC05, Bor07b, Bor07a]. Along with the meaningful attributes, semantic graphs have type information on their vertices and edges, which defines permissible relationships among the specified entities (edge types that may connect two given vertex types). By utilizing such information, we can construct the ontology graph, whose vertex and edges are, respectively, the vertex and edge types of one or more

¹¹<http://www.foaf-project.org/>

semantic graphs.

For example, a small semantic graph of bibliography domain along with its corresponding ontology graph can be found in Figure 2.1. We have four vertex types corresponding to the nodes on semantic graph: *Author*, *Organization*, *Journal* and *Paper*, respectively. Similarly, we have six edge types: *Writes*, *Reads*, *Cites*, *Belongs_to*, *Published_by*, and *Published_in*, where *Published_by* and *Reads* edge type relation are not included in the semantic graph provided above but they can be used as a edge in semantic graph created from the given ontology graph.

In every semantic graphs governed by this ontology graph, they must have only nodes and edge relations available as the vertex and edge types of the ontology graph provided. That means ontology graph monitors the vertexes and edge relations available in instantiation of semantic graph. Which means, for our example, as we don't have more than four node types, if we have the semantic graph that is instantiated considering the ontology graph given in Figure 2.1, we can only have six edge relation types and four node types. If there is a node or an edge relation available in semantic graph which is not as a node type or an edge type in corresponding ontology graph, then the semantic graph is inconsistent with respect to its ontology graph.

The ontology graph may also define one or more attribute types for each vertex type. For example, a vertex of type *person* might have the attributes "name", "date of birth", "city of birth" and "country of birth". The key attributes of a vertex should be chosen to uniquely identify a vertex in the graph. Some semantic graphs also allow vertices to have non-key attributes that may take on multiple values and are not used to uniquely identify a vertex. For a vertex of type *person*, an example of a non-key attribute might be "address". The non-key attribute "address" could have multiple values, since many people have lived at more than one address. When these vertices are published in semantic graph, vertices with the same key attribute values fuse together, i.e., a vertex with a particular set of values will be represented in a graph the once.

If there is extra node or edge of type different than the vertex and edge types associated in ontology graph, the resulting semantic graph is inconsistent with the ontology graph provided and the corresponding ontology graph no more can govern the semantic graph to which it is associated with. There are two possibilities in this situation i) update the ontology graph including the extra node or edge relation included in the semantic graph. ii) remove or prune the nodes and edge relations from the semantic graph that are not available in the corresponding ontology graph. The first task is considerably easier comparison to the second one. This is the case because ontology graph is fairly small in comparison to semantic graph and to prune semantic graph is really a difficult task.

The ontology graph plays vital role in the formalization of semantic graphs and can be formally defined as follows:

Definition 3.7: [Ontology Graph] An ontology graph is a quadruple $T = (T_V, T_E, L, I)$ where $T_V = \{t_1, \dots, t_n\}$ is a finite set of n vertex types, L is a finite set of edge labels in the ontology graph, $T_E \in \{(t_i, l, t_j) \subseteq T_V \times L \times T_V, \text{ where } t_i, t_j \in T_V, l \in L \text{ and } i, j = 1, \dots, n\}$ is a finite set of edge types, and I is the partial order binary relation " $\subseteq$ " over the finite set of vertex types T_V which is reflexive, antisymmetric, and transitive, i.e., for all a, b , and c in T_V , we have that: 1)

$a \subseteq a$ (reflexivity); 2) if $a \subseteq b$ and $b \subseteq a$ then $a = b$ (antisymmetry); and 3) if $a \subseteq b$ and $b \subseteq c$ then $a \subseteq c$ (transitivity).

For example, for the ontology graph given in Figure 2.1, Author, Paper, Organization and Journal are set of vertex types, Writes, Reads, Belongs_to, Cites, Published_in, Published_by are set of edge labels, and (Author, Writes, Paper), (Paper, Published_in, Journal), $\dots$, (Journal, Published_by, Organization) are finite set of edge types. We come to the partial order binary relation for the encapsulation of ontology hierarchy in the ontology graph later at Section Scale in Semantic Graphs at 2.5.

2.4 Importance of the Ontology Graph

An ontology is a formal specification of a set of concepts within a domain and the relationships between those concepts. It is used to reason about the properties of that domain, and may be used to define the domain. According to Tom Gruber at Stanford University “An ontology is a formal specification of a conceptualization of a domain”.

An ontology defines a common vocabulary for researchers who need to share information in a domain. It includes machine-interpretable definitions of basic concepts in the domain and relations among them. Some of the reasons to develop an ontology are: to share common understanding of the structure of information among people or software agents or even in the web, to enable reuse of domain knowledge, to make domain assumptions explicit, to separate domain knowledge from the operational knowledge, and to analyze domain knowledge. Sharing common understanding of the structure of information among people or software agents is one of the more common goals in developing ontologies [NM01].

The artificial intelligence literature contains many definitions of an ontology; many of these contradict one another. For our purpose, an ontology is a formal explicit description of concepts in a domain of discourse (classes which are sometimes called concepts), properties of each concept describing various features and attributes of the concept (slots which are sometimes called roles or properties), and restrictions on slots (facets which are sometimes called role restrictions). An ontology together with a set of individual instances of classes constitutes a knowledge base. In reality, there is a fine line where the ontology ends and the knowledge base begins. Classes are the focus of most ontologies. Classes describe concepts in the domain. For example if we look at the wine domain, a class of wines represents all wines. Specific wines are instances of this class. The Bordeaux wine in glass is an instance of the class of Bordeaux wines. A class can have subclasses that represent concepts that are more specific than the superclass. For example, we can divide the class of all wines into red, white, and rose wines. Alternatively, we can divide a class of all wines into sparkling and non-sparkling wines. Slots describe properties of classes and instances: Chteau Lafite Rothschild Pauillac wine has a full body; it is produced by the Chteau Lafite Rothschild winery. We have two slots describing the wine in this example: the slot body with the value full and the slot maker with the value Chteau Lafite Rothschild winery. At the class level, we can say that instances of the class wine will have slots describing their flavor, body, sugar level, the maker of the wine and so on.

We already know, semantic graphs is a set of vertices and edges constructed from a heterogeneous mixture of data sets, where the graph structure is constrained by an ontology. The ontology graph plays vital role in the formalization and analysis of semantic graphs. It helps us to maintain the abstract view of the semantic graphs and help us to formalize the patterns in semantic graphs. A query pattern may only reference semantic graphs that are associated with the same ontology. Pattern finding across different semantic graphs is difficult because there would not be the unique defined ontological structure for the resulting semantic graph. But what is possible for us to do is, we can able to find the patterns from the two semantic graphs constrained by the same ontology. Ontology graph provides us the useful information about the type of the each and every node in the semantic graph. There would be some cases where, more than one node attributes are same but they are of different types. In this case, finding exact match of the pattern over semantic graph depends on the type information available from the ontology graph. Therefore, type assignment to the query pattern is necessary to find the perfect match or exact match in very large semantic graphs.

2.5 Scale in Semantic Graphs

Since the number of vertex types and edge types are small compared to the number of vertices and edges, an ontology graph provides efficient structures for representing the information contained in semantic graphs albeit at their type levels [ERC05, BERC05]. If we want to represent a collection of data in a semantic graph, the choice of ontology depends on what information needs to be captured in the semantic graph, and how easily certain information needs to be retrieved. The level of detail (or scale) chosen for the ontology (choice of nodes and link types) will have a direct impact on the properties of the corresponding semantic graph.

In the simplest ontology, we have nodes of only one type. In the example of bibliography domain, this ontology is a simple network of authors without any types and two authors are connected if they wrote the same paper. In this case, the ontology is really small and it has only one node type and some edge types connect the same node by some edge type relation.

At the next finer scale, we have the authors and papers as node types. In this case, the ontology is an author connected to a paper if he wrote that paper. This is a special case of semantic graph, which has only two types of nodes with link only between the two types, called bipartite network. At the next finer scale, the semantic graph have authors, papers and journals as node types. In this case the author is connected to the paper if he wrote that paper and that paper is connected to the journal if that paper is published in that journal. In this case, the ontology graph have three node types and couple of edge types between these node types like writes, reads, published_in, etc. and so on. When the node types in the semantic graphs are increased in number, the corresponding ontology graph is also increased with the node types. The properties of ontology graphs not only depend on the node types but also depend on the edge types that will connect one node type with the related node type. When we increase the number of node types, we can represent the information in the finer level. That provide us to help analyze the information in depth. But sometime deep information in the graphs is difficult to handle or we don't need to analyze in deep

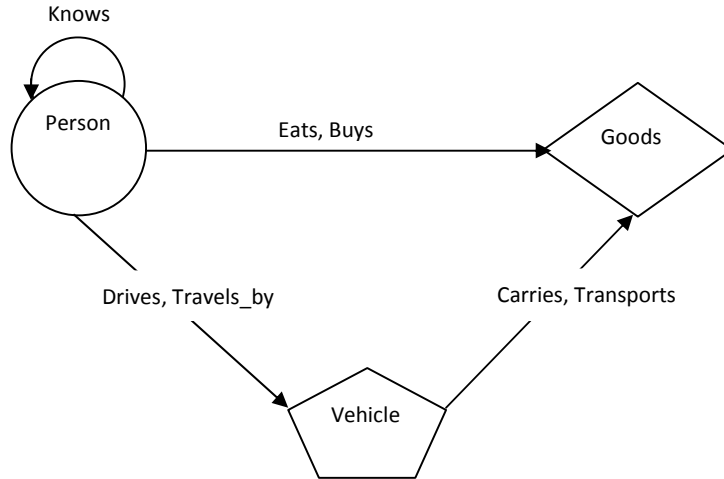


Figure 2.3: A coarser ontology graph with only one vertex type vehicle

to find the information from the graph that would be useful for our purpose. That time, coarser models are suitable. Therefore, the important thing to remember here is that coarser models lose some of the information present in finer models but can be useful for large-scale computations, such as multi-level search techniques or shallow information analysis.

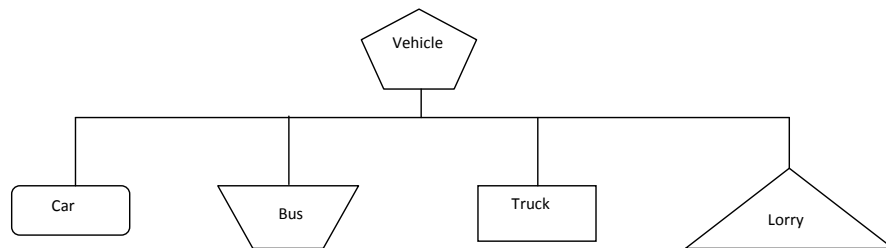


Figure 2.4: A two-level ontology hierarchy for vehicle domain

At the finest scale of a terrorist network, for example, we may have nodes of type ‘Religious Terrorist Organization’ and ‘Political Terrorist Organization’. A coarser model may aggregate nodes of these two types in to a new type ‘Terrorist Organization’ (or the aggregation may occur directly if a type hierarchy is available). Depending on what information needs to be preserved, it may or may not be important to distinguish between these two node types at the structural level of the semantic graph.

For example, in the ontology graph of vehicle domain diagrammed above at Figure 2.3, there are three vertex types: Person, Goods, and Vehicle and seven edge labels: Knows, Eats, Buys, Carries, Transports, Drives, and Travels_by. But the vertex type vehicle may have many objects which are vehicles themselves like Car, Bus, Train, Lorry, etc. Figure 2.4 gives the hierarchy of the superclass-subclass relationship of vehicle node type. When we want to distinguish between the vehicle type we are interested for, we can include the vertex types in more detail instead of the vertex type Vehicle itself as shown in Figure 2.3. In this case, the ontology graph looks like the one given in Figure 2.5 itself. The ontology graph which includes the finer detail of the nodes

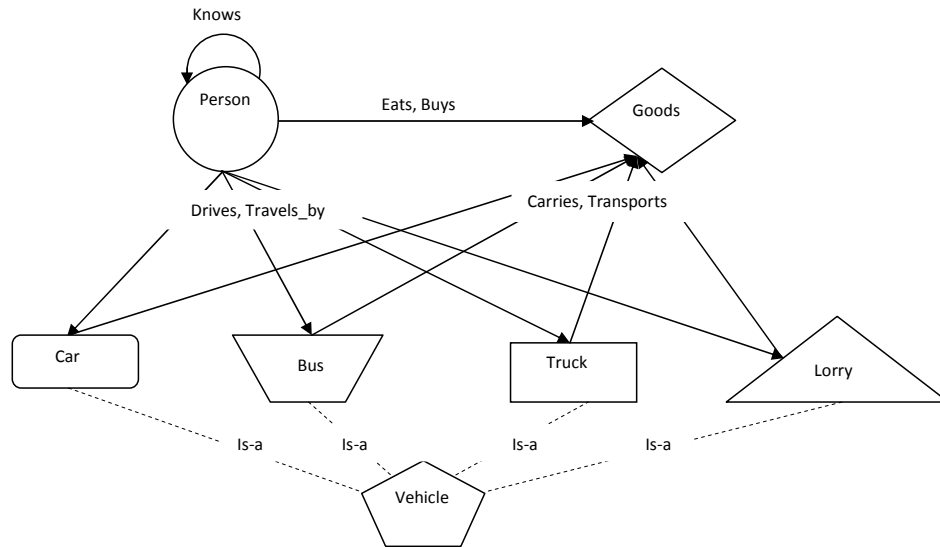


Figure 2.5: A finer ontology graph with possible hierarchy in vertex types

types its construction is called *finer ontology graph*. The one which only includes the upper level or coarse information on it is called *coarser ontology graph*.

In Homeland Security tasks, data analysis more often involves searching for outliers/abnormal patterns rather than commonplace patterns. Thus it is essential that the fine scale data is retained and the coarse scale data is used appropriately. One of the technique that can help us to get finer scale data is to use ontology hierarchy separately in addition to the ontology graph. the corresponding ontology graph of a semantic graph is coarse but the ontology hierarchy help us to maintain the finer scale data for the analysis which help us to put the ontology graph of small size along with finer data if we need.

2.6 Semantic Networks and Ontology Hierarchy

Semantic Networks originate from Quillian’s *semantic memory models* [Qui67], a graphical formalism designed to represent “world concepts” in a definitional way, that means this formalism is based on labeled graphs with different kinds of edges and nodes. Besides others, Quillian’s networks allow *subclass-superclass* edges, and/or *subject-object* edges between nodes. In general, semantic networks distinguish between *concepts* (denoted by *generic nodes*) and *individuals* (denoted by *individual nodes*), and between *subclass-superclass* edges and *property* edges. Using subclass-superclass links, concepts can be organized in a specialization hierarchy. Using property edges, properties can be associated to concepts.

Two kinds of nodes interact with each other: a property is inherited along subclass-superclass edges if not modified in a more specific class. For example, in the animal system hierarchy, birds are equipped with skin because animals are equipped with skin, and birds inherit this property because of the subclass-superclass edge between birds and animals. In contrast, although ostriches are birds, they don’t inherit the property “can fly” from birds because this property is not applicable for ostriches.

Due to the lack of proper semantics, ambiguity is prevalent in the semantic network reasoning and knowledge representation [BCM⁺03]. Following Quillian's model, a great variety of semantic network formalisms were purposed to overcome the ambiguity of semantic networks. As a consequence, new formalism are mainly involved to capture inheritance by default and to capture the non-monotonic aspects of semantic network.

The *structural inheritance networks*, introduced and implemented in the system *KL-ONE* [Bra79] was designed to cover the declarative, non-monotonic aspects of semantic network and it was rather popular although the inheritance property cause many conflicting situations. In structural inheritance networks concepts are defined using a small set of well-defined operators. The structural inheritance networks impose strict inheritance on the entire structure of a concept. The structural inheritance networks also distinct between conceptual and object-oriented knowledge. Later, in the alternative of semantic networks, the frame system and conceptual graph came to play the vital role in the knowledge representation and reasoning which were means to overcome the scarcity of proper semantics of semantic networks. *KL-ONE* is the first implementation of these ideas.

Introducing ontology hierarchy in the semantic graphs means somehow encapsulating the semantic network hierarchy with the semantic graphs. As semantic network has all the superclass-subclass relations and individual instance in a single hierarchy, there is only the superclass-subclass hierarchy in our ontology graph hierarchy.

2.7 Some Issues in Ontology-Assisted Querying

The goal of the ontology assistance is to maintain the graph query paradigm by providing the separation between ontology and schema. By doing this, ontology and schema can be developed independently and provide supports to use of personal ontologies. Typical schema-based graph database systems enable analysts to formulate queries using terms from a schema. The other goal is to exploit domain terminologies and semantics. The use of ontology help us to assist our work by facilitating the operations like subsumption, transitivity and composite classes [SBF⁺07]. Ontology-assisted query can be used for the inference to create semantically consistent models and we can further impose semantics on the graph model. Along with these properties, subclass relationships imply valid relationships among ontology elements and the ontology- schema mapping imply equality between ontology and schema elements. In this case, ontology hierarchy works as a virtual schema, which is composed of an ontology, a graph schema, and mappings between them. A software system can then assist the analysts by extracting the predicates and terms from queries, and in conjunction with the ontology and a reasoner, produce a set of corresponding graph queries that contain only terms from the graph schema. This approach enables intelligence analysts to focus on analysis that are more complex while the ontology-assisted query capability performs lower level reasoning. A distinction is maintained between the ontology reasoning and graph query systems to i) take advantage of the performance of graph query engines while exploiting the semantics of the ontologies, ii) provide multiple analysts with an explicit and consistent semantic model of the graph data, and iii) enable multiple analysts with different semantic models of the data to use their

own personal ontologies for analysis.

Part II

Reasoning on Semantic Graphs using DLP

Chapter 3

Representing Semantic Graphs in DLP

‘Research is what I’m doing when I don’t know what I’m doing.’

Werner Von Braun

This chapter explains the process of representing semantic graphs using the first order binary predicates which are useful for the analysis using disjunctive logic programs. First, we present the core language of disjunctive logic programming system - DLP in detail. Next, we deal in detail how to select the features for the lossless representation of the semantic graphs to capture all the information represented by semantic graphs. The information that we represented in the first order predicates needs to be carefully chosen because of their use in the pattern induction. We provide the concept of inductive learning for pattern induction from given knowledge base at the last section of this chapter. The representation mechanism introduced in this chapter provide us the standard representation system in the rest of the thesis for the analysis of semantic graphs.

3.1 Disjunctive Logic Programs and DLP System

Disjunctive Logic Programs (DLP) [CFPL06] are logic programs where disjunction is allowed in the heads of the rules and negation may occur in the body of the rules. Disjunctive logic programs under answer set semantics are very expressive [LPF⁺06]. It was shown in Eiter et al [EGM97] and Gottlob [Got94] that, under this semantics, disjunctive logic programs capture the complexity of Σ_2^P . As Eiter et al. [EGM97] showed the expressiveness of disjunctive logic programming has practical implications, since relevant practical problems can be represented by disjunctive logic programs, while they cannot be expressed by logic programs without disjunctions, given current complexity beliefs.

DLP is a particular deductive database system [ELM⁺98], based on disjunctive logic programming implementing disjunctive logic programs with constraints, true negation [GL91] and queries.

DLV offers front-ends to several advanced knowledge representation formalisms like planning, abductive diagnosis, etc. As DLV is a disjunctive datalog system, it combines databases and logic programming (hence the name!). For this reason, DLV can be seen as a logic programming system or as a deductive database system. In order to be consistent with deductive database terminology, the input is separated into the extensional database (EDB), which is a collection of facts, and the intensional database (IDB), which is used to deduce facts.

Moreover, DLV system is the state-of-the-art answer set solver for extended logic programs, with no function symbols [LPF⁺06]. It has integer, arithmetic and comparison built-ins and supports answer set generation and brave and cautious reasoning. It has many features which are different to prolog like evaluation independent order of rules, no limitation as far as recursion is concerned, and it has declarative semantics instead of procedural semantics which makes it more expressive than prolog. Furthermore, it has “Guess and Check” paradigm [LPF⁺06] which means the disjunctive rules “guess” a solution candidate and the integrity constraints check its admissibility, possibly using auxiliary predicates defined by normal stratified rules. In other words, disjunctive rules defines the search space of the problem and the integrity constraints prune illegal branches. We will be using DLV to analyze semantic graphs to extract important information useful for intelligence analysis. Before going further, we present the syntax and semantics of disjunctive logic programming in detail below:

3.1.1 Syntax

In this section, we provide a formal definition of the syntax and the semantics of the kernel language of DLV system, which is disjunctive datalog under the answer set semantics [GL91, LPF⁺06] which involves two kinds of negation. Following prolog’s conventions, strings started with uppercase letters denotes variables, while those starting with lowercase letters denote constants. In addition, DLV system also supports positive integer constants and arbitrary string constants, which are embedded in double quotes. A *term* is either a variable or a constant.

An *atom* is an expression $p(t_1, \dots, t_n)$, where p is a predicate of arity n and $t_1, \dots, t_n$ are *terms*. A *classical literal* l is either an atom p (in this case, it is *positive*), or negated atom $\neg p$ (in this case, it is *negative*). A negation as failure (NAF) literal l is of the form l or *not* l , where l is a classical literal; in the former case l is positive and in the latter case it is negative. Unless state otherwise, by literal we mean a classical literal. Given a classical literal l , its complementary literal $\neg l$ is defined as $\neg p$ if $l = p$ and p if $l = \neg p$. A set L of literal is said to be consistent if, for every literal $l \in L$, its complementary literal is not contained in L .

Moreover DLV system provides *built-in predicates* such as comparative predicates equality, less than, greater than ($=, <, >$), and arithmetic predicates like addition and multiplication ($+, \times$).

A *disjunctive rule* (rule, for short) r is a formula

$$a_1 \vee \dots \vee a_n \text{ :- } b_1, \dots, b_k, \text{ not } b_{k+1}, \dots, \text{ not } b_m. \quad (1)$$

where $a_1, \dots, a_n, b_1, \dots, b_m$ are classical literals and $n \geq 0, m \geq k \geq 0$. The disjunction $a_1 \vee \dots \vee a_n$ is the *head* of r , while the conjunction $b_1, \dots, b_k, \text{ not } b_{k+1}, \dots, \text{ not } b_m$ is the *body* of r . A rule without head literals (i.e. $n = 0$) is usually refereed to as *integrity constraints*. A rule having precisely one head literal (i.e. $n = 1$) is called a *normal rule*. If the body is empty (i.e., $k = m = 0$),

it is called a *fact*, and we omit the ‘:-’ sign.

If r is a rule of form (1), then $H(r) = \{a_1, \dots, a_n\}$ is the set of literals in the head and $B(r) = B^+(r) \cup B^-(r)$ is the set of the body literals, where $B^+(r)$ (the positive body) is $\{b_1, \dots, b_k\}$ and $B^-(r)$ (the negative body) is $\{b_{k+1}, \dots, b_m\}$.

A *disjunctive datalog program* (alternatively *disjunctive logic program*, *disjunctive deductive database*) P is a finite set of rules. A *not-free* program P (such that $\forall r \in P : B^-(r) = \emptyset$) is called *positive* and a $\vee$ -free program P (such that $\forall r \in P : |H(r)| \leq 1$) is called *datalog program* (or *normal logic program*, *deductive database*).

The language of DLV system has another construct, which is called *weak constraint*. We define weak constraints as a variant of integrity constraints. In order to differentiate clearly between them, we use for weak constraints the symbol ‘ $\sim$ ’ instead of ‘:-’. Additionally, a *weight* and a *priority level* of the weak constraint are specified explicitly.

Formally, a weak constraint wc is an expression of the form

$$\sim b_1, \dots, b_k, \text{ not } b_{k+1}, \dots, \text{ not } b_m.[w : l].$$

where for $m \geq k \geq 0$, $b_1, \dots, b_m$ are classical literals, while w (the weight) and l (the level or layer) are positive integer constants or variables. For convenience w and/or l might be omitted and are set to 1 in this case.

A DLV program P is a finite set of rules (possibly including integrity constraints) and weak constraints. A rule is *safe* if each variable in that rule also appears in at least one positive literal in the body of that rule, which is not a comparative built-in. A program is *safe* if each of its rules is *safe* and we only consider the *safe* programs.

A term is called *ground* if no variables appear in it. A ground program is also called a *propositional program*.

3.1.2 Semantics

The *semantics* of DLV programs extends the answer set semantics of disjunctive logic programs, originally defined in Gelfond and Lifschitz [GL91], to deal with weak constraints:

- *Herbrand universe*: For any program P , let U_P be the set of all constants appearing in P . In case no constant appears in P , an arbitrary constant ψ is added to U_P .
- *Herbrand Literal Base*: For any program P , let B_P be the set of all ground (classical) literal constructible from the predicate symbols appearing in P and the constants of U_P (for each atom p , B_P contains also the strongly negated literal $\neg p$).
- *Ground Instantiation*: For any rule r , $Ground(r)$ denotes the set of rules obtained by applying all positive substitutions σ from the variables in r to elements of U_P . In a similar way, given a weak constraint w , $Ground(w)$ denotes the set of weak constraints obtained by applying all possible substitutions σ from the variables in w to the elements of U_P .
- *Answer Sets*: For every program P , we define its answer sets using its ground instantiation $Ground(P)$ in three steps: first we define the answer sets of positive disjunctive datalog programs, then we give a reduction of disjunctive datalog programs containing negation

as failure to positive ones and use it to define answer sets of arbitrary disjunctive datalog programs, possibly containing negation as failure. Finally, we specify the way how weak constraints affect the semantics, defining the semantics of general DLV programs.

An *interpretation* I is a set of ground literals, that is, $I \subseteq B_P$ with respect to a program P . A *consistent interpretation* $X \subseteq B_P$ is called *closed* under P (where P is a positive disjunctive datalog program), if for every $r \in \text{Ground}(P)$, $H(r) \cap X \neq \emptyset$ whenever $B(r) \subseteq X$. An interpretation $X \subseteq B_P$ is an *answer set* for a positive disjunctive datalog program P if it is *minimal* (under set inclusion) among all (consistent) interpretations that are closed under P .

Example 3.1: The *positive program* $P_1 = \{a \vee \neg b \vee c\}$ has the answer sets $\{a\}$, $\{\neg b\}$ and $\{c\}$. Its extension $P_2 = \{a \vee \neg b \vee c; :-a\}$ has the answer sets $\{\neg b\}$, and $\{c\}$. Finally, the positive program $P_3 = \{a \vee \neg b \vee c; :-a; \neg b:-c; c:-\neg b\}$ has the single answer set $\{\neg b, c\}$.

The *reduct* of a ground program P with respect to a set $X \subseteq B_P$ is the positive ground program P^X , obtained from P by

- deleting all rules $r \in P$ for which $B^-(r) \cap X \neq \emptyset$ holds.
- deleting the negative body from the remaining rules.

An *answer set* of a program P is a set $X \subseteq B_P$ such that X is an answer set of $\text{Ground}(P)^X$.

For example, suppose we are not aware of being told a joke. In this case, the correct datalog program looks like this:

laugh : $\neg \text{joke}$.

The program itself does not express that joke is false, but the so-called Complete World Assumption (CWA) [CL94, Lif85] does. It is one of the foundations DLV bases its computations on and says that everything about which nothing is known is assumed to be false. Therefore, the model for this program is $\{\}$. (This means that there is a model but it is empty. It is also possible that for a given program there is no model.)

When we use the CWA in one of our programs, we basically view the DLV system as a deductive database system, since we do not ask for what is logically right, but what we can usefully derive from our facts base. Following this approach, we can perform queries on the existing data (the facts base), derive (and “store”) new data using queries (or rules), which again can be used to deduce even more data, and, using the CWA, even ask queries as to what is not in (or derivable from) our database.

In general, by posing a query one looks for ground substitutions such that the substitution applied to the query is true or validated by the rest of the program. Since a disjunctive datalog program may have more than one model, there are different reasoning modes called *brave reasoning* and *cautious reasoning* to decide whether a substituted query is satisfied.

3.2 Modeling Semantic Graphs in First Order Logic

In the rest of this part, we describe a mechanism for identifying useful patterns and abnormal instances in a semantic graph. Our goal is to develop an disjunctive logic program based framework that utilizes the information provided by the semantic graph to identify useful patterns and abnormal behaviors. The central questions we need to ask regarding this are: i) how can we define and find the useful patterns in the context of semantic graphs where each node is different from others, and ii) how can we define and measure the abnormality in a context where each node is different from every other (in the sense that they have different names, occupy different positions and connect to different nodes through different links in the graph).

The plausible answer for these questions are: i) a pattern is defined with the help of the ontology graph and the defined pattern should have continuous path along its root node to its leaf node. The root node for the specific pattern is the node where the pattern starts and the leaf node for the pattern is the node where the specific path of the pattern ends, and ii) a node is abnormal if there are no or very few nodes in the graph similar to it based on some similarity measure. Given this definition, the next question that need to be asked is: how we measure the similarity of the nodes? The following is a list of possible proposals that node similarity could be based on using the information provided by a semantic graph:

- The type of the nodes
- The type of directly connected nodes
- The type of nodes connected indirectly via path of certain length
- The type of their links or edges
- The type of indirectly connected links via path of certain length
- ...
- etc.

However the proposal seems to be capture only partial connectivity information about the nodes and might not be able to capture the deeper meaning of the nodes. One real interest is to find the nodes that contain abnormal semantics or play different roles in the graph. Given that our goal is to build a logic based framework, that model the semantics of the nodes by adapting the concept of *syntactic semantics*. This concept is proposed by Rappaport [Rap02], who claims that the semantics is embedded inside syntax, and, hence syntax is suffice for the semantic enterprise. To extend this concept to our domain, we claim that the semantics of a node is not only determined by its label or name, but also can be determined by the role it plays in the network or its surrounding network structure. The relation to the surrounding network in our case is determined by the binary edge relation that provide the connection between the adjacent pair of nodes connected to any specific node.

Suppose we are given a semantic graph and we are asked to find the relationship between two nodes ‘A’ and ‘B’. In the simplest case, the relationship is manifested as an edge in the graph. These

nodes may be related through the composition of simple edge: 'A is related to B'. If these nodes are not adjacent one, they may be related to each other via connection through other intermediate nodes. In the case, where A and B has the intermediate node X, the the path is described through the help of two edges: 'A is related to X' and 'X is related to B'. In this case the relationship might be encapsulated as a path in the graph. In the large semantic graphs, there may be large number of such type of paths exists due to the large volume of data and they have the same superset ontology graph. These instantiation of the ontology graph gives the large number of relations only different to every other in terms of data they carry in their nodes. When we represent these relations in the DLV like syntax, for above relation 'A' and 'B', we get 'A - relatedto - X - relatedto - B'. By combining and condensing the information provided by those paths, one can come up with the description of nodes similar to the ones we given before.

This observation motivates a key idea of our approach for using these paths to capture the semantics of nodes. To do this, we represent the each edge of the network with its edge nodes in a standard logical notation by representing nodes as constants and links via binary predicates. In this case, the path 'A- relatedto - X - relatedto - B' can be represented in first order binary predicates as ' $relatedto(A, X) \wedge relatedto(X, B)$ '. This logical expression characterizes the meaning of the nodes 'A', 'X' and 'B'. As we have seen , we represent our relations as extensional database(EDB)¹, which has fixed knowledge (facts), and looks like standard relational tuples, rather than propositional atoms. For example in ' $relatedto(A, B)$ ', the 'relatedto' is called a predicate symbol or relation symbol (carry the semantics of edge relation for our case) and the parts with in the parenthesis 'A' and 'B' are constants or arguments. As the standard convention of DLV system, all the constants must be begin with lowercase letters and all the variables must be started with the uppercase letters.

From above observations, our standard approach to capture a semantic profile of a node is by treating all paths in the network as binary features, assigning true to the paths the given nodes participate in and false to the ones it does not. With this approach, we represent semantic profile of nodes in to the propositional representation. We can represent in binary predicates the facts (or knowledge base) in such a way that the edge names contribute the predicate name and the end nodes contribute the arguments of the binary predicates. For the semantic graph shown in Figure 2.1, the binary predicate representation is given as below:

<i>writes(a1, p1).</i>	<i>belongs_to(a2, o2).</i>
<i>writes(a1, p3).</i>	<i>cites(p3, p1).</i>
<i>writes(a2, p2).</i>	<i>cites(p4, p3).</i>
<i>writes(a3, p3).</i>	<i>published_in(p1, j1).</i>
<i>writes(a3, p4).</i>	<i>published_in(p3, j1).</i>
<i>belongs_to(a1, o1).</i>	<i>published_in(p4, j1).</i>
<i>belongs_to(a2, o1).</i>	

This representation supports our idea to represent the whole semantic graph in terms of binary predicates with one edge relation with two end nodes gives one predicate. That predicate exhibit

¹The EDB is the collection of facts only or the unchanged knowledge base represented in terms of predicates with constants as arguments or simply constants.

the relation bounded by the edge relation label indicated in the semantic graph.

We know semantic graphs are associated with the corresponding ontology graph. Ontology graphs are particularly helpful to know the general information about the semantic graphs. Particularly, these graphs reveal the information about how many node types and edges types are there in the semantic graph and how they are interconnected to each other. Ontology graph can have at least the number of edge and node types that are available in the semantic graphs and even more. That helps us to restrict the semantic graphs in a way what to include in the semantic graph and what to ignore. Furthermore, type information in the ontology graph has other important meaning too.

The type information plays particularly important role in the large semantic graphs which have different types of vertexes. The importance of the type information depends on the type of the information the semantic graphs represent and the purpose of the representation. When we do not need to care about the exact information carried by the particular node, we may neglect the type information of that node.

In the very large semantic graphs, type information is very necessary for finding the exact match in the list of several similar matches which we describe in detail in Chapter 4. Here we discuss about how to represent the type information in the first order binary predicate logic formalism. Generally, the type information of the vertexes of the semantic graphs are represented by binary predicate where the predicate name is written as type and the first argument contributes by the node name and the second argument comes from the corresponding vertex type in ontology graph for the node in semantic graph. For the semantic graph diagrammed in Figure 2.1, the type information is represented in binary predicates as given below:

```

type(a1, author).
type(a2, author).
type(a3, author).
type(o1, organization).
type(o2, organization).
...
etc.

```

This representation seems intuitive that we can have only one predicate name with different arguments in it. The type predicate can only distinguish particular node of particular type by matching the arguments on it with the type relation available in semantic graph.

The ontology hierarchy embedded in the ontology graph can also be represented in first order predicate relation by transforming the superclass-subclass relation in terms of rules. For the ontology hierarchy defined in Figure 2.5 at Section 2.5, the vehicle domain ontology hierarchy relations are defined in terms of rules as given below:

$type(X, vehicle) : \neg type(X, car).$
 $type(X, vehicle) : \neg type(X, bus).$
 $type(X, vehicle) : \neg type(X, truck).$
 $type(X, vehicle) : \neg type(X, lorry).$

From the above rules, we can get the information that when something is of type *car*, *bus*, *truck* or *lorry* than that is also of type *vehicle*. This information is particularly useful when there is inexact match exists: like we are searching for a car but we get bus in the answer. This answer is very similar to the car as it is far more nearer match than finding a teddy bear. In this case, we have to extract the answer from the upper level of ontology by finding the superclass that match the requirement similar to the one extracted from the query.

Our next step is to find out the *common* (or *immediate*) rules that may true provided the knowledge base. As semantic graphs has different types of edge types and vertices types, many relations might be true. For example, given the knowledge base below:

$father(luke, harry).$
 $father(harry, sham).$
 $mother(venessa, harry).$
 $mother(elisha, sham).$

We can find *immediate relations* that the knowledge base infer by finding the relations that can immediately seen to be implied from the provided knowledge base:

$parent(X, Y) : \neg mother(X, Y).$
 $parent(X, Y) : \neg father(X, Y).$

These immediate relations helps us to find the obvious relations that holds on the semantic graphs. Here we have to remember that these relations can help us to write pattern queries for finding the common and abnormal patterns in the graph. For example,

$grandparent(X, Y) : \neg parent(X, Z), parent(Z, Y).$

can be written with the help of the immediate parent relation we inferred from the knowledge base. The most prominent area which deal with the induction of rules from the given knowledge base to help ease in the formation of highly complex rules to extract information from the knowledge base provided is studied in inductive logic programming, which we describe in detail later at Section 3.3.

Furthermore, immediate relations are usually helpful to minimize the rules needed to encode some pattern queries. For example, if we write the query to find *grandparent* of *sham* directly from the knowledge base, then we end up with the four possible combination of rules to find the grandparent from the *father* and *mother* relations.

Here, we have to find the *common pattern* queries so that we can extract the similar patterns available in the given semantic graph. Finding common patterns is largely depend on the purpose of the semantic graph analysis. For example, by using common pattern queries we can find the

patterns like “Every author who writes paper and his paper is published in some journal”. That means, we are interested in finding the pattern $writes(X, Z), published_in(Z, Y)$, if we express our purpose in binary predicates.

After finding the common patterns, they help us to focus on analyzing the abnormal behavior of the nodes. Here we can find the nodes that are not exhibiting common patterns (or abnormal ones respective to our common pattern query) from the representations similar to this representation:

$$\begin{aligned} normal(X, Z) &: \neg writes(X, Z), published_in(Z, Y). \\ abnormal(X, Z) &: \neg writes(X, Z), published_in(Z, Y), not\ normal(X, Z). \end{aligned}$$

Further, we can use the integrity constraints to filter out the unnecessary results produced by our disjunctive logic program. In DLV, we have two different type of constraints, namely: *integrity constraints* and *weak constraints* to tackle these problems [BFK⁺98]. Integrity constraints are rules with no head and they filter out the results that satisfies the body of that empty head rule. Weak constraints serve the purpose by allowing us to define the weight measure according to the importance of the model. In the presence of weights, best model minimize the sum of the weights of the violated weak constraints. While standard constraints (integrity constraints or strong constraints) always have to be satisfied, weak constraints express desiderata, i.e., they should be satisfied if it is possible, but their violation does not “kill” the models [BFK⁺98].

Our approach motivates the search for a more condensed feature set that still can adequately capture the role semantics of instances. We can generalize this relation to capture such type of patterns available in whole knowledge base by exploiting them using *variable relaxation*.

Variable relaxation is the process which replace the constants with the variables for finding the paths that go through the same nodes. This view would consider the following two paths as equivalent: $cites(P3, P1) \wedge published_in(P1, J1)$ and $cites(P4, P3) \wedge published_in(P3, J1)$. Alternatively, when we consider two paths as equivalent if they go through the same nodes, this view would consider the following two paths as equivalent: $spouse(Person1, Person2) \wedge act_in(Person1, Movie1)$ and $wife(Person1, Person2) \wedge directed(Person1, Movie1)$.

Given the generalization strategies, the next question is how we can generate a meaningful and representative set of path types? Regarding the path in Figure 2.1, $cites(P3, P1) \wedge published_in(P1, J1)$ (which means “paper P3 cites paper P1 which is published in journal J1”) as an example, we find there are three ground elements in this path: P1, P3, and J1. If we relax one of its elements, say J1 to a variable X1, then we get a new relaxed path type $cites(P3, P1) \wedge published_in(P1, X1)$ which now represents a more general concept: “paper P3 cites paper P1 that is published in some journal”. Further relaxing, we could also generalize element P1 which would give us $cites(P3, Y1) \wedge published_in(Y1, X1)$ which means: “paper P3 cites some paper which is published in some journal”. In this way we can generalize any combination of nodes in a path to arrive at a more general path type. These path types still convey meaning but do so at a more abstract level. This makes them more useful as features to compare and contrast different instances and nodes.

The variable relaxation approach we described above is similar to the inductive learning technique used in logic programming. It is generally known that induction means reasoning from

specific to general. In the case of inductive learning from examples, the learner is given some examples from which general rules or theory underlying the examples are derived. The inductive learning technique is quite useful in our framework for the automatic pattern induction and we describe in detail about it in later section.

To serve our purpose, we are not really interested in finding only the common patterns available in the semantic graph, as they really don't useful for finding the abnormal or suspicious nodes in the large scale semantic graphs used for the intelligence and homeland security purposes. But these common patterns are useful for finding the abnormal patterns. As we can find from the above rules that $abnormal(X, Z)$ is true when an author writes a paper but it is not published in any journal comparing to other authors who published their papers in at least one journal. In our case at Figure 2.1, 'a2' is the node that to be considered for the further analysis as it exhibits abnormal behavior.

In summary, we first represent semantic graph in terms of first order binary predicates then we write rules to find immediate relations along with common patterns. After that, we write pattern queries to find the similar patterns that are matched with the pattern that is already defined and negating the pattern specification to extract abnormal patterns after excluding the common patterns from our analysis. The detailed system description is shown in Figure 5.1 at Chapter 5.

3.3 Inductive Logic Programming

As data and the relations used in the semantic graphs include representation of people, organization, objects, and actions and many types of relations between them, and the pattern finding mechanism for the analysis of semantic graphs plays vital role in the analysis of suspicious or abnormal behavior of any object, the most widely studied methods for inducing/learning relational patterns are those in *inductive logic programming* (ILP). ILP is the study of learning methods for data and rules that are represented in first-order predicate logic [MMT⁺02][LD94]. As an example consider the following rules, written in prolog like syntax, that define the *uncle* relation:

$$\begin{aligned} uncle(X, Y) &: \neg brother(X, Z), parent(Z, Y). \\ uncle(X, Y) &: \neg husband(X, Z), sister(Z, W), parent(W, Y). \end{aligned}$$

The goal of inductive logic programming is to infer rules of this sort given a database of background facts and logical definitions of other relations. For example, ILP system can learn above sort of rules (the *target predicate*) given a set of positive and negative examples of uncle relationships and set of the facts for the relations parent, sister, and husband (the background predicates) for the member of the given family. Alternatively, rules that logically define the brother and sister relations could be supplied and these relationships inferred from a more complete set of facts about only the basic predicates: *parent*, *spouse* and *gender*.

Background knowledge plays a vital role in relational learning, where the task is to define, from given examples, an unknown relation (i.e., the target predicate) in terms of (itself and) known relations from the background knowledge. If the hypothesis language of the relational learner is the language of logic programs, then learning is, in fact, logic program synthesis. In ILP systems,

the training examples, the background knowledge and the induced hypotheses are all expressed in a logical program form, with additional restrictions imposed on each of the three languages. For example, training examples are typically represented as ground facts of the target predicate, and most often background knowledge is restricted to be of the same form.

One of the early systems that made use of relational background knowledge in the process of learning structural descriptions was INDUCE².

A search of the hypothesis space in inductive learning can be performed bottom-up or top-down manner. Generalization techniques search the hypothesis space in a bottom-up manner: they start from the training examples and search the hypothesis space by using generalization operators. Bottom-up learners start from most specific clause that covers a given example and generalize the clause until it cannot further be generalized without covering negative examples.

In bottom-up hypothesis generation, a generalization c' of clause c ($c' < c$) can be obtained by applying a θ -subsumption-based generalization operator. Generalization operators perform two basic syntactic operations on a clause:

- apply an inverse substitution to the clause, and
- remove a literal from the body of the clause.

Preprocessing step handles the missing arguments values in training examples, and when no negative examples are given, the generation of negative examples. Different ways of generating negative examples are possible. Most frequently systems use the so-called generation under the closed world assumption. In this case, all examples which are not given as positive are generated and labeled negative. This method is only appropriate in exact, finite domains where all positive examples are in fact given. The user must be more careful when applying the different methods for negative example generation and must be at least be able to interpret the results accordingly.

One of the technique used for the generalization techniques to find possible pattern in the idea of *inverse resolution*, introduced as a generalization technique to ILP, is to invert the resolution rule of deductive inference [Rob65], e.g., to invert the SLD-resolution proof procedure for definite programs. The basic resolution step in propositional logic derives the resolvent $p \vee r$ given the premises $p \vee \neg q$ and $q \vee r$. Given a wff W , an inverse substitution θ^{-1} of a substitution θ is a function that maps terms in $W\theta$ to variables, such that $W\theta\theta^{-1} = W$.

For example, if $c = \text{daughter}(X, Y) \leftarrow \text{female}(X), \text{parent}(Y, X)$ and the substitution is $\theta = \{X/\text{mary}, Y/\text{ann}\}$, then

$$c' = c\theta = \text{daughter}(\text{mary}, \text{ann}) \leftarrow \text{female}(\text{mary}), \text{parent}(\text{ann}, \text{mary})$$

By applying the inverse substitution $\theta^{-1} = \{\text{mary}/X, \text{ann}/Y\}$ the original clause c is obtained.
 $c = c'\theta^{-1} = \text{daughter}(X, Y) \leftarrow \text{female}(Y), \text{parent}(Y, X)$

In the general case, inverse resolution is substantially complex. It involves the places of terms in order to ensure that the variables in the initial wff W are appropriately restored in $W\theta\theta^{-1}$.

To find all valid available patterns, we input all the background facts and try to validate the conjunction of the two or more predicates with arguments as variables, and try to accommodate

²<http://www.mli.gmu.edu/software.html>

as many predicates as possible till the conjunction is valid.

Chapter 4

Finding Patterns in Semantic Graphs

‘Research is formalized curiosity. It is poking and prying with a purpose.’

Zora Neale Hurston

This chapter investigates the process of pattern finding in semantic graphs in detail. We first introduce structure of pattern query and importance of patterns and abnormal instances. Next, we discuss in detail the pattern analysis in semantic graphs and pattern matching process. Furthermore, we deal with the importance of ontology graph for automated pattern induction. The formalisms in this chapter guide us in the overall performance analysis of pattern finding framework proposed in next chapter.

4.1 Structure of a Pattern Query

A pattern query is a labeled graph that describes the structure and content of the semantic graph entities to be matched. Vertices and edges in the query corresponds to the vertices and edges in the semantic graph. Every query must have at least one vertex and every edge must have vertices at both ends. A pattern query must be a connected graph, with a single vertex being the simplest pattern query.

The pattern queries can be represented graphically. The example query below shows a query that matches two objects connected by a direct link. Every vertex and edge in a pattern query has

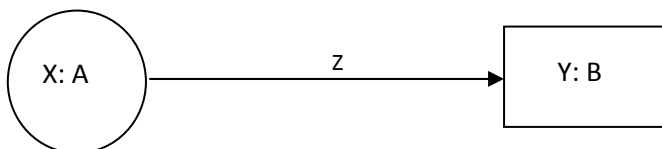


Figure 4.1: Example query structure

a unique name within the context of the query. These names have no intrinsic meaning and serve mostly to provide meaningful labels to the user in a query results [Jen07] [KABK08] [BIJ02].

For the example queries in this report, we use either arbitrary alphabetical labels or semantically meaningful names for ease in understanding the query. For example, if we define a query vertex that matches semantic graph objects with the type value *vertextype* = *movie* and attribute *vertexname* = *Casablanca*, we might call this a *movie* type vertex. Similarly, a query edge defined to match links with the attribute value *edgetype* = *acted-in* will be called *acted-in* in the query. By convention, when using alphabetical labels, we use letters at the end of alphabet such as X, Y, and Z to label vertices and edges and we use letters at the beginning of the alphabet such as A, B, and C to indicate vertex types in the query pattern. For example, a vertex with $X : A$ has meaning – a vertex with attribute name X and which has type of A. Although the given model above uses only directed links, our system supports both directed and undirected (bi-directional in our case, combination of inverse edges and directed edges pointed to both sides) edges.

4.2 The Importance of Abnormal Instances

There are variety of things one can determine from a graph. For example, one can try to identify central nodes, find frequent subgraph patterns, or learn interesting property.

The goal of our work is different. We do not focus on finding central instances or pattern level discovery. Instead we try to determine certain instances, individuals or patterns in the graph that look different from others. There are some reasons to focus on discovering these type of instances in semantic graphs. First we believe that these types of instances can play potentially major role in finding relative information, in the sense that something that looks different from others or from its peers has a high chance to attract people’s attention or suggest new hypothesis, and the analysis of them can potentially trigger new theories. The second reason is that this is a very challenging problem and so far we are not aware of any system that can utilize the relational information in semantic graph to perform anomaly detection effectively. The next reason is that there are number of important applications for a system that can determine abnormal nodes in a semantic graph, as given below in brief.

Application 1: Information Awareness and Homeland Security

Aftermath of the 9/11 attacks show that the implicit and explicit relationships of the terrorists involved do form a relatively large covert network, containing not only the threat individuals but also a large number of innocent people. Moreover, the persons in the network usually have a variety of connection with each other (e.g., friendships, kinships, business associations, religious associations, etc.). Consequently, it makes sense to represent all this information in a very large and complex semantic graphs. This type of data usually contains primarily innocent and benign persons who don’t need to perform certain special actions to pursue threat missions. That is to say, it is reasonable to infer that people who look typical (i.e. , who have a lot similar peers) are not likely to be malicious. Although threat individuals might try to disguise their actions to avoid detection, it is likely that subtle differences still exist, since they still need to perform unusual actions

in order to execute their missions. Such behavior is more likely to create unusual evidence, and therefore, a node with an abnormal evidence profile has a higher chance to be suspicious compared with the one that have more similar peers. Our model exploits the deeper information provided by the semantic graphs with the goal to identify the suspicious individuals. Furthermore, false positives generated by the information awareness system can cause a lot of damage to blameless people, careful observation of the results is, thus, necessary.

Application 2: Fraud Detection and Law Enforcement

Similar to the previous application, the fraud detection and law enforcement domains also offer huge amount of information about the relationships and transactions among persons, companies and organizations. Being able to identify and explain abnormal instances in such a network could be an important tool for police or investigators to detect criminals and frauds.

Application 3: General Scientific discovery

Abnormal instances may also provide some interesting ideas or hints for the scientists. In the domain of biology or chemistry, for example, genes, proteins and chemicals interacts with each other in a number of ways, and these interactions can be described by a semantic graph. One application of our analysis is to provide new insights to scientists by pointing out unusual biological or chemical connections or individuals.

Application 4: Data Cleaning

An abnormal record in a fairly accurate database can also be interesting because it might represent an error, missing record, or certain inconsistency. We can apply our system to a relational dataset (which can be represented by a semantic or relational graph), to mine for potential errors. For example, in our movie dataset, the system found that the node ‘Anjelica Huston’ was abnormal.

Our major concern for knowledge discovery system rests in the difficulty of verification. Unlike a learning system, a discovery system by definition aims at finding something that was previously unknown. Since the discoveries are previously unknown, it is generally not clear how they can be verified. This concern even more serious for systems applied to security-related problems. False positives are a very serious issue for any homeland security system, since even a system with almost perfect 99% accuracy can still result in the mislabeling of the millions of individuals when applied to a large population.

4.3 Graph Matching

Our proposed pattern finding mechanism is quite similar to the graph matching problem, particularly a *sub-graph matching*¹ [Ull76] problem. We are interested in matching the sub-graph which is generated from the query graph and the semantic graph to which we are evaluating the query pattern against. Quite often, we represent a query graph in terms of predicates with variables as

¹http://en.wikipedia.org/wiki/Subgraph_isomorphism_problem

their arguments, which leads to matching one subgraph next to other subgraph, after instantiating all the variables by the respective constants available in the knowledge base. Admittedly, the procedure for comparing two graphs involves to check whether they are similar or not. Generally speaking, we can formalize the *graph matching* problem as follows: given two graphs - the model graph $G_M = (V_M, E_M)$ and the data graph $G_D = (V_D, E_D)$, with $|V_M| = |V_D|$, the problem is to find a one-to-one mapping $f : V_D \rightarrow V_M$ such that $(u, v) \in E_D$ iff $(f(u), f(v)) \in E_M$. When such a mapping f exists, G_D and G_M are said to be *isomorphic* and this property is called an *isomorphism* property.

In semantic graph matching, we can not expect the isomorphism between semantic graph and query graph², i.e., we can not expect that all nodes and edges in the query graph would match with all nodes and edges in the semantic graph. Particularly, finding all common patterns is considered as the matching of the graph using all possible query graph after instantiating the variables for each query satisfiable respective to the semantic graph. In this case, graph matching is to find a subgraph of one graph which is isomorphic to another graph.

In *sub-graph matching*, since we have $|V_M| < |V_D|$ the goal is to find again a mapping $f' : V_D \rightarrow V_M$ such that $(u, v) \in E_D$ iff $(f(u), f(v)) \in E_M$. This corresponds to search for a small graph within a big one which is isomorphic to the another graph. Formally, *sub-graph matching* is a problem in which we have two graphs $G = (V, E)$ and $G' = (V', E')$, where $V' \subseteq V$ and $E' \subseteq E$, and in this case the aim is to find a mapping $f' : V' \rightarrow V$ such that $(u, v) \in E'$ iff $(f(u), f(v)) \in E$. If such a mapping exists, this is called a *subgraph matching* or *subgraph isomorphism*. This types of problems are said to be *exact graph matching*.

The term *inexact* applied to some graph matching problems means that it is not possible to find an isomorphism (or sub-graph isomorphism) between the two graphs to be matched. This is the case when not all of the graph elements of the graph G' match exactly to all of the graph elements of graph G or the nodes and edges of both graphs to be matched have different types³. Therefore, in these cases no isomorphism (or sub-graph isomorphism) can be expected between both graphs, and the graph matching problem does not consist in searching for the *exact* way of matching vertices of a graph with vertices of the other, but in finding the *best* matching between them. The best correspondence of a graph matching problem is defined as the optimum of some objective function which measures the similarity between matched vertices and edges [Ben02].

4.3.1 Partial Graph Matching

In some sub-graph matching problems, the problem is still to find one-to-one, but with the exception of some vertices in the data graph which have no correspondence at all. In other words, one or more graph elements of one graph are missing from the match to another graph, like a node or an edge. More formally, given two graphs $G_M = (V_M, E_M)$ and $G_D = (V_D, E_D)$, the problem consists

²We use the term query graph because we can represent query we are posing on semantic graph to find the patterns as a graph, which gives the concept of graph matching.

³A graph matching problem is considered to be inexact when not all of the graph elements of one graph match exactly to all of the graph elements of another graph or the nodes and edges of both graphs to be matched have different types. However, it is important to note that in the case of some attributed graph matching problems, the fact of even having the same number of vertices and edges does not imply the existence of isomorphism, and in the latter case that would also be an inexact graph matching problem.

in searching for a homomorphism⁴ $f : V_D \rightarrow V_M \cup \{\emptyset\}$, where $\emptyset$ represents a null value, meaning that when for a vertex $a \in V_D$ we have $f(a) = \emptyset$, there is no correspondence in the model graph for a vertex a in the graph. This value $\emptyset$ is known as *null vertex* or *dummy vertex* and this type of graph matching is called *partial graph matching*.

4.3.2 Graph Matching allowing more than one Correspondence per Vertex

Some other graph matching problem allow many-to-many matches, that is, given two graphs $G_M = (V_M, E_M)$ and $G_D = (V_D, E_D)$, the problem consists in searching for a homomorphism $F : V_D \rightarrow W$ where $W \in \mathcal{P}(V_M) \setminus \{\emptyset\}$ and $W \subseteq V_M$. In case of using also dummy vertices, W can take the value $\emptyset$, and therefore $W \in \mathcal{P}(V_M)$. This type of graph matching is more difficult to solve, as the complexity of the search for the best homomorphism has much more combinations and therefore the search space of the graph matching algorithm is much bigger.

4.3.3 Complexity of Graph Matching

The whole category of graph matching problems has not yet been classified with in a particular type of complexity such as P or NP-complete. Some papers in the literature tried to prove its NP-completeness when the two graphs to be matched are of particular types or satisfying some particular constraints [GJ90], but it still remains to be proved that the complexity of the whole type remains with in the NP-completeness at most. On the other hand for some type of graphs the complexity of the graph isomorphism problem has been proved to be polynomial type. The exact sub-graph matching problems has been proven to be NP-complete [GJ90]. However, some specific types of graphs can also have a lower complexity. For instance, the particular case in which the big graph is a forest and the small one to be matched is a tree has been shown to be of a polynomial complexity [GJ90]. In inexact graph matching, where $|V_M| \leq |V_D|$, the complexity is proved to be NP-complete. Similarly, the complexity of the inexact sub-graph problem is equivalent in complexity to the largest common sub-graph problem, which is known to be also NP-complete.

4.3.4 Graph Edit Distance

Graph edit distance defines dissimilarity of two graphs by *minimum* amount of distortion that is needed to transform one graph into another [BA83]. In contrast with other approaches, it does not suffer from any restrictions and can be applied to any type of graphs, including hypergraphs [BA83]. The distortions, or edit operations e_i consists of *insertions* ($\varepsilon \rightarrow \vartheta$), *deletions* ($u \rightarrow \varepsilon$), and *substitutions* ($u \rightarrow v$) of nodes and edges. A sequence of edit operations $(e_1, \dots, e_k)$ that transform a graph G_1 into graph G_2 is commonly referred to as *edit path* between G_1 and G_2 .

In order to represent the degree of modification imposed on a graph by an edit path, a cost function is introduced measuring the strength of the change caused by each edit operation. Consequently, the edit distance of graphs is defined by the minimum cost edit path between two graphs.

⁴A graph homomorphism f from a graph $G = (V, E)$ to a graph $G' = (V', E')$ is a mapping $f : V \rightarrow V'$ from the vertex set of G to the vertex set of G' such that $(u, v) \in E'$ iff $(f(u), f(v)) \in E$.

Edit operations on edges can be inferred by edit operations on their adjacent nodes, i.e., whether an edge is substituted, deleted, or inserted, depends on the edit operations performed on its adjacent nodes. Graph edit distance can be used to address various graph classification problems with different methods, for instance k -nearest-neighbor classifier (k -NN), etc. The main drawback of graph edit distance is this method is feasible for graphs of rather small size only and its computational complexity is exponential in the number of nodes of the involved graphs.

4.4 Pattern Analysis in Semantic Graphs

A graph pattern (or a pattern query) can be formally defined as follows:

Definition 4.1: [Graph Pattern] *A graph pattern is a pair $P = (G, F)$, where G is a subgraph and F is a predicate on the attributes of the subgraph.*

When this graph pattern is matched against semantic graph, we get the matched pattern graph. The process of matching is called graph pattern matching and which is formally defined as follows:

Definition 4.2: [Graph Pattern Matching] *A graph pattern $P(G_1, F)$ is matched with a graph G if there exists an injective mapping $\phi : V(G_1) \rightarrow V(G)$ such that i) for $\forall e(u, v) \in E(G_1)$, $(\phi(u), \phi(v))$ is an edge in G , and ii) predicate $F_\phi(G)$ holds.*

The resulting graph after matching the graph pattern with the semantic graph is called matched graph and which is formally written as:

Definition 4.3: [Matched Graph] *Given an injective mapping ϕ between a pattern P and a graph G , a matched graph is a triple $\langle \phi, P, G \rangle$ and denoted by $\phi_P(G)$.*

Although its a triple, a matched graph has all characteristics of a graph. Thus all concepts that apply to a matched graph, e.g. a collection of matched graphs is also a collection of graphs.

As we have already seen, an ontology graph defines the type and structure of vertices (e.g. the vertex attributes) and the connections between the vertices (e.g. the edge types and edge connections). As we are interested in searching semantic graphs for patterns that reveals the important information, an example of such a pattern is shown below in Figure 4.2.

Pattern query shown diagrammatically in Figure 4.2 is translated in disjunctive datalog predicates as given below:

$$works_at(X, Y) \wedge works_at(X, Z) \wedge type(X, person) \wedge type(Y, city) \wedge type(Z, city).$$

Here in the figure, we write the node values $X:A$ for the simplification to represent pattern in a single diagram. The first part X denotes a particular node in semantic graph and A denotes the corresponding node type of that particular node X .

A semantic graph might include the set of graphs shown in Figure 4.3 and 4.4, which have the equivalent representation in disjunctive logic predicates as follows:

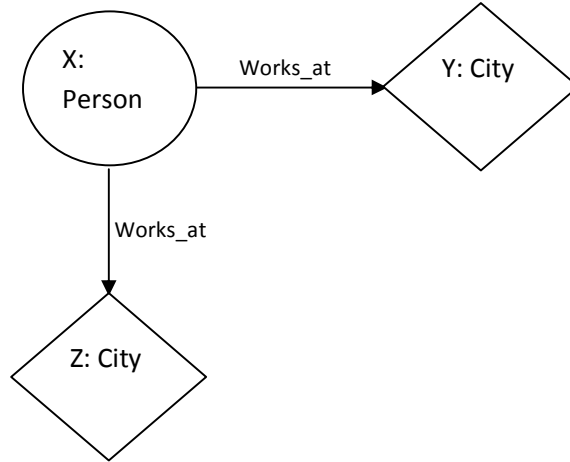


Figure 4.2: A basic pattern represented as a graph with respective types of the nodes

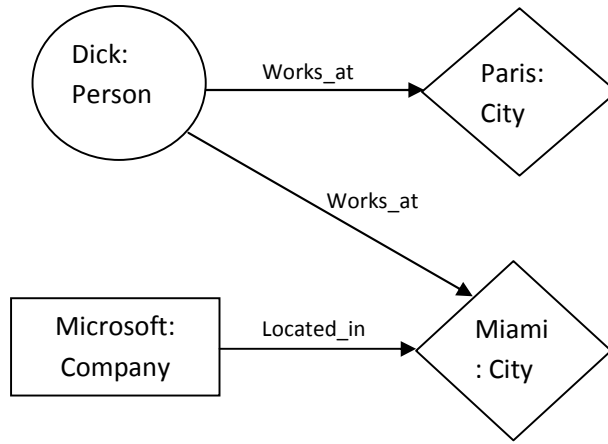


Figure 4.3: A part of a semantic graph that match to the pattern given in figure 4.2

$$works_at(dick, paris) \wedge works_at(dick, miami) \wedge located_in(microsoft, miami) \wedge$$

$$type(dick, person) \wedge type(paris, city) \wedge type(miami, city) \wedge type(microsoft, company).$$

and

$$works_at(chen, berlin) \wedge works_at(chen, dubai) \wedge located_in(yahoo, dubai) \wedge$$

$$type(chen, person) \wedge type(berlin, city) \wedge type(dubai, city) \wedge type(yahoo, company).$$

When the query pattern shown above in Figure 4.2 is matched against these graphs, there would be the following result given in Figure 4.5 and 4.6. The “company” vertex and “located_in” edge are not included in the result because these elements are not included in the query pattern we are interested in.

Again, query pattern in disjunctive datalog for finding the pattern as shown in Figure 4.7 in semantic graph is:

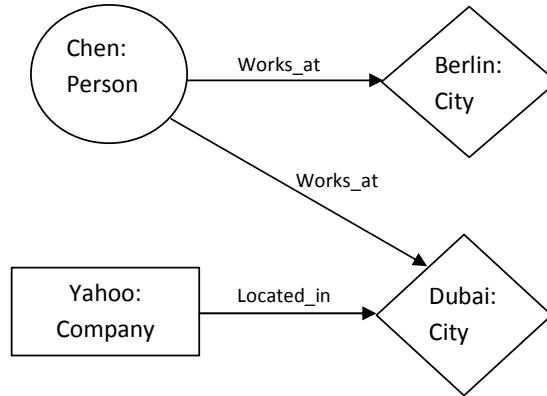


Figure 4.4: A part of a semantic graph that is similar to the pattern in figure 4.2

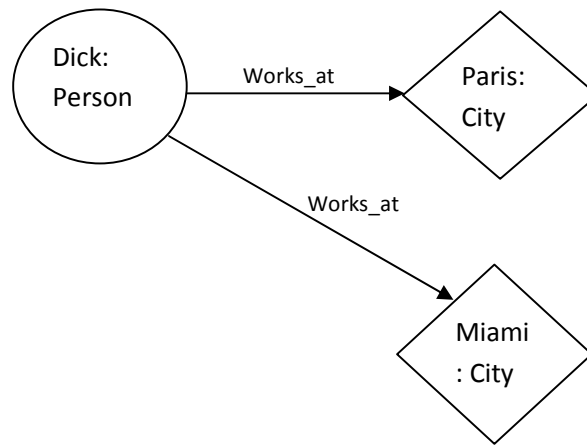


Figure 4.5: The result after matching figure 4.3 with pattern defined in figure 4.2

$$located_in(W, X) \wedge employs(W, Y) \wedge manufactures(W, Z) \wedge \\ type(W, company) \wedge type(X, location) \wedge type(Y, person) \wedge type(Z, product).$$

A semantic graph might include the set of graphs shown in Figure 4.7. When the query pattern shown above is matched against the Figure 4.7, there would be the same result given in Figure 4.8 because Figure 4.8 itself is perfectly matched with the query pattern we are interested to.

The identifiers that are associated with each vertex type in the pattern specification define an instance of that type in the graph. A graph might have many unique instances of the type **person**. In a pattern **X:person** and **Y:person** indicate two different person vertices. In the pattern specified in Figure 4.2 single **person** vertex is joined two city vertices by “worked_at” edges. In this case, the instances of **city** vertices are different. A pattern like this might be used to find people who have worked at two different cities.

In some cases a pattern needs to be able to specify that a link doesn’t exist. For example, if we want to find all of the people who live together but are not married, a pattern like the one formalized below could be used.

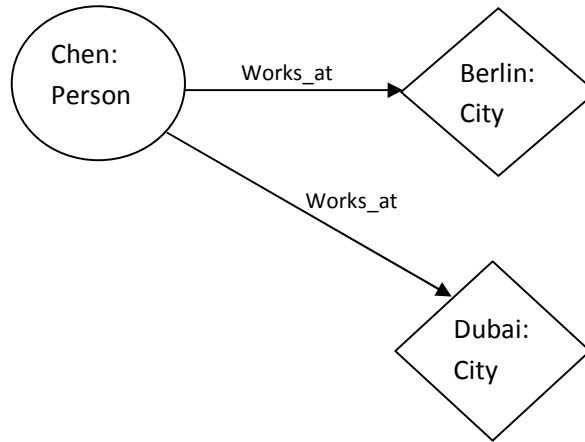


Figure 4.6: The result after matching figure 4.4 with pattern defined in figure 4.2

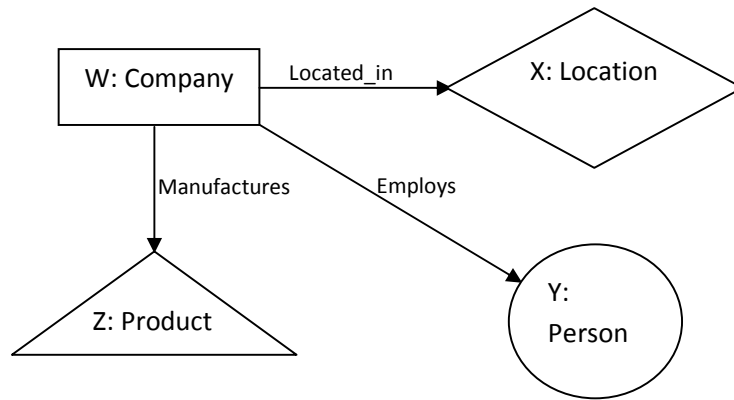


Figure 4.7: A basic pattern represented as a graph indicating their types respectively

$$lives_with(A, B) \wedge type(A, person) \wedge type(B, person) \wedge \neg married_to(A, B).$$

There could also be cases where the objective of a pattern would be to return a pattern that has a link of type “*works_at*” or “*travels_to*”, which will give result a person who either “*works_at*” or “*travels_to*” the given address. These type of queries can be formalized as:

$$(works_at(A, B) \vee travels_to(A, B)) \wedge type(A, person) \wedge type(B, location).$$

4.5 Ontologies for Graph Matching

In semantic graphs, we have labeled nodes and edges, which allows us to search for entities of a particular type with particular types of relationships. When we match query against the semantic graph, we can rule out many more matches because the attributes on the nodes and edges do not match [Gre07, Min07]. This leaves us to focus on a smaller number of matches, but they still need to be exact. What we will do, if, for a particular node, we are looking for a lorry and find a truck, that is very different from finding a teddy bear. Both a lorry and a truck are specializations of

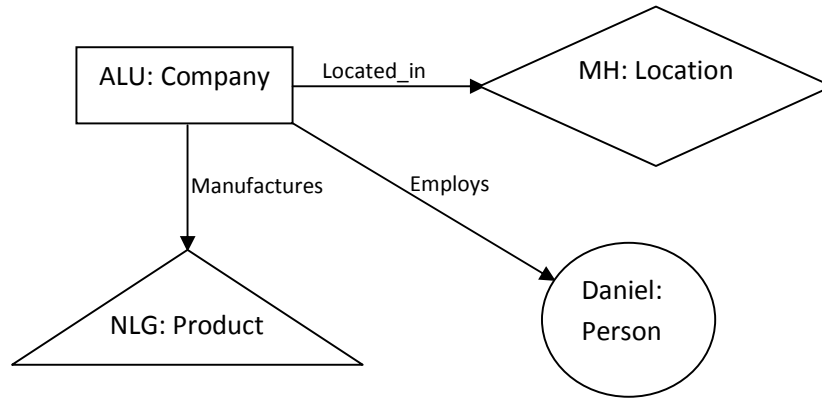


Figure 4.8: A pattern in semantic graph that match to the query pattern defined in figure 4.7

vehicle, so a more slightly generalized search would find both of them. To determine how far we are from an exact match is important in the analysis of semantic graphs. Although a simplest way to measure is to find a minimum graph edit distance metric [BS98, BA83], finding minimum graph edit distance in semantic graphs is very difficult, as semantic graph has very large number of nodes and the query graph has significantly small number of nodes.

To determine lorry and truck are the generalization of vehicle, we have to provide the taxonomic definition in a post-processing step of graph matching to filter results [Gre07, SBF⁺07]. For this job, it is not necessary to provide the full-blown ontology of the specific domain but we should have at least need the generalization hierarchy of the domain. On the one hand, ontology help us to search for a more complex object and infer the missing information in data i.e. if we find the inexact or partial match for a complex object, we can use the ontology to classify the object found and determine what is missing. On the other hand, we can use ontology to infer missing information in patterns in a preprocessing step. If the user can specify properties of the object for which he would like to search, then the system can again classify the object in question and form the appropriate pattern graph to match [HJQW05].

4.6 Pattern Matching

The pattern matching mechanism should be highly efficient to extract the more useful information from the semantic graphs. The pattern matching techniques are studied from many years and one of the technique used in LOOM knowledge representation system is presented in [Gre88]. This pattern matcher has a very rich pattern-forming language, and is logic-based, with a deductive mechanism which includes a truth-maintenance component as an integral part of the pattern-matching logic. The matcher uses an inference engine called a classifier to perform the matches. The paper [YNM91] also described a semantic pattern matcher and a pattern classifier used in CLASsification based Production system (CLASP). The semantic pattern matches extend the pattern matching capabilities of rule-based system through the use of terminological knowledge. The pattern classifier enables the system to compute a rule's specificity and the paradigm also helps to enhance the reasoning capabilities of rule-based systems.

Without any restriction on semantics, in large semantic graphs, we find too many matches for any of them to be meaningful. The restriction imposed in formation of the query pattern is one of the best way to get the narrower domain for the exact match to be found. Introducing type relation in query patterns help us to filter out the matches that are not meaningful and ease us to get the close/perfect matches. The close matches are those matches, which perfectly match with the semantic graphs respective to the vertex types semantics.

We classify the output of our matcher in the following types depending on the query pattern we pose to the semantic graphs and the results we get from them:

4.6.1 Exact Subgraph Matcher

We will get the exact subgraph match when the query pattern we posed is exactly the part of the semantic graph. That means, the query graph of the defined query is exactly the subgraph of the semantic graph to which we pose the query against. When we get the result in this match that result is the exact in which every nodes and edges along with their types of the query graph are perfectly matched with those of semantic graph. For example, in Figure 4.9, the graph given on the left hand side is the query graph and the graph on the right hand side is the semantic graph and the nodes 1, 2, 3, and 4 of the query graph are perfectly matched with the nodes 1, 2, 3, and 4 of the semantic graph, respectively. The exactly matched nodes and edges are shown on the shaded color.

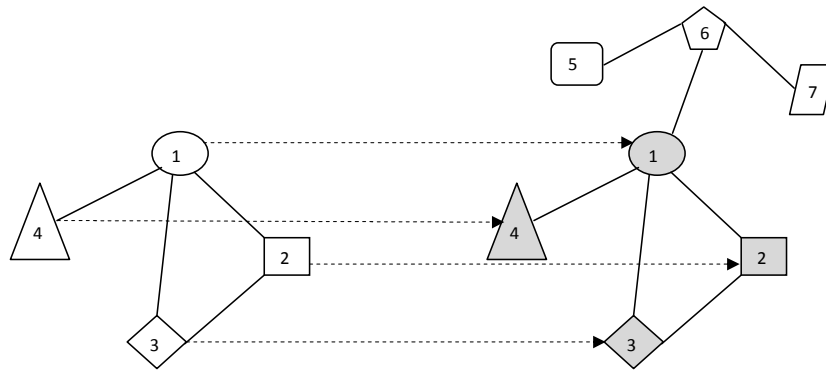


Figure 4.9: Exact graph matching

4.6.2 Partial Subgraph Matcher

In this type of matcher, we cannot expect the match to be perfect because some part of the query pattern is missing on the semantic graph. When we get the result in this match that result is inexact in which some nodes and edges along with their types of the query graph are missing on the semantic graph. For example in Figure 4.10, nodes 1, 3, and 4 of the query graph are perfectly matched with the nodes 1, 3, and 4 of the semantic graph but the node 2 and the edges between node 1, 2 and 3, 2 of the query graph are missing on the semantic graph to match. The exactly matched nodes and edges are shown on the shaded color and the missing node is shown on the dashed one.

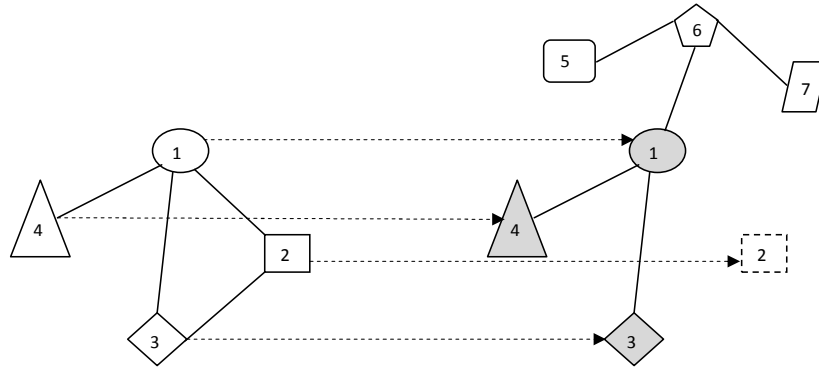


Figure 4.10: Partial graph matching

The missing part may be one node or one edge or number of nodes and edges. In this case we try to find the best match by comparing the query pattern with the knowledge base and the result which has many matches against the conjunction will be retrieved.

4.6.3 Hierarchy Matcher

This type of matcher are helpful when we are looking for the gun and we find the knife. These matches are not the perfect match but of course a better match. In this type of matcher, we don't have exact match but we can get the match when we go to the one or more level top of the ontology hierarchy. In this case this matcher is the subset of inexact matcher given below.

4.6.4 Inexact Matcher

Inexact matcher are of many types depending on how they are similar with the exact matches expected: one is the nearest inexact match: means we are looking for a gun but find a knife. In this case, if we have ontology hierarchy and they have same superclass, then we can say that these are the nearest match after you validate them with the help of domain ontology hierarchy. The other is the farthest match: means we are trying to find gun but we find teddy bear. In this case, finding similarity of the match after reasoning over ontology hierarchy is impossible. Thus, these farthest matches are not a bit helpful for our analysis and we don't consider this type of inexact matches.

For example, in Figure 4.11, the node 2 of query graph is not perfectly matched with the node 2 of the semantic graph except the rest of the nodes and edges.

In summary, the use of ontology hierarchy is useful to find the inexact matches and use of close types help us to filter out unnecessary matches.

4.7 Result Filtering Mechanisms

Constraints in our framework specify conditions which must not become true in any model. In other words, constraints are formulations of possible inconsistencies. This mechanism is very useful

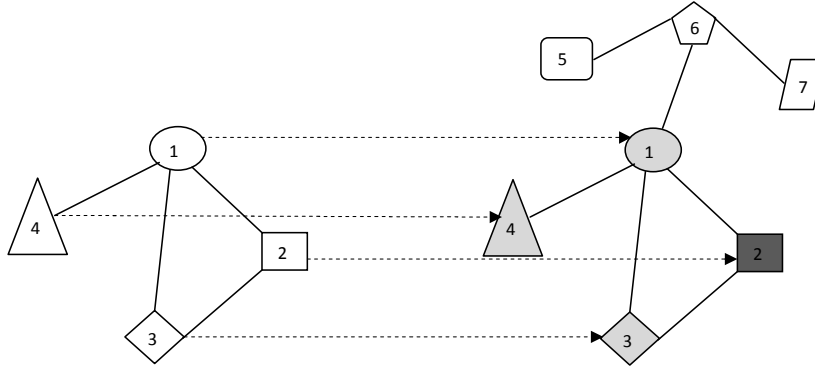


Figure 4.11: Inexact graph matching

in connection with disjunctive rules. The disjunctive rules serve as generators for different models and the constraints are used to select only the desired ones.

As disjunctive logic programming system DLV is based on answer set programming, we get many results that match some general criteria. Those matches may not be useful for the analysis so there are several result filtering techniques available on the DLV system. These filtering mechanisms are solely used to discard the unnecessary results from the stack of result sets and those mechanisms work on the post-processing step of query processing.

We can filter out unnecessary results generated from the query pattern with the help of these constraints:

- *Integrity constraints:* The use of the integrity constraints help us to filter out the results that are not required. They look like rules without heads. As with rules, constraints must meet the safety requirements. Safety of rules or constraints mean that each variable occurring in the head of a rule, in a negation-as-failure literal, in a built-in comparative predicate, as a guard or in the symbolic set of an aggregate also occurs in at least one non-comparative positive literal in the body of the same rule⁵.
- *Weak constraints:* The answer sets of a program P with a set W of weak constraints are those answer sets of P which minimize the number of violated weak constraints. They are called Best Models of (P, W) . One point to remember is that a program may have several best models (violating the same number of weak constraints).

Weak constraints can be weighted according to their importance (the higher the weight, the more important the constraint). In the presence of weights, best models minimize the sum of the weights of the violated weak constraints. Weak constraints can also be prioritized. Under prioritization, the semantics minimizes the violation of the constraints of the highest priority level first; then the lower priority levels are considered one after the other in descending order.

Weights and priority levels are allowed to be variables, provided that these variables also appear in a positive literal. The user can omit the weight or the priority or both, but all weak constraints must have the same syntactic form (i.e., the user is free to specify only

⁵<http://www.dbai.tuwien.ac.at/proj/dlv/>

weights or only priorities, or both, but all constraints of the program must have the same syntactic form).

The use of the weak constraints help us to define the priority of the matches and express the desiderata so that the model which violate the minimum number of constraints can only be the model for the query.

- *True negation/negation as failure:* Negation is treated as “negation as failure” in the DLV system. In other words, If an atom is not true in some model, then its negation should be considered to be true in that model. With this mechanism we can, for example, define the complementary graph of a given graph. This is the graph which has the same nodes, but of all possible arcs, it has exactly those arcs which do not exist in the original graph.

$$node(X) : \neg arc(X, _).$$

$$node(Y) : \neg arc(_, Y).$$

$$comparc(X, Y) : \neg node(X), node(Y), not\ arc(X, Y).$$

DLV implements yet another notion of negation – true negation. Negation as failure, which has been introduced above, does not support explicit assertion of falsity. Rather, if there is no evidence that an atom is true, it is considered to be false.

There are several situations in which negation as failure is not appropriate because it is necessary that something is explicitly known to be false. For this reason, true negation is sometimes referred to as explicit negation.

- *Aggregate Predicates:* aggregate predicates allow to express properties over a set of elements. They can occur in the bodies of rules and constraints, possibly negated using negation-as-failure. DLV programs with aggregates often allow clean and concise problem encodings by minimizing the use of auxiliary predicates and recursive programs, and help the DLV programmer to depict problems in a more natural way. From the point of efficiency, encodings using aggregates often outperform those without, since the size of the ground instantiation tends to be much smaller in this case.

All of these constraints are generally applied in the post-processing step of the query processing.

In general, by posing a query in DLV one looks for ground substitutions such that the substitution applied to the query is true or validated by the rest of the program. Since a disjunctive datalog program may have more than one model, there are different reasoning front-ends called brave reasoning and cautious reasoning to decide whether a substituted query is satisfied. The query syntax is the same for all types of queries. Queries consist of one or more literals, which must be separated by comma and terminated by a question mark. Only one query per program is considered.

A special case arises when the query is ground. In this case, there is only one meaningful substitution to consider, i.e., the empty substitution, and therefore the task of finding a substitution boils down to a decision of whether the empty substitution is admissible or not. For brave reasoning (a query is bravely true for a substitution, if its conjunction, on which the substitution has been

applied, is satisfied in at least one model of the program), this means deciding whether there exists an answer set in which the query holds, and for cautious reasoning (a query is cautiously true for a substitution, if its conjunction, on which the substitution has been applied, is satisfied in all models of the program), the task is deciding whether the query holds in all answer sets. For ground queries, there is a third alternative – one might be interested in which models the query is satisfied, sometime we called plain disjunctive datalog for the datalog which process ground queries.

Chapter 5

Analysis on Movies Database

‘The way to do research is to attack the facts at the point of greatest astonishment.’

Celia Green

This chapter explains the disjunctive logic based pattern finding framework in detail. First, we discuss the proposed system and the experimental setup to evaluate our system. Then, we analyze the overall performance result of our framework using the Movie database available in UCI KDD archive [HB99]. Last, we explain the scalability issues related to disjunctive logic programs and the work in other areas that is related to our work. We claim that our system outperforms the currently available pattern discovery mechanisms operate on semantic graphs.

5.1 System Description

As we have already seen, we can capture the semantic profile of a node by treating all paths in the graph as binary predicates, assigning true to the paths the given node participates in and false to the one it does not. By doing so we essentially transform a semantic graph in to propositional representation, i.e., we translate the semantic graph into a standard logical notation by representing nodes as constants and links via binary predicates. Those binary predicates contain meaning and can also be translated into natural language [LC07].

The pattern finding framework which operates on semantic graphs is shown in Figure 5.1. The logical representation block is for the transformation of semantic graph data in to first order predicate logic representation. Semantic graphs are transformed to the first order predicate logic representation using the representation mechanism described in Section 3.2. Just to recapitulate what we mentioned there, we basically represent all the node and edges in semantic graphs using first order binary predicates which we called knowledge base. Similarly, type information available from ontology graph is also represented similarly in the logical representation. Similarly, the domain ontology hierarchy, if any, associated with the ontology graph is also transformed to the logical rules. This is the intuitive approach we use to find the exact matches that are corresponding to the correct vertex type, because semantic graphs generally are of very large size and there may be other vertex types have the same name with the node of other vertex type. The introduction of

type relation in the pattern query help us to find exact/close useful matches among many several available similar matches.

The feature value computation and pattern induction is the backbone of our framework. All the computations we done on the semantic graphs depend on pattern induction. Inductive learning methods are used to generate patterns from the knowledge base available using positive and negative learning examples. Although negative examples are not available in our knowledge base, we generate the negative examples from the examples available in our knowledge base using inductive learning based example generation techniques.

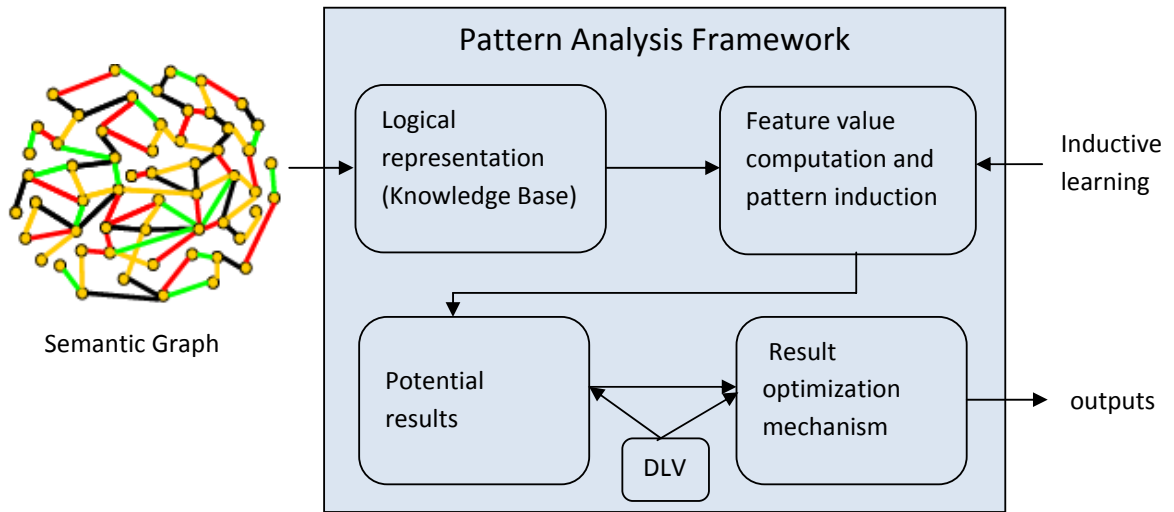


Figure 5.1: Flow graph for analyzing semantic graphs using Disjunctive Datalog

This part is the most crucial part of our work. Writing pattern queries plays vital role in our analysis framework. The selection of particular pattern is based on the objective of our analysis too. The selection of pattern queries to analyze any semantic graph data is based on the subjectiveness of the selection. The generation of pattern queries from inductive learning is particularly helpful here to choose from available patterns that match some criteria. Automatically pick up the queries that are important when comparing to some selection measures may reduce the biasness imposed from the subjectiveness of the selection. We write the patterns for the immediately true relations. The pattern induction can be dealt plays vital role in the analysis of semantic graphs and the patterns can be constrained by the node and edge types available in ontology graph.

After computation of patterns (or selection of patterns generated from inductive learning), these patterns are provided to the DLV system. DLV system generate the potential outputs from matching the query with the semantic graph knowledge base. At this point, DLV generate all possible outputs which are not of interest to us. This is the “Guess” part of the three step paradigm used by DLV to solve any query. These potential results are again provided to the filtering mechanisms specified in our patterns to select only the useful results. The filtering mechanisms range from integrity constraints to weak constraints to aggregate functions. This process fall under the category of remaining two steps of the DLV paradigm called “Check” and “Optimize”. The check and optimize steps of the execution mechanism prune the candidate answer sets by evaluating

them against specified constraints and removing from consideration the ones which do not satisfy those constraints.

The pseudo-code algorithm for the pattern finding framework is given below. The algorithm

```

Input  $G$  as a semantic graph  $(V, E, L, vt, et)$  with nodes  $V = \{v_1, v_2, \dots, v_{|V|}\}$ , edge
relations  $E = \{e_1, e_2, \dots, e_{|E|}\}$ , respectively the vertex types  $T_V = \{t_{v1}, t_{v2}, \dots, t_{|T_V|}\}$ 
of ontology graph  $O = (T_V, T_E, L, I)$ , and each  $E_i$  links a source node  $u$  to a target
node  $v$  via link of type  $e_i$  (same for  $T_E$  too) and  $vt$  maps the  $V$  of the semantic
graph to the  $T_V$  of ontology graph  $O$ .

begin
1.  $P_t := \text{answer\_query\_pattern}(G, O, Q, F)$ 
   define DLV query pattern :  $Q$ 
   for  $n = 1$  to  $|V|$ 
   while  $p_t \in P_t$ 
   extract pattern answers  $p_{t_i} := \text{dlv}(G, O, \phi_{Q_i}(G), F)$ 
   end
2.  $P_t := \text{find\_answer\_sets}(G, O, p_t, F)$ 
   define DLV program :  $p_t$ 
   specify constraints  $c$ 
   while  $p_t \in P_t$ 
   extract answer sets  $p_{t_i} := \text{dlv}(G, O, \phi_{c_i}(G), F)$ 
   end
3.  $P_t := \text{find\_complementary\_answer\_sets}(G, O, p_t, F)$ 
   define DLV program :  $p_t$ 
   specify constraints  $c$ 
   while  $p_t \notin P_t$ 
   extract complementary answer sets  $p_{t_i} := \text{dlv}(G, O, \phi_{c_i}(G), F)$ 
   end
4.  $P_t := \text{find\_abnormal\_instance}(G, O, p_t, F)$ 
   define DLV program :  $p_t$ 
   specify constraints  $c$ 
   while  $p_t \in P_t$ 
   extract abnormal instance  $p_{t_i} := \text{dlv}(G, O, \phi_{c_i}(G), F)$ 
   end
output  $P_t$ 
end

```

Table 5.1: Pseudo-code algorithm for pattern finding framework

designed for the pattern finding framework operates as follows: the DLV knowledge base generated from converting semantic graph including type information encapsulated in the ontology graph are given as input to the system. Then formalized query patterns are executed against knowledge base to find possible answer sets after evaluating the query against DLV knowledge base. If we want to validate or to find the answer for the query, the query is evaluated using brave and cautious reasoning front-ends of DLV. Other wise, the DLV program is evaluated using standard execution procedure of DLV and the answer sets those satisfy the specific constraints are retained and the others fail to satisfy the constraints are pruned.

5.2 Experimental Setup

For the evaluation of our pattern finding framework, we use the “Movies” dataset available at the UCI KDD Archive [HB99]. The data was originally compiled by Gio Wiederhold of Stanford University. We used this data to construct an ontology graph and the semantic graph to express most of the information available in the dataset. The Movies dataset contains information about movies, persons (actors, directors, etc.), studios, distributors, awards, quotes, locales, casts, etc.

The data is stored in relational form across several files. The central file MAIN is a list of movies, each with a unique identifier. The actors for those movies are listed with their roles in CASTS file. More information about individual actors is given in ACTORS file. All directors in the MAIN file are listed in PEOPLE with a number of important producers, writers, and cinematographers. A fifth file REMAKES links movies that were copied to a substantial extent from each other. The sixth file STUDIOS provides some information about studios shown in MAIN table. Outside of the key fields, missing values are common. Sometime the data seems to be unavailable, sometime, as according to author, it hasn’t been entered. Some information, as ‘lived-with’ is inherently incomplete. the minor actors of the movies are ignored and there is the dependency that every film listed in MAIN must have a director in PEOPLE file. We briefly describe each of the files of the database in the paragraphs below.

The MAIN file of the Movie data contains almost 11435 ‘tr/td’ table row entries. The ACTORS file has 6813 ‘tr/td’ table row entries for many of the actors appearing in CASTS file, but not all actors listed in CASTS are documented. A total of almost 3290 ‘tr/td’ table row entries are recorded in the PEOPLE table in which 3011 rows are of directors. The CASTS is the large file of who acted as what in which movie. It has almost 46009 ‘tr/td’ table row entries, only partial for movies and role types. CASTS is an association relation, linking actors with movies. There other files contains around 200 to 1500 rows of data of awards, location, remake of movies etc.

The ontology graph of the Movie database we developed for our evaluation is given in Figure 5.2. The nodes and edges are labeled with their corresponding node and edge types.

The ontology graph of the Movies database has 8 vertex types which are: *Category*, *Distributor*, *Role*, *Person*, *Movie*, *Studio*, *Award*, and *Country* respectively. The corresponding edge types between these vertex types are shown in the Figure 5.2. The movie movie link describes the relation like *remake*, *synonym* or the *sequel* of the movie. Person person link gives us the information about the relation between the persons available in the semantic graph like *lived – with*, *married – to*, *child – of*, *father – of*, etc. relations [HB99]. The ontology graph guides us in the way to find the patterns in our semantic graph. To recall here, our semantic graph can only include the vertex or edge relations which are available in the ontology graph only. This means the ontology graph restricts us to the vertex and edge relations that will be available on the semantic graph.

There is no single ontology graph for the particular semantic graph. The choice of the ontology graph depend on necessity and scope of the analysis. If you are interested to the coarser view of the data and you are not specific to every object that are available in the semantic graph than the small semantic graph with less number of nodes to exhibit only the top view of the data association should be enough. For the extensive role played by each object is important, the definition of finer

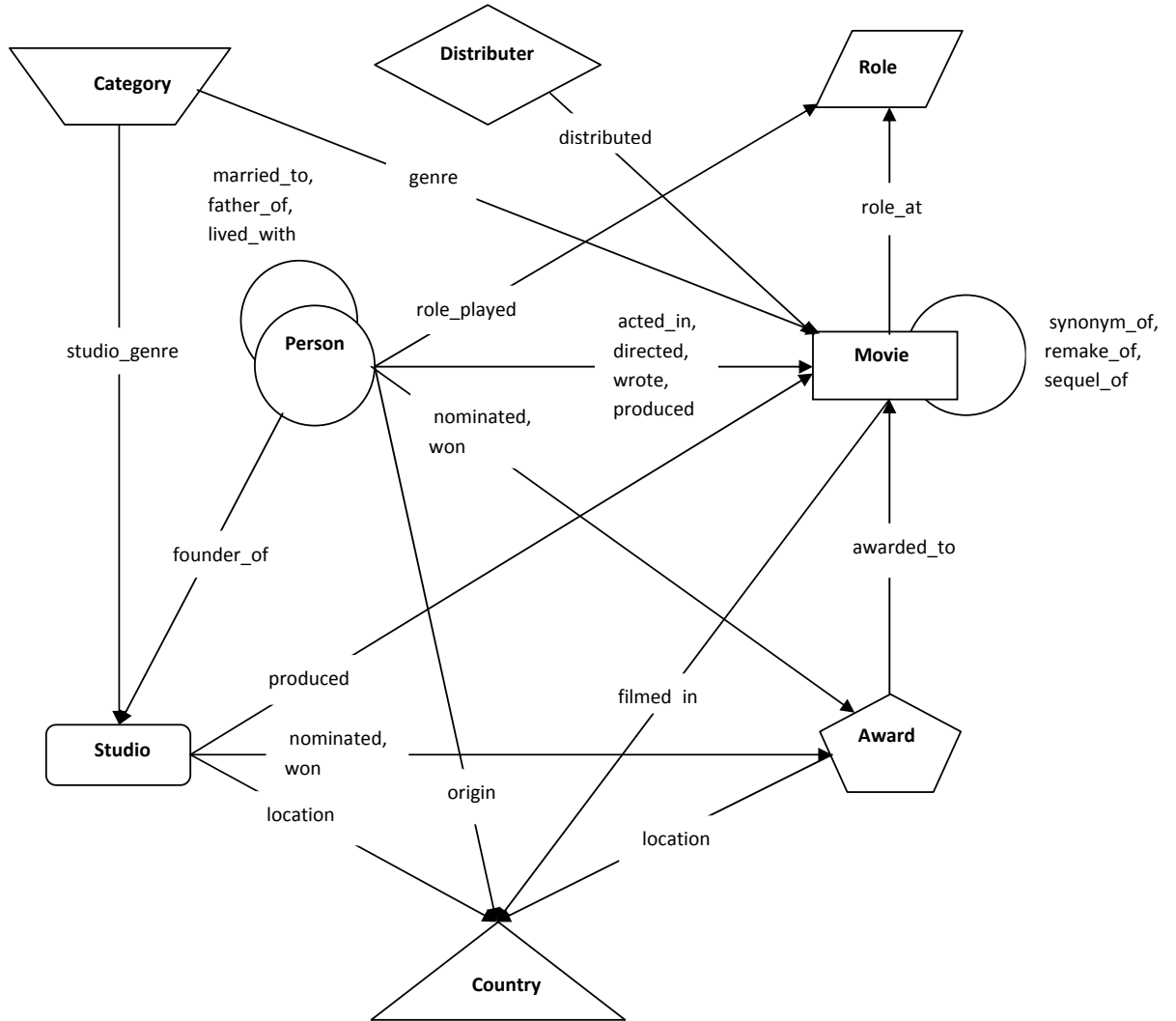


Figure 5.2: Ontology graph of Movies database

ontology graphs as described in Section 2.5 should be adapted. After replacing the node types with the node types that are common ancestor of those nodes give us the coarser view of the ontology graph.

5.3 Analysis and Evaluation

In this section, we describe our experiments performed on Movie database to evaluate our pattern finding framework. The goal of the evaluation is to demonstrate the usefulness of the framework by showing that it can find patterns and identify abnormal nodes, and that it does much better than other state-of-the-art machine learning based algorithms that have been used for the analysis of semantic graphs. Our motivation for using the the Movie database is that this type of data set has an answer key describing which entities are the targets that need to be found. Movie database provide the rich connection between different object types and is freely available.

We posed some pattern queries against our system setting. We select a wide range of patterns

that are to be explored using our setting and to measure the performance of our setting. Here are the list of the some queries we evaluated against our pattern finding framework.

1. For the pattern query “find all the movies which are remake of another movie and the fraction of the movie copied is greater than 95%”, we formalized this query on our framework as given below:

$$abnormal(X) : -remake_of(X, Y), fraction_copied(X, Z), Z > 95, Z \leq 100.$$

We found out 23 movies which match to this criteria. We verified the result given by the query by matching one by one to the knowledge base and we found out that there are exactly 23 movies which satisfy this condition. As copying more than 95% of the movie is kind of abnormal activity, we classify them as abnormal instances which may be useful for the further analysis.

From our assumption, the other movies which are copy of other movies but the copy percentage less than 95% are classified as normal. In this scenario, we run the pattern query given below to find out those instances in our movie data:

$$normal(X) : -remake_of(X, Y), fraction_copied(X, Z), Z < 95.$$

We compared the result given by above query with the one which we formalized using the true negation of DLV:

$$normal(X) : -remake_of(X, Y), fraction_copied(X, Z), not\ abnormal(X).$$

This query correctly classified the normal and abnormal instances of the movies which are remake of other movies. For the movies with more than one remakes over time are extracted as duplicates for the same movie title.

2. For the query “find all the actors who are married and also played in a same movie that is nominated for Oscar”, we formalize the pattern query as

$$married_to(X, Y), acted_in(X, Z), acted_in(Y, Z), nominated(Z, "Oscar").$$

This query lists all the actors married and played a movie together which is nominated for Oscar. In this case, the query works as a pattern which extract all the patterns that are valid in the Movies database. If we formalize the above query as given below:

$$actor(X) : -married_to(X, Y), acted_in(X, Z), acted_in(Y, Z), nominated(Z, "Oscar").$$

This query outputs all the actors who were married with some other actor and played at least a movie together which is nominated for oscar. This query works as a instance retrieval query in semantic graph analysis.

3. For query “List all the movies which are synonym of other movie and filmed in some country”, the query is formalized as:

$$synonym_of(X, Y), filmed_in(X, Z).$$

We have two goals in our approach: one is to find useful patterns and the other is to find instances which may be normal or abnormal. The result given by some of the above queries are instances and some are patterns. These instances constitute both normal and abnormal instances.

For example, in the scenario of the rule with head *abnormal(X)*, that rule point out the instances that are not normal after pruning the instances extracted by the rule with head *normal(X)* using result filtering techniques.

Finding abnormal instances is based on the scenario we defined in the pattern. Pattern generation is very important to find out the suspicious nodes and behaviors. As patterns can be generated from the available knowledge base by using the inductive learning techniques, the selection of specifically useful pattern by considering the purpose of analysis should be done to extract truly abnormal behavior from the semantic graphs so that those result can be useful for further analysis.

As there is no pattern finding framework implemented in knowledge representation area is available, we got difficulty in comparing the performance of our framework. Mostly, we manually checked the knowledge extracted from the framework with the Movies database and found out that our system performs significantly well in the Movie database for pattern retrieval.

One system, which we are also interested in to compare our result with, currently in development for finding patterns and abnormal instances in semantic graphs is KOJAK Link Discovery System¹ by the Information Sciences Institute, at University of Southern California, based on the logic-inferencing supported by the POWERLOOM KR&R System². In KOJAK, semantic graphs might be represented explicitly, or implicitly as views over relational data (e.g., stored in an RDBMS).

5.4 Experience with DLV

Disjunctive datalog system (DLV) combines databases and logic programming. For this reason, DLV can be seen as a logic programming system or as a deductive database system. In order to be consistent with deductive database terminology, the input is separated into the extensional database (EDB), which is a collection of facts, and the intensional database (IDB), which is used to deduce facts. The separation of EDB and IDB in to two different files is fairly a good idea to play with the patterns queries without touching the unchanging background knowledge. In case of dynamic semantic graphs data, this facilitate us to update the knowledge base according to the change in semantic graphs data over time. The expressive power is also the main concern in DLV to analyze very large semantic graphs. A simple conjunctive query even have to check all the predicates available in very large knowledge, which is time consuming and memory requirement is significantly high.

5.5 Related Work

Our problems and solutions are related to a variety of fields including intelligence analysis, knowledge discovery and data mining, scientific discovery, social network analysis and machine learning. For each related field, we will describe its definition, main goals, methodologies, and its similarity as well as difference to our research.

¹<http://www.isi.edu/isd/LOOM/kojak/>

²<http://www.isi.edu/isd/LOOM/PowerLoom/index.html>

Intelligence Analysis for Homeland Security

Our task is related to crime or homeland security analysis in the sense that one major application of our system is to identify abnormal and suspicious individuals and patterns from data. There has been a variety of research focusing on applying intelligent graph analysis methods to solve problems in homeland security and crime mining. Adibi et al. [ACM⁺04] described a method combining a knowledge based system with mutual information analysis to identify groups in a semantic graph based on a set of given seeds. Krebs [Kre02] described social network analysis approach on the 9/11 terrorists network and suggest that to identify covert individuals it is preferable to utilize multiple types of relational information to uncover the hidden connections in evidence. This conclusion echoes our decision of performing discovery on top of semantic graphs. There are also link discovery and analysis algorithms proposed to predict missing links in graphs or relational data sets. Recently, several general frameworks have been proposed to model and analyze semantic graphs such as relational bayesian networks, relational markov models and relational dependency networks. However, these frameworks aim at exploiting the graph structure to learn the joint or posterior probabilities of events or relations between them based on training examples. Our task and goal is different – we are not working from the side of machine learning (supervised or unsupervised approach), but working on the side of logic based approach to identify the abnormal instances and patterns.

Social Network Analysis

Social network consists of finite set of actors (nodes) and the binary ties (links or edges) defined between them. The actors are social elements such as people and organizations while the ties can be various types of relationships between the actors such as biological or behavioral interactions. The goal of SNA is to provide better understanding about the structure of a given social network. Although most of the analysis are focused on finding social patterns and subgroups, there are a small number of SNA tasks resembling our instance discovery. For example, centrality analysis³ [HR05] aims at identifying important nodes in the network based on their connectivity with others: an actor is important if it possesses high node degree (degree centrality) or is close to other nodes (closeness centrality). An actor is importantly connected to two source actors if it is involved in many connections between them. The major difference between centrality analysis and our approach is that centrality looks for central or highly connected nodes, while our system looks for those that are different from others.

Knowledge Discovery and Data Mining

Knowledge discovery and data mining (KDD) research focuses on discovering and extracting previously unknown, valid, novel, potentially useful and understandable patterns from lower-level data. Such pattern can be represented as association rules, classification rules, clusters, sequential patterns, time series contingency tables etc. The most relevant KDD research problems which are related to our problem are: graph mining [CF06] aims at mining data represented in graphs or trees such as mining interesting subclasses in a graph and mining the web as a graph. It is similar

³<http://www.orgnet.com/sna.html>

to our problem in the sense that a network is a type of graph. There are several well-known objective methods that discover important nodes in a graph. However, to our knowledge there is no work addressing how to determine interesting or abnormal instances in a graph using logic based approach. Small world analysis show that with a small amount of links connecting major clusters, it is essentially possible to link any two arbitrary nodes in a network with a fairly short path. The strength of weak ties approach address the issue that weak connections between individuals might be more important than strong ones, because they act like bridges. This concept is in some extent similar to our abnormal pattern discovery in the sense that rare paths are also represent a kind of weak connection given a specific similarity measure. The major difference between our approach and the approaches described above is that our system not only models complex syntactic structure of the typed graphs but also incorporate ontology measures to capture deeper meaning of the nodes.

Relational data mining (RDM) [DÖ0, MMT⁺02] deals with the relational tables in the database. It is related to our problem in the sense that semantic graph is a type of relational data and can be translated in to relational tables. RDM searches a language of relational patterns to find patterns that are valid in a given relational database. Morik [Mor02] proposed a way to find interesting instances in the relational database by first learning the rules and than searching for the instances that satisfy one of the following three criteria: exceptions to an accepted given rule; not being covered by any rule; or negative examples that prevent the acceptance of a rule. Inductive Logic Programming is one of the most popular RDM methods whose goal is mainly to induce a logic program corresponding to a set of given observations and background knowledge represented in logical form. ILP has been successfully applied to discover novel theories in various science domains such as math and biology. The standard ILP problem is not similar to ours, because it works on a completely supervised manner.

Outliers

An outlier is an observation that deviates so much from other applications to arouse suspicion that it was generated by different mechanism. Outlier detection [AY01] is an important technology with a variety of application such as video surveillance and fraud detection.

Part III

Reasoning on Semantic Graphs using Description Logics

Chapter 6

Representing Semantic Graphs in DLs

‘In theory, there is no difference between theory and practice. But, in practice, there is.’

Jan L. A. van de Snepscheut

This chapter presents the background of description logic based knowledge representation and reasoning system for semantic graphs. The mechanism to analyze them by using the ontology reasoning formalisms is appealing because semantic graph has domain ontology associated with its corresponding ontology graph. As family of description logics provide the richer syntax and semantics to represent ontologies consisting of different TBox and ABox size, the study of how to use them to analyze our problem is indispensable. Here, we introduce the background overview of the efficiency and scalability of different DL reasoners and the formalisms of description logic *SHIQ* which is incorporated by various DL reasoners those supports full ABox reasoning on very large knowledge bases.

6.1 Background

Description Logics (DLs) [BCM⁺03] are a family of knowledge representation formalisms with applications in numerous areas of computer science. They have long been used in information integration, and they provide a logical foundation for OWL - a standardized language for ontology modeling in the semantic web [OWL]. The DL reasoners can be used to reason about OWL ontologies and about annotations that are instances of concept descriptions formed using terms from an ontology. A DL knowledge base is typically partitioned in to a terminological (or schema) part, called a *TBox*, and an assertional (or data) part, called an *ABox*. Whereas some applications rely on reasoning over large TBoxes, many DL applications involve answering queries over knowledge bases with small and simple TBoxes, but with very large ABoxes. For example, documents in the semantic web are likely to be annotated using simple ontologies; however, the number of annotations is likely to be large. Similarly, the data sources in an information integration system

can often be described using simple schemata; however, the data contained in the source is usually large. Furthermore, semantic graphs are constrained by simple ontology graph with small domain ontology hierarchy associated with it; however, they contain very large number of instances data.

Many modern applications of description logics require answering queries over large data quantities structured according to relatively simple ontologies. Reasoning with the large data sets was extensively studied in the field of deductive databases, resulting in several techniques that have proven themselves in practice. Motivated by the prospect of using these techniques in query answering in description logics, Ullrich Hustadt at paper [Hus04] proposed a novel reasoning algorithm that reduces a *SHIQ* knowledge base KB to a disjunctive datalog program $DD(KB)$ while preserving the set of relevant consequences.

ABox reasoning truly extends the usefulness of description logics in practical applications. For example, in query answering over semantic graphs, we rely on full-fledged ABox reasoning. The increase of expressiveness is also reflected in an increase of the complexity of the tableau rules. An alternative approach to restrict significant increase in complexity of tableau rules is to use so-called “pre-completion approach” which transform given ABoxes in such a way that ABox satisfiability is reduced to concept satisfiability. Unfortunately, while existing techniques for TBox reasoning (i.e., reasoning about the concepts in an ontology) seem able to cope with real world ontologies [Hor98, HM01a], it is not clear if existing techniques for ABox reasoning (i.e., reasoning about the individuals in an ontology) will be able to cope with realistic sets of instance data. This difficulty arises not so much from the computational complexity of ABox reasoning, but from the fact that the number of individuals (e.g., annotations) might be extremely large. This is the trade-off of the use of general TBox reasoners in the analysis of ontologies with very large ABoxes. As we are pretty much interested in analysis of semantic graphs which have very small hierarchical domain ontology TBoxes and very large ABoxes of instance data, we need a mechanism that can uphold the reasoning on very large ABoxes and provide the efficient answering to the instance retrieval queries.

Several attempts has been made in description logics and its related areas to scale the ABox reasoning in large knowledge bases. One attempt is to use the conventional database for the ABox reasoning. Although using a database in order to support ABox reasoning is certainly not new, the application *instance Store* (*iS*) is the first system that is general purpose for the full ABox reasoning that uses a database. Horrocks et. al [HLTB04] proposed instance Store (*iS*), an approach to a restricted form of ABox reasoning that combines a DL reasoner with a database. The result is a system that can deal with very large ABoxes, and is able to provide sound and complete answers to instance retrieval queries (i.e., computing all the instances of a given query concept) over such ABoxes. While *iS* can be highly effective, it does have limitations when compared to a full fledged DL ABox. In particular, *iS* can only deal with a *role-free* ABox, i.e., an ABox that does not contains any axioms asserting role relationships between pairs of individuals. This approach used well-known techniques for reducing description logic reasoning with individuals to reasoning with concepts. The crucial part of the *iS* implementation is the combination of a description logic terminological reasoner with a traditional relational database. The resulting form of inference is sound and complete and is sufficient for several application which need role-free very large ABox

reasoning.

We are also interested in the architecture of instance Store. The core component of *iS* is a Java application talking to a reasoner via a DIG interface and to a relational database via JDBC. The four basic operations supported by *iS* are *initialize*, which loads a TBox into the DL reasoner, classifies the TBox and establishes a connection to the database; *addAssertion* which adds an axiom $i : D$ to *iS*; *retract*, which removes any axioms of the form $i : C$ (for some concept C) from *iS*; and *retrieve*, which returns the set of individuals that instantiate a query concept Q . Since an *iS* ABox can only contain one axiom for one individual, asserting $i : D$ when $i : C$ is already in the ABox is equivalent to first removing i and then asserting $i : (C \sqcap D)$ [HLTB04].

Horrocks et al. at paper [HLTB04] showed that *iS* provides stable and effective reasoning technique for role-free ABoxes, even those containing very large number of individuals. The implementation details of *iS* are of more interest to us because the full ABox reasoning using the RACER system exhibited accelerating performance degradation with increasing ABox size, and at least the current RACER release was not able to deal efficiently with the very large ABoxes.

For our purpose, to implement this idea for the analysis of semantic graphs, role-free ABox requirement for reasoning is a severe restriction. Semantic graphs generally have very large ABox with significant number of role relations between pair of individuals. In this scenario, our motivation is to investigate the possibility of using other existing reasoning algorithms for the analysis of semantic graphs. Instead of going to develop an application similar to *instanceStore* which can support reasoning with roles and individuals directly, we perform several experiments with the available DL reasoners to know their usefulness, efficiency and scalability for the analysis of very large semantic graph knowledge bases.

6.2 Description Logic *SHIQ*

We briefly introduce the description logic *SHIQ* [HM01b, Hor00] (please see the tables in Table 6.1) using a standard Tarski-style semantics. We are interested to the description logic *SHIQ* because all the DL reasoners which support full ABox reasoning supports reasoning over this logic which is an extension of basic description logic *ALC*. The description logic *SHIQ* extends the description logic *ALC $\mathcal{N}\mathcal{H}_{\mathcal{R}+}$* (which is itself an extension of description logic *ALC*) [HM00, HM01a] by additionally providing qualified number restrictions and inverse roles. Using the *ALC $\mathcal{N}\mathcal{H}_{\mathcal{R}+}$* naming scheme, *SHIQ* could be called *ALCQHI $\mathcal{R}+$* (pronunciation ALC-choir).

ALCQHI $\mathcal{R}+$ is briefly introduced as follows. We assume a set of concept names C , a set of role names R , and a set of individual names O . The mutual disjoint subsets of P and T of R denote non-transitive and transitive roles, respectively ($R = P \cup T$). The concept $\top(\perp)$ is used as an abbreviation for $C \sqcup \neg C$ ($C \sqcap \neg C$).

If R and S are role names, then $R \sqsubseteq S$ is called a *role inclusion* axiom. A *role hierarchy* R is defined by a finite set of role inclusion axioms. Then, we define $\sqsubseteq^*$ as the reflexive transitive closure of $\sqsubseteq$ over such a role hierarchy R .

The concept language of *ALC $\mathcal{N}\mathcal{H}_{\mathcal{R}+}$* syntactically restricts the combinability of number restrictions and transitive roles due to a known undecidability result in case of an unrestricted syntax

Syntax	Semantics
Concepts	
A	$A^{\mathcal{I}} \subseteq \Delta^{\mathcal{I}}$
$\neg C$	$\Delta^{\mathcal{I}} \setminus C^{\mathcal{I}}$
$C \sqcap D$	$C^{\mathcal{I}} \cap D^{\mathcal{I}}$
$C \sqcup D$	$C^{\mathcal{I}} \cup D^{\mathcal{I}}$
$\exists R.C$	$\{a \in \Delta^{\mathcal{I}} \mid \exists b \in \Delta^{\mathcal{I}} : (a, b) \in R^{\mathcal{I}}, b \in C^{\mathcal{I}}\}$
$\forall R.C$	$\{a \in \Delta^{\mathcal{I}} \mid \forall b \in \Delta^{\mathcal{I}} : (a, b) \in R^{\mathcal{I}} \Rightarrow b \in C^{\mathcal{I}}\}$
$\exists_{\geq n} S.C$	$\{a \in \Delta^{\mathcal{I}} \mid \ \{y \mid (x, y) \in S^{\mathcal{I}}, y \in C^{\mathcal{I}}\}\ \geq n\}$
$\exists_{\leq n} S.C$	$\{a \in \Delta^{\mathcal{I}} \mid \ \{y \mid (x, y) \in S^{\mathcal{I}}, y \in C^{\mathcal{I}}\}\ \leq n\}$
Roles	
R	$R^{\mathcal{I}} \subseteq \Delta^{\mathcal{I}} \times \Delta^{\mathcal{I}}$

A is a concept name and $\|\cdot\|$ denotes the cardinality of a set.

Axioms		Assertions	
Syntax	Satisfied if	Syntax	Satisfied if
$R \in T$	$R^{\mathcal{I}} = R^{\mathcal{I}^+}$	$a : C$	$a^{\mathcal{I}} \in C^{\mathcal{I}}$
$R \sqsubseteq S$	$R^{\mathcal{I}} \subseteq S^{\mathcal{I}}$	$(a, b) : R$	$(a^{\mathcal{I}}, b^{\mathcal{I}}) \in R^{\mathcal{I}}$
$C \sqsubseteq D$	$C^{\mathcal{I}} \subseteq D^{\mathcal{I}}$		

Table 6.1: Syntax and semantics of the Description Logic $\mathcal{SHIQ}$

[Hor00]. Number restrictions are only allowed for simple roles. Only simple roles may occur in number restrictions. Roles are *simple* if they are neither transitive nor have a transitive role as descendant. In concepts, instead of a role name R (or S), the *inverse role* R^{-1} (or S^{-1}) may be used.

If C and D are *concept* terms, then $C \sqsubseteq D$ (*generalized concept inclusion* or GCI) is a terminological axiom. A finite set of terminological axioms $\mathcal{T}_{\mathcal{R}}$ is called a terminology or TBox w.r.t. a given role hierarchy R . An ABox $\mathcal{A}$ is a finite set of assertional axioms as defined in Table 6.1. The ABox consistency problem is to decide whether a given ABox $\mathcal{A}$ is consistent with respect to a TBox $\mathcal{T}$ and a role hierarchy $\mathcal{R}$.

An *interpretation* $\mathcal{I}$ is a model of a concept C (or satisfies a concept C) iff $C^{\mathcal{I}} \neq \emptyset$ and for all $R \in R$ it holds that iff $(x, y) \in R^{\mathcal{I}}$ then $(y, x) \in (R^{-1})^{\mathcal{I}}$. An interpretation is a model of a TBox $\mathcal{T}$ iff it satisfies all axioms in $\mathcal{T}$. Please see Figure 6.1 for the satisfiability conditions. An interpretation is a model of an ABox $\mathcal{A}$ w.r.t. a TBox iff it is a model of $\mathcal{T}$ and satisfies all assertions in $\mathcal{A}$. Different individuals are mapped to different domain objects (unique name assumption).

A concept is called *consistent* (w.r.t. a TBox $\mathcal{T}$) if there exists a model of C (that is also a model of $\mathcal{T}$ and $\mathcal{R}$). An ABox $\mathcal{A}$ is consistent (w.r.t. a TBox $\mathcal{T}$) iff $\mathcal{A}$ has a model $\mathcal{I}$ (which is also a model of $\mathcal{T}$). A knowledge base $(\mathcal{T}, \mathcal{A})$ is called consistent iff there exists a model for $\mathcal{A}$ which is also a model for $\mathcal{T}$. A concept, ABox, or knowledge base that is not consistent is called *inconsistent*. *Instance checking* tests whether an individual a is an instance of a concept term C w.r.t. an ABox $\mathcal{A}$ and a TBox $\mathcal{T}$, i.e., whether $\mathcal{A}$ entails $a : C$ w.r.t. $\mathcal{T}$. This problem is reduced to the problem of deciding if the ABox $\mathcal{A} \cup \{a : C\}$ is inconsistent.

A concept D *subsumes* a concept C (w.r.t. a TBox $\mathcal{T}$) iff $C^{\mathcal{I}} \subseteq D^{\mathcal{I}}$ for all interpretations $\mathcal{I}$ (that are model of $\mathcal{T}$). If D subsumes C , then C is said to be subsumed by D .

A *query* Q over KB is a conjunction of literals $A(s)$ and $R(s, t)$, where s and t are variables

or constants, R is a role, and A is an atomic concept. In our experiments, we assume that all variables in a query should be mapped to individuals explicitly introduced in the ABox. Then a mapping θ of the variables of Q to constants is an *answer* of Q over KB if $KB \models Q\theta$.

6.3 Representing Semantic Graphs

For the Movie database, we represent the data in OWL-DL by inputting the data to the famous ontology editor Protege¹. The TBox data come from the conversion of ontology graph of Movie database and the ABox data come from the semantic graph representation using the instances available in Movie database.

¹<http://protege.stanford.edu/>

Chapter 7

Experiments on Semantic Graph Knowledge Bases

‘After all, the ultimate goal of all RESEARCH is not objectivity, but truth.’

Helene Deutsch

In this chapter, we perform several experiments with different semantic graph ontology data sets consisting of different ABox size using various DL reasoners. We describe the reasoning architecture of DL reasoners: KAON2, RACER and Pellet and provide the advantages and weaknesses of these reasoners for the reasoning on those data sets. After analyzing the performance results given by different DL reasoners, we found out that KAON2 performs significantly well with the semantic graph data sets which have very large ABox and the very small hierarchical TBox but the DL reasoners based on tableau proofs outperform the other reasoners when there is large TBox and the very small or no ABox.

7.1 KAON2, RACER and Pellet Architecture

KAON2¹ is a DL reasoner developed at the University of Manchester and the University of Karlsruhe. The system can handle *SHIQ* knowledge bases extended with DL-safe rules - first order clauses syntactically restricted in a way that makes the clauses applicable only to individuals mentioned in the ABox, thus ensuring decidability. KAON2 implements the following reasoning tasks for any DL ontologies: deciding knowledge base and concept satisfiability, computing the subsumption hierarchy, and answering conjunctive queries without distinguished variables (i.e., all variables of a query can be bound only to explicit ABox individuals, and not to individuals introduced by existential quantification). It has been implemented in JAVA.

The central component of KAON2 is the Reasoning Engine, which is based on the algorithm for reducing a *SHIQ* knowledge base KB to a disjunctive datalog program $DD(KB)$ [Hus04]. To understand the intuition behind this algorithm, consider the knowledge base $KB = \{C \sqsubseteq$

¹<http://kaon2.semanticweb.org/>

$\exists R.E_1, E_1 \sqsubseteq E_2, \exists R.E_2 \sqsubseteq D\}$. For an individual x in C , the first axiom implies existence of an R -successor y in E_1 . By the second axiom, y is also in E_2 . Hence, x has an R -successor y in E_2 , so, by the third axiom, x is in D . The program $DD(KB)$ contains the rules $E_2(x) \leftarrow E_1(x)$ and $D(x) \leftarrow R(x, y), E_2(x)$, corresponding to the second and the third axiom, respectively. However, the first axiom of KB is not represented in $DD(KB)$; instead, $DD(KB)$ contains the rule $D(x) \leftarrow C(x)$. The latter rule can be seen as a “macro”: it combines into one step the effects of all mentioned inference steps, without expanding the R -successors explicitly.

RACER² implements a TBox and ABox reasoner for the description logic *SHIQ*. RACER was the first full-fledged ABox description logic reasoner for a very expressive logic and is based on optimized, sound and complete tableau algorithms. RACER also implements a decision procedure for modal logic satisfiability problems (possibly with global axioms) which we are not interested in this report. The ABox consistency algorithm implemented in the RACER system is based on the tableaux calculus of its precursor RACE [HM00]. For dealing with qualified number restrictions and inverse roles, the techniques introduced in the tableaux calculus for SHIQ [HST00] are employed. However, optimized search techniques are still required in order to guarantee good average-case performance of RACER system, especially for very large ABoxes.

RACER is implemented in Common Lisp and is available for research purposes as a server program which can be installed under Linux and Windows³. Client programs can connect to the RACER DL server via a very fast TCP/IP interface based on sockets. Client-side interfaces for Java and Common Lisp are available.

The RACER architecture incorporates the following standard optimization techniques: dependency - directed backtracking and DPLL-style semantic branching. The implementation of these techniques in the ABox reasoner RACER differs from the implementation of other DL systems, which provide only concept consistency (and TBox) reasoning. The latter systems have to consider only so-called labels (sets of concepts) whereas an ABox prover such as RACER has to explicitly deal with individuals (nominals). The architecture of RACER is inspired by recent results on optimization techniques for TBox reasoning, namely transformations of axioms (GCIs), model caching and model merging (including so-called deep model merging and model merging for ABoxes). RACER also provides additional support for very large TBoxes.

Pellet⁴, developed at the University of Maryland, is an open-source Java based OWL DL reasoner, based on the tableaux algorithms developed for very expressive description logics. It is the first system that supports the full expressivity of OWL DL including reasoning about nominals (enumerated classes) taking into account all the nuances of the specification.

7.2 Data and Experiments

Before we propose or go for developing a system like *instanceStore* (*iS*) for the analysis of semantic graphs, we are interested in evaluating the existing reasoners KAON2, RACER and Pellet for the analysis of semantic graphs which have very small TBoxes but very large ABoxes. As KAON2,

²<http://www.racer-systems.com/features.phtml>

³<http://www.racer-systems.com/products/download/index.phtml>

⁴<http://www.mindswap.org/2003/pellet/index.shtml>

RACER and Pellet architecture support very large ABox reasoning, we are interested in evaluating the performance of these systems for the analysis of semantic graphs to get the idea about how the performance of these reasoners changes with different TBox and knowledge base size.

As there is no standard tests for ABox reasoning is available, we constructed our own data set from the Movie database by processing it using Protege ontology editor and the ontology database available from web resources (especially benchmark ontology *Univ-Bench* of LUBM from the Lehigh University archive and the Wine⁵ ontology, more on Section 7.2.1).

For Univ-Bench benchmark ontology of LUBM, the authors also supplied us with the queries used in their projects, which we then reused in our tests. We expect these queries to better reflect the practical use cases of their respective ontologies. The wine ontology is used to check the performance of the DL reasoners when there is ontology which has TBox of significant number of terminologies and ABox of very small size.

7.2.1 Test Knowledge Bases and Queries

*Movie database*⁶ [HB99] is available at the UCI KDD Archive. The data was originally compiled by Gio Wiederhold of Stanford University. The Movies dataset contains information about movies, persons (actors, directors, etc.), studios, distributors, awards, quotes, locales, casts, etc.

The movie data is stored in relational form across several files. The central file MAIN is a list of movies, each with a unique identifier. The actors for those movies are listed with their roles in CASTS file. More information about individual actors is given in ACTORS file. All directors in the MAIN file are listed in PEOPLE with a number of important producers, writers, and cinematographers. A fifth file REMAKES links movies that were copied to a substantial extent from each other. The sixth file STUDIOS provides some information about studios shown in MAIN table. Outside of the key fields, missing values are common. Sometime the data seems to be unavailable, sometime, as according to author, it hasn't been entered. Some information, such as 'lived-with' is inherently incomplete. The minor actors of the movies are ignored. And there is the dependency that every film listed in MAIN must have a director in PEOPLE file.

The database is highly incomplete and we processed extensively the available data to make it useable in our analysis. The detailed description of Movie data is available in Section 5.2. We used this data to construct an ABox, TBox, and test queries to experiment the analysis of semantic graphs from the description logics perspective. We randomly select the following three ABox queries to experiment with the various DL reasoners which use different reasoning algorithms:

$$M_1(x) \equiv person(x)$$

$$M_2(x, y) \equiv person(x), award(y), won(x, y)$$

$$M_3(x, y, z) \equiv movie(x), synonym_of(x, y), country(z), filmed_in(x, z)$$

*LUBM*⁷ [GPH04] is an ontology benchmark developed at the Lehigh University for testing

⁵<http://www.schemaweb.info/schema/SchemaDetails.aspx?id=62>

⁶<http://kdd.ics.uci.edu/databases/movies/movies.html>

⁷<http://swat.cse.lehigh.edu/projects/lubm/>

performance of ontology management and reasoning systems. The ontology describes organizational structure of universities and it is relatively simple. It does not use disjunctions and number restrictions. Due to the absence of disjunctions and equality, the reduction algorithm produces an equality-free Horn program. In other words, query answering on LUBM can be performed deterministically. The LUBM ontology schema and its data generation tool are quite complex. We used the Univ-Bench Ontology of LUBM that describes universities, departments, and the related activities.

To obtain Univ-Bench ontology ABox of sufficient size, we applied *replication* (copying an ABox several times with appropriate renaming of individuals in axioms) using Univ-Bench data generator tool (UBA), whose main generation parameter is the number of universities to consider, which can be used instead of ABox replication to obtain the test data. The test generator creates many small files; to make these ontologies easier to handle, we merged them into a single file. Generated data sets are named as: Univ(1,0), corresponding to 1 university and contains about 18000 axioms, Univ(2,0), corresponding to 2 universities and contains about 30000 axioms, and so on. The LUBM site also provides 14 test queries⁸ for use with the ontology, from which we selected the three queries given below.

$$\begin{aligned}
 U_1(x) &\equiv \text{UndergraduateStudent}(x) \\
 U_2(x, y) &\equiv \text{Chair}(x, \text{Department}(y), \\
 &\quad \text{worksFor}(x, y), \text{subOrganizationOf}(y, \text{"http://www.University2.edu"})) \\
 U_3(x, y, z) &\equiv \text{Student}(x), \text{Course}(y), \text{Faculty}(z), \text{advisor}(x, z), \\
 &\quad \text{takesCourse}(x, y), \text{teacherOf}(z, y)
 \end{aligned}$$

KB	$C \sqsubseteq D$	equivalent	functional	domain	range	$R \sqsubseteq S$	$C(a)$	$R(a, b)$
MOVIE(100)	2	0	0	7	5	10	211	310
MOVIE(500)							719	951
MOVIE(1000)							1500	1731
MOVIE(5000)							8013	9191
MOVIE(11435)							15017	18022
Univ(1,0)	36	6	0	25	18	9	18128	49336
Univ(2,0)							30508	113463
Univ(3,0)							44897	166682
Univ(4,0)							53200	236514
Univ(5,0)							65738	393227

Table 7.1: Test data statistics

Wine ontology contains classification of wines categorized by their color, test and origin. It is a freely available ontology with large nontrivial TBox and a small ABox. It uses nominals, disjunction, and existential quantifiers. We particularly select wine ontology to examine the TBox performance of RACER and Pellet reasoners because the performance of RACER and Pellet is considerably poor in our experiments for ontology with large ABoxes.

Table 7.1 shows the number of axioms for each ontology database. MOVIE(100) denotes the ontology data about 100 movies from the movie database. We created five different ontology

⁸<http://swat.cse.lehigh.edu/projects/lubm/query.htm>

database for the movie database to check with how various DL reasoners perform with the ontology database with increasing ABox size. Univ(1,0) is the ontology database which includes only the data related to only one university. We created five different ontology database for up to five universities to perform the experiments about how DL reasoners perform with increasing ABox size.

Here one important point to remember is that we select the LUBM ontology benchmark in our experiments to evaluate the performance of the different DL reasoners to establish a benchmark of their performance. That help us to get an idea about how their performance is changed with the real world database like MOVIE database. The comparison of performance of different DL reasoners in the standard benchmark database and the real world database give us the general idea of their scalability issues when use in practical applications.

7.2.2 Performance Measurement

The main goal of our performance measurement is to test the scalability of the algorithms incorporated in KAON2, RACER and Pellet to see how performance of query answering depends on the amount of data and on the complexity of different semantic graph ontologies. This should give us idea about the kind of data that is easily handled by the existing reasoners for the semantic graph data analysis. Additionally, the use of KAON2, RACER and Pellet to analyze different semantic graph knowledge bases is to compare the reasoning algorithms used by them. As KAON2 based on reducing a *SHIQ* knowledge base KB to a disjunctive datalog program $DD(KB)$ while preserving the set of relevant consequences, this should be the test to determine that how efficient the algorithm is when we use it to analyze semantic graphs. As RACER use tableau algorithms, the goal is to experiment how efficient is the counterparts of the tableau algorithms to analyze the semantic graph ontologies which have large ABox but comparably simple TBox. Pellet again, in its core, is a DL reasoner based on tableaux algorithms. The tableaux reasoner checks the consistency of the knowledge base and all the other reasoning services are reduced to consistency checking. The reasoner is designed so that different tableaux algorithms can be plugged in.

System Configuration: All tests were performed on a laptop computer with a 1.83 GHz Intel Core 2 Duo processor, 1 GB of RAM, running Windows Vista Ultimate Service Pack 1. For Java-based tools, we used Sun Java 1.6.0 Update 7. Each tests was allowed to run for at most 10 minutes.

Results: The results of all tests are shown in Table 7.2. Test which ran either out of memory or out of time are denoted with ∞ . The results are represented graphically from Figure 7.1 – 7.6 for each query from MOVIE and Univ ontologies. The execution time of each query is plotted on vertical axis while corresponding knowledge base is plotted on horizontal axis for KAON2, RACER and Pellet. The vertical lines exceeding the 600 (10 minutes) line are to indicate the results which ran out of memory in our configuration. The results represented on Figure 7.7 – 7.9 give graphical plot of the execution time required by each reasoner to answer the query for all 6 queries – 3 MOVIE queries and 3 Univ queries, against 5 different ontologies.

As our experiment results show, MOVIE and Univ-Bench do not pose significant problem for

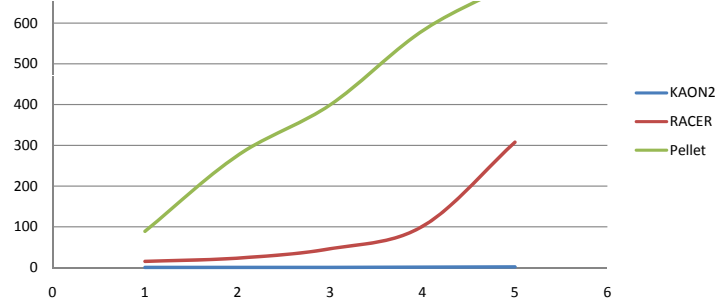
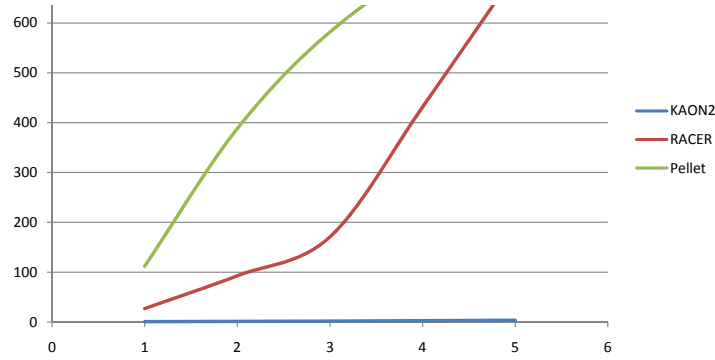
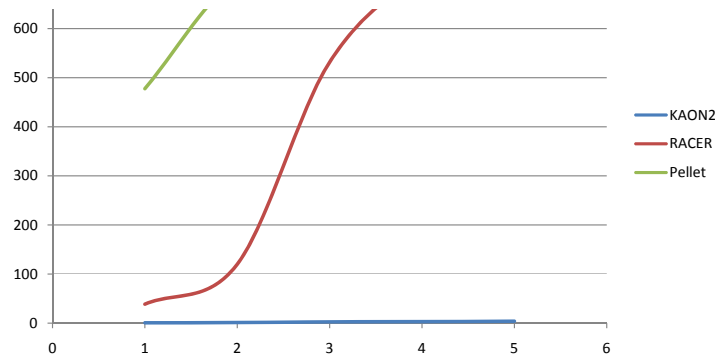
Ontology Name	Query	KB	KAON2	RACER	Pellet
MOVIE	$M_1(x)$	MOVIE(100)	0.31	15.0	89.07
		MOVIE(500)	0.37	23.01	275.08
		MOVIE(1000)	0.51	45.89	399.08
		MOVIE(5000)	0.83	101.0	581.01
		MOVIE(11435)	1.70	307.70	∞
MOVIE	$M_2(x, y)$	MOVIE(100)	0.45	27.05	112.13
		MOVIE(500)	0.80	93.01	388.30
		MOVIE(1000)	1.76	170.99	581.81
		MOVIE(5000)	2.97	431.50	∞
		MOVIE(11435)	3.77	∞	∞
MOVIE	$M_3(x, y, z)$	MOVIE(100)	0.77	38.65	477.52
		MOVIE(500)	1.10	120.00	∞
		MOVIE(1000)	2.68	531.66	∞
		MOVIE(5000)	3.11	∞	∞
		MOVIE(11435)	4.01	∞	∞
Univ	$U_1(x)$	Univ(1,0)	0.48	27.08	310.70
		Univ(2,0)	1.05	110.70	∞
		Univ(3,0)	1.89	537.82	∞
		Univ(4,0)	2.10	∞	∞
		Univ(5,0)	2.70	∞	∞
Univ	$U_2(x, y)$	Univ(1,0)	0.78	45.05	∞
		Univ(2,0)	1.08	115.03	∞
		Univ(3,0)	1.72	581.20	∞
		Univ(4,0)	2.83	∞	∞
		Univ(5,0)	3.81	∞	∞
Univ	$U_3(x, y, z)$	Univ(1,0)	1.10	41.70	∞
		Univ(2,0)	1.26	121.90	∞
		Univ(3,0)	2.37	∞	∞
		Univ(4,0)	3.08	∞	∞
		Univ(5,0)	4.77	∞	∞

Table 7.2: Performance table of queries over different knowledge bases

KAON2, namely, the translation produces an equality-free Horn program, which KAON2 evaluates in polynomial time. As given by the bar chart at Figure 7.7, the time required to answer a query for KAON2 grows moderately with the size of the data sets. If we compare results of Figure 7.7 with Figure 7.8 and 7.9, the execution time of KAON2 is still in terms of seconds while the RACER and Pellet performance is in minutes for most of the queries. For RACER and Pellet, as shown in bar charts at Figure 7.8 and 7.9, the query evaluation takes much time and in some cases they are beyond the computational criteria due to extensive ABox consistency checking before answering the first query. Similarly many optimization of tableau algorithms involve caching computation results, so the performance of query answering should increase with the each subsequent query. Furthermore, RACER and Pellet check ABox consistency before answering the first query, which typically takes much longer than computing query results. As KAON2 did not yet considering caching and does not perform a separate ABox consistency test and ABox inconsistency is discovered automatically during query evaluation; this is an advantage for the KAON2 to have significantly optimistic performance results.

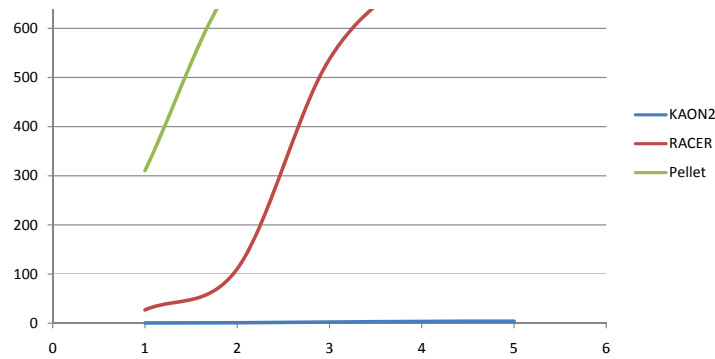
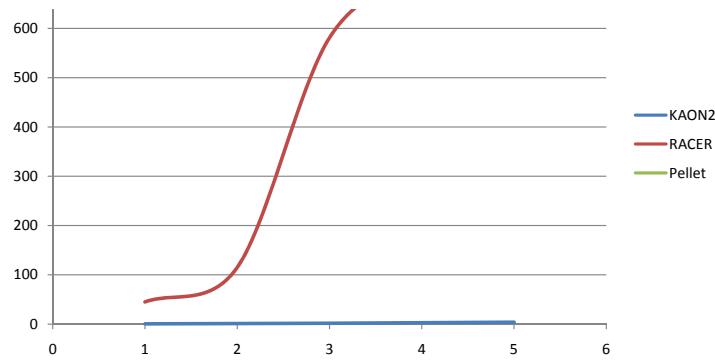
The ABox consistency checking is really important issue for some of the applications areas where checking validity of ABox with respect to the corresponding TBox is necessary for the reasoning

but this issue is not so important for query answering in semantic graph knowledge bases. For the applications which need extensive query answering, the consistency checking, if required, done at the time of ABox generation itself will help us in the query answering process.

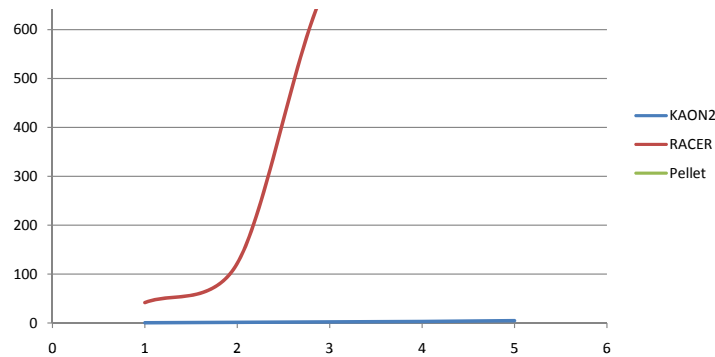
Figure 7.1: Movie query $M_1(x)$ Figure 7.2: Movie query $M_2(x, y)$ Figure 7.3: Movie query $M_3(x, y, z)$

7.3 Analysis

As we mentioned before, disjunctive datalog has been extensively studied and used in practice. Due to efficiency in finding a model, there are several such systems exist work on different semantic models. Although there are several disjunctive datalog engines exist, those engines are not suitable

Figure 7.4: Univ-Bench query $U_1(x)$ Figure 7.5: Univ-Bench query $U_2(x, y)$

for the integration to the system in KAON2. There are several reasons for difficulty in such integration: i) the reduction techniques produce only positive datalog programs – that is program without negation as failure and KAON2 do not rely on the minimum model semantics of disjunctive datalog. ii) model building is an important aspect of reasoning in disjunctive datalog. To compute the models, disjunctive datalog engines usually ground the disjunctive program and then look for the satisfiable models. Although the process has been optimized using intelligent grounding for the use in KAON2, grounding can be very expensive on large data sets. In contrast, although we use intelligent grounding to compute models, those computed model of the programs are of no interest in KAON2. iii) disjunctive datalog engines typically do not provide for the first order equality

Figure 7.6: Univ-Bench query $U_3(x, y, z)$

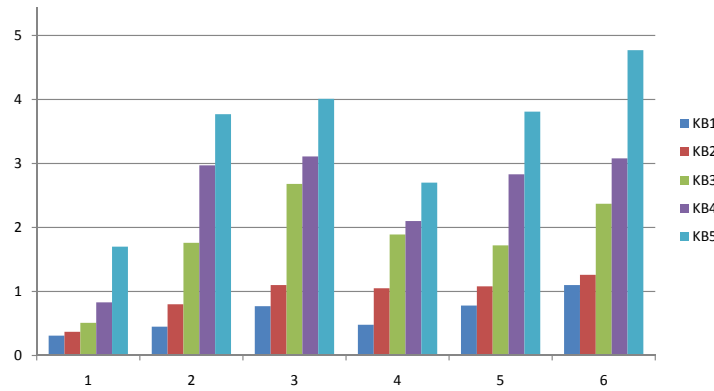


Figure 7.7: KAON2 performance over queries

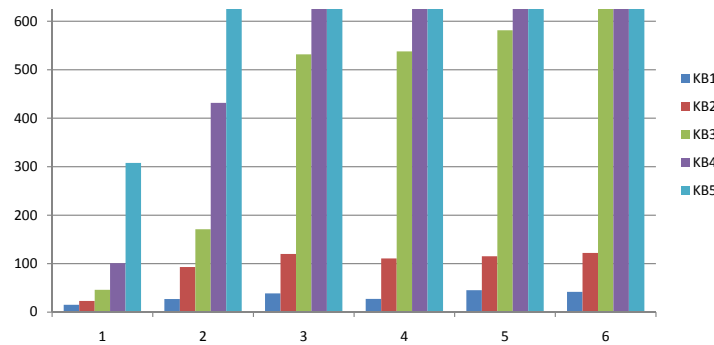


Figure 7.8: RACER performance over queries

predicate, which KAON2 uses to support number restrictions.

As initial results of using KAON2 for answering queries over very large ABoxes are promising, the system has some limitations. The use of horn fragment of *SHIQ* in KAON2 to get the polynomial complexity of reasoning on very large ABoxes make the KAON2 transformation in fact non-disjunctive. Hence for the most of the cases, where the transformation do not exhibit the horn program, the complexity of reasoning is still exponential.

From the perspective of other DL reasoners like RACER and Pellet, they really do not incorporate the concept of minimum model and stable model semantics ideas. That is, they are solely based on tableau proofs. The extensive use of tableau proofs support the TBox reasoning very

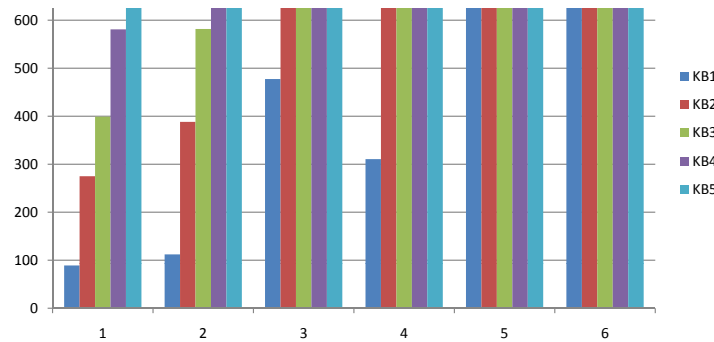


Figure 7.9: Pellet performance over queries

well, indeed.

Although TBox reasoning was not focus of our work, we performed some TBox reasoning tests on different available ontologies like Wine ontology. The Wine ontology is a fairly complex ontology with advanced DL constructors such as disjunctions and equality. We particularly measure the time required to compute subsumption hierarchies. From our experiments, the results indicated that the performance of TBox reasoning on KAON2 significantly lags behind the performance of the tableau reasoners.

7.4 Related Work

As already mentioned, the idea of supporting DL style reasoning using databases is not new. One example is [BB93], which can handle DL inference problems by converting them into a collection of SQL queries. The authors present the architecture and algorithms of a system that converts most of the inference made by the DBMS into a collection of SQL queries, thereby relying on the optimization facilities of existing DBMS to gain efficiency, while maintaining an object-centered view of the world with a substantive semantics and significantly different reasoning facilities than those provided by Relational DBMS and their deductive extensions. They also address a number of optimization issues that arise in the translation process due to the fact that SQL queries with different syntax but identical semantics are not treated uniformly by current database management systems. This approach is not limited to role-free ABoxes, but the DL language supported is much less expressive, and the database schema must be customized according to the given TBox.

Another example is the Parka system⁹ developed by the Parallel Understanding Systems Group at Computer Science Department of University of Maryland at College Park. The Parka system is a frame-based AI language which was designed to be supported efficiently using high-performance computing techniques. The goal of the project is to develop a fairly traditional Artificial Intelligence language/tool that can scale to the extremely large size applications mandated by the needs of today's information technology revolution. Parka is not limited to role-free ABoxes and can deal with very large ABoxes.

One of the key features of Parka is that it has been shown to efficiently handle its inferencing on KBs containing millions of assertions. Early work on the system gained most of its efficiency through the use of massive parallelism, however in recent years they've made increasing use of database management techniques to remove the need for parallelism. The latest version of Parka uses DBMS technologies to support inferencing and data management. In particular, this system, called "Parka-DB" was developed to run on generic, single processor (or parallel) systems with significantly less primary memory requirements than the previous versions. However, Parka also supports a much less expressive language, and is not based on standard DL semantics, so it is not really comparable to *iS* or to the other current approaches .

Furthermore, [Sch94] describes a semantic indexing technique that is very similar to the approach used in *iS* except that files and hash tables are used instead of database tables, and optimizations such as the use of equivalence sets are not considered. A persistent index into a

⁹<http://www.cs.umd.edu/projects/plus/Parka/>

large number of objects is built by classifying the objects with respect to a set of indexing concepts and storing the resulting relation between object-ids and most specific indexing concepts on a file and these files can be incrementally updated. These indexes can be used for efficient accessing the set of objects matching a query concept. The query is classified, and based on subsumption and disjointness reasoning with respect to indexing concepts, instances are immediately categorized as hits, misses or candidates with respect to the query. But the semantic indexing mechanism is highly dependent on reasoning with descriptions as provided by terminological axioms.

In the paper [PLTS08], Le Pham et al. investigated a new technique for optimizing DL reasoning in order to minimize the execution time and storage space requirements of a reasoning algorithm as much as possible. This technique is applied to speed up TBox and ABox reasoning. The incorporation of this technique with previous optimization techniques in current DL systems can effectively solve intractable inferences. That technique is called “overlap ontology decomposition”, in which the decomposition of a given ontology into many sub-ontologies is implemented such that the semantics and inference services of the original ontology are preserved. This paper is concerned about how to reason effectively with multiple KBs and how to improve the efficiency of reasoning over component ontologies. This technique is especially targeted for large TBoxes, hence it has less relevance to our problem.

Part IV

Summing Up

Chapter 8

Conclusions and Further Research

8.1 Conclusions

In this thesis, we proposed an approach based on disjunctive logic programming for identifying useful patterns and abnormal instances in large and complex semantic graphs, and reviewed the work that has been done from the description logic perspective which may help us in analysis of semantic graphs. We performed several experiments on existing DL reasoners considering the very large ABox reasoning on focus.

The contribution of the first part of the thesis is the formalization of semantic graphs and its corresponding ontology graph. We discuss formal definitions of semantic graphs, and its corresponding ontology graph considering their implementation issues and their use in intelligence analysis.

The contribution of the second part of the thesis is the development of disjunctive logic programming framework for the intelligence analysis. Although there are several existing supervised/unsupervised learning frameworks to identify anomalies from graph data, there has been little work aimed at discovering abnormal instances in very large semantic graphs whose nodes are richly connected with many different types of links from the logic programming perspective. We proposed the solution of this problem by designing a novel, disjunctive logic programming framework that utilizes the information provided by different types of nodes and links to identify useful pattern and abnormal instances. Our approach represents the dependencies between nodes and paths in the graph in first order logic predicates to capture what we call "semantic profiles" of nodes, and then applies disjunctive logic rules to find abnormal nodes and patterns that are significantly different from their closest neighbors. In a set of experiments on movies data, our system can almost perfectly identify the abnormal instances/patterns and outperforms several other state of the art machine learning methods that have been used to analyze the same data.

In the third part of the thesis, we describe some of the approaches from description logic perspective that may help analyze semantic graphs. As we already know, semantic graphs has associated ontology hierarchy, the systematic analysis using family of description logics is appealing due to the richer syntax and semantics available for the ontology description languages. In this scenario, we performed some experiments on semantic graph knowledge bases using KAON2, RACER and

Pellet systems and gave an overview of the whole process. Furthermore, we described some of the related work that has been done.

In summary, we discussed in detail disjunctive logic program based DLV system implementation of our approach to handle the issues related to semantic graphs to serve the purpose of extracting patterns and abnormal behaviors to help us ease the intelligence analysis in different areas. The formal analysis of these graphs using the logic based optimization techniques support the ongoing effort to extract relevant/important information reside in the semantic graphs, which is very useful for the intelligence analysis for the homeland security purposes.

On our experiments performed on representative natural data set in the Movie database, we showed that our framework not can only be applied to identify suspicious instances and patterns in intelligence analysis, but also to find abnormal patterns and instances in any relational graph. This leads to potential application in a variety of areas such as crime analysis, scientific discovery, data analysis and data cleaning.

8.2 Further Research

We conclude with four other possibilities for important future research directions. The first is to design a simple, seamless graphical front-end for our pattern finding framework which allows analysts to quickly and transparently pose and retrieve the results for the query graph.

The second one pertains to knowledge representation issues in semantic graphs: from a given data set, how can one determine which information is important and useful and how this information should be connected and represented to generate the semantic graph. This is an important problem, since the construction of the semantic graph can have a lot of impact on the results our system can discover.

The third problem is to provide our framework with the capability of dealing with temporal information or on dynamic semantic graphs whose behaviors changes on time, as information on the way things change over time can doubtlessly lead to abnormal and interesting findings. As semantic graphs include richer information on types of entities and different links between them, these graphs are not static, because data is changing over time as new connections are formed or discovered. Then, the questions we have to consider are how to visualize semantic graphs, how to find paths of connections in semantic graphs and determine if they are significant, how to effectively find instances of a particular subgraph within a large semantic graph, and how to find patterns or anomalies in the data indicating unexpected relations or features to investigate, etc.

The other area to look is regarding including ontologies on semantic graphs. We know that there is no single ontology, for which we should allow concepts to be used from different ontologies (e.g., “truck” from a transportation ontology and “diseases” from medical ontology).

We believe performing knowledge discovery in large, heterogeneous networks with a variety of different types of relations is an important new research direction with many potential applications. By reporting our methods and results in this thesis as one of the pioneer efforts to deal with abnormal instance discovery, we hope to draw more attention and motivate further ideas in this research domain.

The other areas for further analysis are: how to deal with multiple semantic graphs with respect to the ontology that constrains these graphs? The integration of multiple semantic graphs comes to the issue of ontology integration. As there is no one and only true associated ontology for all graphs, the query that needs to be analyzed on the basis of different graph cause the problem. We have two approaches to deal in that situation: i) import terms from one ontology to another, and ii) align an ontology.

We can still study further on how to effectively store and maintain very large graphs in the databases because it is the crucial part of the analysis as effective database storing provide us the significant improvement in the data loading for the specific query, how to validate the associated domain ontologies with respect to the semantic graph data, and how to translate/map between the graphs based on different ontologies, etc. Thus, the mechanism of semantic knowledge discovery plays vital role in the effective analysis of semantic graphs.

From the description logic perspective, development of a particular system that can only perform reasoning on the semantic graph will be the useful part of our work. In such particularly dedicated system, we can incorporate different query optimization and scalability techniques which are effective for the analysis of semantic graphs. The implementation of such type of system help intelligence analysts analyze the semantic graphs for different purposes.

Bibliography

- [ACM⁺04] J. Adibi, H. Chalupsky, E. Melz, A. Valente, and Others. The KOJAK group finder: Connecting the dots via integrated knowledge-based and statistical reasoning. *Innovative Applications of Artificial Intelligence Conference*, 2004.
- [AY01] Charu C. Aggarwal and Philip S. Yu. Outlier detection for high dimensional data. In *In Proceedings of the ACM SIGMOD International Conference on Management of Data*, pages 37–46, 2001.
- [BA83] H. Bunke and G. Allermann. Inexact graph matching for structural pattern recognition. *Pattern Recognition Letters*, (1):245253, 1983.
- [BB93] Alex Borgida and Ronald J. Brachman. Loading data into description reasoners. pages 217–226, 1993.
- [BCM⁺03] Franz Baader, Diego Calvanese, Deborah L. McGuinness, Daniele Nardi, and Peter F. Patel-Schneider, editors. *The Description Logic Handbook: Theory, Implementation, and Applications*. Cambridge University Press, 2003.
- [Ben02] Endika Bengoetxea. *Inexact Graph Matching Using Estimation of Distribution Algorithms*. PhD thesis, Department of Image and Signal Treatment, Ecole Nationale Supérieure des Telecommunications (ENST), Paris (FR), December 2002.
- [BERC05] Marc Barthelemy, Tina Eliassi-Rad, and Edmond Chow. Knowledge representation issues in semantic graphs for relationship detection. *American Association for Artificial Intelligence*, March 2005.
- [BFK⁺98] Robert Bihlmeyer, Wolfgang Faber, Christoph Koch, Nicola Leone, Cristinel Mateis, and Gerald Pfeifer. DLV – an overview. In *In Proceedings of the 13th Workshop on Logic Programming (WLP '98)*, 1998.
- [BIJ02] H. Blau, N. Immerman, and D. Jensen. A visual language for querying and updating graphs. Technical Report 2002-037, Department of Computer Science, University of Massachusetts Amherst, 2002.
- [Bor07a] Alex Borgida. A few words on graphs and semantics. From a Presentation at Workshop on Associating Semantics With Graphs, April 16–17 2007. <http://dydan.rutgers.edu/Workshops/Semantics/slides/graphSem-forPDF.1.pdf>.

- [Bor07b] Alex Borgida. Integrating information with ontologies. From a Presentation at Workshop on Associating Semantics With Graphs, April 16–17 2007. <http://dydan.rutgers.edu/Workshops/Semantics/slides/graphSem-forPDF.2.pdf>.
- [Bra79] Ronald J. Brachman. On the epistemological status of semantic networks. In Nicholas V. Findler, editor, *Associative Networks*, pages 3–50. Academic Press, 1979. Republished in [Brachman and Levesque, 1985].
- [BS98] Horst Bunke and Kim Shearer. A graph distance metric based on the maximal common subgraph. *Pattern Recogn. Lett.*, 19(3-4):255–259, 1998.
- [CF06] Deepayan Chakrabarti and Christos Faloutsos. Graph mining: Laws, generators, and algorithms. *ACM Comput. Surv.*, 38(1):2, 2006.
- [CFPL06] Francesco Calimeri, Wolfgang Faber, Gerald Pfeifer, and Nicola Leone. Pruning operators for disjunctive logic programming systems. *Fundam. Inf.*, 71(2,3):183–214, 2006.
- [CGM04] Thayne Coffman, Seth Greenblatt, and Sherry Marcus. Graph-based technologies for intelligence analysis. *Communications of the ACM*, 47(3):45–47, March 2004.
- [Cha07] Hans Chalupsky. Representing, reasoning with, and querying semantic graphs in powerloom. From a Presentation at Workshop on Associating Semantics With Graphs, April 16–17 2007. <http://dydan.rutgers.edu/Workshops/Semantics/slides/chalupsky.pdf>.
- [CL94] Marco Cadoli and Maurizio Lenzerini. The complexity of propositional closed world reasoning and circumscription. *J. Comput. Syst. Sci.*, 48(2):255–310, 1994.
- [CM04] T.R. Coffman and S.E. Marcus. Pattern classification in social network analysis: a case study. *Aerospace Conference, 2004. Proceedings. 2004 IEEE*, 5:3162–3175, March 2004.
- [CSH⁺05] Deng Cai, Zheng Shao, Xiaofei He, Xifeng Yan, and Jiawei Han. Community mining from multi-relational networks. In *In PKDD*, 2005.
- [D00] Sašo Džeroski, editor. *Relational Data Mining*. Springer-Verlag New York, Inc., New York, NY, USA, 2000.
- [EGM97] Thomas Eiter, Georg Gottlob, and Heikki Mannila. Disjunctive datalog. *ACM Trans. Database Syst.*, 22(3):364–418, 1997.
- [ELM⁺98] Thomas Eiter, Nicola Leone, Cristinel Mateis, Gerald Pfeifer, and Francesco Scarcello. Progress report on the disjunctive deductive database system DLV. In *FQAS '98: Proceedings of the Third International Conference on Flexible Query Answering Systems*, pages 148–163, London, UK, 1998. Springer-Verlag.
- [ERC05] Tina Eliassi-Rad and Edmond Chow. Using ontological information to accelerate path-finding in large semantic graphs: A probabilistic approach. *American Association for Artificial Intelligence*, 2005.

- [Fel98] Christiane Fellbaum, editor. *WordNet: An Electronic Lexical Database (Language, Speech, and Communication)*. The MIT Press, May 1998. <http://mitpress.mit.edu/catalog/item/default.asp?ttype=2&tid=8106>.
- [FMT04] Christos Faloutsos, Kevin S. McCurley, and Andrew Tomkins. Fast discovery of connection subgraphs. In *KDD '04: Proceedings of the tenth ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 118–127, New York, NY, USA, 2004. ACM.
- [Get03] Lise Getoor. Link mining: a new data mining challenge. *SIGKDD Explor. Newsl.*, 5(1):84–89, 2003.
- [GHM04] C. Gutierrez, C. Hurtado, and A. Mendelzon. Foundations of semantic web databases. In *ACM Symposium on Principles of Database Systems (PODS)*, 2004.
- [GJ90] Michael R. Garey and David S. Johnson. *Computers and Intractability; A Guide to the Theory of NP-Completeness*. W. H. Freeman & Co., New York, NY, USA, 1990.
- [GL91] M. Gelfond and V. Lifschitz. Classical negation in logic programs and disjunctive databases. *New Generat. Comput.*, 9:365–385, 1991.
- [Got94] Georg Gottlob. Complexity and expressive power of disjunctive logic programming (research overview). In *ILPS '94: Proceedings of the 1994 International Symposium on Logic programming*, pages 23–42, Cambridge, MA, USA, 1994. MIT Press.
- [GPH04] Yuanbo Guo, Zhengxiang Pan, and Jeff Heflin. An evaluation of knowledge base systems for large OWL datasets. In *In Proc. of the Third Int. Semantic Web Conf. (ISWC 2004), LNCS*, pages 274–288. Springer, 2004.
- [Gre88] Robert M. Mac Gregor. A deductive pattern matcher. pages 403–408, Saint Paul, Minnesota, 1988.
- [Gre07] Seth A. Greenblatt. Ontologies for graph matching: Practice and potential. From a Presentation at Workshop on Associating Semantics With Graphs, April 16–17 2007. <http://dydan.rutgers.edu/Workshops/Semantics/slides/greenblatt.pdf>.
- [HB99] S. Hettich and S. D. Bay. The UCI KDD archive [<http://kdd.ics.uci.edu>]. University of California, Department of Information and Computer Science, Irvine, CA, USA, 1999.
- [HJQW05] Wei Hu, Ningsheng Jian, Yuzhong Qu, and Yanbing Wang. GMO: A graph matching for ontologies. In *K-CAP Workshop on Integrating Ontologies*, pages 43–50, 2005.
- [HLTB04] Ian Horrocks, Lei Li, Daniele Turi, and Sean Bechhofer. The instance store: DL reasoning with large numbers of individuals. In *In Proc. of the 2004 Description Logic Workshop (DL 2004)*, pages 31–40, 2004.
- [HM00] Volker Haarslev and Ralf Moller. Expressive abox reasoning with number restrictions, role hierarchies, and transitively closed roles. pages 273–284. Morgan Kaufmann, 2000.

- [HM01a] Volker Haarslev and Ralf Moller. High performance reasoning with very large knowledge bases: A practical case study. pages 161–168, 2001.
- [HM01b] Volker Haarslev and Ralf Moller. Racer system description. pages 701–705. Springer, 2001.
- [Hor98] Ian R. Horrocks. Using an expressive description logic: Fact or fiction. In *In Proc. of KR-98*, pages 636–647. Morgan Kaufmann, 1998.
- [Hor00] Ian Horrocks. Practical reasoning for very expressive description logics. *Logic Journal of the IGPL*, 8:2000, 2000.
- [HR05] Robert A. Hanneman and Mark Riddle. *Introduction to social network methods*. University of California, Riverside, Riverside, CA, 2005.
- [HSM01] David J. Hand, Padhraic Smyth, and Heikki Mannila. *Principles of data mining*. MIT Press, Cambridge, MA, USA, 2001.
- [HST00] Ian Horrocks, Ulrike Sattler, and Stephan Tobies. Reasoning with individuals for the description logic SHIQ. pages 482–496. Springer-Verlag, 2000.
- [Hus04] Ullrich Hustadt. Reducing SHIQ - description logic to disjunctive datalog programs. pages 152–162. AAAI Press, 2004.
- [Isr07] David Israel. Some thoughts inspired by the workshop on associating semantics with graphs. From a Presentation at Workshop on Associating Semantics With Graphs, April 16–17 2007. <http://dydan.rutgers.edu/Workshops/Semantics/slides/israel.pdf>.
- [Jen07] David Jensen. *Proximity 4.3 QGraph Guide*. Department of Computer Science, University of Massachusetts Amherst, 2007.
- [JRB03] David Jensen, Matthew Rattigan, and Hannah Blau. Information awareness: a prospective technical assessment. In *KDD '03: Proceedings of the ninth ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 378–387, New York, NY, USA, 2003. ACM.
- [KABK08] Ian L. Kaplan, Ghaleb M. Abdulla, S Terry Brugger, and Scott R. Kohn. Implementing graph pattern queries on a relational database. Technical Report LLNL-TR-400310, Lawrence Livermore National Laboratory, January 2008.
- [Kap06] Ian Kaplan. A semantic graph query language. Technical report, Complex Networks Group, LLNL, 2006.
- [Kre02] Valdis E. Krebs. Mapping networks of terrorist cells, 2002.
- [KYL04] Duck Hoon Kim, Il Dong Yun, and Sang Uk Lee. A new attributed relational graph matching algorithm using the nested structure of earth mover’s distance. In *ICPR '04: Proceedings of the Pattern Recognition, 17th International Conference on (ICPR'04) Volume 1*, pages 48–51, Washington, DC, USA, 2004. IEEE Computer Society.

- [LC03a] Shou-De Lin and Hans Chalupsky. Unsupervised link discovery in multi-relational data via rarity analysis. In *ICDM '03: Proceedings of the Third IEEE International Conference on Data Mining*, page 171, Washington, DC, USA, 2003. IEEE Computer Society.
- [LC03b] Shou-De Lin and Hans Chalupsky. Using unsupervised link discovery methods to find interesting facts and connections in a bibliography dataset. *SIGKDD Explor. Newsl.*, 5(2):173–178, 2003.
- [LC07] Shou-De Lin and Hans Chalupsky. Discovering and explaining abnormal nodes in semantic graphs. *IEEE Transactions on Knowledge and Data Engineering*, 2007.
- [LD94] Nada Lavrac and Saso Dzeroski. *Inductive Logic Programming: Techniques and Applications*. Ellis Horwood, New York, USA, 1994.
- [LGMF04] Jure Leskovec, Marco Grobelnic, and Natasa Milic-Frayling. Learning sub-structures of document semantic graphs for document summerization. In *LinkKDD*, August 2004.
- [Lif85] Vladimir Lifschitz. Closed-world databases and circumscription. *Artif. Intell.*, 27(2):229–235, 1985.
- [Lin06] Shou-De Lin. *Modeling, searching, and explaining abnormal instances in multi-relational networks*. PhD thesis, Los Angeles, CA, USA, 2006. Adviser-Kevin Knight and Adviser-Hans Chalupsky.
- [LPF⁺06] Nicola Leone, Gerald Pfeifer, Wolfgang Faber, Thomas Eiter, Georg Gottlob, Simona Perri, and Francesco Scarcello. The dlv system for knowledge representation and reasoning. *ACM Trans. Comput. Logic*, 7(3):499–562, 2006.
- [Min07] Kim Minuzzo. Biodefense knowledge center(bkc) technology overview. From a Presentation at Workshop on Associating Semantics With Graphs, April 16–17 2007. <http://dydan.rutgers.edu/Workshops/Semantics/slides/minuzzo.pdf>.
- [MJ03] A. McGovern and D. Jensen. Identifying predictive structures in relational data using multiple instance learning. In *Proceedings of the Twentieth International Conference on Machine Learning*, 2003.
- [MMT⁺02] R. Mooney, P. Melville, L. Tang, J. Shavlik, I. Dutra, D. Page, and V. Costa. Relational data mining with inductive logic programming for link discovery. In *Proceedings of the National Science Foundation Workshop on Next Generation Data Mining*, Baltimore, Maryland, USA, 2002.
- [Mor02] Katharina Morik. Detecting interesting instances. In *in Proceedings of the ESF Exploratory Workshop on Pattern Detection and Discovery*, pages 13–23. Springer, 2002.
- [NAJ03] J. Neville, M. Adler, and D. Jensen. Clustering relational data using attribute and link information. In *Proceedings of the Text Mining and Link Analysis Workshop, Eighteenth International Joint Conference on Artificial Intelligence*, 2003.

- [New03] M. Newman. The structure and function of complex networks. *SIAM Review* 45(2), 2003.
- [NM01] Natalya F. Noy and Deborah L. McGuinness. *Ontology development 101: A guide to creating your first ontology*. Online, 2001.
- [OWL] OWL web ontology language reference. In Mike Dean and Guus Schreiber, editors, *W3C Recommendation 10 February 2004*.
- [PLTS08] Thi Anh Le Pham, Nhan Le-Thanh, and Peter Sander. Decomposition-based reasoning for large knowledge bases in description logics. *Integr. Comput.-Aided Eng.*, 15(1):53–70, 2008.
- [Qui67] M. Ross Quillian. Word concepts: A theory and simulation of some basic capabilities. In *Behavioral Science*, volume 12, pages 410–430, 1967. Republished in [Brachman and Levesque, 1985].
- [Rap02] William J. Rapaport. Holism, conceptual-role semantics, and syntactic semantics. *Minds and Machines*, 12:3–59, 2002.
- [RDFa] Resource description framework (RDF): Concepts and abstract syntax. In Graham klyne and Jeremy J. Carroll, editors, *W3C Recommendation 10 February 2004*.
- [RDFb] RDF primer. In Franc Manola and Eric Miller, editors, *W3C Recommendation 10 February 2004*.
- [Rob65] J. A. Robinson. A machine-oriented logic based on the resolution principle. *J. ACM*, 12(1):23–41, 1965.
- [Rod07] Marko A. Rodriguez. Social decision making with multi-relational networks and grammar-based particle swarms. In *HICSS '07: Proceedings of the 40th Annual Hawaii International Conference on System Sciences*, page 39, Washington, DC, USA, 2007. IEEE Computer Society.
- [SBF⁺07] David Silberberg, Wayne Bethea, Paul Frank, John Gersh, Dennis Patrone, David Patrone, and Elisabeth Immer. Ontology-assisted query of graph databases. From a Presentation at Workshop on Associating Semantics With Graphs, April 16–17 2007. <http://dydan.rutgers.edu/Workshops/Semantics/slides/silberberg2.pdf>.
- [Sch94] Albrecht Schmiedel. Semantic indexing based on description logics. In *Proceedings of the KI94 Workshop KRDB94*, pages 41–44, 1994.
- [Sco00] John P. Scott. *Social Network Analysis: A Handbook*. SAGE Publications, January 2000.
- [SG02] Ted E. Senator and Henry G. Goldberg. Industry: break detection systems. *Handbook of data mining and knowledge discovery*, pages 863–873, 2002.

- [Sil06] David Silberberg. The graph query language. From a presentation at the Lawrence Livermore National Laboratory, July 18 2006. www.xmdr.org/presentations/Silberberg-Graph\%20Query\%20Language-July\%2018\%202006.ppt.
- [Spa91] M. K. Sparrow. The application of network analysis to criminal intelligence: An assessment of the prospects. *Social Networks*, 13:251–274, 1991.
- [Ull76] J. R. Ullmann. An algorithm for subgraph isomorphism. *J. ACM*, 23(1):31–42, 1976.
- [XC05] Jennifer Xu and Hsinchen Chen. Criminal network analysis and visualization. *Communications of the ACM*, 48(6):101–107, June 2005.
- [YNM91] J. Yen, R. Neches, and R. MacGregor. CLASP: Integrating term subsumption systems and production systems. *IEEE Trans. on Knowl. and Data Eng.*, 3(1):25–32, 1991.