Qualifying Exam Report

Structural Feature Extraction in Graph-Structured Databases

Michel Leonard Goldstein

Summary

Graph-structured databases are gaining popularity lately due to their ease of creation, maintenance and representational power. However, very little research actually can be found in employing these databases for complex tasks such as pattern recognition in databases. In order to perform pattern recognition, it is necessary to extract and exploit the features found in the database. Features can be explicit (usually related to elements called properties of a certain object in the database), structural (related to the way the elements are interconnected) or syntactic (related to a certain grammar to form certain connection patterns). This report presents the results of the analysis of the structural features. With the novel methods proposed for extracting structural features, it is possible to perform some pattern recognition tasks, before unattainable by previous explicit features.

With the same mathematical framework used for structural feature extraction, the research reported here has shown a novel method for transforming database structure in order to accommodate the use of legacy and third-party databases for novel pattern recognition methods. This method is both robust and simple to implement, because it uses the concept of equivalence of semantics between the databases in a step-by-step simplified process.

Scalability and efficiency are two other major concerns when dealing with graph-structured databases. This document presents also a novel modification to the proposed feature extraction algorithm that employs the concept of *sampling*, rarely used for feature extraction. This method can compromise feature accuracy by processing time and memory use. In order to deal with complex networks, a novel evolutionary algorithm-based sampling method was proposed to effectively use heuristics related to the position of high-valued features in order to bias the sampling towards obtaining the values of these features earlier.

Two broad-scale examples of application of the proposed method are shown in order to consolidate the concepts presented and show the effectiveness of the proposed method for obtaining features not easily obtainable using other methods.

Table of Contents

1. Introduction 1
   1.1. Graph-Structured Databases 2
   1.2. Feature Extraction in Databases 7
   1.3. Complex Networks 10
   1.4. Proposed General Pattern Recognition System 12
   1.5. Structure of the Document 17
2. Manipulation of Graph-Structured Databases 18
3. Structural Feature Extraction 25
   3.1. Introduction 25
   3.2. Proposed Concept of Structural Feature Extraction 28
   3.3. Pattern Recognition 30
      3.3.1. Explicitly defined walk semantics 31
      3.3.2. Example-based pattern definitions 33
      3.3.3. Mixed pattern definitions 35
   3.4. User Interface 36
   3.5. Dimensionality Reduction 39
   3.6. Syntactic Features 41
   3.7. Sample Application Examples 43
      3.7.1. Analysis of weight policy on explicitly defined patterns 44
      3.7.2. Explicitly defined patterns 46
      3.7.3. Example-based patterns 47
   3.8. Summary 48
4. Ontology Transformation 50
   4.1. Introduction 50
   4.2. Ontology Transformation Algorithm 51
      4.2.1. Source transformation 53
      4.2.2. Path transformation 54
      4.2.3. Target element disambiguation 55
   4.3. Query Optimization by Transformation 59
   4.4. User Interface 60
   4.5. Sample Application Examples 62
      4.5.1. Earthquake event analysis 62
      4.5.2. Scientific papers database 65
   4.6. Summary 71
5. Approximating Features by Sampling 73
   5.1. Introduction 73
   5.2. Sampling for Approximating Structural Features 74
      5.2.1. The start-finish approach 74
      5.2.2. The start-path-finish sampling procedure 76
   5.3. Naïve Sampling Policy 76
   5.4. Branching Factor in Real-World Databases 79
      5.4.1. Branching factor analysis of graph models 79
   5.5. Real-World Examples of Branching Factors 84
   5.6. Evolutionary Algorithm Approach for Sampling 87

5.6.1. Local mutation 88
5.6.2. Global mutation 90
5.6.3. Selection 90
5.7. Sample Experimental Results 91
5.7.1. Error functions 91
5.7.2. Sampling collections of journal papers 93
5.7.3. Sampling earthquake event databases 94
5.8. Summary 95
6. Implementation Considerations 97
6.1. Introduction 97
6.2. Graph-Structured Database API Definition 98
6.3. Implementation Employing Relational Database Infrastructure 101
6.4. Increasing Efficiency by Identification of Sub-Walk-Semantics 103
7. Application Examples 106
7.1. Introduction 106
7.2. Author Disambiguation in Collections of Journal Papers 106
7.3. Identification of Earthquake Hot Spots 110
8. Conclusions 112
Appendix I – Ontologies Used 123
Appendix II – API for Graph-Structured Database Analysis 128

List of Tables

Table 3-1 - Comparison of different weight policies for identifying important authors in Evolutionary Computation 45
Table 3-2 - Results of co-authorship collaboration without preferential connection correction 47
Table 3-3 - Results of co-authorship collaboration without preferential connection correction using examples for defining patterns 48
Table 4-1 - Top 10 results of the analysis of active earthquake regions 64
Table 4-2 - Top 10 results of the analysis of active earthquake regions using only major earthquakes and integer positions 65
Table 4-3 - Example of some interesting equivalent walk semantic pairs 68
Table 4-4 - Semantic star definition for disambiguation of papers and references 69
Table 5-1 – Pseudo-code for Naïve sampling policy 77
Table 5-2 - Quantiles of the empirical distributions 86
Table 5-3 - Pearson's r correlation coefficient between partitions for Anthrax and earthquake datasets 87
Table 5-4 – Pseudo-code for Local Mutation 89
Table 5-5 - Comparison of the N-S and EA-S for the anthrax database 94
Table 5-6 - Comparison of the N-S and EA-S for the anthrax database 95
Table 6-1 - Pseudo-code for identification of sub-walks within a walk semantics 105
Table 7-1 - Results from author disambiguation 109
Table 7-2 - Top 10 tabular result for earthquake "hot spots" 111

List of Figures

Figure 1-1 - Example of (a) ontology and (b) data for a conceptual network of people and friends 4
Figure 1-2 - Example of different types of networks. (a) random network; (b) preferentially connected network..... 11
Figure 1-3 - Ontology-based pattern recognition system 13
Figure 3-1 - Simplified version of ISI database structure for journal articles 27
Figure 3-2 - Modified structure for papers database..... 27
Figure 3-3 - Main pattern recognition panel showing the results of a simple example-based analysis showing the author bibliographic coupling for a given pair, highlighting a specific paper with its inbound and outbound vertices..... 36
Figure 3-4 - Filtering dialog – users can view only specific elements in the output list .. 37
Figure 3-5 - Graphical exploration of a single element – users can select which neighboring or indirect element types to show by clicking on an element type and then selecting from a dialog box the walk semantics from a manually pre-defined set of interesting walk semantics 37
Figure 3-6 - Feedback from user to add positive and negative examples for example-based pattern recognition 38
Figure 3-7 - Side-by-side comparison of ranking given two different feature extraction/pattern recognition procedures for the same group of elements. 38
Figure 3-8 - Operations on graph-structured databases 41
Figure 4-1 - Example of two different ontology structures for the same problem domain (collection of journal articles)..... 52
Figure 4-2 - Overview of proposed transformation process 53
Figure 4-3 - Pseudo-code for generating the transformation using the equivalent walk pairs.....55
Figure 4-4 - Example of the merging process..... 57
Figure 4-5 - Example of interface for building the equivalent walk semantics..... 61
Figure 4-6 - Example of interface for confirming merge of virtual elements 61
Figure 4-7 - Example of interface for doing the disambiguation of database elements ... 62
Figure 4-8 - Simplified graphical representation of ontology structure of an earthquake event database 63
Figure 5-1 - Example of artifacts caused by start-finish sampling 75
Figure 5-2 - Behavior of the convergence of the minimum branching probability for a real-world database (Anthrax papers database). N_{init} = 1,000 and N_{samp} = 1,000..... 78
Figure 5-3 - Sample Zeta distribution for single partition, with $\gamma = 2.0$ 80
Figure 5-4 - Sample branching factor distribution of 4 parititions with degrees following Zeta distribution with parameter $\gamma = 2.0$ 81
Figure 5-5 - Sample degree distribution of single randomly connected partition with p = 0.4 and $N = 50$ 82
Figure 5-6 - Sample branching factor distribution of randomly connected 4-partite network with $p = 0.4$ and $N = 50$ 83
Figure 5-7 - Log-log plot of the branching function probability distribution for Zeta-distributed partitions 83

Figure 5-8 - Empirical probability distribution of the branching factor for the Anthrax dataset for the author bibliographic coupling feature 84
Figure 5-9 - Empirical probability distribution of the branching factor for the earthquake dataset for the active earthquake areas..... 85
Figure 5-10 - Walk semantics for identification of active earthquake areas 85
Figure 5-11 - Walk to represent important authors in a journal database..... 86
Figure 5-12 - Block diagram of EA-based sampling policy 88
Figure 6-1 - Simplified UML view of the main elements of the GSDB-API..... 99
Figure 6-2 - Database tables for the ontology engine 102
Figure 6-3 - Definition of tables for storing the ontology. The <ID> is the ontology ID from the Ontologies table..... 102
Figure 6-4 - Database tables for storing graph-structured databases. The <ID> is the database ID from the Databases table 103
Figure 6-5 - Abstract representation of a projection divided into three subprojections . 104
Figure 6-6 - Walk semantics and projection for identifying earthquake "hot spots"..... 105
Figure 7-1 - Definitions of walk semantics used for author disambiguation..... 108
Figure 7-2 - Walk semantics for identification of earthquake "hot spots"..... 110
Figure 7-3 - Map of all top 1000 earthquake locations in the world 111
Figure 7-4 - Map of top earthquake locations in the West Pacific highlighting most active regions within the region 111

CHAPTER 1

Introduction

This report presents a vital step towards pattern recognition in graph-structured databases by proposing methods for dealing with structural feature extraction. Structural feature extraction is the process of obtaining quantitative measures of qualitative features related to how the elements in the database are connected to each other. This issue is extremely important in a large number of problems. For example, in collections of journal articles, the number of times a certain author's papers reference a specific article or set of articles is essential to define the area of research of this author than just using information of the names of the papers referenced. Another important area of application of this method is on the analysis of social networks [1]. For instance, accurate analysis of the behavior of social groups is critical for predicting impact of diseases in the population. The analysis of formation and operation of terrorist groups also benefits greatly from social network analysis. In this field of research, not only the existence of contact between people is important, but the frequency of this social contact. The latter is regarded as a structural feature of the database.

With the framework developed for dealing with structural feature extraction, it is also possible to perform ontology mapping, i.e. the transformation of the structure of databases. This process is of extreme importance when integrating data from third-party and legacy databases, in which structures were not created to deal with the requirements of the system in mind.

In summary, the contributions of the research reported in this document are as follows:

1. New formalization and analysis of features in graph-structured databases.
2. Creation of simple algorithms for pattern recognition in graph-structured databases that identify explicitly-defined and example-based patterns.

3. Development of a method for extracting structural features in graph-structured databases that contains basic support for syntactic feature extraction.
4. Use of formal definition of features for performing database structure transformation in order to adapt third-party or legacy databases to project processing requirements.
5. Development of a method for approximating database structural features by using sampling of graph structures through an optimal "black-box" sampling policy and an evolutionary algorithm-based sampling policy.
6. A novel method of semi-automatic method for disambiguation of author names in databases of journal collections by using structural features and simple text similarity ranking.

1.1. Graph-Structured Databases

With the increase in wide-spread use of databases in various complex applications, well-established database system approaches became ill-fitted for dealing with the current processing demands. These approaches have shown to be hard to evolve, interoperate, and merge with other databases, and are especially difficult to understand [2]. Understandability is a two-fold concept: user understandability, which leads to less costly databases to construct and maintain; and computer understandability, related to the ability of applying automatic and semi-automatic reasoning routines to the database to spot trends and inconsistencies [3]. A solution to this problem that is gaining popularity in recent years is the use of ontology languages, standardized methods of defining data structure.

The most well-known definition of *ontologies* is the one given by Gruber [4] as "*an explicit specification of a conceptualization*." In other words, ontologies are used to enable the formal representation of the concepts behind the knowledge that is to be represented. These concepts are usually thought as the interrelation between the different elements of the subset of the reality that is being represented.

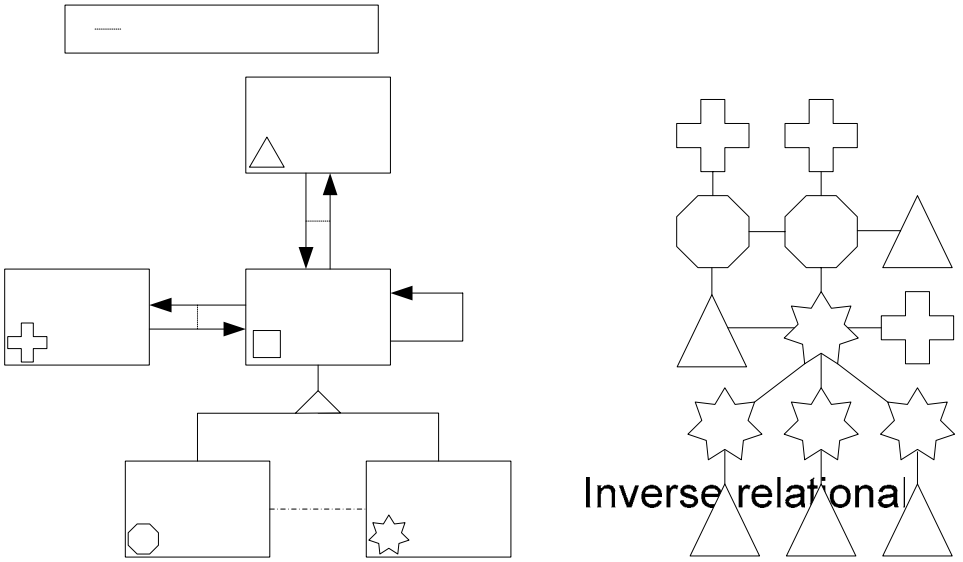
Although the concept of ontology as the structure in which reality is defined dates back to early philosophy, formal algorithms and computational methods appeared only with the emergence of the artificial

intelligence research. These early methods were highly problem-dependent. Initiatives for standardizing these representations are relatively recent. Most current standards are based on first order logic applied on frames or on description logics (DL) [5]. Current ontology language research is fueled by the Semantic Web initiative [6] and is thus based on web technologies, mainly XML, *extensible markup language*. Among the currently used languages, this study will focus on the current state-of-the-art for the Semantic Web, OWL, the Web Ontology Language, recently published by the World Wide Web Consortium (W3C) as a recommendation [7].

OWL is built on two other W3C recommendations: XML and RDF(S) (Resource Description Framework and Schema). This fact facilitates the implementation of ontologies by using already well-established standards. XML defines basic text semantics, organizing the data files as tagged text files. RDF [8] defines a way to describe elements using general triples (subject, verb, object). In other words, it represents concepts based on a graph data model, where the verb is an edge connecting two concepts, the subject and the object. RDFS [9] adds the ability for the definition of vocabularies using RDF, it makes it possible to define meta-data using RDF (class and property type definition). Currently RDF and RDFS are merged in one single W3C recommendation.

Although some skepticism still exists about the restrictions that these standards impose on an ontology representation language, especially the restriction to only use triples [10], these standards are being backed up by important industrial and academic partners. Assessment of the usefulness and relative merits of these standards is outside the scope of this study. The goal of the research covered by this report is to propose and analyze methods that deal with graph-structured databases, and the current standards presented above for data representation are in the end generating this class of databases. In order to support the convergence of databases to using standardized ontologies for representing the structure, it is necessary to provide methods for performing important database processing operations in the resulting graph-structured databases. The scope of this research is to provide one of these database operations, namely structural feature extraction.

Figure 1-1 shows a very simple example of an ontology and the corresponding data. The graphical representation of the ontology does not follow any standardized language, but contains concepts borrowed from the OWL standards.



(a) (b)

Figure 1-1 - Example of (a) ontology and (b) data for a conceptual network of people and friends

The ontology (Figure 1-1(a)) contains *class types* (the rectangles) that represent the types of objects in the database, and *property types* (the arrows) that represent the type of relations that the elements can have among each other. These are called the *ontology elements*. The links between the ontology elements represent special semantic connection between these elements. In this document only three semantic connections will be considered: inheritance (the connection between a person type and a man and a woman element types), disjoint classes (between a man and a woman in this example), and inverse properties. The data (Figure 1-1(b)) contains *objects* (the closed objects) and *properties* (the lines). These are *instances* of the types defined in the ontology, i.e., they are specific cases of the concept presented in the ontology that exist in the "real-world." The symbols inside the class types in the ontology are used when displaying the data to represent the object type. For instance, the element "Peter" is represented as a hexagon, thus it is an instance of a "Man" class type.

Technically, inverse properties are properties that if one property has an instance, i.e., an entry in the database, connecting object A to B, then there has to be an instance of its inverse also, connecting B to A. To simplify the representation of the data, when there is no uncertainty, a single line will be used in the data to represent an instance of the pair of inverse property types. This simplifies the data representation greatly. In few cases when uncertainty is involved (see example in Section 3.7), legends will be used to explicitly define the type of elements that the property is an instance from.

Disjoint element types are defined so that there cannot be any object in the database that is an instance of both types simultaneously. This visual method can facilitate the understanding of the database, but it might increase the difficulty to present for very large databases with a large number of ontology elements, especially when there are many cases of multiple instantiation. One simple example when this could be an issue is if another class type in the ontology is a profession. An instance of a person can also be an instance of a profession. For the databases being used in this study, this is not an issue, thus this will be the standard used for representing ontologies and graph-structured data throughout this document. It has to be emphasized, however, that the algorithms proposed in this document do not have this limitation, only the graphical representation method employed.

The graph structure of the database can be easily observed in this simple example. Some elements do not contain all the properties defined in the ontology. This may be due to incompleteness of the database, or to the lack of this piece of information. For example, in a database of students non-American international students may not have a Social Security Number. Despite the natural representation of the ontology-structured database as a graph, most ontology applications still use a relational database as the basis for their data storage. This is mainly because there is little research being conducted on graph-structured databases in respect to efficient database engines, i.e. off-the-shelf software for storing and searching graph-structured databases; and the incorporation of this paradigm in programming languages. It is not within the scope of this research to seek a solution to this problem. Despite the assumption of a purely graph-structured database, its current implementation relies on relational databases for element persistence. More details about the implementation will be given in Chapter 6.

Graph-oriented databases are similar to object-oriented databases in the sense that they contain a definition of classes (the ontology) and instances of these classes (the data) [11]. Object-oriented database research is fairly advanced. However, there is still a very low amount of adoption of this technology in the industry, mainly due to the need for training and transformation of large databases, and the lack of adequate commercial software to support very large databases [12]. There is an important difference between graph-structured and object-oriented databases, though: that graphs only contain data and relations, while object-oriented databases are able to store functions related to each class also. Most object-oriented databases, mainly because of possible conflicts that the existence of functions may cause, do not support multiple inheritance, an important feature of graph-structured databases [13]. Therefore, despite the fact that there are many commercial implementations of object-oriented databases, and most modern languages support object-oriented design, it is not possible to emulate graph-structured databases employing these databases.

An initial analysis on pure graph-oriented databases was published by Gyssens *et al*. [14], who present a graph-oriented object database model with methods for storing and querying graph-structured databases. Their model makes a clear distinction between data objects (objects that represent numerical or text values) and entity objects (objects that have abstract representation in the computer, usually containing a group of data objects for identification), in a very similar way that relational databases do [15]. A transformation language is also proposed based on four elementary graph manipulation functions, node addition, node deletion, edge addition and edge deletion. It also defines a method to call operation that is inspired by object-oriented designs. The concept of this project is very interesting and has been highly influential on graph-structured and semi-structured database research.

A fair amount of research exists in defining graph (and tree) query languages [16]. Most of these are focused on dealing with semi-structured databases, such as XML-structured databases or BibTex files (bibliographic files for LaTeX document preparation language). Below is a short review and analysis of key proposed languages.

Consens and Mendelzon [17] proposed one of the first graph-structured database query language, GraphLog. This language is based on first-order logic and is restricted by its global concepts. The query language, because of its high expressiveness, is sometimes very cumbersome to use, requiring very complex expressions for fairly simple queries.

Paredaens *et al*. [18] have proposed to combine the power of logic for expressiveness, objects for modeling power and graphs for representing data and queries. The whole query language is based on color and pattern-coded boxes, ellipses and lines. Although this makes most queries easy to understand, more complex queries can become quite cumbersome to build and analyze. Moreover, automatic generation of user-understandable queries is restricted by the ability of element placement in graphs.

Cardelli and colleagues [19] presented a method for querying graphs based on Spatial Logic [20, 21]. Their proposed query language is based on pattern matching and recursion. The query language is fairly concise and shows large similarity with their proposed graph description language, which makes the construction of the queries easier. However, the recursion ability tends to makes the expressions difficult to understand and, thus, to debug. Misuse of recursion can also cause very serious efficiency problems, also observed in recursive query languages for relational databases [22].

Formal definitions of graph-structured databases focusing on the operations of interest will be presented in Chapter 2. The next section presents the goals and current developments in the field of feature extraction in databases.

1.2. Feature Extraction in Databases

Pattern recognition is "a problem of estimating density functions in a high-dimensional space and dividing the space into regions of categories or classes" [23]. On important step in the pattern recognition process is of feature extraction. The objective of the *feature extraction* step is to identify and extract quantitative values that assist in discerning the patters of interest. For example, in a voice signal, it is interesting to

define if the person who is talking is a male or female. One possible feature would be the frequency of the voiced signals. Typically, males have lower pitch than females because their vocal folds are longer and more massive [24].

Features vary in degrees of complexity for extraction. For example, for the same problem in the previous paragraph, the vocabulary used in the speech can also be used to indicate the gender of the speaker. However, if what is obtained is the voice stream, the transformation from this voice stream to a dictionary-based multi-dimensional representation is highly complex (not taking into consideration the overall feasibility of the process with current state-of-the-art algorithms). The ability to use it to classify the pattern, also called separability or classifiability of the patterns given the features [25] is another important concern. For instance, the ability to classify the gender of the speaker based on the content is highly varying with the subject of the conversation [26].

Another important phenomenon that has to be taken into consideration when performing feature extraction for pattern recognition is the "curse of dimensionality" [27]. It has been shown that by increasing the number of features of a problem, the search space increases exponentially and the classifiability diminishes exponentially (due to the compound increase in noise-related errors). Thus it is necessary to correctly choose the minimal features that provide the highest separability. There are many methods devised to reduce the dimensionality of the problem while controlling the loss of information in the process. A brief discussion about these existing methods will be provided in Section 3.5.

Although abundant research exists about feature extraction of signals, such as image and speech signals, very little research was devoted in extracting features from database content. The main difference between extracting features from signals and from databases is that most of the database features are categorical and isolated, while signals are sparse. *Categorical* features contain values that are discrete and cannot be compared. For instance, the name of a person is a categorical feature of this person. *Isolated* features, as opposed to *sparse* features are features obtained directly from single values or structures in the database. *Sparse* features cannot be characterized without considering a large group of values, usually related with

each other by time connection. For example, when analyzing features in the vibration generated from a helicopter engine to identify faulty behavior [28], the features of interest are related to the time correlation of the signal observed. However, when observing a collection of journal papers to extract the most important authors, the features of interest is the number of papers published by the author and the times these papers were referenced by other papers, or the institution where the author works. This difference generates a completely distinct requirement for the processing of this type of data.

Features in databases can be divided into three different types: explicit features, structural features, and syntactic features.

- **Explicit features** are related to numeric or categorical features directly connected to an element of the database. For example, in the aforementioned identification of important authors, the institution where the author works is an explicit feature that can be regarded as an important feature.
- **Structural features** are related to the nature of the connections between certain elements of the database. For example, the number of papers published by an author in the database is not a field in the database, but a number connected to the amount of connections that exist between the author and paper elements. Likewise, the number of times the papers written by the author are referenced is a feature that is related to the number of connections that exist between an author and papers, then by these papers and other papers through citation links. A more detailed explanation of these types of features will be given in Chapter 3.
- **Syntactic features** are features that are related to the adherence of a certain data to a grammar [29]. This can be seen as a more formal method for defining patterns as being formed by explicit production rules. For example, when analyzing social relations, it is sometimes interesting to define as a single feature people that are directly and indirectly associated (because indirect associations may represent direct associations that were not extracted). This can be done easily by defining grammar rules and weights to production rules. More information about these types of features will be provided in Section 3.6.

As explicit features do not require special methods for extracting, the scope of the methods proposed in this document is on dealing with structural features, i.e., features directly related to the connectionist nature of the elements in the database.

A very common issue observed when dealing with the nature of the connection between the elements on a database is that most graph-structured databases present a connection distribution that forms a complex network. The next section will introduce this concept.

1.3. Complex Networks

The term "complex networks" is usually vaguely defined in the literature. One of a few more specific definitions was given by Caldareli [30]: "*complex networks are sets of many interacting components whose elaborate collective behavior cannot be directly predicted and characterized in terms of the behavior of their individual components*."

Unlike random networks, complex networks display a large variance in the graph parameters so that by observing a small area in the graph it is impossible to predict how the rest of the graph looks like. Random networks usually contain a fairly homogeneous connectivity between elements, while complex networks present phenomena such as preferential attachment. *Preferential attachment* is a feature of the dynamic process of network growth in which the elements with the most connections have greater chances of getting a new connection. Figure 1-2 shows a simple example of the difference between a random network and a complex network.

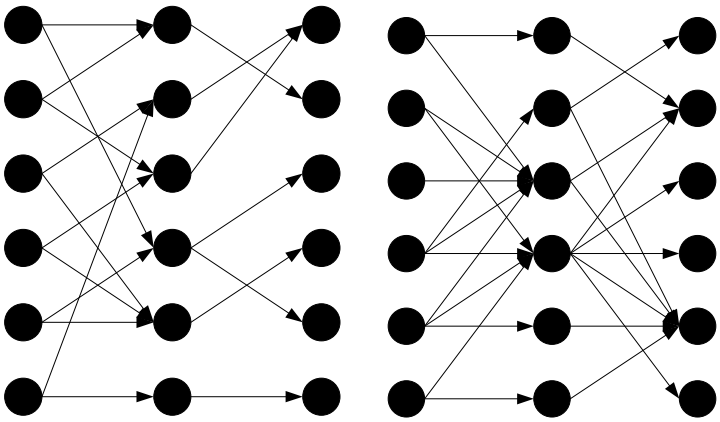


Figure 1-2 - Example of different types of networks. (a) random network; (b) preferentially connected network

It is easy to observe from the examples shown in Figure 1-2 that in random networks, the number of inbound and outbound edges from each vertex does not vary much, while in the preferentially connected network these values do present a very large variance.

As previously mentioned, complex networks have been extensively researched in recent years. Research in these types of networks has been focused mainly on two aspects: analysis of the distribution of real-world networks and modeling to explain the phenomena behind those observed distributions. In the analysis of real-world networks, it has been observed that many of these networks are preferentially connected, presenting an apparent power-law, or Zeta distribution that follows the following formula [31]: (a)

$$P(k) = \frac{k^{-\gamma}}{\zeta(\gamma)}, \tag{1.1}$$

where $P(k)$ is the probability of the given vertex to have degree k (k number of edges), γ is the Zeta distribution exponent and $\zeta(\gamma)$ is the Riemann zeta function.

This distribution was observed in the World Wide Web [32], metabolic networks [33], Internet router connections [34], journal paper reference networks [35], sexual contact networks [36], and other real-world

networks. However, most of the times, the methods used to identify the fitness to the actual Zeta distribution does not follow rigid statistical evidence. In [37] a more rigorous method for determining a goodness-of-fit measurement for this highly skewed distribution is presented. Having good knowledge of the underlying distribution of the network parameters is vital for analyzing possible heuristics that can be used for determining the feature values, as will be discussed in Chapter 5.

Advancement also has been made in generating models of systems that present preferentially connected behavior. One of the most important is the Barabási-Albert model (BA) [38] , a special case of the Yule model [39]. The BA model defines a probability of connection between two elements on a growing directed graph being directly proportional to the number of connections:

$$\Pi\left(k_i\right) = \frac{k_i}{\sum_j k_j}, \tag{1.2}$$

where $\Pi(k_i)$ is the probability of connection to vertex i, and k_i is the in-degree (the number of incoming edges) of vertex i. This simple rule has been shown to produce a preferentially connected in-degree (the number of incoming edges in a vertex) distribution that approximates to a Zeta distribution with exponent $\gamma = 3.0$. A number of studies have been conducted on the generalization of the model and there are important recent review papers on the subject [40-42], worthy of further reading.

1.4. Proposed General Pattern Recognition System

In Figure 1-3 the high-level overall pattern recognition in ontology-structured databases system is envisioned. This system is presented in this section in order to clarify the full requirements for a complete pattern recognition system. Detailed discussion of this system is outside the scope of this study. The goal of this research was to analyze and implement the feature extraction process of the system. However, in order to provide a complete overview, some of the processes involved in Figure 1-3 have also been implemented,

but have not been analyzed in their entirety. Below a brief summary of each of the proposed elements of the overall system will be presented.

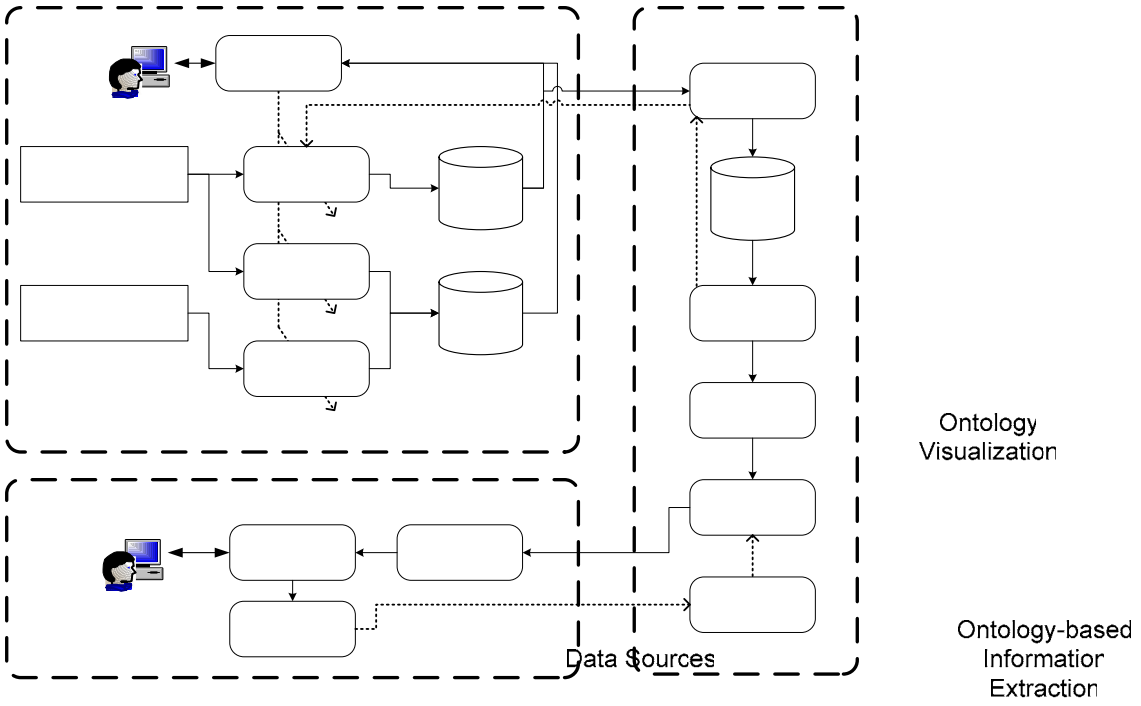


Figure 1-3 - Ontology-based pattern recognition system

The system is divided into three different subsystems that interact with each other, providing feedback for improvement. The first is of Domain Ontology Acquisition, i.e. the extraction and accommodation of the structure and the data of interest. This system makes use of reusability ideas behind modern ontologies [3, 43, 44] and the wealth of automated and semi-automated methods for ontology extraction [5, 45, 46]. It would also employ automated and semi-automated methods for data extraction from different unstructured, semi-structured, or structured databases [47-50]. The specific processes in this subsystem are presented next.

Ontology-based information extraction: This method extracts the data from external databases and structures this data based on an ontology of interest. It employs different methods depending on the data source used. For example, for free-text unstructured or semi-structured data, natural language processing methods may be necessary to extract the required information. In semi-structured databases, such as the World-Wide Web, a very large amount of research has been devoted into using *wrappers* to facilitate the

extraction of the information [51-54]. These wrappers can easily interface with ontologies [55]. However, very little research addresses the use of the ontology for facilitating wrapper creation and maintenance. As of structured databases, sometimes it is necessary to perform structure transformation and mapping to adapt the initial structure to the needs. More information about this process will be given in Chapter 4.

Ontology learning: This process uses the data to obtain the structure of the database, or to suggest structure to a user that will be used to facilitate the construction of the ontology for the database. A number of different methods have been proposed in the literature, such as the use of clustering [56], inductive logic programming [57], association rules [58], frequency-based [45], pattern matching [59], and simple classification methods [60].

Ontology mapping: In this process, the ontologies of the databases being extracted are mapped to elements in the ontology being used for analysis. This mapping can be as simple as changing the name conventions of the elements (human being → person, for example), or more complicated such as the creation or removal of whole element types. More details about ontology mapping will be discussed in Chapter 4.

Ontology visualization: As most of the ontology creation methods are highly human-intensive while natural language processing is still a challenge and there is no standardized structure for all databases ever created, it is necessary to present an expert user with an interface to create and correct the ontology, as well as making possible bindings between the ontology and the data. This is accomplished by the ontology visualization process. A number of different ontology visualization programs have been developed throughout the years, such as Protégé[1]. It would be necessary to make modifications in these off-the-shelf programs, though, in order to be in tune with dealing with the other proposed ontology acquisition processes.

[1] Available at http://protege.stanford.edu

The result of this subsystem is the construction of two databases, one containing the ontologies, or the definition of the graph-structured databases; and the other contain the graph-structured data. To facilitate the reuse of ontologies, these databases follow the OWL specifications, mentioned above.

The second subsystem, on the right of Figure 1-3, performs the pattern recognition *per se*. More details about the pattern recognition process will be given in the following chapters. Some brief descriptions about the processes depicted will be given below.

Ontology pre-processing: This process has the goal of applying modifications to the database so that the processing of the features becomes less time-consuming. More details about this process will be given in Chapter 4. Some methods might require the extraction of more information from the database, thus triggering the information extraction process.

Feature extraction: The feature extraction process obtains the features of interest from the database in order to perform the pattern recognition later. This process will be explained in details in Chapter 3.

Feature selection: After the extraction, in order to cope with the curse of dimensionality, in some cases it is necessary to apply transformations to the features in order to decrease the dimensionality before the pattern recognition. A good amount of research has been devoted to this process, and, although it is not in the scope of this research, a brief discussion about the problem and possible solutions will be given in Section 3.5.

Pattern recognition: The pattern recognition process is the goal of the system. It has the objective of determining the location of the patterns of interest in the database. It returns the group of elements in the database that could be of interest, rating them in degree of possible interestingness. Although pattern recognition is no not within scope of this study, some discussion is necessary to support the results of the feature extraction, provided in Chapter 3.

Pattern learning: Sometimes the pattern of interest is not explicitly known by the user of the system. In this case it is necessary to accept modifications on the results of the pattern recognition based on feedback given from the user after the patterns are displayed. Learning is based on positive and negative feedback from the examples shown and this has to be incorporated into the pattern recognition step. Further discussion about this process will be provided in Chapter 3.

The output of this subsystem is a rated or ranked list of instances in the database that display features that suggest the existence of the pattern of interest. These pattern-bearing candidates are displayed to the user who analyzes their actual similarity to the high-level pattern of interest and feeds back information to improve the process. This visualization is done by a separate subsystem, the Visualization System.

Visualizing large graph-structured databases is a very difficult issue on all graph-structured systems. Methods for presenting data that are structured as graphs dates back to the initial analysis of graph theory and the presentation of planar graphs [61]. A large amount of work has been done by researchers in the last few years to provide methods for visualizing non-planar graphs, or large graphs in which the cost of determining planarity is too high. Methods based on force-directed placement [62], self-organizing maps [63], clustering [64, 65], Pathfinder networks [66], minimum spanning trees [67], among many others, have been used in a number of different types of databases showing their strengths and weaknesses.

As mentioned before, it is not within the scope of this research to develop and present a new visualization method to the user with a way to interact with the results presented. However, because of the need for user feedback to determine if the patterns observed are actually close to what was sought for, it is necessary to at least analyze the requirements of such visualization system. These requirements and the details of the implementation of a very simple but effective visualization system are presented in Section 3.4. Below a short description of each of the processes presented in Figure 1-3 will be presented.

Pattern visualization: This process isolates the elements that suggest the presence of the pattern of interest. It generates simple graph structures that assist in displaying to the user the rationale why the features indicate the pattern.

Data browsing: This process generates a more powerful method for the user to browse the graph-structured data related to any parameters added by the user. It is very important to allow free navigation to the user in order to enable the improvement of the pattern definitions. Only letting the user observe the best matches for the current pattern definition is very likely to provide results that are biased towards certain well-known features, usually less interesting.

Pattern example extraction: It is aimed at extracting the information necessary for the pattern learning algorithm to update its pattern definition and recalculate the ratings of the pattern-bearing candidates.

Having presented the overall pattern recognition system, the next section explains the structure of the rest of the document.

1.5. Structure of the Document

This report is divided in 8 chapters. Next chapter, Chapter 2, introduces some background on important concepts on processing and manipulating graph-structured databases that will be used throughout the other chapters. Chapter 3 introduces the feature extraction concepts, as well as simple pattern recognition procedures and the visualization methods. Chapter 4 presents the database transformation method that adapts the database structure, usually obtained from external sources, to the processing requirements. Moreover it provides methods for making transformations in the database decreasing the processing cost. In Chapter 5 improvements on the pattern recognition processing requirements are provided based on network sampling concepts. Chapter 6 presents discussions about the implementation made for all the methods presented. Chapter 7 shows some application examples for the proposed algorithms. Finally, Chapter 8 presents some conclusions and proposes further modifications and research directions.

CHAPTER 2

Manipulation of Graph-Structured Databases

In order to perform structural feature extraction in graph-structured databases, a number of definitions have to be introduced. These definitions aim towards a formal description of the steps taken in order to:

- perform ontology transformation,
- obtain meanings of structural features, and
- present the results in a graphical mode.

It has to be emphasized that the definitions presented in this document are not complete as of manipulating graph-structured databases. They do not deal with creation and removal of vertices and edges directly, nor do they present explicit mechanisms for querying graph-structured databases. Graph-structured databases when following an ontology are referred throughout the rest of this document interchangeably as an *ontology space*.

Definition 1: An *ontology space* is an *n*-tuple

$$O = \left\{ V(O), E(O), T_{V,0}(O), \ldots, T_{V,NV}(O), T_{E,0}(O), \ldots, T_{E,NE}(O) \right\} \tag{2.1}$$

where $V(O)$ is a set with all vertices, $E(O)$, is a set with all edges, $T_{V,i}(O)$ is a set that contains all vertices of the i^{th} type, and $T_{E,j}(O)$ is a set that contains all edges of the j^{th} type. NV and NE are the total number of types of vertices and edges, respectively. The indices i and j are actually labels for the edges and vertices. For the remaining of this report, most of the examples these will be labels following the standard that all vertex labels start with a capital letter, while the edge labels are not capitalized.

The proposed definition is similar to the definition of a graph space [61], only augmenting the classical definition by adding vertex and edge types. Another known definition from the literature is the one given by Gyssens *et al.* [14] in which two types of vertices and edges are defined (object and printable object vertices, and functional and multivalued vertices). Moreover, a function is used to define the connection between the vertices and edges. Because of the desire to use a more graph-theoretical oriented representation, the given definition was preferred.

Definition 2: A *projection* in the ontology space O is represented as $\pi(O, i, j, k)$ and is a directed graph formed by the vertices in the set type $T_{V,i}(O)$ and $T_{V,j}(O)$ and the edges in the set $T_{E,k}(O)$. All edges in the set $T_{E,k}(O)$ that do not connect an element in the set $T_{V,i}(O)$ to an element in the set $T_{V,j}(O)$ are removed. If $T_{V,i}(O) \cap T_{V,j}(O) = \varnothing$, i.e. there are no common element between the types, this graph is a bipartite graph with the first partition formed by the vertices in $T_{V,i}(O)$ and the second from vertices in $T_{V,j}(O)$.

It is important to note the difference of this definition of projection to what is usually done in bipartite network analysis [40]. In the latter case, a projection is a transformation from a bipartite network into a weighted network formed by the vertices of one of the partitions with edges present based on the co-connection to a common vertex in the other partition. In other words, if there exists an edge that connects vertex v_A to w_X and v_B also to w_X then after the projection there will be an edge connecting v_A to v_B. A similar transformation will be called *graph collapsing* in this document and will be formally defined in Definition 9.

Definition 3: A *walk* (of length l) in the ontology is a sequence of vertices and edges that connect two distinct vertices v_0 and v_l. These vertices are called the start and end-vertices, respectively. The walk, w, can be represented by the sequence

$$w = v_0 e_0 v_1 e_1 \ldots v_{l-1} e_{l-1} v_l \tag{2.2}$$

such that $e_i = \{v_i, v_{i+1}\}$ for all $i < l$.

Definition 4: A *weight policy*, Ω, is a function $O \times w \to \mathbb{R}$ that, based on the ontology O and the walk w returns a real value, the weight of the walk semantics.

Definition 5: A *walk semantics* is the set of projections

$$\omega = \{\pi(O, i_0, i_1, k_0), \pi(O, i_1, i_2, k_1), \ldots, \pi(O, i_{l-1}, i_l, k_{l-1})\} \tag{2.3}$$

that can also be simplified as the sequence of alternated labels

$$\omega = \{i_0, k_0, i_1, k_1, \ldots, i_{l-1}, k_{l-1}, i_l\} \tag{2.4}$$

where the i_j are labels to vertices, and the k_j are labels for edges. This second representation will be the one used throughout this document.

Definition 6: A walk is said to *agree* with a walk semantics if and only if they have the same length and each of the vertices and edges of the walk in sequence are in the vertex and edge types of the walk semantics, also in the same sequence.

Definition 7: The *equisemantic walks* relative to the walk semantics ω is the set of all walks that agree with ω. They are represented as $EW(\omega)$. Most of the times it is important to obtain all equisemantic walks between two specific vertices v_i and v_j relative to the walk semantics ω. This is the subset from $EW(\omega)$ such that the start and end-vertices of the walks are v_i and v_j, respectively. They are represented as $EW(v_i, v_j, \omega)$.

Definition 8: The *weighted equisemantic walks* relative to the walk semantics ω and the weight policy Ω, $WEW(\omega, \Omega)$ is a set of tuples, where each tuple is formed by a walk that agree with ω and the weight of this walk, following the weight policy Ω. As for the equisemantic walk definition, most of the times it is

interesting to obtain its subset with defined start and end vertices v_i and v_j, represented by $WEW(v_i, v_j, \omega, \Omega)$.

Definition 9: A *collapsed graph* given the walk semantics $\omega = \{i_0, k_0, i_1, k_1, \ldots, i_{l-1}, k_{l-1}, i_l\}$ is a directed graph formed by all vertices v_i and v_j such that $v_i \in T_{V,i_0}$ and $v_j \in T_{V,i_l}$, and the edges $e_k = \{v_i, v_j\}$ such that there exists a walk w that agrees with ω and starts at v_i and ends at v_j. It is also possible to define a *weighted collapsed graph* in which the edges are weighted. These weights are defined by a certain function based on the weights of the walks that connect the two vertices in question.

In the special case where $T_{V,i_0} \cap T_{V,i_l} = \varnothing$, the result of this transformation of *graph collapsing* is a bipartite graph. In the case when $T_{V,i_0} = T_{V,i_l}$ and the walk semantics is *reversible*, i.e. if there exists a walk w_i that connects v_i to v_j, there exists another walk w_j that connects v_j to v_i, this procedure produces a non-directed graph, similar to the projection usually employed in bipartite graph analysis, as explained above.

Definition 10: The *ontological connections* between two vertex types *i* and *j*, represented as *OC*(*i, j*) are all the walk semantics that connect the vertex types *i* and *j* that do not conflict with any constraints in the ontology definition.

Note, thought, that the definition of *ontology space* does not incorporate the ontology constraints. These are obtained from the OWL definition. From this definition, a number of constraints exist, but only two are of interest: domain and range of edge types (the types that can and cannot be connected to the in and out positions of the edge).

Another important observation that has to be made is that usually the number of *ontological connections is* infinite. However, it can be argued that long walk semantics have very little meaning in most databases. Therefore, it is possible to limit the maximum length of the walk semantics that form the *ontological*

connections and, thus making this set finite. A length-limited ontological connections operation is defined as $OC(i, j, L)$, where L is the maximum length of the walk semantics in the set.

Definition 11: Two vertex types from two different ontology spaces are considered *equivalent* if they represent the same physical or abstract element.

Definition 12: An *equivalent walk* between two ontology spaces O_1 and O_2 is a pair of walk semantics ω_1 and ω_2 defined in O_1 and O_2 respectively so that the initial vertex types are equivalent, as well as the final vertex types. Moreover, the abstract or physical meaning behind the connection between these elements defined by each of the walks has to be the same.

The above definitions are focused on finding relations between pairs of elements. Although most features of interest are related to pairs of elements, mainly because of the lower memory requirements to deal with pairs of elements instead of three or more elements, sometimes it is interesting to obtain *n*-element features. One example where features of more than a pair of elements are interesting would be one trying to relate geographical position to an element feature. A geographic position is based on two or three values (latitude, longitude and, sometimes, altitude). In order to deal with this kind of problems, it is necessary to extend some of the definitions above.

Definition 13: An *arbitrary equisemantic walk* is a set defined by a group of equisemantic walks

$$AEW\left(EW\left(v_{i0}, v_{j0}, \omega_0\right), EW\left(v_{i1}, v_{j1}, \omega_1\right), \ldots, EW\left(v_{iN}, v_{jN}, \omega_N\right)\right), \tag{2.5}$$

such that the graph formed by the connected vertices v_{ik} is fully connected. This represents a set of walks such that all the vertices v_{ik} and v_{jk} are present and at least one walk between the elements v_{ik} and v_{jk} follows the walk semantics ω_k.

Sometimes it is important to define the walk semantics in which the arbitrary equisemantic walks are defined. Although this is not necessary for the computations involved, it facilitates some discussions.

Definition 14: An *arbitrary walk semantics AP* is a set of walk semantics

$$AP = \{\omega_1, \omega_2, \ldots, \omega_N\}, \tag{2.6}$$

so that the walk semantics ω_i are fully connected, i.e., there are no two vertex types $T_{V,i}$ and $T_{V,j}$ in any of the walk semantics $\omega_k \in AP$ such that it is not possible to find a walk semantics ω' that starts at $T_{V,i}$ and ends at $T_{V,j}$ such that all vertex and edge types in the walk semantics are part of at least one of the $\omega_k \in AP$.

Definition 15: A *weighted arbitrary equisemantic walk* is a set of tuples defined by a group of weighted equisemantic walks

$$AWEW\left(WEW\left(v_{i0}, v_{j0}, \omega_0, \Omega_0\right), WEW\left(v_{i1}, v_{j1}, \omega_1, \Omega_1\right), \ldots, WEW\left(v_{iN}, v_{jN}, \omega_N, \Omega_N\right)\right), \tag{2.7}$$

such that the graph formed by the connected vertices v_{ik} is fully connected. This represents a set of walks such that all the vertices v_{ik} and v_{jk} are present and at least one walk between the elements v_{ik} and v_{ik} follows the walk semantics ω_k. The weight of each of the tuples is defined as the sum of the weights of each of the weighted equisemantic walks that form it.

Definition 16: The *arbitrary ontological connections* set is represented by $AOC\left(v_{(0)}, v_{(1)}, \ldots, v_{(N)}\right)$, or by using the length-limited definition $AOC = \left(v_{(0)}, v_{(1)}, \ldots, v_{(N)}, L_0, L_1, \ldots, L_{N-1}\right)$. The definition follows directly the definition of the standard ontological connections in Definition 10. It is important to note that the walk semantics in this set do not have to adhere to a specific order in which the vertices are traversed.

A special case of the arbitrary equisemantic walks is when all v_{ik} are the same. In other words, all walks start from the same vertex. These *arbitrary equisemantic walks* are referred to as *equisemantic star*. The *arbitrary walk semantics* related to it is referred to as a *semantic star*.

These definitions are sufficient to present the formal proposal of the algorithms. The next chapters will explain these algorithms in detail.

CHAPTER 3

Structural Feature Extraction

3.1. Introduction

This section presents the main proposed feature extraction algorithm. The concept of pattern recognition in graph-structured databases and some basic algorithms will also be analyzed further in this chapter in order to enable the analysis of the feature extraction process.

Most existing approaches for pattern recognition on databases focus on the presence or absence of the connection between two or more elements of interest. For example, when analyzing the contents of a web page, current methods would analyze the use of specific terms in the page or on pages linked by this page [53, 68]. Although this method serves its purpose as being able to detect the pattern of interest (the web page content), it is not applicable to all possible patterns of interest in the graph database.

The concept behind the proposed algorithm is that a path in an ontology space is created based on a certain meaning of connection between the elements. This meaning can then be related to a feature. The presence of multiple paths is an indication that this meaning has a higher relation to the group of elements being connected by these paths than elements that have less paths of the same type connecting them. For example, when trying to analyze a database of journal articles for papers that are more related to a particular subject of interest, higher number of references this paper has to papers that are known to be in the subject of interest indicates that there is a higher probability that the paper is in the subject of interest. Another example would be when analyzing networks of movie actors and movies they have participated, actors that participated more often with particular other actors are more likely to be personal friends.

Therefore, the proposed method not only contains information about the elements that are interconnected, but the structure of this interconnection. More details about this will be shown when the formal algorithm is presented in Section 3.2. Another important analysis made in this chapter is about the pattern recognition process itself. Depending on the type of problem, the pattern recognition process may have different meaning representing different procedures to perform it. The proposed method, unlike most systems ever presented, deals with well-known semantics, i.e., the pattern that a user is trying to identify in the database contains features that is known or partially known by the user.

This is a reasonable assumption in most cases, because the goal of the problem usually already gives hints on what kinds of features may be related, **as long as the features can be related to a meaningful concept**. Features are directly related to the structure of the database. For example, Figure 3-1 shows a simplified version of the database structure for bibliographic citations database given by ISI. The complete ontology, following the standard defined in Chapter 1 is shown in Appendix I-1. This structure is very simple but largely based on a relational model, in which each paper is the center of the database containing a number of fields, such as *Title*, *Source* (the journal name), *Volume number*, *Issue number*, *Beginning Page number*, and *Date*. Actually ISI provides a number of other elements that are either inexistent in a large number of the papers or do not provide any information that is deemed useful for the patterns being analyzed. The use of elements that are inexistent in a large number of papers may generate undesired biases on the results. For example, the *Institution names* of the non-first-author authors are something that only appear in some journals. Any analyses based on this feature, such as to try to identify journals that promote collaboration between authors of different institutions, would only identify these journals, inferring that the journals with incomplete information tend to choose only single-institution papers.

In Figure 3-1, if the pattern of interest may be related to journal issue features it would be necessary to generate a feature that relates a *Paper* to three different elements: *Source*, *Volume* and *Issue*. The processing requirement for obtaining this feature is, as previously mentioned and will be made clear below, both very time- and memory-consuming. However, if the structure presented in Figure 3-2 is used, the feature is simply the relation between the *Paper* and the *SourceIssue* element.

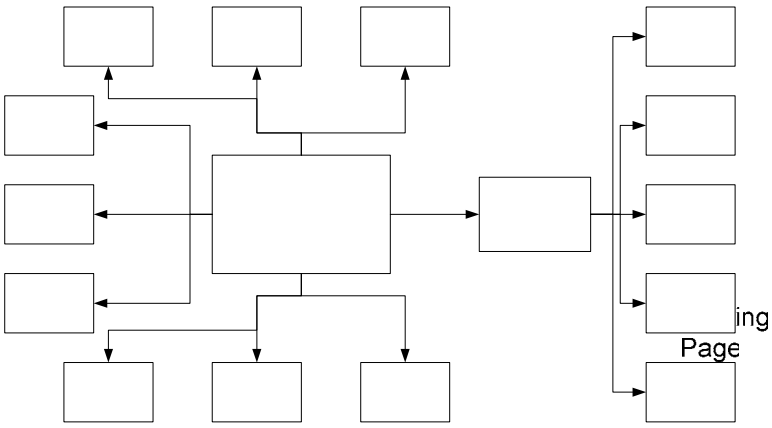


Figure 3-1 - Simplified version of ISI database structure for journal articles

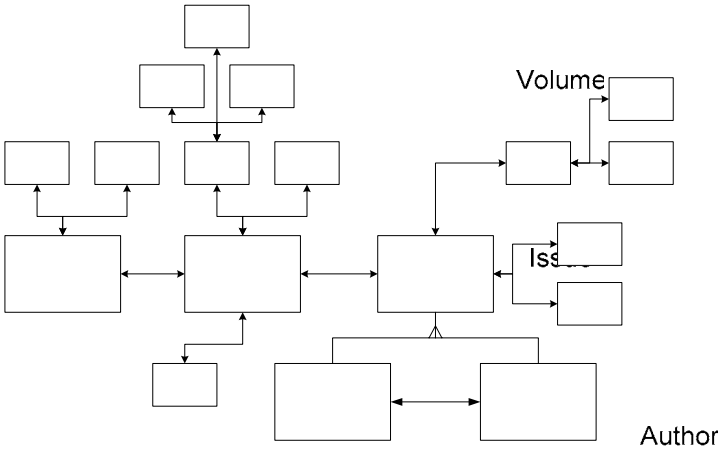


Figure 3-2 - Modified structure for papers database

This transformation for obtaining a structure that is more related to the needs of the processing will be formalized in the next chapter.

This chapter is structured into eight sections. Section 3.2 presents the proposed path-based feature extraction for ontology-structured datasets. Section 3.3 explains the pattern recognition scheme based on the extracted features. A discussion about issues related to user interface will be presented in Section 3.4. Section 3.5 analyzes applicable dimensionality reduction schemes for these datasets, while Section 3.6

presents some extensions that can be easily made to the proposed algorithm to deal with syntactic features. Section 3.7 shows some preliminary experimental results of the main concepts presented. The summary of the observations and some conclusions are presented in Section 3.8.

3.2. Proposed Concept of Structural Feature Extraction

The proposed structural feature extraction algorithm is based on the concept that a feature can be related to a *walk semantics* (see Definition 5, in Chapter 2), i.e. the way two (or more if used an *arbitrary walk semantics*, in Definition 14, Chapter 2) elements are connected has a definite meaning (therefore the term "*walk semantics*"). For example, consider these two possible connections between an *Author* and a *Source* from the structure in Figure 3-2:

$$\omega_{publishes-in} = \begin{Bmatrix} Author, wrote, Publication, inSourceIssue, \\ SourceIssue, ofSource, Source \end{Bmatrix} \tag{3.1}$$

$$\omega_{referred-by-publishes-in} = \begin{Bmatrix} Author, wrote, Publication, citedBy, \\ Publication, inSourceIssue, SourceIssue, \\ ofSource, Source \end{Bmatrix}. \tag{3.2}$$

Although they both connect two elements of the same type, they clearly have different meanings. While one refers to the journal preference of an author, the other presents the journal visibility of this author. The second *walk semantics* (3.2) may be interesting to analyze the diffusion patterns of the knowledge among different journals, while the first (3.1) is related to the specific field of interest represented by the journal.

Below a formal definition of the structural features will be given. To simplify the notation, the formal definition will be given only for pairs of elements. The extension to groups of elements using the concept of arbitrary equisemantic walks (Definition 13, Chapter 2) follows directly from this definition.

Formally, the feature value between vertices v_1 and v_2 following the walk semantics ω can be defined as

$$f(v_1, v_2, \omega) = \left| WEW(v_1, v_2, \omega, \Omega) \right|, \tag{3.3}$$

where the $|\bullet|$ function is defined as the sum of all the weights of the tuples in the weighted equisemantic walks set. The weight function Ω is defined following the interpretation of the properties and some possible database features, such as preferential attachment. Some examples of weight functions will be briefly discussed next.

Element properties can have two different interpretations: static and growing properties. Static properties are relative to the creation of the element. In other words, they are properties that exist since the creation of the element in the real world. For example, in journal articles, the *cites* property is a static property, because an article is created already with the citations and these do not change throughout the history of the element. On the other hand, there are growing properties related to the influence of other elements on this element. In the same example of journal articles, the c*itedBy* property is a growing property, because it changes every time a new paper is written that cites this paper.

Following this concept the following weight policy is used: set the weight to 1 to all outgoing growing properties and $1/k$, where k is the number of outgoing edges, to the vertex to all static properties. However, this requires the user to be able to identify whether a connection is static or growing. This may be a complicated issue in some datasets. In this case, growing connections are considered as default. The consequences of this choice will be exploited later in the simulation examples in Section 3.7.

Sometimes it may be interesting to define other types of weight policies. One possible policy is to decrease the weight based on the number of incoming edges of the target vertex. This may decrease the bias caused by the preferential attachment, however it tends to bias *scatter* elements [69, 70], which may be an undesired effect, as *scatter* elements possess very little information tending to be highly noisy. Some experimentation with this policy will be shown in the examples in Section 3.7.

It is important to note, though, that the features extracted through this method are not normalized. If more than one feature is used in conjunction, it is necessary to provide a normalization method for them to be comparable. Normalization will be treated in the next section when pattern recognition concept will be introduced.

Similar ideas have been applied by Alani *et al*. [71] for identifying communities of practice, in a system called Ontocopi (Ontology-Based Community of Practice Identifier). However, in Ontocopi there is no distinction made between the different type of elements and relations. The main concern is about distance and connectivity. Also there is no application of the static and growing connection differences. Another important difference is that there is no implementation of this concept to pattern recognition; a user enters the source element and the target size element and the system gives a rating to how close these elements are and sorts the results by this rating.

The feature extraction method presented in this section is apparently very simple and natural. However, as it is going to be presented and discussed later, there are a number of issues that emerge when actually implementing this feature extraction process. The choice of the weight function is also a very important element for the success of obtaining the correct feature values. Only a thorough study, as the one provided in this report, will provide enough information to enable future exploration of these structural features for the important process of pattern recognition.

3.3. Pattern Recognition

As mentioned earlier, pattern recognition in graph-structured databases is a concept that has still to be further discussed. This section does not provide an in-depth discussion about the subject, but intends to present methods that are useful enough to provide insights about the feature extraction process, described in the previous section.

Before delving into the specifics of the proposed pattern recognition algorithms, it is important to identify what is the concept of a *pattern* in graph-structured databases. Fukunaga defines pattern recognition as "a problem of estimating density functions in a high-dimensional space and dividing the space into regions of categories or classes" [23]. Therefore, a pattern from this point of view is a category or class that the elements of interest may be part of. More specifically, a pattern is a high-level concept that is likely to produce certain effects on the elements of interest such that the properties and surrounding elements are affected by it. But a question remains on how the user can present, or teach to an algorithm of what constitutes the pattern of interest.

Classically, there are three different methods of presenting patterns: by explicit definition of the features of interest, by implicit definition using examples of elements that present the pattern of interest, or by unlabeled clustering methods (methods that do not require the user to present any definition of the patterns besides usually the number of patterns of interest – the system then automatically groups elements that are most likely to be following the same pattern). This last method is not going to be investigated in this study. This document will focus on the first two methods of pattern recognition, explicit-based and example-based. Later these two methods are combined. In the following subsections, each of these algorithms will be presented.

3.3.1. Explicitly defined walk semantics

On the explicit definition, the user presents to the system a walk semantics, or a set of walk semantics that would represent the pattern that is being searched for. For a single walk semantics, the higher the $f(v_1, v_2, \omega)$ value is, the higher is the similarity of the pair of entities to the pattern being searched for. However, single walk semantics are only useful in very limited applications. Most interesting patterns involve more than two known database entities, or more than two walk semantics between elements. However, it is important to note that multiple-entity walk semantics, or arbitrary walk semantics can be treated as the combination of the results of each of its composing walk semantics. Thus, in this document very little analysis will be made specifically on these kinds of patterns.

On the other hand, it is important to deal with patterns that are formed or that can be inferred from a conjunction of walk semantics. Each of the walk semantics defines a feature towards the extraction of this pattern, forming a *feature vector* for each pair of vertices, defined as

$$\vec{f}\left(v_i, v_j\right) = \left[f\left(v_i, v_j, \omega_1\right) \quad f\left(v_i, v_j, \omega_2\right) \quad \cdots \quad f\left(v_i, v_j, \omega_N\right) \right]^T . \tag{3.4}$$

However, it is not possible to compare two feature vectors in order to establish which follows the pattern the most. It is only possible to compare scalar values. Therefore, it is necessary to perform a transformation in the vectors that would make them scalars. This transformation is performed in three steps: normalization, dimensionality reduction and importance weighting.

In the **normalization step**, the objective is to ensure that the dynamic range of each of the features is the same. Different dynamic ranges are observed when features are related to walk semantics of different length and different nature (i.e., more interconnected walk semantics show higher number of equisemantic walks than very sparsely connected ones, but this does not imply that the higher connected walks are more meaningful). A linear normalization is applied for all features:

$$\tilde{f}_i(j) = \frac{f_i(j)}{\sum_k f_i(k)}, \tag{3.5}$$

where $f_i(j)$ is the feature i for group j of vertices (note that the normalization is defined for features on any number of vertices).

The **dimensionality reduction step** is very important when dealing with high-dimensional feature vectors. However, its analysis falls out of the scope of this study. Section 3.5 presents some of well-known

dimensionality reduction schemes. However, none of the applications presented employ a very large number of dimensions, thus requiring no dimensionality reduction.

The **importance weighting step** enables the user to define the patterns and add information to which patterns are more informative, or more important, than others by choosing weight values for each walk semantics.

A threshold value can be applied to the resulting feature value to define which group of vertices shows the patterns and which do not. Yet, for some applications, it is preferable to enable the user to browse through all results because very low feature values can also be a sign of increasing importance, especially for *scatter* vertices, i.e. vertices that initially were not considered the center of the data collection. This, and other visualization issues, will be discussed in Section 3.4.

3.3.2. Example-based pattern definitions

In the example-based case, the user inputs to the system examples of the pattern being searched by presenting groups of p vertices that represent the pattern, named the *positive examples*, and groups of p vertices that do not represent the pattern, the *negative examples*. In this method there is no prior knowledge of which walk semantics are important for inferring the pattern. With this information it is possible to generate all *feasible* walk semantics between the p elements within each group, by first generating at least one *type template* $\tau_1,\ldots,\tau_p$ so that for all the groups of examples, there exists at least one enumeration v_1, $v_2,\ldots, v_p$ for each group so that:

$$v_i \in \tau_i, \forall i = 1\ldots p\ . \tag{3.6}$$

If it is possible to generate more than one type template, all can be considered at the same time as compound features. This is very common in a case where there are types that are subtypes of other types, i.e. givent two types T_x and T_y, T_x is a subtype of T_y if $\forall v_i \in T_x$, $v_i \in T_y$. In ontologies, this shows that type

T_x is a specialization of type T_y. If there is more than one possible enumeration for a certain example, each possible enumeration has to be considered as a different example to which the features of the possible group will be compared and rated. A very simple example of this phenomenon of multiple enumerations can be seen in the publications dataset for author collaboration. When a set of pairs of authors are given as examples, for a given pair (a_1, a_2) the unique mapping $M: V^2 \rightarrow$ (*author, author*) can be done by either the enumeration (a_1, a_2) or (a_2, a_1).

Having defined the type templates and the possible enumerations for each template, one can find all possible ontological connections between the types of lengths less than or equal to L_i, $AOC(\tau_1, \tau_2, \ldots, \tau_p, L_1, L_2, \ldots, L_{p-1})$, in each of the templates. Where the L_i parameters are defined by the user based on the way the ontology is made, as mentioned previously.

Now, for each of the walk semantics, it is possible to use the algorithm explain in the previous subsection to calculate a feature for all groups, including the positive and negative example groups. This will form again a feature vector for all groups. The normalized, and possibly dimension-reduced vectors of all groups can be compared to the vectors from the example groups by means of a distance calculation. Given that all vectors are normalized, the simplest distance method is the Euclidean distance:

$$d(i,j) = \left(\sum_k \left(f_k(i) - f_k(j) \right)^2 \right)^{1/2} . \tag{3.7}$$

Usually the number of examples is much less than the size of the ontological connections sets. Each group now has, as features, a vector of distances to the pattern examples. The simple sum of the distances to the positive examples subtracted by the distances to the negative examples can give an approximate single-variable decision of the groups that exhibit behavior similar to the ones on the examples. Also it is very simple to increment this by introducing a weighted sum of these distances in the case that the user is able to define an importance to each of the examples provided. It is important to note that in this case the elements

that have the lowest distance is the ones that are more likely to be an instance of the pattern of interest, while in the previous case the highest feature would indicate it.

Increasing the number of examples improves the efficiency of this method. However, most of the times it is costly for the user to be able to identify a large number of examples. There are two possible methods for dealing with this shortcoming: the use of the mixed method, explained below, or the use of feedback from the results. A user may look at the first results of the application of the method for a small group of examples and identify in these results other positive and negative examples that would be fed back to the system to define new results. This process will be explored in the examples in Section 3.7.

It is important to note here the large effect that the maximum length parameter when calculating the ontological connections has on the efficiency of the method. In most cases, the number of possible walk semantics increases rapidly with the increase of the maximum length of a walk. Enlarging the walk semantics set increases the amount of equisemantic walks sets that have to be extracted. Keeping a low maximum length is directly related to being able to model the dataset more efficiently.

3.3.3. Mixed pattern definitions

Finally, when there is enough knowledge to generate examples of the pattern and to indicate the important walk semantics that should suggest the patterns, a mixed method can be used. This method is very similar to the example-based method discussed above, but the first feature analysis does not require the generation of the ontological connections sets and from this generate the feature vectors. The feature vectors are then calculated in an analogous way as the one explained in Subsection 3.3.1. With these features it is possible to calculate the distance to the examples, as seen in Subsection 3.3.2. The remaining of the treatment is also identical to the example-based method.

3.4. User Interface

One of the most important steps when implementing a pattern recognition system in complex systems is being able to display the patterns in such a way that the system presents a simplified view and centered on the patterns under analysis. However, when dealing with large datasets, sometimes with hundreds of thousands of vertices, the number of possible patterns to display is also too large to show in an understandable way. The most natural way of observing graph-based datasets is using graphs, as done in [72], for example. However, in most cases, specialists prefer the use of tables to show the result of the processing because of its natural "clean" interface without the problem of crossing edges and cluttered vertices. However, the tabular method cannot show the interaction between the different results. Nor can it show enough information to display the reason for the suggested match.

In this document, a mixed hierarchical tabular and specialized graphical method is proposed. A hierarchical table is used to easily display the strongest matches giving an option of displaying related matches (other strong matches that have common vertices). A graphical method is used for displaying information about the reasoning behind the pattern. This is done by showing the equisemantic walks between the selected group of vertices and emphasizing the most important of them (either by being high in the given examples, or by being the highest feature for this particular group). Figures 3-3 to 3-7 present some snapshots and explanations on the visualization system.

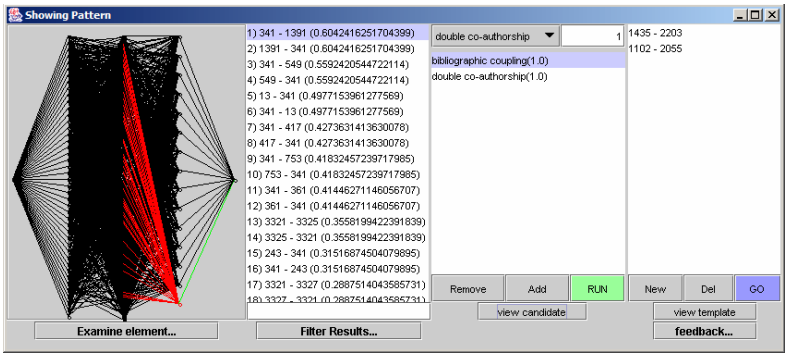


Figure 3-3 - Main pattern recognition panel showing the results of a simple example-based analysis showing the author bibliographic coupling for a given pair, highlighting a specific paper with its inbound and outbound vertices

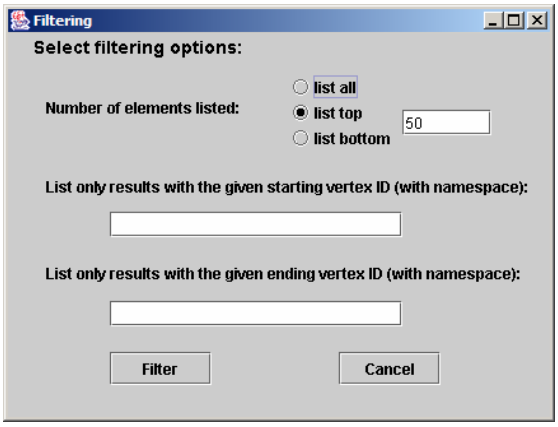


Figure 3-4 - Filtering dialog – users can view only specific elements in the output list

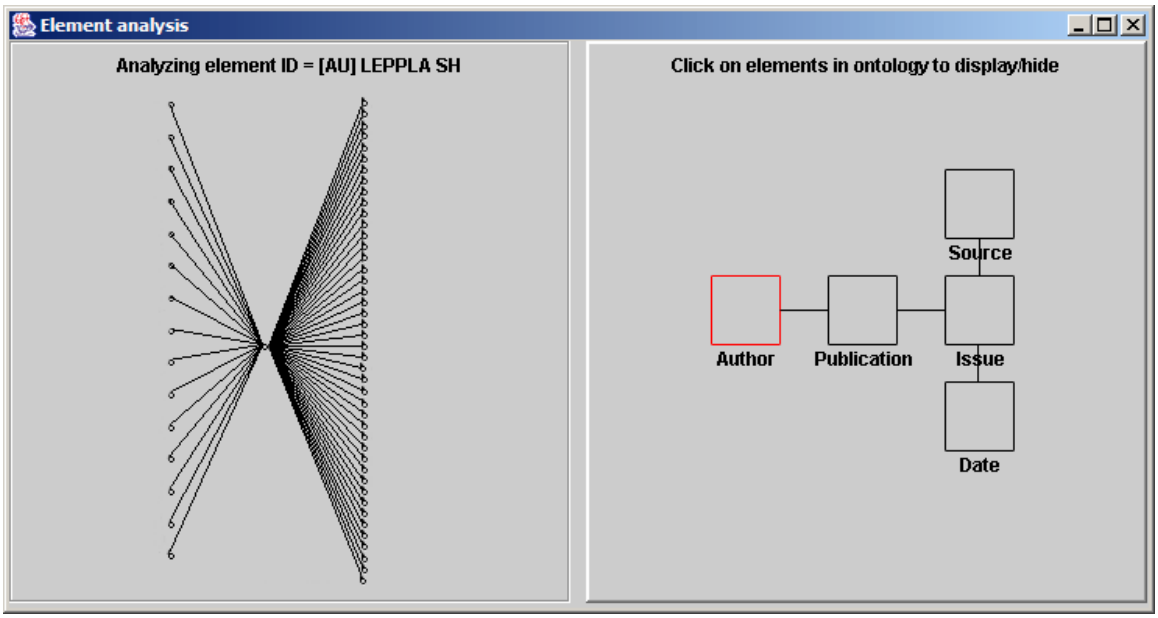


Figure 3-5 - Graphical exploration of a single element – users can select which neighboring or indirect element types to show by clicking on an element type and then selecting from a dialog box the walk semantics from a manually pre-defined set of interesting walk semantics

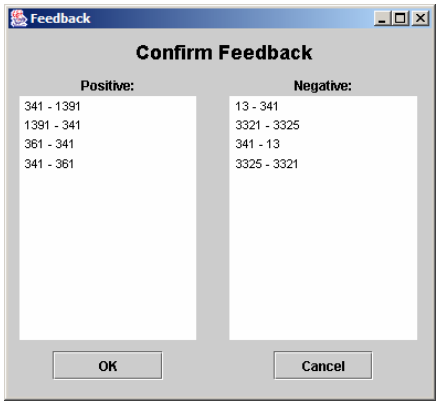


Figure 3-6 - Feedback from user to add positive and negative examples for example-based pattern recognition

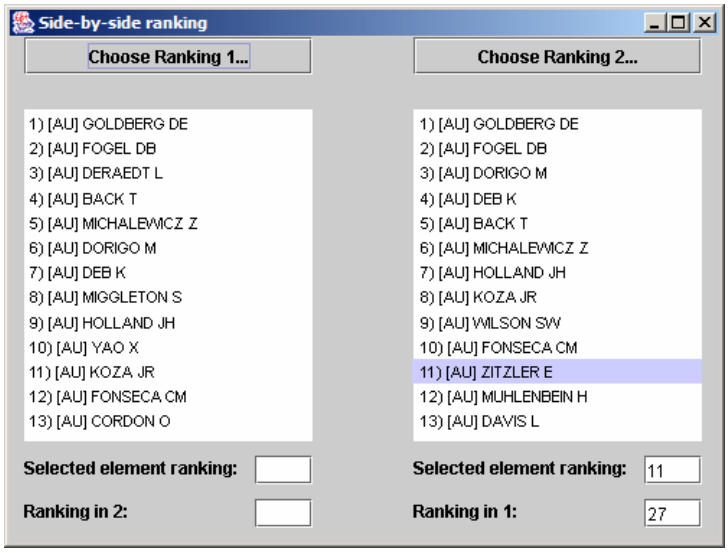


Figure 3-7 - Side-by-side comparison of ranking given two different feature extraction/pattern recognition procedures for the same group of elements

Further information could be added by mapping selected patterns to other visualizations, such as timelines [64] that incorporate temporal information that can be very beneficial in pattern understanding.

When the user has the ability to interpret the results offered by the algorithm it is also interesting for the user to be able to iterate through the results and possibly change some examples, or weighting choices for the given walk semantics (see Figure 1-3, the visualization system). Moreover, it may be interesting to gather more data to increase the amount of details on certain areas of the network. These feedback events must be available for the user in a timely manner in order to enable the analysis of their results by the user. Methods such as side-by-side comparison between past and current results can be used to further assist the

user to choose the correct options for the algorithm. This functionality was also implemented and an example of the application can be seen in Figure 3-6.

One final observation about the visualization system is that it allows the user to browse through the whole database if desired. The ranked results presented only serve as suggestion to the user of which groups *seem* more likely to be related to the patters of interest. However, sometimes because of the *core and scatter* properties of real-world databases, it is necessary to investigate also elements that end up receiving high rank values because of their low connectivity to the acquired data, but present important exceptions among other low connected elements. The automatic identification of these elements is complex because of the underlying skewed connectivity distribution; therefore it is necessary to enable the expert analysis of the database.

3.5. Dimensionality Reduction

As mentioned previously, dimensionality reduction is very important for dealing with features with a large number of dimensions. This section provides a short review on the current dimensionality reduction methods from the literature and analyzes the usability of these methods in this application.

One of the first methods analyzed is of variable selection. The variable selection procedure uses results from the analysis of variance (ANOVA) and analysis of covariance (ANCOVA) for selecting the most statistically significant variables [73]. This method is straight-forward and the features selected are simple to interpret. Moreover, the results of this method provide good insight on the chosen variable independently. However, its power is limited to how much information available on each isolated initial feature. In most cases, the output of this method is still a large number of variables.

In more advanced methods, the objective is to obtain composite variables that isolate the variance of the initial variables, trying to minimize covariance. Among these methods, four are very popular and basic

methods: principal component analysis (PCA), factor analysis (FA), multidimensional scaling (MDS), and Latent Class Analysis (LCA).

PCA transforms the original set of variables into a smaller set of linear combinations that account for most of the variance of the original set [74]. This method generates very good and robust results in the case when the correlation between the elements is linear. However, a significant amount of real-world observed correlation is not linear. Moreover, the new variables created with this method are sometimes difficult to interpret, thus making the patterns obtained hard for users to understand and interact. In order to deal with non-linear correlations, Schölkopf *et al*. [75] proposed a kernel-based PCA, KPCA, that has shown to be effective when non-linear kernels are defined. This method is very appropriate for image feature extraction, but methods for porting this concept to feature extraction in databases are not well-understood.

FA is focused on dealing with the interpretability limitation on the principal components. It defines *factors* that are combinations of variables that are known to be correlated [73]. By construction it is natural to generate features that are easy to interpret. However, this method is as good as the factors chosen. There is no constraint on variance explained, as there is in PCA.

MDS is a method in which the features are transformed into distances, similar to what was done in Section 3.3.2. A large number of different distance calculations were analyzed in the literature [76], but the main goal of this method is to use these distances to present the data points to the user in a two-dimensional representation. The goal sought in this step is to decrease the dimensionality in order to avoid the "curse of dimensionality" and not for visualization. Therefore, this method will not be explored beyond the analysis of distance metrics mentioned above.

Finally, the LCA is an algorithm focused on observing the underlying correlation between binary-type variables, where the features are whether a certain event was observed or not, given the element of interest [77]. This method is widely used in text analysis, where the features extracted are the presence of

keywords. In this field, this algorithm is usually called Latent Semantic Indexing (LSI) [78]. Although this method is widely used, some scalability issues still exist [79].

3.6. Syntactic Features

Syntactic features are defined by a construction grammar. There are many methods of defining grammars for graph-structured databases. This study will use the representation proposed by Gonzales and Thomson [29] on directional graphs. They present four binary operations, +, ×, −, and ∗, and one unary operation ~, shown in Figure 3-8.

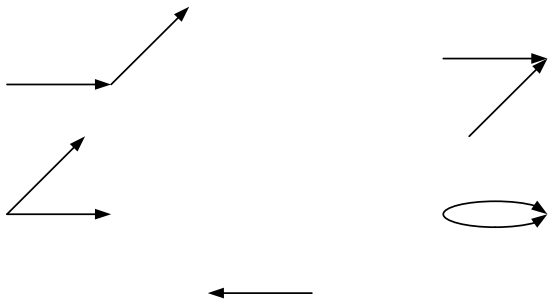


Figure 3-8 - Operations on graph-structured databases

After defining the operations, it is possible to define a grammar as a 4-tuple

$$G = (N, \Sigma, P, S), \tag{3.8}$$

where N is the set of non-terminal elements, Σ is the set of terminal elements, P are the production rules and $S \in N$ is the starting symbol.

The most important part of a grammar is the production rules, i.e., transformation rules that can be applied to transform a non-terminal element into a group of terminal and/or non-terminal elements. A sequence, or structure is in the language generated by G, $L(G)$ if and only if it is possible to generate this structure by

successively applying the rules in P starting at the element S. Take the example below, where the production rules are $P = \{ S \rightarrow a + S, S \rightarrow b \}$, $N = \{ S \}$, and $\Sigma = \{ a, b \}$. The following structure

$$a + a + b + b \in L(G),$$

while

$$a + a + b + b + a \notin L(G).$$

The proposed structural feature extraction algorithm can be modified to deal with these kinds of features, **considering that a grammar generates a language that has the same physical meaning**, i.e. if the meaning of each of the graphs in $L(G)$ is the same (although they can vary on strength to the meaning – the weight). For example, a friendship grammar has the following production rules

$$P_{friend} = \{ S \rightarrow S + friendOf + Person, S \rightarrow Person + friendOf + Person \}. \quad (3.9)$$

The terminal elements are the elements in the ontology, *Person* and *friendOf*, and the only non-terminal element is S, the starting symbol. However, it can be argued that each time an indirect friend is added, the less likely the initial person is to actually know the final person, thus considering him/her a friend. Therefore, it is reasonable to make a small modification in the production rules set adding a weight for each of the production rules, building a *weighted grammar*.

Definition 17: A *weighted grammar* is a 4-tuple

$$WG = (N, \Sigma, WP, S) \quad (3.10)$$

where *WP* is the weighted productions rules, a duple containing the production rule, as defined in the classical grammar and a weight relative to the application of this tuple. This weight is used at the end of the weight policy to define the weight of each of the syntactic equisemantic walks.

Definition 18: The *weighted syntactic equisemantic walks* relative to the weighted grammar *WG* and the weight policy Ω, $WSW(WG, \Omega)$ is a set of tuples, where each tuple is formed by a walk that agrees with a walk semantics $\omega \in L(WG)$, and the weight of this walk, following the weight policy Ω multiplied by the walk semantics weight. As for the weighted equisemantic walk definition, most of the times it is interesting to obtain its subset with defined start and end vertices v_i and v_j, represented by $WSW(v_i, v_j, WG, \Omega)$.

It is also possible to easily define an *arbitrary weighted syntactic equisemantic walks* set, following naturally the definition above. It will not be explicitly defined here because it is not of interest to any of the analysis done here. Actually, in none of the databases used for testing the algorithm an interesting syntactic feature was identified, so it will not be possible to demonstrate the implementation of this algorithm variation.

3.7. Sample Application Examples

The examples below are aimed at demonstrating the concepts presented in this chapter. They are simple examples made to be simple proof-of-concept ideas and do not aim on presenting highly usable patterns. More interesting applications on larger datasets will be presented in Chapter 7.

This example session is divided into three subsections to present each of the three concepts shown above, namely the choice of weight policies, feature-based patterns, and example-based patterns. The case of mixed features will be shown in Section 7.1, where this method will be applied for obtaining patterns for author disambiguation.

3.7.1. Analysis of weight policy on explicitly defined patterns

As a result of the discussion presented earlier, five different weight policies were implemented and compared side-by-side to confirm the natural intuition about the effects of the choice of weight policy. In order to make the example simple to understand, a very simple single feature was chosen: author self-bibliographic coupling, i.e. authors self-connected by having papers referenced by a common paper. This feature gives a good idea of the importance of the author in the field, because it presents a number that is directly related to the number of papers that this author published in the field. This feature, defined using the following walk semantics:

$$\begin{Bmatrix} Author, wrote, Publication, citedBy, Publication, \\ cites, Publication, writtenBy, Author \end{Bmatrix}, \tag{3.11}$$

is simple and important enough to be a good example to test the results obtained using various possible weight policies.

Below, five weight policies will be compared side-by-side:

a) **EW**: Equal weight to all properties regardless of the number of incoming and outgoing connections.
b) **SP**: Consider the concept that the *cites* property is static, therefore the weight policy sets the weights of these properties to be $1/k$ where k is the number of outgoing edges of the type.
c) **SPA**: Consider the property *wrote* also to be static. This means that an author has a limited knowledge that is divided among the papers written. Although this may not be a correct assumption based on the definition of a static property, its application is insightful to the way the weight policy works.

d) **CS**: Use the concept of core and scatter to remove the bias of the core by applying a weight of $1/k$ to the *cites* property in which k is the number of incoming edges to the target *Publication* object.

e) **CSS**: Combine the weights used in tests *b* and *d.*

The results can be seen in Table 3-1. This analysis shows interesting results on the effects of the weight policy. First, if there is no correction for static and growing elements, there is an observable bias on elements that have been referenced many times in a single publication. For example, 43 different papers by "Deraedt, L" were referenced in a single review paper. The simple visualization methods proposed earlier in this chapter have proven to be very effective in spotting this bias.

Table 3-1 - Comparison of different weight policies for identifying important authors in Evolutionary Computation

	Authors				
Rank	EW	SP	SPA	CS	CSS
1	Goldberg, DE	Goldberg, DE	Zhang, F	Deraedt, L	Deb, K
2	Fogel, DB	Fogel, DB	Srinivas, N	Muggleton, S	Dorigo, M
3	Deraedt, L	Dorigo, M	Nelder, JA	Dzeroski, S	Goldberg, DE
4	Back, T	Deb, K	Dorigo, M	Perelson, AS	Sakawa, M
5	Michalewicz, Z	Back, T	Elrad, T	Goldberg, DE	Fogel, DB
6	Dorigo, M	Michalewicz, Z	Nix, AE	Takagi, H	Poli, R
7	Deb, K	Holland, JH	Wolpert, DH	Fogel, DE	Michalewicz, Z
8	Muggleton, S	Koza, JR	Aoki, T	Deb, K	Back, T
9	Holland, JH	Wilson, SW	Ochotta, ES	Michalewicz, Z	Koza, JR
10	Yao, X	Fonseca, CM	Ritzel, BJ	Sakawa, M	Wilson, SW
11	Koza, JR	Zitzler, E	Back, T	Poli, R	Conrad, M
12	Fonseca, CM	Muhlenbein, H	Runarsson, TP	Dorigo, M	Ishibuchi, H
13	Cordon, O	Davis, L	Goldberg, DE	Yao, X	Cordon, O
14	Dzeroski, S	Beyer, HG	Grasse, P	Conrad, M	Muhlenbein, H
15	Glover, F	Poli, R	Fogel, DB	Cordon, O	Zitzler, E
16	Perelson, AS	Sakawa, M	Sareni, B	Glover, F	Burke, EK
17	Muhlenbein, H	Schwefel, HP	Clymer, JR	Back, T	Vose, MD
18	Wilson, SW	Rudolf, G	Bayer, HG	Delmoral, P	Yao, X
19	Schwefel, HP	Ishibuchi, H	Wilson, SW	Ishibuchi, H	Holland, JH
20	Poli, R	Vose, MD	Fonseca, CM	Lavrac, N	Rudolf, G

If the *wrote* property is also considered static, the results obtained are biased towards authors that have published few very important papers in the field, instead of many papers of lower importance. For example, "Zhang, F" published a single paper in 1997 that was referenced 19 times. While if the concept of core and

scatter is used in the weight policy, there is a natural bias towards authors that wrote a large number of papers that have not received many references in the dataset. “Deraedt, L” is again a good example of this behavior as the 43 different papers written by this author are not referenced by any other papers in the dataset.

Finally, if both weight policies are applied at the same time, the result presents authors that have been highly active in the field, with large number of papers that have received good visibility throughout the dataset, and not only by a few number of elements. This result (CSS) and the single static policy (SP) are the ones that better represent the abstract feature of interest, but it is important to note that it is possible to obtain other types of information using the same walk semantics, thus making the algorithm more powerful, enabling the observation of more subtle patterns.

3.7.2. Explicitly defined patterns

This example has the objective of showing that by manually defining the features that suggest the existence of specific patterns it is possible to apply the algorithm presented above in Subsection 3.3.1 to obtain candidates of groups that present the pattern of interest.

One important pattern to look for in collections of journal articles is of author collaboration. The most important feature for author collaboration is of co-authorship:

$$\{Author, wrote, Publication, writtenBy, Author\}\ . \tag{3.12}$$

This single feature was used to define the pattern of interest in the collection of papers in the Anthrax field. The results are shown in Table 3-2. By presenting these results to a specialist in the field, it was possible to confirm that the results are reasonable, but it was not possible to rate the ranking obtained, a common issue when analyzing the results on these databases.

Table 3-2 - Results of co-authorship collaboration without preferential connection correction

	From	To	Feature Value	Used as example?
1	Klein F	Lincoln RE	0.2360	*
2	Lincoln RE	Klein F	0.2360	
3	Mahlandt BG	Lincoln RE	0.1798	
4	Lincoln RE	Mahlandt BG	0.1798	
5	Leppla SH	Klimpel KR	0.1685	*
6	Klimpel KR	Leppla SH	0.1685	
7	Klein F	Mahlandt BG	0.1685	
8	Mahlandt BG	Klein F	0.1685	
9	Walker JS	Klein F	0.1461	
10	Klein F	Walker JS	0.1461	
11	Lincoln RE	Walker JS	0.1461	
12	Walker JS	Lincoln RE	0.1461	
13	Leppla SH	Singh Y	0.1236	*
14	Singh Y	Leppla SH	0.1236	
15	Smith H	Stanley JL	0.1124	*
16	Stanley JL	Smith H	0.1124	
17	Collier RJ	Milne JC	0.1124	
18	Milne JC	Collier RJ	0.1124	
19	Mock M	Sirard JC	0.1124	
20	Sirard JC	Mock M	0.1124	

The last column of the table will be used in the following subsection, where these elements will be given as examples to try to obtain the same results without having explicit knowledge of the feature.

3.7.3. Example-based patterns

Using the same pattern as the previous example, of spotting important authors, instead of manually defining a walk semantics of the features of interest, some examples are provided for the system and, by using all walk semantics in the ontological connections with maximum length of 4. The examples presented to the system were the same ones identified in Table 3-2. The results obtained using this method are shown in Table 3-3.

Table 3-3 - Results of co-authorship collaboration without preferential connection correction using examples for defining patterns

	From	To	Feature Value	In Top 20?
1	Ivins BE	Friedlander AM	0.5385	
2	Lincoln RE	Mahlandt BG	0.5667	*
3	Sirard JC	Mock M	0.5677	*
4	Mock M	Sirard JC	0.5704	*
5	Mahlandt BG	Lincoln RE	0.5727	*
6	Friedlander AM	Ivins BE	0.5777	
7	Milne JC	Collier RJ	0.5814	*
8	Mock M	Fouet A	0.6033	*
9	Collier RJ	Milne JC	0.6043	*
10	Klein F	Mahlandt BG	0.6113	*
11	Mahlandt BG	Klein F	0.6146	
12	Fouet A	Mock M	0.6156	
13	Ivins BE	Little SF	0.6172	
14	Klimpel KR	Singh Y	0.6234	
15	Singh Y	Klimpel KR	0.6264	
16	Little SF	Ivins BE	0.6369	
17	Smith H	Collier RJ	0.6407	
18	Collier RJ	Smith H	0.6704	
19	Smith H	Stanley JL	0.6799	*
20	Friedlander AM	Leppla SH	0.6814	

The table identifies the cases where the elements in the top 20 results using the example-based approach are present in the top 20 of the previous example. This gives a 45% match of the top 20. If these results are compared to the top 50 results of the previous example, a 100% match is observed proving that the example-based method does provide good results with a reasonably low number of examples.

3.8. Summary

This chapter presented the core method for pattern recognition that is being proposed in this study. The method is based on the use of walk semantics as features that are related to the patterns of interest. It is possible to define the pattern by direct definition of the walk semantic set possibly with weights; by presenting examples to the system; or by a combination of both. The algorithm also provides a method for introducing or dealing with database biases that is useful for extracting subtle features in the database by defining different weight policies.

This chapter also provided relevant discussions about user interface necessary for running the examples and analyzing the results obtained graphically and by a ranked table. Improvements were proposed for using dimensionality reduction methods for dealing with the curse-of-dimensionality and for dealing with syntactic patterns.

Finally, some simple application examples were shown to support the proposed algorithm. These examples have enabled the understanding of the way the algorithm works and how important is each of the parameters and feature choices given. It also provides verification of the robustness of the example-based approach, alleviating the user from hard decisions related to manually defining walk semantics of interest.

However, when implementing the algorithm, two issues were observed: using externally built databases and scalability. Databases built by external sources, such as ISI, usually have data structures that are not compatible to walk semantics that are easy to understand. Therefore, it is necessary to employ methods for transforming the database structure to facilitate the creation of these walk semantics. A method for performing this change will be presented next, in Chapter 4.

Scalability is also a very important issue in the algorithm presented in this chapter. Because of its need to perform branching in the projections related to each feature of interest, it is necessary to load into memory the whole projection for each feature in turn. For features with large walk semantics, the amount of memory required is sometimes very large. Moreover, if a large number of walk semantics is used (for example, if the user defines an example-based approach in which the maximum length of the ontological connections is too large), the time required to calculate each of the features and then obtain the total feature, or the distance to the examples is also very large. Chapter 5 proposes a sampling-based method for providing a more scalable design, both in memory and time requirements.

CHAPTER 4

Ontology Transformation

4.1. Introduction

This chapter discusses methods for database structure transformation. One of the most important processes when defining the requirements of a data processing system is to identify a database structures that are informative for the goal sought, simple to decrease the cost in data acquisition and processing, and the cost related to maintaining overall system consistency. This process, though, is only applicable when the creators of the system are the ones that are acquiring and managing the database. Lately, with the decrease in cost for data sharing and the growth of size and quality requirements for the state-of-the-art data processing systems, an increasing amount of databases used in large systems is obtained from external sources. These specialized data gathering and distribution companies usually have different system requirements when defining the database structure, thus usually not following the ideal structure for the data processing system, as mentioned above. Therefore, it is not unusual for the first step for most data processing systems to be of structure transformation, or ontology transformation.

Methods for ontology transformation, also called ontology mapping, or schema transformation, have been around for many years since the initial implementations of database systems. However, they were created in order to deal with legacy systems or the integration of multiple databases with different structures and do not relate directly to the processing requirements [47, 80, 81]. The lack of connection between these two processes in the system is usually due to the inability to define the processing requirements in compatible terms as the ontology transformation. Usually an ontology transformation contains source and target ontologies and a definition of which are the equivalent elements, and possible transformation functions that have to be applied to transform elements from one ontology to another [82]. As for the analysis process, the

usual requirements are defined on the features that are being sought for in the database (the queries) and the processing to be applied to the result of these queries. Queries are very specific to certain database structures and only in special cases it is possible to efficiently translate them from one structure to another [83]. Moreover, sometimes the processing to be applied to the result of the queries has to be translated too. This translation complexity is usually higher because of the inexistence of general formal methods for representing and manipulating the required functions.

Another main difference between the concept of ontology transformation presented here and the ontology mapping discussed in the literature is that for this implementation there is no data in the target ontology. Therefore, there are no uncertainties about entity matching (the process of defining which entities in one database relate to the entities in the other database), which may be the one of the most time and processor-consuming elements in the process [84].

This chapter is structured as follows. Section 4.2 introduces the ontology transformation algorithm. Section 4.3 analyzes the use of the ontology transformation to optimize queries in graph-structured databases and proposes an automatic transformation method to apply on databases for query optimization. Section 4.4 presents the user interface created to enable the ontology transformation. Section 4.5 describes some simple examples of application of the algorithm in two different datasets. Finally, Section 4.6 presents some conclusions obtained from the experimental results of the algorithms presented.

4.2. Ontology Transformation Algorithm

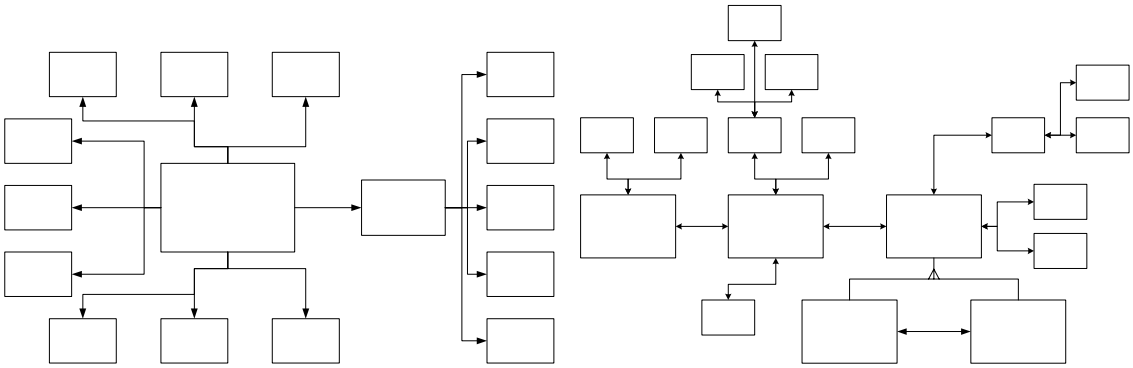
The proposed algorithm for ontology transformation is rooted in two main assumptions to which the reasoning was discussed above.

Assumption 1: The objective of the ontology transformation step is to adapt a data structure to analysis or processing. There is no interest in explicitly merging two or more databases with different data structures,

so there are no requirements for intensive element disambiguation, although some would be necessary in some transformations.

Assumption 2: It is easy for users to understand and to be able to provide the interesting walk semantics and the equivalent walk semantics on the different ontologies, given a simple and intuitive user interface.

The rationale justifying the proposed algorithm is on deciding and performing the ontology transformation in order to employ information that can be easily extracted from databases in other structures. Figure 4-1 shows an example of two ontology structures for bibliographical analysis. These ontologies were already presented in the previous chapter. For completeness of the representation, we will repeat these figures here. In the first ontology (Figure 4-1(a)), it is not possible to directly infer patterns that could suggest information related to each journal issue, such as special issues. Moreover, as references are treated as "codes" and not as papers that could be present in the database, it is impossible to directly perform analysis such as journal referencing, or self-referencing. On the other hand, the second ontology enables these kinds of analyses.



(a) **(b)**

Figure 4-1 - Example of two different ontology structures for the same problem domain (collection of journal articles)

An important detail that can be observed in the example on Figure 4-1 is that sometimes elements have to be modified in order to generate elements that can, in turn, be related to the elements in the target ontology. For example, in the ontology given in Figure 4-1(a), the date is given by a month, a month range, a season, or sometimes a day and month, while in Figure 4-1(b), in order to keep homogeneity, all dates contain day,

month and year. Therefore, extra information is necessary to be added in order to make this transformation, such as an assumption that the publication date is the middle of the given range. As can be seen in this simple example, the transformation step is highly problem-dependent and usually requires database engineers to generate a good amount of code to enable it. It is reasonable to assume that there is no general solution to this problem that can be automatically generated, but an effective algorithm has to take this step into account, because it modifies the structure of the initial database considerably.

A high-level view of the proposed transformation system can be seen in Figure 4-2. It contains three different processes, source transformation, path transformation, and target element disambiguation. Each of these processes will be discussed below in separate. After the analysis of the algorithmic steps, some details about the implemented user interface will be given. The construction of the interface is vital to enable the applicability of the process. The definition of each of the walk semantic pairs can be a highly time-consuming and error-prone process if not carefully designed.

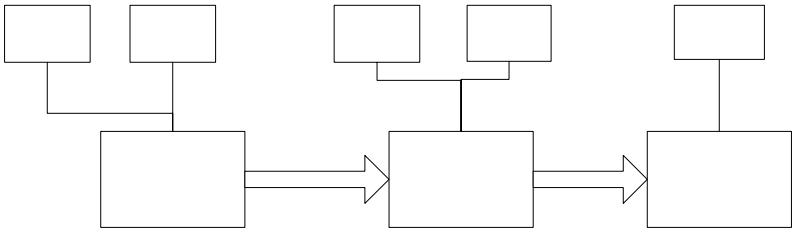


Figure 4-2 - Overview of proposed transformation process

4.2.1. Source transformation

As mentioned, this first step is focused on performing modifications in a database, when needed, to enable the identification of entities and relations that are implicit or not present in the original database structure. This process is highly problem dependent and cannot be easily automated. The input of the process is the whole database, or a selected group of entities. The output is a modified database with added entities and relations. These new entities form a subgraph that is attached to the initial source ontology and is used to determine the translation. Therefore, what is required for the analyst programming this transformation is

the definition of the ontology of the new information to be incorporated into the database and how this new information is connected to the initial data.

For example, when analyzing the ISI database of papers, one common observation is about the change in the standards in which the paper authors are assigned a name comparing to the reference authors. Reference author names are always capitalized and do not contain any non-letter character (for example, "O'Neil, J" becomes "ONEIL J"). In order to enable the matching of paper and reference author names it is necessary to transform the paper author names to this "reference author scheme." It is important to note that the opposite, although would be more interesting, is not possible, because the reference author name does not contain enough information to do the reverse transformation.

In this case, the input for the transformation function is each paper author name, and the output is the modified paper author name connected to the original paper author name. The original name is kept because of the additional information it contains. Thus, the new transformed ontology contains one extra element and one extra relationship, and this is used to enable the path transformation.

4.2.2. Path transformation

The path transformation step generates the translations of the elements and their properties. It uses pairs of equivalent walks generated by the user to define the equivalent elements and populate the elements that do not have equivalence determined. The overall algorithm can be seen in Figure 4-3.

It is important to note that, without the initial disambiguation step, the elements that would be present in between these two equivalent elements will be duplicated in this step. For example, the transformation in Figure 4-1 from a paper's *issue* and *volume* would create two distinct *source issue* objects. In order to enable the merging of these two objects, it is necessary for the user to provide information of whether the two *virtual elements* created are the same. This process, if added to the definition of the equivalent walk pair, may decrease the simplicity of the step, thus making the step harder to implement and use. Therefore,

```
function transform_walk(walk_O1, walk_O2)
for each walk w that agrees with the walk_O1 in ontology O1
      if starting_vertex(w) does not have equivalent in O2
            v1 = create equivalent
      else
            v1 = existing equivalent
      end
      if ending_vertex(w) does not have equivalent in O2
            v2 = create equivalent
      else
            v2 = existing equivalent
      end
      if in both vertices there was an equivalent already
            // Initial Disambiguation Step
            perform inner disambiguation finding the common “abstract”
                  type(s) and making them the same.
      else
            create elements and properties in between both elements
      end
end function
```

Figure 4-3 - Pseudo-code for generating the transformation using the equivalent walk pairs

as a second step after all the equivalent walk pairs are specified, the system identifies all possible equivalent *virtual elements* and displays them to the user who confirms or rejects the equivalence. The algorithm in identifying the possible equivalent *virtual elements* is actually very simple. It is based on identifying all *virtual elements* by getting the elements that never appear at the beginning or end of any equivalent walk pairs, and presenting to the users pairs of walk semantics to inquire whether in these walk semantics the *virtual elements* are equivalent.

This is an initial process of disambiguation that is directly related to the virtual elements created that are not present in the source ontology. This is a structural disambiguation step, important for completing the walk transformation step in order to obtain the actual desired structure. However, sometimes some disambiguations are related to the element values, not only the structure. These will be analyzed and dealt with in the next subsection.

4.2.3. Target element disambiguation

This final step is called for in order to further disambiguate elements that cannot be disambiguated in the previous step. These are disambiguations that are not present in the source ontology, usually related to

virtual elements and the elements that uniquely define these elements. Modern ontology representation languages add initial support to these kinds of disambiguations by defining functional dependencies and cardinality restrictions. An important deficiency that most languages have is that they support only functional dependency of single elements, i.e. if a single element is of a certain type, or has a certain value, then the element that is connected to it has its value uniquely defined. However, in most cases, a group of values determine an element. For example, a source issue is determined by the source it is an instance of, the volume number, issue number and date. If only some of these elements match, it is not possible to confirm that the virtual elements are equivalent, but it is possible to analyze the probability distribution of the equivalence and use this information to help in decision-making.

The proposed algorithm deals with these multi-valued matches and the incomplete matching scheme, where only part of the disambiguating elements is present. Each of the cases will be discussed in separate below.

4.2.3.1. **Deterministic matching scheme.** In this method, all the values that determine an element are given. The most direct method for presenting these values is by generating a group of walk semantics that connect the virtual element to its determining elements, namely the *semantic star*, as mentioned in Chapter 2. All the walk semantics start at the same element, thus making it simple for user to generate it. It is also possible to present pre-built walk semantics that connects the virtual element to all equivalent elements with short walk semantics (walk semantics of length greater than 2 is usually not related to determining elements).

After the definition is given, the algorithm extracts all virtual elements of interest and their referenced elements. If a virtual element does not contain one or more of the elements referenced, it is discarded. Using a successive sorting method, all elements that have the same values for all elements are identified and merged. Merging is based on transferring all relations that used to connect to one of the elements to the other, given that this relation was not given in the disambiguating walk semantics, and removing the first element from the database. Figure 4-4 depicts this process, in which the elements identified with the dashed lines are the matching elements. A merge requires the definition of a direction that specifies which value to

keep for elements that are of the same type. In Figure 4-4, the elements marked with X and Y are of the same type, but as the merge direction is onto the graph in the left, in the result the Y element is discarded. It is important to note that all virtual elements that are referenced in the semantic tree are also merged, because in order for the walk semantics to be equivalent, they require equivalence too, as can be seen in the circular-shaped element in Figure 4-4.

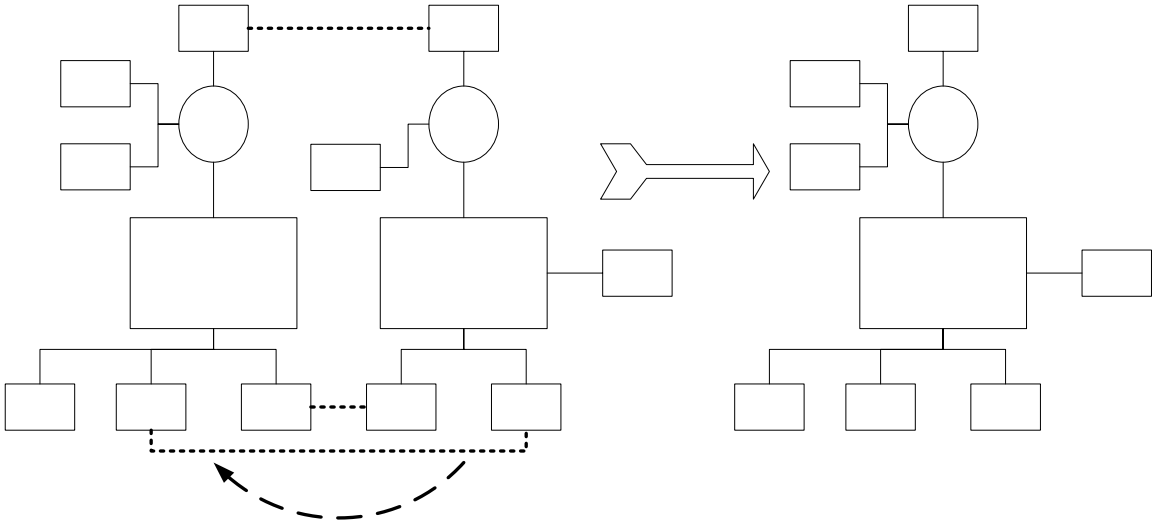


Figure 4-4 - Example of the merging process

4.2.3.2. **Incomplete matching scheme.** This method can have the same input as the previous method, or an expanded input with a larger number of walk semantics. However it does not assume that all the elements references must match. Again the values are sorted by the number of matching values and displayed to the user. The user then gives a feedback to the method on the actual matches or mismatches. This feedback information is used to generate a model of the probability of a match given the matches and mismatches of elements. This model is used to rank the possible matches not only by the number of matches, but also by the known probability of the quality of the element match being related to the virtual element match.

Specifically, the ranking scheme is based on a binomial probability measure, where initially the probability is based on the proportion of matches that the algorithm had and then it gets modified based on the responses obtained for the same types of matches following the heuristic equation given below.

$$p(M_i) = e^{-\alpha \cdot N} \left(p_0(M_i) - \hat{p}(M_i) \right) + \hat{p}(M_i), \tag{4.1}$$

where $p(M_i)$ is the final probability measure used for ranking the type of matching M_i, α is a heuristic parameter that controls the number of observations it takes for the system to base the statistics on the observed events rather than the number of elements matched, N is the total number of matches already made, $p_0(M_i)$ is the ratio of the number of matched elements by the total number of elements, and $\hat{p}(M_i)$ is the empirical probability obtained from the number of times the user identified a match divided by the total number of times the given matched types were observed and presented to the user.

The parameter α is mainly defined by the number of actual examples that exist in the database and how precise is the initial assumption that it is possible to approximate the match by the ratio of the matching values. This assumption may seem, in principle, very problematic. In the case of the disambiguation of source issues, suppose the elements that are to be matched to be the volume number, issue number, source name, year of publication and date of publication. If the matching elements do not contain the year of publication, this may most likely mean that the year of publication was wrong, because the volume usually corresponds to this value. On the other hand, if the source is different, most probably these elements are different, although they both have only one mismatch. Although with this simple example the assumption may seem incorrect, experimental results show that the ease of determination does compensate for these small errors. Moreover, a simple increase in the parameter α does make the system converge faster to the empirical probability that accounts for these effects.

In the next section, the concept of ontology transformation will be applied for optimizing queries in the database. Then, before presenting some applications of the transformation method for increasing the efficiency of the feature extraction process and showing some experimental results obtained, the user interface that was created for this system will be presented.

4.3. Query Optimization by Transformation

As introduced above, the proposed ontology transformation process can be used to optimize the feature extraction process by first transforming the data structure in such a way to minimize the length of the interesting walk semantics. Features can be extracted by analyzing the number of different equisemantic walks that exist between given two or more elements. This will be covered in more details in the next chapter. However, it was noted that the longer the length is, the larger the branching factor will be, thus the more the processing requirements will need to calculate the features. In an optimal case, it would be interesting to have all interesting walk semantics of unitary distance. However, this process does remove important information that could be used for correctly defining the weights of each of the transformed connections. In some cases it may be interesting to lower the weight of connections that pass through very "popular," i.e. highly connected, elements because of the lack of actual information that these connections provide and the fact that they tend to introduce biases into the analysis, as discussed earlier in subsection 3.7.1. By erasing the information of the actual path, it is impossible to obtain a weighting scheme.

However, in some cases, walks may not have these kinds of problems. One example when this is observed is when there are walks where each element has only a single inbound and a single outbound connection. For example, in Figure 4-1(b), the source issue element has only one connection to the source and one connection from a publication. Thus, if it would be interesting to obtain features related to connections between a publication and a source, an ontology that contained a property that relates the publication directly to its source would have its walk semantic length reduced by one. This may seem a small reduction, but it may contribute to large difference in processing time, large enough to support the cost of the transformation.

It must be noted, though, that this transformation makes the ontology harder for the user to understand. Experience has shown that the closer the ontology is from the reality, the easier it is for a user to define the walk semantics. In real life publications are contained in source issues that are "periodic instantiations" of a source; therefore, it is easier for a user to use this structure. For that reason, it would be interesting to

generate a “hidden” ontology structure that is only used for processing purposes. We name this ontology the “processing ontology,” while the transformed ontology that is best for the user to define the walk semantics is called the “user ontology.”

The construction of the processing ontology is automatic and based on the following simple functional dependency:

> If each element of type A connects only to a single element from type B that, in turn, connects only to a single element of type C, then the C elements are functionally dependent on A, therefore there can be a direct connection from A to C.

It is important to observe that most of these dependencies are created by the ontology transformation itself. Reducing all functional dependencies of the ontology in Figure 4-1(b) would obtain roughly the same elements as in the original ontology in Figure 4-1(a). The only difference is that this process does not remove the virtual elements, because they may be interesting for some of the analysis.

After the dependent elements were connected, the functional dependency and the transformation are recorded and used as a macro to transform the user’s walk semantics. For example, if there is a transformation that generated a connection between A → C where only the connection through B was available (A → B → C), now every time that the user inputs a walk semantics that contains the transformed sequence, the system automatically transforms this part of the walk semantics into a simplified version.

4.4. User Interface

As discussed in Chapter 1, the implementation of this part of the system is targeted to being a proof-of-concept study, and will not make use of highly graphical interaction methods that could be beneficial for the user experience, but does not add much to the information of whether the system would be applicable.

The proposed user interface for this process contains three main screens. The first one loads the databases and defines the interesting equivalent walk semantics pairs and can be seen in Figure 4-5. The second screen, shown in Figure 4-6, is called when the definition of walk semantics is done and the software calls the algorithm to analyze possible disambiguation of the virtual elements. Finally, in Figure 4-7 the interface for performing the disambiguation can be seen. This is a very simple interface and would definitely benefit greatly by adding a visual method to observe the disambiguation candidates. However, it has proved to be efficient for more experienced users, because they can easily observe, just using few values, whether the elements should be merged or not.

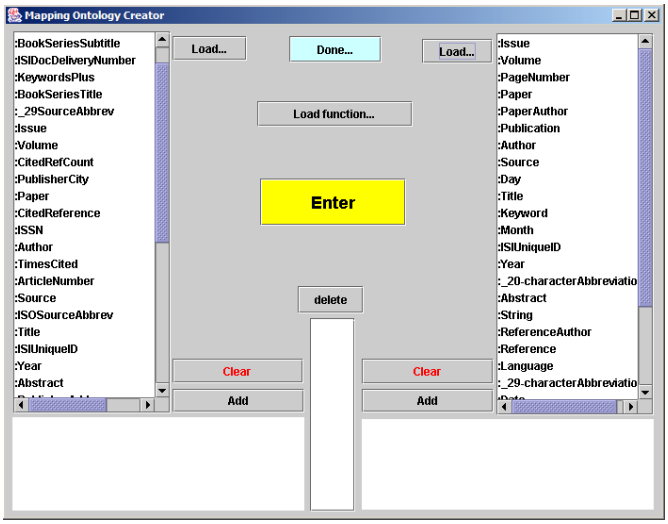


Figure 4-5 - Example of interface for building the equivalent walk semantics

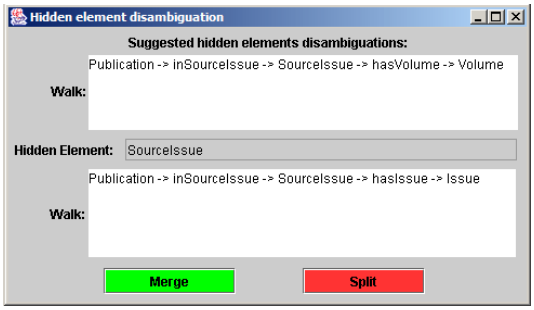


Figure 4-6 - Example of interface for confirming merge of virtual elements

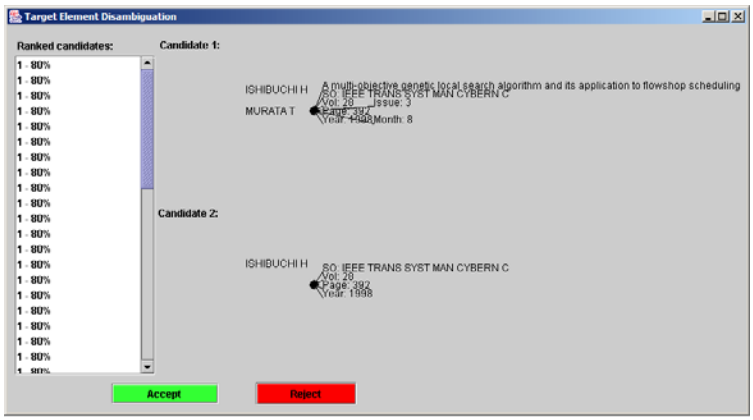


Figure 4-7 - Example of interface for doing the disambiguation of database elements

4.5. Sample Application Examples

The proposed process was applied to two different datasets with different types of transformation requirements, an earthquake events dataset from the U.S. Geological Survey database, and a scientific papers dataset from ISI Web of Science database. The simplified versions of these databases are presented in Figures 4-8 and 4-9, respectively. The complete version of these ontologies is presented in the Appendix I.

4.5.1. Earthquake event analysis

Following arguments made in [85], the information about the depth, position, time and magnitude of the earthquakes is not very efficient to use, and is not the elements that are being sought for when analyzing this kind of databases. What is interesting, on the other hand, is to identify the relations between these elements and the events. For example, it is inspiring to identify similarities between close-by events, in space and time. Moreover, it is interesting to analyze event magnitudes grouped by certain ranges such as the Modified Mercalli Intensity Scale [86], or any heuristic fuzzy interpretation of the earthquake magnitudes. For this example, five magnitude levels were used: low (1-3), medium-low (3-5), medium (5-7), high (7-9), and very-high (9 and above). Similarly the depth information was divided into shallow, medium and deep earthquakes. In summary, the desired transformed ontology can be seen in Figure 4-8. The original ontology does not contain the fuzzy relations shown in bold, only the numerical values.

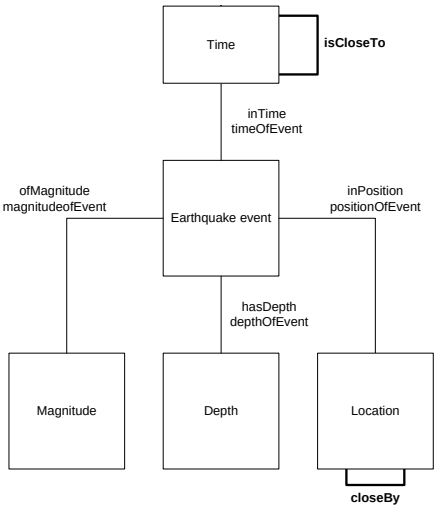


Figure 4-8 - Simplified graphical representation of ontology structure of an earthquake event database

In order to perform this transformation, the main work is done by the transformation functions that calculate the distances and time displacements for generating the close-by relation, and then an aggregation function to group the magnitudes. The generated ontology, as depicted in Figure 4-8, does not contain any structural changes besides the ones generated by the function changes, therefore there is no need to generate complex equivalent walk semantics. All generated equivalent walk semantics contain similar elements, only varying the namespace. No disambiguation process is required.

One simple example of the application of such transformation is on identification of the position of active tectonic regions. This can be performed by analyzing the following walk semantics:

$$\{Location, hasEvent, Event, inLocation, Location, closeBy,$$
$$Location, hasEvent, Event, inLocation, Location\}$$

There are three main concepts behind the choice of this walk semantics: 1) it is interesting to first identify a place that has a large number of events connected to it. It is important to note, though, that because the locations are given in terms of coordinates of the center of the earthquake, there are very few actual repeated locations; 2) tectonic regions are areas in which a large number of earthquakes are observed

throughout time mainly related to a position where there is collision between tectonic plates; 3) the presence of a large number of earthquakes in a location near by the first highly active location is another important indication of highly active regions.

The results from the application of this walk semantics can be seen in Table 4-1, after removing the redundant cases (where both locations are the same and where the locations are just switched). It shows the number of equisemantic walks, the coordinates and a manual description of the place. By simple inspection, it can be observed that these is a large bias toward a specific region in the Arabian Sea because the database has 236 events registered to that position (50.20, 12.50), where most of these events occurred between September and November 2000. This shows that for this high detail in the position of earthquakes, the use of the given walk semantics may not be appropriate as identified before.

Table 4-1 - Top 10 results of the analysis of active earthquake regions

	Location 1			Location 2			
Rank	**Latitude**	**Longitude**	**Place**	**Latitude**	**Longitude**	**Place**	**Count**
1	50.20	12.50	Arabian Sea	50.20	12.40	Arabian Sea	55,696
2	50.20	12.50	Arabian Sea	46.06	14.77	Arabian Sea	13,924
3	50.20	12.50	Arabian Sea	46.05	14.79	Arabian Sea	6,844
4	50.20	12.50	Arabian Sea	46.05	14.78	Arabian Sea	4,956
5	46.22	-122.19	California	46.21	-122.18	California	4,225
6	50.20	12.50	Arabian Sea	50.22	12.44	Arabian Sea	4,030
7	46.22	-122.18	California	46.21	-122.18	California	3,900
8	46.07	14.77	Yemen	50.20	12.50	Arabian Sea	3,304
9	50.20	12.50	Arabian Sea	50.22	12.43	Arabian Sea	2,832
10	44.26	8.21	Ethiopia	44.26	8.20	Ethiopia	2,795

A simple modification of the database was then proposed. Instead of employing all events, something that has shown to bias the results to positions where there are better equipments for spotting very weak earthquakes, only the events which the magnitude of the earthquake is mild or higher (5 degrees or higher) were considered. This decreases the number of events of interest to 48,725. A second modification was that all positions were rounded to the closest integer. This decreases the precision of the measurements making it more possible for events to be in the same position. The results obtained after this modification in the mapping methods is shown in Table 4-2 after removing the redundant elements.

Table 4-2 - Top 10 results of the analysis of active earthquake regions using only major earthquakes and integer positions

	Location 1			Location 2			
Rank	**Latitude**	**Longitude**	**Place**	**Latitude**	**Longitude**	**Place**	**Count**
1	-5	153	Papua New Guinea	-7	155	Papua New Guinea	65,084
2	-5	153	Papua New Guinea	-5	152	Papua New Guinea	62,965
3	2	127	Indonesia	1	126	Indonesia	51,912
4	-6	155	Papua New Guinea	-5	153	Papua New Guinea	47,278
5	-7	155	Papua New Guinea	-5	153	Papua New Guinea	43,460
6	-21	-179	Fiji	-18	-179	Fiji	42,354
7	-7	156	Papua New Guinea	-5	153	Papua New Guinea	42,059
8	-18	-179	Fiji	-18	-178	Fiji	41,625
9	44	149	Japan	44	148	Japan	40,044
10	6	126	Philippines	2	127	Philippines	39,552

This test shows a major difference in the results obtained depending on the pre-processing choices made. For this new database, the walk semantics chosen is more meaningful, because a location is not that sensitive to location accuracy and sensitivity of earthquake spotting equipment. It is easy to see, from these results, that the islands in the west of the Pacific "Ring of Fire" are highly active in moderate to large earthquake activity. More detailed analysis of highly active earthquake regions will be given in Section 7.2.

4.5.2. Scientific papers database

ISI's scientific papers database is centered on the information surrounding a journal paper. As a direct consequence of this focus, by the way the database is presented it is a great challenge to obtain information about specific journal issues, for instance, because this information is not contained as a single entity in the database.

In order to alleviate this problem and to also enable the treatment of some of the references as papers existing in the database, the ontology structure presented in Figure 4-1(b) was devised. However, because of the creation of new elements and the special treatment to references, the ontology transformation process

requires the full use of all proposed processes. Each of the processes will be explained in more details below.

4.5.2.1. **Transformation functions.** The chosen transformation functions have to deal with a number of details in the ISI database.

Paper author names are stored using capital letters only for the first letter of the family name (or also for middle letters when the name is compound) and other graphical symbols, e.g., “DeGould, J,” or “O’Reilly, TB.” At the same time, reference author names are all capitalized and do not contain non-letter characters, e.g. “DEGOULD J” or “OREILLY TB.” This small difference creates an extra complication, because all paper author names, in order to match to their reference counterparts, they have to be translated into the way the references are stored. The inverse transformation is not possible, because there is information loss in the process of generating reference author names.

References contain only the first author, the reference source, the volume number, the beginning page number and the year. All this information is actually obtained only on a comma separated string that has to be parsed before the analysis. Moreover, it is very common for some of these values to be incorrect, in particular volume numbers and page numbers. It is necessary to do a statistical analysis on these values in order to enable disambiguation with the papers in the dataset.

The source names on the references are abbreviated following a scheme that is contained in a different database. This has to be added to the original data structure by means of another transformation function. However, not all reference source names are present in this extra abbreviation database. Especially when the source is a book name, this book is abbreviated but the abbreviation does not appear in the database. Therefore, as in the first transformation, the inverse transformation is not feasible.

The dates are divided into two different elements, the year and a date element that may contain a day and month, only a month, a month range or a season. It would be useful, in order to facilitate analysis, to join

this information in a single element that would contain information about the date range (when the specific data is not known).

The following transformation functions were devised to deal with the issues listed:

- **referencize-names:** generates the name in the reference standard (all capital letters and with no non-letter characters) and connects to the original name.
- **split-reference:** parses the reference string into the single elements (first author name, source in reference format, volume number, starting page number and year).
- **referencize-sources:** obtains and attach the abbreviations from the source names.
- **specify-date:** combines the date elements into a single date range object that contains a standardized information (not using non-specific definitions such as seasons) about the date of publication.

After these four functions are applied, the generation of the equivalent semantic pairs follows. This will be discusses in the next section.

4.5.2.2. **Walk transformation.** The generation of the equivalent walk semantic pairs is straightforward in most of the cases. It was seen that the most efficient method was to try and keep all walk semantics as short as possible (in this case, always a pair of elements in the source side worked in all cases). As for the hidden element disambiguation, in all cases it was chosen to merge the elements, because in no case there is a doubt whether the elements should be treated separately or not. Table 4-3 shows a sample of interesting equivalent walk semantic pairs. The source elements in italics are the ones that were created by the transformation functions described in Subsection 4.2.1. The target elements on italics are the virtual elements created in the ontology.

Table 4-3 - Example of some interesting equivalent walk semantic pairs

Source Walk Semantics	Target Walk Semantics
{Paper, hasIssue, Issue}	{Paper, ofSourceIssue, *SourceIssue*, hasIssue, Issue}
{Paper, hasSource, Source}	{Paper, ofSourceIssue, *SourceIssue*, ofSource, *Source*, hasName, Name}
{Paper, hasReference, Reference}	{Paper, cites, Reference}
{Reference, *hasFirstAuthor*, *FirstAuthorName*}	{Reference, hasFirstAuthor, Author, hasReferenceName, ReferenceName}
{Reference, *hasSource*, *ReferenceSource*}	{Reference, ofSourceIssue, *SourceIssue*, ofSource, *Source*, hasReferenceName, ReferenceName}

The source in the target ontology became a virtual element because it is more general to code a name entity as a string than to accept the transformation of a physical element into a string value. This concept was also used with authors and author names. The only issue about this transformation is that, as expected, the average length of the paths increased with this transformation, because the target structure is larger. On the other hand, this new structure is more informative and allows for the definition of more interesting features.

4.5.2.3. **Target element disambiguation.** There are two main elements that should be disambiguated: the authors and paper-reference. For the authors, it is necessary to match the authors that have the same reference name. It is assumed, for this application that having the same reference name is a deterministic property to disambiguate authors. This assumption may not be completely valid when dealing with large and highly heterogeneous databases, but for the scope of this study there will not be any considerations regarding these concerns.

The disambiguation semantic star is actually a single walk semantics: Author → hasReferenceName → ReferenceName. It is important to note that this disambiguation also works for disambiguating paper authors, because the transformation function creates the reference name for all authors and it is assumed that this process, although it does result in information loss, does not cause problems in disambiguating the names for the case under analysis.

The disambiguation of papers and references is a little bit more complicated. The matching has to be done on a number of elements that may be reasonably distant from the disambiguating element (the publication).

A second assumption has to be made here. It is assumed that if a reference matches the first author of a paper, the abbreviated source name, the year of publication, the beginning page and the volume number, they are surely the same paper. This assumption is very reasonable. Experience with a number of datasets never presented an exception to this rule. Therefore, the semantic star for papers and references disambiguation is given in Table 4-4.

Table 4-4 - Semantic star definition for disambiguation of papers and references

Walk semantics
{Publication, hasFirstAuthor, Author}
{Publication, hasSourceIssue, SourceIssue, inDate, Date, inYear, Year}
{Publication, hasStartingPage, StartingPage}
{Publication, hasSourceIssue, SourceIssue, ofSource, Source, hasReferenceAbbreviation, ReferenceAbbreviation}
{Publication, hasSourceIssue, SourceIssue, ofVolume, Volume}

It is important to note that in this case if there were repeated papers acquired in the dataset, because of a data gathering mistake, they would be disambiguated in this step too. The disambiguation step, due to the proposed algorithm, would merge the elements of the whole walk semantics presented in the semantic star, thus combining the source issue and source elements. The author elements do not need to be disambiguated because of the previous disambiguation step.

After this deterministic merging has been performed, it is interesting to analyze cases where only some of the values of the paper-reference disambiguation match. Based on the proposed ranking method, the candidate disambiguation elements were presented to the user. From the results of the feedback from the user, a final probability for merging based on the type of matches was obtained. These probabilities for the cases where at most 2 elements did not match are shown in Table 4-4 for a dataset composed of 24,858 publications in the Evolutionary Computation field, in which 2,568 are papers (i.e., in the ISI database as papers and not only as references). In order to obtain the statistics, a sample of 20 pairs of each of the cases were obtained and presented to the user. Most of these results are expected, such as the fact that a volume number and source name are enough to define the year, as well as the source name and year are enough to define the volume number. When there is a high merge probability, such as non-matching first author name, or year, after presenting all 20 examples to the user, the system could already assume that this

disambiguation could be done automatically for all other cases without adding much error to the database. However, the results showing very similar probabilities for both outcomes have to be either analyzed using possibly other elements, not only the five elements shown here (when other elements are available – in this case the only information obtained for the references are these five elements), or decided manually.

From the number of cases observed in this reasonably sized dataset, it would be necessary for the user to manually go through a large number of cases to ensure a low error in the disambiguation process for the database. However, this is originated from the nature of the database, due to the large amount of mistakes in the references. The most common mistakes observed are: wrong page number, volume numbers, year of publication, spelling of the author's name, registering second authors as first authors, switching the first and last name of authors, not registering middle initials of author's names, and the use of different abbreviations for reference sources. A large number of cases were identified in which the author name and page numbers are different is due to multiple publications in the database from the same source issue (for example, papers in the proceedings of large evolutionary computation conferences receive a large amount of references for a number of different papers).

In summary of all the experimental results presented here, it can be concluded that the proposed method can be applied for both simple transformations, such as the one of the earthquake dataset, or on more complex problems, in which there are a number of element disambiguation concerns that are not present in the original dataset. The proposed transformation of the scientific papers dataset is a good example of this later case. Both transformations directly present some interesting aspects of the databases. At the earthquake database, for instance, by applying the transformation it was observed that a point location (defined by a single pair of latitude and longitude) is not very effective to use for defining areas of high earthquake occurrences, because of artifacts created by single positions in which a large sequence of events were observed in a short period of time. As for the scientific papers dataset, by observing the disambiguation patterns it was possible to have a better insight on the common mistakes that exist in the database. In some cases, it is possible to feed the mistakes back to the data acquisition system to facilitate the improvement of the initial database.

4.6. Summary

In order to deal with the common need to transform the ontology of databases for specific use, this chapter proposes a method based on the translation of the walk semantics of the database. It was argued that this process is more efficient and less time-consuming for the user to generate, and it contains enough information to perform element disambiguation, a very important operation usually needed after ontology transformations.

The proposed algorithm is efficient and of easy implementation. The ease of defining walk semantics increases the simplicity of the user interface required to generate the needed information for the process, namely the equivalent walk semantic pairs and the equivalent virtual elements. It was demonstrated that the disambiguation most of the times is very simple, but can also require extensive manual work if there are large problems in the database, such as when using a dataset of scientific papers and references. At the same time, for simple transformations such as the one exemplified with the earthquake dataset, the whole process is very simple and provides a transformation that is easy to understand and creates a structure that is more efficient to use for many applications.

The implementation demonstrated here was a simple proof-of-concept to enable the understanding of the process and to assure its effectiveness. However, there are some elements that could still be improved upon, such as a more natural analysis of the disambiguation. In the disambiguation process, sometimes numerical elements could be essential. For example, if there are three publications in the database that are being disambiguated, A, B and C, where A has a publication year of 2000, B of 1999 and C of 1990, there is a greater chance that A and B are equivalent than A and C or B and C. Also the knowledge of common aliases for certain sources can also be of great use to decrease the amount of manual work required in this step. For example, if the software learned from observation that publications with a source “T EVOL COMPUT” always matches with publications with source “TRANS EVOLUT COMP,” the next time that this is observed in the database it may not need to present it to the user.

Overall, the algorithms and processes presented here are important steps toward increasing the ability for database users to effectively employ databases from external sources. By successfully applying the proposed processes, users are alleviated from the tedious and time-consuming programming usually required to transform the data into a usable and efficient form. Moreover, by being able to efficiently examine the data in different structures for regularities and disambiguation, a better insight in the database contents and possible shortcomings from its use can be acquired. Finally, by employing the concepts present in the proposed algorithm, automatic and transparent modifications can be added to the database so that it would improve the efficiency of the feature extraction process.

The next chapter will analyze another important issue when trying to perform the feature extraction, namely the scalability of the method, especially when dealing with memory constraints.

CHAPTER 5

Approximating Features by Sampling

5.1. Introduction

As mentioned earlier, the algorithm presented in this document suffers greatly with scalability. This is a common issue reported in the literature of algorithms dealing with graph-structured databases in general [16]. The solution that is most deemed as natural to alleviate this deficiency is by employing methods that can be implemented in distributed and parallel systems [85, 87]. In this study a new approach to this problem is taken: sampling. It is based on the assumption that by observing just part of the total population, it is possible to infer what the general behavior of the total population is.

Some applications of sampling for social network characterization were reported in the literature [88, 89], as well as for analysis of large databases in search for association rules [90-92]. However, in the case of graph-structured databases, the use of this procedure is not yet fully explored. These databases present some interesting properties that might cause sampling to be highly inefficient. The main source for this possible inefficiency is the high branching factors caused by highly skewed connection distribution and positive correlation between the degrees from the various partitions. This inefficiency will be better explored in Section 5.4.

This chapter proposed two algorithms for performing sampling: naïve sampling (N-S) and evolutionary-algorithm sampling (EA-S). The first algorithm will be used as the basis for the sampling analysis because its behavior is more predictable and it yields optimal sampling results asymptotically. The second algorithm was developed to make use of simple heuristics to speed up convergence of the elements with higher feature values. In most cases, the main goal is to identify and rank these elements, while the elements of

low feature values can be under-sampled as the inherent noise of the database renders analysis of these elements unusable.

The remaining of this chapter is organized in six sections. Section 5.2 briefly presents the concept of random sampling in graph-structured databases used throughout the methods discussed in the chapter. Section 5.3 presents the naïve sampling approach and highlights the cases when the use of this method is not very efficient. Section 5.4 analyzes some well-known features of graph-structured databases, especially when they are cases of complex networks. Section 5.5 proposes the evolutionary algorithm sampling approach. Section 5.6 presents a couple of experimental results and Section 5.7 discusses the conclusions that can be taken from the methods developed in the chapter.

5.2. Sampling for Approximating Structural Features

As mentioned before, sampling is a well-known method for approximating real-world signals or elements when it is impossible, due to physical or cost constraints, to use the whole information in the system. In all sampling systems, the most important parameter to define is the amount of sampling that should be taken, or the sampling rate. In order to derive the "optimal" sampling frequency, first it is necessary to define what is sampling in the context of structural feature extraction. In this direction, two different sampling approaches are proposed and analyzed below, the start-finish and the start-path-finish approaches.

5.2.1. The start-finish approach

As explained in previous chapters, when performing the structural feature extraction, the elements and property types to be traversed during a walk are defined. The most important elements of this walk are the starting and finishing elements, because the walk defines a connection between them. Therefore, sampling can be performed by choosing a starting element of the starting element type and performing a *random walk* that follows the feature of interest and then returning to the finishing element.

The *random walk* procedure is defined as a sequence of random selections by a "walker" of which out-edge to traverse. As the sampling objective is to employ the least amount of memory and approximate the feature values, each out-edge has equal probability of being chosen. For example, if a simple walk of length 1 is performed where there are 4 out-edges of the initial vertex, each output vertex would be traversed with a 25% chance.

One first problem that is observed when performing this kind of sampling is that what is interesting is the amount of different walks that connect two elements (the size of the equisemantic walks set). This method does not offer any information to assist in determining if two distinct samples that start and end at the same elements are distinct or not. Figure 5-1 shows a very simple example how this lack of information can create artifacts when only using the start-finish sampling approach. In the figure, Ps is the probability of sampling and F is the actual feature value. Note that for this method to work, it is necessary that the values of Ps and F to be proportional, which does not happen in this simple example.

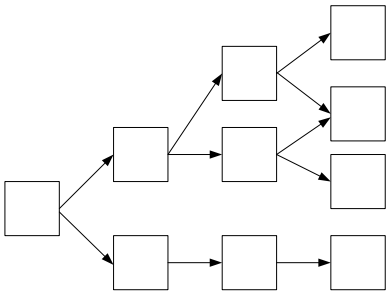


Figure 5-1 - Example of artifacts caused by start-finish sampling

This approach contains the least information possible, and uses the least amount of memory to run. On the other hand, the aforementioned artifacts created can be very problematic in most datasets, because even with a very large number of samples, the results obtained would not reflect the actual value of the feature. This observation led to the need of incorporating information about the path and the creation of the start-path-finish sampling procedure.

5.2.2. The start-path-finish sampling procedure

In this method, besides the output of the walk, the random walk method also returns a unique identifier of the path traversed. This unique identifier can be the sequence of the index of the out edge chosen (if they are indexed), the labels of the elements traversed, or even a hashing function that maps these values to another domain making it challenging to reconstruct the whole path (in case where the elements inside the network should not be made identifiable to the user). In this case, there will not be any undesired artifact in the sampling. On the other hand, it will be necessary to maintain a table of all the paths traversed and this table has to be checked every time there is an input-output match to see if the path was not sampled before. This extra processing and information stored can lead to an undesired increase in the cost of the sampling, but it does not require any extra memory use, because this information can be stored in disk and is not expected to be highly accessed even for large networks.

This method still uses equal probability for each out-edges at each walk step. It will be shown below that adding information about the number of choices that have to be made in the remaining part of the walk to each out-edge would help greatly on decreasing the amount of samples required. However, this additional information is highly costly in the amount of memory needed to store it. As the objective of the proposed sampling method is to decrease the memory requirements, this modification will not be implemented, only analyzed in the following section, where the needed amount of samples for obtaining a good approximation is calculated.

5.3. Naïve Sampling Policy

When analyzing the process of sampling, one important concept is presented: the sampling discovery rate. *Sampling discovery rate* is the rate in which the sampling mechanism is able to discover similar walks and, thus, approximate the structural feature value. Because of the nature of the random walk defined above, it is only possible to determine a probability measure that all paths have been visited. This probability measure

is directly related to the maximum branching factor, i.e. the walk in which the product of the number of out-edges in each step is maximal, b_{max}.

$$P(v_i) = 1 - \left(1 - \frac{1}{b_{\max}(v_i)}\right)^{n(v_i)}, \tag{5.1}$$

where $P(v_i)$ is the probability that all vertices of v_i were sampled, also called the probability of complete traversal, and $n(v_i)$ is the number of times the vertex v_i was sampled. This equation comes directly from the composite binomial distribution with probability $q = 1/b_{\max}(v_i)$.

With this probability measure it is possible to define a simple policy for sampling that has the objective of stepwise increasing this probability of sampling throughout the whole dataset. The algorithm, hereby called naïve sampling algorithm, is presented in pseudo-code in Table 5-1. It is important to note that without sampling it is impossible to determine the value of the maximum branch factor. While sampling there is a chance that the sampling path will pass through the maximum branching factor path. However, until all paths have been traversed it is impossible to correctly determine this value. As the sampling of all paths requires, in theory, infinite time, the approximation of the maximum branching factor to the currently observed maximum branch factor is used.

Table 5-1 – Pseudo-code for Naïve sampling policy

```
Initialize bmax by sampling all vertices N_init times
Calculate the P(vi) for all vertices
While (min(P(vi)) < Threshold)
       Get the vertex vi with the smallest P(vi)
       Sample the vertex vi N_samp times
       Check if the bmax has to be updated
       Update P(vi)
End While
```

Another observation that has to be made is that there is one case in which there is no need for sampling more than once, when the maximum branching factor is 1, i.e. the random walk procedure does not have to make any choices. In all other cases, it is necessary to perform infinite number of samples to ensure that all paths have been traversed.

Experimental results of applying this method have shown that the rate of increase of the traversing probability is very low, as can be seen in Figure 5-2. This figure was obtained when performing sampling in the Anthrax dataset for the author bibliographic coupling feature (more information about this dataset and the features that can be extracted can be found in Section 7.1), a 5-partite case. This slow convergence has led to the need for a new algorithm, presented in Section 5.5. The large amount of spikes downwards in the convergence is caused by the discovery of a new path with a higher branching factor, therefore decreasing the probability of complete traversal. As the number of samples for each step of the algorithm is $N_{samp} = 10^3$, and the total number of steps displayed here are 10^5, the total number of samples that brought the approximated minimum complete traversal probability to approximately 3% is of 10^8 samples. This is a very large number of samples for such coverage of this reasonably small network (more details about the network used can be found in Section 7.1).

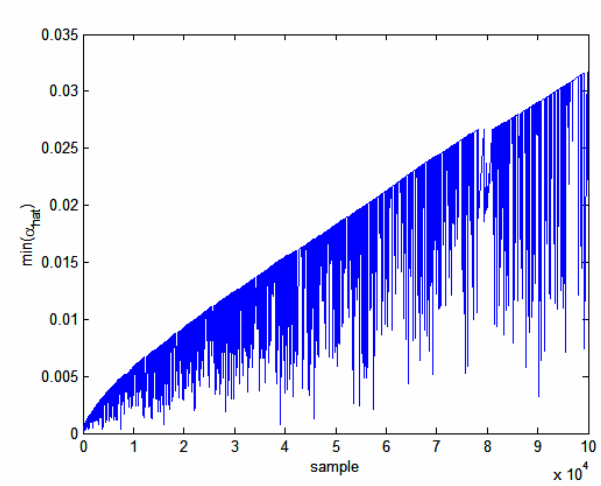


Figure 5-2 - Behavior of the convergence of the minimum branching probability for a real-world database (Anthrax papers database). N_{init} = 1,000 and N_{samp} = 1,000.

One of the most important aspects for the success of the sampling method is the maximum branching factor that a network possessed and the amount of paths with large branching factors. This aspect will be analyzed in the following section, relating this feature with the nature of the database.

5.4. Branching Factor in Real-World Databases

Real-world graph-structured databases usually are instances of complex networks, as discussed in Section 1.3. Because of the highly varying degree throughout a complex network, the branching factor is also highly varying, usually being very low for most of the walks, but very high for a few cases. However, as explained above, the decisive factor for memory requirements for the processing of graph-structured databases is the maximum branching factor. As for sampling, the lower the number of highly branching vertices is, the more efficient the sampling process will be. This section will first analyze the branching factor behavior using theoretical distributions of the degree probabilities. Later in the section, empirical observations will be made to compare to the expected theoretical values.

5.4.1. Branching factor analysis of graph models

Reviewing the concepts presented in Section 1.3, in complex network theory, it is common to model the distribution of the degree of the vertices by a Zeta distribution [40-42, 93]. The Zeta distribution, also called discrete Pareto distribution, is defined as [31]:

$$P(k) = \frac{k^{-\gamma}}{\zeta(\gamma)}, \tag{5.2}$$

where $P(k)$ is the probability of the given vertex to have degree k, γ is the Zeta distribution parameter and $\zeta(\gamma)$ is the Riemann zeta function that has the objective of normalizing the probability distribution.

When combining N partitions of preferentially connected vertices, the closed-form expression of the probability distribution of the branching factor becomes hard to determine, because it is directly related to the factorization of the final branching factor b:

$$P(b) \propto \sum_{k_1 \cdot k_2 \cdots k_{n-1} = b} \prod_{i=1}^{N-1} k_i^{-\gamma_i}, \tag{5.3}$$

where k_i is the degree of partition i, and γ_i is the Zeta distribution parameter for the partition i. It is important to note that Equation (5.3) assumes independence between partitions. This actual independence is not observed in real-world problems, but it is assumed in this part of the analysis for simplification purposes. Section 4.2. will present an empirical analysis of this dependency and its consequence.

The degree distribution of a walk of length 1 is given in Figure 5-3 for $\gamma = 2.0$. For a walk of length 4 (4 partitions) and using the same parameter $\gamma = 2.0$ for all paritions, the degree distribution is given in Figure 5-4. Note that the latter distribution is very steep, thus it is very difficult to observe anything when plotting the distribution, only a very large peak at 0 and a long and shallow tail.

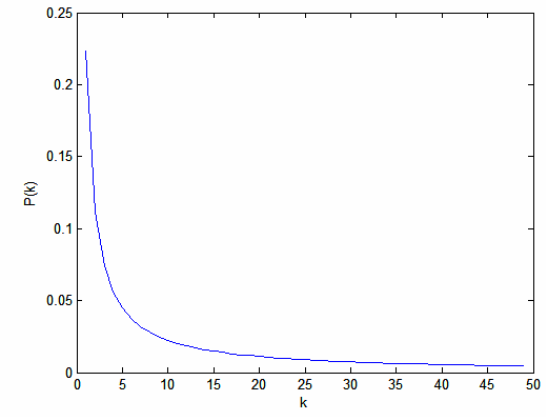


Figure 5-3 - Sample Zeta distribution for single partition, with $\gamma = 2.0$

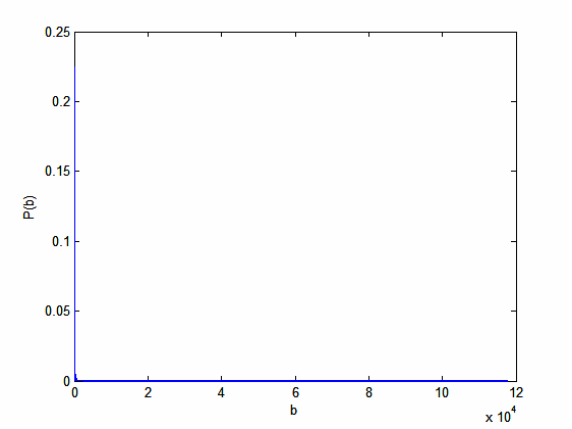


Figure 5-4 - Sample branching factor distribution of 4 parititions with degrees following Zeta distribution with parameter $\gamma = 2.0$

In order to enable the understanding of the effects of Zeta-distributed degree distributions, it is important to compare these distributions with a classic randomly connected network. In this case, the degree distribution follows a binomial distribution [94]:

$$P(k_i) = \binom{N_{i+1}-1}{k_i} p_i^{k_i} (1-p_i)^{N_{i+1}-1-k_i}, \tag{5.4}$$

where N_i is the number of edges in partition i, and p_i is the probability of connection when the network is being formed for partition i. Figure 5-5 shows the distribution of the degree for a single partition when p = 0.4 and N = 50. Following the same procedure when combining partitions to obtain the branching factor, the closed-form distribution of this branching factor is also difficult to obtain:

$$P(b) = \sum_{k_1 \cdot k_2 \cdots k_{n-1} = b} \prod_{i=1}^{n-1} \binom{N_{i+1}-1}{k_i} p_i^{k_i} (1-p_i)^{N_{i+1}-1-k_i}. \tag{5.5}$$

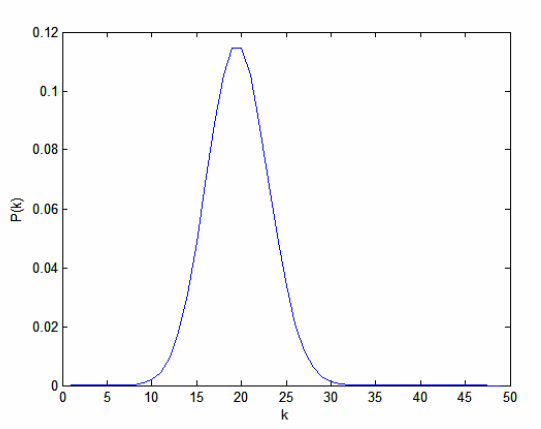


Figure 5-5 - Sample degree distribution of single randomly connected partition with p = 0.4 and N = 50

The plot of the sample distribution for 4 partitions of 50 edges each and p_i = 0.4 is given in Figure 5-6. It is important to note that the derivation of this combined probability distribution also assumes independence among the partitions. Visual inspection for comparing both distributions shows that the Zeta-distributed partitions present a much higher level of low branching factors. However, when analyzing the width of the tail of the distribution, one observes that the Zeta-distributed network contains a much larger amount of elements with high branching factors than the randomly connected network. The 99th percentile of the branching factor for the Zeta-distributed network is at 21,021, while for the randomly distributed network it is at 14,000. One percent of the vertices for a large network can be a very large number of vertices. Even with the fact that for large branching factors the amount increased in the number of samples required to obtain the same probability of complete traversal is linear with the increase on the maximum branching factor:

$$1-\left(1-\frac{1}{b_{\max}(v_i)}\right)^{n(v_i)} = 1-\left(1-\frac{1}{\alpha \cdot b_{\max}(v_i)}\right)^{n'(v_i)} = P(v_i)$$

$$\frac{n'(v_i)}{n(v_i)} = \frac{\log\left(1-\frac{1}{b_{\max}(v_i)}\right)}{\log\left(1-\frac{1}{\alpha \cdot b_{\max}(v_i)}\right)} \approx \alpha \quad for\ b_{\max}(v_i) \gg 1 \qquad (5.6)$$

In these equations, $n'(v_i)$ is the number of samples made after increasing the branching factor by the factor α.

Because of the large amount of vertices with high branching factors in the simple Zeta distributed networks, these are significantly more costly to deal with.

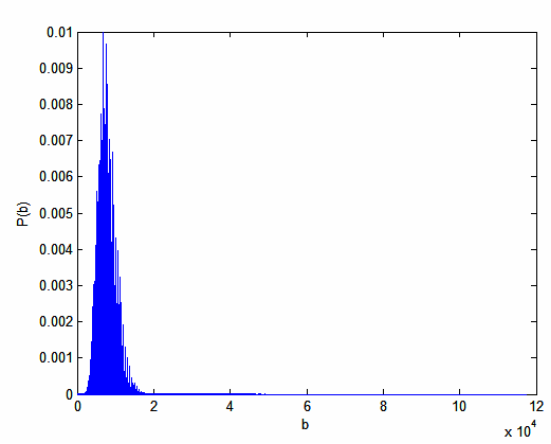


Figure 5-6 - Sample branching factor distribution of randomly connected 4-partite network with p = 0.4 and N = 50

In order to observe the tail of the Zeta distribution, Figure 5-7 shows the same distribution as Figure 5-4 but using a log-log scale. In both Figures 5-6 and 5-7 it is interesting to observe a very large amount of discontinuities in the distribution. This is due to factorization problem. For example, for the randomly connected networks, the maximum degree acceptable is the size of the other partition. Therefore, for the given example where all partitions have 50 vertices, the maximum degree for each partition is 50, thus all

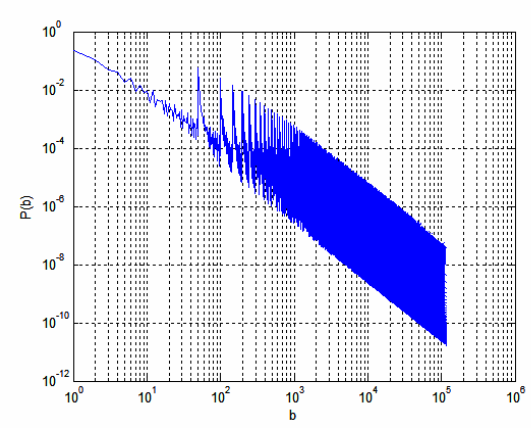


Figure 5-7 - Log-log plot of the branching function probability distribution for Zeta-distributed partitions

possible branching factors for the whole network has to be a factor of three numbers between 1 and 50. The branching factor can never be a prime number of value greater than 50 (51, for example). The same is observed on the Zeta-distributed case, although the Zeta distribution assumes infinitely large partitions.

5.5. Real-World Examples of Branching Factors

As mentioned before, in actual databases, the distributions do not follow exactly what was shown above. There are actually two important factors that contribute to this difference: 1) the empirical distributions do not always follow a well-known theoretical distribution function [93]; 2) the assumption of independence between the degrees of the vertices in each partition is not necessarily true. This section will concentrate on the second factor; because if proven, it can be exploited for improving the sampling procedure.

Figures 5-8 and 5-9 respectively show some examples of the empirical distribution of branching factors for two different features: the determination of important earthquakes in the earthquake database (more details about this dataset is presented in Section 7.2) and the identification of important authors in a journal paper database containing articles about Anthrax research (see Section 7.1 for details about the dataset).

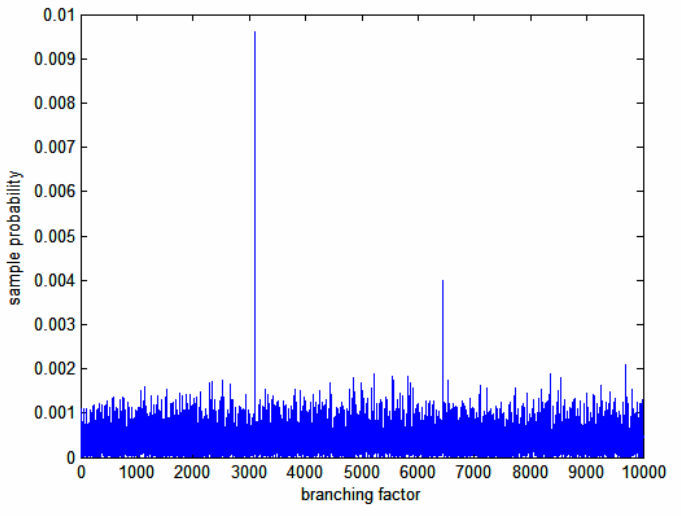


Figure 5-8 - Empirical probability distribution of the branching factor for the Anthrax dataset for the author bibliographic coupling feature

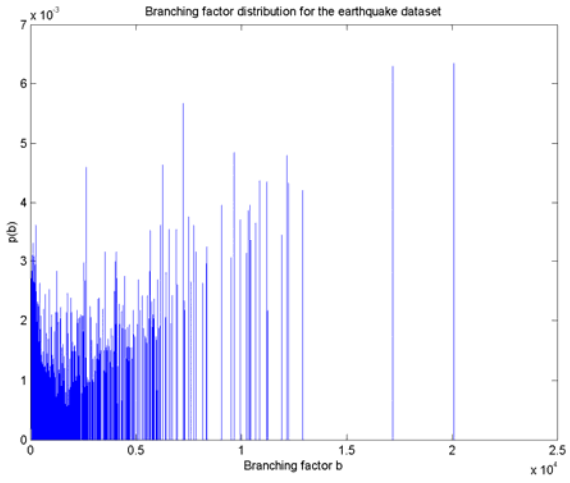


Figure 5-9 - Empirical probability distribution of the branching factor for the earthquake dataset for the active earthquake areas

When analyzing a database with earthquake events with structure given in Figure 4-8 (and in Appendix I-4), it is possible to define a single walk semantics to identify areas with high incidence of earthquake events. This walk semantics is shown in Figure 5-10. In this case the area is defined by the pair of locations.

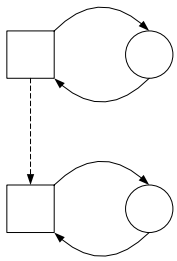


Figure 5-10 - Walk semantics for identification of active earthquake areas

The feature of interest when determining the important authors in a journal database is directly related to the amount of citations that papers written by this person receives. This can be translated into the walk semantics shown in Figure 5-11.

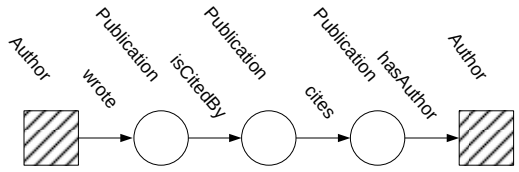


Figure 5-11 - Walk to represent important authors in a journal database

None of these distributions visually resembles the theoretical distributions observed in Section 5.4. It is interesting to observe the quantiles of these distributions. Table 5-2 presents the 0.95, 0.99 and 0.999 quantiles of these empirical distributions.

Table 5-2 - Quantiles of the empirical distributions

	Databases	
Quantile	**Earthquake areas**	**Important authors**
0.95	10,340	520,659
0.99	17,192	1,489,860
0.999	20,085	3,747,612

This table shows that the branching factor of earthquake events is degrees of magnitude lower than the ones observed in the journal database. This is expected, because most degree distributions in this database are very simple. For example, all events happen in only one position. Also the number of close-by of a position is limited by the threshold definition of what can be a close-by element. Therefore, it requires fewer samples for obtaining a good approximation of the feature values, as it is going to be shown in the next section. On the other hand, there is still a positive correlation between the number of events in a position and the number of events in the neighboring position.

In order to test if an increased number of high branching values is caused by a dependency between the degree of the vertices in the partitions, i.e. if vertices of high degree tend to connect to vertices of higher degree in the other partition, the Pearson's *r* test was used [95]. Table 5-3 shows the results of the correlation between some neighboring partitions in both datasets. The collection of journal papers in the Anthrax field presents a medium positive correlation between the partitions tests (Authors - Publications), while the earthquake dataset presents only a very low correlation. The correlation between non-neighboring partitions is not that straight-forward to calculate, because it has to consider possible repeated connections

as new samples for the calculation of the correlation coefficient. However, as the neighboring partitions already show positive correlation, it is safe to assume that the second level correlations will be positive too. Thus, this increases the number of high branching paths, as observed in the empirical distributions.

Table 5-3 - Pearson's r correlation coefficient between partitions for Anthrax and earthquake datasets

Source	Pearson's *r*
Anthrax – between Author (wrote) and Publication (citedBy)	0.352
Earthquake – between Location (hasEvent) and Location (closeBy)	0.032

This correlation characteristic of the databases can be employed for improving the sampling efficiency. *If one vertex was sampled and seem to contain a high feature values, its neighboring vertices, i.e., vertices that share common intermediary vertices, also will have the tendency to have high feature values*. The opposite is also valid: *vertices with low feature values will have neighbors that have low feature values*. Based on these observations, an evolutionary algorithm approach was devised to make use of this heuristic and will be presented next.

5.6. Evolutionary Algorithm Approach for Sampling

The Evolutionary Algorithm approach chosen is aimed at solving the following problem: how to define a sampling policy so that the elements with higher feature values have a better approximation faster. In order to solve this problem, it is necessary to deal with two conflicting goals: identifying new potentially highly connected vertices and sampling the vertices that are already known to be well connected to quickly approximate their feature values. In an Evolutionary Algorithm method there are basically three main operations [96], combination (or crossover), mutation, and selection. However, for sampling, there are no suitable combination methods, because there is no meaning on a general combination of the information of two vertices. In place of the combination that is supposed to preserve the information of important vertices, a local mutation was defined. The overall algorithm is represented in the block diagram in Figure 5-12. The main blocks will be explained in separate below.

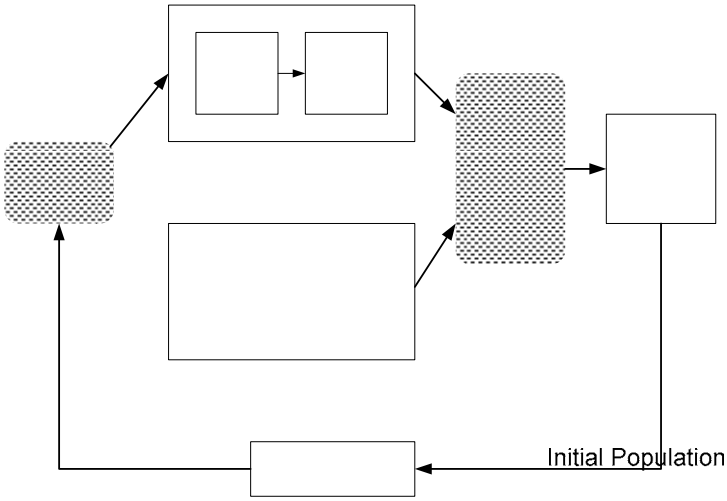


Figure 5-12 - Block diagram of EA-based sampling policy

The genotype of each individual is the name of an input vertex that is going to be the initial point for sampling. Because it is assumed that there is no knowledge about the goal of the feature values, the fitness function is not related to these values. The fitness is related to the sampling policy, trying to foster the exploration of unknown vertices to improve the overall chances of having obtained the complete traversal of all paths. In numeric terms, the fitness value for individual i, $f(i)$ is defined as

$$f(i) = \frac{1}{n(v(i))}, \tag{5.7}$$

where $n(j)$ is the number of times the vertex j was sampled, and $v(i)$ is the vertex represented by the individual i.

5.6.1. Local mutation

The goal of the *local mutation* is to *exploit* the areas that are known to have high feature values, in order to obtain a better approximation of the feature values. This process uses the observed fact that there usually is a positive correlation between the degrees in the different partitions in some datasets, thus it is possible to predict high feature values through all the partitions by just observing the feature values of the vertices that

are connected in the initial partitions. For example, in a journal paper dataset, when trying to identify the most important authors, if it is known that an author is important, there is a high chance that this author's co-authors are important too.

Therefore, the proposed algorithm for the *local mutation* does obtain neighboring vertices by exploiting the positively correlated relations. The pseudo-code for the algorithm can be seen in Table 5-4. The output of this algorithm, like the output of a common mutation, is a new individual that is inserted in the population. The first important step of this process is determining which elements should be mutated locally. This is simply solved by using a *roulette wheel* to select based on their feature values, i.e., the higher the feature value is, the more likely the vertex is to be selected for *local mutation*. Besides this step, there are two important parameters to be set.

The first one is the locality definition, defining which walk semantics should be used to obtain the positively correlated neighbor. For example, in the earthquake dataset the most natural neighbor to a position is the geographical neighbor defined by the property "closeBy." However, when defining a neighbor of a paper author, it is defined as a sequence of a property ("wrote"), a vertex type ("Publication") and then another property ("writtenBy").

Table 5-4 – Pseudo-code for Local Mutation

```
function mutate_local(from_element)
        1. distance_left = d
        2. loop
            a. Calculate all possible neighbors in the subgraph counting the number
                of instances of the walk to the neighbor
            b. Choose a neighbor randomly
            c. if the distance to the neighbor (dist) is less than the distance
                left
                i. distance_left = distance_left – dist
                ii. Move to neighbor
            d. else, get out of the loop
3. return the current element
```

The second parameter is the maximum distance that this neighbor can be located to still be defined as "local." This is also highly dependent on the application. For an earthquake dataset, maybe more than two transitions through the "closeBy" property may lead you to a very distant position (more than 10^o from the initial position). At the same time, for authors, the co-author of a co-author of a co-author may still have

some meaning, because it usually means the analysis is still in the same field, and probably in the same research group. Moreover, sometimes it may be interesting to weight the connection based on the inverse structural feature, e.g., when identifying important authors, the co-authors that published many papers with this author may sometimes be considered "closer" to the author. For the examples chosen, this assumption was always considered, and the distance was the inverse of the structural feature value for the connection between the two elements:

$$d\left(i,j\right)=\frac{1}{f_{local}\left(i,j\right)}, \tag{5.8}$$

where $d(i,j)$ is the distance between vertices i and j. $f_{local}(i,j)$ is the feature following the path defined as local for the algorithm. In other words, the neighborhood can be seen as the process of weighted graph collapsing defined in Definition 9, Chapter 2, where the weights are the inverse of the feature value. Finding vertices that are within a distance d from a vertex v is equivalent to a weighted walk in this weighted collapsed graph such that the total weight of the walk is less than d.

5.6.2. Global mutation

The global mutation process has the objective of weighting towards *exploration* of unknown elements. It is simply implemented by randomly selecting a vertex from the database and adding it to the population.

5.6.3. Selection

After this new modified population was added to the old population, a selection process has to take place to define which individuals represent the vertices that should be sampled. This selection is based on the fitness function discussed above, using the *roulette wheel* approach for selecting the final population, thus having the probability of preserving some highly sampled vertices at the same time focusing on sampling unknown regions.

Next section presents some experimental results of the proposed method, in comparison to the naïve sampling presented above.

5.7. Sample Experimental Results

Before presenting the experimental results, a method for rating the performance has to be proposed.

5.7.1. Error functions

Error determination is vital to any experimental results, because they offer quantitative information of the amount of improvement that the method offers. However because of the lack of research on the approximation of structural features of databases, no standard error functions were ever developed. There are three possible types of error functions that can be of interest: absolute feature value magnitude error, global ranking error, and top ranking error.

5.7.1.1. **Absolute feature value magnitude error.** This error measure is based on the expected feature value and the approximated feature value. It can be defined as the squared error

$$E_a = \sum_{i=1}^{N} \left(f(i) - \hat{f}(i) \right)^2 , \tag{5.9}$$

where N is the number of elements to which the feature is calculated (for example, if a feature is calculated for all possible pairs of input and output vertices, then N is the product of the number of input and output vertices); $f(i)$ is the actual feature value for the element (or element group) i; and $\hat{f}(i)$ is the approximated value of the feature.

It is the most natural error measures defined, but suffers from very important drawbacks. First, it is a global error measure, thus it penalizes the error in approximating features values that are too low to even be of interest. Secondly, it does assume that the feature value itself is of importance. However, due to the highly skewed degree distributions observed in the databases, as discussed in Section 5.5, the amplitude of the feature value is usually not as important as the actual rank that this feature receives compared to the same feature in other vertices. Having this interest in ranks in mind, the next two error measure types were proposed.

5.7.1.2. **Global ranking error.** The global ranking error comes from comparing the ranking values of the approximated features compared to the expected feature ranks. Inspired by nonparametric distance measures [97], the following error measure is proposed.

$$E_{KG} = \sum_{i=1, X(i)>0}^{N} \left(R_T(i) - R_S(i) \right)^2 \tag{5.10}$$

where R_T is the mid-ranking value following the actual feature values, while R_S is the ranking of the approximated feature values.

This method does take into consideration that usually it is interesting to use the ranking as the actual feature than the numeric feature observed. However, the problem with using ranking for highly skewed distributions is that there are a very large number of ties, sometimes close to 50% of the values are tied. The mid-ranks procedure prescribes the use of the average value of the ranks that would be assigned to the tied values if they were not tied [95]. For example, when ranking three elements, the two lowest valued ones are tied, both receive a rank value of 2.5 (the average between 2 and 3). Therefore, if there are, for example 2,000 elements with the same feature value, but one is incorrect, this makes the ranking value of all of them to be incorrect and off by 0.5. Thus, very small errors caused by sampling, especially on the elements with very low ranking, can generate large ranking errors.

However, it can be argued that these elements with very low feature values are not of interest in any processing, so their error should not be considered. With this in mind, a third error measure was defined, the top ranking error.

5.7.1.3. **Top ranking error.** If the interest in the analysis is only to identify the top ranked elements for the given feature when calculating the approximation error, it is not reasonable to consider the ranking of all the elements. Therefore, one new error measures were proposed where only the ranking of the first L elements is considered.

$$E_T^L = \sum_{i=1}^{L} R_T\left(X_S^{(i)}\right) \tag{5.11}$$

where $X_S^{(i)}$ is the i^{th} ranked element according to the approximated feature values.

This error measure does not present the problems indicated above for the other global error measures. However, it is important to note that for some applications, not only the L highest valued features are important; therefore, all three proposed error measures are used to rate the sampling algorithms in the application examples presented next.

5.7.2. Sampling collections of journal papers

As identified before in Table 5-3, the collection of journal papers in anthrax research contains moderate positive correlation between the author and the number of times cited. Therefore it is safe to expect that the proposed EA-S algorithm is able to efficiently identify the most important authors by analyzing the author bibliographic coupling and using co-authorship as the neighborhood (see Section 7.1 for a description of each of these walk semantics). In order to obtain statistically significant results, each of the sampling methods were applied 20 times and the average result and the standard deviation are presented below.

For both methods, 10^7 samples were taken. In the N-S approach, 10^5 choices were made and in which choice, the chosen element was sampled 100 times. While in the EA-S approach, 1,000 generation were simulated, with population size of 100 individuals per generation and 100 samples for each individual in each generation. 10 new individuals are chosen based on the local mutation, and 10 individuals are chosen using the global mutation. Table 5-5 shows the average error measures and standard deviation for this experiment. The distance used was chosen empirically to be $d = 1$, because it provided the best results.

Table 5-5 - Comparison of the N-S and EA-S for the anthrax database

Algorithm	$\overline{E_a}$	σ_a	$\overline{E_{KG}}$	σ_{KG}	$\overline{E_T^{20}}$	σ_T^{20}
N-S	1.812×10^8	2.170×10^7	6.110×10^9	7.063×10^8	8.433×10^4	6.041×10^3
EA-S	1.884×10^8	2.062×10^7	9.619×10^9	6.331×10^8	5.948×10^4	1.957×10^3

As expected, the EA-S method does obtain a lower error in the top-20 rank error, but does offer a higher error for the errors based on all the features.

5.7.3. Sampling earthquake event databases

As seen in Table 5-3, the earthquake dataset offers a much lower correlation between partitions. In this case, one of the assumptions made in the EA-S approach is not kept. However, it was decided to use the proposed algorithm in this application in order to understand the effect of the violation of this assumption. The feature used was the same as the previous chapter, of identification of important earthquake regions. Again in this case the algorithms ran for 20 times in order to obtain statistically-sound results.

Because the branching factor of the elements in this dataset for the given walk semantics is lower than the one in the previous example, only 5×10^5 samples were used. In the N-S algorithm, 50,000 choices were made and each chosen element was sampled 10 times. While in the EA-S approach, 500 generations were used, with population size of 100 individuals and 10 samples for each individual per generation. 10 new individuals are chosen based on the local mutation, and 10 individuals are chosen using the global mutation. The results are shown in Table 5-6. The distance used was $d = 1$, also chosen empirically.

Table 5-6 - Comparison of the N-S and EA-S for the anthrax database

Algorithm	$\overline{E_a}$	σ_a	$\overline{E_{KG}}$	σ_{KG}	$\overline{E_T^{20}}$	σ_T^{20}
N-S	8.368×10^{7}	2.112×10^{6}	8.244×10^{13}	4.153×10^{12}	1,434.5	94.5
EA-S	1.902×10^{8}	9.873×10^{6}	9.955×10^{13}	6.133×10^{12}	6,443.2	247.1

As it can be seen, the EA-S algorithm did not perform very well when one of its assumptions is not met. When analyzing the behavior of the sampling to understand the reason for this difference, it was observed that the EA-S ends up having the tendency to over-sample some elements with very low branching factor but are neighbors from elements with high feature values. Because of the neighborhood being so sparse in this dataset, comparing to the previous dataset, and the number of global mutated individuals being low, there are a large number of high branching elements that never get to be sampled enough, causing this large error.

5.8. Summary

This chapter presents two algorithms for performing approximate feature extraction based on sampling. This process is vital when dealing with large databases and when there are not enough resources to store the whole database in memory for feature extraction.

The first algorithm, the naïve sampling algorithm, is asymptotically optimal when trying to obtain the feature values for all the elements throughout the database. It is very simple to implement, but requires a large number of samples to obtain good approximations of the feature values. The second algorithm, the evolutionary algorithm-based sampling is efficient in obtaining faster approximations to the feature values of the elements with the highest feature values as long as there is a way to define a neighborhood in which elements with high feature values are neighbors. This is the case when analyzing important authors in a collection of journal papers, by using author bibliographic coupling as the feature of interest and co-authorship as the neighborhood. However, when trying to employ the same algorithm to the earthquake dataset for determining active earthquake areas and using geographic proximity as the neighborhood, this

correlation is not high enough. Therefore, the EA-S method tends not to give results that are as good as the N-S.

The following chapter discusses implementation of the graph-structured database and the algorithms used, as well as some methods that can be used to increase efficiency of the algorithm without using approximations or requiring user intervention.

CHAPTER 6

Implementation Considerations

6.1. Introduction

When implementing graph-structured databases, the first challenge comes from the lack of off-the-shelf solutions for these kinds of databases. There exists a lack of implementations of efficient algorithms and data storage methods for tackling problems that require the use of these kinds of databases despite the reasonable amount of research that exist in this field [16]. Not only there is no available implementation as there is very little discussion in the scientific literature on what operations these databases should implement to be usable to the community, or API (Application Program Interface). An interesting exception is the creation of the RDF API by researchers at Stanford, later revised into Jena, a Java API for RDF, developed by Brian McBride at HP Labs [98]. However, the scope of this implementation is the representation of the RDF structure and not the efficient use for processing.

This short chapter contains information about the implementation of the underlying graph-structured database, as well as choices about importing and exporting elements from an ontology. The objective of this chapter is to fill a gap in the literature about graph-structured databases by proposing an API for dealing these kinds of databases for enabling feature extraction and pattern recognition. This proposed API, named GSDB-API, contains the basic functions that have to be implemented according to the operations presented in this report. This chapter also presents an analysis of the efficiency of implementing this API employing a relational database as the method for increasing search efficiency, enabling persistence and alleviating memory requirements. Finally, it discusses some simple implementation choices to improve the efficiency of the overall algorithm by observing sub-walk-semantics within the walk semantics of interest.

Like the discussion in Chapter 2, the methods proposed here do not intend to cover all possible operations on graph-structured databases and ontologies. Their scope is on the implementation of the algorithms analyzed in this research. Further work would be necessary to implement graph searching capabilities and more specific ontology information to the database, such as cardinality restrictions. Consistency check on the database was not implemented either, because it is assumed that this will be done in a higher level when building the database (the consistency restrictions on the relational database do not offer much help for coding the restrictions on the graph-structured database).

6.2. Graph-Structured Database API Definition

Prior to presenting the API, it is important to list the assumptions made when developing it. However, in order to understand the assumptions, it is important to note that in this chapter the word "ontology" represents the structure of the graph-structured database, with the vertex and edge types and their relations (for this implementation, only three relations were considered of interest: vertex and edge type inheritance, edge domain and range restrictions, and semantic inverse-of relation). While the word "database" represents vertices and edges that are instances of one or more of these types. The word "database collection" is the group of all databases and ontologies that are stored.

- The database collection can store more than one database and more than one ontology. These databases and ontologies have to be represented by a unique name.
- Each database has to reference to at least one ontology in which the element types are defined.
- In order to enable the use of more than one ontology for each database, each element contains a prefix that define the namespace in which the element type is defined. This follows the same concept of the XML namespace definition.
- An ontology may also use elements from another ontology, especially when defining domains and ranges for edge types.
- A database can import data from another database and the system should not replicate the data in both databases when this happens to facilitate the support for changes in the database.

- Vertices and edges can have multiple inheritances.
- In the database level there is no need to check for consistency of the ontology restrictions. However, it assumes that it is kept throughout the database.

With these assumptions, seven main elements were identified: a general database engine (OntologyEngine), an encapsulator for each ontology (Ontology), an encapsulator for each database (GraphData) a vertex and edge type elements (VertexType and EdgeType, respectively), and a vertex and edge elements (Vertex and Edge, respectively). A simplified UML (Unified Modeling Language) [99] diagram of these elements and their composition relations can be seen in Figure 6-1.

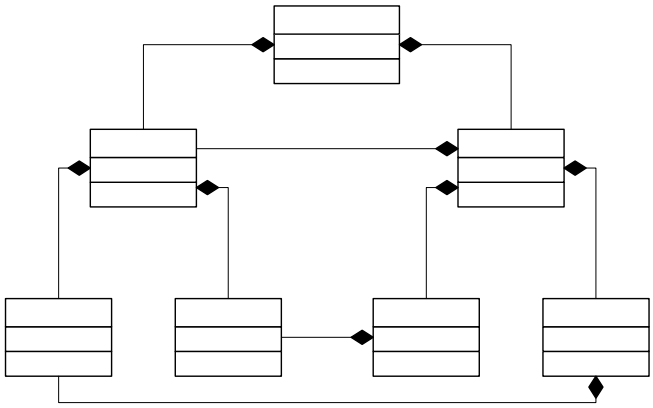


Figure 6-1 - Simplified UML view of the main elements of the GSDB-API

Below each element will be discussed in more details separately. The API specification with all the functions that enable this API for the feature extraction is included in the Appendix II.

- **OntologyEngine:** This element serves to control all the ontologies and databases registered. It can parse external ontology files into new ontologies and databases. In the current implementation, only OWL format is supported, but the API does not give any specific constraint to which format is loaded, as long as it is contained in a file.
- **Ontology:** This element encapsulates an ontology providing access, creation and query of vertex and edge types, as well as imported ontologies.

1 Onto

*

*

1 Ontology

1

- **GraphData:** This element encapsulates a database offering access, creation and query to the vertices and edges, queries on elements of given type, and importing ontologies and other databases. It also provides methods for performing the projection operation given a walk semantics, and the calculation of a feature matrix given the walk semantics.
- **VertexType:** Provides the information about a vertex type, such as its parents and children, as well as ability to modify this information.
- **EdgeType:** With this element it is possible to define and modify an edge type, its parents and children, inverse functional edge, ranges and domains.
- **Vertex:** This element is the most basic component of a database, representing an object. It enables setting the vertex types. It also provides methods for obtaining all the outgoing or incoming edges of a specific type.
- **Edge:** Finally this element deals with abstracting the edge element, providing functions to set source and target vertices, as well as types.

As mentioned above, one of the functions that is implemented in a GraphData is of projection. The projection generates another GraphData that contains only the elements in walks that agree with the given walk semantics. Optimally this projection GraphData is made in such a way that obtaining the feature values (the ultimate goal for performing the projection) is efficient. If the resulting projected database is small enough, it would be even interesting to keep it in the memory for very quick processing.

Another important element is the result of the calculation of the features. It is stored in a container EWTable that efficiently saves and load the element groups. It can also quickly provide a part of the database based on the ranks for quick visualization of the results. In order to enable its use for the selection procedures in the EA-based algorithm explained in Chapter 5, it also needs to obtain random element groups based on the feature values (roulette wheel selection).

With this simple API it is possible to perform all the processing proposed above. In order to store the information collected above, as there is no off-the-shelf graph-structured database engine, a relational database was chosen. In the next section details about the tables used in this database will be provided.

6.3. Implementation Employing Relational Database Infrastructure

Relational databases have been used in real-world large-scale applications for many years. Therefore, the technology behind these databases is fairly mature, providing very efficient storage and quick retrieval of elements. The proposed system makes use of this maturity and, thus, the implementation of the graph-structured database is done using relational databases. This is done by defining a series of tables for each ontology and database in order to store all the values effectively. These tables will be briefly shown below in Figures 6-2 to 6-4 organized based on the basic elements presented in the previous section.

The database structure presented in this section is self-explanatory. It is interesting to note that there is a different set of tables for each database and ontology in order to support a number of independent datasets. The names of the tables contain the ID of the ontology or database defined by the OntologyEngine.

When using this structure, one major deficiency was observed, the need to always perform multiple selection operations to obtain the type of elements. As it is assumed that vertices and edges may present multiple inheritances, it is necessary to separate the definition of these elements and the definition of their types to minimize redundancy and database space use. For example, in order to obtain all out-edges of type "t" of a vertex "a" from database "1," it is necessary to perform the following selection operation in the database:

```
SELECT ToVertex FROM edges_1 INNER JOIN edge_type_table_1
                    ON edges_1.name = edge_type_table_1.name
       WHERE edges_1.FromVertex = 'a' AND
             edge_type_table_1.EdgeTypeName = 't';
```
(6.1)

Join operations are usually very expensive to realize in databases and should be avoided if possible. However, there is no known way to avoid it in this case. The only method that can be used to mitigate this efficiency problem is not to use a relational database. However, as mentioned before, this is not feasible with the current level of implementation of graph-structured databases.

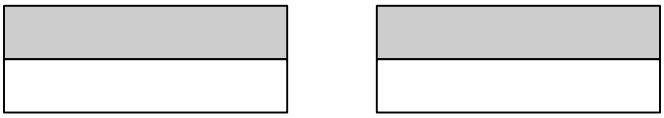


Figure 6-2 - Database tables for the ontology engine

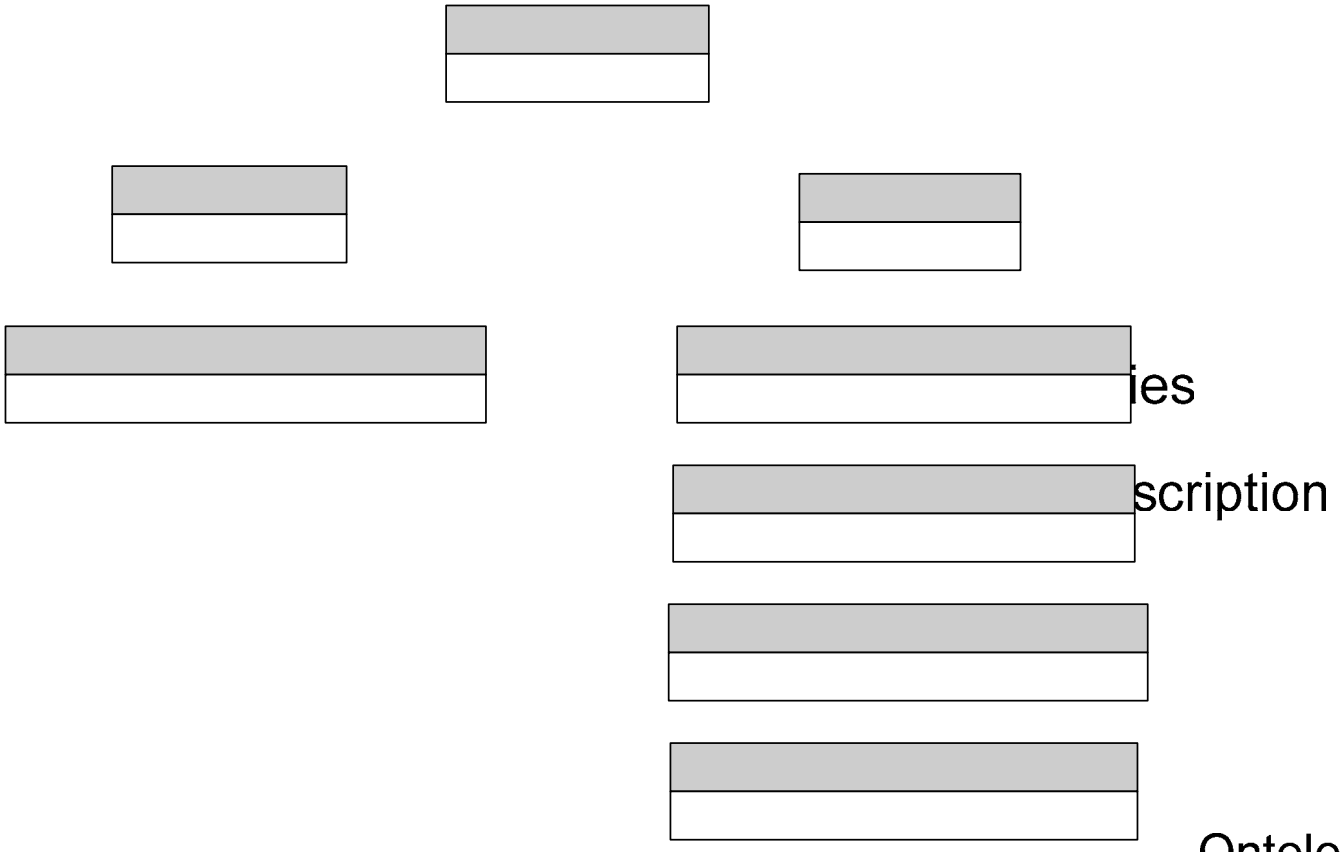


Figure 6-3 - Definition of tables for storing the ontology. The <ID> is the ontology ID from the Ontologies table

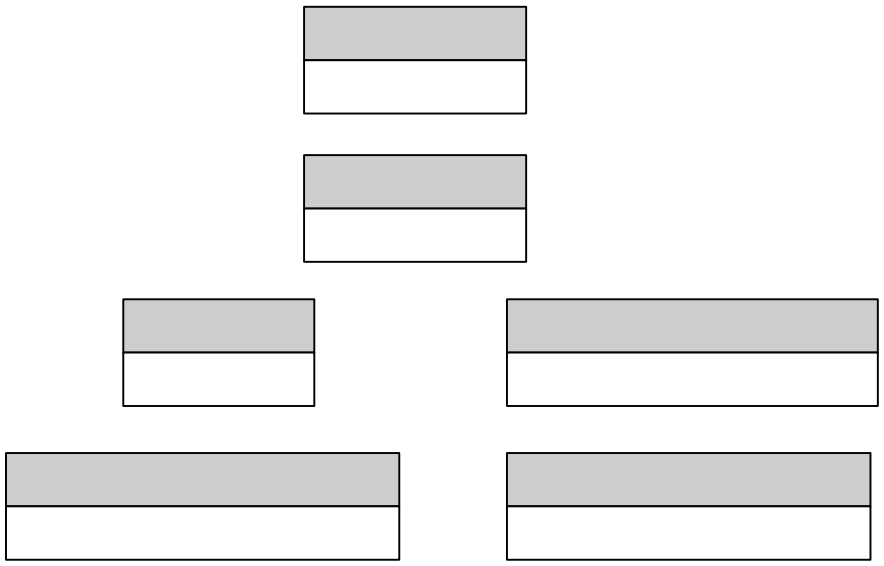


Figure 6-4 - Database tables for storing graph-structured databases. The <ID> is the database ID from the Databases table

6.4. Increasing Efficiency by Identification of Sub-Walk-Semantics

In the construction of the table of features, the software requires to obtain all the possible ~~equi~~semantic walks between a certain set of elements. It has been discussed before that this process is highly memory and time-intensive, even when using the sampling method discussed in Chapter 5. When implementing the method a very simple heuristic was observed that assisted in decreasing, sometimes greatly, the amount of resources necessary for obtaining the feature values. Given the abstract projection given in Figure 6-5, it is easy to see that the features between two vertices v_{Ai} and v_{Dj}, $f_{AD}(i,j)$ can be defined as:

$$f_{AD}(i,j) = \sum_{m=i,n=j} f_{AB}(m,p) \cdot f_{BC}(p,q) \cdot f_{CD}(q,n), \qquad (6.2)$$

where $f_{AB}(m,n)$ is the feature value between the vertices of type A and B (subprojection 1), and the same is valid for the other subprojections.

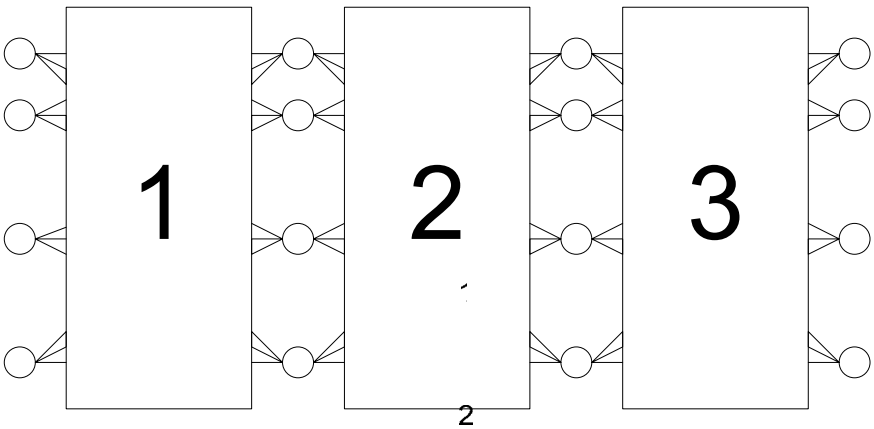


Figure 6-5 - Abstract representation of a projection divided into three subprojections

This result apparently does not assist in decreasing the processing expense for the algorithm. However, given the real-world example of a walk semantics of interest in Figure 6-6, and its subprojection representation, it is interesting to observe that subprojections 1 and 2 are the same. In this case, calculating the features of only one of them is sufficient for later determining what the feature of the whole projection is. Applications for this walk semantics will be presented in Section 7.2.

In order to implement this simplification, an important process is required: the identification of optimal sets of subprojections. It is important to note that the calculations presented in Equation (6.2) do require extra computation, but it is usually lighter than the computation required if there is a need of obtaining the whole projection. There are many algorithms in the literature for efficiently obtaining frequent subsequences [100], however, this algorithms are fine-tuned to deal with very large data strings. It is a safe assumption that all walk semantics of interest are reasonably short, thus enabling the use of simple brute-force-type algorithm. The shortest sub-walk has the length of 2, because a sub-walk of length 1 is a single edge and does not require any additional calculation. The pseudo-code for the algorithm used is given in Table 6-1.

After the largest recurrent sub-walks are isolated, the system calculates the features for each of these sub-walks and then composes the results. This is all done transparently for the user. Processing time and memory use improvements of 10-40% were observed in databases in which there are walk semantics of interest that can be decomposed.

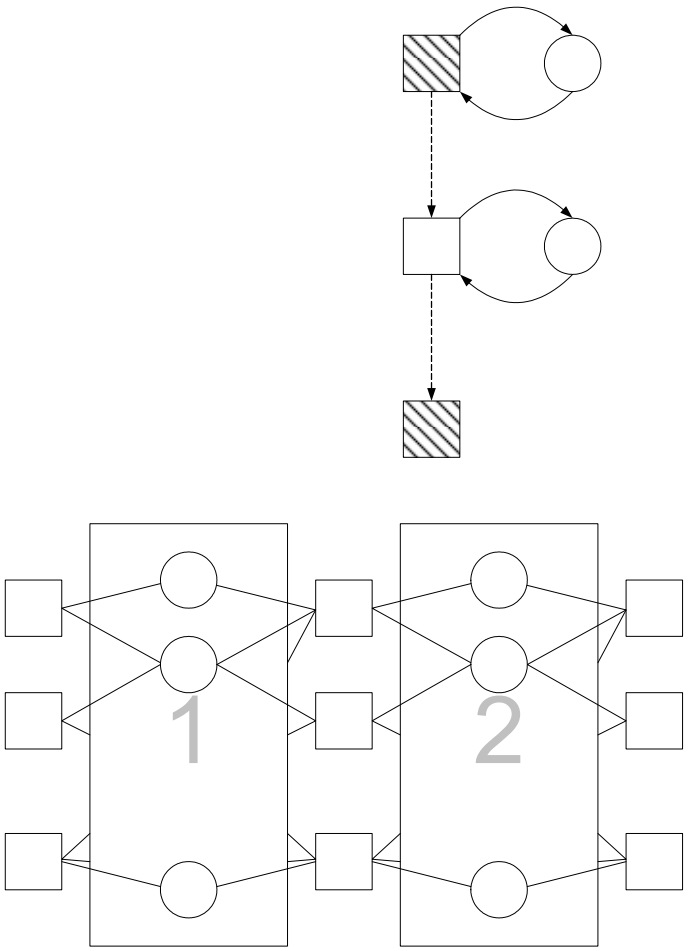


Figure 6-6 - Walk semantics and projection for identifying earthquake "hot spots"

Table 6-1 - Pseudo-code for identification of sub-walks within a walk semantics

```
Max_length = length(walk_semantics)/2
Terms = Find all elements that appear at least twice in the walk semantics
For each of the Terms
        Search forward for same terms in at least two of the occurrences
        If there are same terms for at least 2 occurrences with length at least 2
                If these sequences do not overlap
                        Save sequences in Candidates
                End if
        End if
Next
If some candidates overlap
        Choose the largest candidate, or randomly select one of them and delete the other
              candidates
End if
Sub-walks = candidates
Augments sub-walks by the elements that are missing to complete the walk semantics
```

CHAPTER 7

Application Examples

7.1. Introduction

This chapter will present examples of larger-scale applications of the algorithms proposed in this document. These examples have the goal of providing further support that structural features are useful for the identification of patterns in graph-structured databases.

Two examples will be shown in two different databases. The first example is on applying structural feature extraction to dealing with the problem of author name disambiguation in collections of journal papers. The second example is on identifying hot spots for earthquakes in an earthquake database. While the first example is more challenging in understanding the features involved and finding the patterns, the second example uses a much larger database and will make use of the sampling algorithms proposed in Chapter 5 to provide the scalability necessary for solving this problem.

7.2. Author Disambiguation in Collections of Journal Papers

Name disambiguation has been one of the grand challenges in bibliometric research [101-103]. Its applicability extends beyond just journal papers. Some examples of other applications of name disambiguation are geographical and historical analysis [104], text mining [105], criminal analysis, and speech recognition.

In bibliometric research, the main methods for name disambiguation use keyword information to obtain information that could suggest that authors are the same. However, in many datasets, some papers do not contain keyword information. Other information in publication databases can also be used for author disambiguation, such as citation information (authors tend to reference the same papers), co-authorship

(authors tend to publish with the same authors) and co-journal (authors tend to publish in the same journals). One important observation that has to be made while trying to disambiguate two authors is that if there is a paper that both these authors appear co-authoring a paper, it is enough evidence to support that these authors are not the same. Finally, name problems with authors in publication databases are usually caused by the omission of middle initials, misspelling, and lack of standard when dealing with compound family names.

With this information, the pattern recognition based on examples can be applied for disambiguating authors using the following algorithm:

1. Examples of the same authors are created artificially;
2. The distance from each pair of authors to the examples is calculated based on the following walk semantics with manually assigned weights (in parenthesis). The explanation of each walk semantics is given in Figure 7-1:
    a. Co-authorship (100): two co-authors cannot be the same.
    b. Double co-authorship (1)
    c. Cites (1)
    d. Is cited by (1)
    e. Bibliographic coupling (1)
    f. Co-citation (1)
    g. Co-keyword on title (1)
    h. Co-source (1)
3. The top 1% are extracted and ranked based on name similarity removing redundant pairs (from author A to B and B to A).
4. The top matches are presented to the user based on word similarity threshold.

Applying this algorithm to the Anthrax dataset using six artificially created examples (authors with high number of papers were selected and one paper at random from each had the name of the author modified so

that it was considered a different vertex) and adding 2 test datasets created in the same way, the following results in Table 7-1 were obtained. The word similarity measure used was the number of matching pairs of letters between the names.

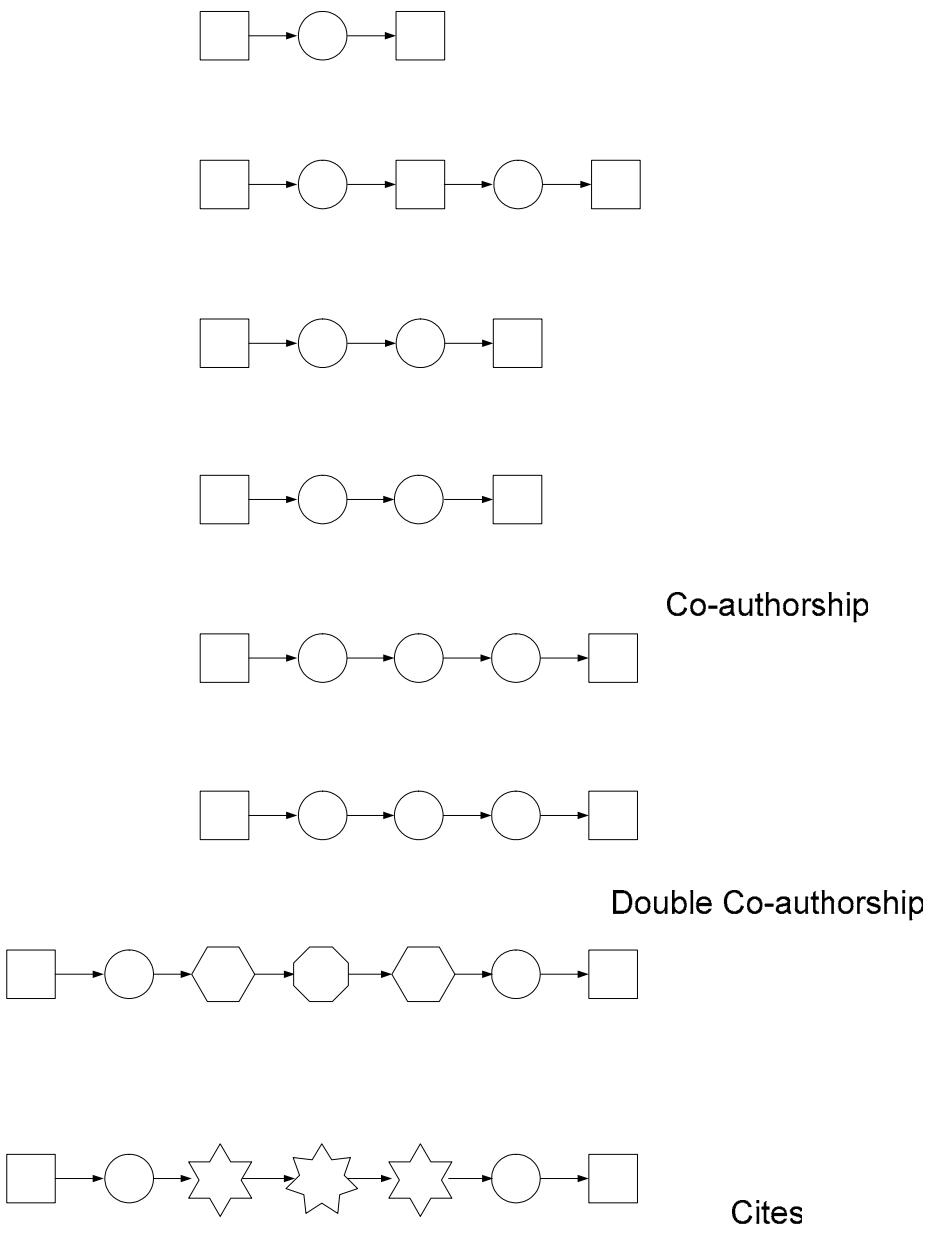


Figure 7-1 - Definitions of walk semantics used for author disambiguation

The Anthrax dataset contains 12,337 publications obtained from the ISI database with 20,701 citations (all citations are to the publications in the database – most publications do not contain citation information) in the Anthrax research field. The ontology of this database is shown in Appendix I-2. It contains a total of 33,826 vertices, not counting the numerical vertices (such as year value and journal volume and issue) and text vertices (such as author and journal name, keywords and publication title). These are not considered in this analysis. Keywords could be interesting, however most publications in the database do not contain this information, thus it only increases complexity without a significant increase in the information.

Table 7-1 - Results from author disambiguation

Rank	Authors	
1	Friedlander AM	Friedlander A
2	Farchaus JW	Farchaus J
3	Fellows PF	Fellows P
4	Jackson P	Jackson PJ
5	Little SF	Little S
6	Welkos S	Welkos SL
7	Eitzen E	Eitzen EM
8	Leppla SH	Leppla S
9	Gibbs P	Gibbs PH
10	Hanna PC	Hanna P
11	Ivins BE	Ivins B
12	Tosi-Couture E	TosiCouture E
13	Shlyakhov EN	Shlyakhov EN 2
14	Brachman PS	Brachman PS 2
15	Shlyakhov EN 2	Shlyakhov E
16	DALLDORF FG	DALLDORF FE
17	Baillie L	Baillie LWJ
18	HUGHJONES M	Hugh-Jones ME
19	Singh S	Singh Y
20	Woude GFV	Vande Woude GF
21	Hanna PC	Khanna H
22	BRACHAM PS	BRACHMAN PS
23	Pitt L	Pitt MLM
24	WEBER M	Weber-Levy M
25	Perkins B	Ivins BE

On ranks 13, 14 and 15 the test values were identified. It is interesting to note that this algorithm was able to find matches on all three common mistakes. The most common mistake is of absence of middle initial. But changes in spelling standard were also detected, such as in 18, 20 and 22. However, when the restrictions in word matching start to become looser, some matching errors start to appear, such as in ranks

19, 21 and 25. These errors can be easily observed by quick visual inspection of the results. They could have been avoided by a stricter definition of the word matching criteria, but this would lead to longer processing time, and possible loss of some of the mistake types.

7.3. Identification of Earthquake Hot Spots

Performing a similar analysis to the one made in Chapter 5, it is possible to obtain earthquake "hot spots" by analyzing the walk semantics given in Figure 7-2. As it is natural to assume that the result of this algorithm should provide a single location, only the cases in which the initial and final positions are the same are considered.

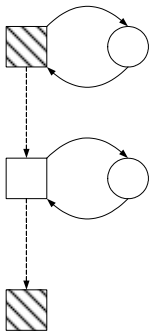


Figure 7-2 - Walk semantics for identification of earthquake "hot spots"

Two possible results can be taken from this analysis. The first one is a tabular result that presents all the ranked earthquake locations. Although this method is very precise, it does not offer a natural visualization for non-experts in earthquake analysis. The tabular result was presented in Table 7-2, where the top 10 earthquake positions are shown. However, when plotting the positions on top of a world map, it becomes easier to see and interpret the results. Figure 7-3 shows an example of this map in which all top 1,000 earthquake events are marked with red dots. Figure 7-4 shows the same type of analysis but focusing on the West Pacific islands. With this increase in magnification it is possible to plot the circles to define the regions proportional to the feature values obtained. This helps to identify possible patterns within the top 1,000 earthquakes. For example, it is possible to see that, large earthquakes (the red large circles) appear spread around the areas of high earthquake incidence, not only centered in a single spot.

Location, locationC
inLocation, L

Location, locationC

Table 7-2 - Top 10 tabular result for earthquake "hot spots"

Rank	Latitude	Longitude	Place	Feature value
1	-5	153	Papua New Guinea	806,796
2	-5	152	Papua New Guinea	572,155
3	-18	-179	Fiji	510,072
4	-15	167	Vanuatu	509,418
5	-18	-178	Fiji	507,872
6	-21	-179	Fiji	481,614
7	1	126	Indonesia	467,544
8	44	149	Japan	406,617
9	-21	-174	Fiji	371,656
10	-6	155	Papua New Guinea	367,086

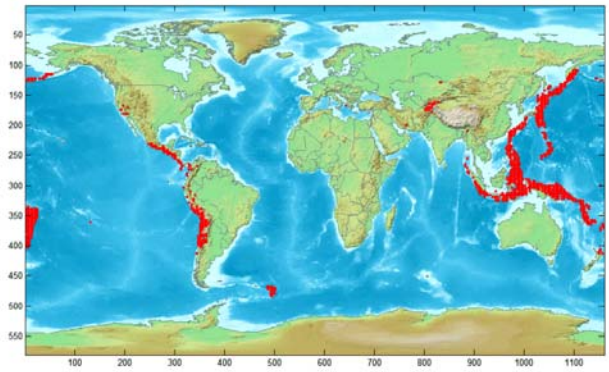


Figure 7-3 - Map of all top 1000 earthquake locations in the world

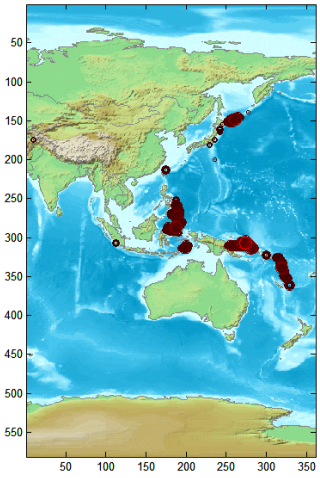


Figure 7-4 - Map of top earthquake locations in the West Pacific highlighting most active regions within the region

CHAPTER 8

Conclusions

This document presents novel methods for analyzing graph-structured databases by defining features based on the structure of the graph, rather than by the conditional existence of certain connections as previous studies have presented. The algorithm is centered on the concept that information in a database is directly related to a certain semantic walk in the data structure of this database. By defining the feature of interest using a semantic walk, it is possible to calculate the feature values for each group of vertices in the database. After calculating the feature values, it was shown that pattern recognition methods can be employed to identify patterns in the database, either defined as explicit set of walk semantics, by presenting examples of interest, or by a combination of both.

Following the same framework of information related to a semantic walk, this report presented a method for performing transformation of the structure of a database. This transformation is essential when employing third-party or legacy databases in which structures were defined aiming other specific applications or ease of storage and retrieval in relational databases. Having a database structure that relates more to the application is essential for the generation of meaningful walk semantics that, in turn, provide more powerful features for pattern recognition. Another interesting result presented from the implementation of the transformation framework is the ability to improve the efficiency of the feature extraction algorithm by applying transformations that decrease the length of the walk semantics without significant information loss.

Scalability is an important issue on graph-structured databases. This document presents two methods for performing scalable feature extraction by using feature approximation by sampling. This is a novel way of pursuing scalability where in most other applications it was achieved by requiring a large number of parallel processing. The methods offered in this document have the benefit of enabling the exchange of computer resources for accuracy. If more computer power is available, very precise results can be obtained,

while if very quick and rough results are only needed, the algorithm can present still usable findings. By employing an evolutionary algorithm approach incorporated with heuristics for efficiently searching, it is possible to obtain even better approximations on elements with high feature values, as long as there is a method to define positively correlated neighborhood to the elements being sampled.

Another contribution presented in this document is an API for dealing with feature extraction in graph-structured databases. Although this API is not complete for all possible applications in graph-structured databases, it is a very important starting point for the wide-spread uses of graph-structured databases in pattern recognition. Implementation also provided a third method for improving the efficiency of the feature extraction by observing recurrence of sub-walks in semantic walks. This improvement, like the one presented by transformation, can be implemented easily without user intervention, simplifying the experience the user has with the system.

Two major application examples were presented in this document to further support the concepts proposed. The first one is on author disambiguation, a very important step when dealing with very noisy collections of journal papers. Without author disambiguation, some artifacts can be created in the analysis. Simple binary value presence analysis, like the ones proposed in the literature, is impossible to obtain good features for this task. The use of structural features has improved the quality of these features enough to make this task solvable. The second application was on the identification of earthquake “hot spots” or locations with high earthquake activity. This analysis cannot be performed easily without taking into consideration the structure of the graph-structured database alone. But with a single structural feature, it is possible to easily observe with a large number of details in these problematic areas.

A very large number of other applications can be envisioned that would benefit from the extraction of structural features. Among them are:

- market basket analysis for extraction of consumption patterns;
- analysis of networks of movie actors for identification of groups and actor preferences;

- terrorist network identification for homeland security;
- analysis of complex interactions of large molecular databases;
- authorship identification in text databases;
- identification of possible interaction between events in newspaper articles; and
- analysis of the subtle connections between stock market behavior and news stories.

Also a very large number of possible improvements can be made to the proposed algorithm by further study of the following aspects not covered in this research:

- **Definition of a systematic method for identifying walk semantics of interest** – although most of the times the walk semantics are easy to identify, it would be interesting to present a method for systematically analyzing the requirements and identifying which walk semantics may represent better these requirements.
- **Dealing with numeric-element-related features** – the current algorithm is tuned to categorical elements, thus it is not able to implicitly deal with relations formed from numerical connections. For example, one of the most common numerical feature in databases is time. The current algorithm can only accept time as another entity, but in some cases it is necessary to deal with time as a special entity. For example, when augmenting the analysis of important authors to authors with important current publications, it is necessary to add some effect from time in the definition of the weights.
- **Analysis of improvement of efficiency by parallelism –** the use of parallel algorithms has been shown to be very efficient in dealing with graph-structured databases. If the proposed algorithm can further improve its scalability by using this feature, it can effectively increase the applicability of this algorithm for very large databases.
- **Explicit inclusion and testing of syntactic features** – although the addition of syntactic features was proposed in this document, no examples were found in which their use is applicable. Only after implementation it is possible to understand and possibly engage further analysis of defining weight policies.

- **Analysis of the effects of dimensionality reduction** – Like the inclusion of syntactic features, only by effectively applying the dimensionality reduction it is possible to understand its effects. Because of the correlation between some of the walk semantics used, it is expected that it would be possible to perform a very large dimensionality reduction if it would be necessary. However, it is possible that some non-linear effects may hinder the applicability of the method in some applications.
- **Inclusion of explicit features in performing the pattern recognition** – in this document, all the features of interest were structural. Sometimes it may be interesting to combine structural and explicit features, such as the social contacts of a person and this person's physical characteristics. Mixing features of very different natures can be very challenging for pattern recognition, but will ultimately improve its performance greatly.
- **Implementation of the large-scale pattern recognition system –** as presented in Section 1.4, the fundamental goal for this research is to obtain part of the features that would be later used in a large-scale pattern recognition system. In order to implement this system, a number of other elements have to be realized, such as ontology extraction and user interface.

In summary, the results obtained in the research documented here are an indication that the use of structural features is feasible and vital for pattern recognition in graph-structured databases. The preliminary results on test datasets have shown that the features are easy to identify, understand and powerful enough to enable the recognition of some even complex patterns in these datasets. This study is just a preliminary step towards a more general and wide-reaching pattern recognition system. Its successful implementation will increase in many ways the applicability of graph-structured databases in the real-world, supporting the current trends of employing this structure because of its good representation power and understandability properties.

References

[1] S. Wasserman and K. Faust, *Social network analysis: methods and applications*. Cambridge, NY, USA: Cambridge University Press, 1994.

[2] V. Gopalkrishnam, Q. Li, and K. Karlapalem, "Issues of object-relational view design in data warehousing environment," proceedings of the 1998 IEEE International Conference on Systems, Man, and Cybernetics, 1998.

[3] M. Uschold and M. Gruninger, "Ontologies: principles, methods and applications," *Knowledge Engineering Review*, vol. 11, pp. 93-136, 1996.

[4] T. R. Gruber, "A translation approach to portable ontology specifications," *Knowledge Acquisition*, vol. 5, pp. 199-220, 1993.

[5] Y. Ding and S. Foo, "Ontology research and development. Part 1 - A review of ontology generation," *Jounal of Information Science*, vol. 28, pp. 123-136, 2002.

[6] T. Berners-Lee, J. Hendler, and O. Lassila, "The Semantic Web - A new form of Web content that is meaningful to computers will unleash a revolution of new possibilities," *Scientific American*, vol. 284, pp. 34-43, 2001.

[7] D. L. McGuinness and F. van Harmelen, "OWL Web ontology language overview," http://www.w3.org/TR/owl-features/, 2004.

[8] D. Beckett, "RDF/XML Syntax Specification (Revised)," http://www.w3c.org/TR/rdf-syntax-grammar/, 2004.

[9] D. Brickley and R. V. Guha, "RDF Vocabulary Description Language 1.0: RDF Schema," 2004.

[10] S. Bechhofer, C. Goble, and I. Horrocks, "DAML+OIL is not enough," proceedings of the Semantic Web working symposium, Stanford, CA, USA, 2001.

[11] A. R. Bobak, *Data modeling and design for today's architectures*. Boston, MA, USA: Artech House, 1997.

[12] A. Thakar, A. Szalay, P. Kunzt, and J. Gray, "Migrating a multiterabyte archive from object to relational databases," *Computing in Science & Engineering*, vol. 5, pp. 16-29, 2003.

[13] M. Atkinson, F. Bancilhon, D. DeWitt, K. Dittrich, D. Maier, and S. Zdonik, "The object-oriented database system manifesto," proceedings of the 1st International Conference on Deductive and Object-Oriented Databases, Kyoto, Japan, 1989.

[14] M. Gyssens, J. Paredaens, J. Van den Bussche, and D. Van Gucht, "A graph-oriented object database model," *IEEE Transactions on Knowledge and Data Engineering*, vol. 6, pp. 572-586, 1990.

[15] R. Elmasri and S. Navathe, *Fundamentals of Database Systems*, 4th ed. Boston, MA, USA: Addison Wesley, 2003.

[16] D. Shasha, J. T. L. Wang, and R. Giugno, "Algorithmics and applications of tree and graph searching," proceedings of the Symposium on Principles of Database Systems, Madison, WI, USA, 2002.

[17] M. P. Consens and A. O. Mendelzon, "Graphlog: a visual formalism for real life recursion," proceedings of the Ninth ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems, Nashville, TN, USA, 1990.

[18] J. Paredaens, P. Peelman, and L. Tanca, "G-Log - a graph-based query language," *IEEE Transactions on Knowledge and Data Engineering*, vol. 7, pp. 436-453, 1995.

[19] L. Cardelli, P. Gardner, and G. Ghelli, "A spatial logic for querying graphs," *Automata, Languages and Programming*, vol. 2380, pp. 597-610, 2002.

[20] L. Cardelli and A. Gordon, "Anytime, anywhere: Modal logics for mobile ambients," proceedings of the 27th ACM SIGPLAN-SIGACT Symposium on Principloes of Programming Languages, Boston, MA, USA, 2000.

[21] L. Caires, "A model for declarative programming and specification with concurrency and mobility," in *Demartamento de Informatica, FCT*. Lisbon, Portugal: Universidade Nova de Lisboa, 1999.

[22] J. D. Ullman and J. D. Widom, *A First Course in Database Systems*, 2nd. ed. Upper Saddle River, NJ, USA: Prentice Hall, 2001.

[23] K. Fukunaga, *Introduction to statistical pattern recognition*, 2nd ed. Boston, MA, USA: Academic Press, 1990.

[24] T. F. Quatieri, *Discrete-Time Speech Signal Processing*. Upper Saddle River, NJ, USA: Prentice-Hall, 2002.

[25] Y. Xu, J.-Y. Yang, and Z. Jin, "A novel method for Fisher discriminant analysis," *Pattern Recognition*, vol. 37, pp. 381-384, 2004.

[26] L. Cherny, "Gender differences in text-based virtual reality," proceedings of the Berkeley Conference on Women and Language, Berkeley, CA, USA, 1994.

[27] R. Bellman, *Adaptive Control Processes: A Guided Tour*. Princeton, NJ, USA: Princeton Univ. Press, 1961.

[28] P. Meesad and G. G. Yen, "Pattern classification by neurofuzzy network: application to vibration monitoring," *ISA Transactions*, vol. 39, pp. 293-308, 2000.

[29] R. C. Gonzalez and M. G. Thomason, *Syntatic Pattern Recongnition*. Reading, Massachusetts, USA: Addison-Wesley Publishing, 1978.

[30] G. Caldarelli, "Introduction to complex networks," proceedings of the 7th Conference on Statistical and Computational Physics, Granada, Spain, 2002.

[31] N. L. Johnson, S. Kotz, and A. W. Kemp, *Univariate discrete distributions*, 2nd ed. New York: John Wiley & Sons, 1992.

[32] R. Albert, H. Jeong, and A. L. Barabasi, "Internet - Diameter of the World-Wide Web," *Nature*, vol. 401, pp. 130-131, 1999.

[33] H. Jeong, B. Tombor, R. Albert, Z. N. Oltval, and A. L. Barabasi, "The large-scale organization of metabolic networks," *Nature*, vol. 407, pp. 651-654, 2000.

[34] M. Faloutsos, P. Faloutsos, and C. Faloutsos, "On power-law relationships of Internet topology," *Computer Communication Review*, vol. 29, pp. 251-262, 1999.

[35] S. Redner, "How popular is your paper? An empirical study of the citation distribution," *European Physical Journal B*, vol. 4, pp. 131-134, 1998.

[36] F. Liljeros, C. R. Edling, L. A. N. Amaral, H. E. Stanley, and Y. Aberg, "The web of human sexual contacts," *Nature*, vol. 411, pp. 907-908, 2001.

[37] M. L. Goldstein, S. A. Morris, and G. G. Yen, "Problems fitting to the power-law distribution," *(submitted)*, 2004.

[38] A.-L. Barabási, R. Albert, and H. Jeong, "Mean-field theory for scale-free random networks," *Physica A*, vol. 272, pp. 173-187, 1999.

[39] G. U. Yule, "A methematical theory of evolution based on the conclusions of Dr. J. C. Willis, F. R. S.," *Philosophical Transactions of the Royal Society of London, Series B*, vol. 213, pp. 21-87, 1924.

[40] S. N. Dorogovtsev and J. F. F. Mendes, "Evolution of networks," *Advances in Physics*, vol. 51, pp. 1079-1187, 2002.

[41] R. Albert and A.-L. Barabási, "Statistical mechanics of complex networks," *Reviews of Modern Physics*, vol. 74, pp. 47-97, 2002.

[42] M. E. J. Newman, "The structure and function of complex networks," *SIAM Review*, vol. 45, pp. 157-256, 2003.

[43] O. Corcho and A. Gómez-Pérez, "A roadmap to ontology specification languages," proceedings of the 12th International Conference on Knowledge Engineering and Knowledge Management, 2000.

[44] Y. Ding and D. Fensel, "Ontology library systems: the key to successful ontology re-use," proceedings of the First Semantic Web Working Symposium, Stanford University, CA, USA, 2001.

[45] J.-U. Kietz, A. Maedche, and R. Volz, "A method for semi-automatic ontology acquisition from a corporate intranet," proceedings of the EKAW'2000 Workshop Ontologies and Texts, 2000.

[46] A. Maedche and S. Staab, "Ontology learning for the Semantic Web," *IEEE Intelligent Systems & Their Applications*, vol. 16, pp. 72-79, 2001.

[47] Y. Ding and S. Foo, "Ontology research and development. Part 2 - A review of ontology mapping and evolving," *Journal of Information Science*, vol. 28, pp. 375-388, 2002.

[48] D. W. Embley, D. M. Campbell, R. D. Smith, and S. W. Liddle, "Ontology-based extraction and structuring of information from data-rich unstructured documents," proceedings of the International Conference on Information and Knowledge Management, Bethesda, Maryland, 1998.

[49] A. Gal, G. Modica, and H. Jamil, "Improving Web search with automatic ontology matching," *submitted*, 2003.

[50] N. F. Noy and M. A. Musen, "SMART: Automated support for ontology merging and alignment," proceedings of the 12th Workshop on Knowledge Acquisition, Modeling and Management, Banff, Canada, 1999.

[51] N. Kushmerick, "Wrapper induction: Efficiency and expressiveness," *Artificial Intelligence*, vol. 118, pp. 15-68, 2000.

[52] R. Kohavi and G. H. John, "Wrappers for feature subset selection," *Artificial Intelligence*, vol. 97, pp. 273-324, 1997.

[53] M. Craven, D. DiPasquo, D. Freitag, A. McCallum, T. Mitchell, K. Nigam, and S. Slattery, "Learning to construct knowledge bases from the World Wide Web," *Artificial Intelligence*, vol. 118, pp. 69-113, 1999.

[54] W. May and G. Lausen, "Information extraction from the Web," Institut für Informatik Albert-Ludwigs-Universität, Freiburg, Germany, Technical Report 136, March 2000.

[55] Y.-S. Chang, M.-H. Ho, and S.-M. Yuan, "A unified interface for integrating information retrieval," *Computer Standards & Interfaces*, vol. 23, pp. 325-340, 2001.

[56] D. Faure and C. Nedellec, "A corpus-based conceptual clustering method for verb frames and ontology acquisition," proceedings of the LREC-98 Workshop on Adapting Lexical and Corpus Resources to Sublanguage and Applications, Paris, France, 1998.

[57] F. Esposito, S. Ferilli, N. Fanizzi, and G. Semararo, "Learning from parsed sentences with INTHELEX," proceedings of the Learning Language in Logic Workshop, New Brunswick, NJ, USA, 2000.

[58] A. Maedche and S. Staab, "Discovering conceptual relations from text," proceedings of the 14th European Conference on Artificial Intelligence, Amsterdam, 2000.

[59] E. Morin, "Automatic acquisition of semantic relations between terms from technical corpora," proceedings of the 5th International Congress on Terminology and Knowledge Engineering, Vienna, Austria, 1999.

[60] U. Hahn and K. Schnattinger, "Towards text knowledge engineering," proceedings of the 15th National Conference on Artificial Intelligence, Menlo Park, CA, USA, 1998.

[61] R. Diestel, *Graph Theory*, 2nd ed. New York: Springer, 2000.

[62] T. Kamada and S. Kawai, "An algorithm for drawing general undirected graphs," *Information Processing Letters*, vol. 31, pp. 7-15, 1989.

[63] S. A. Morris, Z. Wu, and G. G. Yen, "A SOM mapping technique for visualizing documents in a database," proceedings of the IEEE International Joint Conference on Neural Networks, Washington, DC, USA, 2001.

[64] S. A. Morris, G. Yen, Z. Wu, and B. Asnake, "Time line visualization of research fronts," *Journal of the American Society for Information Science and Technology*, vol. 54, pp. 413-422, 2003.

[65] D. Auber and M. Delest, "A clustering algorithm for huge trees," *Advances in Applied Mathematics*, vol. 31, pp. 46-60, 2003.

[66] R. W. Schvaneveldt, D. W. Dearholt, and F. T. Durso, "Graph theoretic foundations of Pathfinder networks," *Computers & Mathematics with Applications*, vol. 15, pp. 337-345, 1988.

[67] G. J. Szabo, M. Alava, and J. Kertesz, "Geometry of minimum spanning trees on scale-free networks," *Physica A: Statistical Mechanics and its Applications*, vol. 330, pp. 31-36, 2003.

[68] L. Eikvil, "Information extraction from the World Wide Web - A survey," Norwegian Computer Center, Technical Report 945, 1999.

[69] D. E. Chubin, "The conceptualizaton of scientific specialties," *The Sociological Quarterly*, vol. 7, pp. 448-476, 1976.

[70] H. D. White and K. W. Mccain, "Bibliometrics," *Annual Review of Information Science and Technology*, vol. 24, pp. 119-186, 1989.

[71] H. Alani, S. Dasmahapatra, K. O'Hara, and N. R. Shadbolt, "Idenifying communities of practice through ontology network analysis," *IEEE Intelligent Systems*, vol. 18, pp. 18-25, 2003.

[72] M. Girvan and M. E. J. Newman, "Community structure in social and biological networks," *Proceedings of the National Academy of Sciences of the United States of America*, vol. 99, pp. 7821-7826, 2002.

[73] D. E. Johnson, *Applied Multivariate Methods for Data Analysis*. Pacific Grove, CA, USA: Brooks/Cole Publishing, 1998.

[74] W. R. Dillon and M. Goldstein, *Multivariate Analysis: Methods and Applications*. New York, NY, USA: John Wiley & Sons, 1984.

[75] B. Schölkopf, A. Smola, and K.-R. Müller, "Nonlinear component analysis of kernel eingenvalue problems," *Neural Computation*, vol. 10, pp. 1299-1319, 1998.

[76] I. Borg and P. Groenen, *Modern Multidimensional Scaling: Theory and Applications*. New York, NY, USA: Springer, 1997.

[77] J. A. Hagenaars and A. L. McCutcheon, "Applied latent class analysis." Cambridge, U.K.: Cambridge Press, 2002.

[78] S. Deerwester, S. Dumais, G. Furnas, T. Landauer, and R. Harshman, "Indexing by latent semantic analysis," *Journal of the American Society for Information Science*, vol. 41, pp. 229-236, 1990.

[79] C.-M. Chen, N. Stoffel, M. Post, C. Basu, D. Bassu, and C. Behrens, "Telcordia LSI Engine: implementation and scalability issues," proceedings of the 11th International Workshop on Research Issues in Data Engineering, Heidelberg, Germany, 2001.

[80] C. Batini, M. Lenzerini, and S. B. Navathe, "A comparative-analysis of methodologies for database schema integration," *Computing Surveys*, vol. 18, pp. 323-364, 1986.

[81] M. Tresch and M. H. Scholl, "Schema transformation without database reorganization," *SIGMOD Record*, vol. 22, pp. 21-27, 1993.

[82] H. S. Pinto, A. Gómez-Pérez, and J. P. Martins, "Some issues on ontology integration," proceedings of the IJCAI-99 Workshop on Ontologies and Problem-Solving Methods: Lessons Learned and Future Trends, Stockholm, Sweden, 1999.

[83] A. Poulovassilis and P. McBrien, "A general formal framework for schema transformation," *Data & Knowledge Engineering*, vol. 28, pp. 47-71, 1998.

[84] D. J. Cook and L. B. Holder, "Graph-based data mining," *IEEE Intelligent Systems*, vol. 15, pp. 32-41, 2000.

[85] J. Gonzalez, I. Jonyer, L. B. Holder, and D. J. Cook, "Efficient Mining of Graph-Based Data," proceedings of the AAAI Workshop on Learning Statistical Models from Relational Data, Austin, TX, 2000.

[86] C. F. Richter, *Elementary Seismology*. San Francisco, CA, USA: W. H. Freeman, 1958.

[87] D. J. Cook and L. B. Holder, "Scalable discovery of informative structural concepts using domain knowledge," *IEEE Expert*, vol. 11, pp. 59-68, 1996.

[88] O. Frank, "Sampling and estimation in large social networks," *Social Networks*, vol. 1, pp. 91-101, 1978.

[89] E. Costenbader and T. W. Valentine, "The stability of centrality measures when networks are sampled," *Social Networks*, vol. 25, pp. 283-307, 2003.

[90] H. Toivonen, "Sampling large databases for association rules," proceedings of the Very Large Databases Conference, Mumbai, India, 1996.

[91] D. Madigan and M. Nason, "Data reduction: sampling," in *Handbook of data mining and knowledge discovery*. New York, NY, USA: Oxford University Press, 2002, pp. 205-208.

[92] J. S. Park, P. S. Yu, and M.-S. Chen, "Mining association rules with adjustable accuracy," proceedings of the 6th International Conference on Information and Knowledge Management, Las Vegas, NV, USA, 1997.

[93] M. L. Goldstein, S. A. Morris, and G. G. Yen, "Fitting to the power-law distribution," *cond-mat/0402322*, 2004.

[94] P. Erdös and A. Rényi, "On random graphs," *Publications Mathematicae*, vol. 6, pp. 290-297, 1959.

[95] W. J. Conover, *Practical Nonparametric Statistics*, 3rd ed. New York, NY, USA: Wiley, 1999.

[96] D. E. Goldberg, *Genetic Algorithms in Search, Optimization and Machine Learning*. Reading, MA, USA: Addison-Wesley, 1989.

[97] C. Dwork, R. Kumar, N. Naor, and D. Sivakumar, "Rank aggregation methods for the Web," proceedings of the 10th International World Wide Web Conference, Hong Kong, 2001.

[98] B. McBride, "Jena: Implementing the RDF model and syntax specification," proceedings of the 2nd International Semantic Web Workshop, Hong Kong, China, 2001.

[99] G. Booch, I. Jacobson, and J. Rumbaugh, *The Unified Modeling Language User Guide*. Reading, MA, USA: Addison-Wesley Professional, 1998.

[100] W.-K. Loh, S.-W. Kim, and K.-Y. Whang, "A subsequence matching algorithm that supports normalization transform in time-series databases," *Data Mining and Knowledge Discovery*, vol. 9, pp. 5-28, 2004.

[101] M. E. J. Newman, "Who is the best connected scientist? A study of scientific coauthorship networks," *Physics Review*, vol. 64, 2000.

[102] G. S. Mann and D. Yarowsky, "Unsupervised personal name disambiguation," proceedings of the 7th Conference on Natural Language Learning, Edmonton, Canada, 2003.

[103] V. I. Torvik, M. Weeber, D. R. Swanson, and N. R. Smalheiser, "A Probabilistic Similarity Metric for Medline Records: A Model for Author Name Disambiguation," *Submitted to JASIST*, 2003.

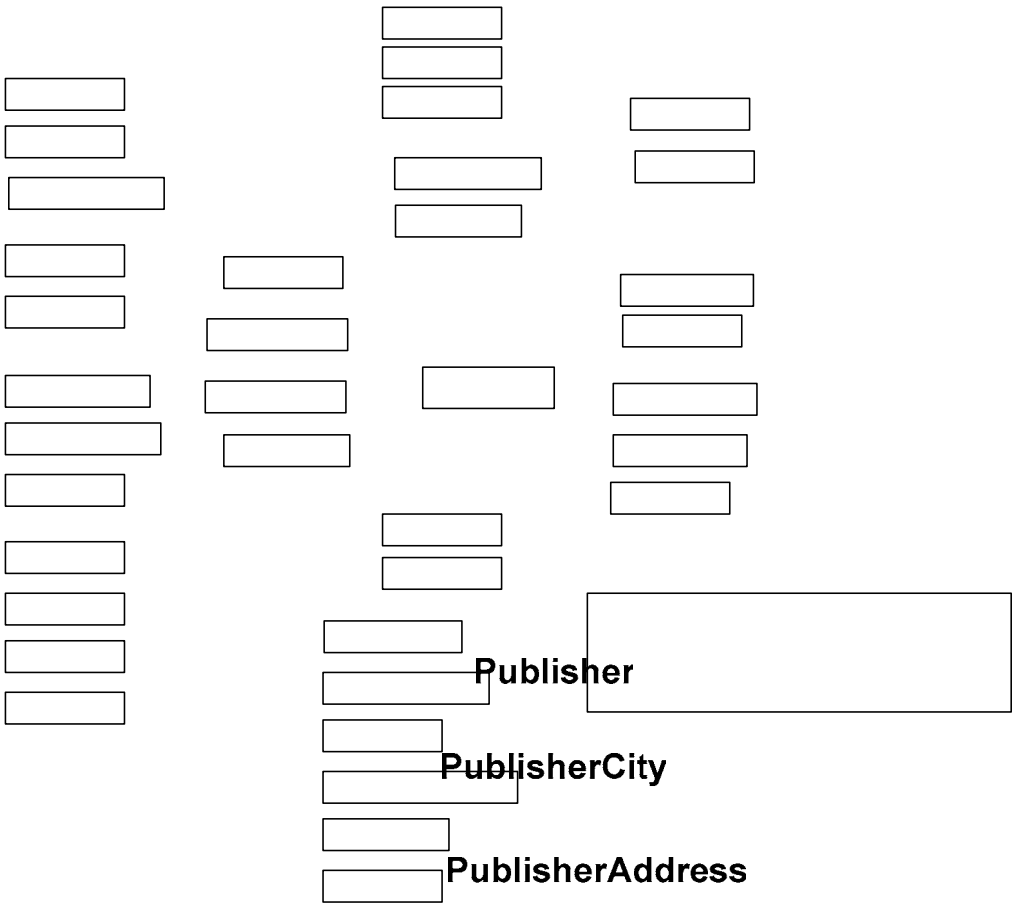
[104] D. A. Smith and G. Crane, "Disambiguation Geographic Names in a Historical Digital Library," proceedings of the 5th European Conference on Digital Libraries, Darmstadt, Germany, 2001.

[105] Z. Kazi and Y. Ravin, "Who's who? Identifying concepts and entities across multiple documents," proceedings of the 33rd Hawaii International Conference on System Sciences, Maui, Hawaii, 2000.

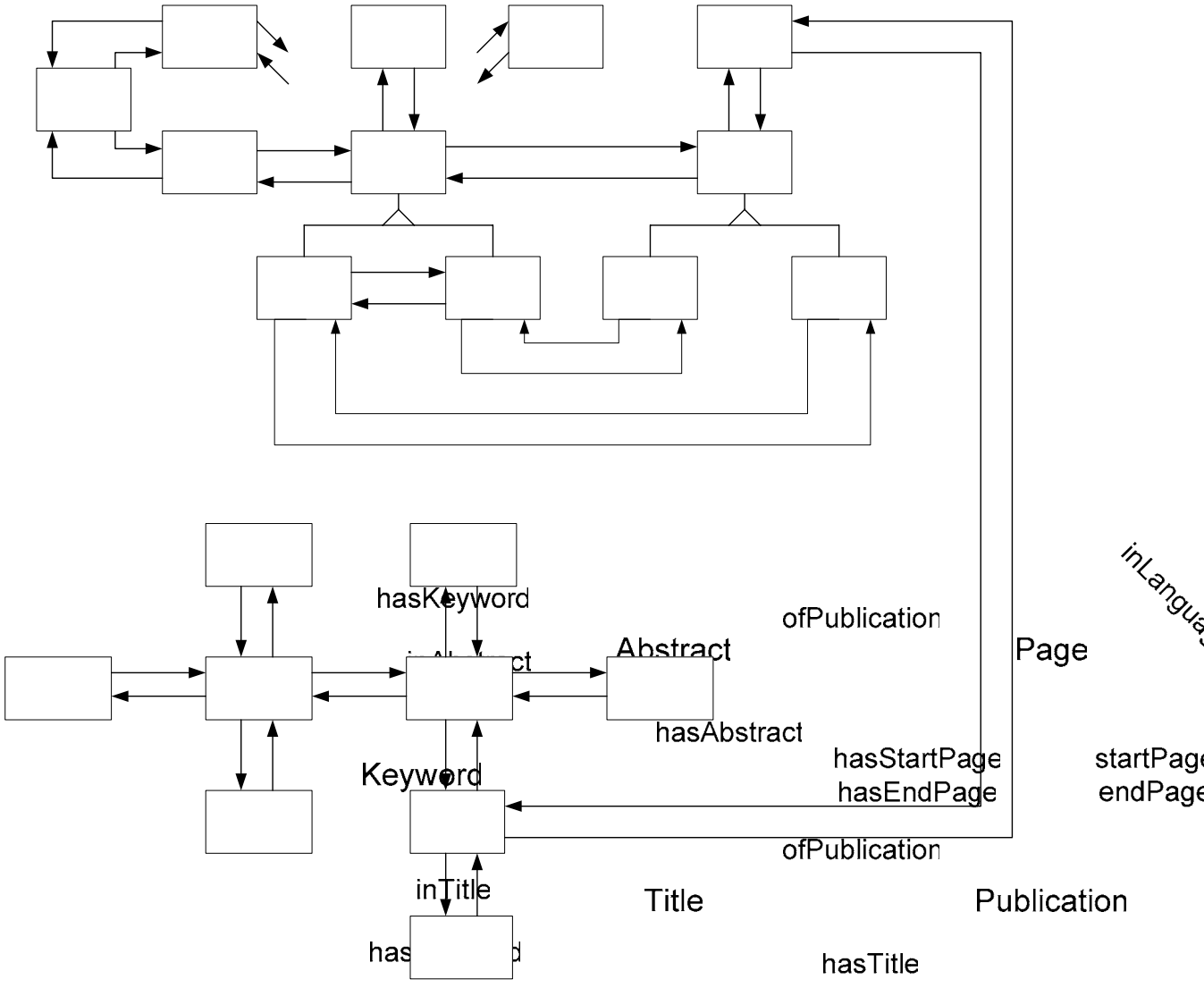
Appendix I

Ontologies Used

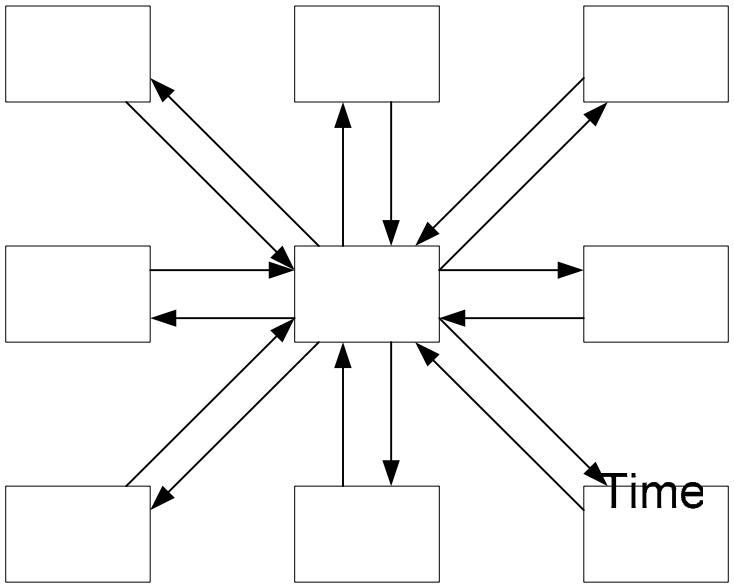
1. ISI Ontology for Collections of Journal Articles



2. Modified Ontology for Collections of Journal Articles



3. United States Geological Survey Ontology for Earthquake Events



inTime

hasEvent

hasEvent

Magnitude

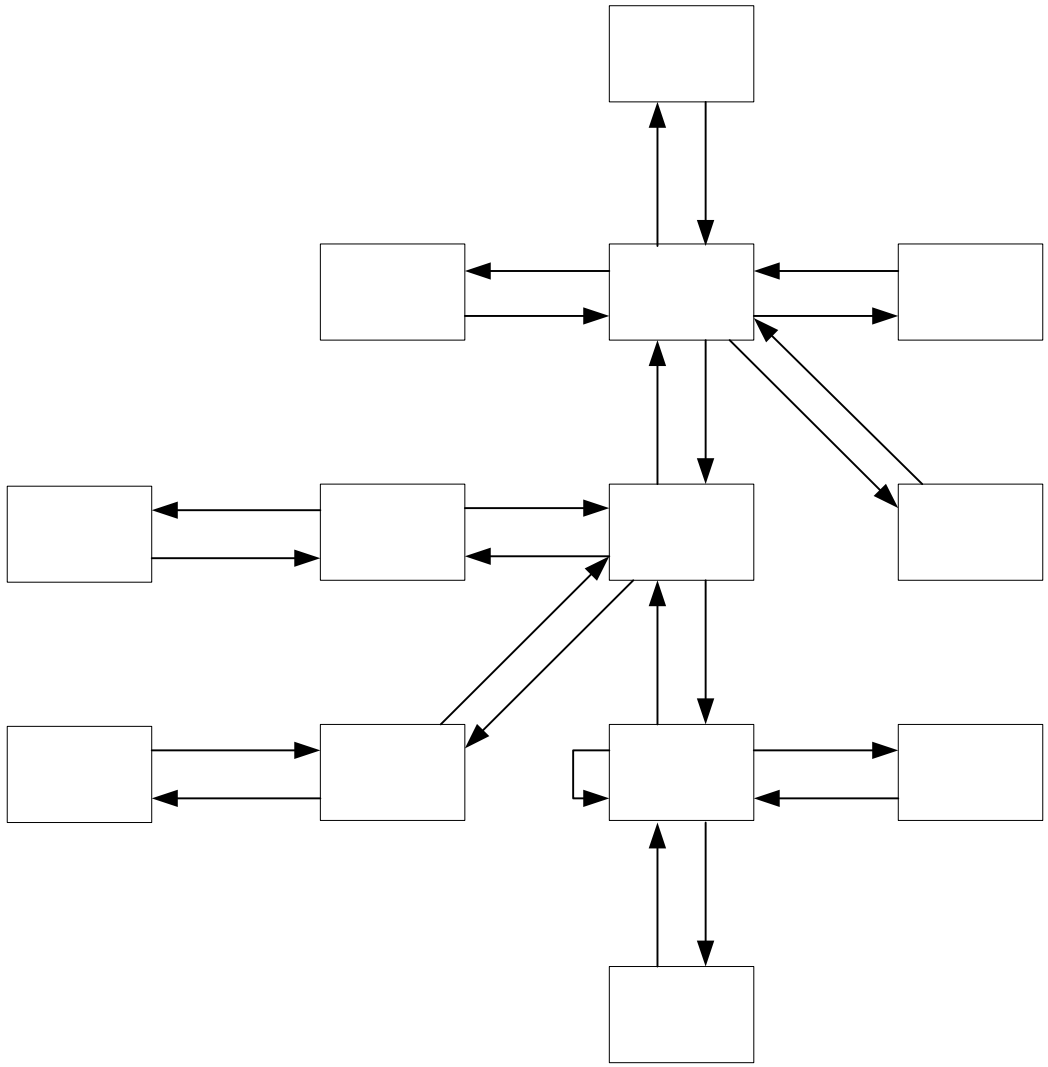
ofMagnitude

hasEvent

ofDepth

Depth

4. Modified Ontology for Earthquake Events



Time

isIn

Fuzzy Magnitude

Magnitude

hasMagnitude

Appendix II

API for Graph-Structured Database Analysis

OntologyEngine

```
/**
 * Tests if an ontology exists in the database
 * @param name The name of the ontology to test
 * @return True if the ontology exists
 */
public boolean existsOntology(String name);

/**
 * Gets the given ontology
 * @param name The name of the ontology
 * @return The ontology or null if the ontology does not exist
 */
public Ontology getOntology(String name);

/**
 * Create a new empty ontology
 * @param name The name of the ontology
 * @param description The description of the ontology
 * @return The ontology. It the name already exists it would return
 * null.
 */
public Ontology createOntology(String name, String description);

/**
 * Checks if a certain database exists in the database
 * @param name The name of the database that is being sought for
 * @return True if the database exists.
 */
public boolean existsDatabase(String name);

/**
 * Gets the ID of the database with given name
 * @param name The name of the database
 * @return The ID of the database. -1 if the database was not found.
 */
public int getDatabaseID(String name);

/**
 * Gets a database engine for the given database
 * @param name The name of the database
 * @return The database engine or null if the database does not exist
 */
public GraphData getDatabaseEngine(String name);

/**
 * Create a new database engine
 * @param name The name of the database
 * @param description The description of the database
 * @return The database engine for this new database, or null if the
 * database already exists.
 */
public GraphDate createDatabaseEngine(String name, String description);
```

Ontology

```
/**
 * Get the name of the ontology
 * @return The name of the ontology
 */
public String getName();

/**
 * Setter for the name of the ontology
 * @param newName The new name of the ontology
 */
```

```
public void setName(String newName);

/**
 * Get the description of the ontology
 * @return The description of the ontology
 */
public String getDescription();

/**
 * Setter for the description of the ontology
 * @param newDescription The new description of the ontology
 */
public void setDescription(String newDescription);

/**
 * Create a new vertex type
 * @param name The name of the type
 * @param description The description of the type
 * @return The VertexType interface implementation with the newly created
 * vertex type
 */
public VertexType createVertexType(String name, String description);

/**
 * Create a new edge type
 * @param name The name of the type
 * @param description The description of the type
 * @return The EdgeType interface implementation with the newly created
 * edge type
 */
public EdgeType createEdgeType(String name, String description);

/**
 * Remove a vertex type from the database
 * @param name The vertex type name
 */
public void deleteVertexType(String name);

/**
 * Remove an edge type from the databse
 * @param name The edge type name
 */
public void deleteEdgeType(String name);

/**
 * Checker for a vertex type
 * @param name The type being checked
 * @return True if the type exists in the ontology
 */
public boolean hasVertexType(String name);

/**
 * Checker for the edge type
 * @param name The type being checked
 * @return True if the type exists in the ontology
 */
public boolean hasEdgeType(String name);

/**
 * Getter for a vertex type from the ontology
 * @param name The name of the type
 * @return The vertex type interface implementation. Null if the type does
 * not exist.
 */
public VertexType getVertexType(String name);

/**
 * Getter for an edge type from the ontology
 * @param name The name of the type
 * @return The edge type interface implementation. Null if the type does
 * not exist.
```

```
     */
    public EdgeType getEdgeType(String name);

    /**
     * Get the full name of a vertex type. If the type is of the same
     * ontology, it just returns the name. If it is from one imported ontology,
     * it gives the alias and the name.
     * @param vt The vertex type
     * @return The name with the alias, when necessary
     */
    public String getFullName(VertexType vt);

    /**
     * Get the full name of a edge type. If the type is of the same
     * ontology, it just returns the name. If it is from one imported ontology,
     * it gives the alias and the name.
     * @param et The edge type
     * @return The name with the alias, when necessary
     */
    public String getFullName(EdgeType et);

    /**
     * Import an ontology, given the database name and its alias
     * @param name The database name
     * @param alias The alias given to the database
     */
    public void importOntology(String name, String alias);

    /**
     * Import a database given the ontology
     * @param onto
     * @param alias
     */
    public void importOntology(Ontology onto, String alias);

    /**
     * Getter for the ontology alias, given an ontology
     * @param ontologyName The name of the ontology (unique name in the
     * database)
     * @return The alias of the ontology in this database
     */
    public String getOntologyAlias(String ontologyName);
```

GraphData

```
    /**
     * Getter for the name of the database
     * @return The name of the database
     */
    public String getName();

    /**
     * Setter for the database name
     * @param newName The name of the new database (has to be unique)
     */
    public void setName(String newName);

    /**
     * Getter for the database description
     * @return The database description
     */
    public String getDescription();

    /**
     * Setter for the database description
     * @param newDescription The new database description
     */
    public void setDescription(String newDescription);

    /**
     * Method for importing a given ontology
```

```
 * @param name The name of the ontology to be imported
 * @param alias The alias to give to this ontology connection
 */
public void importOntology(String name, String alias);

/**
 * Method for importing a given ontology, given the ontology object
 * @param name The ontology object to be imported
 * @param alias The alias to give to this ontology connection
 */
public void importOntology(Ontology onto, String alias);

/**
 * Method for importing a database
 * @param name The database name
 * @param alias The alias to give to the database
 */
public void importDatabase(String name, String alias);

/**
 * Method for importing a database given the database object
 * @param name The database object
 * @param alias The alias to give to the database
 */
public void importDatabase(GraphData db, String alias);

/**
 * Creates a vertex in the ontology
 * @param name The name of the vertex – has to be an unique identifier
 * @param type The type of the vertex
 * @return The newly created vertex
 */
public Vertex createVertex(String name, VertexType type);

/**
 * Creates an edge in the ontology
 * @param name The name of the edge – has to be an unique identifier
 * @param type The type of the edge
 * @return The newly created edge
 */
public Edge createEdge(String name, EdgeType type);

/**
 * Getter for a given vertex
 * @param The name of the vertex being searched
 * @return The found vertex. Null if the vertex does not exist
 */
public Vertex getVertex(String name);

/**
 * Getter for a given edge
 * @param The name of the edge being searched
 * @return The found edge. Null if the edge does not exist
 */
public Edge getEdge(String name);

/**
 * Delete a given vertex from the database
 * @param name The name of the vertex to be deleted
 */
public void deleteVertex(String name);

/**
 * Delete a given edge from the database
 * @param name The name of the edge to be deleted
 */
public void deleteEdge(String name);

/**
 * Creates and gets a projection following the walk semantics
 * @param walkSemantics The walk semantics of the projection
```

```
 * @return A GraphData element that represents the projection
 */
public GraphData getProjection(String [] walkSemantics);

/**
 * Calculate the feature given a walk semantics and get the feature table. The
 * weight policy is, following the API, hard-coded.
 * @param walkSemantics The walk semantics
 * @return An instance of the EWTable with the values of the features
 */
public EWTable getEquisemanticWalkTable(String [] walkSemantics);

/**
 * Getter for the ontology alias, given an ontology
 * @param ontologyName The name of the ontology (unique name in the
 * database)
 * @return The alias of the ontology in this database
 */
public IOntologyEngine getOntology(String alias);

/**
 * Getter for an imported database
 * @param alias The database alias
 * @return An object encapsulating the desired database, or null if
 * the database was not found
 */
public GraphData getDatabase(String alias);

/**
 * Getter for the ontology alias, given a database
 * @param ontologyName The name of the ontology (unique name in the
 * database)
 * @return The alias of the ontology in this database
 */
public String getOntologyAlias(String ontologyName);

/**
 * Getter for the database alias, given the database
 * @param databaseName The name of the database (unique name in the
 * database)
 * @return The alias of the ontology in this database
 */
public String getDatabaseAlias(String databaseName);

/**
 * Getter for all edges to a certain vertex
 * @param vertex The vertex in question
 * @return A vector with all the edges
 */
public Vector getEdgesTo(Vertex vertex);

/**
 * Getter for all edges to a certain vertex of a given type
 * @param vertex The vertex in question
 * @param type The type of the edges being sought for
 * @return A vector with all the edges of the given type
 */
public Vector getEdgesTo(Vertex vertex, EdgeType type);

/**
 * Getter for all edges from a certain vertex
 * @param vertex The vertex in question
 * @return A vector with all the edges
 */
public Vector getEdgesFrom(Vertex vertex);

/**
 * Getter for all edges from a certain vertex of a given type
 * @param vertex The vertex in question
 * @param type The type of the edges being sought for
 * @return A vector with all the edges of the given type
```

```
     */
    public Vector getEdgesFrom(Vertex vertex, EdgeType type);

    /**
     * Get the full name of a vertex given the current aliases
     * @param vertex The vertex
     * @return The full name of the vertex
     */
    public String getFullName(Vertex vertex);

    /**
     * Get the full name of an edge given the current aliases
     * @param edge The edge
     * @return The full name of the edge
     */
    public String getFullName(Edge edge);

    /**
     * Get the full name of a vertex type given the current aliases
     * @param vt The vertex type
     * @return The full name of the vertex type
     */
    public String getFullName(VertexType vt);

    /**
     * Get the full name of a edge type given the current aliases
     * @param et The edge type
     * @return The full name of the edge type
     */
    public String getFullName(EdgeType et);

    /**
     * Get all the vertices of the given type
     * @param type The type of the vertices being sought for
     * @return A vector with all vertices of given type.
     */
    public Vector getVertices(VertexType type);

    /**
     * Get all the edges of a given type
     * @param type The type of the edges being sought for
     * @return A vector with all the edges of the given type
     */
    public Vector getEdges(EdgeType type);
```

VertexType

```
    /**
     * Gets the ontology name
     * @return The ontology name
     */
    public String getOntologyName();

    /**
     * Get all the direct parents of the given vertex type
     * @return The parents, a vector of VertexTypes.
     */
    public Vector getParents();

    /**
     * Get all the direct children of the given vertex type
     * @return The children, a vector of VertexTypes.
     */
    public Vector getChildren();

    /**
     * Add a parent to the current type
     * @param parent The parent vertex type
     */
    public void addParent(VertexType parent);
```

```
/**
 * Add a child type to the current type
 * @param child The child vertex
 */
public void addChild(VertexType child);

/**
 * Checks if a given vertex type is a parent of the current vertex type.
 * This funcion is recursive, so it checks not only the direct parents,
 * but the parents up to the highest level.
 * @param parent The parent being tested for
 * @return True if it is a parent.
 */
public boolean isParent(VertexType parent);

/**
 * Checks if a given vertex type is a child of the current vertex type.
 * This funcion is recursive, so it checks not only the direct children,
 * but the children up to the lowest level.
 * @param child The child being tested for
 * @return True if it is a child.
 */
public boolean isChild(VertexType child);

/**
 * Gets the name of the vertex type.
 * @return The name of the vertex type
 */
public String getName();

/**
 * Gets the description of the vertex type.
 * @return The description of  the vertex type.
 */
public String getDescription();

/**
 * Sets the name of the vertex type. Renames it if the name was already
 * known! However on has to be careful with this operation because it can
 * make some databases and ontologies inconsistent.
 * @param newName The new name of the vertex type
 */
public void setName(String newName);

/**
 * Sets the description of the vertex type. It changes the description if a
 * description already existed.
 * @param newDescription The new description for the vertex type.
 */
public void setDescription(String newDescription);

/**
 * Gets the ontology of the given type.
 * @return The ontology interface.
 */
public Ontology getOntology();

/**
 * Compares two vertex types to see if they are the same. Compare name and
 * ontology name.
 * @param vertexType The vertex type to compare
 * @return True if they are the same
 */
public boolean equals(VertexType vertexType);
```

EdgeType

```
/**
 * Get the name of the edge
 * @return The name of the edge type
 */
```

```
public String getName();

/**
 * Get the decription of the edge type
 * @return The description of the edge type
 */
public String getDescription();

/**
 * Gets the ontology name
 * @return The ontology name
 */
public String getOntologyName();

/**
 * Set the name of the edge type. One should be careful about this function
 * because it can make the database inconsistent.
 * @param newName The new name of the edge type.
 */
public void setName(String newName);

/**
 * Sets the description of the edge type
 * @param newDescription The new description of the edge type
 */
public void setDescription(String newDescription);

/**
 * Get the parent types of the edge type
 * @return A vector of edge types with the direct parents of the edge
 */
public Vector getParents();

/**
 * Get the children of the edge type
 * @return A vector of edge types with the direct children of the edge
 */
public Vector getChildren();

/**
 * Add a new parent type to the edge type
 * @param newParent The new parent
 */
public void addParent(EdgeType newParent);

/**
 * Add a new child type to the edge type
 * @param newChild The new child type
 */
public void addChild(EdgeType newChild);

/**
 * Checks if a given type is parent of this edge type. This function is
 * recursive, so it checks not only the direct parents, but all the way
 * to the highest parent.
 * @param type The type of edge being tested
 * @return True if it is a parent
 */
public boolean isParent(EdgeType type);

/**
 * Checks if a given type is parent of this edge type. This funcion is
 * recursive, so it checks not only the direct children.
 * @param type The type of edge being tested
 * @return True if it is a child
 */
public boolean isChild(EdgeType type);

/**
 * Getter for all the VertexTypes that can be in the domain of the
 * given edge
```

```
     * @return A vector of Vertex types that are the domain of the edge type
     */
    public Vector getDomain();

    /**
     * Getter for all the VertexTypes that can be in the range of the given
     * edge type
     * @return A vector of vertex types that are the range of this edge type
     */
    public Vector getRange();

    /**
     * Adds a domain to the edge.
     * @param newDomain The new domain
     */
    public void addDomain(VertexType newDomain);

    /**
     * Adds a range to the edge type
     * @param newRange The new range
     */
    public void addRange(VertexType newRange);

    /**
     * Checks if a certain vertex type is in the domain of the edge type
     * @param type The vertex type being tested for
     * @return True if it is in the domain
     */
    public boolean inDomain(VertexType type);

    /**
     * Checks if a certain vertex type is in the range of the edge type
     * @param type The vertex type being tested for
     * @return True if it is in the range
     */
    public boolean inRange(VertexType type);

    /**
     * Delete a certain vertex type from the domain of the edge type
     * @param type The vertex type to be removed
     */
    public void deleteDomain(VertexType type);

    /**
     * Delete a certain vertex type from the range of the edge type
     * @param type The vertex type to be removed
     */
    public void deleteRange(VertexType type);

    /**
     * Gets the ontology of the given type.
     * @return The ontology interface.
     */
    public Ontology getOntology ();

    /**
     * Compares two edge types to see if they are the same. Compare name and
     * ontology name.
     * @param edgeType The edge type to compare
     * @return True if they are the same
     */
    public boolean equals(EdgeType edgeType);
```

Vertex

```
    /**
     * Getter for the name of the vertex
     * @return The name of the vertex
     */
    public String getName();
```

```
/**
 * Setter for the name of the vertex. This setter has to be used with care
 * because it can make the database inconsistent.
 * @param newName The new name for the vertex
 */
public void setName(String newName);

/**
 * Gets the database name
 * @return The database name
 */
public String getDatabaseName();

/**
 * Gets the types of the vertex
 * @return A vector of VertexTypes with the types this vertex is registered
 */
public Vector getTypes();

/**
 * Add a new type to the vertex
 * @param newType The new type of vertex
 */
public void addType(VertexType newType);

/**
 * Deletes a type from the vertex
 * @param type The type to be deleted
 */
public void deleteType(VertexType type);

/**
 * Get all the edges that are outgoing from this vertex
 * @return A vector with all the edges that have this vertex as the source
 */
public Vector getOutEdges();

/**
 * Typed getter for the outgoing edges of the vertex. Returns all edges
 * of the given type.
 * @param ofType The type of edge
 * @return A vector of Edge elements that are the edges of the given type
 */
public Vector getOutEdges(EdgeType ofType);

/**
 * A general getter for all incoming edges of this vertex
 * @return A vector of Edge elements that contains all edges that have this
 * vertex as range
 */
public Vector getInEdges();

/**
 * A typed getter for all incoming edges of this verted of the given type
 * @param ofType The type of edge
 * @return A vector of Edge elements that contains all edges that have
 * this vertex as range and the given type.
 */
public Vector getInEdges(EdgeType ofType);

/**
 * Remove an edge from the list of incoming or outgoing edges. Note that
 * the edge is not disconnected from the other end.
 * @param edge The edge to remove
 */
public void removeEdge(Edge edge);

/**
 * Adds an input edge. Automatically sets the target of the edge to be this
 * vertex
 * @param edge The edge to connect.
```

```
     */
    public void addInEdge(Edge edge);

    /**
     * Adds an output edge. Automatically sets the source of the edge to be
     * this vertex.
     * @param edge The edge to connect.
     */
    public void addOutEdge(Edge edge);

    /**
     * Getter for the database
     * @return The database that this element belongs to
     */
    public GraphData getDatabase();

    /**
     * Checks the type of the vertex - does the type test recursively.
     * @param type The type being asked
     * @return True if it is of the given type
     */
    public boolean isOfType(VertexType type);

    /**
     * Checks if it is the same vertex. Compare the name and the database.
     * @param vertex The vertex being checked
     * @return True if they are the same
     */
    public boolean equals(Vertex vertex);
```

Edge

```
     * Getter for the name of the edge
     * @return
     */
    public String getName();

    /**
     * Setter for the name of the edge. It is important to note that this
     * function does not maintain the consistency of the database, so it has to
     * be used with care.
     * @param newName The new name of the edge.
     */
    public void setName(String newName);

    /**
     * Gets the database name
     * @return The database name
     */
    public String getDatabaseName();

    /**
     * Getter for the types registered to the edge.
     * @return A vector of EdgeTypes containing all types directly registered
     * to this edge
     */
    public Vector getTypes();

    /**
     * Register a type to the edge
     * @param type The type to add to the edge
     */
    public void addType(EdgeType type);

    /**
     * Delete a type from the edge
     * @param type The type to delete
     */
    public void deleteType(EdgeType type);

    /**
```

```
     * Getter for the vertex in the source of the edge
     * @return The source vertex
     */
    public Vertex getSourceVertex();

    /**
     * Getter for the vertex in the target of the edge
     * @return The target vertex
     */
    public Vertex getTargetVertex();

    /**
     * Sets the source vertex of this edge
     * @param sourceVertex The source vertex
     */
    public void setSourceVertex(Vertex sourceVertex);

    /**
     * Sets the target vertex of this edge
     * @param targetVertex The target vertex
     */
    public void setTargetVertex(Vertex targetVertex);

    /**
     * Getter for the database
     * @return The database that this element belongs to
     */
    public GraphData getDatabase();

    /**
     * Checks the type of the edge - does the type test recursively.
     * @param type The type being asked
     * @return True if it is of the given type
     */
    public boolean isOfType(EdgeType type);

    /**
     * Checks if it is the same edge. Compare the name and the database.
     * @param edge The edge being checked
     * @return True if they are the same
     */
    public boolean equals(Edge edge);
```

EWTable

```
    /**
     * Get the top count features in the feature table
     * @param count The number of top feature to find
     * @return A vector of vector with all the count best features with
     * element vector with the source, target and feature value.
     */
    public Vector getTop(int count);

    /**
     * Adds a feature count to the given pair of input and output vertices
     * @param v_source The input vertex
     * @param v_target The output vertex
     * @param count The feature to add
     * @return The current feature value
     */
    public double addCount(Vertex v_source, Vertex v_target, double count);

    /**
     * Get the current feature value given the pair of input and output vertices
     * @param v_source The input vertex
     * @param v_target The output vertex
     * @return The current feature value
     */
    public double getCount(Vertex v_source, Vertex v_target);

    /**
```

```
 * Getter for the description of the EWTable
 * @return The description of the table
 */
public String getDescription();

/**
 * Setter for the description of the table
 * @param newDescription The description of the table
 */
public void setDescription(String newDescription);

/**
 * Getter for the registered from vertices
 * @return The names of all from vertices
 */
public Vector getFromVertices();

/**
 * Getter for the registered to vertices
 * @return The name of all to vertices
 */
public Vector getToVertices();

/**
 * Choose one random element using the roulette wheel approach – ADDITION FOR
 * USING THE GA ALGORITHM
 * @return The selected starting vertex ID
 */
public String chooseRoulette();

/**
 * Choose an element based on neighborhood to the starting element and a
 * distance
 * @param startingVertex The selected starting vertex for finding the neighbor
 * @param distance The maximum distance for the neighborhood
 * @return The ID of the target vertex
 */
public String getNeighbor(String startingVertex, double distance);
```