



Collège doctoral

*N° attribué par la bibliothèque*

--	--	--	--	--	--	--	--	--	--	--	--

# THESE

pour obtenir le grade de

**Docteur de l'Ecole des mines de Paris**

Spécialité « Informatique Temps Réel, Robotique et Automatique »

présentée et soutenue publiquement

par

**Kevin HUGGINS**

le 29 avril 2005

<p><b>LINEA: UNE MÉTHODE DE CALCUL PAR GRAPHES DE LA PROXIMITÉ DES OBJETS SUR INTERNET</b></p>
--

Jury

M. D. Carteret	Examineur
M. C. Carver	Rapporteur
M. J. J. Girardot	Rapporteur
M. P. Jouvelot	Examineur
M. R. Mahl	Directeur de thèse



## ACKNOWLEDGMENTS

I would first like to thank my thesis advisor, Robert Mahl, for deftly guiding me through the dissertation process. His wise counsel and time that he unselfishly invested in me were key to the successful completion of my research. I am truly grateful.

This work is the fruit of a partnership with I-Nova, an innovative software development company that specializes in knowledge management systems. David Cateret, co-founder of I-Nova, developed the intuitive idea of the algorithm. He sought out the Centre de Recherche en Informatique (CRI) to develop the algorithm. Through this partnership, our research was awarded a formal approval of the French National Software Technologies Network (RNLT) under the Linea project. The goal of this project was to develop a generic proximity algorithm that could be applied to multiple domains. In addition, we were also awarded funding by the French government after getting the EUREKA label for the Response project. This project's goal includes the development of a helpdesk system that employs case-based reasoning in combination with Bayesian calculations. Linea is integrated into this system in order to measure the proximity between potential and known cases. This innovative approach overcomes the challenge posed by some partially defined datasets that are difficult to handle in pure case-based reasoning approaches.

I am also indebted to Pierre Jouvelet for the countless hours he shared with me discussing algorithms, making the many revisions to my dissertation and debating various important life issues.

The two rapporteurs also contributed significantly to the quality of my thesis. Both Dr. J.J. Girodot and Lieutenant Colonel Curtis Carver provided extremely valuable and insightful comments and suggestions. I am very appreciative of their efforts.

Next, I would like to thank the members of the Centre de Recherche en Informatique (CRI). Each person in the center went out of their way to make me feel welcomed. I appreciate their patience with me as I not only learned their beautiful language, but a little of their culture as well.

Last but definitely not least, I want to thank my wife and kids for all of their support. This dissertation was undoubtedly a team effort and I could not have accomplished it without my family team.

## ABSTRACT

Cette thèse présente Linea, une nouvelle approche pour calculer la proximité d'objets. Pour cela, nous utilisons une combinaison des mesures de distance dans un espace métrique et l'analyse des liens pour définir la proximité des objets sur Internet. Cette structure est générique et peut être appliquée à plusieurs domaines comme la gestion de la relation client, les ressources humaines ou les systèmes de recommandation. Puis, nous implémentons plusieurs versions de l'algorithme et nous comparons non seulement la précision des implémentations mais aussi leurs performances à travers des expériences. Ensuite, nous décrivons la conception et implémentation de Linea en présentant les défis que nous avons rencontrés et les améliorations que nous avons faites. Enfin nous donnons quelques domaines d'applications potentiels et champs de travaux futurs.

This dissertation presents Linea, a novel approach to proximity calculations in a linked metric space. We use a combination of metric space distance calculations and link analysis to determine the proximity between internet objects. This framework is generic and can be applied to various domains such as help desk support, human resources or recommender systems. We implement several versions of the algorithm and compare not only their accuracy versus manual approaches, but also their performances through experimentation. We also describe the design and implementation of Linea providing insights on challenges and improvements made. Finally we give possible application domains and areas for future work.

## TABLE OF CONTENTS

<b>ACKNOWLEDGMENTS .....</b>	<b>ii</b>
<b>ABSTRACT.....</b>	<b>iv</b>
<b>LIST OF FIGURES .....</b>	<b>ix</b>
<b>Chapter 1 .....</b>	<b>13</b>
<b>Introduction.....</b>	<b>13</b>
1.1    Research Objectives and Approach .....	15
1.2    Practical Problems .....	15
1.2.1    A Human Resource Problem .....	15
1.2.2    A Case-based Reasoning Problem.....	17
1.3    The Proximity Calculator Algorithm: A first look .....	19
1.3.1    The supporting domain graph .....	20
1.3.2    The algorithm.....	20
1.4    Applications .....	22
1.4.1    Web pages.....	22
1.4.2    Recommender systems.....	23
1.4.3    Netfires.....	24
1.5    Organization of the Dissertation .....	24
<b>Chapter 2 .....</b>	<b>27</b>
<b>State of the art .....</b>	<b>27</b>
2.1    Link Analysis .....	27
2.1.1    Library Science .....	28
2.1.2    Internet Applications.....	29
2.2    Proximity Searching in Metric Space .....	34
2.2.1    Pivot techniques .....	35
2.2.2    Clustering.....	37
2.3    Recommender systems.....	43
2.4    Conclusion .....	46
<b>Chapter 3 .....</b>	<b>49</b>
<b>Mathematical Preliminaries.....</b>	<b>49</b>
3.1    Terminology.....	49
3.2    Metric Space Distance Properties .....	53
3.3    Metric Space Proximity Properties .....	54
3.4    Graph Properties .....	55
3.5    Similarity Measures .....	56
3.5.1    Finite number of attributes.....	56
3.5.2    Relationships.....	58
3.5.3    Valued relationships.....	61
3.6    Conclusion .....	61
<b>Chapter 4 .....</b>	<b>63</b>
<b>Linea Proximity Calculator .....</b>	<b>63</b>
4.1    Introduction.....	63
4.2    Naïve Algorithm .....	64
4.2.1    Definition .....	64
4.2.2    A Visual Example.....	67
4.2.3    The algorithm implementation.....	70
4.2.4    Time complexity .....	73
4.3    Bottom-up .....	81
4.3.1    Proximity of an element to an image set.....	81

4.3.2	Proximity of 2 image sets .....	82
4.3.3	The algorithm.....	83
4.3.4	Time complexity .....	85
4.4	Iterative .....	87
4.4.1	Algorithm description .....	87
4.4.2	Time complexity .....	88
4.4.3	Proof of convergence .....	89
4.5	Conclusion .....	95
<b>Chapter 5</b>	<b>.....</b>	<b>97</b>
<b>Experiments</b>	<b>.....</b>	<b>97</b>
5.1	Introduction.....	97
5.2	Data description .....	97
5.2.1	Web pages.....	98
5.2.2	Corporate.....	99
5.3	Accuracy .....	105
5.3.1	Intuitive Accuracy Experiment.....	106
5.3.2	Manual calculations .....	113
5.3.3	Accuracy of Automated methods vs. Manual.....	117
5.4	Performance .....	118
5.4.1	Approach.....	118
5.4.2	Impact of precision on iterative method .....	118
5.4.3	Results.....	119
5.5	Conclusions.....	121
<b>Chapter 6</b>	<b>.....</b>	<b>123</b>
<b>Implementation</b>	<b>.....</b>	<b>123</b>
6.1	Introduction.....	123
6.2	System Architecture Overview .....	123
6.2.1	Introduction.....	123
6.2.2	Architecture Elements.....	124
6.3	Multi-tier approach .....	130
6.3.1	Introduction.....	131
6.3.2	Benefits of the approach .....	131
6.3.3	DAO.....	133
6.4	DB Design and Implementation.....	134
6.4.1	Introduction.....	134
6.4.2	Object modeling.....	135
6.4.3	Use cases.....	136
6.4.4	Class diagrams .....	137
6.4.5	Database schema conversion .....	138
6.5	Implementation Improvements .....	142
6.5.1	Object instantiation efficiency .....	142
6.5.2	Lazy local proximity initialization.....	143
6.6	Conclusion .....	144
<b>Chapter 7</b>	<b>.....</b>	<b>146</b>
<b>Conclusion</b>	<b>.....</b>	<b>146</b>
7.1	Summary of Contributions.....	146
7.1.1	A novel approach to metric space calculations.....	146
7.1.2	A comparison of implementation methods.....	147
7.1.3	Linea implementation .....	147
7.2	Directions for Future Research .....	147

7.2.1	Response integration .....	147
7.2.2	Netfires.....	148
7.2.3	Parallelization .....	149
7.2.4	Web Service .....	149
	<b>Bibliography .....</b>	<b>150</b>
	<b>Annex A.....</b>	<b>154</b>
	<b>A Simple Example.....</b>	<b>154</b>
	<b>Annex B.....</b>	<b>160</b>
	<b>UML to Schema guide .....</b>	<b>160</b>



## LIST OF FIGURES

Figure 1 Major Corporation .....	16
Figure 2 Major Corporation with links .....	17
Figure 3 Partial listing of the Renault incident database .....	18
Figure 4 Partial graph representation of the Renault incident database.....	19
Figure 5 Sample domain graph .....	21
Figure 6 Sample domain graph with an image set pair.....	22
Figure 7 According to Salton's and Garfield's research document A is more important due to the number of references. ....	29
Figure 8 Examples of types of queries. a) Range b) Nearest Neighbor, and c) k- nearest neighbors .....	35
Figure 9 Example of a pivot in a metric space.....	36
Figure 10 A clustering taxonomy .....	38
Figure 11 Hierarchical clustering.....	39
Figure 12 a. Recursive asymmetrical partitions b. search space and query balls .....	42
Figure 13 Our work.....	48
Figure 14 Node .....	50
Figure 15 Relationships & links .....	51
Figure 16 Inbound and outbound relationships .....	52
Figure 17 Graph representing a problem domain .....	53
Figure 18 Image Set of x.....	56
Figure 19 City Block distance.....	57
Figure 20 Similarity coefficients .....	59
Figure 21 Probability distribution constraints.....	60
Figure 22 Spanning tree induced by previously visited nodes. Nodes which are already visited in a path do not recursively call any other node computation .....	65
Figure 23 Detailed example –Node <i>a</i> .....	67
Figure 24 Detailed example-node <i>b</i> .....	68
Figure 25 Detailed example-node <i>c</i> .....	69
Figure 26 Detailed example-nodes <i>a</i> , <i>b</i> and <i>c</i> .....	70

Figure 27 Base case .....	74
Figure 28 Tree induced by a node with no outbound relationships.....	75
Figure 29 Time complexity when $R=2$ and $h=1$ .....	76
Figure 30 Time complexity at $h=2$ .....	78
Figure 31 Time complexity at $h=3$ .....	79
Figure 32 Computing the distance of any element $x$ of $X$ to image sets of all elements $u$ of $Y$ or $Z$ .....	82
Figure 33 Elements of an image set proximity measure.....	83
Figure 34 Directed graph with no loops converted into modified tree .....	84
Figure 35 Movie recommendation system example with undetermined values.....	92
Figure 36 With the intersection of the two image sets at $E$ , we are assured a non-zero constant.....	93
Figure 37 The proximity between $A$ and $B$ can be computed due to the path between them created from the links and intersections of the other elements. ....	94
Figure 38 Three class example .....	95
Figure 39 Web pages data model.....	98
Figure 40 Corporate Data model.....	100
Figure 41 Competences per Person comparison.....	101
Figure 42 Competencies per Person comparison.....	102
Figure 43 Groups per Person comparison.....	103
Figure 44 Super-groups per Sub-group.....	104
Figure 45 Groups per Competency .....	105
Figure 46 Calculation paths .....	107
Figure 47 Heuristics.....	109
Figure 48 Experimentation software architecture.....	110
Figure 49 Path Manager Class Diagram .....	111
Figure 50 Path manager example.....	112
Figure 51 ExperimentDriver class diagram .....	113
Figure 52 Terms related to WP2950 and WP2951 .....	114
Figure 53 Proximity calculations in the Node Terms .....	115

Figure 54 Second path starting at Web Pages.....	116
Figure 55 Manual Proximity Results for Web Pages.....	117
Figure 56 Calculation time versus precision.....	119
Figure 57 Comparison performance results experiments .....	120
Figure 58 Experiment result insights .....	121
Figure 59 Proximity Calculator System Architecture.....	125
Figure 60 ProximityDataElement and ProximityDataPair .....	127
Figure 61 ElementProximity and NodeProximity .....	128
Figure 62 GraphProximity .....	129
Figure 63 Multi-tier architecture.....	131
Figure 64 DAO related classes .....	134
Figure 65 Data object model - first cut .....	136
Figure 66 Linea use case diagram.....	137
Figure 67 Database class diagram.....	138
Figure 68 Guide to converting a strong class to table schema.....	139
Figure 69 Guide to converting a many-to-many association to table schema .....	140
Figure 70 Guide to optimizing the schema by eliminating unneeded many-to-one table schema.....	141
Figure 71 Table schema for the graph database.....	142
Figure 72 Sample graph.....	155
Figure 73 Strong class.....	160
Figure 74 Weak class.....	161
Figure 75 Super/sub classes .....	161
Figure 76 Many-to-many associations.....	162
Figure 77 Many-to-one associations.....	163
Figure 78 One-to-one associations.....	163
Figure 79 Multi-valued attributes .....	164
Figure 80 Existence dependencies on many-to-one associations .....	165
Figure 81 Existence dependencies in one-to-one associations .....	166
Figure 82 Many-to-one reduction .....	166



# Chapter 1

## Introduction

The Internet, created over 20 years ago by the US Department of Defense Advanced Research Projects Agency (DARPA) as an experimental communications medium in the event of a nuclear holocaust, has instead spurred one of the greatest economic booms in history. The major driver for this success is the World Wide Web (WWW) invented 15 years ago by Tim Berners-Lee. The WWW fueled a tremendous explosion of online data. From 1990, with Mr. Berners-Lee's original website of a few pages to today with over 8 billion pages, the WWW has also created a vast opportunity for searching technologies. This need for searching technologies, which is conceptually based on finding the proximity between two objects, has allowed companies like Google and Yahoo to profit well. However, there remains many more challenges and opportunities within this domain.

Whereas much work has been done in the areas of text-based analysis sometimes combined with some form of link analysis, there has been relatively little work that exploits highly linked environments with minimal, semi-structured text. By semi-structured we refer to meta-data used with content, such as databases or XML documents. Popular search techniques used today determine proximity primarily by document content with a document's relevance measured by link analysis. Semantic analysis and statistical methods depend on significant textual content to function well.

Our approach does not require a large amount of text. Hence, data sets with reduced textual content can be measured more effectively.

Another source of motivation for our research is recommender systems. Over the last decade much work has been done in this area. Most of this work has concentrated in developing new methods for recommending items to users and vice versa, such as recommending books for customers and movies to Web site visitors. These recommendation methods are usually classified into collaborative, content-based and hybrid methods (Balabanovic and Shoham, 1997) and are described in more detail in Chapter 2. There is a recent interest in applying multi-dimensional methods to recommender models (Adomavicius, 2005).

In all of these methods, there are distance calculations that could be implemented with Linea. Also, in all of these methods, there is an inherent weakness. When a user is given a recommendation, say for a book in an online bookstore scenario such as Amazon, it is the result of other users actually buying the recommended book. In set theory terms, the recommended items is always the result of an intersection of the set **A** of books that the user is planning to purchase and the set **B** of books that other users have purchased. The set of recommendations comes for  $\mathbf{B} - \mathbf{A}$ . However, there may be items that cannot be ascertained based on an intersection. There may be items that are closely *related* to the item about to be purchased. For example, using Linea, users could be recommended books that are *close* to the book that the user is purchasing. This flexibility provides a more robust recommendation approach as we are not limited to making recommendations based matches between books purchased by the user and books previously purchased by other clients.

This dissertation presents Linea, a solution for searching in this type of linked environment. We develop Linea by using a generic proximity calculating algorithm that is optimized for highly linked environments with minimal semi-structured text. We take a plug and play approach that allows Linea to easily be applied to different domains.

## **1.1 Research Objectives and Approach**

The objective of this dissertation is two fold. First we develop a generic algorithm that calculates the proximity between 2 objects in a linked metric space consisting of minimal, semi-structured text. Our second goal is to implement the algorithm within Linea and apply it to multiple domains.

To this end, after developing the basic algorithm, we then implement three versions, naïve, bottom-up and iterative. The three approaches present distinct opportunities and challenges and we seek to compare and contrast not only these developmental approaches, but also their experimental results.

## **1.2 Practical Problems**

I-nova requested assistance from the École des Mines de Paris in addressing two kinds of practical problems: A human resource problem related to Electricité de France (EDF) which is described in section 1.2.1. And then a case-based reasoning problem related to equipment failures of Renault Vehicular Industriel (RVI) which is described in 1.2.2.

### **1.2.1 A Human Resource Problem**

As a motivation to the subject area, we present a simple example. Consider the research and development (R&D) division of EDF which consists of thousands of employees. In this division, there are three kinds of things that interest us: people, competencies, and groups. In this corporation, people can have competencies and they can belong to groups. In addition, groups may also have competencies.

---

## Major Corporation

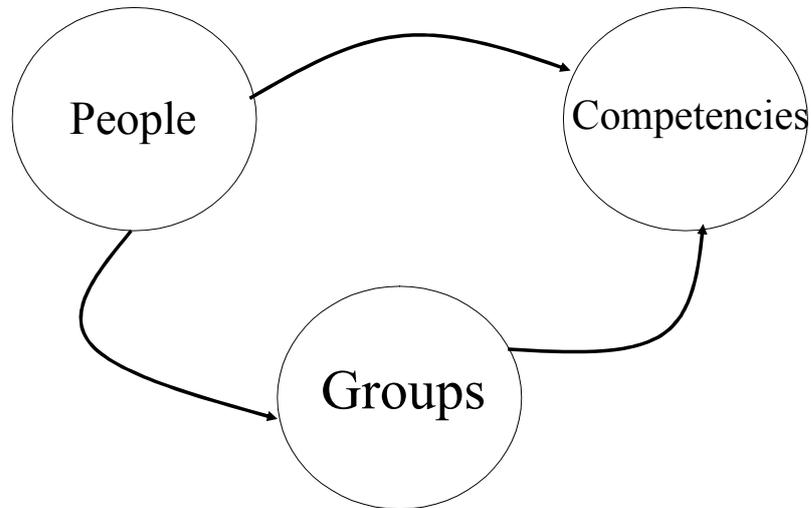
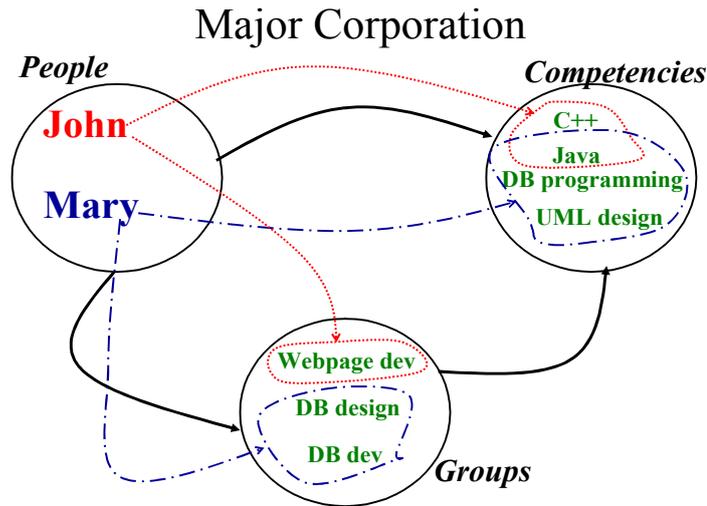


Figure 1 Major Corporation

---

Consider two employees: John and Mary. They have attributes which describe them, such as their name, their age, their education level, etc. In our example, we know that John and Mary may be associated with certain competencies and groups within the corporation. Accordingly, we would like to find out how ‘close’ John is to Mary. There are several approaches to determining this. The approach that we present in this work considers not only the attributes that directly describe John and Mary, but also the elements that are related to John and Mary (competencies and groups).

Let us consider John first. Let John be 30 years old, and have a master’s level education. He has 2 competencies: java programming and C++ programming. He also belongs to the ‘webpage development’ group. Now, let us consider Mary. She is 27 years old and has a bachelor’s level education. She has 3 competencies: database programming, java programming, and UML design. She belongs to two groups in the corporation: database design, database development.




---

Figure 2 Major Corporation with links

---

The approach that we present to see how close John is to Mary takes into account not only the direct attributes of John and Mary (we call this the **LocalProximity**) but also the proximity of the sets of elements related to John and Mary (we call this the **ImagedProximity**). We combine these two proximity values for our final value which we call the **GlobalProximity**.

### 1.2.2 A Case-based Reasoning Problem

The Renault Trucks division of RVI is responsible for managing help desk requests from garages that repair and maintain Renault trucks. To help manage this process Renault maintains a database of these incidents that are called in. Here is an extract of the incident file.

---

<b>ID</b>	<b>Part</b>	<b>Problem</b>	<b>Corrective Action</b>
8	Chassis	Back fender/Support ruptured	Support modification
106	Engine	Actuator causes limit on speed	Mount special model
556	Steering	Column locks	Replace steering column

Figure 3 Partial listing of the Renault incident database

---

As a client calls in with an incident, an entry is added to the database. The goal of each session is to identify the corrective action to the problem. Once this is done, it is also logged into the database under corrective actions.

The data file can be modeled within a graph structure with four nodes:

- ID number
- Vehicle part
- Problem description
- Corrective action

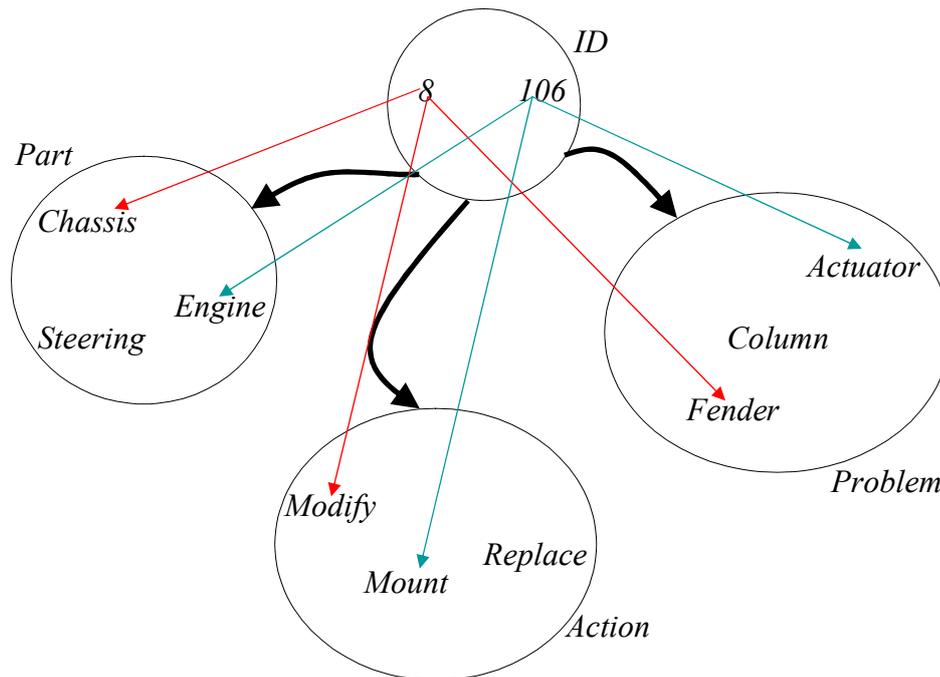


Figure 4 Partial graph representation of the Renault incident database

Incidents already in the database normally have a corrective action. New incidents do not. Hence, the aim would be to find the incident that is currently in the database that is closest to the new incident. If the data-based incident is close enough to the new one, then its corrective action should apply to the new incident. Otherwise, it should help finding a solution faster since the system is finding similar incidents.

### 1.3 The Proximity Calculator Algorithm: A first look

In this section I will describe in general terms the process of the algorithm. Before doing so, I will explain the supporting graph structure, which is integral to the proximity calculations.

### 1.3.1 The supporting domain graph

The proximity is calculated with a domain defined within a graph structure. A node represents a class of elements. Elements within a node represent instances of the class of elements described by the node. For example, in Figure 1, the node Persons holds people. The elements in Persons are examples of people within the domain. The associations between the nodes indicate that there is a defined relationship between elements in both nodes. For example, the Persons and Groups have an association. It defines the association that people in this domain can belong to groups. The links between elements in one node to elements in another indicate an instance of a link between the two elements. For example, the link between John in node Persons and Webpage Development in node Groups in Figure 2 indicates that John is a member of the group Webpage Development.

### 1.3.2 The algorithm

The proximity calculator algorithm measures the GlobalProximity between 2 elements in the same node. As indicated in the previous section, this calculation includes not only the direct attributes of each element (LocalProximity) but also the proximity of elements associated with each of the 2 elements (ImagedProximity). We use a weight value,  $\alpha$ , to balance the importance between the LocalProximity and ImagedProximity results.

We will now expand this description somewhat. Consider a domain represented by a graph with 5 nodes. We name each node A through E. In Node A, consider 2 elements, x and y. See Figure 5.

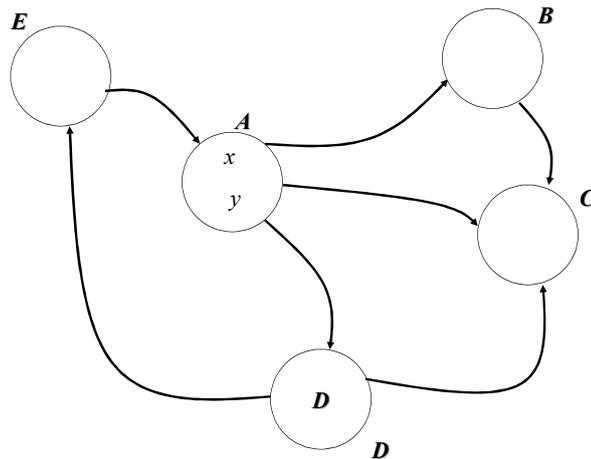


Figure 5 Sample domain graph

---

Each element in the domain has a set of attributes which describe it. Recall that in the large corporation example from the previous section, each employee in the Persons node had 2 attributes: age and education level. The LocalProximity calculation compares the element attributes. This particular calculation is domain specific. We will discuss some possible approaches in Chapter 4.

For the ImagedProximity calculation, we consider the sets of elements that are linked to  $x$  and  $y$  in other nodes. These sets are called image sets and can be located in any node that is associated with Node A by an outbound directed relationship. For example, in Figure 5, Nodes B, C and D can contain image sets for  $x$  and  $y$  in Node A.

We determine an ImagedProximity by measuring the set distance between the pair of image sets associated with  $x$  and  $y$  in each node that is related to Node A. We can see an example of this in Figure 6 with the 2 image sets of  $x$  and  $y$  that are located in Node B.

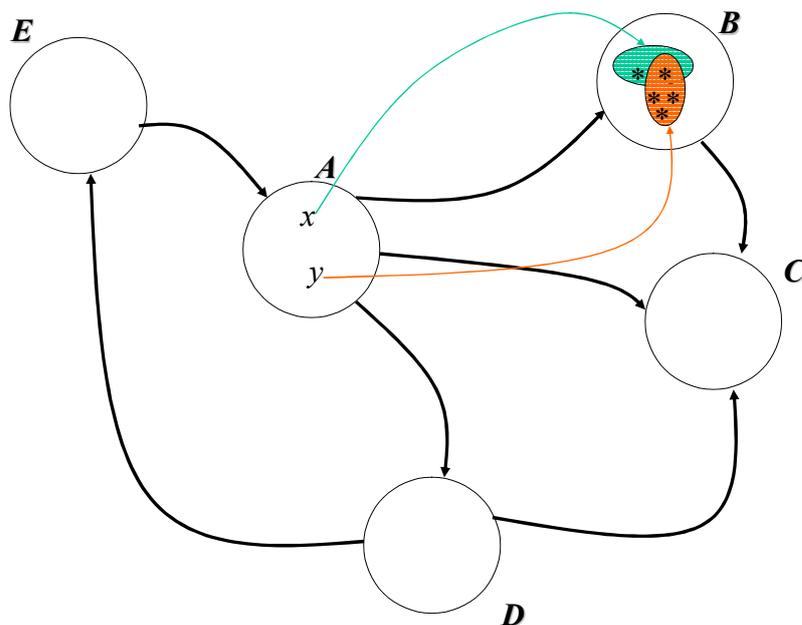


Figure 6 Sample domain graph with an image set pair

We perform an ImagedProximity calculation for each of the nodes that can have image sets of  $x$  and  $y$ . These are Nodes B, C and D. During the set distance calculations, we recursively calculate the GlobalProximity, which we will discuss in more detail in Chapter 4.

## 1.4 Applications

In the previous sections, we have mentioned the human resources and helpdesk domains as application areas for the Linea agent. However, there are other domains that would also benefit from Linea.

### 1.4.1 Web pages

Linea can be applied to computing the proximity of web pages to one another. In particular web pages could be characterized by their title, their author, their

publication date, the terms included on the page and their hyperlinks. We could use these characteristics to define the nodes for a graph representation of this domain. Using the Linea algorithm and by considering all the elements of each node, we can then compute the proximities of two Web pages, or of 2 different terms. This could be used in electronic commerce to give suggestions to customers about other pages to visit, or other information related to the same subject. These proximities can also be used to classify Web pages using clustering techniques.

### **1.4.2 Recommender systems**

Recommender systems have been well researched over the past decade. Most of this work has been focused on developing new methods of recommending items to users and vice versa. With the advent of the Internet and e-commerce, recommender systems have found many applications.

One application is movie recommender systems that find films that would interest a potential movie-goer. The domain graph would possibly consist of 3 nodes: people, film preferences and movies. The Linea agent would find the movies closest to a person as described by their preferences and the set of movies that other movie-goers liked.

Similarly, Amazon.com has a well known recommender system for its clients. After the customer has chosen a book and is ready to finalize a purchase, Amazon proposes additional books. This set of books is determined by analyzing what other customers purchased when buying the same book. Linea could be used to enhance this system by providing contextual information via multi-dimensional data. Instead of just considering what other customers purchased, with the Linea system, they could also take into account attributes of the customer, time or date of purchase, or books that are *close* to the book purchased but possibly not the same as other customers have purchased.

### **1.4.3 Netfires**

The US Defense Advanced Research Agency (DARPA) is sponsoring the development of the next generation of artillery systems. Today, artillery systems are ‘point-and-shoot.’ When an army unit encounters resistance from the adversary, it requests artillery support. The requesting unit provides the enemy location data and with this, the supporting artillery unit fires rounds onto the target. This system has changed very little since the time of Napoleon.

Netfires is a radical change. Instead of waiting to be called by an army unit confronting an enemy, the Netfires system launches a group of small missiles that loiters over the battlefield beforehand. When a friendly army unit encounters enemy resistance and calls for artillery support, the group of loitering missiles flying overhead immediately decide among themselves which will respond. The one that is chosen then targets the enemy and attacks it by falling out of the sky.

The Linea agent could be used to determine which loitering missile flying overhead is ‘closest’ to the enemy target. Factors that could be considered to compute proximities include the type of target (infantry, armor), fuel level on the missile, enemy location and missile location, to name a few.

## **1.5 Organization of the Dissertation**

Chapter 2: The state of the art related to our problem domain is presented. Our domain is actually a combination of multiple domains. We present the current work in these domains and show how our work is related and how we extend it.

Chapter 3: The terminology used throughout the dissertation is introduced. The main areas focus on algorithmic graph terms. We have introduced terms specific to our problem. The mathematical preliminaries are also described in this chapter. This will provide the foundation for our algorithm development.

Chapter 4: The Linea agent algorithm is described. After presenting the basic equations, we then present three approaches for its implementation: naïve, relaxed iterative, and bottom-up.

Chapter 5: The experiments are detailed in this chapter. The experiments are centered on the 3 implementation approaches to the Linea algorithm: naïve, relaxed iterative and bottom-up. We develop experiments to test for not only correctness as compared to manual calculations of the basic algorithm, but also to compare their performances.

Chapter 6: The Linea agent implementation is the focus of this chapter. We discuss not only the development of the software that implements the algorithm, but also the design and development of the supporting database.

Chapter 7: Concluding remarks and directions for future work are presented.



# Chapter 2

## State of the art

In this chapter we will describe the state of the art of our domain of research. Our work touches on several domains as it is a hybrid approach to measuring proximity. As we discuss each of the supporting domains, we will also relate it to our work. Our research draws mainly from three domains: link analysis, proximity searching in metric spaces and recommender systems. We will discuss each area in turn.

### **2.1 Link Analysis**

Link analysis can be considered a subset of relationship analysis. In library science, scientists study relationships and patterns of co-citation and bibliographic coupling. Sociologists, on the other hand, are concerned with social networks of people. Link analysis is commonly associated with Web relationships between pages. All cases share a common theme of defining two same-type entities related via a direct link, co-occurrence or co-citation.

Intuitively, link analysis is the study of the authoritativeness of a document based on its links. Consider Documents  $d_1$  and  $d_2$ . A link from  $d_1$  to  $d_2$  implies an endorsement of  $d_2$  from  $d_1$ . This implication provides the basis for an important body

of research into link analysis. In the following sections we will discuss the development of this domain and describe key algorithms.

### **2.1.1 Library Science**

#### **Foundational work**

Link analysis has its origins in library and information science. Garfield (Garfield, 1973) performed a systematic analysis of journal citation patterns across the science and technology domain. He used the Science Citation Index (SCI) which was a database covering 2400 journals that contained 27 million references to about 10 million different publications. He discovered that the majority of references cited only a few journals. Based on this, he postulated that a good multidisciplinary journal collection does not have to contain a large number of titles to adequately cover the domain (Garfield, 1973). See Figure 7.

Salton (Salton, 1975) extended Garfield's work by combining keyword evidence with citation evidence to improve document retrieval performance. Small and Koeng (Small et al, 1977) provided an algorithm that improved the clustering of journals by the use of two-step bibliographic coupling linkages, instead of one-step linkages, which were the norm at the time.

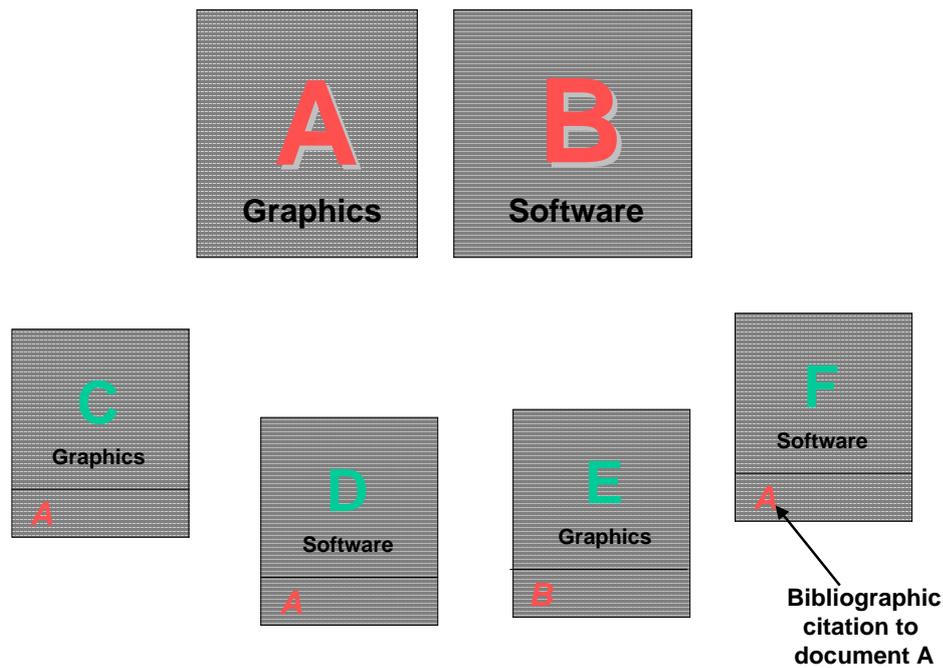


Figure 7 According to Salton's and Garfield's research document A is more important due to the number of references.

---

### **Our work**

In our work, we rely heavily on link analysis. Garfield's and Salton's work are foundational to the field on link analysis. The concept of gleanings and analyzing a set of documents solely by their links is a predecessor to present day Internet search techniques. Although we use link analysis, Salton's extension is closer to our work as we use a combination of link and content analysis to determine proximity. We will discuss other work influenced by this combinational approach in the next section.

### **2.1.2 Internet Applications**

With the explosive growth of the internet in the 90's, link analysis was soon applied to the problem of internet searching. Two well-known approaches, Kleinberg's HITS algorithm (Kleinberg, 1998) and the Google PageRank algorithm (Brin et al,

1998), are eigenvector methods. Essentially, they compute the eigenvectors of particular matrices related to the adjacency graph to find the importance of a document. In the next few subsections, we will discuss the HITS and PageRank algorithms. We will also describe various improvements that have been proposed.

### 2.1.2.1 Eigenvectors

Both the HITS and PageRank algorithms use eigenvectors in their calculations. Hence we provide a brief review of the subject in preparation of our discussions on the two algorithms.

Let  $\mathbf{M}$  be an  $n \times n$  matrix. The number  $\lambda$  is the eigenvalue of  $\mathbf{M}$  if there exists a non-zero vector  $\vec{v}$  such that

$$M\vec{v} = \lambda\vec{v}. \quad [2.1]$$

In this case,  $\vec{v}$  is called the eigenvector of  $\mathbf{M}$  that corresponds to  $\lambda$ .

### 2.1.2.2 HITS

Kleinberg considered *broad-topic queries* which produce thousands of relevant pages on the WWW. He identified this situation as the *Abundance Problem*: *The number of pages that could reasonably be returned as relevant is far too large for a human user to digest.* (Kleinberg, 1998) His approach to address this problem was to filter these types of responses by identifying authoritative pages. By this distilling process, users would be given not only a set of relevant pages, but a smaller (and hopefully more manageable) set of more authoritative pages.

Though an analysis of the link structure of the Internet, Kleinberg found that one could discern not only authoritative pages, but also hub pages. He defined hub pages as web pages that point to good authoritative pages. Similarly, authoritative pages are pointed to by hub pages. Hence, similar to Garfield's work Kleinberg found that all web-pages do not have the same importance. Authoritative and hub pages are more valuable.

### Original algorithm

Kleinberg presented the Hyperlinked-Induced Topic Search (HITS) algorithm which consisted of the following steps. (1) Use a search engine, such as MSN search or AltaVista, to form a root set of pages as a start point; (2) Create the base set by adding pages that either point to or are pointed by the root pages; (3) Total the authority and hub weights of each page in the base set with an iterative algorithm.

Specifically, we can describe the HITS algorithm as follows: For each page let  $a(p)$  and  $h(p)$  indicate its authority and hub weights respectively, which can be determined as below:

$$a(p) = \sum_{q \rightarrow p} h(q) \text{ and } h(p) = \sum_{p \rightarrow q} a(q) \quad [2.2]$$

Let  $M = [m_{ij}]$  denote the adjacency matrix of the base set where  $m_{ij} = 1$  if page  $i$  has a link to page  $j$ , else 0 otherwise. We can then find the authority and hub scores by calculating the eigenvector of the matrix  $M^T M$  and  $MM^T$  respectively.

### HITS Extensions

In (Chakraborti et al, 1998) the authors modify the HITS algorithm to consider also keyword-based evidence. Bharat and Henzinger (Bharat et al, 1998) also added content evidence to the HITS algorithm, but they computed the relevance using the whole document instead of just a window surrounding the hyperlink. Lempel and Moran (Lempel et al, 2001) introduced SALSA, a stochastic approach for link-structure analysis. They proved that their approach was computationally efficient and showed that their algorithm did better than HITS in Tight Knit Community (TKC) effect situations.

### Our work

Our work is related to Kleinberg's in that we employ a similar iterative approach in one of our implementations. We initialize our system with local proximity values for all elements. Then we iterate through the system, calculating the total proximity at step  $i + 1$  using proximity values from step  $i$ . Instead of

determining the hub or authority scores, we determine the proximity. As we discuss in Chapter 5, our iterative approach stabilizes quickly.

Our work has more similarities to Chakroborit and Bharat in that they both considered content in the search algorithms. However, since our approach to content analysis for proximity measurement is generic, the domain specific parts of our calculations are in the content or ‘local’ proximity measures. We apply a more plug and play approach. The domain content measure is user-defined depending on the domain in which Linea is being applied. The results are then combined with the linked proximity measurements.

### 2.1.2.3 PageRank

#### Original algorithm

PageRank is the core algorithm for the Google search engine (Google, 2004). When a page  $u$  has a hyperlink to page  $v$ , it is assumed that the author of  $u$  considers page  $v$  an authority on a certain topic. This is a key assumption in the approach that PageRank finds relevant pages. Now let  $N_u$  represent the number of pages that  $u$  points out to, and  $R(u)$  denote the rank score of page  $u$ . The hyperlink  $u \rightarrow v$  implies  $1/N_u$  units of rank for page  $v$ .

We then iteratively execute the following computation to determine the rank vector for all webpages:

$$\forall v R_{i+1}(v) = \sum_{u \in B_v} R_i(u) / N_u \text{ and } R(u) = \lim_{n \rightarrow \infty} R_n(u) \quad [2.3]$$

We let  $B_v$  equal the set of pages pointing to  $v$ . As we iterate over the set of web-pages, the successive rank scores are recursively calculated from the previous ranks scores of all other pages pointing to them.(Ingongngam et al, 2004)

The PageRank algorithm has an intuitive basis in random walks on graphs. This initial equation is a simple implementation and corresponds to the probability distribution of a random walk on the Web graph. In this Web context, we can

consider the behavior as that of a random surfer. The random surfer just keeps clicking on successively clicking on links. However, if the surfer gets into a loop of web pages, then the model collapses. It will simply stay in this ‘island’ of linked web pages. A real Web surfer is unlikely to continue to in the loop indefinitely. Instead he will jump to another page. A similar challenge is Web pages with no outbound links. In both cases, the random surfer model that the ‘simplified’ PageRank algorithm represents, fails.

The solution that Page and Brin suggested was to prune the nodes with no outbound links and add random jumps to the surfer process in PageRank. Hence the following equation:

$$\forall v R_{i+1}(v) = (1 - \alpha) + \alpha \sum_{u \in B_v} R_i(u) / N_u \quad [2.4]$$

We call  $\alpha$  the damping factor used to modify the transitional probability of the random surfer model.

### **PageRank extensions**

There are several proposed improvements to the PageRank algorithm. Ingongngam and Rungsawang (Ingongngam et al, 2004) propose modifying the PageRank algorithm by propagating a portion of the scores of the source web pages to the destination pages in accordance with the content found on both ends. Although very interesting, their results were inconclusive. In (Xue, et al, 2003) the authors propose an approach that constructs implicit links by mining user access patterns. These implicit links are then incorporated into the PageRank calculations. The authors show a 20% improvement, but only for small web searches.

Ng (Ng et al, 2001) applied the ‘reset’ functionality of the PageRank algorithm to HITS in their Randomized HITS. Their goal was to improve the stability of HITS to small perturbations of a document collection.

### **Our work**

We used three approaches to implementing the Linea agent: naïve, iterative and bottom-up. We discuss each of these implementations in chapter 5. The iterative

implementation quickly reaches a fix point due to a large eigenvalue, as in PageRank. However, instead of traversing the entire graph structure to rank pages, we recursively define proximity measurements based on the measurement of elements in the previous node. This difference stems from the fact that our node does not represent the item we intend to measure. Instead, it represents the class of items that we are measuring.

## 2.2 Proximity Searching in Metric Space

Our work is also influenced by the field of metric space distance algorithms. This active domain seeks to find close objects under an appropriate similarity function, among a finite set of elements. (Chávez et al, 2001).

Consider a universe  $\aleph$  of objects. Furthermore, let

$$d : \aleph \times \aleph \rightarrow \mathfrak{R} \quad [2.5]$$

denote a distance function over the objects. Also let  $\mathfrak{T}$  represent a subset of the universe that must have the following properties:

$$\begin{aligned} d(x, y) &= 0 \Leftrightarrow x = y \\ d(x, y) &= d(y, x) \\ d(x, y) &\leq d(x, z) + d(z, y) \end{aligned} \quad [2.6]$$

Chávez (Chávez et al, 2000b) identifies three typical types of queries:

**Range queries:** retrieve all objects which are within distance  $r$  from query object  $q$ . Or,  $\{p \in \mathfrak{T} \mid d(p, q) < r\}$ .

**Nearest neighbor (NN) queries:** retrieve the closest objects to  $q \in \mathfrak{T}$ . Or,  $nn(q) = \{p \in \mathfrak{T} \mid \forall e \in \mathfrak{T}, d(p, q) \leq d(e, q)\}$

**k-NN queries:** retrieve the  $k$  closest objects to  $q \in \mathfrak{T}$ . Or, return a set  $A \subseteq \mathfrak{T}$  such that  $|A| = k$  and  $\forall s \in A, u \in \mathfrak{T} - A, d(q, s) \leq d(q, u)$ .

There are several approaches to answering these three typical queries. We will discuss some of the most common ones in the following sections.

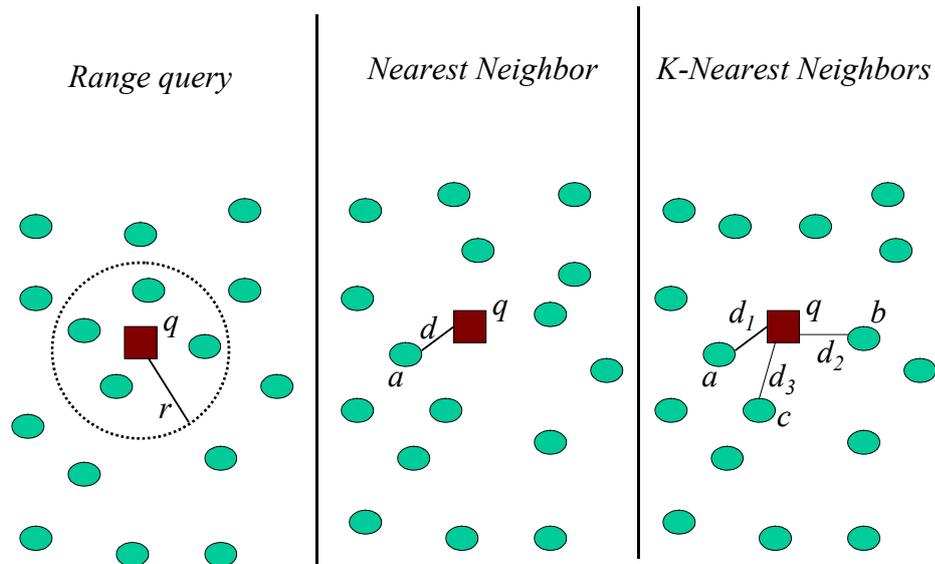


Figure 8 Examples of types of queries. a) Range b) Nearest Neighbor, and c) k-nearest neighbors

### 2.2.1 Pivot techniques

In metric space similarity searches, pivots are a common approach to more efficiently calculating distances. The term pivot refers to any type of object that can be used to prune the search space. Consider  $(S, d)$  which represents a metric space  $S$  that is covered by the distance function  $d$ . A pivot  $p \in S$  is a reference point in  $S$  from which we can ascertain distance information from at least some objects in  $S$ . Let  $S'$  be the set of objects associated with pivot  $p$ . See figure 6. We can then say that for  $u \in S' \subseteq S$  we know

1. the exact value of  $d(p, u)$ ,
2. that  $d(p, u)$  falls within a certain range of values, or
3. that  $u$  is closer to  $p$  than to some other object  $u' \in S$

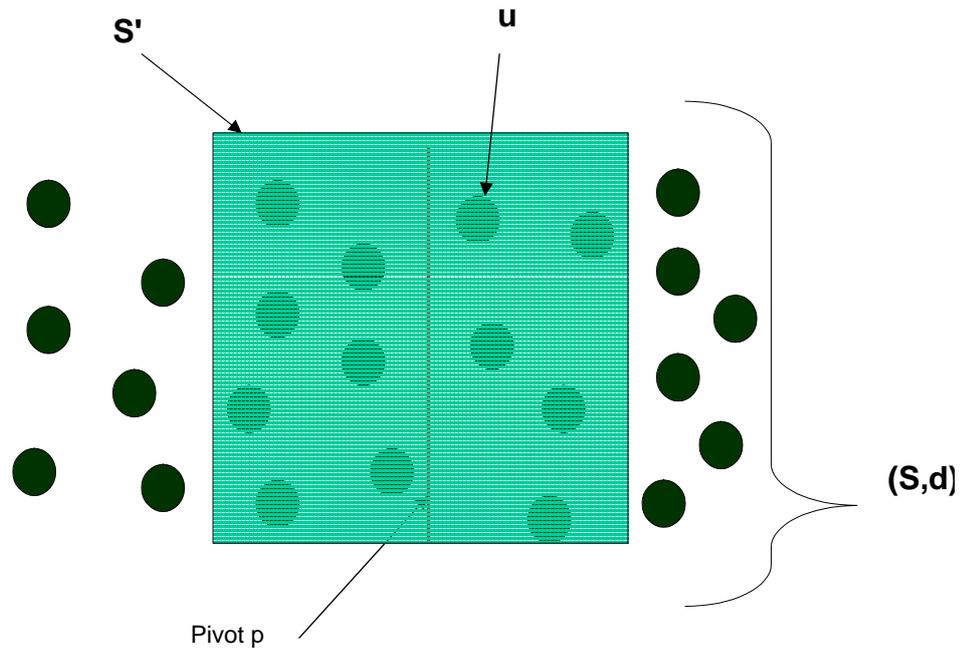


Figure 9 Example of a pivot in a metric space

By exploiting the triangular inequality property between the pivot, query and other objects in  $S$ , search spaces can be pruned and hence performance enhanced.

It is well known that pivot selection affects the performance of the algorithm. When considering two same-sized pivot sets, the set with the better chosen pivots will perform better. Also, a well chosen small set of pivots that require less space can perform as well as a much larger set of pivots. However, most methods choose pivots at random. Bustos (Bustos et al, 2001) nevertheless proposed a technique for selecting efficient pivot sets.

There are several algorithms that use pivots. They include Fixed-Queries Tree (FQT) (Baeza-Yates et al, 1994), Fixed Height Tree (Baeza-Yates et al, 1994), Vantage Point Tree (VPT) (Yianilos, 1993), Approximate Eliminating Search Algorithm (AESA) (Vidal, 1986), and Linear AESA (LEASA) (Micó et al, 1994).

### **2.2.2 Clustering**

Clustering is the process of grouping a set of objects into categories based on similarity. This similarity function can vary but normally based on cosine distance, Euclidean or a similar variant. Data clustering is a well researched field with many areas of application. Most recently, it has gained attention in the data mining and document search domains. However, a significant amount of research has also been performed in other areas such as image segmentation, object recognition, and computational biology. This reflects its broad appeal and effectiveness as an important step in data analysis.

#### **Approaches**

There are various approaches to data clustering. Figure 10 provides a taxonomy of these clustering approaches. There are other taxonomic clustering representations, but our description is based on (Jain et al, 1999). We will discuss them in the following sections.

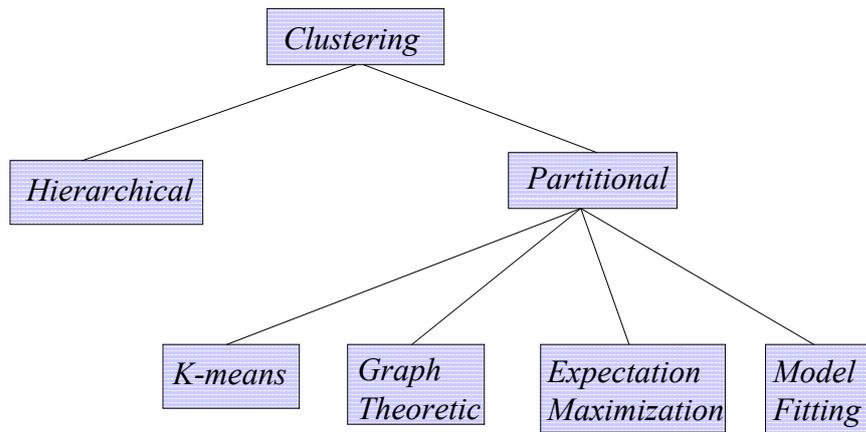


Figure 10 A clustering taxonomy

---

### **Hierarchical**

There are basically two approaches to clustering data: hierarchical and partitional clustering techniques. Hierarchical techniques produce a tree-like nested sequence of partitions that have singleton clusters as leaves and a signal all-encompassing cluster as the root. Each intermediate level is a combination of the nested clusters at the next lower level. This tree-like structure is called a dendrogram. There are two approaches to creating hierarchical clustering. The agglomerative approach starts with individual elements as clusters and progressively forms clusters by merging the most similar or closest pair of clusters. Contrarily, the divisive approach begins with one all-inclusive cluster and progressively divides until there are only singletons. See Figure 9. Agglomerative approaches are most common. The basic algorithm is as follows:

1. Compute the similarity between all pairs of clusters (hence a similarity matrix where the  $ij^{\text{th}}$  element holds the similarity measure between the  $i^{\text{th}}$  and  $j^{\text{th}}$  clusters);
2. Merge the most similar two clusters;
3. Update the similarity matrix to reflect the pair-wise similarity between the new cluster and the original clusters;
4. Repeat steps 2 and 3 until only 1 cluster remains (Steinbach et al, 2000).

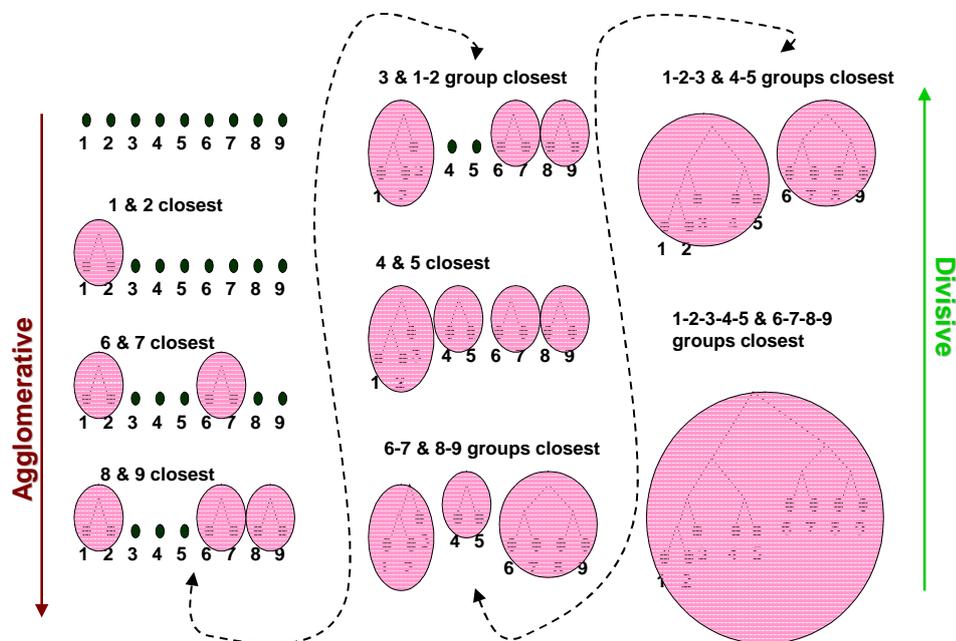


Figure 11 Hierarchical clustering

Zhoa and Karypis in (Zhoa and Karypis , 99) provided a survey of hierarchical clustering algorithms. They also presented an efficient technique, called constrained agglomerative, that employs both divisive and agglomerative features that faired well in terms of quality with larger datasets compared to standard techniques. Mandhani

(Mandhani et al, 2003) also proposed a hybrid technique employing a density-based, partitional-agglomerative technique with a document-word co-cluster. The partitioning step identified dense sub-matrices to partition the respective row set of the complete matrix. The hierarchical agglomerative step involves merging the similar sub-matrices to a predefined  $k$  cluster (for flat clusters) or a single complete matrix (for hierarchical clusters).

### **Partitional**

In contrast to the hierarchical techniques, partition clustering creates a single-level partitioning of datasets. There are several partition clustering techniques, but the most common is  $K$ -means which divides the data set into  $K$  clusters. To do so,  $K$  points are chosen that represent centroids. Through an iterative approach, the algorithm stabilizes on  $K$  clusters. A basic algorithm for  $K$ -means is as follows:

1. Select  $K$  points as initial centroids;
2. Assign all the points to the closest centroid using a distance function;
3. Recompute the centroid for each cluster;
4. Repeat steps 2 and 3 until the centroids stabilize (Steinbach et al, 2000).

Hammouda and Kamel in (Hammouda et al, 2003) proposed an incremental clustering algorithm based on maintaining cluster cohesiveness. They employed a cluster similarity histogram, a concise statistical representation of the pair-wise similarities within each document. Clusters needed to maintain a high cohesiveness as documents are added. The authors accomplished this by enabling the algorithm to reassign documents to clusters that were perhaps created after the documents were introduced. In effect, they presented a dynamic  $k$ -means approach to clustering.

In (Dhillon et al, 2003), the authors extend the  $k$ -means approach in a 2-step manner. First, for each current cluster, they execute a *first variation*, in which either a single or a string of document moves between clusters that will increase the overall clustering score. Second, they perform a standard spherical  $k$ -means iteration globally

to further increase the score. This dual iterative approach performed best in clustering high-dimensional and sparse text data.

Chávez et al in (Chávez et al, 2000a) presented a clustering technique to index metric spaces that recursively partitions the space into asymmetrical internal and external circular buckets, see Figure 11a. Given the asymmetry of the data structure they are able to prune the search space in two directions. First, one searches exhaustively inside the internal (I) bucket only if the query ball has some intersection with the search space ball  $c$ . Equally, if the query ball is totally contained in the search space ball  $c$ , there is no need to consider elements in the external (E) space, see Figure 11b. This asymmetrical structure and *double* pruning feature significantly increases the efficiency over standard pivot and clustering metric space techniques.

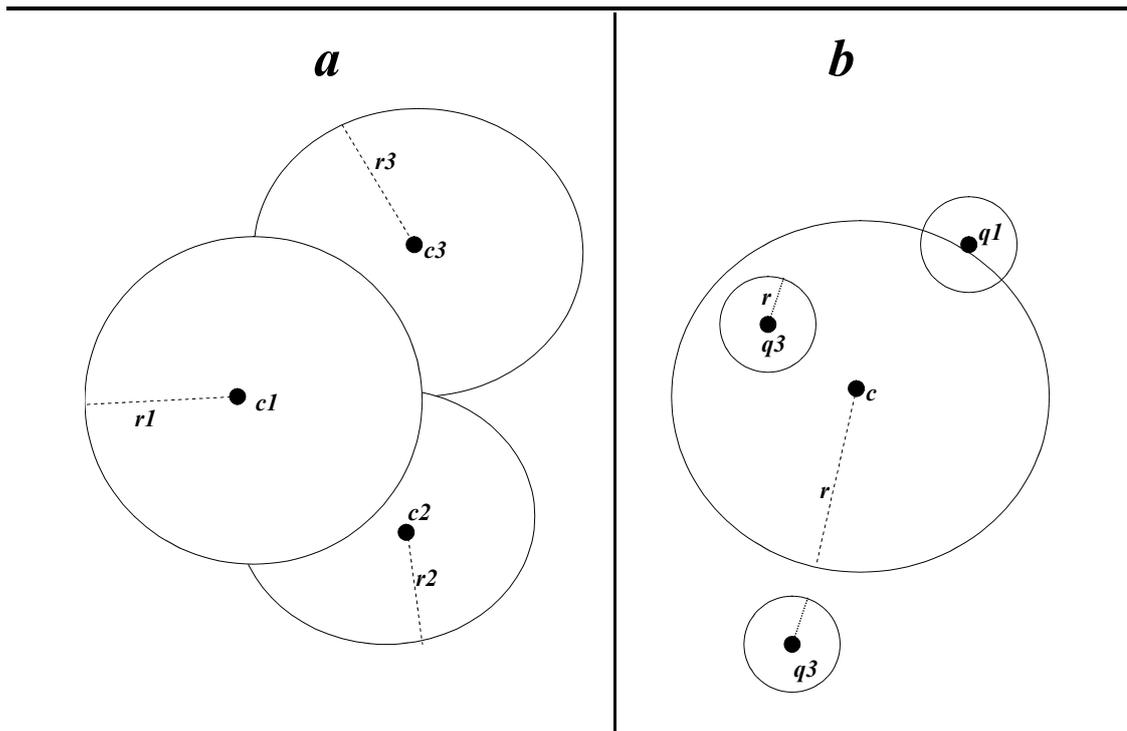


Figure 12 a. Recursive asymmetrical partitions b. search space and query balls

Zha, Ding and Gu in (Zha et al, 2001) presented a graph-theoretic partitioning scheme to data clustering. Their method uses an underlying bi-partite graph. The partition is constructed by minimizing a normalized sum of the edge weights between unmatched pairs of nodes in the bi-partite graph. They show that an approximate solution to the minimization problem can be obtained by computing the singular value decomposition (SVD).

Finally, Lin and Kondadadi in (Lin et al, 2001) introduced a soft clustering algorithm called SISC (SIMilarity-based Soft Clustering), that improves on the Expectation Maximum approach by using a similarity function instead. This frees the algorithm from relying on any underlying probability assumptions. In many cases it outperformed  $k$ -means approaches.

### **Our work**

As we have seen, proximity means different things to people depending on the domain. In this section we have discussed current research on proximity measures in

metric spaces. This approach for finding the similarity between objects is similar to our work because we also measure this proximity. However, we also consider the proximity of items related to the objects we are measuring. This distinction is what sets our approach apart from pure proximity calculations in metric spaces.

## 2.3 Recommender systems

Recommender systems traditionally deal with applications that have two entities, users and items. After obtaining an initial set of ratings, the recommender system tries to estimate the rating function  $R$

$$R : Users \times Items \rightarrow Ratings \quad [2.7]$$

for the (user, item) pairs that have not yet been rated by users (Adomavicius et al, 2005). Conceptually, once  $R$  is estimated for the entire  $Users \times Items$  space, a recommender system can select the item  $i'_u$  with the highest rating for a user  $u$  and recommend that item or items to the user:

$$\forall u \in Users, i'_u = \arg \max_{i \in Items} R(u, i) \quad [2.8]$$

Instead of estimating the unknown ratings from the entire  $Users \times Items$  space (which would be quite expensive) various methods have been developed to finding more efficient solutions requiring smaller computational efficiency such as (Goldberg et al, 2001). According to (Balabanovic and Shoham, 1997) the recommender system domain is divided into three areas: content-based, collaborative and hybrid.

### Content-Based Recommender Systems

The rating  $R(u, i)$  of item  $i$  for user  $u$  is normally determined based on ratings  $R(u, i')$  given by the same user  $u$  to other items  $i' \in Items$  that are similar to item  $i$  in terms of content (Adomavicius et al, 2005).

Content-based recommender systems have their short-comings. First they are limited to domains where content can be extracted automatically (Shardanand and Maes, 1995). Secondly, they suffer from the new-user problem. If the user has rated only a small number of items, the content-based recommender system may not

understand his preferences and suggest good recommendations (Adomavicius et al, 2005). Finally, since the user is being recommended only items that are similar to ones he has already highly rated, content-based recommender systems can suffer from ‘over-specialization.’ For example, the user could be recommended different articles about the same event in a news-feed system (Adomavicius, et al, 2005).

### Collaborative Recommender Systems

Collaborative recommender systems make their recommendations for a particular customer based on how other customers had previously rated the item. The rating  $R(u, i)$  of item  $I$  for user  $u$  is calculated by considering the ratings  $R(u', i)$  given by users  $u'$  who are similar to user  $u$ .

The similarity measure between  $u'$  and  $u$  is used as a weight for the ratings. The higher the similarity measure for user  $u'$ , the higher the weight value given. There have been several approaches measuring the similarity between users. The two most popular approaches are the correlation-based (Resnich et al, 1994) and (Shardanand and Maes, 1995):

$$sim(x, y) = \frac{\sum_{s \in S_{xy}} (r_{x,s} - \bar{r}_x)(r_{y,s} - \bar{r}_y)}{\sqrt{\sum_{s \in S_{xy}} (r_{x,s} - \bar{r})^2 \sum_{s \in S_{xy}} (r_{y,s} - \bar{r})^2}}, \quad [2.9]$$

and the cosine-based approach (Bresse et al, 1998) and (Sarwar et al, 2001):

$$sim(x, y) = \cos(X, Y) = \frac{X \bullet Y}{\|X\|_2 \times \|Y\|_2} = \frac{\sum_{s \in S_{xy}} r_{x,s} r_{y,s}}{\sqrt{\sum_{s \in S_{xy}} r_{x,s}^2} \sqrt{\sum_{s \in S_{xy}} r_{y,s}^2}}, \quad [2.10]$$

where  $X \bullet Y$  is the dot product of the rating vectors of the respective users.

Furthermore  $r_{x,s}$  and  $r_{y,s}$  are the ratings of item  $s$  of users  $x$  and  $y$  respectively,

$S_{xy} = \{s \in Items \mid r_{x,s} \neq \varepsilon \wedge r_{y,s} \neq \varepsilon\}$  is the set of all items rated by both users  $x$  and  $y$  (Adomavicius et al, 2005).

Collaborative recommender systems share the new user problem with content-based recommender systems. Collaborative systems also suffer from the new item

problem since they rely solely on rating data to make recommendations. Finally the lack of a sufficient number of ratings is another challenge for these types of rating systems.

### **Hybrid Recommender Systems**

Hybrid systems usually combine content and collaborative methods to overcome some of the weaknesses discussed in earlier sections. There are several approaches that have been proposed.

One technique for combining content and collaborative methods is by (1) learning and maintaining user profiles based on content analysis using content-based techniques or information retrieval methods and (2) then directly comparing the profiles to find similar users in order to make collaborative recommendations. The Fab system (Balabanovic and Shoham, 1997) and the ‘collaboration via content’ approach in (Pazzani, 1999) apply this hybrid technique.

Implementing separate collaborative and content-based recommender systems is another approach to building hybrid recommender systems. One can either combine the ratings from individual recommender systems into a final recommendation (Claypool et al, 1999) and (Pazzani, 1999) or we can use one of the recommender systems and at a given instance choosing to use the one that is better than others based on some recommendation quality matrix. For example (Tran and Cohen, 2000) choose the one that is more in line with the user’s past ratings.

A third hybrid approach uses the combined content-based and collaborative methods about both users and items in a single recommendation model. This approach was broached in (Condliff et al, 1999) and (Ansari et al, 2000) where they both used Bayesian mixed-effects regression models for parameter estimation and prediction.

Finally Adomavicius and Tuzhilin (Adomavicius and Tuzhilin, 2001a) proposed a multidimensional approach to recommendations which extended the traditional user/item paradigm. The additional dimensions capture the context in

which recommendations are made. This approach was proven more accurate in certain situations than 2-dimensional approaches in (Adomavicius et al, 2005).

### **Our work**

Our work is related to recommender systems in three ways. First, the Linea algorithm can be applied to the user similarity calculations for the traditional collaborative recommender systems. However, this use may not be the most interesting since these methods use a 2-dimensional approach that doesn't take into account other dimensions which is Linea's strong point.

Second, we can apply the correlation-based or cosine-based approaches suggested in the collaborative approach recommendation systems as local proximity implementations. These implementations not only would apply well to recommender systems but to any domain that can be represented as vectors.

Finally, our approach extends the present hybrid methods. When a user is given a recommendation, say for a book in an online bookstore scenario such as Amazon, it is the result of other users actually buying the recommended book. In set theory terms, the recommended items is always the result of an intersection of the set **A** of books that the user is planning to purchase and the set **B** of books that other users have purchased. The set of recommendations comes for  $\mathbf{B} - \mathbf{A}$ . In other words, for a book to be recommended, it has to have been purchased with another book that is in set **A**. However, there may be items that can not be ascertained based on an intersection. There may be items that are closely *related* to the item about to be purchased. Linea can take better advantage of the multi-dimensional data making up the context to provide a recommendation.

## **2.4 Conclusion**

As we have shown, our work is grounded in three research domains: link analysis, metric space distance algorithms and recommender system. We have

discussed each of these domains in detail and noted some of the important work in these areas.

With its foundation in library science, link analysis has become a major domain of research with the expansion of the internet. The PageRank and HITS algorithms and their derivatives are important advancements in this area. Our approach extends these domains, combining them in a generic way that allows our proximity calculations to be applied to many areas.

Proximity searching in metric space seeks to find close objects under an appropriate similarity function. We discussed current research in two broad sub-domains: pivot techniques and clustering.

Finally we discussed recommender systems which have found wide use on the internet. This domain is broken into three areas. Content-based systems provide recommendations based on ratings given by the same user to items that are similar to the item being recommended. While collaborative-based systems base their recommendations for an item based on the recommendations of other users who are similar to the user who is receiving the recommendation. Both approaches have their strengths and weakness. Hybrid systems, the third area, seek to combine facets of the previous two approaches in a way that minimizes their inherent weaknesses.

Our work extends the body of research in two ways. First we combine link analysis and proximity searching in a novel, generic way to determine the proximity between objects. Secondly, our work extends present hybrid models. Since our approach is multi-dimensional, we are able to determine similarity without having an intersection of sets. This flexibility allows users to find (and make recommendations for) items that are similar to other items that would not be apparent with current approaches.

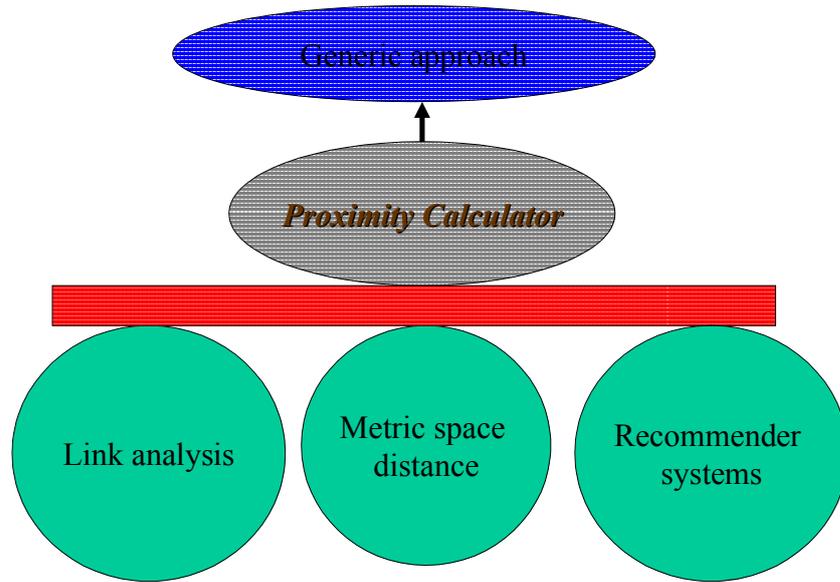


Figure 13 Our work

---

# Chapter 3

## Mathematical Preliminaries

The development of this thesis depends on both graph and set theory. In this chapter we will define key terms used throughout the work and then present the mathematical concepts that underpin the proximity framework.

### **3.1 Terminology**

To better communicate our approach, we provide definitions for terminology used in calculations throughout this work. A node represents a class of entities within a problem domain. Also, nodes contain elements, which are specific instances of the type of entity described by the node. Please note that the proximity,  $p(x, y)$ , is the proximity between elements  $x$  and  $y$ . See Figure 14.

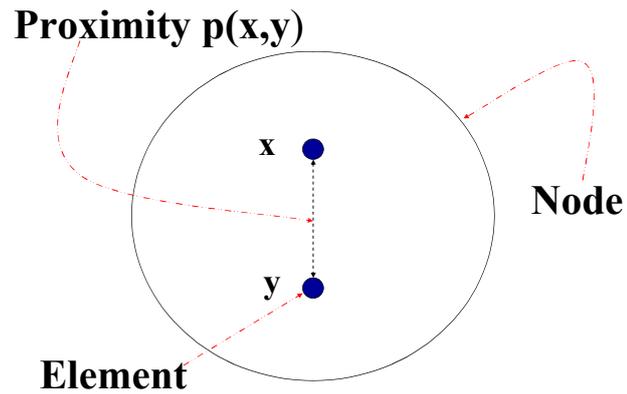


Figure 14 Node

---

We define an association as an affiliation between objects in the graphs. Furthermore there are two types of associations: relationships and links. A link is an association between elements. And a relationship is an association between nodes. We also define an image set. To do so, first consider Node A and Node B. Node A contains Elements  $x$  and  $y$ . Node B contains Elements  $t$  and  $u$ . Element  $x$  has a link to Elements  $t$  and  $u$ . Hence, we define the set that consists of  $t$  and  $u$  as the image set of  $x$  in Node B. See Figure 15.

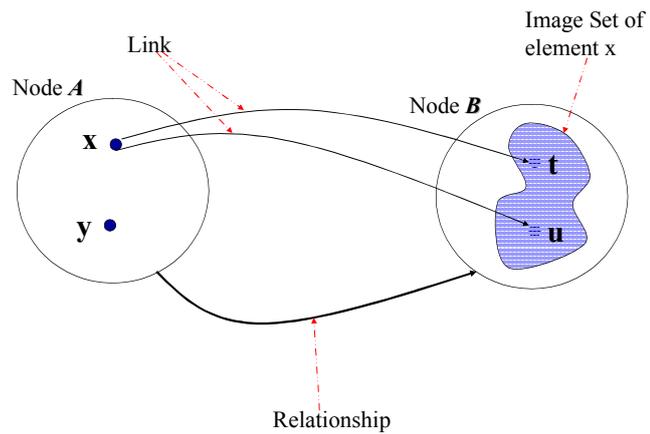


Figure 15 Relationships & links

---

A relationship can either be outbound or inbound. An inbound relationship is a directed association into the referenced node, and an outbound relationship is a directed association out of a referenced node. Finally, we define  $\Gamma(A)$  as the set of nodes associated with Node A via outbound relationships from Node A. See Figure 16.

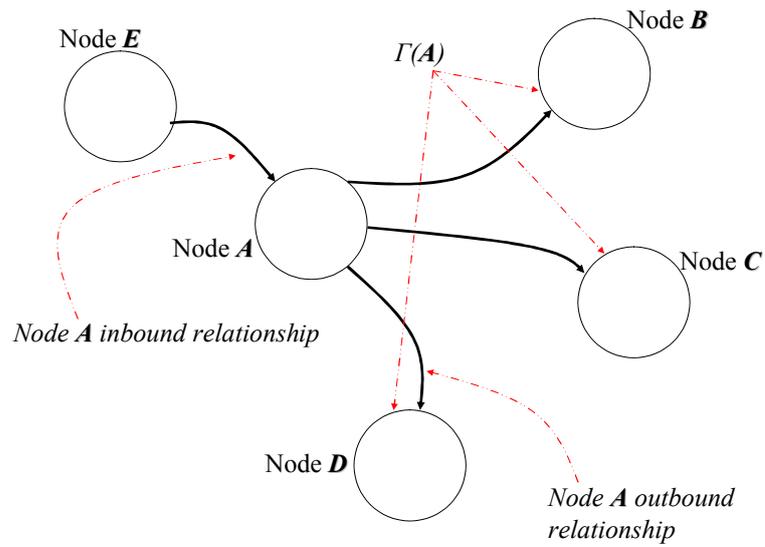


Figure 16 Inbound and outbound relationships

---

Considering the collection of nodes and relationships, we use a graph structure to represent problem domains for our proximity calculations. This allows us to exploit graph properties when implementing the proximity calculations. See Figure 17.

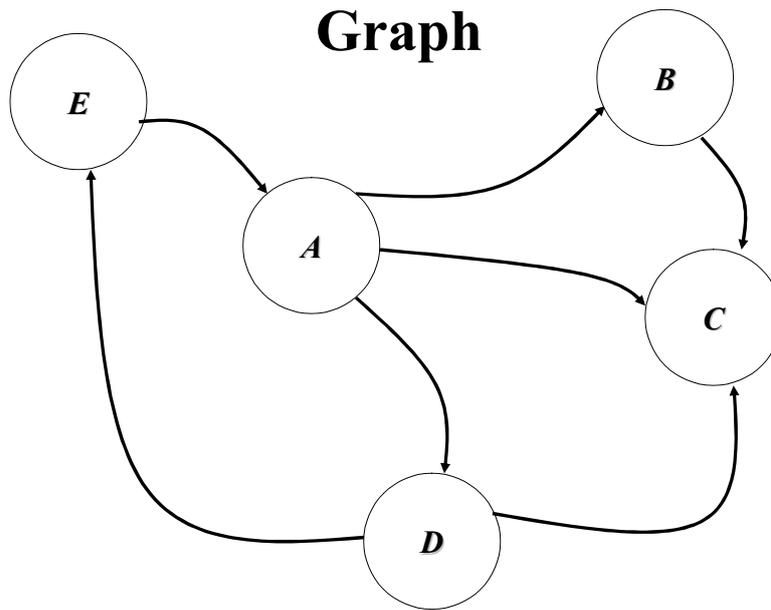


Figure 17 Graph representing a problem domain

---

### 3.2 Metric Space Distance Properties

Our main objective with this research centers on finding how close one object is to another. To perform this kind of measure, we want to operate in a space  $X$  that has the properties of metric spaces. First, the distance  $d$  between any two elements is at least zero. Negative distances cannot exist.

$$\forall x, y \in X, d(x, y) \geq 0 \quad [3.1]$$

We also assume a normalized distance between 0 and 1 since our sets will always be finite.

Second, the distance between any two elements must be symmetric:

$$\forall x, y \in X, d(x, y) = d(y, x) \quad [3.2]$$

Next, the metric space must support reflexivity:

$$\forall x \in X, d(x, x) = 0 \quad [3.3]$$

Finally, the distance within the metric space should adhere to the triangular inequality property:

$$\forall x, y, z \in X, d(x, y) \leq d(x, z) + d(z, y) \quad [3.4]$$

### 3.3 Metric Space Proximity Properties

The metric space properties we discussed apply to distances. Our work is with a normalized proximity. We define the relationship between a distance  $d$  and proximity  $p$  as follows:

$$p(x, y) = 1 - d(x, y) \quad [3.5]$$

We assume that the distance  $d$  is normalized. Furthermore, we define the following characteristics for a proximity  $p$ :

$$p(x, x) = 1 \quad [3.6]$$

$$\text{and } p(x, y) = 0, \text{ if } x \text{ or } y \text{ is an isolated element (see below).} \quad [3.7]$$

The proximity of an element to itself is one. In Equation [3.7], we assume that one of the elements is isolated. We define an isolated element as an element that has no relationship with any other values in the domain. We can consider it a null value. We therefore define the proximity between an isolated element and any other element as zero. Hence, the proximity approaches one the closer two objects are to each other. Additionally, proximity retains the same metric space properties as for a distance. The non-negative, symmetric and reflexivity properties follow from Equations [3.1], [3.2], [3.3], and [3.5].

$$0 \leq p(x, y) \leq 1, \text{ non-negativity} \quad [3.8]$$

$$p(x, y) = p(y, x), \text{ symmetry} \quad [3.9]$$

$$p(x, x) = 1, \text{ reflexivity} \quad [3.10]$$

We provide a more in-depth discussion for triangular inequality. Consider the triangular inequality for distance  $d$

$$\forall x, y, z \in X, d(x, y) \leq d(x, z) + d(z, y) \quad [3.11]$$

Now, we can substitute  $p(x,y)$  for  $d(x,y)$  according to the relationship we defined in Equation [3.5].

$$1 - p(x, y) \leq [(1 - p(x, z)) + (1 - p(z, y))] \quad [3.12]$$

We can algebraically rearrange the equation as follows:

$$1 - p(x, y) \leq (1 + 1) + (-1)[p(x, z) + p(z, y)] \quad [3.13]$$

$$-1 - p(x, y) \leq (-1)[(p(x, z) + p(z, y))] \quad [3.14]$$

$$1 + p(x, y) \geq p(x, z) + p(z, y) \quad [3.15]$$

Hence, we have the proximity property for triangular inequality

$$p(x, y) \geq p(x, z) + p(z, y) - 1 \quad [3.16]$$

which we apply to our normalized proximity. Let us consider the boundary values.

Consider that  $x$  and  $y$  are the same. Hence,  $p(x, y) = 1$ . If  $p(x, z) = 1$  and  $p(z, y) = 1$ , then the triangular inequality would still hold. Any other values of  $p(x, z)$  and  $p(z, y)$  would still hold since all values would be less than one. Consider the case when  $p(x, y) = 0$ . This implies that either  $x$  or  $y$  (or both) are isolated elements. Let  $x$  be the isolated element. Since  $x$  is an isolated element,  $p(x, z) = 0$  also. Then the values for the triangular inequality for proximity are

$$0 \geq 0 + \phi - 1 \quad [3.17]$$

where  $\phi$  is the value of  $p(z, y)$ . Since  $0 \geq \phi \geq 1$ , the above relation holds true.

### 3.4 Graph Properties

We represent the problem domain using a directed graph. Accordingly, we present some basic graph properties, along with definitions that support our algorithm.

Consider Graph  $G = \{V, E\}$ , where  $V$  is the set of vertices and  $E$  is the set of edges. Throughout this work we refer to vertices as nodes and edges as relationships. We do this for clarity as we have multiple things represented within one graph. This will become more apparent when we introduce the Linea agent in the next chapter. Within our domain,  $V$  represents the set of entities that make up the domain.

Likewise,  $E$  represents the set of relations between entities. Each node  $v \in V$  represents a class of objects in the domain, e.g. People from our example in Figure 1.

In our proximity algorithm, we work with the set  $\Gamma(v)$  of nodes that are linked to a node  $v$  via an outbound relationship from  $v$ . Hence, we define  $\Gamma(v)$  as follows:

$$\Gamma(v) = \{y \in V \mid (v, y) \in E\} \quad [3.18]$$

where  $(v, y)$  represents an outbound relationship from  $v$  to  $y$ .

Finally, consider two nodes  $v, z \in V$ . Let  $v$  contain the element  $x$ . Further, let Relationship  $t$  be the outbound relationship from  $v$  to  $z$ . Then we define the function  $r(x)$  as the set of elements in  $z$  that are associated with  $x \in v$ . We further define this set of elements in  $z$  as the image set of  $x$  in  $z$ . See Figure 18.

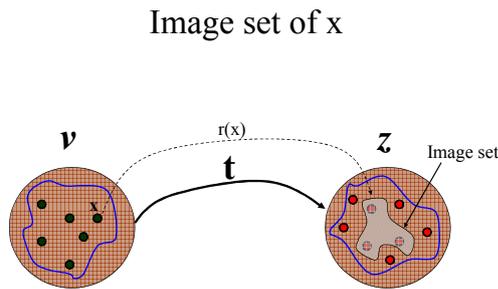


Figure 18 Image Set of  $x$

## 3.5 Similarity Measures

### 3.5.1 Finite number of attributes

The proximity used in the Linea calculator consists of two parts: the local proximity and the image proximity. The local proximity will vary with the domain to which the proximity calculator is applied. Our design goal was to enable this algorithm to be used easily in various domains. This calls for a plug and play

approach to the local proximity calculations. This suggests various approaches to measuring the local proximity. There is a large amount of research on similarity measures. We will discuss a few in this section as a means of illustrating the different areas in which the proximity calculator can be applied. Similarity and distance are inversely related; the greater the similarity, the smaller the distance. Hence we will use the terms interchangeably during our descriptions.

Probably the most well know distance measure is the Euclidean distance. It measures the shortest distance between two points. It is also referred to as the Standard metric:

$$d = \sqrt{\sum_{i=1}^n (x_i - y_i)^2}$$

Another distance measure is the City Block or Manhattan distance. It is named this way because in most American cities it is not possible to move from point *a* to point *b* in a straight line. Instead, it is necessary to follow the grid like city blocks.

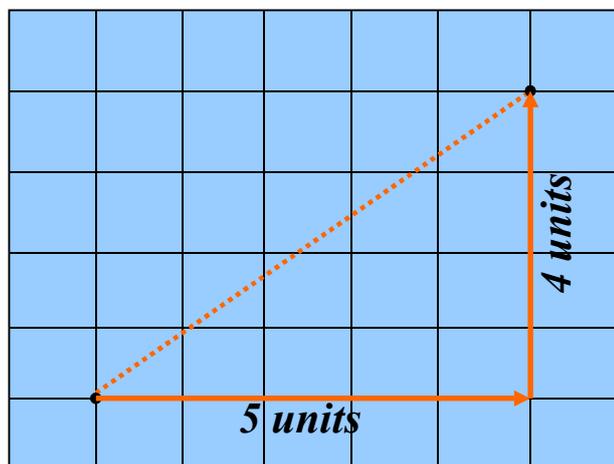


Figure 19 City Block distance

---

The equation for the City Block distance is as follows:

$$d = \sum_{i=1}^n |x_i - y_i| \quad [3.19]$$

It is best suited for measuring discrete data. Distances are measured between 2 points as if someone would move along city blocks.

The Chebyshev distance measures only the 2 elements from the two vectors (or sets) that are farthest apart.

$$d = \max_{1 \leq i \leq n} |x_i - y_i| \quad [3.20]$$

Hence, it measures the greatest distance between 2 vectors. This measure is very useful when computational efficiency is a priority.

The Minkowski distance can be seen as a type of meta-distance:

$$d = \left\{ \sum_{i=1}^n (x_i - y_i)^p \right\}^{1/p} \quad [3.21]$$

When  $p=1$ , it is the same as the City Block distance. When  $p=2$ , it is the Euclidian distance. As  $p$  increases, the metric approaches the Chebyshev distance. (Patel, 2004)

### 3.5.2 Relationships

In information retrieval, there are 5 common similarity measures (Van Rijsbergen, 2004). The simple matching coefficient measures the intersection between 2 sets of elements. It does not take into account the size of either set, thereby providing only a non-normalized result. There are 4 other coefficients (Overlap, Cosine, Jaccard and Dice) that are very similar.

<b><i>Simple Matching</i></b>	$ X \cap Y $	<i>Increasing penalty for non- intersecting elements</i> 
<b><i>Overlap</i></b>	$\frac{ X \cap Y }{\min( X ,  Y )}$	
<b><i>Cosine</i></b>	$\frac{ X \cap Y }{ X ^{1/2} *  Y ^{1/2}}$	
<b><i>Jaccard</i></b>	$\frac{ X \cap Y }{ X \cup Y }$	
<b><i>Dice</i></b>	$\frac{ X \cap Y }{ X  +  Y }$	

Figure 20 Similarity coefficients

The differences are the penalties they impose for non-intersecting elements. The Overlap coefficient is the most lenient while the Dice coefficient gives the highest penalty.

We also consider proximity distribution similarity measures. These types of similarity metrics are often applied to information retrieval. Given a document set with a probability distribution of words, we can compare two documents via their proximity distributions to determine the probability of them being related. One such metric is called the Jensen-Shannon divergence, which we can define as follows:

consider two probability distributions  $\mathbf{p}^{(1)} \equiv (p_1^{(1)}, p_2^{(1)}, \dots, p_k^{(1)})$  and  $\mathbf{p}^{(2)} \equiv (p_1^{(2)}, p_2^{(2)}, \dots, p_k^{(2)})$  that satisfy the constraints, listed in Figure 21:

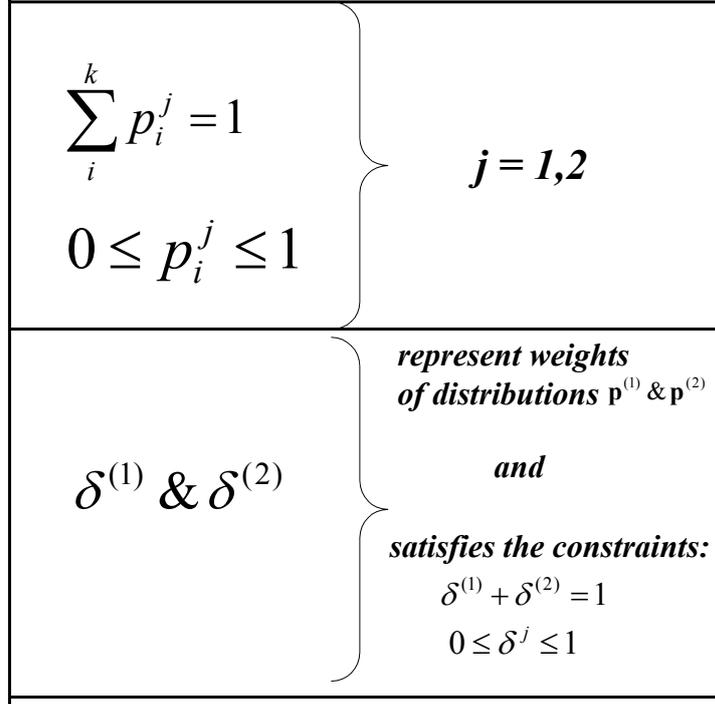


Figure 21 Probability distribution constraints

Then the Jensen-Shannon divergence is

$$D[\mathbf{p}^{(1)}, \mathbf{p}^{(2)}] = E[\delta^{(1)} \mathbf{p}^{(1)} + \delta^{(2)} \mathbf{p}^{(2)}] - (\delta^{(1)} E[\mathbf{p}^{(1)}] + \delta^{(2)} E[\mathbf{p}^{(2)}]) \quad [3.22]$$

where

$$E[\mathbf{p}] = -\sum_{i=1}^k p_i \log_2 p_i \quad [3.23]$$

is the Shannon entropy of the probability distribution  $\mathbf{p} \equiv (p_1, p_2, \dots, p_k)$ . (Grosse et al, 2002)

As we have described in this section, there are many options for measuring the local proximity. The type selected is based on the domain. This flexibility in local proximity measures lends significantly to the generic nature of the Linea agent.

### **3.5.3 Valued relationships**

This type of similarity measure occurs in domains where we place a value on a relationship. For example

--People movies and grading them

--Documents including words, each word appearing 0 or more times in every document

If we have such values associated with relationships, we may define similarities using the same formulas as in section 3.5.2 and also the correlation method defined in Chapter 2.

## **3.6 Conclusion**

We used this chapter to introduce the foundation concepts that we will build upon through the dissertation. In particular we introduced the terminology to be used throughout the remainder of the thesis. Next we discussed the basic distance properties then converted them to proximity relationships. We then discussed the key graph properties that support our work. We ended the chapter with a discussion of the various similarity measures. These similarity measures are concrete examples of local proximities that will be further explained in the following chapter.



# Chapter 4

## Linea Proximity Calculator

### 4.1 Introduction

The Linea proximity calculator that we propose consists of two parts. First, we determine the local proximity, which is calculated between two elements  $x$  and  $y$  within the same node  $\mathbf{v}$ . This calculation uses the attributes of the elements in question and is domain dependent. We discussed various approaches in the previous chapter. Second, the image proximity considers the proximity between the image sets of  $x$  and  $y$  for each node in  $\Gamma(\mathbf{v})$ . Finally, we manage the balance between the local and image proximities via a weight variable. The global proximity then, is the local proximity combined to the imaged proximities, taking into account a user-defined weight between the local and imaged proximity results. Hence, we define the Linea proximity calculator as

$$p^*_a(x, y) = (\delta_a)Local\ Proximity_a(x, y) + (1 - \delta_a)Imaged\ Proximity_a(x, y) \quad [4.1]$$

where  $p^*_a(x, y)$  is the global proximity between  $x$  and  $y$  elements in Node  $\mathbf{a}$ . In addition,  $\delta_a$  is a weight variable with a value between 0 and 1, inclusive.

We present and compare three approaches to implementing the Linea agent. They are the **naïve**, **bottom-up** and **iterative** variants. The difference between the naïve and iterative approaches centers on recursion. In the naïve approach we stop recursion artificially, whereas in the iterative method, it stabilizes naturally. The

bottom-up method is used for special case situations when the graph has no loops. In the following sections, we describe each of these approaches in more detail.

For each of the three approaches, the local proximity measures are the same. The choice of the local proximity algorithm is dictated by the domain. Please refer to Section 2.2 for a discussion of the state of the art proximity measures that could be applied. Section 3.4 discusses other approaches that can be local proximity measures that are more traditional.

## **4.2 Naïve Algorithm**

### **4.2.1 Definition**

The naïve approach recursively visits the nodes linked to the source node which contains the  $x$  and  $y$  elements for which we want to find the global proximity. The algorithm moves to each node following the image set links. As the algorithm functions within a graph structure with possible loops, we have to stop the recursion somewhere. This algorithm is based on a heuristic determination of the visited nodes. In short, if a node has been visited previously, we only consider the local proximities of the elements of that node. This spanning tree approach is illustrated in below.

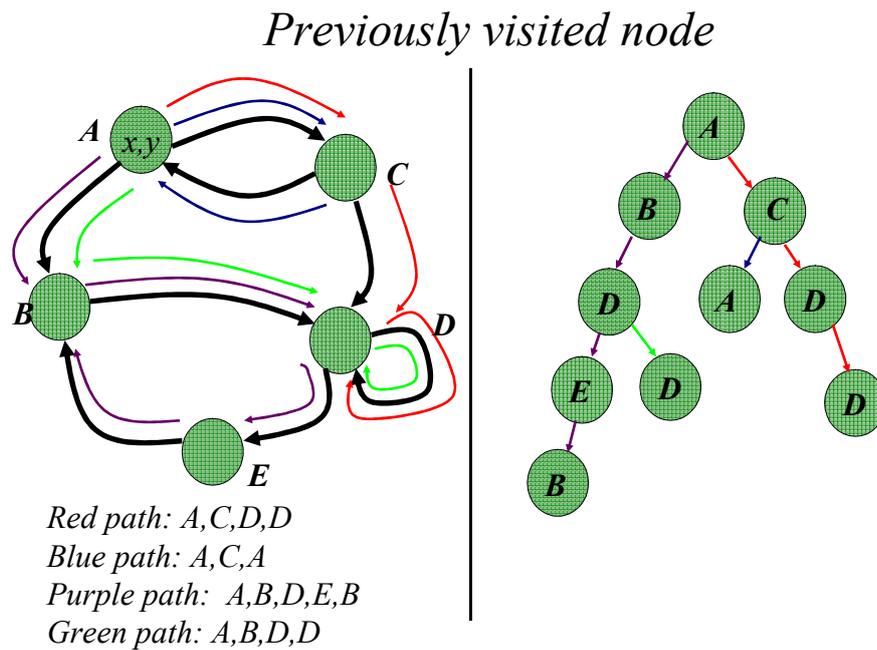


Figure 22 Spanning tree induced by previously visited nodes. Nodes which are already visited in a path do not recursively call any other node computation

The original algorithm is changed slightly to keep track of paths. We write the equation for the naïve approach as

$$p^*_{al}(x, y) = (\delta_a)Local\ Proximity_{al}(x, y) + (1 - \delta_a)Imaged\ Proximity_{al}(x, y) \quad [4.2]$$

where  $L$  represents the set of nodes already visited in the current path. This variable is used to implement the artificial fix point.

As discussed above,  $LocalProximity_{al}(x, y)$  will change by domain, but not by implementation approach. Accordingly, we will focus our attention on the implementation of the  $ImagedProximity$ . The two elements for which we are trying to determine their proximity are related to other elements in other nodes. As discussed earlier, we call these groups of elements image sets. The  $ImagedProximity$  considers the set distance between the image set of each of the two elements in which we are

trying to determine the proximity. Thus, we state that the elements  $x$  and  $y$  in Node  $\mathbf{a}$  have the following ImagedProximity:

$$\text{Im aged Pr oximity}_{\mathbf{aL}}(x, y) = \sum_{Z \in \Gamma(\mathbf{a})} \alpha_z [P_{ZL}^*(X, Y)] \quad [4.3]$$

where  $X$  and  $Y$  represent the image sets of  $x$  and  $y$  in node  $Z$  respectively and  $P_{ZL}^*(X, Y)$  calculates the set distance between the two image sets. We sum the image proximities for every node  $Z$  that is linked to  $\mathbf{a}$  via an outbound relationship, or every node  $Z \in \Gamma(\mathbf{a})$ . The variable  $\alpha_z$  is a weight function such that for any node  $\mathbf{v}$  in Graph  $G$ :

$$\sum_{Z \in \Gamma(\mathbf{v})} \alpha_z = 1 \quad [4.4]$$

The value of  $\alpha_z$  is domain dependent. It represents the relative importance of each node  $Z$  in  $\Gamma(\mathbf{v})$ . In our examples we assumed that each  $\alpha_z$  was equal. We further define  $P_{ZL}^*(X, Y)$  as follows:

$$P_{ZL}^*(X, Y) = \begin{cases} \frac{|X \cap Y|}{|X \cup Y|}, & \text{if } |\Gamma(Z)| = 0 \text{ or } Z \in L \\ \frac{1}{|X| + |Y|} Q_{ZL}(X, Y) \end{cases} \quad [4.5]$$

where

$$Q_{ZL}(X, Y) = \sum_{x \in X} \max_{y \in Y} \{p_{ZL+\{Z\}}^*(x, y)\} + \sum_{y \in Y} \max_{x \in X} \{p_{ZL+\{Z\}}^*(y, x)\} \quad [4.6]$$

If either the current node  $Z$  has no outbound relationships or the node has been visited previously in the algorithm, we calculate the set distance by dividing the cardinality of the intersection of the image sets by the cardinality of their union. Otherwise, we calculate the set distance by recursively calling the global proximity function for each pair of elements in the image sets. We also normalize this equation to the total number of elements in both sets.

## 4.2.2 A Visual Example

We provide the following example to illustrate how the naïve algorithm works. Although simple, the example illustrates all of the possible cases in our algorithm. We define a directed graph  $G = (V, E)$ . The nodes in  $V$  are  $a$ ,  $b$ ,  $c$ , and  $d$ . The relationships in  $E$  are  $q$ ,  $u$ ,  $t$ ,  $s$ ,  $w$ . Each node represents a class of objects. The actual instances are located in the set of elements within each node. Consider the elements  $x$  and  $y$  located in  $a$ . We want to determine their proximity to each other within this domain, which is described by the structure of the graph (see Figure 23).

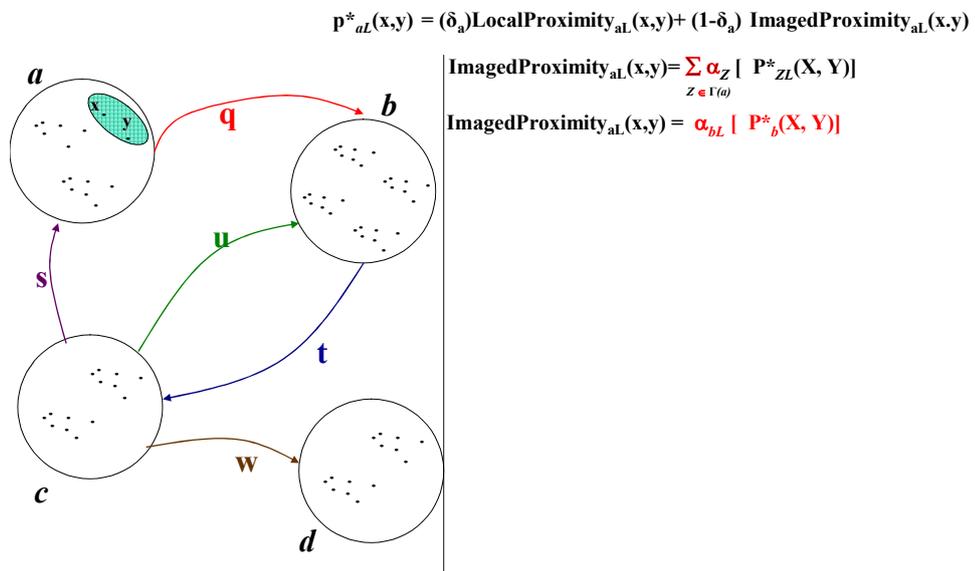


Figure 23 Detailed example –Node  $a$

---

For brevity, we omit the LocalProximity variable, as it is a trivial calculation and varies by domain. We also omit  $\delta_a$  and L updates for clarity. Hence, we focus our attention on the ImageProximity for this example. To calculate the ImagedProximity, we only consider the nodes that are linked to  $a$  via an outbound

relationship from Node **a**. Node **a** only has one outbound relationship, **q**, which links Node **a** to **b**. So we calculate the set distance between the image set of **x** that is in **b** and the image set of **y** that is in **b**. Since node **b** has outbound relationships and it has not been visited yet in the algorithm, we use the recursive equation to determine the set distance. In Figure 24 we assume that element **k** is in the image set of **x** and Element **h** is in the image set of **y**. We take these two elements as an example and recursively apply the global proximity function.

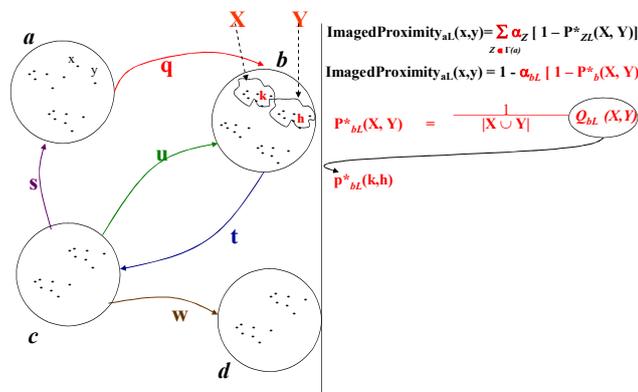


Figure 24 Detailed example-node **b**

We continue this process, systematically walking through the graph, following all outbound relationships from each node of interest. Hence, from Node **b** we would calculate the set distance in Node **c** between the image sets of Node **b**'s elements **k** and **h** (see Figure 25)

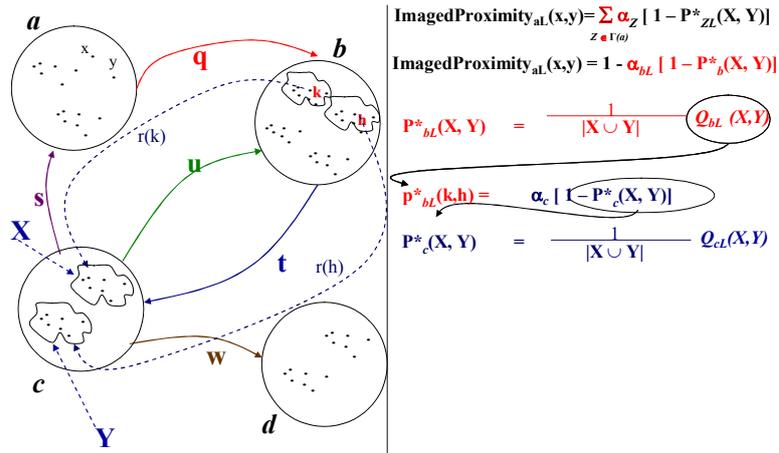


Figure 25 Detailed example-node **c**

Node **c** has three outbound relationships, **s**, **u**, and **w**; so we use the recursive set distance calculation for this calculation also. As we follow the relationships **s** and **u**, we return to Nodes **a** and **b** respectively which have already been visited. Hence, we calculate the set distance between the image sets of **m** and **n** (that are located in node **c**) by dividing the cardinality of the intersection of the sets by the cardinality of their union. We do this as an artificial fix point to this recursive algorithm. We perform the same type of calculation to determine the set distance of the image sets in Node **d**. However, the reason we use this case here is because Node **d** has no outbound relationships (see Figure 26).

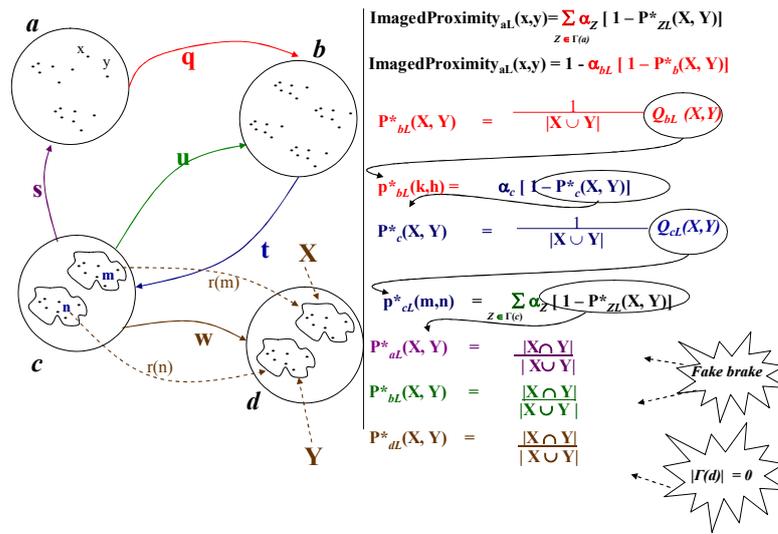


Figure 26 Detailed example-nodes *a*, *b* and *c*

### 4.2.3 The algorithm implementation

The naïve algorithm progresses through the graph structure beginning with the source node that contains the *x* and *y* elements that we desire to find the proximity. As it progresses through each node, it calculates both the local and imaged proximities. It determines the image proximity by comparing the proximity between the image sets of the *x* and *y* elements of the preceding node. During this image set calculation, we measure the proximities between each element in one image set to the each element in the other. It's during these calculations that we recursively call the global proximity procedure that moves us to the next Node *n* that is associated with the current node by an inbound directed association. The algorithm's fix point is artificial and we stop the fix point calculation when we arrive at a node with no outbound associations or if the node has already been visited for the current calculation path. We define a calculation path as the path followed through recursive calls from one node to another. Each outbound association from a Node *n* implies a unique path.

#### 4.2.3.1 Naïve algorithm helper procedure

The Calculate Global Proximity (CGP) procedure uses a helper procedure we call Calculate Raw Proximity (CRP). The CRP procedure houses the actual recursive call to CGP. Since the steps executed in CRP are called multiple times in CGP, we put them into a separate procedure. Hence, we begin our discussion with CRP.

```
CRP(iSet1, iSet2, graph, alpha)
1 iSetMaxProximity  $\leftarrow$  0
2 iSetProximity  $\leftarrow$  0
3 1To2iProximity  $\leftarrow$  0
4 for each element e1 in iSet1
5   for each element e2 in iSet2
6     iSetProximity  $\leftarrow$  CGP(e1, e2, graph, alpha)
7     if iSetProximity > iSetMaxProximity
8       iSetMaxProximity  $\leftarrow$  iSetProximity
9   1To2iProximity  $\leftarrow$  1To2iProximity + iSetMaxProximity
10 iSetMaxProximity  $\leftarrow$  0
11 return 1To2iProximity
```

Lines 1-3 initialize the internal variables used to keep track of the intermediate image proximities between each pair of image sets and the eventual final image proximity that is returned. The main body of the procedure is located in lines 5 – 10 where we loop through the set of elements in each image set to measure the proximity between each pair. After recursively calling the CGP procedure in Line 6 (which we will discuss next), the result is compared to the current maximum proximity value for the current element in the first image set. After looping through all of the elements in image set 2 (iSet2), the current maximum proximity value (iSetMaxProximity) is added to the total image proximity (1To2iProximity) in Line 9. This procedure, as we shall see shortly, is called twice in the CGP procedure for the same pair of image sets, but the order is changed. Hence iSet1 is compared to iSet2. And then iSet2 is compared to iSet1. This is done to implement the algorithm.

#### 4.2.3.2 Naïve algorithm main procedure

The naïve algorithm main procedure, CGP, calculates the global proximity between the elements  $x$  and  $y$  located in node  $n$ . It does so by recursively calculating the image set proximities and global proximities in nodes  $n' \in \Gamma(n)$ . The procedure continues until each path that is followed arrives at an artificial fix point.

```
CGP(x, y, graph, alpha)
1 globalProximity  $\leftarrow$  0
2 imagedProximity  $\leftarrow$  0
3 localProximity  $\leftarrow$  0
4 currentImagedProximity  $\leftarrow$  0
5 for each node  $n$  in  $\text{gamma}(\text{currentNode})$ 
6   xImageSet  $\leftarrow$  getImageSet(x,  $n$ )
7   yImageSet  $\leftarrow$  getImageSet(y,  $n$ )
8   if  $|xImageSet| = 0$  and  $|yImageSet| = 0$ 
9     currentImagedProximity  $\leftarrow$  1
10  else if  $|xImageSet| = 0$  or  $|yImageSet| = 0$ 
11    continue
12  else
13    if node  $n$  has already been visited or  $|\text{gamma}(n)| = 0$ 
14      currentImagedProximity  $\leftarrow$  calculateSetDistance(xImageSet,
yImageSet)
15    else
16      xToxIP  $\leftarrow$  CRP(xImageSet, yImageSet, graph, alpha)
17      yToxIP  $\leftarrow$  CRP(yImageSet, xImageSet, graph, alpha)
18      currentImagedProximity  $\leftarrow$  (xToxIP + yToxIP) / (xToxIP + yToxIP)
19      currentImagedProximity  $\leftarrow$  currentImagedProximity * weight
20      imagedProximity  $\leftarrow$  imagedProximity + currentImagedProximity
21      currentImagedProximity  $\leftarrow$  0
22 globalProximity  $\leftarrow$  alpha * localProximity + (1-alpha)*imagedProximity
```

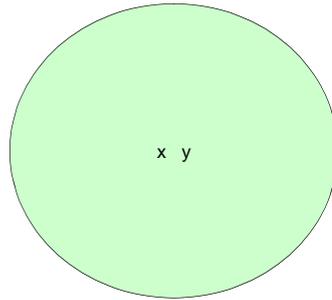
After initializing the various proximity variables used in the algorithm, the CGP procedure loops through all of the nodes  $n$  in  $\Gamma(\text{currentNode})$ . The `currentNode` variable represents the node where the elements  $x$  and  $y$  reside. For each

image set pair, it first checks the size of both sets. If both image sets are empty and the local proximity is 1, we set the `currentImageProximity` value to 1. If both image sets are empty and the local proximity is any value less than 1 then we set the `currentImageProximity` value to 0. If only one image set is empty, we continue, leaving the `currentImageProximity` value initialized to zero, since the `currentImageProximity` is reset to zero after each iteration of the loop (Line 21). If the node  $n$  has previously been visited on this calculation path or  $|\Gamma(n)|=0$ , then in Line 14 we implement the artificial fix point calculation for determining the image proximity. If all of the previous **if** statements are false, then we make the recursive call to the CGP procedure for each image set in Lines 16 and 17. In Lines 18 and 19 we normalize the results and factor in the weight value for the current node. The last calculation in the loop at Line 20 is to add the current image proximity value to the running total. The CGP procedure ends by calculating the global proximity using the totaled imaged proximity, local proximity and alpha value.

We define a path as the sequence of nodes visited by the Linea algorithm until it reaches either a previously visited node or a node  $n$  where  $\Gamma(n) = 0$ . Normally, a domain graph is contains multiple paths. The order of the paths that Linea follows does not affect the final score. We provide empirical evidence in Chapter 5 to support this claim.

#### **4.2.4 Time complexity**

In order to estimate the time complexity of the proximity algorithm, we first consider the base case. If there are no outbound relationships, this means there are no image sets. Accordingly the only calculation would be the local proximity, which would be constant  $C$ .



$$O(x, y) = C$$

Figure 27 Base case

---

In order to consider the follow-on cases, we must first look closer at how the algorithm progresses. The proximity calculator works on a graph. It is a recursive algorithm that follows the outbound relationships from the original node (that has the  $x$  and  $y$  elements) outward. The algorithm stops in 2 cases: 1) when it reaches a node that has already been visited on the current path, or 2) when it reaches a node that has no outbound relationships. Given these two induced termination cases, any graph can be represented as a tree. For example, Figure 22 shows a tree induced from a graph due to previously visited nodes.

Similarly, Figure 28 shows a partial tree induced by a node that has no outbound relationships.

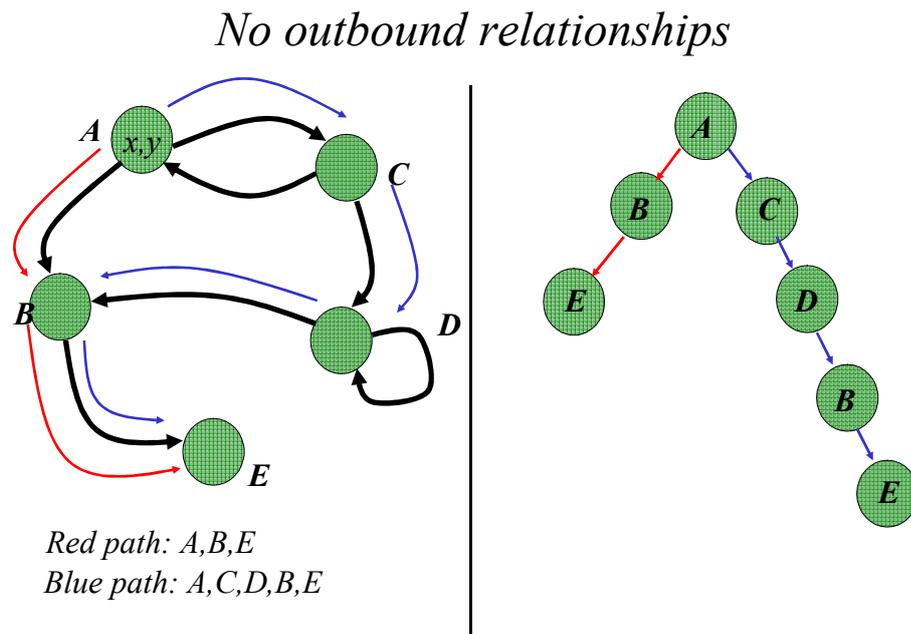


Figure 28 Tree induced by a node with no outbound relationships

With these two cases, any graph can be represented as a tree because eventually, as the algorithm follows the outbound relationships of the nodes in a graph, it will either reach a node it has previously visited, or it will reach a node that has no outbound edges. We use this tree characteristic of our graph to determine the time complexity upper-bound.

There are three variables that influence the time complexity of the algorithm. First, consider  $R$ , which represents the number of outbound relationships from the node that has the most outbound relationships in the graph. Then, let  $n$  equal the number of elements in the node that has the most elements in the graph. Finally let  $h$  be the height of the tree induced from the graph. As we shall show in Chapter 5, the order of the paths that the algorithm follows has no effect on the end result. Accordingly, the height  $h$  is simply the longest path in the graph. We define  $R$  and  $n$  as the worst case to ensure we capture the upper-bound of our algorithm complexity.

In the base case we described earlier, there were no outbound relationships, hence  $R$  was zero. Now, let us consider the case when  $R=2$  and the height  $h$  of our induced tree is 1. In the base case with node outbound relationships ( $R=0$ ), the time complexity is a constant  $c$ . This represents the calculation of the local proximity. As we follow the outbound relationships, we must perform the image proximity calculations. Let's consider Node B. According to the proximity calculator algorithm, we would consider the image sets for both  $x$  and  $y$  that are in Node B. To capture the upper bound, we assume that the size of the image set of both  $x$  and  $y$  in B is  $n$ . Hence we would perform  $2n^2c$  calculations in Node B to calculate both the local proximity at constant time  $c$  and the image proximity. We would repeat this calculation for each outbound relationship from Node A. Hence, since  $R=2$ , the time complexity when  $h=1$  is  $2n^2Rc+c$ . See Figure 29.

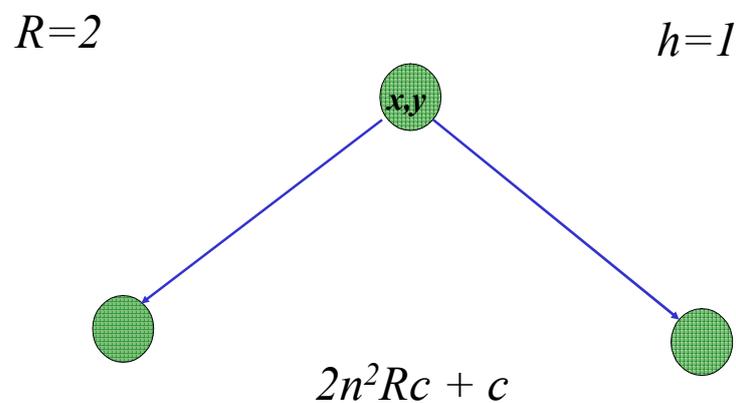


Figure 29 Time complexity when  $R=2$  and  $h=1$

---

Now let's consider the follow-on cases in order to detect a defining time complexity equation. Assume we have now progressed in the induced tree to the  $h=2$  level. At level  $h=1$ , we made  $2n^2Rc+c$  calculations. Each of those calculations included a recursive call to the proximity calculator and the local proximity calculation at a cost of constant  $c$  time. This involves the calculation of the image sets in  $\Gamma(v)$  where  $v$  represents a node at level  $h=1$ . All nodes in  $\Gamma(v)$  will be located in level  $h=2$ . So if we consider an arbitrary node in  $\Gamma(v)$ , named  $v_1$ , then the image set calculation number will be as follows:

$$(2n^2)^2 R^2 c \quad [4.7]$$

To perform the image set calculations in any node, it takes  $2n^2$  calculations. However, at level  $h=2$ , this calculation will be performed  $2n^2$  times due to the recursive calls from level  $h=1$ . Furthermore, at level  $h=2$ , there are a total of  $R^2$  outbound relationships from level  $h=1$ . This is a progressive calculation. Accordingly, we must add this calculation to the previous one for level  $h=1$ . Hence you would have the following:

$$(2n^2)^2 R^2 c + 2n^2 R c + c \quad [4.8]$$

See Figure 30 for a visual representation of the equation at level  $h=2$ .

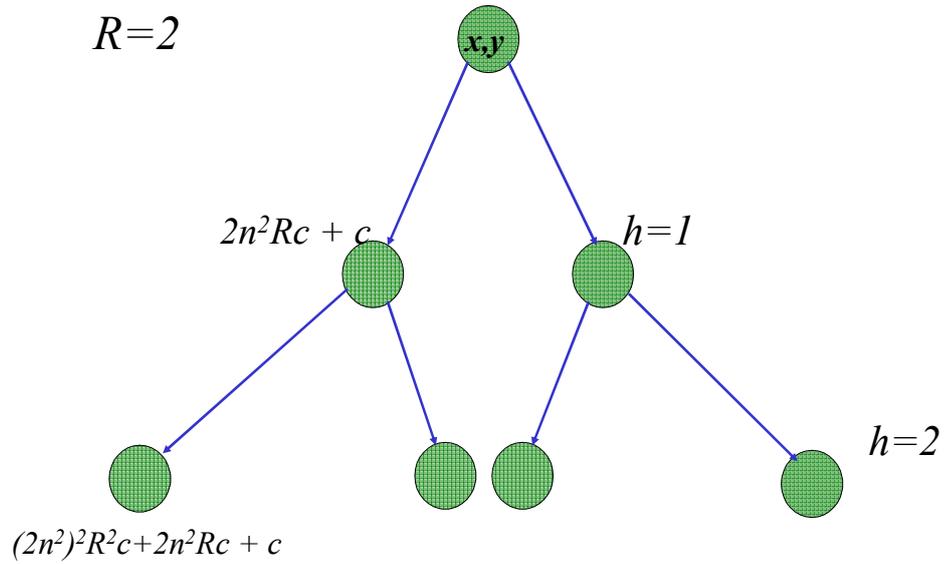


Figure 30 Time complexity at  $h=2$

---

We continue this process for an additional step. At level  $h=3$ , there would be 8 nodes added. See Figure 31. Hence we have

$$(2n^2)^3 R^3 c + (2n^2)^2 R^2 c + 2n^2 R c + c \quad [4.9]$$

which now takes into consideration this new level. Similar to our calculations at level  $h=2$ , at  $h=3$ , we must make the image set calculations  $2n^2$  times for each node.

However, we must do this  $(2n^2)^2$  times due to the recursive calls from level  $h=2$ .

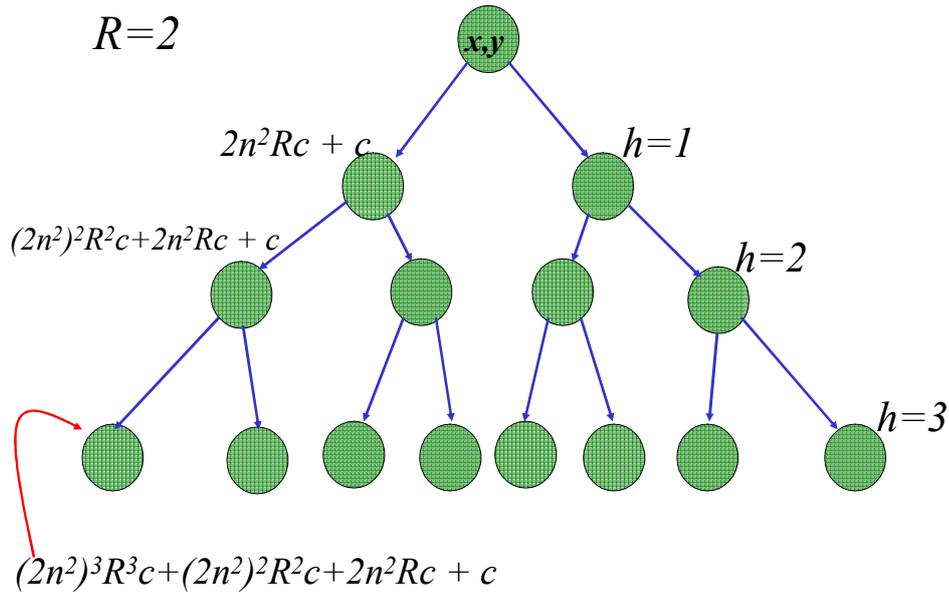


Figure 31 Time complexity at h=3

---

At this point, we can begin to see a pattern develop. The height  $h$  of our induced tree is related to the equation that describes the time complexity. For example at  $h=3$ , the equation that represents the time complexity can be rewritten in terms of  $h$ .

$$(2n^2)^h R^h c + (2n^2)^{h-1} R^{h-1} c + (2n^2)^{h-2} R^{h-2} c + c \quad [4.10]$$

The exponents for the  $(2n^2)$  and  $R$  factors are tied to the value of  $h$ . Specifically, the height  $h$  of the tree defines the largest exponent value for these 2 factors. Hence if  $h=2$ , then the highest exponent values would be 2 or  $(2n^2)^2 R^2 c$ . Likewise, if  $h=5$  then the highest exponent value would be 5 or  $(2n^2)^5 R^5 c$ . We can rearrange equation [4.10] by factoring out the constant  $c$ . We then have:

$$\left( (2n^2 R)^h + (2n^2 R)^{h-1} + (2n^2 R)^{h-2} + 1 \right) c \quad [4.11]$$

Accordingly, we can generalize this series to

$$\left( (2n^2 R)^h + (2n^2 R)^{h-1} + \dots + (2n^2 R)^1 + 1 \right) c \quad [4.12]$$

which describes the complexity for any height  $h$ . This series can be now rewritten as the summation

$$\frac{1}{c} \sum_{m=1}^h (2n^2R)^m = (2n^2R)^h + (2n^2R)^{h-1} + \dots + (2n^2R)^1 + 1 \quad [4.13]$$

for values 1 through  $h$ . The constant  $c$  represents the value for  $h=0$ . This well known summation has the following value:

$$\frac{1}{c} \sum_{m=1}^h (2n^2R)^m = \frac{(2n^2R)^{h+1} - 1}{2n^2R - 1} \quad [4.14]$$

Hence, we state the time complexity for the recursive approach to the proximity calculator as follows:

$$T(n, h, R) = \left( \frac{(2n^2R)^{h+1} - 1}{2n^2R - 1} \right) c \quad [4.15]$$

or

$$O([n^2R]^h) \quad [4.16]$$

This is a worst case estimation. Given a graph, we chose  $n$  as the number of elements in the node with the largest number of elements. We then assumed that each node has this same number,  $n$ , of elements. Similarly, we defined  $R$  as the number of outbound edges from the node with the most outbound edges. We then applied this value for each node in the graph. The induced tree has a height  $h$ . Within the context of the graph,  $h$  is the longest path in the graph. We assumed that every path length in the graph was the same for as the longest path. Within the context of the induced tree, that means that the induced tree is a complete ‘ $R$ -ary’ tree in which all leaves have the same depth and all internal nodes have degree  $R$ .

This covers the worst case for our time complexity. In many cases, this value will be much larger than in actuality. However, the time complexity depends on the height, degree and number of elements in the node. We are unaware of a way to get a closer approximation of the time complexity.

## 4.3 Bottom-up

The bottom-up approach works on a modified tree version of the domain graph. This modified tree is developed either by using the artificial fix point method from the naïve approach (section 4.2) or from directed graphs with no loops in them. In this latter approach, we arrange the graph into a modified tree (explained later) with nodes  $\mathbf{n}$  with  $|\Gamma(\mathbf{n})| = 0$  as leaves. Once the graph has been converted to a modified tree, we pre-process the image set calculations and hence improve the efficiency of the algorithm.

### 4.3.1 Proximity of an element to an image set

In order to improve the speed of the algorithm, we do not directly compute the proximities of image sets, but we first compute the proximities of elements to image sets. Before we continue, we shall provide a couple of definitions.

$S$ : is the image set, located in Node  $X$ , of  $u \in Z$

$\Pi_X(x, S)$ : is the proximity of an element  $x \in X$  to the image set  $S$

For instance, consider node  $X$  and parent nodes  $Y$  and  $Z$  which have relationships towards Node  $X$ . Given any element  $x \in X$ , and given any image set  $S$ , we compute the proximity

$$\Pi_X(x, S) = \max_{t \in S} (p_X^*(x, t)) \quad [4.17]$$

We take the maximum of the global proximities,  $p_X^*(x, t)$  between the given element  $x$  and each element  $t$  in the image set  $S$ .

Another way to look at the proximities between image sets and elements is to consider the element of origin of the image set. For example  $S$ , located in Node  $X$ , is the image set of Element  $u$  which is located in Node  $Z$ . Hence, we wish to define the proximities between  $S$  and elements in  $X$  from the perspective of  $u$ . Accordingly, consider

$$\Pi_{XZ}(x, u) = \Pi_X(x, \text{ImageSet}(u)) \quad [4.18]$$

which captures this representation. The subscript X is the node where the image set resides. Whereas the second subscript, Z, represents the source node of the image set found in Node X. Next, x is the element in Node X and u represents the source element located in Node Z of the image set located in Node X.

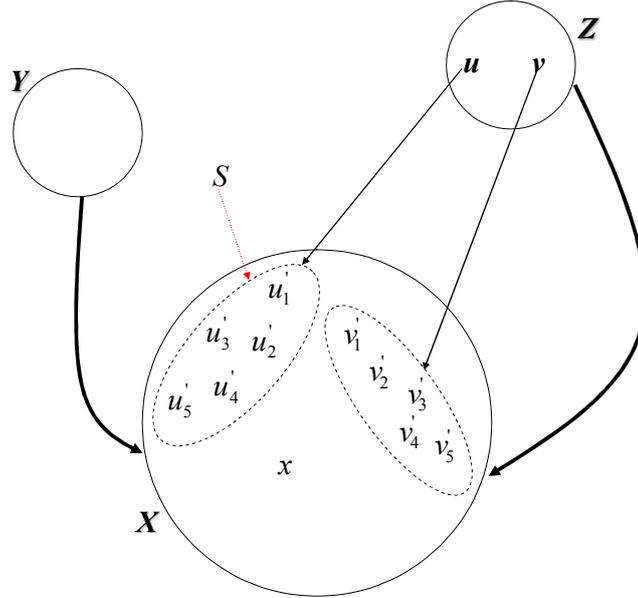


Figure 32 Computing the distance of any element x of X to image sets of all elements u of Y or Z

### 4.3.2 Proximity of 2 image sets

After computing the proximities of the image sets of elements u and v of node Z to any element in the child node, it is easy to compute the proximities of image sets of elements u and v : those are given by :

$$p_{image_X}^Z(U, V) = \frac{1}{|U| + |V|} \left[ \sum_{u' \in U} \Pi_X(u', V) + \sum_{v' \in V} \Pi_X(v', U) \right] \quad [4.19]$$

The two image sets U and V that are located in Node X. They are image sets of a pair of elements located in Node Z. To find the proximity between these two

image sets, we sum the proximities between each element  $u'$  in the image set  $U$  and image set  $V$ . Similarly, we also sum the proximities between each element  $v'$  in the image set  $V$  and the image set  $U$ . See Figure 33 below. After adding the two summations, we normalize the result by dividing by the total number of elements in each image set.

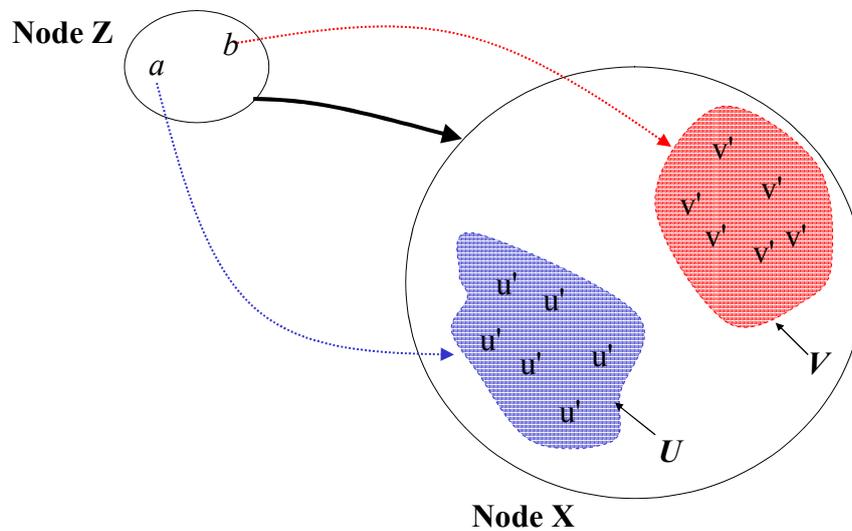


Figure 33 Elements of an image set proximity measure

### 4.3.3 The algorithm

As mentioned in the beginning of this section, we convert the given graph into a modified tree. For cyclic graphs we convert them into trees using either nodes with no outbound edges or nodes that have already been visited as leaves. In the case of a directed acyclic graph  $G$ , we can transform  $G$  into a modified tree by only considering nodes with no outbound edges. So in effect, acyclic graph conversion is a special

case of the naïve approach conversion method. We call it modified because children nodes are allowed to have multiple parents, such as the case for Node E in Figure 28.

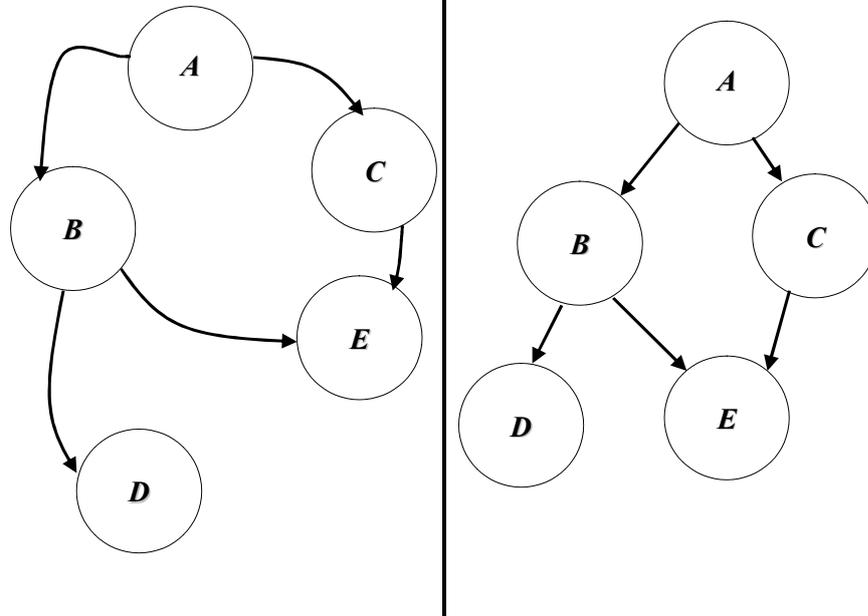


Figure 34 Directed graph with no loops converted into modified tree

The bottom-up algorithm follows two phases. Phase 1 collects image sets for each node and moves in a downward direction. It begins at the source node, the node that has the original 2 elements for which we want to find the proximity. Phase 2 then starts at the leaves and calculates proximities between elements and image sets in an upward direction until the source node.

Fore each couple of nodes A and B, where  $B \in \Gamma[A]$ , phase 2 includes 2 steps:

- Step 1: for each element  $b \in B$  and for each element  $a \in A$ ,
  - Compute  $\Pi_{BA}(b, a)$  according to equations [4.17] and [4.18]
- Step 2: for each couple of elements  $(a, a')$  with  $a \in A$ 
  - Compute  $p_{image_B^A}(\text{ImageSet}(a), \text{ImageSet}(a'))$ , using [4.19]

The algorithm that implements Phase 1, collecting image sets for all elements, is named  $\text{CollectIS}(n)$  and takes a Node  $n$ . This algorithm employs a data structure  $\text{IS}(p,e)$  that holds image sets of individual elements.  $\text{IS}$  is accessible in Phase 1 and Phase 2. Finally, we use the function  $\Gamma(n)$  that returns the set of nodes that have an inbound relationship from Node  $n$ . Hence the algorithm is stated as

```

CollectIS( n)
1  for each  $p \in \Gamma(n)$ 
2    for  $e \in n$ 
3      compute  $\text{IS}(p,e)$ 
4    if not examined( $p$ )
5      CollectIS( $p$ )
6  examined( $n$ ) = true

```

This algorithm is called recursively starting with the top node:

$\text{CollectIS}(\text{TopNode})$ . The algorithm then ends when we reach leaf nodes or nodes that have already been examined.

The goal of Phase 2 is to use the image sets collected in Phase 1 to calculate the global proximity between two elements. We accomplish this by starting at the bottom of the modified tree and working upwards. As in Phase 1 above, we employ supporting data structures in this phase. A Proximity Element Image Set (PEI) table is used to globally keep track of element to image set proximity measures. See equation [4.17]. This table also holds the element node identifying information.

#### 4.3.4 Time complexity

Let  $N_1, N_2, \dots, N_k$  be the  $k$  nodes in graph  $G$ . Furthermore, let  $n_1, n_2, \dots, n_k$  their sizes. The total number of elements of the graph is  $n = n_1 + n_2 + \dots + n_k$ . The time required for ordering the nodes in a bottom-up manner is a constant  $O(c)$ .

Consider now step 1 of the algorithm for some upper node  $N_i$  and some lower node (son)  $N_j$ . Remember that step 1 applies equation [4.17]. Computing the proximity of an element of  $N_j$  to the image set of an element of  $N_i$  takes at most  $n_j$

operations. Doing this for each element of  $N_i$  and for each element of  $N_j$  takes at most  $n_i * n_j^2$  operations.

Consider now step 2 of the algorithm. Computing the proximity of 2 image sets of  $N_i$  requires at most  $n_{j_1} + n_{j_2} + \dots + n_{j_h}$  operations, where  $h$  is the number of child nodes of  $N_i$  and  $j_1, j_2, \dots, j_h$  are the sons of  $N_i$ . For each couple of nodes  $i, j$  where  $i$  is the parent of child node  $j$ , the time component is  $n_j$ . If we do this operation for any 2 nodes of  $N_i$ , we get  $n_i^2 * n_j$  operations. To summarize, the time of the algorithm is the sum of  $(n_i^2 n_j + n_j^2 n_i)$  for any relationship  $(i, j)$  where  $i$  designates the parent node of the relationship and  $j$  the child node.

We may roughly say that  $O(n^3)$  is a higher bound of the algorithm, which may be considered better than the recursive algorithm of section 4.2, but which may be improved in practice because of the sparse data available.

Take for instance a collaborative filtering problem like the one given on the site <http://www.grouplens.org/> This site has collected data about rating of movies by moviegoers. Two databases are given on this site: one of 100,000 ratings for 1682 movies by 943 users. The second one consists of approximately 1 million ratings for 3900 movies by 6040 users. Pure applications of the complexity formula  $n_1^2 n_2 + n_2^2 n_1$  would give enormous numbers.

Step 1 of the algorithm requires each link between  $N_1$  and  $N_2$  to be examined only once for each element of  $N_2$ . Step 2 requires each link to be examined only once for each element of  $N_1$ .  $L_{12}$  being the number of links from node  $N_1$  towards  $N_2$ , step 1 of the algorithm has a time reduced to  $n_2 * L_{12}$ , and step 2 has its time reduced to  $n_1 * L_{12}$ . Thus, the time is only  $L_{12}(n_1 + n_2)$ . With the above examples, this represents a gain of time between 15 and 18 due to the sparsity of the matrix of links. In the case of  $k$  classes, with  $n = n_1 + n_2 + \dots + n_k$  elements, the total complexity is  $O(nL)$ .

## 4.4 Iterative

In this section I will explain the time complexity for the iterative approach to solving the proximity calculations.

### 4.4.1 Algorithm description

The algorithm given in this section solves the general problem expressed in equations [4.1] – [4.5]. The graph may include several nodes with loops. Our equations may be written in the vector form:

$$\vec{x} = M(\vec{x}) + \vec{c} \quad [4.20]$$

where:

$\vec{x}$ : is a vector representing the total proximity.

M: is an operator containing the coefficients and max, addition and multiplication operands from proximity equations

$\vec{c}$ : is a constant vector

We employ, under certain restrictions, some theorems and properties of linear algebra. See (Elsner and Driessche, 1999).

If we look at the linear-max equation system represented by the M object and the  $\vec{c}$  vector, it has the following properties by construction:

- 1) The diagonal elements are all ones
- 2) All coefficients are positive
- 3) The sum of all row coefficients other than the diagonal coefficient is less than or equal to one.
- 4) The  $\vec{c}$  vector has all of the coefficients in the [0, 1] range.

Property 3 is usually referred to as ‘diagonal dominance’ in linear algebra. It is well known that a linear system holds the form

$$\vec{x} = M\vec{x} + \vec{c} \quad [4.21]$$

where M is a strictly diagonally dominant matrix and is solvable by iterative methods called relaxation algorithms in the following way:

1) Choose any  $\vec{x}_0$

2) Compute  $\vec{x}_{m+1} = M\vec{x}_m + \vec{c}$  starting with  $m=0$  and continuing until convergence.

(Press et al, 1995) shows that convergence may be slow, but in practice it gets faster as the sum of the non-diagonal coefficients of matrix M becomes significantly lower than the rank of the matrix.

Because of the similarity of linear algebra and max-plus algebra, we are entitled to use the same iterative method of equation solving, known as the Gauss-Jacobi Algorithm. Compute:

$$\begin{aligned}\vec{x}_1 &= M(\vec{x}_0) + \vec{c} \\ \vec{x}_2 &= M(\vec{x}_1) + \vec{c} \dots\end{aligned}$$

#### 4.4.2 Time complexity

We use the same notation found later in the bottom-up algorithm. First  $p$  is the size of the  $\vec{x}$  vector. The size of the matrix-like M object is  $p \times p$ , where

$$p = n_1(n_1 - 1) + n_2(n_2 - 1) + \dots + n_k(n_k - 1) \quad [4.22]$$

with  $k$  being the number of nodes in our graph and  $n_i$  being the number of elements in one node. So  $p$  is of the order of  $n_1^2 + \dots + n_k^2$ , however M is a very sparse object as we have already seen.

An iteration of our algorithm consists of computing the  $\vec{x}_m$  vector elements using  $\vec{x}_{m-1}$ . Remember that the size of  $\vec{x}$  is  $p$  (see above). Using the same algorithm from 4.3.3, we get a computation time of  $nL$  for each basic step where:

$$n = n_1 + \dots + n_k \quad [4.23]$$

and  $L$  is the total number of links of any element to any other element. Iterating  $r$  times, the total complexity is given by:

$$t = O(rnL) \quad [4.24]$$

Intuitively, the convergence speed is determined by the sum of all non-diagonal elements of M. For instance, in the example presented in Annex A, the rank

of  $\mathbf{M}$  is 4 and the sum of all non-diagonal elements is  $7/4$ . Convergence speed is related to the ratio of this sum and the rank ( $7/16$  in our example). The time needed to converge decreases as this ratio decreases.

### 4.4.3 Proof of convergence

The convergence of this relaxation algorithm can be established in the same way as if  $\mathbf{M}$  was a true matrix and  $M(\vec{x})$  was the usual matrix-vector multiplication. We will first review the pure linear case in section 4.4.3.1, then introduce the max operator in section 4.4.3.2 and finally conclude on some graph characteristics which guarantee absolute convergence.

#### 4.4.3.1 Pure linear equations without max operators

Suppose that

$$\vec{x} = M\vec{x} + \vec{c} \quad [4.25]$$

would be a pure linear equation with:

- A constant matrix  $M$
- A constant vector  $\vec{c}$
- A vector of unknown  $\vec{x}$

The iterative method described in section 4.3.1 is then the so called power method. Given a matrix  $M$  with all positive elements, convergence of this method requires the highest eigenvalue to be less than one. However, as we know that the sum of each row is less than or equal to 1, we can deduce that the highest eigenvalue is less than or equal to one. For example, see (Pillai, 2005). Relaxation equations such as:

$$\vec{x}^n = M\vec{x}^{n-1} + \vec{c} \quad [4.26]$$

can be detailed as:

$$x_i^n = \sum_j m_{ij} x_j^{n-1} + c_i \quad [4.27]$$

We consider the difference  $\vec{x}^n - \vec{x}^{n-1}$  of the values of  $\vec{x}$  at two consecutive iterations.

The following vector norm measures the distance between consecutive values of  $\vec{x}$

$$\|\bar{x}^n - \bar{x}^{n-1}\| = \max(|x_j^n - x_j^{n-1}|) \quad [4.28]$$

which is the maximum of the absolute values of the components. Now we should show that

$$\|\bar{x}^n - \bar{x}^{n-1}\| = k \|\bar{x}^{n-1} - \bar{x}^{n-2}\|, k < 1 \quad [4.29]$$

which proves the absolute convergence of  $\bar{x}^0, \bar{x}^1, \dots, \bar{x}^n$ . For any  $\mathbf{i}$ , we have:

$$\begin{aligned} |x_i^n - x_i^{n-1}| &= \left| \sum_j m_{ij} (x_j^{n-1} - x_j^{n-2}) \right| \\ &\leq \sum_j m_{ij} |x_j^{n-1} - x_j^{n-2}| \\ &\leq \sum_j m_{ij} \|\bar{x}^{n-1} - \bar{x}^{n-2}\| \end{aligned} \quad [4.30]$$

Thus,  $\|\bar{x}^n - \bar{x}^{n-1}\| = \max_i |x_i^n - x_i^{n-1}|$  which is less than or equal to  $\|\bar{x}^{n-1} - \bar{x}^{n-2}\|$  because  $\sum_j m_{ij} \leq 1$

Under which conditions can the symbol ' $\leq$ ' be replaced by '<', which guarantees absolute convergence? This will be discussed in section 4.4.3.3.

#### 4.4.3.2 Equations with max operators

The proof of convergence of the previous section can be easily transposed into our equations which include maxes, pluses and multiplications by constants. The Linea model has equations like the following:

$$x_i^n = \sum_j m_{ij} \max(x_{j_1}^{n-1}, x_{j_2}^{n-1}, \dots, x_{j_k}^{n-1}) + c_i \quad [4.31]$$

where  $(j_1, j_2, \dots, j_k)$  is a set of indices of vectors  $\bar{x}$ , given as a function of  $j$ . Remember that  $\sum_j m_{ij} \leq 1$ .

We are still using the following measure of distance of two successive approximations of  $\bar{x} = \|\bar{x}^n - \bar{x}^{n-1}\| = \max_j (|x_j^n - x_j^{n-1}|)$ . For any  $\mathbf{i}$ , we have:

$$|x_i^n - x_i^{n-1}| = \left| \sum_j m_{ij} \left( \max(x_{j_1}^{n-1}, \dots, x_{j_k}^{n-1}) - \max(x_{j_1}^{n-2}, \dots, x_{j_k}^{n-2}) \right) \right| \quad [4.32]$$

Observe now that if a, b, c, and d are four positive numbers, it is always true that:

$$|\max(a,b) - \max(c,d)| \leq \max(|a-c|, |b-d|) \quad [4.33]$$

Thus,

$$\begin{aligned} |x_i^n - x_i^{n-1}| &\leq \sum_j m_j \max(|x_{j_1}^{n-1} - x_{j_1}^{n-2}|, \dots, |x_{j_k}^{n-1} - x_{j_k}^{n-2}|) \\ &\leq \left( \sum_j m_j \right) \|\bar{x}^{n-1} - \bar{x}^{n-2}\| \leq \|\bar{x}^{n-1} - \bar{x}^{n-2}\| \end{aligned} \quad [4.34]$$

Thus,

$$\|\bar{x}^n - \bar{x}^{n-1}\| \leq \|\bar{x}^{n-1} - \bar{x}^{n-2}\| \quad [4.35]$$

Of course, absolute convergence can only be achieved if strict ' $<$ ' inequality is guaranteed, like in equation [4.27].

#### 4.4.3.3 Equations of graph structures to achieve absolute convergence

First remember that, for every equation like

$$x_i = \sum_j m_{ij} \max(x_{j_1}, \dots, x_{j_k}) + c_i \quad [4.36]$$

we have, by construction of the Linea model

$$\sum_j m_{ij} + c_i = 1 \quad [4.37]$$

We shall call  $c_i$  the “constant term” of the  $i^{\text{th}}$  equation. If, for all equations, the constant term  $c_i$  is equal to zero, there is no convergence of the relaxation algorithm.

If some equations have  $c_i = 0$  and some other are not, we have to eliminate all variables  $i$  such that  $c_i = 0$ , by row additions. If variables eliminate trivially, the system is degenerate and hence we can replace these variables by zero. At the end of the elimination process, every equation has non-zero constant. Thus,

$$\sum_j m_{ij} < 1 \quad [4.38]$$

for any  $i$ .

It may of course happen that some zero-constant variables may not be expressed in terms of non-zero constant variables. Here is one example:

$$\begin{aligned}
 u &= v \\
 v &= u \\
 w &= .5(\max(x, y)) + .5 \\
 x &= .5w + .5 \\
 y &= .7w + .5
 \end{aligned}$$

Note that  $u$  and  $v$  are given by equations where there is a zero-constant term, and cannot be expressed as functions of  $w$ ,  $x$ , nor  $y$ . Their values are undetermined. Practically, we give them a value of zero. This situation is quite common in recommendation systems with 2 classes: fans and movies. If fan **A** saw movie **K** and fan **B** saw movie **L**, and neither **A** nor **B** saw any other movie, and **K** and **L** were not seen by any other fans. See Figure 35.

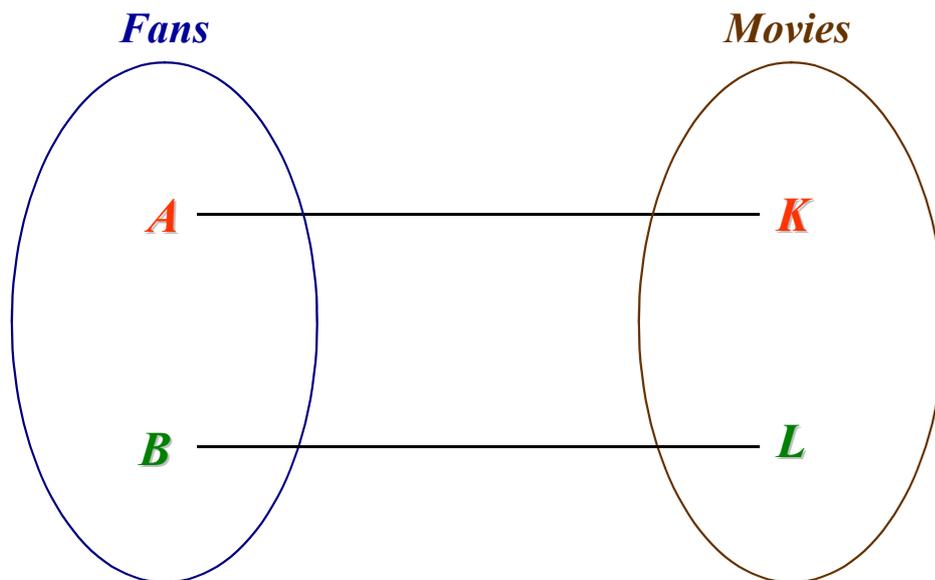


Figure 35 Movie recommendation system example with undetermined values

---

Considering this example and assuming all local proximities are zero, we have the following equations

$$u = P^*(A, B)$$

$$v = P^*(K, L)$$

$$u = v$$

$$v = u$$

which have no unique solution. Thus we set  $u$  and  $v$  to zero. With two classes (fans and movies in this example), a non-zero constant factor appears in the equations as soon as two elements of one class share the same element in the same element in their image sets within the image set node. See Figure 36.

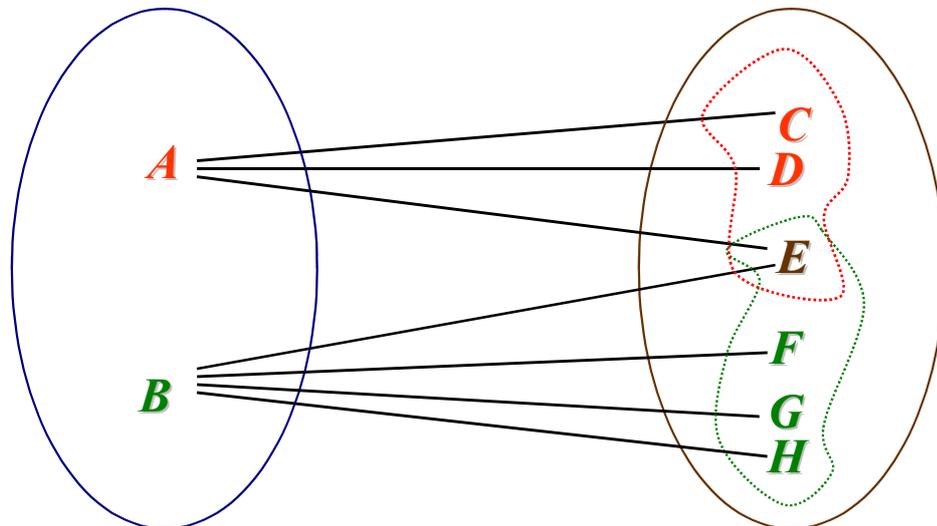


Figure 36 With the intersection of the two image sets at E, we are assured a non-zero constant

---

The equation giving the proximity of A and B includes a constant element. In this example, the constant is  $1/12$  because the image set cardinalities of A and B are 3 and 4 respectively.

If we still have two classes and there exists a path between two elements A and B of the same class, their proximity can be computed. Here is one example:

---

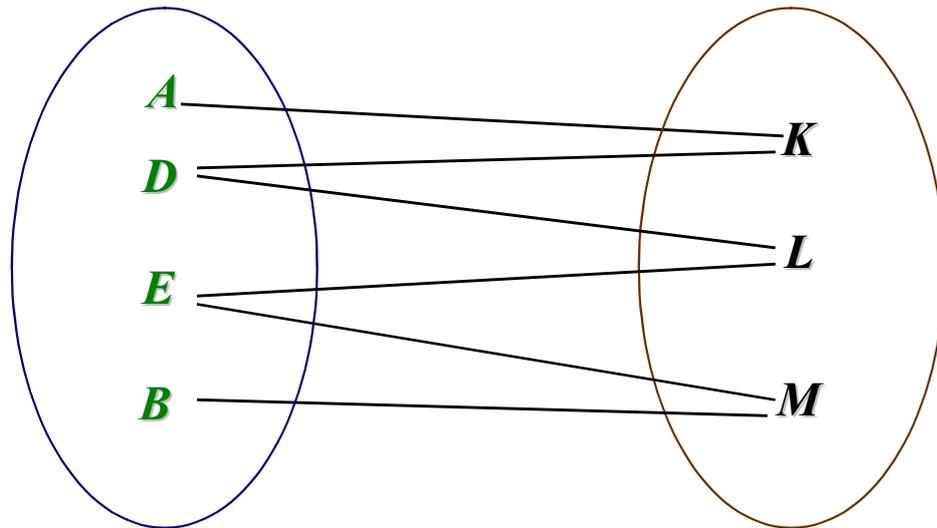


Figure 37 The proximity between A and B can be computed due to the path between them created from the links and intersections of the other elements.

---

The equations giving  $P^*(A,D)$ ,  $P^*(K,L)$ ,  $P^*(D,E)$ ,  $P^*(L,M)$ ,  $P^*(E,B)$  have non-zero constants. Therefore  $P^*(A,E)$  can be computed from  $P^*(K,L)$ ,  $P^*(K,M)$  can be computed from  $P^*(A,E)$  and  $P^*(A, B)$  can be computed from  $P^*(K,M)$ .

However, this kind of 'path reasoning' can not be generalized with more than 2 classes. Here is an example with 3 classes:

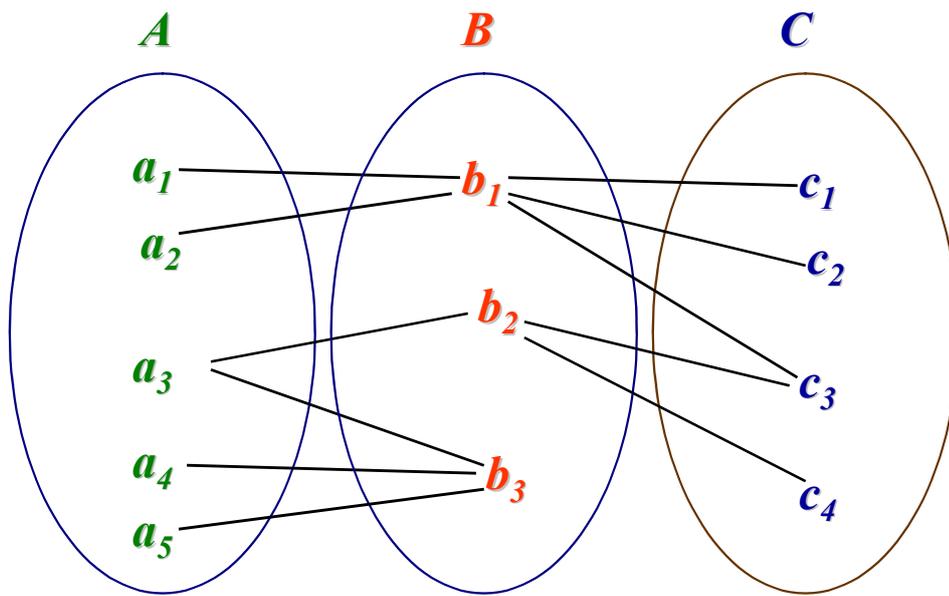


Figure 38 Three class example

---

Proximities of  $\mathbf{b}_1$  and  $\mathbf{b}_2$  can be computed and the proximities of  $\mathbf{b}_2$  and  $\mathbf{b}_3$  can also be computed. However, the proximity of  $\mathbf{b}_2$  and  $\mathbf{b}_3$  cannot be computed without other connected elements.

## 4.5 Conclusion

In this chapter we have introduced the Linea proximity calculator. In doing so, we described three implementation approaches—naïve, bottom-up and iterative. Furthermore, we provided time complexity estimations for each approach and provided a proof convergence for the iterative method. In chapter 5 we will discuss our experiments and their results as they pertain to each of the three methods mentioned in this chapter.



# Chapter 5

## Experiments

### 5.1 Introduction

In this chapter we will present the various experiments we conducted with the Linea proximity calculator. We will start with a description of the data we worked with during our experiments. Next, we will compare the results between various approaches we employed with the proximity calculator. Specifically, we will first measure the accuracy of the three approaches as compared to the manual results. Next, we will compare their performance. Finally, we will make some conclusions and recommendations for algorithm choice.

### 5.2 Data description

For our experiments, we refer to data from two different domains. The first one is a simple data-store that represents a set of web pages. We used this set of data because it allows us to manually perform the proximity calculations for verification. The second set of data is more complicated. It contains sample information representing employees of research and development (R&D) division of Électricité de France (EDF), a large French corporation. I will discuss both databases in detail.

## 5.2.1 Web pages

The web pages data-store provides a simple representation of a set of web pages. There are two entities: web pages and terms. There are three associations. Web pages contain terms. Terms are contained in web pages. And web pages point to other web pages.

---

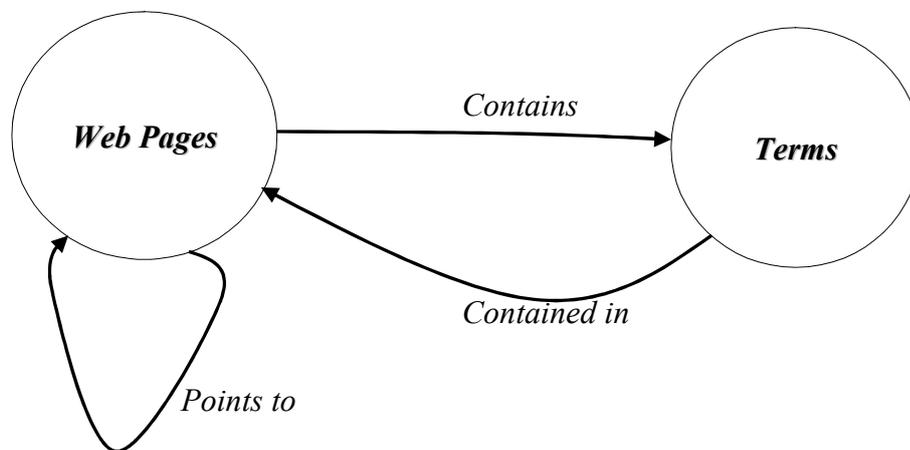


Figure 39 Web pages data model

---

The web pages data store consists of 27 elements in total. Specifically, there are 17 web pages and 10 terms. There are 6 web pages that have terms in their image set. Likewise, there are 6 web pages that have other web pages in their image set. Finally, there are 4 terms that have web pages in their image sets.

Since this is such a small dataset, we use it mainly for comparisons between manual and automated methods. For more robust testing, we use the corporate data structure described in the next section.

### **5.2.2 Corporate**

The corporate data set contains sample information representing employees of the R&D division of EDF. This data set is more complicated mainly due to two factors. First, it contains many more elements. Second, there are seven associations in the graph, significantly more than in web pages. There are three entities modeled by the corporate data store: persons, competences and groups. There are bi-directional associations between all entities. In addition, there is a reflexive association in groups, which represents a super/sub group association.

The source node used throughout our experiments is the person node. The structure of the data contained in the corporate data sets has a much lower percentage of image set intersections. Accordingly, we can expect that the proximity results will be lower than those of the web pages database. In the following paragraphs, we provide a detailed description of the corporate data set that provides support to our conclusion of lower expected proximity scores.

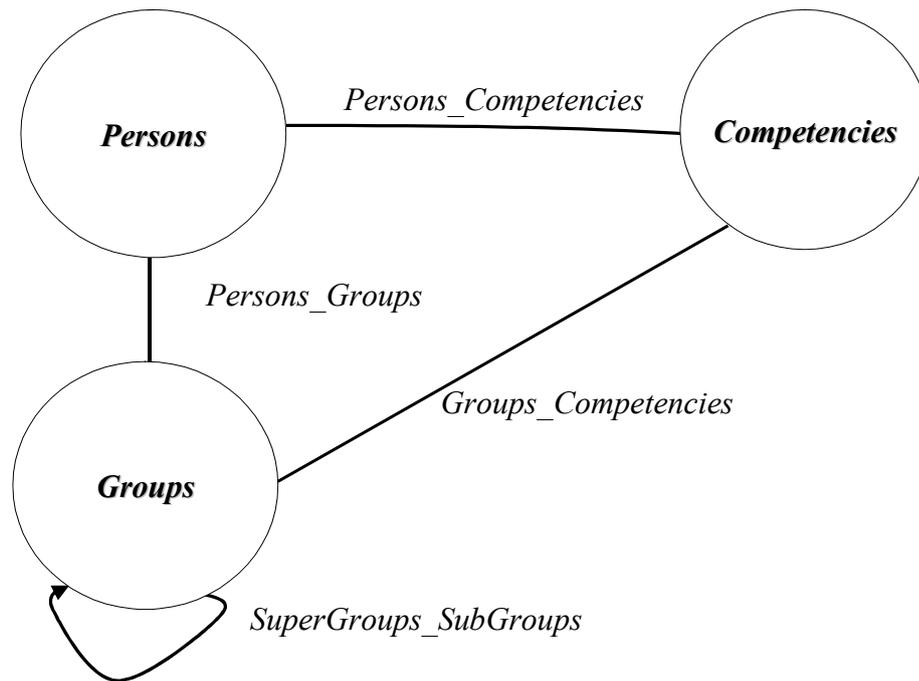


Figure 40 Corporate Data model

---

There are 2573 people represented in the corporate database. There are also 214 groups and 128 competences. On closer examination, other interesting characteristics of the data emerge. First, only 193 of the 2573 corporate personnel have a listed competence. Of this population, 99 have only one competence. Afterwards, there are 33 people who have three competences. There is one person who has six competencies, the most in the database.

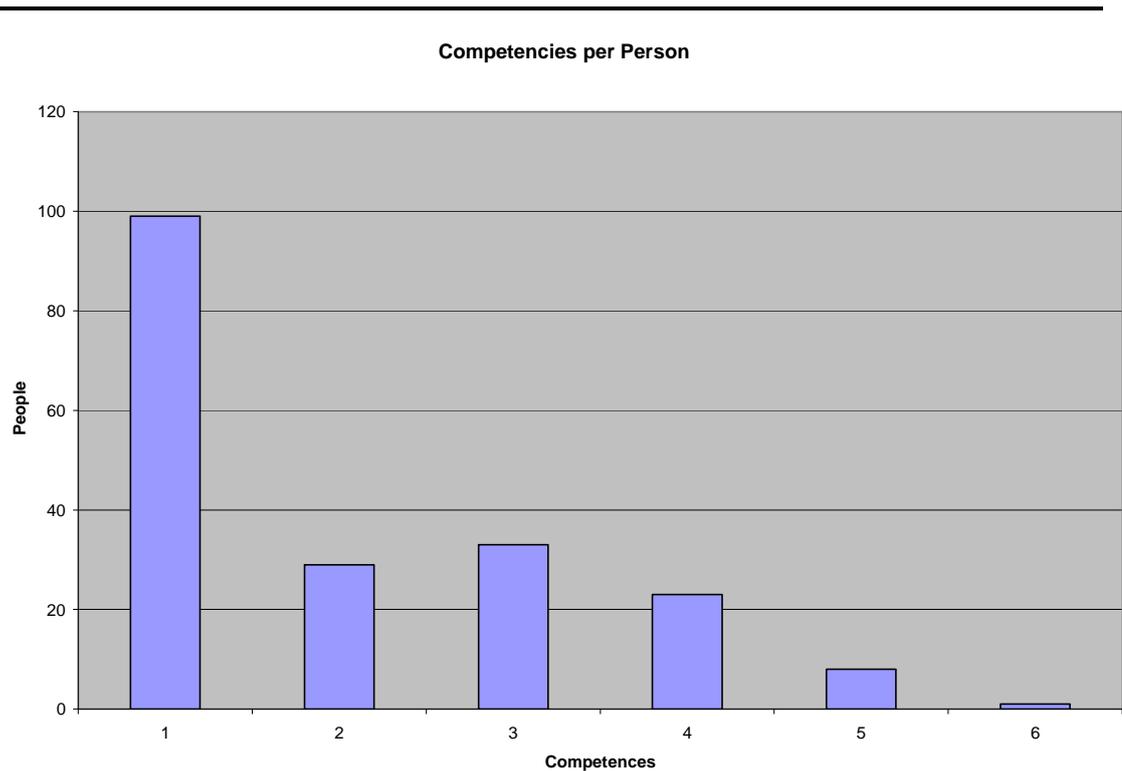


Figure 41 Competencies per Person comparison

---

Inversely, of the 128 competencies listed, 24 are not associated with a person. Of the 104 competencies associated with people, 22 of them were associated with 1 person only. Equally, 22 competencies were associated with 2 persons. There were three competencies associated with 11 or more people. The most common competency, Networked Computer Systems & Telecommunications, was associated with 15 people.

When we consider the image sets of elements in persons in the competencies, we can see that the chances of overlap between different image sets are low. The effect will be a lower overall proximity measure as a result of competencies. As we move further away from the source node, in this case people, the effects of image set overlap are lower. Accordingly, although there is a higher chance of image set overlaps when taking competencies as the originating node and persons as the image set node, the positive effects are lesser since we are considering image sets node

directly associated with persons. In this case, there is an intermediate node, competences, between the source node (persons) and the node containing the image sets (persons).

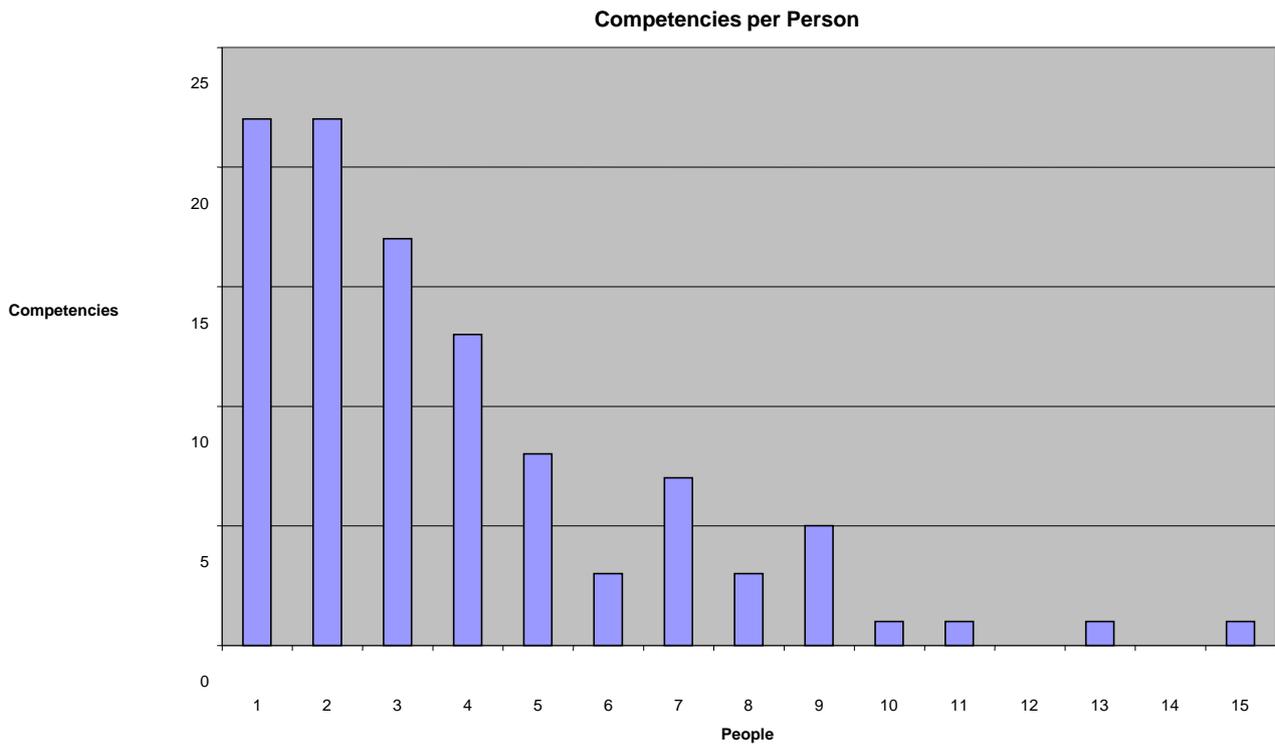


Figure 42 Competencies per Person comparison

---

When comparing groups to people, we found that most people, by far, belonged to only one group. In fact, out of the 2545 people in the corporate data store that are associated with a group, 2512 were members of only a single group. An additional 30 people belonged to 2 groups while only 3 people belonged to either 3 or 4 groups. No one was associated with more than 4 groups.

The distribution of people among groups is more disperse than people with competences. Whereas most people tend to have one or two competencies, the

number of people per group follows more of a normal distribution. There are 16 groups that have 15 people in them. At the extremes, only 1 populated group contains the lowest number of people: one. Similarly, there is only 6 groups contain 29 or more people including the outlier group that has 58 members.

When we consider image sets in node groups, which originate in the persons node, there is a higher chance of set intersection since the vast majority of people belong to a group and many groups contain 15 or more people. Accordingly, image set proximity scores here should be higher.

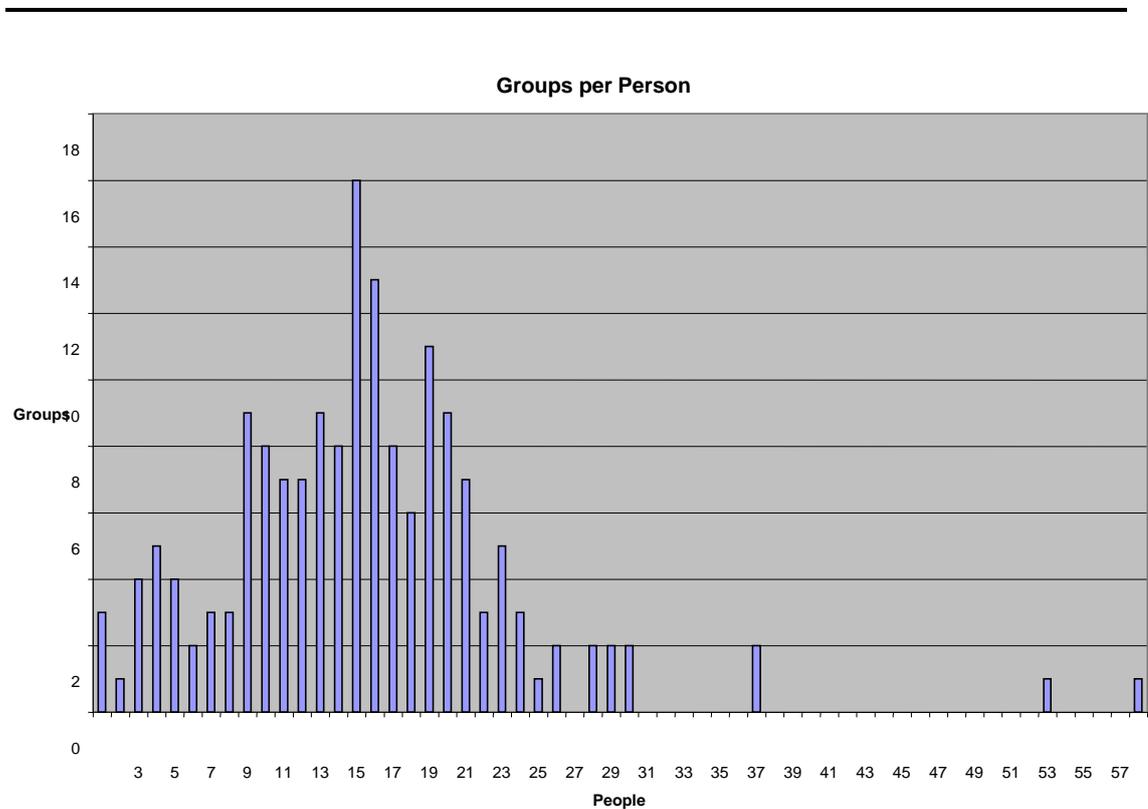


Figure 43 Groups per Person comparison

---

Groups can also contain other groups. There are 30 groups that contain other groups. Most of these super-groups contained between 4 and 9 sub-groups. However

there is one case in which a group contains 20 sub-groups. Also there is one instance in which a sub-group is a member of 2 super groups. Finally, there are 26 super-groups that are also a sub-group of another.

Of the 214 groups in the corporate data store, 30 have sub-groups. Hence less than 15% of the groups have subgroups. If we consider groups as the originator node for image sets contained in subgroups, we can conclude that this structure feature also lends itself to lower proximity scores.

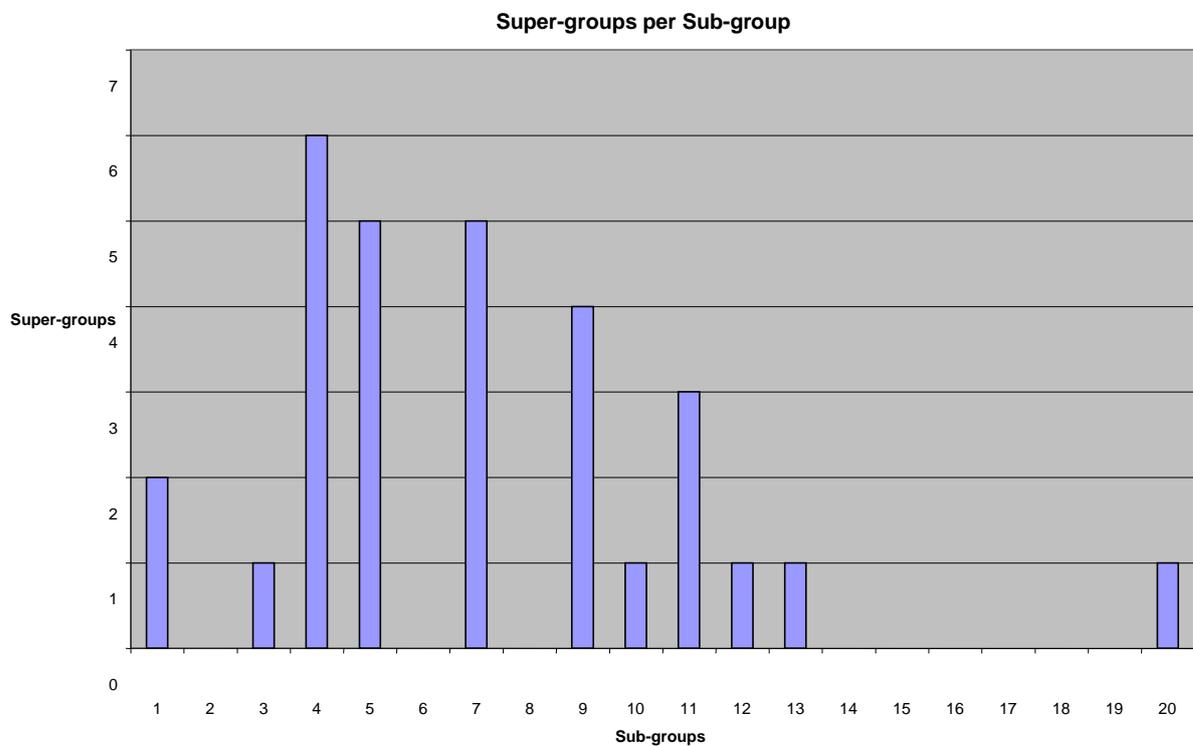


Figure 44 Super-groups per Sub-group

---

In the corporate database 20 groups have competences. The majority of these groups had 8 or fewer competences. There were 2, however, that had 12.

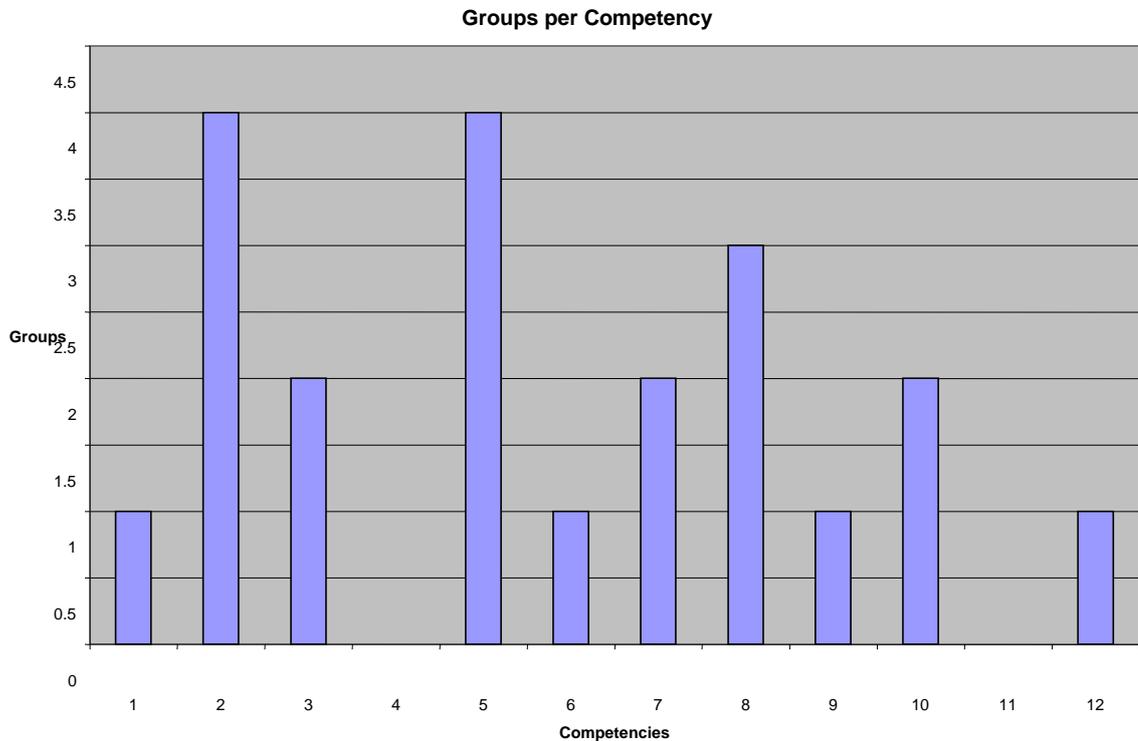


Figure 45 Groups per Competency

---

Out of the 128 competences in the database, 120 were associated with a group. Furthermore, each competency was associated with only one group.

As we have considered all permutations for image sets, we can expect to have lower proximity scores. ( $\sim .25$  and lower) This analysis helps us to have an expectation of values when we run the experiments.

### 5.3 Accuracy

In this section, we discuss the accuracy of all three implementations. First we confirm for this example the consistency of our results regardless of the order in

which the paths were followed by the algorithm. We then compare results from the different implementations to the manual results

### **5.3.1 Intuitive Accuracy Experiment**

The naïve implementation is different from the other approaches in several ways. One of these differences is that the naïve implementation is tied to the structure of the graph. In other words, the implementation follows the relationships from node to node based on the location of current elements' image sets. This attachment to the graph structure led to a hypothesis that we needed to confirm through experimentation.

#### **5.3.1.1 Recursive tree consistent proximity results**

With the naïve implementation of the proximity calculator, the algorithm follows paths outward from the initial node, which contains the x and y elements whose proximity we are interested in obtaining. Since the results of each path's calculation are added to the next path's calculation, the order in which the paths are calculated should not change the result of the proximity calculation.

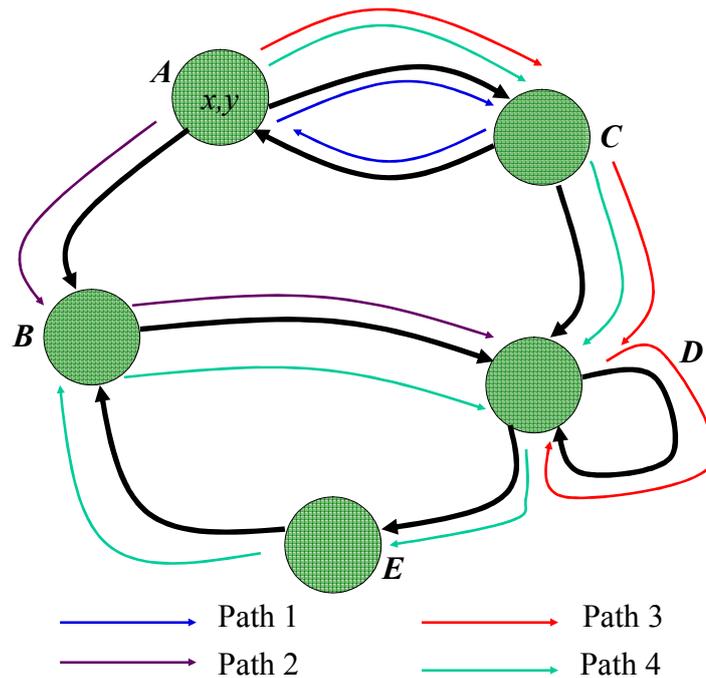


Figure 46 Calculation paths

To confirm this, we decided to run some experiments to compare the results from calculations made from following paths in different order. We defined two simple heuristics. We then developed an application that analyzed the graph structure of the problem domain and ordered the paths according to the desired heuristic. We then ran the calculations over each heuristic, collected the results and then compared them. We did this for both data sets (Web pages and corporate). The results were inline with our hypothesis. This, in turn provided a partial validation of the correctness of our implementation.

### 5.3.1.1.1 Heuristics

Our first step was to define differentiable heuristics for calculating the global proximity between two elements. The global proximity included both the local and image proximities. The local proximity is a static calculation, in that there is only one way to calculate it. Regardless of the implementation, the local proximity should

remain consistent. However, the image proximity consists of recursive calls that follow a path through the graph based on image sets. During the recursive step of an element proximity calculation, there may be several image sets, each located in distinct nodes. Hence there lies the possibility of different paths. How does the proximity calculator determine which image set to calculate next? Initially it was random. There were no heuristics.

In turn, we proposed two heuristics: shortest-to-longest and longest-to-shortest. As the names imply, the shortest-to-longest heuristic chooses the shortest path first (assuming multiple paths) then progressively chooses the next longer path. Conversely, longest-to-shortest heuristic chooses the longest path first then progressively chooses the next shorter path.

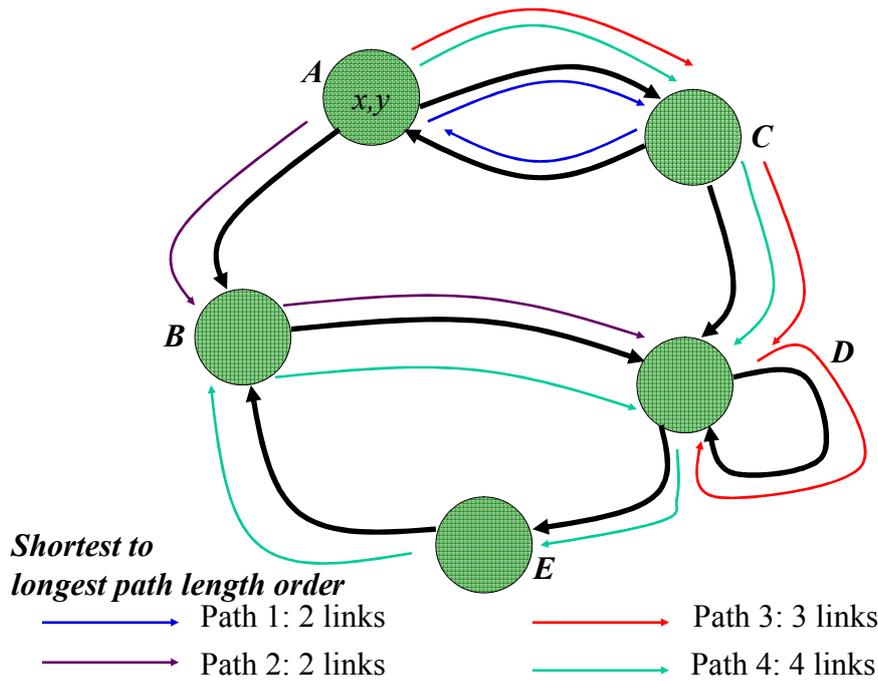


Figure 47 Heuristics

### 5.3.1.1.2 Approach

Until this point, we were not concerned with path lengths. Hence, the proximity calculator had no facilities to analyze them. Accordingly, our first task was to add a feature to the proximity calculator which would allow us to consider path lengths. Next we ran the algorithm over the Websites and corporate data sets. For each graph we chose one node to work with. In this node we measured the proximity between an element and every other element in the node. We then repeated this process for every element in this node. We stored all of the results on disk for later access. We did these calculations for each heuristic.

### 5.3.1.1.3 Software Implementation

Adding the path following functionality to the proximity calculator was accomplished by first implementing a path manager function. We then implemented an experiment class which ran the various calculations.

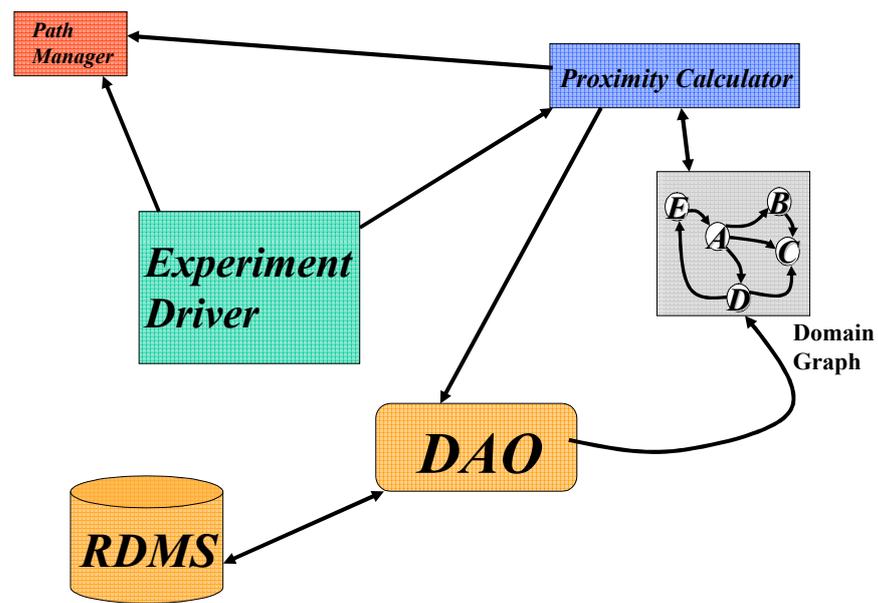


Figure 48 Experimentation software architecture

### 5.3.1.1.3.1 Path Manager

The path manager class is the heart of the experimentation software. First, given a start node in a graph, it identifies and catalogs each path in the graph by length. Then it arranges and stored the paths in both increasing and decreasing order. These actions are done during the construction of the path manager object. Afterwards, the experiment class calls on the path manager when calculating the proximities.



Figure 49 Path Manager Class Diagram

---

Within the image proximity implementation, we adjusted the software to make a call to the path manager when considering the next image set to process. The next image set provided by the path manager correlates to the heuristic used.

As an example, consider Figure 50. Assume we are following a shortest to longest path heuristic and the start Node is A. Furthermore, we have arrived at Node D for the first time. We are at the step in the algorithm where we are recursively calculating the global proximities of each element in the two image sets located in Node D. There are 2 outbound relationships from D: a reflexive relationship back to D and a relationship to Node E. The path that would take us to Node D is 3 links. The path that would take us to Node E is 4 links. Since we are following a shortest to longest heuristic, the next node to consider for calculating image sets is Node D.

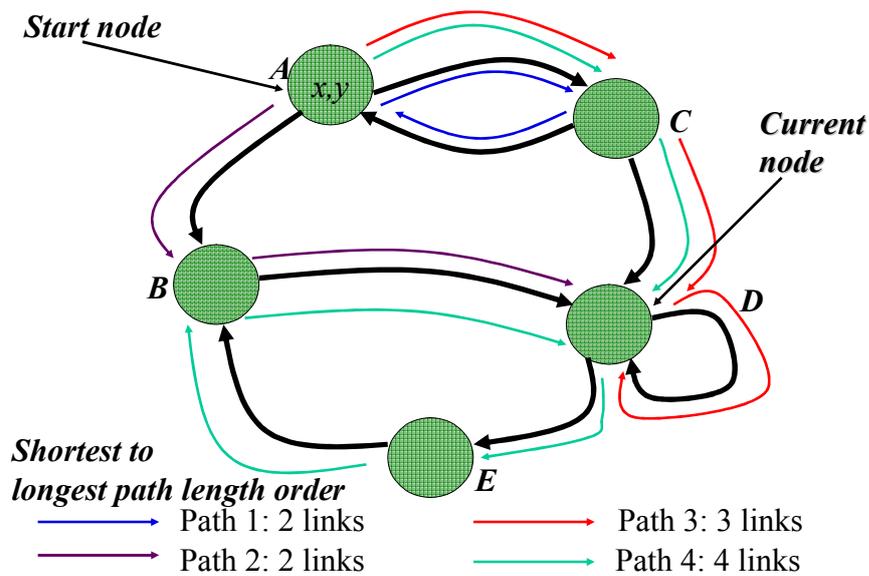


Figure 50 Path manager example

This is exactly how the path manager interacts with the proximity calculator. At each point in the algorithm where the proximity calculator has more than one possible node to step to for image set calculations, it calls the path manager to determine the correct node, based on the current heuristic.

#### 5.3.1.1.3.2 Experiment Driver

The code that drives the experiment is encapsulated in the ExperimentDriver method. This simple method makes the necessary calls to 1) initialize the proximity calculator with the proper graph, 2) instantiate a PathManager object 3) run the multiple iterations of the proximity calculations and 4) store the results to disk. The principal methods in this class are listed in Figure 51.

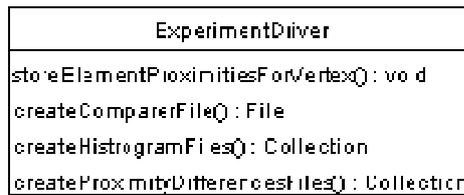


Figure 51 ExperimentDriver class diagram

---

### 5.3.2 Manual calculations

We first examine the accuracy of the proximity calculator as compared to a manual approach. We use the Web Pages data set to perform our experiments. We structured our experiments with three representative examples, based on the measure of proximity. More specifically, we consider the cases when there is a partial, complete and absence of imaged proximity.

The first example is of partial image proximity. We obtain our sample elements from the Web Pages node. The two Web Pages are WP2950 and WP2951. In the manual calculation we need to follow all of the links based on image set relationships. Both WP2950 and WP2951 contain terms and point to other Web Pages. The following series of diagrams show the progression of the algorithm as we manually calculated the proximity between WP2950 and WP2951. In Figure 52, you can see the terms that WP2950 and WP2951 are associated with. These two set of terms make up the image sets of WP2950 and WP2951 in the Node Terms.

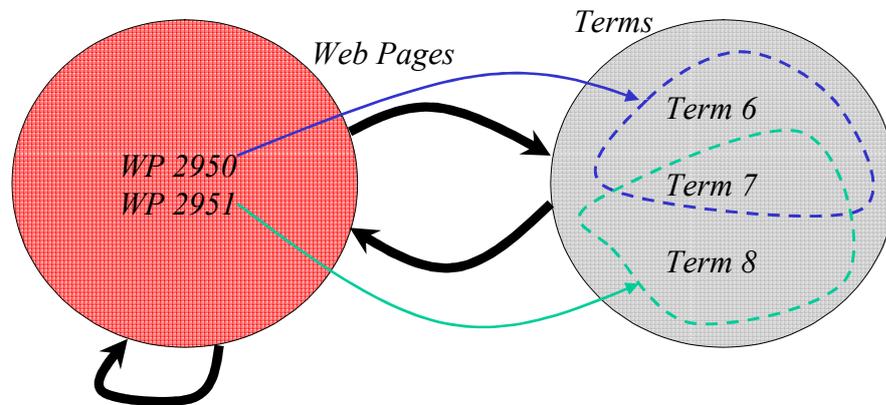
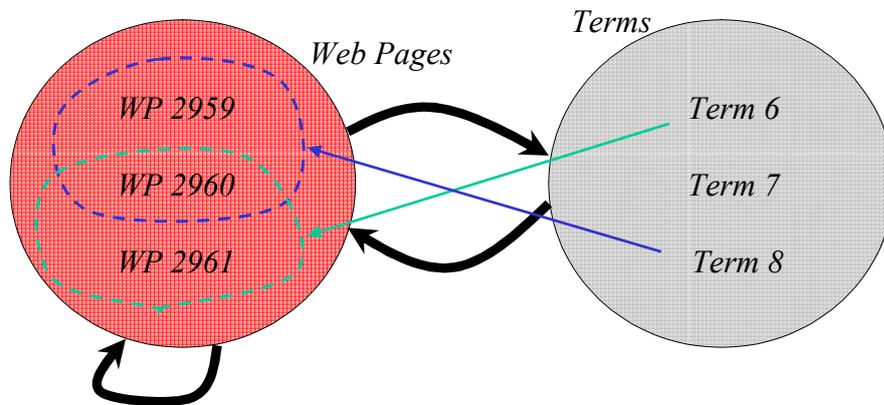


Figure 52 Terms related to WP2950 and WP2951

---

Next we calculated the set distance between the two image sets in Terms. In doing so, we must calculate the total proximity between the set Term 6 and Term 7 and the set consisting of Term 7 and 8. Calculating the total proximity requires finding both the local and image proximities. Hence, identify the image sets of the two groups in the Node Web Pages. See Figure 53.



---

Figure 53 Proximity calculations in the Node Terms

---

The image sets in the Web Pages node intersect on one Web Page: WP2960. Furthermore, since the Web Page node has already been visited, we do not consider image sets; we only calculate the local proximities when determining the proximity between the image sets.

At this point, we have followed the path: Web Pages-Terms-WebPages. We must also consider the other path from Web Pages: Web Pages-Web Pages. The calculations are the same; we just follow the links of image sets over the identified path.

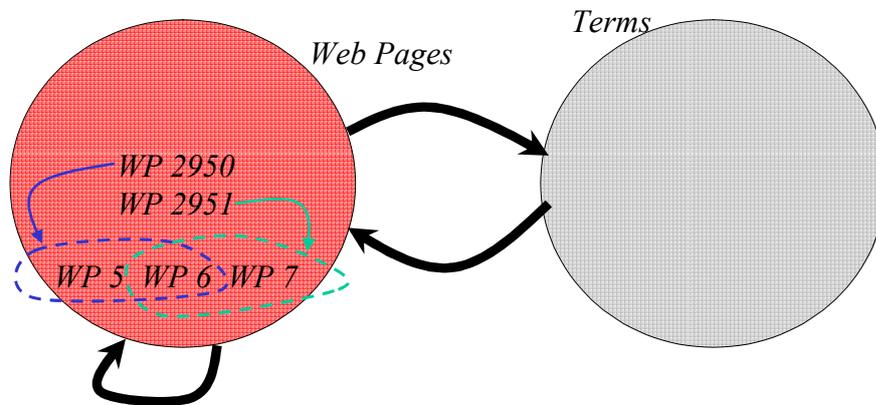


Figure 54 Second path starting at Web Pages

---

In this second path, we consider the same 2 original Web Pages (WP2950 and WP2951) and identify the other Web Pages to which they point. See Figure 54. We calculate the proximity between the 2 image sets in Web Pages in the same manner as above when we arrived at a previously visited node: we consider only the local proximities when determining the set distance.

We performed the same manual calculations on Web Pages that had both a complete overlap on their image proximity and that had no image proximity. The results of the manual calculations are listed in the table below.

---

## Manual Results

<i>Web Page 1</i>	<i>Web Page 2</i>	<i>Proximity</i>
<i>WP2950</i>	<i>WP 2951</i>	0.281
<i>WP 2949</i>	<i>WP 2952</i>	0.500
<i>WP 2949</i>	<i>WP 2950</i>	0.125

Figure 55 Manual Proximity Results for Web Pages

---

Note that the proximity between WP2949 and WP2950 is greater than zero. This is the case because the secondary image sets (from Terms back to Web Pages) have image sets that overlap.

### 5.3.3 Accuracy of Automated methods vs. Manual

After determining the proximities manually, we then determined the proximities using each of the implantation methods (naïve, relaxed iterative and bottom-up). Before presenting the results, a discussion of the precision for the relaxed iterative method is due.

Recall that with the relaxed iterative method, iteration stops with the difference between  $p_n$  and  $p_{n-1}$  is less than the precision variable  $\epsilon$  where  $0 \leq \epsilon \leq 1$ . In our implementation, we compared the average  $\epsilon$  for all proximity values in the graph. We randomly chose a  $\epsilon$  default value of .01 for our calculations. In practice,

this value will adjust based on the needs of the application. We found that at  $\epsilon = 0.01$ , the proximity results were consistent with the other methods to 5 decimal places. All three automated results were consistent with the manual calculations.

## **5.4 Performance**

In this section we compare the performances of the three implementations. In particular, we consider both initiation time and proximity calculation time for each version of the Linea proximity calculator.

### **5.4.1 Approach**

To capture the times, we added timing methods during object initialization and during the actual proximity calculations. We then stored all of the timing information in a convenient data structure we created to manage the results of our experiment runs. After the each run we then collected the timing data for further analysis.

### **5.4.2 Impact of precision on iterative method**

In the relaxed iterative method, the performance is influenced by the precision variable  $\epsilon$ . As  $\epsilon$  decreasing, the calculation time increases until the algorithm stabilizes. As discussed in chapter 4, our relaxed iterative algorithm was expected to stabilize quite quickly due to the diagonal dominance of the system of equations. We can see this behavior reflected in the calculation time. After seven to ten iterations, the calculation time levels out. See Figure 56.

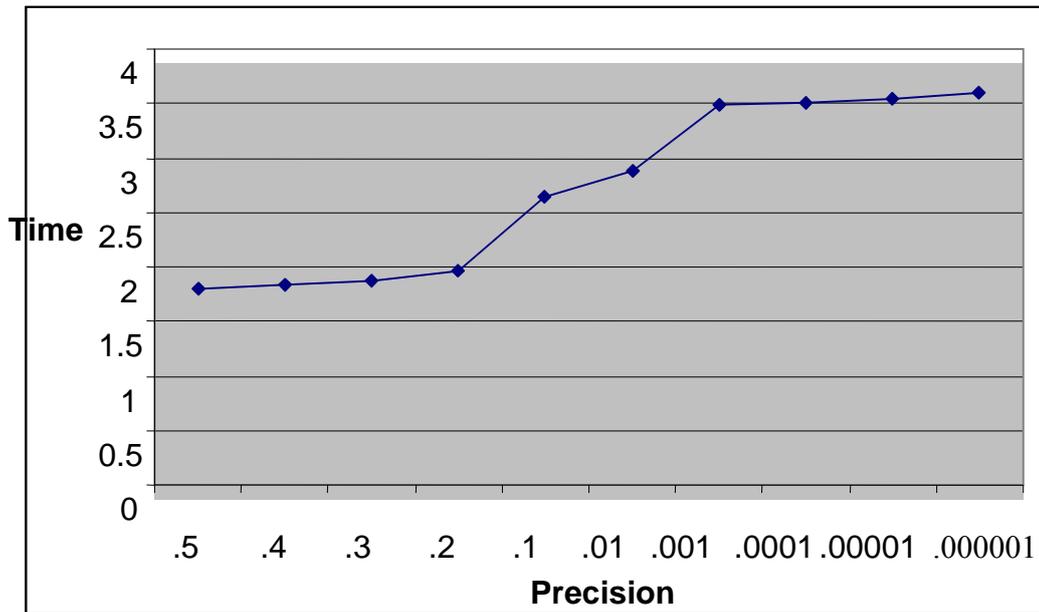


Figure 56 Calculation time versus precision

---

### 5.4.3 Results

We then calculated the proximities of the entire Web Pages node using the NodeProximity data structure found in chapter 6. We performed these calculations using each of the three Linea implementations. During each run of the experiment, we captured the time.

---

<i>Implementation Approach</i>	<i>System Initialization time</i>	<i>Initial Answer time</i>	<i>Subsequent Answer time</i>
Naïve	5.458	.0165	.0165
Iterative	0	130102	0
Bottom-up	5.458	306670	0

Figure 57 Comparison performance results experiments

---

To better see the behavior of the algorithms, we divided the times into three classes: 1) system initialization 2) initial answer time and 3) subsequent answer time. The system initialization time describes the time needed for any initializing steps such as the graph class initialization. The initial answer time captures the time needed to produce the first answer while the subsequent answer time describes the time needed to calculate each result afterwards.

The naïve and bottom-up approaches take the same amount of time for system initialization as they both use the graph structure in their calculations. The naïve approach's initial and subsequent times to calculate a result are both .0165 seconds. This is as to be expected since each calculation (whether initial or subsequent) is performed the same way. The bottom-up approach's initial calculation time is much larger than the naïve approach. However, the subsequent times to calculate a result are negligible as the computational significant calculations are performed while gathering all of the proximity results for the initial calculation. Finally, we see that

the iterative approach has by far the most costly calculation time for the initial calculation. This is due to the fact that we calculate all of the proximities in the graph multiple times. Note though that the subsequent calculation time is negligible.

## 5.5 Conclusions

Each of the implementation approaches has both their advantages and disadvantages. They have different situations in which they would be best employed. We compare each approach's advantages and disadvantages and suggest the situation in which it would best be used. In order to accomplish this we compare the results of the EDF dataset run with 1 calculation to 1000 calculations.

---

<i>Implementation Approach</i>	<i>1 Request</i>	<i>1000 Requests</i>
Naïve	5.458	16.5
Iterative	130102	0
Bottom-up	306670	0

Figure 58 Experiment result insights

---

As we can see from Figure 58 the naïve approach becomes costly as the requests increase. By contrast the iterative and bottom-up methods are extremely

efficient time-wise for multiple calculations. Their time costs are concentrated in the initial calculation. Finally, the iterative approach is some about more than twice as fast as the bottom-up approach. Accordingly, for problems that involve one or only a few calculations, the naïve approach is the best suited. However, for problems which call for many proximity calculations, the iterative approach is the best choice.

# Chapter 6

## Implementation

### **6.1 Introduction**

In this chapter, I will discuss the software implementation of the proximity calculator. I will first give an overview of the system architecture, pointing out the major components and the communications. Next, I will discuss the multi-tier approach employed to implement this application. Afterwards, I will discuss in some detail the design of the underlying database and its implementation. Finally, I will touch on some implementation improvements that have been made over the course of development.

### **6.2 System Architecture Overview**

#### **6.2.1 Introduction**

The key factor in driving the design of the proximity calculator was flexibility. Given that our partner, I-nova was interested in applying the proximity calculator to several domains, it was imperative that we design the calculator for easy adaptability. For the most part, we accomplished this by applying software best practices including abstraction and encapsulation. Another factor that influenced the system design was the eventual need for extendibility. Accordingly, we applied several features of the

Java 2 Enterprise Edition J2EE architecture. In other cases we designed the software to be easily scaled to support larger applications, which will be discussed in the Multi-tier section.

In this section I present the overall system architecture, explaining the various components. Afterwards, I will discuss the sequence of communications between the system elements during a normal operation.

### **6.2.2 Architecture Elements**

The key to the system is the actual proximity calculator. It performs the calculations that measure the proximity between 2 given elements. However, there are several other parts to the system that are also crucial to the proximity calculator. The supporting graph that's used by Linea is implemented as a java object. When the graph class is instantiated, the Data Access Object (DAO) accesses the Relational Database Management System (RDMS) and populates the graph class with the proper graph. The Path Manager controls the path the proximity calculator follows based on the chosen heuristic. Finally, the proximity calculator can be accessed from various sources: web browsers, java applications and web services clients.

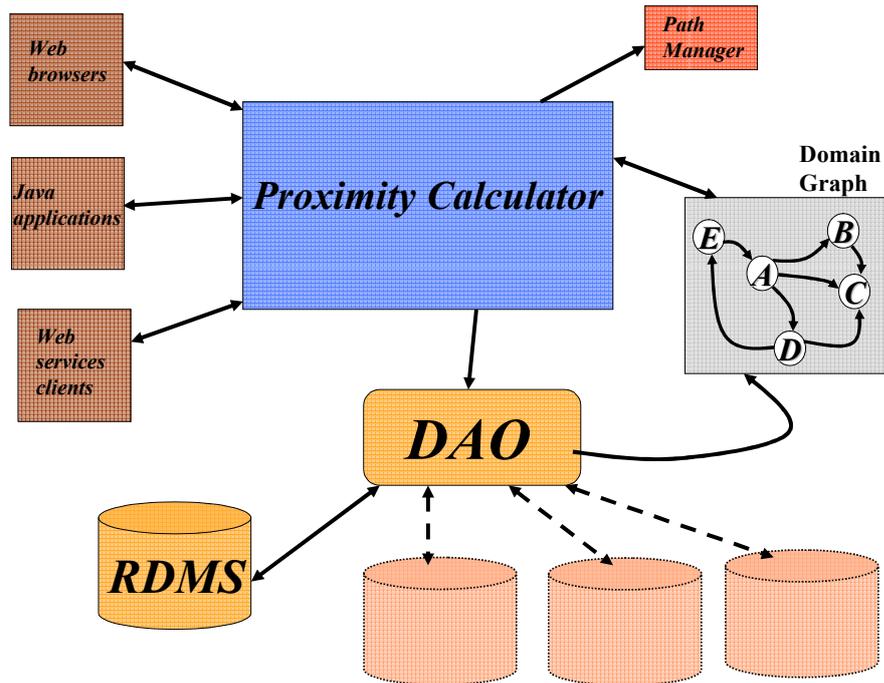


Figure 59 Proximity Calculator System Architecture

### 6.2.2.1 Proximity Calculator

The proximity calculator does one function. It calculates the proximity between 2 elements in a node of a graph. Accordingly, its public interface contains only one method:

GlobalProximity(Element x, Element y, Graph graph, Double alpha, PathManager pathManager)

The CalculateGlobalProximity method takes the 2 elements between which we want to obtain the proximity. It also takes a graph object that contains the elements and a value for the alpha, which is between 0 and 1, and provides the weight between the local and image proximities. An alpha value of 0 would mean only the image proximity would count. An alpha value of 1 would mean that only the local proximity would count. An alpha value of .5 would give the local and image proximities equal weight. Finally, the method also takes a Path Manager object which manages the

order of the paths followed during calculation. The `CalculateGlobalProximity` method returns a proximity result as `BigDecimal` object.

There is also a set of helper classes that have been added for managing larger sets of proximity calculation results. By using these classes, a client, with one request, can obtain all of the proximity measurements in a domain graph. In addition, the results are returned in a single data structure for easy manipulation.

A foundation class is the `ProximityDataElement`, see Figure 60. For a proximity calculation, we identify the two elements in question as either the source or destination element ID. This data structure contains the destination `elementID`. It stores the proximity result. This data structure is never instantiated directly by a client application. Instead, it is either subclassed or contained in another class.

The `ProximityDataElementPair` extends `ProximityDataElement`, adding the source element ID. See Figure 60. This class holds the information for a complete proximity calculation. Hence, a client could actually use this class directly.

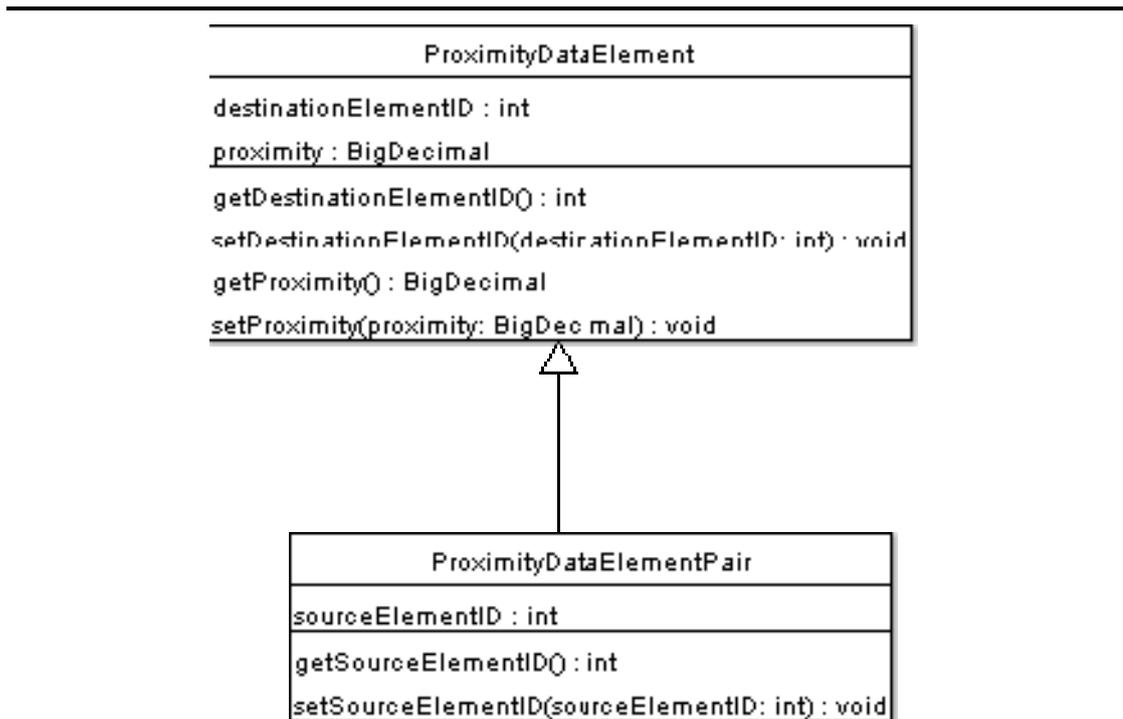


Figure 60 ProximityDataElement and ProximityDataPair

---

The ElementProximity class also uses the ProximityDataElement. However, instead of sub-classing it, the ElementProximity contains a collection of ProximityDataElements. This class holds all of the proximity measurements from one element to all of the other elements in a node. See Figure 51.

Following a logical progression, the NodeProximity class holds an ElementProximity for each element in the node. This gives a client application, in one data structure, all of the proximity calculations from any element in a node to any other element. See Figure 61.

ElementProximity
referenceID : int
proximityDataElements : Collection
getProximityDataElements() : Collection
getReferenceElementID() : int
setProximityDataElements(proximityDataElements: Collection) : void
setReferenceElementID(referenceElementID: int) : void
addProximityDataElement(proximityDataElement: ProximityDataElement) : void
removeProximityDataElement(proximityDataElement: ProximityDataElement) : void

NodeProximity
nodeID : int
elementProximities : Collection
getElementProximities() : Collection
getNodeID() : int
setElementProximities(elementProximities: Collection) : void
setNodeID(nodeID: int) : void
addElementProximity(elementProximity: ElementProximity) : void
removeElementProximity(elementProximity: ElementProximity) : void

Figure 61 ElementProximity and NodeProximity

Finally, there is GraphProximity class which holds a NodeProximity for each node contained in the graph. These data structures allow a client application to request possibly thousands of proximity calculations with one method call and in addition receive a data structure in response that has all of the results available for further manipulation.

---

GraphProximity
graphID : int
nodeProximities : Collection
setGraphID(graphID: int) : void
getGraphID() : int
getNodeProximities() : Collection
setNodeProximities(nodeProximities: Collection) : void
addNodeProximity(nodeProximity: NodeProximity) : void
removeNodeProximity(nodeProximity: NodeProximity) : void

Figure 62 GraphProximity

---

### 6.2.2.2 Path Manager

As described in the Experiments chapter, the Path Manager class provides the means for controlling the order in which the paths are followed during the proximity calculation process. We developed this class to support the experiments where we wanted to strengthen the conjecture that the order in which the paths were followed had no impact on the proximity calculation results. The experimental results supported our hypothesis.

The path manager also analyzes the path structure of the graph. It captures all of the paths that exist in a directed graph given a reference node. The path manager also orders the paths by length. Given these additional features, we decided to keep the path manager functionality.

### 6.2.2.3 Graph Class

The Graph Class models a directed graph and provides the environment in which the proximity calculator performs its calculations. The Graph class contains a set of node and edge objects.

Other than the getter and setter methods, the Graph class also has some methods especially tailored for the proximity calculator. The `getInboundWeight()` method is used to get the weight for a particular node when calculating the image proximity.

There is also a set of methods that track when a node has been visited (`addVisitedVertice(int VerticeID)`, `addVistiedVertices(Collection items)`, `clearVisitedVertices()`) which is used to determine if a node has been visited in the current path. If it has been visited, then we consider the path ended.

### 6.2.2.4 RDMS

We used the MySQL Server 4.1 database system for our application. It is a fast and simple open source database system. Also, several advanced features, such as *subquery* and *join*, have been added to this version that were crucial for the proximity calculator implementation.

## 6.3 Multi-tier approach

The proximity calculator is a web-based agent written with the J2EE facilities. The system was developed following a three-tier approach. There is a thin interface tier developed using Java Server Pages (JSP) technology. The middle tier houses the business logic and is written in Java. The final tier is the data-access level.

### 6.3.1 Introduction

The proximity calculator system uses a 3-tier design for separating software functions. In this section, I will discuss the benefits of this approach. I will also discuss the DAO functionality in more detail.

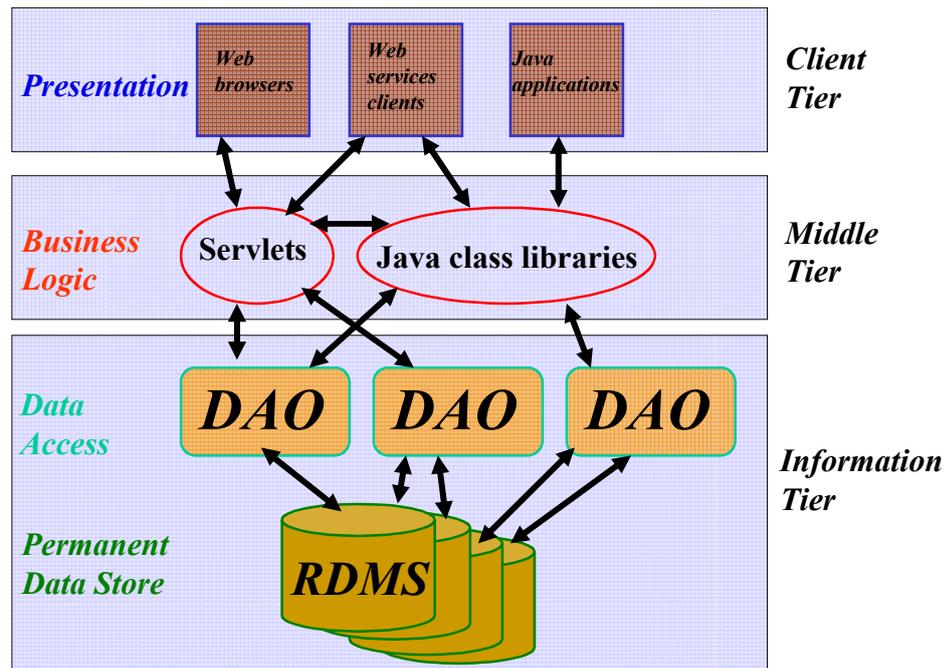


Figure 63 Multi-tier architecture

---

### 6.3.2 Benefits of the approach

This separation simplifies design and development and software packaging. The multi-tier approach enhances code separation, encapsulation and scalability.

#### 6.3.2.1 Code separation

By following the multi-tier approach in developing the proximity calculator application, code can be separated by function. Code in the Information tier was concerned only with assessing or storing permanent data. Likewise, software written

for the middle tier was concerned with the actual proximity calculations. Similarly, the applications developed in the presentation-tier were focused on displaying results.

The code separation helped tremendously with software organization. It allowed for a logical grouping of software functionality. Furthermore, we employed J2EE standards such as Java Database Connectivity (JDBC), JavaBeans, Servlets and Java Server Pages (JSPs) to simplify inter-tier communications.

#### **6.3.2.2 Encapsulation**

We employed encapsulation to isolate and hide complex software functionality. For example, the `GlobalProximityCalculator` class only works on a `Graph` object to perform the proximity calculations. The `GlobalProximityCalculator` knows nothing about the underlying database that actually houses the data in the graph object. This allowed us to focus on the implementation of the `GlobalProximityCalculator` without having to be concerned about the operations of the underlying database. The separation of code by functionality not only simplified development, but improved maintainability.

#### **6.3.2.3 Scalability**

We have used many tenants of the J2EE standard, but not all. Enterprise JavaBeans (EJBs) are also part of the J2EE standard. An EJB is a body of code that implements modules of business logic (Armstrong, 2004). EJBs are always deployed within an EJB container which provides support services for distributed applications, such as security protocols, authentication and distributed resource directory services. We designed the proximity calculator application to easily incorporate EJBs.

## **6.3.3 DAO**

### **6.3.3.1 Purpose**

The data access layer is implemented using a data access object (DAO). The DAO is the interface between permanent storage and the middle layer. All of the database specific code is contained here. The advantage of this approach is that it hides the details of the database access from the business layer. Furthermore, through the use of generic classes, one is able to switch database implementations without any modifications to the business layer.

### **6.3.3.2 Structure**

The DAO actually consists of an interface and the database specific concrete class that implements it, and another interface that defines the data access methods available to client applications. When a client application wants to instantiate a graph object that is in the database, it works through the DistanceManager interface. The client application must pass it a DAO object that extends the GenericDAO interface. The GenericDAO interface defines the database operations required to support the DistanceManager. The DAO classes that extend the GenericDAO interface house the database vendor specific code for accessing that particular database. For example, we use the MySQL database system. The class we developed (MySQLDAO) that extends the GenericDAO contains all of the SQL statements specifically suited for the MySQL database management system.

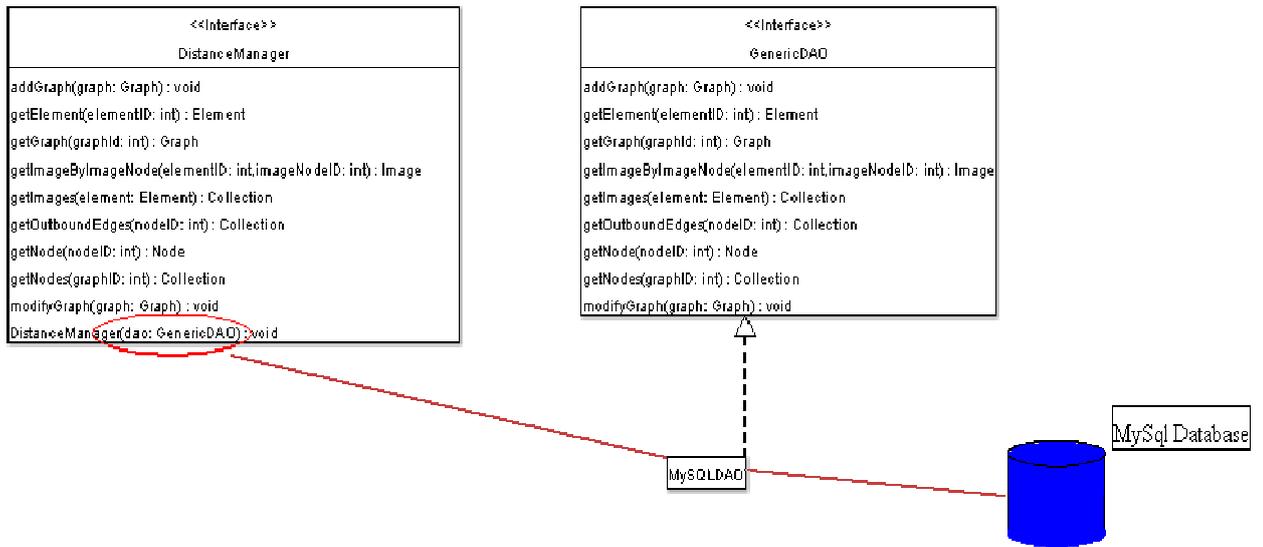


Figure 64 DAO related classes

## 6.4 DB Design and Implementation

### 6.4.1 Introduction

We chose the Unified Modeling Language (UML) to capture the design of the database that supports the Linea agent. UML is a broadly accepted standard for modeling software applications. It is extremely rich and flexible, able to express diverse aspects of a development project. Accordingly, we were able to consistently use UML for not only the database design, but for the entire system. Using one, rich language simplified the analysis and design of the entire system.

In this section, we will detail the steps we took in designing the database. Specifically, we will discuss our analysis with object modeling and use case diagrams. We will also explain the class diagram development. Finally, we will discuss our technique for converting class diagrams into database schemas.

## 6.4.2 Object modeling

When designing the Linea database, we followed the basic practices of Object Oriented Analysis (OOA). We started by considering our domain and then extracting descriptive nouns and verbs.

- A graph contains nodes
- A graph contains edges
- Nodes contain elements
- Elements have image sets

We then used these phrases to construct a first cut of our data object model. We did so by converting all nouns to classes. Next we created associations between the classes whenever we had a verb phrase containing the names of the classes.

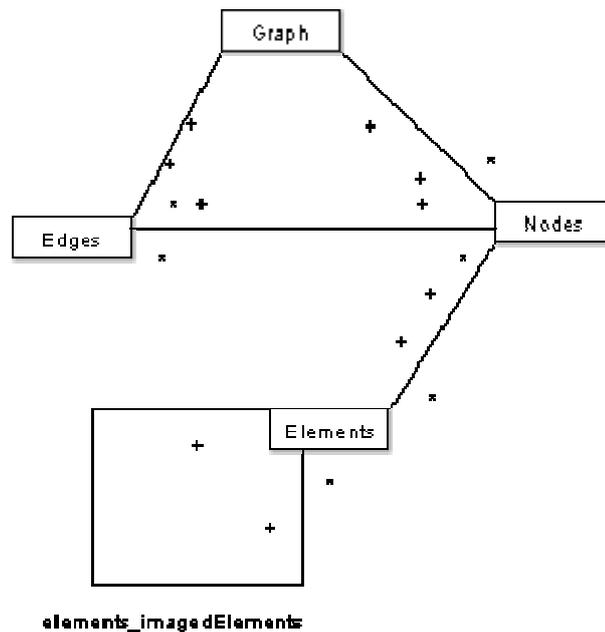


Figure 65 Data object model - first cut

However, as the name implies, this is only a first step. We performed more analysis to capture other data requirements that may not have been apparent from the noun-verb analysis. The next step we performed was use case analysis.

### 6.4.3 Use cases

We use a use case diagram to capture the functional aspects of the database within the context of the entire system (Chitnis et al, 2005). Rather than looking at the problem of proximity calculations, with use cases, we focus on *how* we calculate the proximity calculation. Use cases are another way we can consider the problem that can possibly provide more insight on data requirements.

In Figure 66, we provide an example of the Linea use case. This simple use case contains 2 actors, a client who wishes to know the proximity between 2 elements and the database. The primary use case associated with the Linea client is *calculate proximity*. This use case includes 2 other use cases: *calculate local proximity* and

*calculate image proximity*. These 2 included use cases are associated with the database actor in order to obtain the domain graph information.

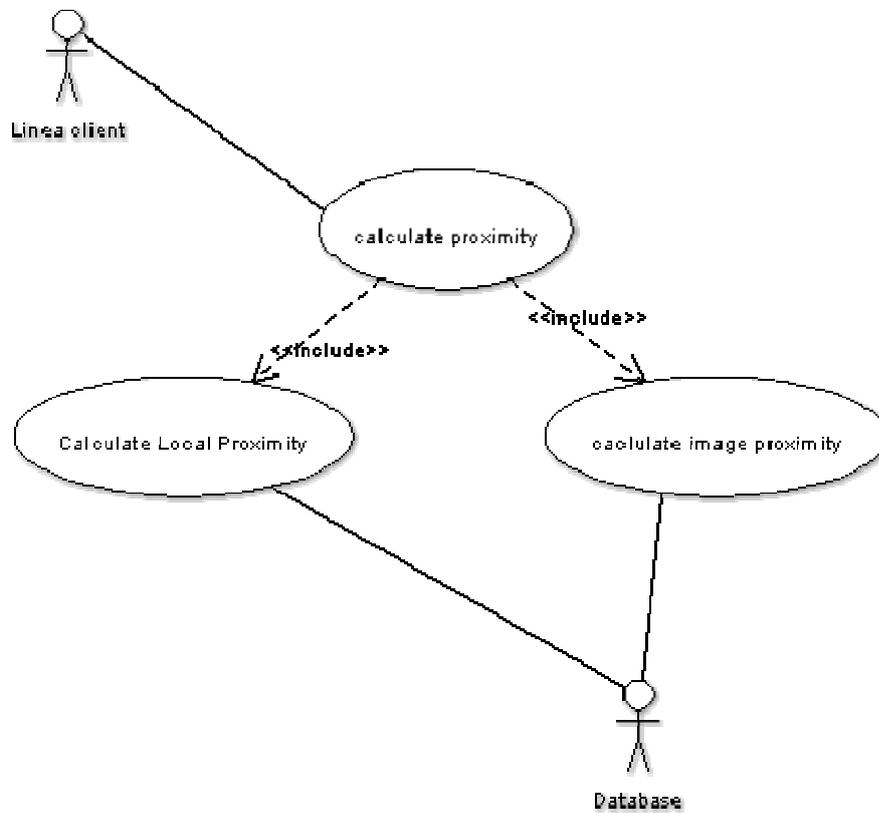


Figure 66 Linea use case diagram

---

Through this analysis, we realized that we needed to capture the local proximity measures between elements in a node. For this work, we stored local proximities in the database. Hence, this needed to be captured in our finalized data model.

#### 6.4.4 Class diagrams

We used the class diagram to provide a logical view of the database design. It is based on the original object model we developed, with consideration for other

analysis such as the use case diagrams. We can consider the object model we developed in the previous section a ‘first draft’ while the class diagram is the ‘final cut.’ As you can see in Figure 67 we have extended the original object model to include the local proximities. We also included attributes for each class.

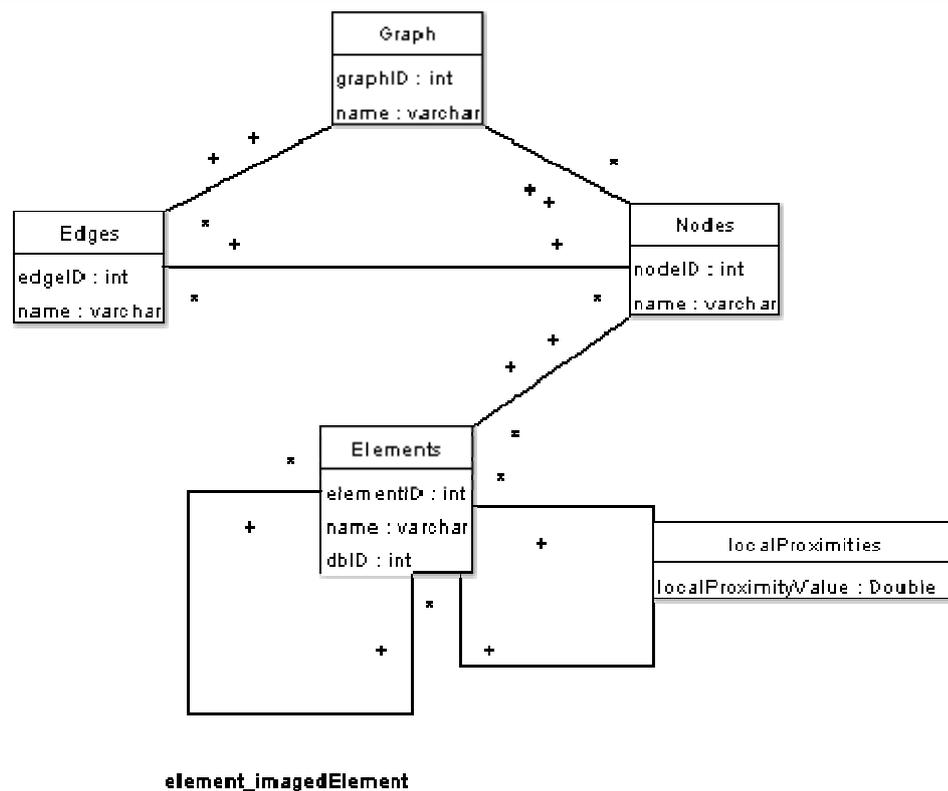


Figure 67 Database class diagram

### 6.4.5 Database schema conversion

The bridge for our design and implementation was the conversion of the UML diagram into schema. We employed the UML-to-schema guide developed by the author while teaching a database design course at the United States Military Academy at West Point. The guide consists of a set of pattern matching rules that allows a class diagram to be translated to a database schema almost automatically. The rules

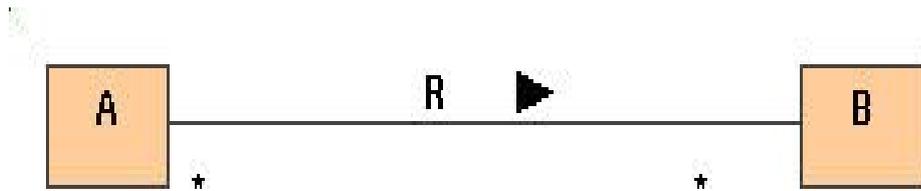
are organized with the most frequently used and necessary ones first. The two most basic conversion rules, for strong classes and many-to-many associations are shown in Figures 68 and 69



if  $\text{key-attributes}_A = \{ \text{keyAttr}_1, \dots, \text{keyAttr}_j \}$   
and  $\text{other-attributes}_A = \{ \text{singleAttr}_1, \dots, \text{singleAttr}_k \}$   
then  $A\text{-schema} = ( \underline{\text{key-attributes}}_A, \text{other-attributes}_A )$

Figure 68 Guide to converting a strong class to table schema

---



if A-schema = ( key-attributes<sub>A</sub>, other-attributes<sub>A</sub> )

and B-schema = ( key-attributes<sub>B</sub>, other-attributes<sub>B</sub> )

then R-schema = ( key-attributes<sub>A</sub>, key-attributes<sub>B</sub>, other-attributes<sub>R</sub> )

with a foreign key from R-schema.key-attributes<sub>A</sub> to A-schema.key-attributes<sub>A</sub>

and a foreign key from R-schema.key-attributes<sub>B</sub> to B-schema.key-attributes<sub>B</sub>

---

Figure 69 Guide to converting a many-to-many association to table schema.

---

In the last section of the guide, there are ways to optimize the schema. An example, which is to drop an association table when the association is an existence dependency, is shown below. If the many-to-many association was an existence dependency association, the association table could be dropped and the table schema representing the class on the many side of the association (in this case A-schema) would be altered.



if A-schema = ( key-attributesA, other-attributesA )

and B-schema = ( key-attributesB, other-attributesB )

and R-schema = ( key-attributesA, key-attributesB, other-attributesR )  
with foreign keys and unique constraints as defined earlier

then drop R-schema

and alter A-schema=( key-attributesA, other-attributesA, key-attributesB, other-attributesR )  
with a foreign key from A-schema.key-attributesB to B-schema.key-attributesB

and a unique constraint on A-schema.key-attributesB

Figure 70 Guide to optimizing the schema by eliminating unneeded many-to-one table schema.

---

Given this guide (which can be found in annex B), we present the schema that we developed from our class diagram. The database used throughout our system was implemented from this schema.

---

```

Strong classes
Graph_schema = {graphID, name}
Edges_schema = {edgeID, name}
Elements_schema = {elementID, name}
Nodes_schema = {nodeID, name}

Many-many
element-node_schema = {elementID, nodeID}
    foreign key reference from elementID -> Elements.elementID
    foreign key reference from nodeID -> Nodes.nodeID
edge-node_schema = {edgeID, nodeID}
    foreign key reference from edgeID -> Edges.elementID
    foreign key reference from nodeID -> Nodes.nodeID
element_imagedElement = {elementID, imagedElementID}
    foreign key reference from elementID -> Elements.elementID
    foreign key reference from imagedElementID -> Elements.elementID
localProximities = {firstElementID, secondElementID, localProximityValue}
    foreign key reference from firstElementID -> Elements.elementID
    foreign key reference from secondElementID -> Elements.elementID

Many-one
edge-graph = {edgeID, graphID}
    foreign key reference from edgeID -> Edges.elementID
    foreign key reference from graphID -> Graphs.nodeID
graph-node = {nodeID, graphID}
    foreign key reference from nodeID -> Nodes.nodeID
    foreign key reference from graphID -> Graphs.nodeID

```

Figure 71 Table schema for the graph database

---

## 6.5 Implementation Improvements

Throughout the development of the Linea agent, there were some implementation inefficiencies that needed to be overcome. Specifically, we addressed problems with object instantiation and initialization.

### 6.5.1 Object instantiation efficiency

The first challenge we encountered was that the system continuously crashed during initialization in certain situations. For small data sets, such as the web pages database, the system would run fine. However, for larger data sets, such as the corporate database, it would crash.

After extensive debugging we discovered that we had inadvertently developed the software to instantiate a new DAO every time an element was pulled from the database into the graph object. For small databases, this didn't pose a problem.

However, for the corporate database with over 2700 elements and 6 million local distance values, memory was quickly depleted.

We easily corrected our oversight and adjusted the software by only instantiating one DAO object per application and reusing it for all database accesses. After making the correction, the system did not crash at initiation anymore.

### **6.5.2 Lazy local proximity initialization**

Shortly after solving the problem with object initialization, we discovered another challenge. The system continued to crash with large data sets. After performing more program analysis, we discovered the root cause to be the initialization of the local proximity values. The graph object contains nodes and edges. A node contains a set of elements called a payload. This payload object contains a collection of local proximity values of type `BigDecimal` collected from the database at initialization. For the corporate data set, this included over 6 million items. As with the object initialization, the system ran out of memory at instantiation. Upon further analysis, we determined that all of the local proximity values were not needed to start the proximity calculations. As a result, we implemented a ‘lazy’ local proximity initialization scheme. In effect, when a graph was instantiated, the local proximity values would not be read in. Instead, they would only be read in if needed. More specifically, we created an  $n \times n$  table for each node where  $n$  was the number of elements in the node. The table was initialized to -1. During the proximity calculations, when there was a need for a local proximity value, this table is consulted. If the value was -1, then the system would make a database call to fetch the value and then store it into the table. If a non-negative value was already in the table, then it would simply be read.

After this adjustment, the system worked without problems. In hindsight, we believe the ‘lazy’ initialization is a better approach since it not only reduces the need for memory, but also improves speed by only making a database access when needed.

## **6.6 Conclusion**

In this chapter, we presented the software implementation of the Linea proximity calculator. We first gave an overview of the system architecture. Next we explained our multi-tier development approach. And after a detailed discussion of our supporting database, we concluded with some implementation improvements that have been made.



# Chapter 7

## Conclusion

We have presented through this dissertation the Linea algorithm, a new approach for measuring the proximity between 2 objects in a linked metric space. This linked metric space is represented within a graph. The algorithm is generic and has been designed to be easily applied to various domains.

Along with the algorithm we presented three implementation approaches: naïve, iterative and bottom-up. The naïve approach implements the theoretical algorithm and uses an artificial fix point. The bottom-up works on a modified tree version of the support domain graph. In this case, we are able to take advantage of certain preprocessing steps that improve our efficiency. Finally, the iterative approach begins with an initial value for all proximities in the graph then iterates until the difference between successive proximity values in the system is less than a pre-defined precision limit. We compared the three approaches and provided insight on the situations they are best used in.

### **7.1 Summary of Contributions**

#### **7.1.1 A novel approach to metric space calculations**

The Linea algorithm is a novel approach to measuring metric space calculations. It provides an efficient and generic way of determining the proximity

between two objects in a linked metric space. It is optimized for linked environments with minimal, semi-structured text. From our research of the domain, we have found no other approaches that combine both the direct proximity measurements and the link analysis. Our work, in partnership with I-nova, will be applied to developing innovative and flexible solutions for I-nova's many and diverse customers.

### **7.1.2 A comparison of implementation methods**

We also proposed and provided a comparison of implementation methods. In addition, we described situations when a particular approach is best used. In particular we found that the naïve approach was good for single (or a few) calculations. This is because the naïve approach had very little preprocessing time. By contrast we found that the iterative approach was best for large numbers of calculations such as batch jobs. Although the one-time preprocessing step was long, all subsequent calculations were negligent.

### **7.1.3 Linea implementation**

The Linea algorithm is implemented and can be used for further research. We have a version for each of the three implementation methods described in this work.

## **7.2 Directions for Future Research**

There are various areas of improvement and further research. They include Response integration, Netfires, parallelization, and web services.

### **7.2.1 Response integration**

This work was performed in partnership with I-nova. A design goal was for Linea to be able to be integrated into various domains. One of those domains is helpdesk support. Response, a European Union funded research project, has a goal of applying artificial intelligence technologies to the helpdesk support domain. In

particular, the Response system employs case-based reasoning and Bayesian analysis techniques to help determine whether a new incident received at a helpdesk is a known case with a solution.

How to employ Linea into the Response project is an interesting area of research. There are two areas to consider. In order to determine if a group of incidents make-up a case, the project will use clustering techniques. Can Linea be used effectively as the proximity algorithm when determining clusters? Second, can Linea be used to measure the proximity between a new incident and an already established case? This area of research could further support the generic nature of the Linea agent

### **7.2.2 Netfires**

Netfires is a next-generation munitions project sponsored by DARPA. Since the times of Napoleon, artillery has been an important weapon on the battlefield. When friendly forces would come under attack from the adversary, artillery rounds would often be fired in response with devastating effectiveness. Nevertheless, there always has been the time delay between when friendly forces needed the support from artillery and when it actually arrived on target. Through many communication advancements over the last two centuries, this delay has been minimized. However, there still exists the delay because normally, artillery forces are not located on the front-line. Instead they are often several kilometers behind lines. Hence, even if artillery units receive a request of support, the time it takes for a round to travel after being fired can be minutes.

DARPA seeks to radically change the way artillery, or more generally, indirect fires are executed. Instead of waiting for a request from friendly forces for indirect fire, artillery units would fire their rounds before a unit becomes engaged with the enemy. However, instead of traveling point to point, like a bullet from a gun, the next-generation rounds would loiter over the battlefield, awaiting a request from

friendly forces below. The rounds would exhibit intelligent behavior by flying in a flocking formation. Once there is a request for artillery fire, the loitering rounds decide among themselves who is best qualified and the chosen round or rounds then fall from the sky. The response time now reduces from minutes to seconds.

One area of research is how do the rounds decide among themselves to respond to the request for fire support? Another way to look at the problem is to frame it differently. The problem can be seen as a proximity question. How do you determine which round is closest to the target of the fire request? Proximity in this case is not limited to physical distance between the round and the target. It would also include type of round, type of target, and remaining fuel on the round, among others. Hence, Linea could also be applied to this domain.

### **7.2.3 Parallelization**

The iterative and bottom-up approaches have potential applications in web-based applications that require many proximity calculations. However, the initial calculation step is very expensive. There are opportunities to improve Linea's performance with parallelization. An interesting problem would be to investigate and apply various parallelization techniques to the iterative and bottom-up implementations and compare their performances.

### **7.2.4 Web Service**

The Linea agent is generic by design. It would be very interesting to extend it as a web service. This would allow it to be used by various applications much simpler. The interface would need to be analyzed to determine how to support a web services interface. We have already implemented the Linea agent as a `JavaServlet`.

# Bibliography

- Adomavicius, G., Sankaranarayanan, R., Sen, S., Tuzhilin, A. "Incorporating contextual information in recommender systems using a multidimensional approach." *ACM Transactions on Information Systems*, Vol. 23, No. 1, January 2005, pp. 103-145.
- Adomavicius, G., Tuzhilin, A., "Multidimensional recommender systems: a data warehousing approach," *Proceedings of the 2nd International Workshop on Electronic Commerce (WELCOM'01)*. *Lecture Notes in Computer Science*, vol 2232, Springer-Verlag, 180-192.
- Amato G., Rabitti F., Savino P., Zezula P., "Approximate Similarity in Metric Data by Using Region Proximity" *First DELOS Network of Excellence Workshop—Information Seeking, Searching and Querying in Digital Libraries*, Zurich, Switzerland, 11-12 December 2000, Sophia Antipolis, Editions ERCIM, p 101-106.
- Ansari, A., Essegai, S., Schaal, S. "Internet recommendation systems." *Journal of Market Research*, vol 37, no 3, 2000, pp. 363-375.
- Armstrong, E., J. Ball, S. Bodoff, D. Carson, I. Evans, D. Green, K. Haase, E. Jedrock. *The J2EE™ 1.4 Tutorial*, Second Edition. Boston: Addison-Wesley, 2004.
- Arya S., Mount D., Nethanyahu N., Silverman R., Wu A., "An Optimal Algorithm for Approximate Nearest Neighbor Searching in Fixed Dimensions" *Journal of the ACM*, vol. 45, n° 6, November 1998, p. 891-923.
- Baeza-Yates, R., Cunto, W., Manber, U., Wu, S. "Proximity matching using fixed-query trees. *Proceedings of the 5<sup>th</sup> Combinatorial Pattern Matching (CPM'94)*, LNCS 807, pages 198-212, 1994
- Balabanovic, M. and Shoham, Y. "Fab: Content-based, collaborative recommendation." *Communications of the ACM*, Vol. 40, o. 3, 1997, pp. 66-72.
- Berry W., Raghavan P., Zhang X., "Symbolic Preprocessing Techniques for Information Retrieval Using Vector Space Models" *Computational Information Retrieval*, Berry, M., ed., SIAM, Philadelphia, 2001.
- Bharat K., Henzinger M., "Improved Algorithms for Topic Distillation in a Hyperlinked Environment" *Proceedings of the 21st annual international ACM SIGIR conference on Research and development in information retrieval*, Melbourne, Australia, 1998, p. 104-111.

- Blom, K., Ruhe, A., "Information Retrieval Using Very Short Krylov Sequences." Computational Information Retrieval, Berry, M., ed., SIAM, Philadelphia, 2001.
- Breese, J, Heckerman, D., Kadie, C. "Empirical analysis of predictive algorithms for collaborative filtering," Proceedings of the 14th Conference on Uncertainty in Artificial Intelligence, Madison, WI, 1988.
- Brin S., Page L., "The Anatomy of a Large-Scale Hypertextual Web Search Engine" Proceedings of the World Wide Web Conference 1998, Brisbane, 1998.
- Bustos, B., Navarro, G., Chávez, E. "Pivot Selection Techniques for Proximity Searching in Metric Spaces" Proceedings of the XXI Conference of the Chilean Computer Science Society (SCCC'01), pages 33-40. IEEE CS Press, 2001.
- Chahlaoui Y., Galivan K., Van Dooren P., "An Incremental Method for Computing Dominant Singular Spaces" Computational Information Retrieval, Berry, M., ed., SIAM, Philadelphia, 2001.
- Chakrabarti S., Dom B., Raghavan P., Rajagopalan S., Gibson D., Kleinberg J., "Automatic resource compilation by analyzing hyperlink structure and associated text." Proceedings of the World Wide Web Conference 1998, Brisbane, 1998.
- Chávez, E., Navarro, G. "An Effective Clustering Algorithm to Index High Dimensional Metric Spaces" Proceedings of the Seventh International Symposium on String Processing and Information Retrieval(SPIRE'00), September 27 - 29, 2000 ,Coruña, Spain.
- Chavez E., Navarro G., Baeza R., Marroquin J., "Searching in Metric Space" ACM Computing Surveys, Vol. 33, No. 3, September 2001, p. 221-273.
- Chitnis, M., Tiwari, P., Ananthamurthy, L. "Creating use-case diagrams," found online at develop.com accessed from <http://www.developer.com/design/article.php/2109801>, 19 March 2005.
- Condliff, M., Lewis, D., Madigan, D., Posse, C. "Bayesian mixed-effects models for recommender systems," ACM Special Interest Group on Information Retrieval (SIGIR'99) Workshop on Recommender Systems: Algorithms and Evaluation. 1999.
- Dhillon, I., Guan, Y., Kogan, J. "Iterative Clustering of High Dimensional Test Data Augmented by Local Search." Proceedings of the 2002 IEEE International Conference on Data Mining. December 9 - 12, 2002. Maebashi TERRSA, Maebashi City, Japan.
- Ding C., "A Probabilistic Model for Latent Semantic Indexing in Information Retrieval and Filtering" Computational Information Retrieval, Berry, M., ed., SIAM, Philadelphia, 2001.
- Elsner, L., Driessche, P. "On the Power Method in Max Algebra," "Linear Algebra and its Applications, Vol 302-303, No. 1-3, 1999, pp 17-32.
- Garfield E., "Citation Analysis as a Tool in Journal Evaluation" Science, Vol. 178, No. 4060, 1972, p. 471-479.
- Google, <http://www.google.com>, 2004.

- Grosse, I., Bernaolo-Galván, P., Carpena, P., Román-Roldán, R., Oliver, J., Stanley, H. "Analysis of symbolic sequences using the Jensen-Shannon divergence." *The American Physical Society*, Vol. 65, 2002
- Hammouda, K., Kamel, M. "Incremental Document Clustering Using Cluster Similarity Histogram" *Proceedings of the IEEE/WIC International Conference on Web Intelligence 2003*. October 13-16 2003, Halifax, Canada.
- Haveliwala T., Gionis A., Klein D., Indyk P., "Evaluating Strategies for Similarity Search on the Web," *Proceedings of the World Wide Web Conference 2002*, Honolulu, Hawaii USA 2002.
- Holt F., Wu J., "Information Retrieval and Classification with Subspace Representations" *Computational Information Retrieval*, Berry, M., ed., SIAM, Philadelphia, 2001.
- Huggins, K., Carteret, D. "A graph-based, metric space proximity calculator for Internet objects," *Colloque International sur la Fouille de Textes (CIFT'2004)*, La Rochelle, France 23-25 June 2004.
- Ingongngam, P., Rungsawang, A. "Topic-Centric algorithm: A novel to web link analysis," *Proceedings of the 18th International Conference on Advanced Information Networking and Application (AINA'04)*, March 29 - 31, 2004. Fukuoka, Japan
- Jain, A., Murty, M., Flynn, P. "Data Clustering: A Review," *ACM Computing Surveys*, Vol. 31, No. 3, September 1999.
- Kleinberg J., "Authoritative Sources in a Hyperlinked Environment," *Proceedings of the Ninth Annual ACM-SIAM Symposium on Discrete Algorithms*, 25-27 January 1998, San Francisco, CA USA.
- Kogan J., "Clustering Large Unstructured Document Sets." *Computational Information Retrieval*, Berry, M., ed., SIAM, Philadelphia, 2001.
- Lempel R., Moran S., "SALSA: The Stochastic Approach for Link-Structure Analysis" *ACM Transactions on Information Systems*, Vol. 19, No. 2, April 2001, pp. 131-160.
- Lin, K, Kondadadi, R. "A Similarity-Based Soft Clustering Algorithm for Documents" *Proceedings of the Seventh International Conference on Database Systems for Advanced Applications*. April 18 - 21, 2001. Hong Kong, China.
- Mandhani, B., Sachindra, J., Kummamuru, K. "A Matrix Density Based Algorithm to Hierarchically Co-Cluster Documents and Words." *Proceedings of the WWW 2003*. May 20-24, 2003, Budapest, Hungary.
- Micó, L., Oncina, J., Vidal, E. "A new versión of the nearest-neighbor approximating and eliminating search (AESAs) with linear preprocessing-time and memory requirements" *Pattern Recognition Letters*, 15:9-17, 1994.
- O'Mahony, M., Hurley, N., Kushmerick, N., Silvestre, G. "Collaborative Recommendation: A Robust Analysis." *ACM Transactions on Internet Technology*, Vol. 4, No. 4, November 2004, pages 344-377.
- Park H., Jeon M., Ben Rosen J., "Lower Dimensional Representation of Text Data in Vector Space Based Information Retrieval." *Computational Information Retrieval*, Berry, M., ed., SIAM, Philadelphia, 2001.

- Patel, M. "Distance Measures," accessed on 9 August 2004 from [http://www.ucl.ac.uk/oncology/MicroCore/HTML\\_resource/Distances\\_detailed\\_opup.htm](http://www.ucl.ac.uk/oncology/MicroCore/HTML_resource/Distances_detailed_opup.htm).
- Pazzani, M. "A framework for collaborative, content-based and demographic filtering," *Artificial Intelligence Review*, Vol. 13, No. 5/6, 1999, pp. 393-408.
- Pillai, S., Suel, T., Cha, S. "The Perron-Frobenius Theorem," *IEEE Signal Processing Magazine*, March 2005, pages 62-75.
- Pottenger W., Yang T., "Detecting Emerging Concepts in Textual Data Mining." *Computational Information Retrieval*, Berry, M., ed., SIAM, Philadelphia, 2001.
- Press, W., Teukolsky, S., Vetterling, W., Flannery, B. *Numerical Recipes in C – The Art of Scientific Computing*. New York: Cambridge University Press, 1995.
- Resnick, P., Iakovou, N. Sushak, M., Bergstrom, P., Riedl, J. "GroupLens: An open architecture for collaborative filtering of netnews," *Proceedings of the 1994 Computer Supported Cooperative Work Conference*.
- Salton G., Wong A., Yang C.S. "A Vector Space Model for Automatic Indexing" *Communications of the ACM* Vol. 18, No. 11, November 1975.
- Saraw, B., Karypis, G., Konstan, Riedl, J. "Item-based collaborative filtering recommendation algorithms," *Proceedings of the 10th International WWW Conference*, 2001.
- Shardanand, U., Maes, P. "Social information filtering: Algorithms for automating 'word of mouth'," *Proceedings of the Conference on Human Factors in Computing Systems*, 1995.
- Small H., Koenig M. "Journal clustering using bibliographic coupling method." *Information Processing & Management*, Vol. 13, Issue 5, 1977, pp. 277-28.
- Steinbach, M., Karypis, G., Kumar, V. "A Comparison of Document Clustering Techniques," *The Sixth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD-200)*, August 20 - 23, 2000 Boston, MA, USA.
- Van Rijsbergen, C., "Information Retrieval" accessed from <http://www.dcs.gla.ac.uk/~iain/keith/index.htm> 2 October 2004
- Vidal, E. "An algorithm for finding nearest neighbors in (approximately) constant average time" *Pattern Recognition Letters*, 4:145-157, 1986.
- Yianilos, P. "Data structures and algorithms for nearest neighbor search in general metric spaces. *Proceedings of the 4th ACM-SIAM Symposium on Discrete Algorithms (SODA'93)*, pages 311-321, 1993.
- Zha, H., Ding, C., Gu, M. "Bipartite Graph Partitioning and Data Clustering" *CIKM'01*, November 5-10, 2001, Atlanta, Georgia, USA.

# Annex A

## A Simple Example

We give a simple example of constructing proximity equations and solving them using various approaches. Consider two nodes with 3 elements each.

### A.1 Example and its graph

We show the individual links between the elements (instead of showing relationships between nodes). Each directed link points to an element in the image set of the element from which the link originated. For example, in Figure 62, Element  $a$  in Node 1 has two image set elements in Node 2. They are elements  $x$  and  $y$ .

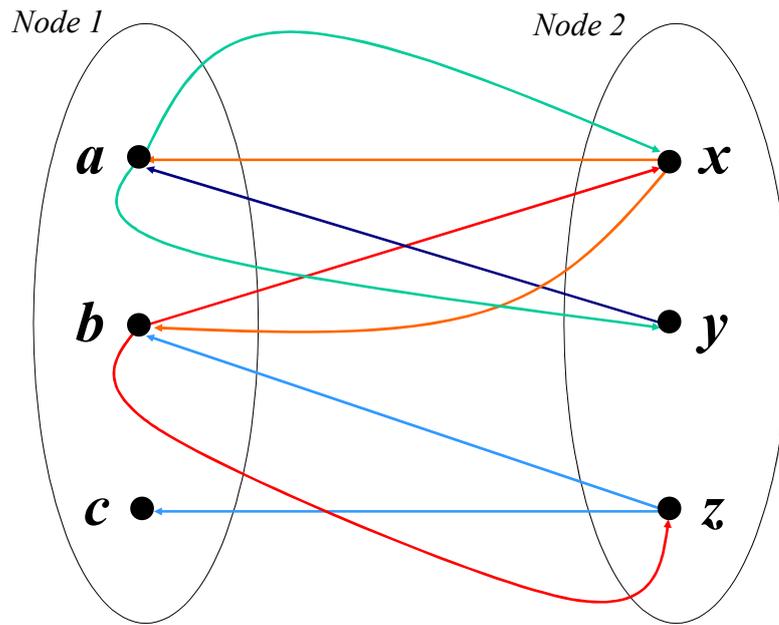


Figure 72 Sample graph

## A.2 Equations of the Linea model for this graph

For this annex, we simplify the notation as follows:

$xy$ : is the proximity between elements  $x$  and  $y$ .

$a[bc]$ : is the proximity between element  $a$  to the set of elements  $\{b, c\}$

$$a[bc] = [ab, ac]$$

$(ab)(cd)$ : is the proximity between the sets  $\{a, b\}$  and  $\{c, d\}$

We handle this example model with the assumptions that all local proximities are zero except for the identity case, which is one. The basic equations we begin with are:

$$xy = (ab)(a)$$

$$yz = (a)(bc)$$

$$xz = (ab)(bc)$$

$$ab = (xy)(xz)$$

$$ac = 0$$

$$bc = 0$$

### A.3 Solving the equations

We observe that:

$$\begin{aligned}(ab)(a) &= 1/3 (a[ab] + aa + ba) \\ &= 1/3 (2 + ab)\end{aligned}$$

Transferring the equations using the same method gives:

$$\begin{aligned}xy &= 1/3 (2 + ab) \\ yz &= 1/3 (ab + ac + a[bc]) = 2/3 ab \\ xz &= 1/4 (2 + a[bc] + c[ab]) = 1/4 (2 + ab) \\ ab &= 1/4 (2 + y[xz] + z[xy]) \\ ac &= 0 \\ bc &= 0\end{aligned}$$

Now observe that:

$$y[xz] = [xy, xz] = \left[ \frac{1}{3}(2 + ab), \frac{2}{3}ab \right]$$

Since  $0 \leq ab \leq 1$ ,

$$y[xz] = \frac{(2+ab)}{3}$$

and thus:

$$z[xy] = \frac{1}{4}(2 + ab)$$

The equation given ab can be rewritten:

$$ab = \frac{1}{2} + \frac{1}{12}(2 + ab) + \frac{1}{16}(2 + ab)$$

thus giving the final values as:

$$\begin{aligned}ab &= 38/41 \\ xy &= 40/41 \\ yz &= 76/123 \\ xz &= 30/41\end{aligned}$$

## A.4 Iterative approach to solving the equations

We will now present a solution that applies an iterative approach to solving the equations. Our approach stabilizes very quickly.

Step 1 : We start with an initial vector of 6 values, all equal to 1 except for those which have an already known value :

$$ab_0=1$$

$$xy_0=1$$

$$yz_0=1$$

$$xz_0=1$$

For convenience we do not repeat for each step that  $ac=0$  and  $bc=0$

We then compute the new values of the 4 unknown variables :

$$ab_1 = .5 + .25 [xy_0, xz_0] + .25 [zx_0, zy_0] = 1$$

$$xy_1 = .333 (2 + ab_0) = 1$$

$$yz_1 = .666 ab_0 = .666$$

$$xz_1 = .25 (2 + ab_0) = .75$$

Step 2: using the same equations, we compute  $ab_2, xy_2, yz_2, xz_2$ , using the values obtained at the end of step 1 :

$$ab_2 = .5 + .25 + .25 \times .75 = .938$$

$$xy_2 = 1$$

$$yz_2 = .666$$

$$xz_2 = .75$$

Step 3 : we compute in the same way  $ab_3, xy_3, yz_3, xz_3$ , using the values obtained at the end of step 2 :

$$ab_3 = .938$$

$$xy_3 = .979$$

$$yz_3 = .625$$

$$xz_3 = .735$$

Step 4 : it yields  $ab_4 = .928, xy_4 = .979, yz_4 = .625, xz_4 = .735$  Notice that these values are very close to those obtained by direct computation :

$$ab=38/41, xy = 40/41, yz = 76/123, xz = 30/41$$

## A.5 A completely linear approximation

In order to make the model completely linear, we may be can replace max (a,b,c) with (a+b+c)/3. Of course, the model changes : this means that the proximity of an Element x to a set is not more the max of the proximities of x to all elements of the set, but the average proximity of x to these elements. We get a set of 6 linear equations:

$$ab = 1/8 xy + 1/4 yz + 1/8 xz + 1/2$$

$$ac = 0$$

$$bc = 0$$

$$xy = 1/3 ab + 2/3$$

$$yz = 2/3 ab$$

$$xz = 1/4 ab + 1/2$$

Solving this system by a direct method, like for instance the Gauss-Jordan elimination method, yields the exact solution:

$$xz = 3588/5037 = .712 , ab = .849 , yz = .506 , xy = .919$$

Of course we may also solve this system by the Gauss-Seidel iterative method. This looks like the iterative method employed on the other system. Results are:

$$\text{Initial values : } ab = 1, ac=0, bc = 0, xy = 1, yz = 1, xz = 1$$

$$\text{Step 1 : } ab = 1, xy = 1, yz = .666, xz = .75$$

$$\text{Step 2 : } ab = .884, xy = 1, yz = .666, xz = .75$$

$$\text{Step 3 : } ab = .884, xy = .961, yz = .589, xz = .721$$

$$\text{Step 4 : } ab = .857, xy = .961, yz = .589, xz = .721$$

Convergence is rather good, though a little bit slower than with the “max-plus” algebra.



# Annex B

## UML to Schema guide

### UML to Table Schema Conversion - Generating Rules

1. Convert strong classes to table schema.

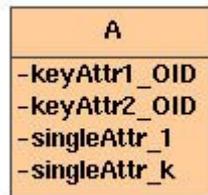


Figure 73 Strong class

---

if  $\text{key-attributes}_A = \{ \text{keyAttr}_1, \dots, \text{keyAttr}_j \}$   
and  $\text{other-attributes}_A = \{ \text{singleAttr}_1, \dots, \text{singleAttr}_k \}$   
then  $A\text{-schema} = ( \underline{\text{key-attributes}_A}, \text{other-attributes}_A )$

*Important:* Multi-valued attributes are covered in a separate rule.

2. Convert all weak classes to table schema.

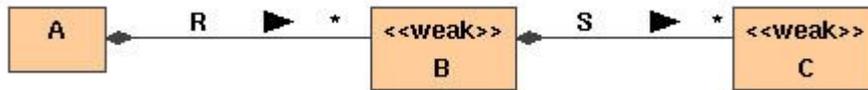


Figure 74 Weak class

if A-schema = ( key-attributes<sub>A</sub>, other-attributes<sub>A</sub> )

then B-schema = ( key-attributes<sub>A</sub>, discriminators<sub>B</sub>, other-attributes<sub>B</sub> )

with a foreign key from B-schema.key-attributes<sub>A</sub> to A-schema.key-attributes<sub>A</sub>

and C-schema = ( key-attributes<sub>A</sub>, discriminators<sub>B</sub>, discriminators<sub>C</sub>, other-attributes<sub>C</sub> )

with a foreign key from C-schema.key-attributes<sub>A</sub>, discriminators<sub>B</sub>

to B-schema.key-attributes<sub>A</sub>, discriminators<sub>B</sub>

*Important:* "Chained" weak classes should be handled from "strongest to weakest". Also, you don't need schema for the weak associations (R-schema & S-schema).

### 3. Convert all super-/sub-classes to table schema.

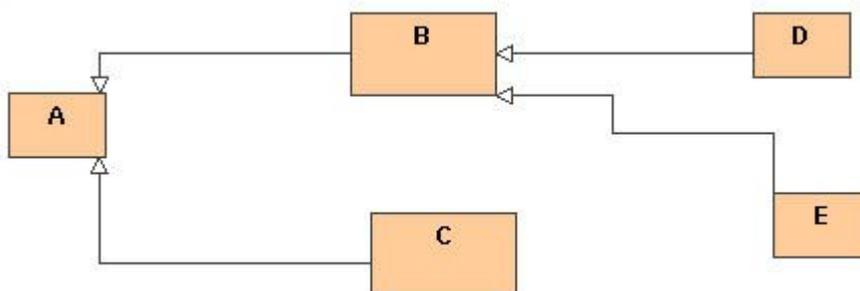


Figure 75 Super/sub classes

if A-schema = ( key-attributes<sub>A</sub>, other-attributes<sub>A</sub> )

then B-schema = ( key-attributes<sub>A</sub>, other-attributes<sub>B</sub> )

with a foreign key from B-schema.key-attributes<sub>A</sub> to A-schema.key-attributes<sub>A</sub>

and C-schema = ( key-attributes<sub>A</sub>, other-attributes<sub>C</sub> )

with a foreign key from C-schema.key-attributes<sub>A</sub> to A-schema.key-attributes<sub>A</sub>

also D-schema = ( key-attributes<sub>A</sub>, other-attributes<sub>D</sub> )

with a foreign key from D-schema.key-attributes<sub>A</sub> to B-schema.key-attributes<sub>A</sub>

and E-schema = ( key-attributes<sub>A</sub>, other-attributes<sub>E</sub> )

with a foreign key from E-schema.key-attributes<sub>A</sub> to B-schema.key-attributes<sub>A</sub>

#### 4. Convert all many-to-many associations to table schema.



Figure 76 Many-to-many associations

---

if A-schema = ( key-attributes<sub>A</sub>, other-attributes<sub>A</sub> )

and B-schema = ( key-attributes<sub>B</sub>, other-attributes<sub>B</sub> )

then R-schema = ( key-attributes<sub>A</sub>, key-attributes<sub>B</sub>, other-attributes<sub>R</sub> )

with a foreign key from R-schema.key-attributes<sub>A</sub> to A-schema.key-attributes<sub>A</sub>

and a foreign key from R-schema.key-attributes<sub>B</sub> to B-schema.key-attributes<sub>B</sub>

#### 5. Convert all many-to-one associations to table schema.

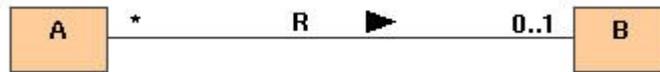


Figure 77 Many-to-one associations

if A-schema = ( key-attributes<sub>A</sub>, other-attributes<sub>A</sub> )  
 and B-schema = ( key-attributes<sub>B</sub>, other-attributes<sub>B</sub> )

then R-schema = ( key-attributes<sub>A</sub>, key-attributes<sub>B</sub>, other-attributes<sub>R</sub> )  
 with a foreign key from R-schema.key-attributes<sub>A</sub> to A-schema.key-attributes<sub>A</sub>  
 and a foreign key from R-schema.key-attributes<sub>B</sub> to B-schema.key-attributes<sub>B</sub>

*Important:* Handle one-to-many associations by reversing the roles of A and B. Also, composition relationships may permit optimizations.

6. Convert all one-to-one relationships to table schema.



Figure 78 One-to-one associations

if A-schema = ( key-attributes<sub>A</sub>, other-attributes<sub>A</sub> )  
 and B-schema = ( key-attributes<sub>B</sub>, other-attributes<sub>B</sub> )

then R-schema = ( key-attributes<sub>A</sub>, key-attributes<sub>B</sub>, other-attributes<sub>R</sub> )  
 with a foreign key from R-schema.key-attributes<sub>A</sub> to A-schema.key-attributes<sub>A</sub>  
 and a foreign key from R-schema.key-attributes<sub>B</sub> to B-schema.key-attributes<sub>B</sub>  
 and separate unique constraints on R-schema.key-attributes<sub>A</sub> & R-schema.key-attributes<sub>B</sub>

## 7. Convert all multi-valued attributes to table schema.

---

A
-keyAttr1_OID
-keyAttr2_OID
-multiAttr_1
-multiAttr_n
-singleAttr_1
-singleAttr_k

Figure 79 Multi-valued attributes

---

if  $\text{key-attributes}_A = \{ \text{key-attr}_1, \dots, \text{key-attr}_j \}$

then  $A\text{-schema} = (\text{key-attributes}_A, \text{single-attributes}_A)$

and  $M_1\text{-schema} = (\text{key-attributes}_A, \text{multi-attr}_1)$

with a foreign key from  $M_1\text{-schema.key-attributes}_A$  to  $A\text{-schema.key-attributes}_A$

and  $M_2\text{-schema} = (\text{key-attributes}_A, \text{multi-attr}_2)$

with a foreign key from  $M_2\text{-schema.key-attributes}_A$  to  $A\text{-schema.key-attributes}_A$

...

and  $M_n\text{-schema} = (\text{key-attributes}_A, \text{multi-attr}_n)$

with a foreign key from  $M_n\text{-schema.key-attributes}_A$  to  $A\text{-schema.key-attributes}_A$

## UML to Table Schema Conversion - Optimizing Rules

8. Use existence dependencies to eliminate unneeded many-to-one table schema.



Figure 80 Existence dependencies on many-to-one associations

---

if A-schema = ( key-attributes<sub>A</sub>, other-attributes<sub>A</sub> )

and B-schema = ( key-attributes<sub>B</sub>, other-attributes<sub>B</sub> )

and R-schema = ( key-attributes<sub>A</sub>, key-attributes<sub>B</sub>, other-attributes<sub>R</sub> )

with foreign keys as defined earlier

then drop R-schema

and alter A-schema = ( key-attributes<sub>A</sub>, other-attributes<sub>A</sub>, key-attributes<sub>B</sub>,  
other-attributes<sub>R</sub> )

with a foreign key from A-schema.key-attributes<sub>B</sub> to B-schema.key-attributes<sub>B</sub>

9. Use existence dependencies to eliminate unneeded one-to-one table schema.



Figure 81 Existence dependencies in one-to-one associations

---

if A-schema = ( key-attributes<sub>A</sub>, other-attributes<sub>A</sub> )  
 and B-schema = ( key-attributes<sub>B</sub>, other-attributes<sub>B</sub> )  
 and R-schema = ( key-attributes<sub>A</sub>, key-attributes<sub>B</sub>, other-attributes<sub>R</sub> )  
 with foreign keys and unique constraints as defined earlier

then drop R-schema  
 and alter A-schema = ( key-attributes<sub>A</sub>, other-attributes<sub>A</sub>, key-attributes<sub>B</sub>,  
 other-attributes<sub>R</sub> )  
 with a foreign key from A-schema.key-attributes<sub>B</sub> to B-schema.key-attributes<sub>B</sub>  
 and a unique constraint on A-schema.key-attributes<sub>B</sub>

**10. Eliminate many-to-one and one-to-one table schema at the "risk" of using NULLs.**



Figure 82 Many-to-one reduction

---

if A-schema = ( key-attributes<sub>A</sub>, other-attributes<sub>A</sub> )  
 and B-schema = ( key-attributes<sub>B</sub>, other-attributes<sub>B</sub> ) and R-schema exists  
 then drop R-schema  
 and alter A-schema = ( key-attributes<sub>A</sub>, other-attributes<sub>A</sub>, key-attributes<sub>B</sub>,  
 other-attributes<sub>R</sub> )  
 with a foreign key from A-schema.key-attributes<sub>B</sub> to B-schema.key-attributes<sub>B</sub>

and (for one-to-one relationships only) a unique constraint on A-schema.key-attributes<sub>B</sub>

*Important:* The A-schema.key-attributes<sub>B</sub> column may contain NULLs, which may cause problems or unexpected results when used in many common database/query operations