UNIVERSITAT POLITÈCNICA DE VALÈNCIA
DEPARTAMENT DE SISTEMES INFORMÀTICS I COMPUTACIÓ

# Bridging the Gap between Distance and Generalisation: Symbolic Learning in Metric Spaces

PhD dissertation by:
Vicent Estruch

Advisors:
José Hernández Orallo
Mª José Ramírez Quintana

València, 10 April 2008

# Abstract

Machine Learning is a discipline devoted to design algorithms that can learn from experience. Most of these algorithms are fundamental in many challenging applications such as data mining, natural language processing, artificial vision, etc.

Despite the applicability and high performance shown by these algorithms, we find that most of them are custom-designed for a particular context, which is equivalent to say that new algorithms must be invented or existing algorithms must be redesigned every time the learning context changes. For this reason, there is a need for developing general-purpose methods which can be run or be systematically adapted to several learning scenarios.

One of these scenarios concerns data representation. Generally speaking, data or examples can be described by tuples of numerical/categorical values (propositional data) or using other more expressive but complex representations which involve sets, sequences, graphs, etc. (structured data). Thus, a major goal consists of designing learners that can cope with different data representations.

In this line, most of the proposals existing in the literature can be classified into two groups, the so-called distance-based group (or more generally, similarity-based) and the symbolic-based group, each one of which has its own pros and cons. Distance-based algorithms can deal with any data representation by just changing the distance while the learning algorithm remains invariable. However, the model learnt, if any, is hardly comprehensible by the user. On the contrary, symbolic learners return comprehensible models at the expense of restricting the variety of data types they can handle. This is so because symbolic learners rely on the idea of generalisation (the way to go from data to models) which is implemented differently depending on the data representation.

An integration of both approaches would inherit their advantages. That is, we would obtain general methods which can easily be adapted to obtain symbolic models from any data representation. However, the concepts of distance and generalisation clash in many different ways, especially when data and pattern languages are complex (e.g. structured data). In fact, if the notion of generalisation is stated separately or independently from the distance, the patterns or models obtained via generalisation can be inconsistent with the distance employed.

This work establishes a framework where these two paradigms in machine learning can be integrated in a consistent way. We introduce the concept of distance-based generalisation, which can be intuitively seen as any pattern which connects all the generalised examples in such a way that all of them are reachable inside the generalisation by making straight paths through the metric space. As a result, the generalisation of these elements is coherent with the distances between them, or in other words, the generalisation obtained can be viewed as a comprehensible explanation of the distances between the elements. This makes the hypothesis space and metric space compatible. In order to organise the search and to characterise the solutions in this new integrated space, we also introduce a definition of minimal distance-based generalisation to control over- and under-fitting, which can

be seen as the first formulation of the MDL/MML principle in terms of a distance function.

We instantiate and develop the framework for the most common data types (both structured and non-structured data) and distances, where we show that some generalisation operators and pattern languages work well with only certain distances. Consistent combinations can be found in this document for numerical data, nominal data, sets, lists, tuples, graphs, atoms and clauses. In this way, we cover a wide range of different possibilities for data representation which are used in machine learning, since each representation language suits certain learning purposes in a better way.

As a result, general machine learning methods that integrate the best features from distance-based and generalisation-based methods can be defined. We illustrate this for the unsupervised and supervised case, showing that these methods can be adapted to any specific problem by appropriately choosing the distance, the pattern language and the generalisation operator. Nonetheless, we must remark that the algorithms can return different solutions if we change the distance and the pattern language, as we illustrate for the unsupervised case. With regard to the supervised case, we introduce a distance-based decision tree learner which (jointly with the generalisation operator) produces global symbolic models instead of the local models we would obtain if other distance-based learners (e.g. $k$-NN) were used.

Summing up, this is an extensive work that introduces a new general framework as well as numerous theoretical results which allow us to identify pairs of distances and generalisation operators that work well together. Compatible distances and operators can be used to convert distance-based learning algorithms into symbolic learners that return patterns consistent with the distance, a crucial property that does not necessarily hold when classical hybridisation proposals are employed. In other words, this work unveils the relationship between two key concepts in inductive inference, distance and generalisation, leading to the practical benefit of developing symbolic learners that are more independent of the data representation.

**Key words:** distance-based learning, symbolic learning, generalisation operators, learning from structured data, conceptual clustering, decision tree learning.

# Acknowledgements

This thesis arises after several years of research since I joined the ELP (Extensions of Logic Programming) group. I have been really fortunate to meet and work with great people whose contributions and encouragement in the making of this thesis deserve special mention.

First of all, I would like to thank my advisors, José Hernández and María José Ramírez, for their dedication, for teaching me about machine learning and how to do research, for their supervision from the very early stages of this thesis and for their many ideas and suggestions of inestimable value which I have benefited from and which are at the core of this document.

Also, I would like to thank Cèsar Ferri for his continuous attention and his helpful collaboration throughout the development of this thesis. Of course, the other members of the data mining research team have also contributed to this research. Therefore, thanks go to Toni Bella, Ana Funes, Arístides Dasso and Ricardo Blanco.

Individual goals owe their achievement to an excellent team. Thus, thanks to ELP group and its leader María Alpuente for their professional support and for providing me with such a pleasant and stimulating working atmosphere. Likewise, I must express my gratitude to all my colleagues in the E.P.S.G.: Jordi Bataller, Vicent Vidal, Paco Torres, José Antonio Lozano, Fermín Cerezo and Carlos García. Their great work has made my work much easier.

Friendship has also been present during these years. My heartfelt appreciation goes to my former lab mates Alicia Villanueva, Demis Ballis and very specially to Josep Silva, a charming person who is always ready to help you with a smile. I love the time we shared, the fun we had... thanks for everything.

I cannot forget the advice and suggestions I received from other companions of the department. I thank Alfons Juan, Salvador España, Carlos D. Martínez and Toni Cano for their time and kind attention.

I had the chance to visit some research centres in several countries during this time. All these visits left me unforgettable memories. I am grateful to professor Peter Flach for allowing me to be involved in the research group that he leads. His hospitality and scientific quality made my stay in the University of Bristol very productive. In fact, my appreciation to professor Peter Flach dates back to before my stay in Bristol, due to the motivating suggestions he made in connection with this work during his visit to the *Department de Sistemes Informàtics i Computació* of this university. I am also grateful to Thomas Gärtner for having introduced me into the research group headed by professor Stefan Wroble at Fraunhofer Institute (Sankt Augustin, Bonn). Thanks Thomas for your amiability, friendship and your professionalism by showing me novel research topics.

As for financial support, this research has been funded by several institutions. I thank the *Department de Sistemes Informàtics i Computació*, *Universitat Politècnica de València*, *Generalitat*

*Valenciana* and *Ministerio de Educación y Ciencia*.

Last but not least, I would like to thank my Greek colleague whose personal testimony left a deep impression on me.

Finally, my apologies to anyone who was involved in this thesis and may not have been mentioned here. Of course, I offer my sincere gratitude for all your help as well.

Vicent Estruch Gregori
València, April 2008.

# Contents

# List of Figures

# List of Tables

A Aurora.
Als meus pares i germans.
A l'Esperit del Renasa.

Gràcies a vosaltres sé, que un altra manera de viure és possible.

# Chapter 1

# Introduction

## 1.1 Motivation

Data analysis is a common need in multiple disciplines. The understanding of historical information is fundamental to make accurate predictions or to explain past events satisfactorily. However, we live at a time where the amount of recorded information far overcomes human capacity to explore it in-depth. The size and the complexity of the repositories in companies, governments, research centres and other institutions call for computer-aided tools that help specialists to find out patterns that are hidden in data.

The design of systems which automatically extract patterns from data concerns the field of Machine Learning [113]. This is a very active research area whose results are at the core of computational systems for real-life problems such as medical diagnosis, fraud detection, stock market analysis, speech recognition, search engines, etc.

The inductive learning techniques embedded in the aforementioned applications or the ones required in other scenarios might differ from each other. This is because the problem of learning from data can be formulated and/or presented in many different ways. In this sense, the main aspects to be taken into account in order to develop a learning method are: how data is described, what type of learning task is to be carried out, how the patterns or the models learnt from data are represented and in which contexts these models are going to be used [113]. For instance, the data to be analysed can be stored in a database or can be given in another format (e.g. a collection of text documents); the learning task might be how to classify data (classification), or discover significant groups in data (clustering), etc.; the patterns we find in data (e.g. the definitions of the different classes or groups) can be represented by means of diagrams, rules, mathematical functions, etc.; finally, the context always has the last word on the success and quality of the extracted patterns in terms of error, cost, utility and very especially, the comprehensibility of the extracted knowledge and its integration with previous knowledge.

The combination of all of these aspects raises a wide variety of possible learning scenarios and, in principle, a wide variety of learning techniques that must be developed to address them, since a method that is adequate for a specific family of problems might not be for others. This fact can easily be appreciated in learning problems where the data representation changes. Most learners are designed to deal with a propositional data representation (a data description consisting of fixed-

length tuples with numerical or categorical attributes) which is easily represented as a table. These systems are called propositional learners and are typical in data mining applications. However, data involved in more challenging problems, such as bioinformatics [10] or web mining [89] have no natural descriptions as a single tuple [63]. This kind of data needs a more expressive representation language. In these cases, data is usually represented by means of structured data types such as sets, lists, graphs, etc. Given that propositional learners cannot deal with structured data, new methods (e.g. relational learning [32]) have been specially designed to extract patterns from these complex descriptions.

However, patterns cannot be obtained disregarding the context of application. The comprehensibility of the models is a key factor which makes a learning method not suitable for some application domains. In many cases, not only are good predictions needed but an understanding about why these predictions are made is also needed. It happens that many learning methods return a model which is not directly interpretable by humans (e.g. artificial neural networks [169]), and consequently, it is difficult to contrast and integrate this model with previous knowledge.

As a result of this wide variety of data and pattern representations and context requirements, machine learning has a high disaggregation of methods and techniques. As a matter of fact, thousands of techniques which aim at very specific goals have been developed. However, from a conceptual point of view, designing specific methods every time the data representation, learning task, pattern language or context changes cannot be considered a convenient practice, since in the long term, this attitude tends to provide little understanding about the phenomenon we are studying and the real effectiveness of each interaction. In this sense, developing generic methods or unifying specific existing methods will always help us to detect the relevant traits, giving us a better comprehension of the problem as well as providing a more solid basis for future research. Thus, one of the challenges in machine learning is to design methods that are flexible enough to be directly applied to different scenarios, or at least, modular methods that can systematically be adapted when the learning context is modified [110, 171]. For instance, in the literature there are methods that are capable of carrying out different types of learning [16], methods that integrate different learning approaches [73, 136, 96, 85] or proposals giving a general theory of learning metrics [54].

Following this line of integration, we must highlight the effort that the machine learning community has made to develop methods that are capable of handling different data representations, particularly, learners that work with both propositional and structured domains. However, when we analyse this group of methods, several limitations can be detected.

On the one hand, there are learners (the so-called similarity-based methods [83]) that are based on the idea that similar (close) objects will behave similarly. At a general level, the notable advantage in most of these learners is that the learning algorithm remains invariable when data representation changes, since it is enough to adapt the similarity or distance function employed by the algorithm to the new data representation. This explains the high flexibility of the method since these learners can handle any data representation as long as a similarity function has previously been defined. However, the downside is that the model learnt, if any, is too cryptical for many applications, specially for problems with complex data.

On the other hand, there are other learners (the so-called symbolic-based methods) that can return comprehensible models in a specific pattern language. However, they are more restrictive about the data representations that each method can manage. Generally speaking, these methods are based on the idea of generalisation expressed in terms of symbolic patterns (e.g. expressions built from constant and variable symbols) that inform about the common traits of a group of elements. Given that the concept of generalisation states a relationship between examples (or data)

and patterns, we find that the pattern language is defined according to the data representation, and the idea of generalisation is implemented according to both pattern and data representation. Hence, this approach results in learners whose design is conditioned by knowledge representation. In other words, symbolic algorithms work well for some representations but certainly not for others.

As a result of the above considerations, it seems that an important contribution would be to design learning algorithms that combine the best of similarity-based and symbolic-based learners. That is to say, the possibility of having learners that can systematically be adapted to every data representation at the same time that the learnt model can be expressed symbolically. In this way, not only the similarity function but also the data and pattern representations and the generalisation notion that connects them should correspond to "parameters" or inputs to the learner. Note that a learner like this could be adapted to one context or others by properly setting these "parameters".

However, the combination of these two paradigms must be done carefully. This means that there should exist a coherent connection between the notion of similarity and the notion of generalisation, otherwise inconsistencies may arise in the learnt models. The intuition behind this is that if similarity and generalisation are set independently (with no connection), it is likely that the regularities in data found via similarity do not correspond or are even contradictory with the meaning of the patterns extracted via generalisation.

Besides the need for relating the concepts of similarity and generalisation, underfitting and its opposite, overfitting, are also two important aspects to be considered because they negatively affect the quality of the patterns obtained via generalisation. That is, we are not only interested in patterns that are consistent with the distance employed but also in learning patterns that are neither excessively general (underfitting) nor specific (overfitting). In the literature, there are concepts and techniques that deal with this problem, but they are particularly defined for the knowledge representation adopted (e.g. the concept of *least general generalisation* introduced for first-order logic representations [126], pruning techniques to avoid overfitting in decision trees [37], etc.). The design of general algorithms that are capable of dealing with arbitrarily complex data and pattern languages requires general and flexible criteria to optimally assess the model and to avoid the above-mentioned problems. Additionally, we also need to ensure that the notion of overfitting/underfitting stated by these generic criteria is consistent with the notion of overfitting/underfitting that is implicitly managed by the similarity-based algorithm, otherwise we could obtain contradictory results again. Therefore, as happened with the notion of generalisation, these generic criteria must somehow be related with the similarity function as well.

These problems become more relevant when the similarity function is a distance and the data representation language is seen as a metric space. On the one hand, distances are profusely used as similarity functions due to the fact that their solid mathematical foundations offer both interesting conceptual and practical advantages w.r.t. other more informal definitions of similarity [82, 164]. When distances are employed, we refer to distance-based methods instead of similarity-based methods. On the other hand, a metric space provides a geometric and intuitive interpretation of the notion of similarity which can be exploited to compare this space with the pattern space in the search for consistent integration.

.

## 1.2 Objectives

The main goal of this thesis is to establish a theoretical framework that allows us to integrate the most prominent advantages of distance-based and symbolic-based learners. More specifically, we look for a framework that gives us some understanding about how to design systems which can, on the one hand, deal with any data representation (as distance-based methods do), and on the other hand, extract meaningful symbolic models from data (as symbolic-based learners do).

We organise the study of this integration process into two main stages: first, we detect and analyse which aspects must be considered in order to systematically transform the output of distance-based methods into coherent symbolic patterns. Second, we see how the ideas derived from this analysis can be applied for pattern extraction when concrete combinations of distances and data representations are given.

All of this is detailed in the following partial objectives:

- Study the connection between distances and generalisation: given that the output provided by a distance-based method will be expressed by means of symbolic patterns, a crucial aspect that must be taken into account is to ensure that there is no gap or inconsistency between the output of the distance-based algorithm and the symbolic description. Inconsistencies can come up when the notion of distance and generalisation are set independently. Therefore, this part of the research aims to investigate these inconsistencies in the context of the relation between distance and generalisation in order to avoid them.

- Provide a definition of generalisation based on distances: once the relation between distance and generalisation is clear, the next step is to state a definition that allows us to determine (for a given set of examples, a distance and a pattern) whether or not the pattern is a generalisation of the examples that is consistent with the distance. Given that this definition comprises three components, which are data, patterns and distance, we need the definition to be valid for any data representation, pattern language and distance.

- Provide a definition of minimal generalisation based on distances: when speaking of generalisation, another important issue to be treated is the notion of *least general generalisation*, which allows us to control how general the pattern that we have learnt is. Traditionally, the concept of *least general generalisation* is associated to pattern representations and not to distances, which in our case means that we have no guarantee that least general patterns are really "least general" (at a smaller scope) w.r.t. the underlying distance. This implies that we need a definition of minimal generalisations that is consistent with the underlying distance and that can be applied to any pattern representation.

- Instantiate the framework for specific combinations of distances and data representations: once the framework has been stated, we aim at instantiating it for specific datatypes (nominal and numerical data, sets, lists, etc.) and distances. That is, we use the framework to find out which patterns are consistent with the distance employed, and which of these are minimal.

- Use the framework to transform distance-based learners into symbolic learners or to design new learning methods: a representative and illustrative demonstration of this must include algorithms for the unsupervised and supervised cases, preferably adapting techniques for different families of machine learning algorithms.

Note that the achievement of these objectives gives us the possibility of developing symbolic learners from any data representation in a systematic way as long as this representation is endowed with a distance function. To this end, it is sufficient to define distance-based generalisation operators for the representation at hand and plug them into a proper distance-based learner to obtain consistent symbolic patterns.

However, although important, this would not be the only practical implication. Another appealing consequence relies on identifying if the generalisation mechanisms at the core of the best-known symbolic learners are compatible with the most usual distances defined for the same data representation that these learners are designed for. In the negative cases, we must then propose alternative generalisation mechanisms (or even, alternative distances from which the original mechanisms perform coherent generalisations). By doing so, a nice feedback for symbolic and distance-based learning research is provided, since these new operators could be used to redesign existing symbolic learners or even to define others, and new distances would make the applicability of distance-based methods broader.

## 1.3 Organisation of this Dissertation

This dissertation is organised in four parts.

- The first part (state of the art): we present the state of the art of those machine learning topics that this dissertation is based on. Basically, since the major aim of this work is to analyse the relation between symbolic and distance-based learning, we review the most relevant contributions in the literature that refer to these two approaches. This part comprises Chapters 2 and 3. In Chapter 2, we focus on propositional learning. That is, we outline algorithms and techniques concerning symbolic and distance-based learning when data is described by means of a tuple of constants. Of all the methods presented, we pay special attention to decision tree learning and conceptual clustering. The reason for describing decision tree learners is not only because of their popularity and the importance that they have within symbolic learning but also because the last part of this thesis upgrades decision trees via distances. With regard to conceptual clustering, one of the contributions of this work is related to pattern extraction from unsupervised (unlabelled) data, which is precisely the major goal in conceptual clustering.

  In Chapter 3, the main problems and methods related to learning from structured data are reviewed. Again, we pay special attention to symbolic methods, which in this case, are mainly based on first-order logic. We conclude this chapter with a discussion about the difficulties that first-order logic methods show when handling some well-known structured datatyes (e.g. lists, sets, etc.). We also suggest the possibility to overcome this by extending some key concepts from inductive logic learning (the concepts of generalisation and minimality) to other data types as well as extending these concepts in order to take the concept of distances into account.

- The second part (distance based generalisation operators): this part presents a new framework which lets us put together distance-based and symbolic learning for both propositional and structured data. The keystone of this framework is the concept of distance-based generalisation operators which formalises the task of extracting patterns from data when it is embedded in a metric space (see Chapter 4).

Minimality is an important concept that concerns generalisations. This tells us if the generalisation overfits or underfits the data we are given. In Chapter 5, we present a definition of minimality for data which is embedded in metric spaces. A definition of minimality that takes the metric of the representation space into account is necessary because, in this way, we can ensure that minimal generalisations are also minimal when they are viewed or analysed from the metric space.

- The third part (distance-based generalisation operators for several datatypes and distances): this part (from chapters 6 to 9) contains definitions of distance-based generalisation operators for the most common datatypes (nominal, numerical, sets, sequences, etc.) and distances, as well as a detailed study concerning their minimality. Note that each data representation possesses its own particularities which must be considered in order to instantiate the framework and to endow each representation with a proper generalisation mechanism. For each data representation, we also investigate the advantages/disadvantages offered by pattern languages endowed with different expressiveness in order to define distance-based generalisation operators.

  Therefore, this part consists in a detailed study of a wide variety of scenarios that leads to concrete practical and theoretical results that are useful in transforming distance-based methods into symbolic learners that are adaptable to most of the data representations used in machine learning.

- Fourth part (applicability of our framework to unsupervised and supervised contexts): Chapter 10 contains a simple example about how our framework could be used for pattern extraction in unsupervised contexts. To do this, we plug distance-based generalisations operators into a hierarchical clustering algorithm. Several representations of data and distance are studied. Chapter 11 shows the possibilities that our framework offers for the supervised case. Here, we mainly focus on developing a supervised learning algorithm that is suitable to use in combination with distance-based generalisation operators. This algorithm is an upgrade of decision-tree learning via distances.

Finally, the last chapter concludes with a summary of the contributions and points out new directions for further research.

# Part I

# State of the Art

# Chapter 2

# Learning from Propositional Data

This chapter and the next one review the concepts and techniques in Machine Learning that will be needed for the original part of the work which starts in Chapter 4. Among the concepts that we will survey here, we focus on two important groups of techniques, the so-called distance-based and symbolic-based methods, restricted to propositional data representations, i.e. data which is expressed as tuples of scalars values (either nominal or numerical).

## 2.1 Machine Learning, an Overview

As said in the introduction, Machine Learning or Inductive Learning [113] is a scientific discipline based on developing computational methods which can learn and improve from experience, where the term "experience" is usually referred as *(training) data* and the elements in this set or collection of data are known as *examples*. An example corresponds to a real-world object together with, optionally, a label. The label encodes a property, a measurement of this object or a class/group membership. The objects are described by means of a *representation language* with the *representation space* being the set of all the descriptions permitted by the language. In a formal notation, we have that $x, x_1, x_2, \ldots$ denote object descriptions, $X$ is the representation space, letters $y, y_1, y_2, \ldots$ denote labels and $Y$ is the set of labels. Let us illustrate this by means of an example. A risk-mortgage scorer is a system used by the banks to predict the risk about a mortgage concession. The system learns how to predict the risk of future mortgage applications from a table of customers (training set) who has been previously granted with a mortgage. Thus, each customer (examples) is represented by a pair $(x, y)$ where $x$ could be a tuple of nominal/numerical attributes like $(age, profession, incomes, houseprice)$ with $X = Domain(Age) \times Domain(profession) \times \ldots \times Domain(hose\ price)$ and $y \in Y\{$no risk, risk$\}$, which is determined by an expert. The system informs the clerk whether a mortgage concession is risky or not depending on the values of these attributes in the customer. The process of learning from a propositional representation is known as *propositional learning*. Other problems might require more complex descriptions for data (e.g. examples represented as sets, sequences, graphs, etc.). In this case we speak of *structured data learning*.

So far, we are considering that all the examples from the training set are labelled, and i.e. each example is of the form $(x, y)$. This setting is called *supervised learning*. However, this is not

always the case since, in some problems, labelling data might have a very high cost, or labels are simply unknown. The fields known as *semi-supervised* and *transductive learning* [175] comprise all the techniques which deal with partially labelled data. Furthermore, there exists a very important learning scenario called *unsupervised learning* where all the examples are unlabelled. Here, the major challenging consists of precisely discovering hidden groups or unknown classes (clusters) in data. The groups discovered depend on the clustering algorithm employed (see [12, 169]).

Back to supervised contexts, the aim is to obtain a function $h : X \rightarrow Y$, also called *hypothesis* or *model*, which labels unseen new examples [113]. According to this schema, we differentiate between *classification* when the set $Y$ is without order (e.g. a set of nominal values) and *regression* when $Y$ is a continuous (or not) set of ordered values (e.g. an interval of real numbers). For instance, in the mortgage concession problem, the system *classifies* an operation as being risky or not. If we would like to grade the level of risk (e.g. by means of an index ranging from 0 to 10), then we would be talking of a *regression* problem.

The set of all the hypotheses we can formulate for a given problem is called *hypothesis space* (denoted by $H$). Naturally, among all the possible hypothesis one can draw from $H$, we want $h$ to label correctly as many future examples as possible. Observe that there would be no point if our system misclassifies most of the new mortgage applications. This property is known as *accuracy* or *error rate*. But besides accuracy, other evaluation measure or properties can also be important in a model.

For instance, comprehensibility is another interesting aspect to be considered in a hypothesis. There exist problems where the user need to understand the predictions made by the hypothesis in order to give explanations or make future decisions. For instance, a clerk might need to give some explanations to those customers who have not been granted with a mortgage. To talk of comprehensibility in a proper way, we need to previously introduce the concept of hypothesis representation. There exist two possibilities for this.

On the one hand, we have the so-called *lazy learning methods* where the knowledge learnt from data is not explicitly represented. In this case we have no comprehensibility about how predictions are made. However, these methods are widely-used because they can easily be adapted to different problem domains. An important (sub)family of lazy methods are the so-called *similarity-based* or *distance-based methods*. These rely on the idea that similar (or near) examples must have similar predictions. Some popular similarity-based methods are $k$-NN ($k$ Nearest Neighbours [22]), Fisher's discriminant and LVQ (Learning Vector Quantization).

The second possibility is to introduce a special language from which the hypothesis can be defined or represented. This group of techniques is called *model-based* or *eager* learners. However, this does not ensure a priori that the hypothesis can be comprehensible for the user. In fact, there are methods where the hypothesis learnt is little comprehensible or even behaves like a black-box. Examples of this are Support Vector Machines (SVM) [151], some Bayesian methods [28, 111], ensemble learning [26], Neural Networks (NN) [106, 146] or some genetic algorithms [68, 76]. To overcome this limitation, there exist a group of model-based techniques which aims to express hypotheses in a more natural language for users. These methods are called *symbolic-based learners*. By a more "natural language" we mean *if-then-else rules*, *regular expressions* or any other kind of symbolic patterns. For instance, our mortgage-risk scorer could learn a collection of rules like:

- If the salary of the customer is greater than a VALUE1 and the house price is less than VALUE2 then NO RISK.
- If the age of the customer is greater than VALUE3 then RISK.

Rules are usually employed for hypothesis description because they are close to human language. The problem of extracting rules from data is called *rule learning*. Some well-known rule learners are the algorithms AQ [167], CN2 [21], ID3 and C4.5 [131] and CART [18].

Besides the learning settings mentioned above, we can find other equally important paradigms such as *reinforcement learning* (when the learning system is awarded or punished depending on the decisions it makes over time [99]), *active learning* [163] (a concrete case of semi-supervised learning where the algorithm asks for informative labelled examples) or *metalearning* (where the purpose is to learn which learning algorithm performs best for the problem at hand).

After this short overview and given that this thesis focuses on combing the best of distance-based and symbolic-based learning, the rest of this chapter is devoted to describe with more detail these two family of methods for the specific case of propositional data.

## 2.2 Learning with Distances

Distance-based learning (DBL) is quite related to the field called case-based reasoning (CBR) [83]. The latter is inspired on the psychological principle known as "reasoning by analogy" which assumes that similar problems must have similar solutions.

In order to transform this principle into learning algorithms, it becomes obligatory to quantify how similar two objects are. A way to do this consists of using functions (called similarity function) which map a pair of objects into a real value. The higher (lower) this value is, the more similar the objects are. If a function $d$ satisfies the following axioms:

1. nonnegativity: $d(x_i, x_j) \geq 0$, for every $x_i, x_j \in X$ (representation space).
2. reflexivity: $d(x_i, x_j) = 0 \Leftrightarrow x_i = x_j$.
3. symmetry: $d(x_i, x_j) = d(x_j, x_i)$.
4. triangular inequality: $d(x_i, x_j) \leq d(x_i, x_k) + d(x_k, x_j)$, for every $x_i, x_j, x_k \in X$.

then we say that $d$ is a distance function (or a metric) and the pair $(X, d)$ is a metric space. As the reader can foresee, DBL gathers all those techniques which formalise the notion of similarity by means of distance functions. Thus, the adjective *similar* in CBR is substituted by the adjective *near* when speaking of DBL. This results in that near objects must belong to the same class (for classification), or must have similar values (for regression), or must belong to the same group (for clustering).

In principle, algorithms employ generic distances. Regarding a representation based on tuples of constants, the most common distance functions are:

- $L_p$ or Minkowski distance: $d(x_1, x_2) = (\sum_{i=1}^{n} |x_{1i} - x_{2i}|^p)^{1/p}$ with the Euclidean distance ($p = 2$) and the Manhattan distance ($p = 1$) being particular cases.
- Chebychev's distance: $d(x_1, x_2) = \max_{i=1,\ldots,n} |x_{1i} - x_{2i}|$
- Mahalanobis distance: this takes the correlation of the attributes into account. This is calculated as
$$d(x_1, x_2) = ((x_1 - x_2)^t S^{-1} (x_1 - x_2))^{1/2}$$

where $S$ is the covariance matrix. Note that Euclidean distance is a special case of this one when attributes are not correlated.

One of the most remarkable advantages in DBL is that a distance function can be viewed as a parameter of the learning algorithm. This means that an algorithm can be tuned for a specific problem by defining a proper distance for it. For instance, in the domain of OCR (*Optical Character Recognition*), tangent distance [30] becomes especially effective. Or equivalently, the algorithm can be used for any data representation if a distance function is defined over it. For instance, the distances we report above are used for propositional data, in case we have structured data, most learning methods can directly be used by simply defining a distance for the structured data type at hand (see Section 3.4.1 of the Chapter 3 for a description of distances for the most common data types). Obviously, this also implies that the distance can negatively affect on the performance. Frequently, a general-purpose distance is used, but some problems might require a previous analysis to determine the distance function to be employed. In fact, there exist some works in the literature which concern on learning the distance which best fits a problem [149, 152].

Next, we describe the widest-used distance-based methods for supervised and unsupervised cases.

### 2.2.1 Supervised Distance-based Algorithms

All the methods we are going to see are really intuitive. They consider that every example corresponds to a point in a metric space and yield their predictions by comparing the proximity of an unseen example w.r.t. some concrete labelled examples. In fact, these algorithms differ from each other in the way that these labelled examples are chosen. A description of these methods can be found in any introductory machine learning handbook (see [113, 30, 75]). Let us see a short description of some of the most common.

- $k$-Nearest Neighbours: probably, the best-know distance-based supervised algorithm. The classification rule $h$ is really simple. For an unseen new example $x$, we set $V(x) = \{(x_i, y_i)\}_{i=1}^{k}$ as the $k$ nearest examples to $x$. Then, we have that

$$h(x) = \underset{y \in Y}{\operatorname{argmax}} \sum_{\forall (x_i, y_i) \in V(x)} \delta(y, y_i)$$

  where $\delta(y, y_i) = 1$ if $y = y_i$ and 0 otherwise. That is, $x$ is labelled according to the majority class among its $k$ nearest examples. For this reason, the entire training set must be stored in memory which results in a serious disadvantage due to the space requirements. Additionally, the algorithm can easily be adapted for regression. To this end, it is enough to calculate the mean value of the $k$ nearest training examples. Obviously, as the value $k$ is increasing, the analysed area gets larger, the method tends to be more global and the average error decreases (up to a point), but it makes the process slower. Note that it does not matter how the data is represented, if a distance function exists then $k$-NN can be employed directly.

- Fisher discriminant: each class is represented by means of a centroid (a point which minimises the sum of the distance to the elements belonging to this class). A new example is tagged according to the label of its nearest centroid. Thus, this technique is eager and examples are removed from memory once the model (the set of centroids) has been obtained. Given that the representation space is made up of tuples of nominal/numerica data, a more expressive model

can be derived: centroids of adjacent classes are joint by their straight lines. The median of each one of these lines are the discriminant rules. Unfortunately, this model cannot be obtained when structured data is involved.

- Learning Vector Quantization: the learnt model is also a collection of prototypes where an unseen example is classified according to the proximity to the prototypes. Initially, the so-mentioned prototypes are chosen randomly, and their positions are updated until a threshold is reached. This process is performed iteratively. First, an example of the training set is chosen and the nearest prototype is determined. The position of the prototype changes depending on the labels of the example and the prototype. If both labels match, the prototype comes closer to the example. Otherwise it goes further. The process stops when the changing of positions of the prototypes is not greater than a threshold or there are not more training examples. Again, the way in which prototypes are updated makes that this method is complicated to be applied in structured domains.

### 2.2.2 Unsupervised Distance-based Learning

Next, we review some well-known clustering methods such as *hierarchical clustering* and *K-means*. Although not being based on distances, other important approaches will also briefly be described.

- Hierarchical clustering: the algorithm builds a tree of clusters (*dendrogram*) from a collection of data in such a way that each cluster in the tree also contains its descendant clusters. Two categories exist: (*i*) *top-down (divisive) methods* which start with a cluster containing all the training data and this is split into smaller clusters according to some quality criterion. Then, the smaller clusters are split again and so on; (*ii*) *bottom-up (agglomerative) methods* consider that each element in the training set is a cluster. Then, they are progressively merged according to some criterion.

  The way in which clusters are split/merged is guided by the principle that nearest elements should remain in the same cluster. In consequence, we need to know whether two clusters are near or not. Given that a distance function is defined only over a pair of elements we need to extend it in order to measure inter-cluster distance. This extension is called *linkage distance*[1] being the most common:

  - Single linkage [155]: the distance between two clusters is the minimum of all the pairwise distances between elements drawn from the two clusters. It has a tendency to produce elongated clusters [123].
  - Complete linkage [86]: as before but the distance is given by the furthest pair of elements. This criterion tends to produce more compact clusters than the single linkage does [9].
  - Average linkage: the distance between two clusters is the distance between their centroids.

  Regarding bottom-up methods, the pair of clusters which are nearest according to the linkage distance will be merged. If the linkage distance is the average linkage then the centroids must be recomputed. The process goes on until a previously fixed number of clusters is achieved or the complete dendrogram is built (to see in more detail how a linkage distance is used in top-down algorithms we refer the reader to [81]).

[1] Although called distance, it does not mean that this function satisfies the axioms of a distance.

Although hierarchical methods directly based on linkage distances are easy to be implemented, they tend to return convex-shaped clusters. However, data may lie in components of arbitrary shape. In this line, the hierarchical algorithms CURE and CHAMELEON were proposed (see [12] for further information).

- K-means: *partitioning methods* is another relevant family of clustering techniques. Here no linkage-metric is required and the clusters are gradually improved according to an optimisation function. Among them, the best-known is $k$-means [104]. Its mechanism is quite similar to Kohonen's self-organising maps [88]. Initially, a set of centroids are randomly chosen. The examples are assigned to their nearest centroid forming clusters. The centroids of the new clusters are computed. The process is repeated until the centroids remain invariable. Although this algorithm has succeeded in industrial and scientific applications, suffers from some inconveniences. Generally speaking, its performance strongly depends on the initial guess and the number of centroids we set. The comprehensibility is also limited since the algorithm returns the prototypes as representation of each cluster.

In the literature, there exist other important variants of clustering not exclusively based on distances. Thus, we have the so-called probabilistic-based clustering where the clusters are described by means of a probabilistic function instead of a distance to one prototype. Or conceptual clustering where the clusters are described by means of symbolic expressions. We will analyse conceptual clustering in more detail at the end of this chapter.

Although distance-based methods have successfully been used in many real-world problems (e.g. speech recognition), their main limitation can be, in some contexts, the low expressiveness of the model obtained, if any. Note that in most of the learners, the "model" is defined in terms of the distance to some examples, which makes this hardly comprehensible for the user. However, things are different when symbolic learners are employed.

## 2.3 Symbolic Learning

Many real-life scenarios require explanations on the way learners make their decisions. Remember the unsatisfied customer asking for his/her mortgage rejection. Or imagine that a team of researchers may need a suitable description of a cluster of molecules in terms of chemical properties, characteristic substructures and so on rather than a prototype. By symbolic learners, we refer the set of learning techniques which return a model which can be interpreted by the user. This property makes that symbolic models can be supervised, modified and adapted by the user and can also be integrated with previous knowledge or easily assembled into software applications. We review next the best known symbolic approaches for both supervised and unsupervised contexts.

### 2.3.1 Supervised Symbolic Learning: Decision Trees and Rules

Perhaps, decision tree and rule learning are by far the most popular families of techniques directly producing interpretable models. Both perspectives, decision trees and rules, are somehow related since a decision tree can be viewed as a collection of propositional rules organised into a tree structure. Roughly speaking, each inner node contains a test over an attribute where each branch descending from this node corresponds to a Boolean condition over a possible value of this attribute. The

leaves contain predictions. By predictions, we refer to labels (classification) or numerical values (regression). Decision trees for clustering have also been defined.

Each new example will traverse the tree from the root to a leaf guided by the Boolean conditions that it satisfies. Let us illustrate this by means of an example extracted from [75]. The decision tree depicted in Figure 2.1 informs us about when myopia can be treated with surgery. Observe that this decision tree is semantically equivalent to the rules below:

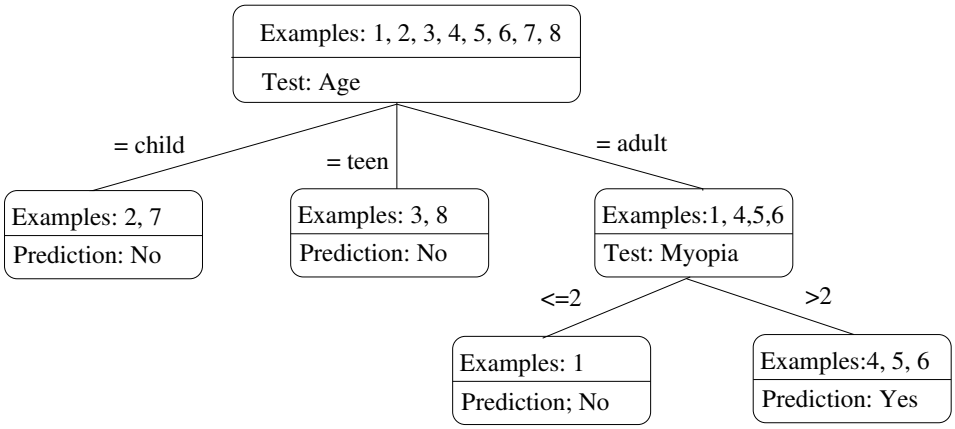


Figure 2.1: A decision tree for surgery-treated myopia.

| Index | Myopia | Age | Surgery |
|---|---|---|---|
| 1 | 1.5 | adult | No |
| 2 | 0.5 | child | No |
| 3 | 0.75 | teen | No |
| 4 | 3 | adult | Yes |
| 5 | 6 | adult | Yes |
| 6 | 6.25 | adult | Yes |
| 7 | 2 | child | No |
| 8 | 7 | teen | No |

Table 2.1: Examples for the myopia problem (the first column indexes the examples).

- IF Age=child THEN Surgery=No
- IF Age=teen THEN Surgery=No
- IF Age=adult and Myopia<=2 THEN Surgery=No
- IF Age=adult and Myopia >2 THEN Surgery=Yes

Once the decision tree has been learnt, it can be used for prediction. Suppose that we are given two new patients $Patient_1 = (adult, 8)$ and $Patient_2 = (teen, 3)$. According to the decision tree, $Patient_1$ would be surgery-treated but $Patient_2$ will not, because of his age.

| Algorithm | Splitting Selection | $f(p_j^1, \ldots, p_j^c)$ |
|---|---|---|
| ID3 | Information gain | $\sum_{i=1}^{c} p_j^i log_2(p_j^i)$ (Entropy) |
| C4.5 | Gain ration | Entropy |
| CART | Gini Index | $1 - \sum_{i=1}^{c} (p_j^i)^2$ |
| DKM | | $2(\prod_{i=1}^{c} p_j^i)^{1/2}$ |
| Others | Accuracy | $max(p_j^1, \ldots, p_j^c)$ |

Table 2.2: Impurity function for some splitting criteria.

The key question is how a decision tree can be learnt from examples. For instance, the decision tree above has been learnt from the examples in Table 2.1. The learning process is reduced to find out the proper tests and in which order these appear in the tree. Note that a test involves two components, an attribute and Boolean conditions over it. Due to the fact that a huge variety of conditions can be formulated over an attribute, each learning algorithm establishes some kind of restriction (representation bias). Thus, we have that $ID3$ [133], which works with nominal data only, fixes conditions of the form

$$x_i = a_1 \text{ or } x_i = a_2 \text{ or } \ldots \ x_i = a_n$$

where $\{a_i\}_{i=1}^{n}$ is the domain of the attribute $x_i$ (e.g. $Age = child$ or $Age = teen$ or $Age = adult$). However, the algorithm $CART$ [18] sets a binary split such as

$$x_i = a_j \text{ or } x_i \neq a_j$$

where $a_j$ is selected according to a quality criterion (e.g. $Age = adult$ or $Age \neq adult$). In fact, $CART$ handles numerical data as well by using a binary splitting for numbers as:

$$x_i \leq h \text{ or } x_i > h$$

For instance, $Myopia \leq 2$ or $Myopia > 2$. Furthermore, this split is employed by the algorithm $C4.5$ [131] (an extension of $ID3$ dealing with numerical data, missing values, etc.), although in this latter case the value $h$ is computed in a different way.

It still remains to be answered how the tests in the tree are chosen. Most of the algorithms follow a greedy strategy guided by a quality function, the so-called *splitting criterion*. Generally speaking, each possible test applied over each possible leaf in the tree is examined. The pair $(test, leaf)$ selected is the one which performs best according to the specified criterion. When the test is applied, new leaves are generated. The splitting criterion employed by most of the algorithms is a particular case of the expression

$$I(s) = \sum_{j=1}^{n} p_j f(p_j^1, p_j^2, \ldots, p_j^c) \tag{2.1}$$

where $s$ is the split, $n$ is the branching factor of the split, $p_j$ is the prior probability that an example drops in the $j$-th node generated by the split $s$, $p_j^k$ is the proportion of examples belonging to class $k$ in the $j$-th node and $f$ is a particular function named impurity function (see Table 2.2).

The selection of the split at each step of the learning process might be different depending on the splitting criterion we use. Thus, Gini prefers those splits that put the largest class into one pure node

and all the rest others into the other, whereas entropy-based criteria put their emphasis on balancing the size of the children nodes [30]. Despite some authors consider that accuracy prematurely stops the tree growing process, early work has recently put this into question at least for the case of decision trees based on higher-order logic [124] (see the subsection 3.3.3 of the Chapter 3). Other splitting criteria, which do not fit equation 2.1, are proposed in [50] and in [49]. The first one suggests a hypothesis search driven by a MDL-based heuristic and the second one investigates the possibilities of the AUC metric as splitting criterion.

Note that we have said nothing yet about when the inference process stops (*stopping criterion*). Normally, this happens when all the leaves in the tree only contain examples belonging to the same class. Unfortunately, this is not the best option in case the training set is not representative of the true hypothesis or there are examples with mistakes (noisy data). Under these circumstances the decision tree learning algorithm will tend to overfit the data. That is, this will learn irregularities which do not correspond to the true hypothesis. To avoid overfitting, most of the techniques introduced rely on *pruning* those branches which are considered too much specific. Depending on when tree pruning is done, we distinguish between *pre-pruning* (during the decision tree construction) and *post-pruning* (once the decision tree has been learnt). In general, post-pruning performs better (since the whole tree is available) than pre-pruning but it is obviously less efficient. The most relevant pruning techniques are deeply analysed and empirically evaluated in [37].

Apart from the so-mentioned decision tree algorithms (ID3, C4.5, CART and DKM [84]), there exist other outstanding approaches in the literature. For instance, Quinlan proposed C5.0 as an improvement of C4.5. In [122, 160], multivariant decision trees, that is, decision trees where more than one attribute can take part in the same split, are considered. We cannot overlook fuzzy decision trees where the splits are expressed in terms of fuzzy logic [173], or decision graphs [159] which is a remarkable conceptual extension of decision trees.

As the size of the decision tree increases, its readability becomes more and more difficult. In this way, *if-then-else rules* provide a better comprehensibility of the model than the tree structure itself because of its proximity to natural language. As seen, decision trees could trivially be transformed into a collection of rules by traversing the structure, but this might result in rules with a huge amount of redundant antecedents which brings the comprehensibility of the model down. For instance, using C4.5, it would not be surprising to get a rule such as

$$\text{IF } x_1 < 3 \text{ and } x_2 = a \text{ and } x_1 < 2 \text{ and } x_3 = b \text{ and } x_1 < 1 \text{ THEN } \mathit{Prediction}.$$

which could be simplified into

$$\text{IF } x_1 < 1 \text{ and } x_2 = a \text{ and } x_3 = b \text{ THEN } \mathit{Prediction}.$$

Given this problem, C4.5 has a module for rule derivation named C4.5Rules which translates a decision tree into an equivalent set of rules while taking care about redundant antecedents. Another aspect to be considered is the number of rules, since an unnecessary number of rules makes the model less understandable.

Observe that the 4 rules at the beginning of this section are semantically equivalent to the unique rule:

$$\text{IF } \mathit{Age} = \mathit{adult} \text{ and } \mathit{Myopia} < 2 \text{ THEN Yes ELSE No.}$$

The reason why a decision tree algorithm cannot come up with this one-rule model is because it learns several rules simultaneously. On the contrary, when focusing on a unique rule, it is possible

to directly look for the most discriminative ones. As for rule learning, a rule is generally built in a similar fashion as a decision tree. The first difference is that, initially, the rule is empty and tests are progressively added according to a performance measure over the training set. The process halts when a quality criterion (normally accuracy) is satisfied. The second difference from decision tree learning is that the examples covered by the new rule are removed from the training set. This method is called *sequential covering*, instead of the "divide-and-conquer" philosophy of decision trees. The rule induction usually fits a greedy depth-first search and, as any greed search, it is likely to reach a suboptimal solution instead of the optimal. The *beam search* is an attempt to minimise this risk. Here a list of $k$ candidates is maintained in each step instead of a unique candidate. The algorithm explores all the candidates and updates the list with the new most promising ones. The most popular propositional rule learners AQ [167] and CN2 [21] implement such a beam search.

### 2.3.2 Unsupervised Symbolic Learning

So far, we have focused on extracting comprehensible patterns for supervised problems. However, for unlabelled data, the clustering methods mentioned in Subsection 2.2.2 offer no symbolic interpretations of the clusters discovered. To overcome this, *Conceptual Clustering* and *Formal Concept Analysis* (FCA) techniques can be used. Both were introduced and mainly developed during the 80's and are distinguished from ordinary data clustering because they generate comprehensible descriptions.

Conceptual clustering relies on Michalski's definition stating that clusters in data are those which easily fit concepts that are used to describe clusters [108]. From a very general point of view, the most relevant point w.r.t. classical clustering is the presence of a hypothesis space (named *conceptual class* in Michalski's terminology) during the learning process. Clearly, this sounds like an attempt to translate symbolic learning techniques from a supervised to an unsupervised context. This is accomplished by intuitively extending the ideas of:

- Covering: as seen, in supervised learning a concept (hypothesis) covers a group of data depending on whether its predictions are correct or not. When labels are unknown, talking about predictions does not make sense and a more general interpretation becomes necessary. Conceptual clustering states that a concept is a pair made up of a property and a set of elements satisfying (being covered by) this property (e.g. the concept $evenNumbers$ can be expressed by $(divisibleByTwo, \{0, 2, 4, \ldots\})$). We will see later that this viewpoint is in common with FCA.

- Quality function (cohesion): learning processes are driven by a quality function in their search for the best solution. In supervised learning this function is of the form $f(h, D)$ measuring how well a hypothesis $h$ predicts a set of examples $C$. In unsupervised learning we have $f(C_1, \ldots, C_k)$ quantifying how adequate the partition $\{C_i\}_{i=1}^n$ of $C$ is; conceptual clustering combines both expressions in such a way that $f(C_1, Concept_1, C_2, Concept_2, \ldots)$ says how adequate a partition $\{C_i\}_{i=1}^k$ of $C$ induced by a group of concepts $\{Concept_i\}_{i=1}^k$ is.

Let us outline how these latter ideas are implemented in two of the most relevant systems: Cluster/2 and Cobweb. Cluster/2 was proposed by R.S. Michalski and R.E. Stepp in [108] and restricts itself to examples described by a fixed number of nominal attributes where the domain of an attribute $x_i$ (e.g. $Dom(x_i) = \{low, medium, high\}$) might be endowed with an order relation (e.g. $low < medium < high$). The hypothesis space is composed of *conjunctions of selectors*

(Conjunctive Conceptual Clustering) where a selector is just a relational statement over an attribute (e.g. $x_i = high$, $x_i <= medium$, etc.). The quality function is defined as

$$f(C_1, Concept_1, \ldots) = \sum_{i=1}^{k} Complexity(Concept_i) + Sparseness(Concept_i, C_i) \qquad (2.2)$$

where *complexity* counts the number of conjunctions in $Concept_i$ and *sparseness* the number of unseen examples covered by $Concept_i$, that is, it is a measure about how general the concept is. The former is, in part, one of the reasons why the system is restricted to nominal attributes since $sparsennes(x_i)$ cannot be computed when $Dom(x_i)$ is continuous or infinite. We will turn to this problem later.

The interesting thing in 2.2 is that the more complex the concept is, the less sparse. Therefore, the system aims to find out $k$ clusters whose descriptions are a trade-off between complexity and sparseness. To do this, $k$ randomly examples (*seeds*) $\{e_i\}_{i=1}^{k}$ are drawn. From these, the hypothesis space is explored in both directions. The generalisation operator *m-bound star* returns the $m_i \leq m$ best most general descriptions which cover each seed $e_i$ but not the others. Then the descriptions are conveniently specialised. From the $m_1 \times \ldots \times m_k$ possible concepts $(Concept_1, \ldots, Concept_k)$, the most promising one is selected according to the quality function. This process is repeated by taking $k$ new seeds $\{e'_i\}_{i=1}^{k}$ such that $e'_i$ belongs to $Concept_i$, until a number of iterations specified by the user is reached or the quality of the concepts cannot be improved more.

Unlike Cluster/2, COBWEB [51] introduces a probabilistic conjunctive language that supplies robustness for noisy data. Another advantage is that it returns a hierarchy of concepts rather than a unique level of concepts. The quality function (*Partition Score*), which is of a probabilistic nature as well, is based on the so-called *Category Utility* (CU) proposed by [67]. Therefore, the $CU$ of a cluster $C_k$ is:

$$CU(C_k) = P(C_k)(\sum_i \sum_j P(A_i = V_{ij}|C_k))^2 - \sum_i \sum_j P(A_i = V_{ij})^2)$$

where $P(C_k)$ is the proportion of elements in $C_k$ (a priori probability), $P(A_i = V_{ij})$ is the probability of $A_i = V_{ij}$ in the whole data set and $P(A_i = V_{ij}|C_k)$ its the conditional probability, that is, the proportion of elements satisfying $A_i = V_{ij}$ in the cluster $C_k$. The intuition behind $CU$ is that it measures how accurate the attributes can guess the elements in $C_k$. Also note that $CU$ balances the size of the cluster and the predictability of the attribute values. Finally, the quality function is the average of all the $CU(C_k)$. That is,

$$Partition\ Score = \frac{\sum_{i=1}^{k} CU(C_i)}{k}$$

The tree of concepts is inferred incrementally, that is, example by example. To do this, the algorithm places each new given example in an existing category (cluster) or creates a new singleton category. The problem is that this learning mode is very sensitive to the order the examples lie in. To reduce this risk, two new operators are incorporated: *merging* and *splitting*. The first one generates a new pair $Cluster/Concept$ from two existing $Cluster_1/Concept_1$ and $Cluster_2/Concept_2$ placed at the same level of the tree, in such a way none of them is removed but $Cluster_i/Concept_i$ $(i = 1, 2)$ are made children of $Cluster/Concept$. The second operator might sound a bit confusing since it has few to do with dividing a $Cluster/Concept$ but removing it from the tree. Each operation is

selected according to the quality criterion and the process stops when there are no training examples left.

As Cluster/2, COBWEB cannot explicitly deal with numerical attributes either, given that $CU$ is defined in terms of discrete probability functions. A possible solution is the use of probability density functions for numerical attributes, what is precisely done by the extended versions COBWEB/3 [107] and ECOWEB [143].

Other recent approaches are Cluster/3 [153], which extends Cluster/2 by including a method that selects those attributes which are more relevant w.r.t. a predefined criterion, and [112] which is an original proposal based on finding the maximum edge biclique problem.

As mentioned before, an alternative proposal to Conceptual Clustering is $FCA$ [61]. Roughly speaking, given a set of examples and a set of properties, $FCA$ finds a lattice of *formal concepts*, that is, pairs of the form $(\{properties\}, \{examples\})$ where the lattice is defined by a generality relation among concepts and the set $\{properties\}$ completely characterises the set $\{examples\}$. Unlike conceptual clustering, the goal is to obtain a compact representation of the data rather than the group discovery itself. The lattice also provides a first step for *frequent pattern mining* and *association rules discovery*.

To conclude with this chapter, notice that we have reviewed the most notable algorithms for supervised and unsupervised contexts where some of them generate a model while others not. Although all the algorithms mentioned in this chapter have successfully been applied in real-life problems, a significant restriction is that they work with nominal/numerical data only. However, the examples of a problem may have an inner structure. This rises the question whether propositional learners are powerful enough to deal with the structured objects. This issue is analysed in the next chapter.

# Chapter 3

# Learning from Structured Data

Learning from structured data has become one of the most active research areas within machine learning. This importance is given by the fact that a propositional representation is not expressive enough when examples have an internal structure. Essentially, structured examples (e.g. web sites, XML documents, molecules, etc.) are made up of distinguishable (sub)parts related to each other. This makes it hard to represent them with a single row in a table as propositional learners do. Therefore, new learning techniques must be developed to cope with this complexity. This is just what learning from structured data aims to.

## 3.1 Why Learning from Structured Data?

This section attempts to show the need of learning from structured data as well as to point out the difficulties this new scenario entails. For this purpose, we start our discussion by introducing two classic problems which will help us to illustrate what structured learning consists of:

**(Bunch of keys problem)** This widely-known problem was introduced by T. Dietterich in [25]. This problem is about a collection of bunches of keys and a locked door. Some of the bunches have at least one key which opens the door but others not. The goal is to obtain a classifier for the bunches of keys (not for the keys because the class label of each key its unobservable) recognising which bunches have a key opening the door and which do not. Each bunch contains a variable number of keys and from each key is known a tuple of properties (see Table 3.1): material (abloy, chubb, rubo, yale), number of prongs $(1, 2, 3, \ldots)$, length (short, medium or long) and width (narrow, normal or broad). Consequently, we must set some conditions over the keys in a bunch to determine whether this bunch contains a key opening the locked door or not. For instance, the rule we are looking for could be *"a bunch of keys opens the door iff it contains an abloy key of medium length"*. Note that this does not necessarily mean that a medium-length abloy key opens the door.

**(East-West challenging)** This problem was proposed in the mid 80's by R.S. Michalsky in his research about learning from structured data [158]. The training set contains ten trains where each train consists of a variable number of cars arranged in some order. Each car is, in turn, described by

| Index | | Opens |
|---|---|---|
| Bunch 1 | Key1=(Abloy, 3, Short, Normal), Key2=(Abloy, 4, Medium, Broad), Key3=(Chubb, 3, Long, Narrow) | Yes |
| Bunch 2 | Key1=(Abloy, 3, Medium, Broad), Key2=(Chubb, 2, Long, Normal), Key3=(Chubb, 4, Medium, Broad) | Yes |
| Bunch 3 | Key1=(Yale, 3, Long, Narrow), Key2=(Yale, 4, Long, Broad) | No |
| Bunch 4 | Key1=(Abloy, 4, Short, Broad), Key2=(Chubb, 3, Medium, Broad), Key3=(Rubo, 5, Long, Narrow) | No |

Table 3.1: A reduced version of the original training set for the bunch of keys problem.

a fixed number of traits: size (long or short), shape (rectangular, oval, u-shapped, etc.), number of wheels (2 or 3), type of roof (flat, peak, none, etc.), type of load (circle, square), etc. Finally, each train is labelled according to the direction it takes, namely, eastbound or westbound (see Figure 3.1). Logically, the goal is to find a classifier which tells the direction of the train. A possible solution to this problem is *"A train is eastbound iff it has a short closed car"*.

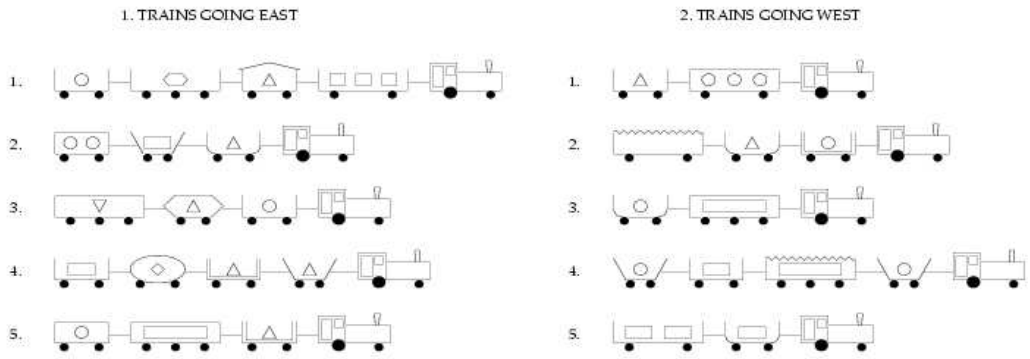


Figure 3.1: Trains going East and West.

One could judge the previous problems to be artificial and poorly connected to real-life. But the true thing is that many real problems look structurally similar to these two examples. For instance, as for the first problem, this illustrates a particular learning scenario known as *multiple-instance learning*. This extends classical propositional learning in the sense that examples are described by a set of tuples instead of only one, where at least one of the tuples in a set might be responsible for its classification. This phenomenon specially occurs in drug activity prediction (e.g. the musk problem also introduced in [25]) where a molecule takes more than one possible chemical conformation and all of the different conformations must be taken into account for classification. Regarding the second problem, if we change trains by documents and cars by words, what we have now is a text classification problem which is a hot topic in information retrieval.

In order to develop learners for structured data, an important question to be considered is to know how structured data is going to be represented. Unlike propositional representations where data consists of fixed-length tuples of nominal or numerical values, structured data can be from a very different nature: sets, lists, graphs, or even nested data types such as lists of tuples or lists of sets, etc. Therefore, there are in the literature several approaches for structured data representation, where as usual, each one of them has its own strengths and weaknesses. Next, we review some of the most important.

## 3.2 Structured Data Representation

Since structured objects are made up of (sub)parts related to each other, the languages or techniques we need to represent structured information must be able to refer these (sub)parts and connections among them. In this line, we can find learning algorithms whose input data is described using formal languages (e.g. first-order logic) or algorithms (specially non-symbolic learners) which simply require from classical data structures to represent complex objects such as lists, trees, graphs, etc.

Given the importance that logic has in automatic reasoning and very specially in symbolic learning, this section outlines some of the data representations based on this formalism.

### 3.2.1 Representations Based on First-Order Logic

First-order logic constitutes an elegant framework for structured data learning. This formalism allow us not only to represent complex objects but also, if needed, to state rules or conditions the objects must satisfy [126]. For instance, we can define rules such as *"if a part of an object satisfies that... then..."*, *"all the objects having certain parts in common then..."*, etc. In other words, first-order logic can be used for both structured data and hypothesis representation.

Due to the high expressiveness of first-order logic, only a subset of it is employed for learning purposes. In this way, within the context of first-order logic, there exist two widely-used approaches for data representation, the so-called ISP (individual-structural-property) and term-based representation.

1. ISP (also called DATALOG or flat) representation: this is a subset of Prolog (which is, in turn, a subset of first-order logic) where the only functors permitted are constants (indeed, objects and (sub)parts in an object are represented or identified by constants. For instance, if we look at the description of the first westbound train given below, we have that this train is identified by the constant $c1$, the first car in this train by $c1$, the second car by $c_2$ and so on). Atoms are used to describe objects and these are grouped into three categories: individual, which represents the target predicate (the concept to be learnt), structural, which represents relationships among (sub)parts in an object, and property, which describes these (sub)parts.

   In every data description, two sections are distinguished. The first one is where the predicates to be utilised by the object descriptions are declared (declarative section), and the second one contains the object descriptions. For instance, the declarative section corresponding to the East/West challenge could be as follows (this example has been adapted from the one reported in [53]):

   ```
   --DECLARATIVE SECTION
   ```

```
--INDIVIDUAL
eastbound 1 train_dom
--STRUCTURAL
first_car 2   1:train_dom 1:car_dom
next_car  3   1:train_dom 1:car_dom 1:car_dom
has_load  2   1:car_dom   1:load_dom
--PROPERTIES
car_shape   2  car_dom   cshape_dom
car_length  2  car dom   length_dom
car_roof    2  car_dom   roof_dom
car_wheels  2  car_dom   wheels_dom
load_shape  2  load_dom  lshape_dom
load_number 2  load_dom  lnumber_dom
class       2  train_dom class_label
```

The names at the leftmost column are predicate symbols while the numbers beside them are their respective arity. Next, the domains of the arguments are specified. The domains *train_dom*, *car_dom* and *load_dom* are constants referring to objects and (sub)parts in an object, while the domains *cshape_dom*, *length_dom*,... and *lnumber_dom* limit the values that properties can take. For instance, *car_roof* is a binary predicate whose first argument belongs to the domain *car_dom* and identifies a car while the second belongs to *roof_dom* and says what kind of roof (*roof_dom=none, peaked, flat, jagged*) the car has. A specially mention goes for structural predicates where, in addition, the cardinality of the relationship is expressed. For instance, *first_car* relates a train with one and only one car. One-to-n relationships are denoted by the symbol $*$. For instance, if the order of the cars in the train were irrelevant, then we could substitute the atoms *first_car* and *next_car* by the atom *has_car*. The declaration of this atom would be then

```
...
has_car 2 1:train_dom *:car_dom
...
```

The atoms defined in the declarative section allow us to describe every train in terms of the characteristics of its cars, the load of every car, and the order of the cars in the train. For instance, the first westbound train is defined as follows:

```
--DESCRIPTION
% 1st train going west
eastbound(t1) % negative example
first_car(t1,c1)              next_car(t1, c1, c2)
car_shape(c1, rectangular)    car_shape(c2, rectangular)
car_length(c1, long)          car_length(c2, short)
car_roof(c1, flat)            car_roof(c2, open)
car_wheels(c1, two)           car_wheels(c2, two)
has_load(c1,l1)               has_load(c2, l2)
load_number(l1, 3)            load_number(l2,1)
load_shape(l1, rectangular)   load_shape(l2, triangle)
```

According to this description, we have that this train is identified by the constant $t1$, its two cars are identified by the constants $c_1$ and $c_2$ and finally, the load of each car in the train is referred by $l_1$ and $l_2$, respectively.

The advantages derived from this representation when learning is a simplification of generalisation/specialisation operators utilised by the learning algorithms [53]. Additionally, ISP allows the learning algorithms to consider several individual viewpoints without changing the initial representation [95]. For instance, in the current formulation of the train problem, we are interested in learning the direction every train takes, but we could also have been interested in learning rules telling the amount of load the first car carries. In any of these two cases, the representation of the trains, except from the individual predicate, remains identical.

As the reader will have noticed, the main drawback of this representation is a loss of structure since the relationships between (sub)parts must be represented explicitly (e.g. the predicates symbols $first\_car$ and $next\_car$) [53]. This turns out to be scarcely natural and leads to the fact that simple patterns have to be expressed in a rather complex way. For instance, according to this representation, a first-order formula which states "trains having at least one short car" would be as follows:

$$\begin{aligned} \exists Y, \quad & (first\_car(X,Y) \ \wedge \ car\_length(Y,short)) \vee \\ & (next\_car(X,Y,Z) \ \wedge \ (car\_length(Y,short) \ \vee \ car\_length(Z,short)) \end{aligned}$$

2. Term-based representations: it is an attempt to obtain more intuitive representations and hopefully more intuitive patterns as well. In this new framework objects and (sub)parts in an object are not represented by constants but thereby ground terms as complex as it is required. Thus, $n$-ary functors and list structures can be used for data description. This has the advantage that all the information relative to one object is kept together. For instance, the first train going West would be something as simple as

```
westbound([car(rect,short,none,2,load(triangle,1)),
           car(rect,length,flat,2,load(circle,3))])
```

where now the only predicate symbol is the one referring the object while functors refer to (sub)parts of the objects and constants refers to properties of the (sub)parts.

One of the consequences derived from this representation is that we need auxiliary predicates to extract requested information which is packed in a term. Normally, these predicates are provided by the user thereby examining the semantic of the terms, that is, a *projection* operator will be useful for tuples, *member* for aggregates, *head* and *tail* for lists, and so on. For instance, the pattern above saying "trains having at least one short car" would be written as follows:

$$\exists Y, eastbound(X) \wedge member(car(Y, short, Z, T, U), X)$$

where $member(X,Y)$ returns true if $X$ is an element of the list $Y$. Although patterns are more readable when using a term-based representation we must notice that useful patterns might not be learnt if we have not previously defined the correct auxiliary predicates. Additionally, other shortcomings of this representation is that there exist some limitations to deal with trees and graphs and that data description is not independent on the individual viewpoint we take [95].

### 3.2.2 Representations Based on Relational Data Bases and Higher-Order Logic

Although widely-used, first-order logic is not the only formalism for structured data representation. Note that ISP framework holds a strong parallelism to relational data bases (RDB): namely, constants referring to objects or (sub)parts in the objects would correspond to *primary keys*, properties to *attributes* in a table, structural predicates to *foreign keys* (see [95] for a thorough analysis). Thus, it is not surprising then that RDB has been used as an alternative setting for structured data representation as well [6]. Furthermore thanks to the similarity between RDB and ISP, learning systems developed for the latter representation could easily be adapted to deal with RDB repositories becoming this a point in its favour.

Despite the success of first-order logic in machine learning, there exist relevant works in the literature which consider that languages based on typed higher-order logic can be a more convenient framework for knowledge representation [101, 102, 55, 52, 74, 50]. According to them, there are several reasons for this. For instance, first-order languages have a limited type system which makes it difficult to model aggregate structures such as sets and multi-sets. Furthermore, a term-based representation can be seen, considering a strong typed language, as a natural integration of propositional and structured-data learning since elements in tuples are not restricted to constants (nominal or numerical values) but these can be any data type (e.g. sets, lists, trees, etc.). Additionally, strong typed languages introduce a natural bias for hypothesis representation because every type carries its own set of operations (e.g. predicate *member* for sets, predicate *root* for trees, etc.).

The concept of function is another utility which can be very helpful for hypothesis representation. First-order logics artificially simulate functions by "considering" one of the arguments of an atom as the output argument. However, higher-order languages (e.g. Escher) can deal with functions and atoms in a uniform way since an atom is just a Boolean function. Given that functions can be nested, this leads to a very expressive setting for hypothesis representation. For instance, the pattern "trains having at least one short car" could be expressed as

$$Exists(X, length(\cdot) = short)$$

where $X$ is a train and $Exists$ is a higher-order function which returns true if there exists an element of $X$ satisfying the condition $length(\cdot) = short$.

After reviewing the different possibilities for structured data representation, the next step consists in defining methods capable to extract patterns from these descriptions. We will see how this can be carried out from two different perspectives: symbolic learners, which mainly deal with logical-based representations, and non-symbolic learners (with special emphasis on similarity-based methods) which do not need input data to be represented by any specific-purpose formalism.

## 3.3 Symbolic Learning over Structured Domains

The problem of learning symbolic theories from structured data has mostly been addressed in the context of first-order logic since, as seen, this formalism is close to human language which makes it adequate for hypothesis representation. The main work in this line is due to the field of Inductive Logic Programming (ILP). Other remarkable attempts concern the field of grammar inference which deal with list-based representations, extensions of decision trees and conceptual clustering algorithms for relational data.

### 3.3.1 Inductive Logic Programming

The subfield of machine learning concerned with learning from (and to) first-order logic representations is known as ILP. In fact, this name was coined by S. Muggleton when referring to the field which results from the intersection of inductive learning and logic programming. The advantages provided by first-order logic in order to develop learning algorithms are multiple: as seen, besides its expressiveness, we can take advantage of well-understood concepts, techniques and theoretical results from logic programming and its main language Prolog. This leads to well-founded learning algorithms. In addition, this formalism comes closer to natural language which implies that learnt hypothesis can directly be interpreted by users.

The ILP problem is defined as follows: given a finite set of positive and negative examples (respectively denoted by $E^+$ and $E^-$) and a finite set of clauses representing background knowledge $B$, we need to find a logic program (hypothesis) $H$ such that $B \cup H$ is correct w.r.t. $E^+$ and $E^-$. This is formally expressed by the following conditions:

- $B \not\models E^+$ (*prior necessity*)
- $\forall\ e^- \in E^-,\ B \not\models e^-$ (*prior satisfiability*)
- $B \cup P \models E^+$ (*posterior sufficiency*)
- $\forall\ e^- \in E^-,\ B \cup P \not\models e^-$ (*posterior satisfiability*)

Again, the central question is how to find a hypothesis $H$ satisfying these conditions. Different solutions to this problem are implemented in different ILP systems. Generally speaking, there exist two approaches the so-called bottom-up and top-down approaches. The former methods start from examples which are progressively generalised until a proper hypothesis is obtained. The latter start from an empty hypothesis which is progressively refined using the available examples. Instead of directly reviewing the main learning methods, we consider that it might be more instructive to previously explain, by means of a toy problem, the ideas these systems are inspired on. Definitions and learning techniques will progressively be introduced as the problem requires it. For this purpose, we start with a simplified version of the East-West challenging.

```
% Trains going East
(example 1) eastbound([car(rect, short, flat, 2, load(circle,1))])
(example 2) eastbound([car(rect, short, flat, 2, load(circle,1)),
                       car(rect, long, flat, 2, load(none,0))])
% Trains going West
(example 3) eastbound([car(rect, short, none, 2, load(circle,1))])
(exemple 4) eastbound([car(rect, long, flat, 3, load(circle,1)),
                       car(u-shaped, short, none, 2, load(none,0))])
```

Suppose we take eastbound trains as positive examples. Intuitively, we could see the classification problem at hand as finding a description which shows those traits (subterms) that the eastbound trains have in common but not the westbound trains. In our case, this description will be expressed by a first-order formula and initially, we will consider the simplest formulas, that is, atoms. Let us sketch a naive method to do this.

It seems reasonable to assume that a description, fitting positive examples as much as possible, does not cover (or at least, not many) negative examples. Then our method will have to find the

atom of the form $eastbound(something)$ which best fits eastbound trains. But we need to know first how to measure this fitness. Such a measurement could be given by a function over the number of functor and constant symbols an atom has. For instance, everybody understands that a description such as "*eastbound trains have a first short closed car with two wheels which carries one circle*" would fit better $E^+$ than the one saying "*eastbound trains have a first closed car*". In fact, the first description covers only positive examples while the second one covers positive examples but one negative example as well. In first-order logic this is formalised by the so-called *$\theta$-subsumption*. Thus, let $A_1$ and $A_2$ be two atoms, $A_1$ subsumes (is more general than) $A_2$ if there exists a substitution $\theta$ such that $A_1\theta = A_2$, where a substitution is a mapping which changes variables in an atom by terms. For instance, the two descriptions used before could respectively be formalised in the language Prolog as

$$A_1 \equiv eastbound([car(rect, short, flat, two, load(circle, 1))|X])$$

and

$$A_2 \equiv eastbound([car(U, V, flat, X, Y)|Z]).$$

If we set $\theta = \{U/rect, V/short, X/two, Y/load(circle, 1), Z/Y\}$ it is clear that $A_2\theta = A_1$ and hence, $A_2$ is more general than $A_1$. It is known that the $\sigma$-subsumption induces an order relation over the atoms headed by the same predicate symbol (*subsumption lattice*). Given that we initially want the description to best fit $E^+$, this is equivalent to find the *least upper bound* description of $E^+$ in the so-mentioned lattice. This description is also called the *least general generalisation* of $E^+$ ($lgg(E^+)$ in short) and was proposed by G. Plotkin in [129]. The $lgg$ of two atoms can algorithmically be computed by just taking the syntax of the atoms into consideration, thus we do not need to previously know the lattice. A definition of this can be found in any text dealing with logic programming or ILP. The one we report next is from [11]:

The $lgg$ of two literals $A_1 = (\neg)q_1(t_1, \ldots, t_n)$ and $A_2 = (\neg)q_2(s_1, \ldots, s_m)$ is undefined if $A_1$ and $A_2$ do not have the same predicate symbol and sign; otherwise:

$$lgg(A_1, A_2) = (\neg)q_1(lgg(t_1, s_1), \ldots, lgg(t_n, s_n))$$

where $t_i = t_i(t_{i1}, \ldots, t_{in_i})$, $s_i = s_i(s_{i1}, \ldots, s_{in_i})$ and $lgg(t_i(t_{i1}, \ldots, t_{in_i}), s_i(s_{i1}, \ldots, s_{in_i})) = V_i$ if $t_i \neq s_i$. Otherwise we proceed recursively.

Applying this procedure for the $lgg$ computation over the eastbound trains, we have that $lgg(E^+) = A_1$ and, as viewed, $A_1$ exclusively describes positive examples. Hence, we would already have learnt a plausible hypothesis. An interesting question to be mentioned here is what happens if $E^+$ possesses more than two examples. It can be proved that the $lgg$ is a commutative and an associative operator and i.e. the $lgg$ of a finite set of literals can be pairwise computed.

Back to our problem, it could be that $A_1$ adjusts too much (i.e. overfits) positive examples and unseen eastbound trains might not have all the traits required by $A_1$ (e.g. the fact of having a first closed short car...). All these other positive examples would be uncovered. More formally, we say that a new positive example might not be subsumed by $A_1$ and will be misclassified. To avoid overfitting, $A_1$ should be generalised a little more. This is achieved by changing terms by new variables. This operation is called (*inverse substitution*). As expected, this process is driven by the negative examples in such a way that generalisation is performed as long as negative examples remain uncovered. Therefore, a whole term in $A_1$ is substituted by a variable if the new resulting atom does not subsume negative examples.

For instance, although the atom $A_2$ above is more general than $A_1$ (and can be obtained by successively applying several inverse substitution steps over $A_1$), $A_2$ is not an example of this, since

$A_2$ covers one negative example. We would need then something between $A_1$ and $A_2$, namely, $eastbound([car(rect., short, flat, X, Y|Z])$. It is easy to see that this latter hypothesis performs better than $A_1$ over the original training set.

Although simple, this scheme (generalise while not covering negative examples) is followed by most of the bottom-up systems. Of course, the generalisation and refinement operators involved are much more complex. In connection with this, the reader can foresee that the $lgg$ operator for atoms might not be powerful enough for learning in general. We illustrate this in the context of our toy problem. In this way, also note that we could have considered westbound trains as positive examples instead of eastbound trains. Let us see what rules we would find out by applying the learning process we have sketched. First, we have to compute the $lgg$ of westbound trains which, according to the definition, is $westbound(X)$. That is to say, the most specific description which covers positive examples also covers all the negative examples!. However, this result turns out to be counterintuitive since westbound trains have traits in common which are not shared by eastbound trains, for example, we could say that "*westbound trains have a short open car*". The reason why this explanation cannot be obtained is because the position of the short open car in westbound trains does not match and i.e. the $lgg$ for atoms cannot detect this common trait!. This problem did not come up in the previous case (when eastbound trains were taken as positive examples) because of a simply matter of luck, that is, because the position in the trains of the short closed cars matched.

Therefore, despite existing potential classification rules for a problem, a learning method based on the $lgg$ for atoms might be unable to find them because of the fact that the common subterms simply occur at different positions in two different atoms. We could try other more sophisticated generalisation techniques than $lgg$ for atoms or see what possibilities top-down strategies can offer in order to get round this limitation. Since we have started with a bottom-up approach, let us explore first the former option.

The problem when applying $lgg$ over term-based representations is that only (sub)terms in the same position are compared, which is too much restrictive for our case. A solution to this problem consists of defining a new $lgg$ which disregards the position of the information. In order to introduce this, we need to work with flattened representations instead of term-based representations. When the examples are flattened, all the information concerning the structure and the properties of the examples is viewed as background knowledge. An example of this is shown next (for the sake of simplicity, we opt here for a more straightforward representation by changing the predicates $first\_car$ and $next\_car$ by the more generic $has\_car$):

```
% negative examples (Eastbound trains)
westbound(t1)
westbound(t2)
% positive examples (Westbound trains)
westbound(t3)
westbound(t4)
% Background theory
% e1
has_car(t1, c11). car_shape(c11, rect). car_length(c11, short). car_roof(c11, flat).
car_wheels(c11, 2). has_load(c11, l11). load_shape(l11, circle). load_number(l11, 1).
% e2
has_car(t2, c21). car_shape(c21, rect). ...
% e3
```

```
has_car(t3, c31). car_shape(c31, rect). car_length(c31, short). car_roof(c31, none).
car_wheels(c31, 2). has_load(c31,l31). load_shape(l31,circle). load_number(l31,1).
% e4
has_car(t4, c41). car_shape(c41, rect). ...
```

If we apply *lgg* over this representation, for example over the third and fourth examples, we obtain something as trivial as $lgg(example_3, example_4) = westbound(X)$ because it does not take the background knowledge into account. Thus, our new *lgg* must be able to take the background information relative to individuals into account. This extension is known as the *lgg* relative to a background theory (*rlgg* in short) and was proposed by G. Plotkin. This comes motivated by the fact that useful generalisations can be missed if the background knowledge is ignored by the *lgg*. Although *rlgg* avoids this, its computation might not be so easy. In fact, Plotkin showed that *rlgg* might not exist or it can be computationally very expensive due to the number of literals generated. As a matter of fact, in [120] we can find an example of the *rlgg* for the quicksort program which has a length of the order of thousand of millions of literals.

In particular, it is known that if $B$ is a finite set of background literals (as it is in our case) the *rlgg* of two atoms $e_i$ and $e_j$ corresponds to:

$$lgg(e_i, e_j) = lgg(e_i \leftarrow B_{ij}, e_j \leftarrow B_{ij}) \tag{3.1}$$

where $B_{ij} \subset B$, which contains all the literals referring the example $e_i$ and $e_j$. Regarding our problem, if we consider the examples $e_3$ and $e_4$ then

$$\begin{aligned} B_{3,4} \quad = \quad & \{has\_car(t3, c31), car\_shape(c31, rect), car\_length(c31, short) \ldots \\ & has\_car(t4, c41), car\_shape(c41, rect), \ldots\} \end{aligned}$$

The logical equation above (3.1) shows that in this particular case the *rlgg* can be computed as the *lgg* of two well-defined clauses. Thus, we need to define how the *lgg* for clauses can be calculated. Viewing a clause as a finite set of literals, that is, $C_1 = \{l_1, \ldots, l_k\}$ and $C_2 = \{m_1, \ldots, m_n\}$ then

$$lgg(C_1, C_2) = \{lgg(l_i, m_j),\ \forall l_i \in C_1 \text{ and } m_j \in C_2\}$$

Additionally, the *lgg* for clauses is also commutative and associative, so it can easily be extended for sets of clauses. Once explained, let us calculate the *rlgg* of the westbound trains. This is, by definition:

$$\begin{aligned} rlgg(e_3, e_4) \quad = \quad & lgg(e_3 \leftarrow B_{3,4}, e_4 \leftarrow B_{3,4}) \\ = \quad & westbound(X) \leftarrow has\_car(X, Y),\ car\_shape(Y, rect),\ car\_length(Y, L), \\ & car\_roof(Y, R),\ has\_load(Y, Z),\ load\_shape(Z, circle),\ load\_number(Z, 1), \\ & has\_car(X, Y'),\ car\_shape(Y', S),\ car\_length(Y', short),\ car\_roof(Y', none) \\ & has\_load(Y', Z'),\ load\_shape(Z', LS),\ load\_number(Z', LN) \end{aligned}$$

Therefore, the *rlgg* does not result in the trivial generalisation $westbound(X)$ as *lgg* does. On the contrary, this might be now much too specific. Note that most of the literals in the clause body are redundant and do not provide relevant information at all. According to Plotkin [129], a literal $l$ is logically redundant in a clause $C$ w.r.t. $B$ if $B \wedge C \vdash C - \{l\}$. From the example above, some literals could dropped and the following clause would be obtained.

$$\begin{aligned} westbound(X) \quad \leftarrow \quad & has\_car(X, Y),\ car\_shape(Y, rect), \\ & has\_load(Y, Z),\ load\_shape(Z, circle),\ load\_number(Z, 1), \\ & has\_car(X, Y'),\ car\_length(Y', short),\ car\_roof(Y', none) \end{aligned}$$

Unfortunately, redundant literal detection is at worst a semi-decidable and at best highly inefficient since it requires theorem proving [120]. Furthermore, even having removed all redundant literals, the clause can still contain a high number of literals. As mentioned for $lgg$ with the inverse substitution, in the case of $rlgg$, this can be generalised if more literals can be dropped, by using negative examples. That is, a literal is removed as long as negative examples are not covered. By applying this in our clause, we could have:

$$westbound(X) \quad \leftarrow \quad has\_car(X, Y'),\ car\_length(Y', short),\ car\_roof(Y', none)$$

which is just the pattern we were searching for. Removing redundant literals and clause refinement using negative examples together with other important concepts such as *h-easy models* [154] and *ij-determinism* [120] are employed by the ILP system GOLEM [120]. Indeed, GOLEM is the only system explicitly based on $rlgg$ and also considered as the first $ILP$ learner capable to extract meaningful knowledge from real-world problems.

Let us see which other possibilities exist to design bottom-up ILP learners. Note that, in terms of first order logic, the $lgg$ operator is the dual version (the inversion) of a deductive operator named *most general unifier* ($mgu$), which computes the greatest lower bound of two atoms in the lattice of atoms. This formal relationship between both the $lgg$ and the $mgu$ suggests that other generalisation techniques could be obtained by inverting well-known deduction mechanisms. In this line, we have the so-called *inverse resolution* and *inverse entailment* proposed in [119] and [115], respectively.

Before going into inverse resolution it is convenient to briefly mention something about *resolution*. This is a *procedure proof* (deduction rule) (a way to see whether a formula is a logical consequence of a set of premises or not) broadly used in automatic theorem proving and logic programming. It was introduced in [145] and is formulated as follows: let $\phi$, $\mu$ and $\aleph$ be three well-formed logic formulas. Then

$$\frac{p_1 \equiv \phi \vee \mu,\ p_2 \equiv \neg\phi \vee \aleph}{R \equiv \mu \vee \aleph}$$

where $p_i$ are the premises of the theory $\Sigma$ and $R$ (the logical conclusion) is called the *binary resolvent* of $p_1$ and $p_2$. Resolution is a sound deductive principle since every formula $F$ which can be derived by iteratively applying this procedure over a theory $\Sigma$ is a logical consequence of $\Sigma$. Informally speaking, this is so because if $p_1$ and $p_2$ are true then at least one of the formulas $\phi$ or $\aleph$ is true and i.e. $R$ is true (see [126] for a formal proof).

Let us change the initial conditions of this framework and suppose that $p_1$ and $R$ are known but not $p_2$. Now, the goal is to see whether the premise $p_2$ can algorithmically be obtained from $p_1$ and $R$. If this is so, we would have a generalisation operator. Why? if $\Sigma = \{p_1\}$ is viewed as our current hypothesis and $R$ is viewed as an example not covered by $\Sigma$, what we want then is to extend our initial $\Sigma$ (make the theory more general) so that the new example is covered. In this way, $\Sigma' = \{p_1, p_2\}$ is just the extended theory which covers $R$. Let us see that it is possible to algorithmically derive $p_2$ from $p_1$ and $R$.

1. Let $p_1 = L_1 \vee C_1$ such that $L_1$ is a literal and $C_1\sigma_1 \subset R$ for some substitution $\sigma_1$.
2. If we let $p_2 = L_2 \vee C_2'$, then it is enough to find a substitution $\sigma_2$ such that $L_1\sigma_1 = L_2\sigma_2$ and $C_2'\sigma_2 = R - C_1'\sigma_1$.

This algorithm is known as *V-operator*. Next we put an example to show how it could be used for inductive learning. Our theory is $\Sigma = H \cup B$ where initially $H = \emptyset$. A new positive

example $e$ is provided, for instance, $e = westbound(t_3)$. Initially, $\Sigma \not\models e$. Thus, we set $p_1 \equiv has\_car(t_3, c_{31})$ and $R \equiv e$. By applying the algorithm above, the clause $p_2 \equiv westbound(t_3) \leftarrow has\_car(t_3, c_{31})$ is obtained. Now, $H = \{p_2\}$ and $\Sigma \models e$. However, we could think that the hypothesis $train(e_3) \leftarrow has\_car(e_3, c_{31})$ gives little information about the example $e$, then we can iterate the process as many times as we need. For example, by successively iterating the clause $westbound(X) : -has\_car(X, Y), car\_length(X, short), car\_roof(Y, none)$ could be obtained (see Figure 3.2).

Unfortunately, the method we have just sketched is still far from being practical because both the method itself and the algorithm computing the $V$-operator are not deterministic: namely, we have said nothing about which literal in $B$ should be drawn when letting the premise $p_1$ and how the substitutions $\sigma_1$ and $\sigma_2$ should be chosen since there might be many possibilities. All these open issues (among others) must be addressed by any learning system which employs inverse resolution as a generalisation operator. One of these systems (and perhaps the best-known) is CIGOL [119] which implements inverse resolution via the so-called truncation, $V$, and $W$ operators. This latter operator, which has not been explained here, is even able to introduce new atoms which did not exist previously in $B$. This property is called *predicate invention*. The *rlgg* operator is implicitly utilised by being embedded in the truncation operator. Additionally, a heuristic based on structural complexity and the user himself/herself also guides the hypothesis search. Other similar approaches in this line are IRES [148] and ITOU [147] which introduce a variation of the $V$-operator called *saturation* in order to overcome some completeness limitations.

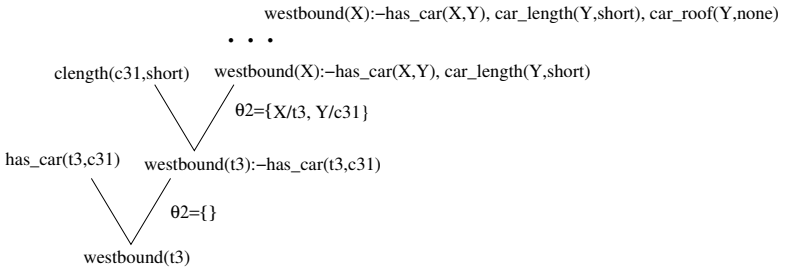


Figure 3.2: Applying inverse resolution iteratively.

Finally, to conclude with bottom-up methods, let us mention the generalisation technique called *inverse entailment*. It might happen that a clause $C$ does not $\theta$-subsume a clause $D$ but $C \models D$. This specially occurs in recursive clauses (see [11, 126] for concrete examples of this). Hence, it seems interesting to define generalisation of clauses in terms of entailment ($\models$) instead of $\theta$-subsumption. That is, $C$ is more general than $D$ if $C \models D$. This implies that if we are able to invert entailment, we would have a generalisation operator playing the same role that the *lgg* plays w.r.t. $\theta$-subsumption. The problem is that inverse entailment is undecidable even for Horn clauses [105]. However, since inverse entailment offers a more suitable framework for recursive clauses induction, some important research has been done in this line in order to apply inverse entailment in practise. One of these achievements is the system PROGOL [115] which has shown effectiveness in biomedical domains.

As mentioned, the problem of learning from first-order descriptions can also be addressed from a top-down perspective where the most general hypothesis is this time used as starting point for the search (e.g. $westbound(X) \leftarrow true$ in our case) and since this undoubtedly covers negative examples

we have to refine (specialise) this initial hypothesis. Roughly speaking, top-down methods tend to fit the covering schema followed by propositional rule learners (see Chapter 2): that is, if the current hypothesis $H$ still covers negative examples, then a new clause $C$ is learnt and added to $H$ while the positive examples covered by $C$ are removed. The process stops when all positive examples are covered or no new clauses can be built. In a similar fashion, a clause is built by adding literals until a negative example is covered. Perhaps, the most meaningful difference w.r.t. the propositional version is the huge number of possibilities coming up when building the body of a clause. The main top-down ILP learners deal with this fundamental issue in a different manner, basically, imposing specific bias for the search process, the hypothesis representation, etc.

The system MIS (*Model Inference System*) was the first real ILP learner and was developed by E. Shapiro in the early 1980's [154]. The system starts with the most general clause and the hypothesis is refined as examples are incrementally provided. If the example provided is positive and it is not covered by the current hypothesis, then the hypothesis is generalised by removing literals in the body of existing clauses or by adding a new clause or (*refinement process*). Let $Q(e_1, \ldots, e_n)$ be the new positive example not covered by the hypothesis, then the clause $Q(X_1, \ldots, X_n)$ and refinements of this are explored in a breadth-first search. The search stops when a clause which covers the new positive example but negative ones is found. On the contrary, if the example provided is negative and the current hypothesis covers it then the hypothesis must be specialised. This is done by adding new literals in the body of existing clauses or by removing a clause which might be incorrect (*contradiction backtracking algorithm*).

Perhaps, the system FOIL (proposed by J.R. Quinlan) is one of the most popular and broadly used ILP learner [134, 135]. Basically, this is a top-down system which can be viewed as an upgraded ID3. This is so because, unlike MIS, a best-first search (guided by an entropy-based criterion) is implemented. In principle, only those literals with the greatest gain (*gainful literals*) are included in the body clause. Literals to be chosen do not have ground symbols (except from a particular kind of literals of the form $X$ *symbol* $a$. where *symbol* can be $=, >, <, \ldots$) and, at least, one of their variable symbols must have appeared in a previous literal in the clause. As mentioned, literals are one-by-one selected and included in the clause body as long as the clause resulting from adding this literal does not cover negative examples.

FOIL is specially sensitive to the look-ahead problem. Thus, if a combination of two literals $A$ and $B$ turns out to be relevant for the hypothesis to be learnt, and $A$ and $B$ separately have a low (even negative) gain, then none of them will never be chosen. For example, in our case, such a combination could be $A \equiv car\_length(X, Y)$ and $B \equiv Y = short$. Observe that $A$ and $B$ alone do not say much about the train direction but things change when combined. To get this round, the notion of *determinate literals* is introduced. Intuitively, this refers to a literal such that all positive examples satisfy but negative ones do necessarily not. Election of determinate literals is priorised unless a literal with a really high gain exists.

Another interesting trait in FOIL to be mentioned is its capability of learning terminating recursive theories by defining a syntactic order over terms in a similar fashion as *term writing systems* do [24, 20].

FOIL is not the only ILP system inspired on propositional learners. For example, following with the trend of upgrading decision trees we find TILDE [73, 15]. The test in each node is a conjunction of literals (a clause without a head), that is, a test in a sibling node extends the test in its parent node by adding a new literal. To reduce the branch factor, the user previously introduces some specifications about how many times a literal can appear in a conjunction, the input/output condition of the variables along with their type. A remarkable difference w.r.t. FOIL is that TILDE can be used

for regression. Furthermore, an extension of TILDE (called TIC and proposed by the same authors [16]) looks for bringing together classification/regression and clustering. Another system in this line is S-CART [91] proposed by S. Kramer which extends CART in order to deal with first order logic theories. Hence the extension is able to do both classification and regression. Furthermore, in [96] a methodology is proposed for extending propositional learners towards first-order learners where the methodology is based on upgrading CN2.

The ILP systems described and others not described here have been used in a wide range of real-life domains. For example, in scientific knowledge discovery (chemistry, ecology, biology, medicine, etc.) [121, 118], in relational data base mining [31], computational linguistics [117], software engineering [60, 11], computer games [140], music[8], etc.

### 3.3.2 Propositionalisation

According to [92], propositionalisation is the set of techniques which allow the transformation of relational problems into attribute-value representations. Note that once the attribute-value representation is obtained, then we can use any appropriate propositional learner to solve the problem.

The main advantage of solving the structured problem in the *feature space* (the space of tuples where each tuple represents an structured object) comes from the wide variety of propositional learners we have in order to choose the one which fulfils our purposes. However, it is easy to see that the performance achieved by the propositional learner will strongly be conditioned by the quality of the transformation, which is a serious question because transforming structured data into tuples of values is not a straightforward task. The main drawback is the low expressiveness of attribute-based representations compared with structured-based representations. Basically, the number of attributes in a tuple is fixed beforehand and it constitutes a serious restriction when trying to represent structured objects. Obviously, each structured object consists of a variable number of (sub)parts and of a variable number of relations among the (sub)parts, hence, we would also need a variable number of attributes in each tuple which is not permitted.

In principle, this problem could be overcome by setting an upper bound for the number of (sub)parts an object is made up. But this rough solution brings two immediate drawbacks. On the one hand, if the number of subparts in an example is lower than the so-mentioned bound, this implies that this example will be represented by a tuple with a considerable number of null components. On the other hand, we might have unseen examples with a number of (sub)parts exceeding this bound which causes this example not to be correctly represented [136]. For instance, back to the East-West challenging, we could be optimistic and set a maximum of 4 cars per train. Since 5 traits are required for a car description, this means that a train would be represented by a 20-dimensional tuple. Now observe that most of trains have only 2 or 3 cars which make a total of 10 and 5 null attributes per tuple, respectively. But what if a train with 5 cars is provided in the future? How will it be represented then? Which car should be disregarded?

From the discussion above, it seems that propositionalisation cannot be effective if we decide to map the structure of an example into a tuple. However, things can be more promising if we focus on finding properties describing subsets of the training data [58]. Initially, if this collection of properties is found, then every example in the training set can be represented by a tuple of Boolean components where each component in a tuple says if one given property holds or not. These properties are usually called *first-order features* and the process of extracting features is known as *feature construction*. For instance, suppose the following first-order features for the East-West Challenging problem.

- Feature 1 ≡ There exists a short car in the train.
- Feature 2 ≡ There exists a long car in the train.
- Feature 3 ≡ There exists a car with more than two wheels in the train.
- Feature 4 ≡ There exists an empty car in the train.

According to this, the first train going East would be represented by $(1, 1, 1, 0)$, the second one would be $(1, 0, 0, 0)$ and so on. Of course, the more features we have, the more accurate the mapping will be. Observe that with the features above, the first, the third and the fifth train going East, and even worse, the second and third train going West have the same representation. To avoid loss of information during the transformation, we should consider all the possible features, which is called *complete propositionalisation*, but this not reasonable because of the huge amount of potential features we might have. Note how many features could be written as for our small problem: *"all cars in a train have a roof"*, *"at least one of the cars has a roof"*, *"only one car at the train has a roof"*, ...,*" the first car satisfies whatever'*, ...,*" the last car satisfies whatever"*, ....

To face this combinatorial explosion of possibilities, we need to introduce some kind of reasonable bias over the feature space. In [58], features are restricted to a well-defined combination of structural (relationships) and utility (properties) predicates; in [59] only features with one utility predicate are permitted; we can also limit the number of literals in features or to use some kind of utility threshold over features, for instance, most frequent features (e.g. APRIORI [1]) and correlation with the class label for the supervised case (TERTIUS [57]).

Well-known systems based on propositionalisation are SINUS (see a description in [93]), which is an evolution of the propositionalisation-based systems LINUS and DINUS [97], RSD [174], MIDOS [168] and RELAGSS[94].

### 3.3.3 Learning from Multi-relational Data and Higher-Order Logic

As pointed out in Section 3.2, besides first-order logics other formalisms could be employed for structured data representation. The Relational Model (and hence relational data bases, RDB) was one of them. Given the strong relationship between RDB and ISP representations, every ILP learner dealing with ISP descriptions becomes an attractive method for pattern discovery in data bases: namely, transform a database into an ISP representation and then run the ILP system. Another possibility is proposed in [87] where we find a method which extends classical decision trees to be able to directly deal with databases. Essentially, each Boolean test is a *multi-relational pattern* (an SQL query) in such a way that the records in a *target table* satisfying this SQL query are placed in a node while the remaining ones in the other. An initial implementation of this algorithm is found in [98] while an enhanced version is reported in [6].

Recall that higher-order logic was another formalism for structured data representation. In [17, 102] a decision tree learner is presented where Boolean tests correspond to higher-order formulas which are automatically derived from a term rewriting system previously supplied by an expert. An accuracy-based heuristic is employed for hypothesis search.

#### Grammar inference

Sequences is another data type widely used in machine learning. Sequences are employed for data representation in multiple scenarios (e.g. control theory, pattern extraction, speech translation,

computational biology). The area devoted to learn from sequences is called grammar inference. Here, the central problem is to learn a formal representation of a language from a set of positive examples (sequences belonging to the language) and a set (perhaps, empty) of negative examples (sequences belonging to the complementary language). There exist several approaches to solve this problem. A widely-studied family of techniques is the so-called *state merging algorithms*. Among them, the RPNI (regular positive and negative inference) algorithm might be the best-known. In a nutshell, a prefix tree acceptor (PTA in short) is built from positive examples (see left-hand side picture in Figure 3.3). A generalisation is performed by merging states. This process carries on until a negative example is covered (see right-side in Figure 3.3). The manner in which states are selected to be merged depends on the algorithm.

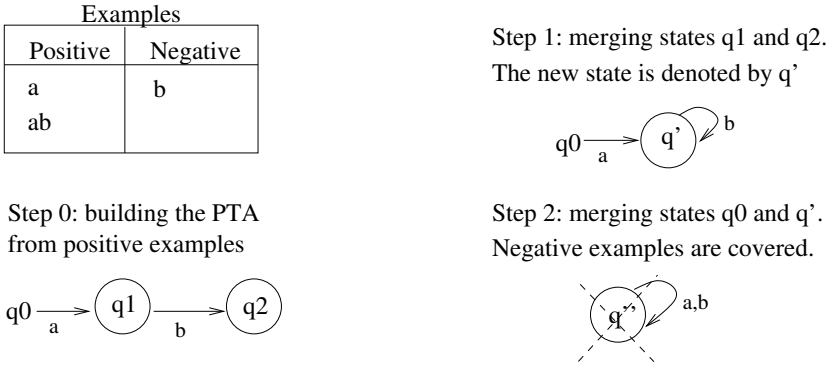


Figure 3.3: An overview of the RPNI algorithm. Accepting states are represented by a simple circle.

RPNI algorithm assumes that the target language is regular, i.e., it can be expressed as a regular expression or a finite automaton. But there exist in the literature research about grammar inference over other (sub)families of formal languages. For instance, subfamilies of regular languages from which the learning process can efficiently be done using non deterministic finite state automata as hypothesis representation, subfamilies of context-free languages (e.g. even linear languages), etc. (see [23] for further information).

### 3.3.4 Conceptual Clustering over Structured Domains

Most of the conceptual cluster systems work over propositional data. A first attempt to extend conceptual clustering for complex domains is reported in [158] which introduces the system CLUSTER/S. Basically, this algorithm upgrades its predecessor CLUSTER/2 by making use of a first-order language for data and hypothesis representation as well as background knowledge provided by an expert. As for the language, this is in fact an extension of first-order logic called *annotated predicate calculus* (APC). The major differences w.r.t. standard first-order logic is that functions and data types are permitted. As a consequence of this, certain elements of the language such as variables, functions and predicates are allowed to store information (*annotations*) about the domain over which are defined, constraints, etc. For instance, the train 1 in the Example reported in Section 3.3.1 is

written in APC as

$$contains(train, car), shape(car) = rect, length(car) = short, \ldots, number\_of\_wheels(car) = 2, \ldots$$

Background knowledge is organised in the basis of two important concepts: firstly, the so-called *goal interaction network* (GIN) which is a graph designed by an expert informing about which features can be more relevant for the learning process. This information is taken into account by adding/dropping relevant/irrelevant features to/from example descriptions. In fact, the heuristic function used by CLUSTER/2 is modified here in order to increase the probability that relevant features are incorporated to data descriptions. Secondly, we have in addition *background rules* (also provided by the expert) which allow the derivation of high-level features from other more basic ones. Higher-level features can also be added to object descriptions as long as they are relevant enough for the learning problem at hand.

Another interesting work in this line was proposed in [13] long time after the system above had been published. Here, the author presents the $KBG$ system which takes examples described by a set of first order predicates. A similarity function and a generalisation procedure are defined for every predicate. Similarity between examples is obtained from the similarity among the predicates in the example. The algorithm follows a hierarchical clustering schema. When similar examples or clusters are put together, a generalisation of these examples is computed. Unlike conventional hierarchical algorithms, the output is a an acyclic graph of concepts (logic formulae) instead of a tree, since generalisations are treated by the system as new examples.

## 3.4 Non Symbolic Learning over Structured Data

In this section we review several approaches for non-symbolic structured learning. We mainly focus on similarity-based learning, and more concretely, the specific case when similarity functions are distance functions. Other proposals, although really important, are mentioned briefly because they are less related to this thesis.

### 3.4.1 Similarity-based Methods for Structured Data: Distances, Pseudo-Distances and Semi-Distances

Similarity-based methods can directly handle structured data as long as a similarity function over the structured domain is provided. A distance or a metric function [1] is a desirable form for similarity (or better, dissimilarity). According to the set of properties a distance must satisfy, we can ensure that when the distance between two elements whatever $x$ and $y$ is equal to zero then $x = y$. This important consequence might not be satisfied by a generic similarity function, as we will see below. It would be nice a similarity function to be symmetric as well. Otherwise, it would turn out hard to understand that the similarity between two objects depends on the order they are compared. Finally, the last property (triangular property) is somewhat more difficult to be interpreted. Suppose that we have three objects $x$, $y$, and $z$; $x$ and $y$ are very similar to $z$ but it turns out that $x$ and $y$ are scarcely similar between them. More or less, this is what happens when triangular inequality is violated. Next, we review the most important distances for the most common data types.

[1] In what follows, the terms distance and metric are equivalent.

- Sets: it is known that the cardinality of the symmetric difference between two finite sets $A$ and $B$, that is, $|(A-B)\cup(B-A)|$ is a distance function. Clearly, identity and symmetry are satisfied. The triangular inequality can be easily shown by proving that, for any finite set $C$, if an element $x$ is in $(A-B)\cup(B-A)$ then $x$ is in $(A-C)\cup(C-A)$ or in $(C-B)\cup(B-C)$ too, which implies that $|(A-B)\cup(B-A)| \leq |(A-C)\cup(C-A)| + |(C-B)\cup(B-C)|$.

  When this metric is employed, the distance between two sets is given by the number of elements they have in common. The more elements they have, the nearer they are. For instance, given the following sets of sequences $A = \{ab, a^4\}$, $B = \{ab, d^4\}$ and $C = \{ab, a^3\}$ then $d(A,B) = d(A,C) = 2$. Note that this metric function somehow assumes that the 0-1 distance ($d(x,y) = 1$ if $x \neq y$, 0 otherwise) has previously been defined over the elements from which the sets are built. In other words, the symmetric difference leads to a *non-grade* distance since all the elements are equally different to the rest.

  In some contexts, we need to capture the differences between elements in a more precise way. In this line, we have the *Hausdorff's distance* which is defined as

$$d_H(A,B) = \max\left\{ \sup_{a\in A}\inf_{b\in B} d(a,b), \sup_{b\in B}\inf_{a\in A} d(a,b)\right\}$$

  where $A$ and $B$ are two sets and $d(\cdot,\cdot)$ is a distance defined over the elements. A more detailed analysis of this distance as well as a nice explanation about its importance in topology and fractal geometry can be found in [33]. A use of this distance in machine learning can be read in [4].

  Despite this, Hausdorff's distance is not totally suitable for some applications since it does not take the overall structure of the set of elements into account. A consequence of this is that outliers can negatively influence on the final distance value.

  The work [34] is an attempt to overcome this. The authors propose a family of semi-distances (in general, triangular inequality is not satisfied) when a similarity function over the elements has been previously defined. For instance, in the example above, if we define a similarity function over sequences in such a way that the sequence $a^4$ is more similar to $a^3$ than to $d^4$ (something which is quite reasonable, indeed), then the resulting semi-distance over sets would establish that the set $A$ is more similar to $C$ than to $B$. A step further is taken by [137] stating under which conditions these semi-distances become a proper distance function. This is briefly explained next.

  The semi-distances we are talking about are calculated in the basis of a family of mappings between two sets. As a mapping $f$ links the elements of two sets $A$ and $B$, then $d_f(A,B)$ is the sum of the distances of the elements linked by the mapping plus a penalty for those elements missed by the mapping. The optimal $d_f$ is a semi-distance function. In the case that these mappings are matchings [2] and the elements of the sets are embedded in a metric space then the optimal mapping leads to a distance function for sets.

  More formally, given two sets $A$ and $B$ and the elements $a \in A$ and $b \in B$, we will say that the ordered pair $(a,b)$ belongs to the matching $\alpha_{A,B}$ between $A$ and $B$ (a subset of $A \times B$), if $\alpha_{A,B}(a) = b$. By $D(\alpha_{A,B})$, we denote the domain of the matching, that is, $D(\alpha_{A,B}) = \{a \in$

[2] A matching from the set $A$ to the set $B$ is an injective mapping not necessarily defined over all the elements in $A$.

$A : \exists(a,b) \in \alpha_{A,B}\}$, and by $\alpha_{A,B}(A) = \{b \in B | (a,b) \in \alpha_{A,B} \wedge a \in A\}$ we denote the codomain of the matching.

Thus, given two sets $A$ and $B$ and a matching $\alpha_{A,B}$, a similarity measure $d(\alpha_{A,B}, A, B)$ can be defined by summing the distances between the elements from the ordered pairs belonging to $\alpha_{A,B}$ and adding a penalty $M/2$ for each element in $A$ and $B$ not included in the matching. More formally,

$$d(\alpha_{A,B}, A, B) = \sum_{\forall (a_i,b_j) \in \alpha_{A,B}} d(a_i, b_j) + \frac{M}{2}(|B - \alpha_{A,B}(A)| + |A - D(\alpha_{A,B})|) \tag{3.2}$$

Finally, the distance between $A$ and $B$ is given by the optimal (minimal) matching among all the possible ones.

$$d_m(A,B) = \min_{\forall \alpha_{A,B}} d(\alpha_{A,B}, A, B) \tag{3.3}$$

According to [137], the constant $M$ must be equal to or greater than the maximal distance between two elements in $X$ in order to have a $d_m(\cdot,\cdot)$ which satisfies all the axioms of a metric. Next we illustrate this by means of an example.

**Example 1.** *Given the metric space $(X,d)$ where $X = \{a,b,c,d\}$ and $d$ is the distance given by the matrix below*

| $d(\cdot,\cdot)$ | $a$ | $b$ | $c$ | $d$ |
|---|---|---|---|---|
| $a$ | 0 | 1 | 2 | 3 |
| $b$ | $-$ | 0 | 2 | 3 |
| $c$ | $-$ | $-$ | 0 | 1 |
| $d$ | $-$ | $-$ | $-$ | 0 |

*It is deduced from the matrix above that $M \geq 3$ (we set $M = 3$). Let $A = \{a,b\}$ and $B = \{a,c,d\}$ be two sets in $2^X$ . The optimal matching is $\alpha_{A,B} = \{(a,a),(b,c)\}$. Then, the distance between $A$ and $B$ is:*

$$d(A,B) = d(a,a) + d(b,d) + 3/2 \cdot (1+0) = 0 + 2 + 3/2 \cdot 1 = 3.5$$

- Lists: several distance functions for lists can be found in the literature. One of them is the *Hamming distance* which is used for equally-length lists [70] where the distance between two lists is the number of positions for which the corresponding symbols are different. For instance, given the sequence $s_1 = aabb$ and $s_2 = accb$ then $d(s_1, s_2) = 2$. In [33] a distance is defined for infinite-length lists which can easily be adapted for finite lists by just appending a special symbol an infinite number of times at the end of each list. Given two sequences $s_1$ and $s_2$, $d(s_1, s_2) = (1/|\Sigma|)^n$ where $\Sigma$ is the alphabet and $n$ is the number of the longest contiguous prefix. For example, if $\Sigma = \{a,b\}$, $s_1 = aaab$ and $s_2 = aab$ then $d(s_1, s_1) = (1/2)^2$. Another distance for lists is the one proposed by J. Ramon in [139]. However, due to the character of this distance, this will be explained in more detail when we will talk about first-order atoms.

  The widest-used distance function for sequences is the edit distance (or Levenshtein distance [100]) which is a generalisation of the Hamming distance in that variable-length sequences are

permitted. This distance is the number of operations (insertion, deletion and substitution) required to transform one into the other. If the weight given to these operations is a distance over the alphabet of symbols then the edit distance is a proper distance over the set of finite sequences (see example below).

**Example 2.** *We assign* 1 *for insertions and deletions and* 2 *for substitutions. Given the sequences* $s_1 = aabb$ *and* $s_2 = bbcc$ *their distance is* 4*. The optimal transformation is specified by the alignment below.*

$$\begin{array}{cccccc} a & a & b & b & & \\ & & b & b & c & c \end{array}$$

*That is,* $s_1$ *is transformed into* $s_2$ *by deleting the first two a symbols and then by adding two c symbols at the end.*

The edit distance can be computed in polynomial time by applying dynamic programming (see [161]).

- Atoms: a distance for ground terms and atoms is proposed in [125] by defining a bounded distance which takes the depth of the symbols occurrences into account in such a way that differences occurring close to the root symbols count more. Given two ground terms/atoms $e_1 = p(s_1 \ldots, s_n)$ and $e_2 = q(t_1, \ldots, t_m)$, this distance is recursively defined as follows.

$$d(e_1, e_2) = \begin{cases} 0, & \text{if } e_1 = e_2 \\ 1, & \text{if } p \neq q \\ \frac{1}{2n} \sum_{i=1}^{n} d(s_i, t_i), & \text{otherwise.} \end{cases} .$$

For instance, if $e_1 = p(a, a)$ and $e_2 = p(b, b)$ then $d(e_1, e_2) = 1/4(d(a, b)+d(a, b)) = 1/4(1+1) = 1/2$. Atoms with variables as well as other more complex first-order objects, such as clauses, can be treated by considering them as sets of ground terms/atoms given by their respective least Herbrand's models and computing the distance between these sets via the Hausdorff's distance defined from the previous distance $d$ for ground atoms.

A different approach which allow us to deal with variable symbols based on syntactical aspects rather than semantical is presented in [80]. However, the similarity function proposed in this work is a pseudo-distance instead of a distance. Starting from the ideas developed in the so-mentioned work, [139] introduces a distance for first-order atoms. Basically, this distance returns an ordered pair of integer values $(i, j)$. This pair expresses how different two atoms are in terms of function symbols and of variable symbols respectively. An auxiliary function, the so-called $size(e) = (F, V)$, is required to compute $d$. This function encodes the structure of one atom $e$. That is, $F$ is a function counting the number of function symbols occurring in $e$ and $V$ returns the sum of the squared number of occurrencies of each variable in $e$. Finally, given the atoms $e_1$ and $e_2$, $d(e_1, e_2) = [size(e_1) - size(lgg(e_1, e_2))] + [size(e_2) - size(lgg(e_1, e_2))]$ (see Section 3.3.1 for $lgg$).

For instance, if $e_1 = q(a, f(a))$ and $e_2 = q(b, f(X))$ and knowing that $lgg(e_1, e_2) = q(Y, f(Z))$, $size(e_1) = (3, 0)$, and $size(e_2) = (2, 1)$, $size(lgg(e_1, e_2)) = (1, 2)$, the distance between $e_1$ and $e_2$ is given by the expression: $d(e_1, e_2) = [(3, 0) - (1, 2)] + [(2, 1) - (1, 2)] = (2, -2) + (1, -1) = (3, -3)$

For non-unifiable atoms, the distance is defined by means of introducing an artificial second-order symbol $\top$, which is considered the most general element, such that $size(\top) = (0, 1)$. Note that a total order relation (lexicographic order), defined over the set of ordered pairs, is needed to express how far two atoms are. Given two ordered pairs $A = (F_1, V_1)$ and $B = (F_2, V_2)$, $A < B$ iff $F_1 < F_2$ or $F_1 = F_2$ and $V_1 < V_2$. As the set of tuples is ordered, it permits us to handle these objects as they were real numbers.

- clauses: if we represent clauses as sets of atoms (or literals) then we can use the distance for sets based on optimal matchings in combination with the distance for atoms presented above, to obtain a distance for sets of literals, or equivalently, to obtain a distance for clauses. This is the idea proposed in [137].

- for graphs: in [19], an edit distance is defined for labelled graphs. To illustrate it, some previous definitions and concepts are required. Thus, a graph is a triple $g(V, \alpha, \beta)$ where $V$ is a finite set of vertices and $\alpha : V \to L$ and $\beta : V \times V \leftarrow L$ are a node and an edge labelling function, respectively. In this context graphs are always complete since missing edges are considered to be labelled with a special *null label*. Next, we introduce the concepts of *error-correcting graph matching* (ecgm), which is a set of edit operations which can transform one graph into other, and the cost associated to an ecgm.

  **Definition 1.** *Let $g_1 = (V_1, \alpha_1, \beta_1)$ and $g_2 = (V_2, \alpha_2, \beta_2)$ be two graphs. An error-correcting graph matching (ecgm) from $g_1$ to $g_2$ is a bijective function $f : \hat{V}_1 \to \hat{V}_2$ where $\hat{V}_1 \subset V_1$ and $\hat{V}_2 \subset V_2$.*

  **Definition 2.** *The cost of an ecmg $f : \hat{V}_1 \to \hat{V}_2$ from a graph $g_1 = (V_1, \alpha_1, \beta_1)$ to a graph $g_2 = (V_2, \alpha_2, \beta_2)$ is given by*

$$\begin{array}{rcl} c(f) & = & \sum_{v \in \hat{V}_1} c_{ns}(v) + \sum_{v \in V_1 - \hat{V}_1} c_{nd}(v) + \sum_{v \in V_2 - \hat{V}_2} c_{ni}(v) \\ & & + \sum_{e \in \hat{E}_1} c_{es}(e) + \sum_{e \in E_1 - \hat{E}_1} c_{ed}(e) + \sum_{e \in E_2 - \hat{E}_2} c_{ei}(e) \end{array}$$

  *where $c_{ns}$ is the cost of a node substitution, $c_{nd}$ the cost of a node deletion, $c_{ni}$ the cost of a node insertion, $c_{es}$ the cost of an edge substitution, $c_{ed}$ the cost of an edge deletion and $c_{ei}$ the cost of an edge insertion.*

Finally, the minimum distance between two graphs $g_1$ and $g_2$ is the minimum cost taken over all *ecgm*'s from $g_1$ to $g_2$. We illustrate this using the same example as in [19].

**Example 3.** *Let $g_1 = (V_1, \alpha_1, \beta_1)$ and $g_2 = (V_2, \alpha_2, \beta_2)$ be the two graphs depicted in Figure 3.4. We set the following cost:*

$$c_{ns}(v) = \begin{cases} 0 & , \quad \alpha_1(v) = \alpha_2(v) \\ +\infty & , \quad otherwise. \end{cases}$$

$$c_{nd}(v) = 1, \ \forall v \in V_1 - \hat{V}_1$$

$$c_{ni}(v) = 1, \ \forall v \in V_2 - \hat{V}_2$$

$$c_{es}(e) = \begin{cases} 0 & , \quad \beta_1(e) = \beta_2(e) \\ +\infty & , \quad otherwise. \end{cases}$$

$$c_{ed}(e) = 0, \ \forall e \in E_1 - \hat{E}_1$$

$$c_{ei}(e) = 0, \ \forall e \in E_1 - \hat{E}_1$$

*We define the mappings $f_1$ and $f_2$ where*

$$\begin{array}{rcl} f_1 & : & V_1 \to V_2 \\ & & v_1 \mapsto v_4 \end{array}$$

*and*

$$\begin{array}{rcl} f_2 & : & V_1 \to V_2 \\ & & v_1 \mapsto v_4 \\ & & v_2 \mapsto v_5 \end{array}$$

*It is also easy to see that $f_2$ is the lowest-cost mapping and i.e. $d(g_1, g_2) = c(f_2) = 2$.*

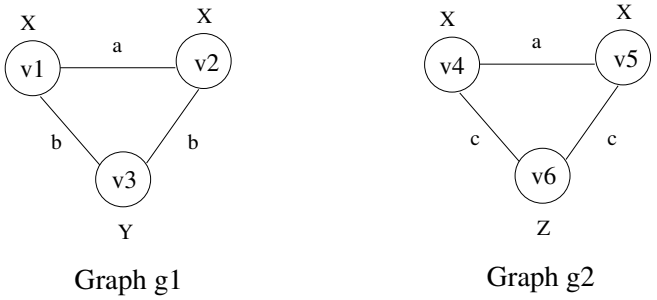


Figure 3.4: Computing the distance between the graphs $g_1$ and $g_2$.

**Learning with pseudo-distances and semi-distances**

To conclude with this section we also have to mention that although the notion of similarity can be expressed by means of distances, sometimes it is difficult or even unfeasible to make a similarity function satisfy all of the axioms mentioned above. In this case, it might be convenient to introduce some relaxations. Some well-known kind of "relaxed" distances are *pseudo-distances* and *semi-distances* which have deeply been studied both in mathematics and machine learning. Regarding pseudo-distances, identity is not ensured whereas triangular inequality might not be preserved when speaking of semi-distances. For instance, the similarity functions expressed via kernel functions (see subsection below) can easily be transformed into pseudo-distances. The similarity functions RIBL [35] and RIBL2 [79] proposed for multi-relational data are two examples of semi-distances employed in Machine Learning. Basically, they calculate the similarity between two objects by considering their properties at different levels. Recall that in RDB an object is represented by a collection of

tuples placed at different tables in the database. Level 0 corresponds to the tuple of the object placed at the main table, level 1 correspond to tuples placed at those tables directly linked to the main table and so on. Differences at high-level are weighted more than differences occurring at lower levels. RIBL2 is an extension of RIBL in that edit distances for lists are also used.

### 3.4.2 Other Approaches

Two important non-symbolic approaches for structured data learning are the so-called probabilistic-based and kernel-based methods. We briefly sketch the former group of techniques since this is little connected to the objectives of this thesis. In a nutshell, probabilistic methods deal with estimating a class conditional probability $P(x|c)$ over the structured domain. Once this function is known, we can build a classifier by applying Bayes' rule (see [56, 59, 66, 116, 85]).

On the contrary, given the relation which exists between kernels and (pseudo)-distances, a more detailed explanation about kernel-based methods is given below.

#### Kernel-based methods

An important family of similarity functions are the so-called kernels. For a better understanding of the role played by kernel functions in the field of learning from structured data, we start by giving some basic notions about one of the most popular kernel-based methods: *Support Vector Machines* (SVM). Thus, suppose a binary classification problem where examples $\{\mathbf{x}\}_{i=1}^{k}$, originally, lie in the *representation space* $R^n$. Given that a hyperplane splits $R^n$ into two disjoint parts, a classifier can be derived by simply labelling each one of these parts according to the majority class of the training examples falling into. Since infinite hyperplanes exist, a SVM aims to find the most adequate hyperplane for the problem at hand. The search for the optimal hyperplane requires the representation space to be endowed with an inner product. This latter thing suggests that if we are able to define an inner product in an abstract space, that is, a space of strings, trees, graphs, etc., a SVM could be used for classification problems. Well, a kernel function is nothing more than a way of expressing inner products (it is not exactly so, but it is enough for our purposes). When structured objects are involved, a kernel function can be defined from kernels defined over the parts of the objects. This "designing principle" was proposed in [72] by introducing the so-called *convolution kernels*. A common example of this is as follows: if we are interested in defining a kernel $K$ for sets of symbols, we can write

$$k(S_1, S_2) = \sum_{\forall (s_1, s_2) \in S_1 \times S_2} k_\delta(s_1, s_2)$$

where $k_\delta$ is, in turn, a kernel over the elements in $S_1$ and $S_2$ such that

$$k_\delta(s_1, s_2) = \begin{cases} 1, & s_1 = s_2 \\ 0, & \text{otherwise} \end{cases}.$$

Note that $k(S_1, S_2)$ is just $|S_1 \cap S_2|$, that is, the kernel (similarity) between both sets is given by the number of elements they have in common. Consequently, this suggests that the similarity between two structured objects can be measured by weighting those substructures they have in common. However, given that the number of these shared substructures can be enormous, a trade-off between expressiveness and efficiency is desired [141]. Thus, we need to specify which common substructures must be taken into account as well as the way of weighting them. We illustrate this by means of the

example reported in [103]. Given the words $w_1 = car$ and $w_2 = card$, by intuition we decide that the similarity between $w_1$ and $w_2$ is expressed in terms of their common, not necessarily continuous, 2-length substrings: $ca$, $cr$ and $ar$. Now, we have to weight them. To do this, each word $w_i$ is projected over the common substrings in such a way that each projection encodes both how many times and the different ways every common substring appears in the word. The resulting projection is expressed by means of a tuple of polynomials. Denoting this projection by $\phi$, then $\phi(w_1) = (\lambda^2, \lambda^3, \lambda^2)$ where, for instance, regarding the component $\lambda^3$ referring to $cr$, we can read that $cr$ appears only once in $w_1$ because the coefficient of the polynomial is 1. Additionally, the exponent means that there is a letter between $cr$ in $w_1$, because 3 is equal to the length of the substring plus the number of gaps this substring has in $w_1$. The same result is obtained for $w_2$. Finally, the parameter $\lambda$ is substituted by a value in $[0, 1]$ and the inner product of both tuples computed. This kernel for strings was proposed for text classification and can be implemented in polynomial time without explicitly computing the projection $\phi$. For further information about kernels on structured data one can read the survey [63].

We have pointed out that kernels can be viewed as similarity functions over objects, but unfortunately this intuition disappears for highly-complex kernel definitions. A nice property as for kernels is that the underlying similarity function can be recovered when transforming kernels into a pseudo-distance function. This is automatically done by setting $d(x, y) = (k(x, x)^2 + k(y, y^2 - 2k(x, y))^{1/2}$ and i.e. the lower $d$ the nearer (more similar) $x$ and $y$ are. Although kernel-based methods has been successfully applied over real-world data, its major drawback is, as distance-based methods, the lack of comprehensibility of the model learnt. It does not matter how data is represented, since the model is not defined inside the representation space but inside a (possibly infinite) abstract feature space where the examples are mapped.

## 3.5 Symbolic-based vs. Similarity-based Learning over Structured Data

As said in the introduction of this thesis, an interesting contribution would be to set a framework which allow us to develop symbolic learners which work for any data representation. In other words, general learners which can be parametrised with any particular representation language. Observe that all the symbolic learners explained in in this and the previous chapter are specifically designed depending on the data representation adopted.

Although first-order logic constitutes an elegant and powerful framework for symbolic knowledge representation, this cannot be used for the so-mentioned purpose since this has some serious weaknesses in order to deal with common structured data types such as numeric data, sets, lists, etc., as we illustrate next.

- term-based: problems related to pattern extraction with term-based descriptions have been mentioned in Section 3.2. In any case, we review it again in a more explicit and detailed way. Given the lists $l_1 = abcde$ and $l_2 = bde$, we want to represent them using a term-based approach. We need first to define a specific-purpose term to build lists, for instance, $l()$. This term must be binary in order to recursively represent lists with more than one element. Thus, we have:

  $$l_1 = l(a, l(b, l(c, l(d, l(e, [])))))), \;\; l_2 = l(b, l(d, l(e, [])))$$

  Note that this representation forces us to introduce another term (denoted by []) which simply stands for "empty list". Now, suppose we want to extract a common pattern from $l_1$ and $l_2$.

If we apply $lgg$ then we have a pattern like $l(X_1, l(X_2, l(X_3, X_4))$ when in fact $l_2$ is a sublist of $l_1$!. Furthermore, this condition ($l_2$ is a sublist of $l_1$) cannot be expressed using a term of the form $l(something)$. As a result, it could never be obtained by means of $lgg$. That is, one could think about a pattern such as $l(X_1, l(b, l(X_2, l(d, e))))$, but according to the order stated by $\theta$-subsumption this pattern is not more general than $l_2$! We could try another more expressive logic formula, for instance, clauses built up from special predicates such as *member*, to express sublist condition. In this way, we could write a pattern like this

$$\leftarrow member(b, L), member(d, L), member(e, L),$$

This pattern is satisfied by both $l_1$ and $l_2$ but offers little information about how the elements $b$, $d$ and $e$ are placed in the lists. That is to say, questions referring to the order of these symbols in the lists (e.g. $b$ is placed before $d$ and $d$ before $e$), or to which contiguous subsequences exist (e.g. in our case $b$ and $de$), etc. To capture these latter traits, a background knowledge containing predicates such as *previous*, *contiguous_sequence*, etc. would be needed. Either only the required predicates are given (which would be like given most of the solution beforehand) or many predicates must be included and the search space would be larger and the learning process would be slower and more difficult. For instance if we want to express the property of "all the lists having the subsequence *bde*", this would be as follows

$$\leftarrow \quad member(b, L)\ member(d, L)\ member(e, L)\ previous(b, d, L)\ previous(d, e, L)$$

It we want to restrict this latter property a bit more and say that "all the lists having the subsequence *bde* such that the subsequence *de* is contiguous" then we would have to write,

$$\begin{array}{ll} \leftarrow & member(b, L)\ member(d, L)\ member(e, L)\ previous(b, d, L), \\ & previous(d, e, L)\ contiguous\_subsequence(l(d, l(e, !)), L) \end{array}$$

- flattened: lists, trees, and graphs can be expressed in a flattened form as well. To illustrate this, we use the approach introduced in [78]. Let us change the lists above by other simpler ones, for instance, $l'_1 = abc$ and $l'_2 = ac$. These lists $l'_1$ and $l'_2$ are described as follows:

$$\begin{array}{lcll} l'_1 & \equiv & \leftarrow & a(a_1), b(a_2), c(a_3), r(a_1, a_2), r(a_2, a_3) \\ l'_2 & \equiv & \leftarrow & a(b_1), c(b_2), r(b_1, b_2) \end{array}$$

where constants identify elements in a list, unary predicates $a(\cdot), b(\cdot), \ldots$ label the elements and the binary predicate $r(\cdot, \cdot)$ states an order relation among the elements in a list. For pattern extraction, we could compute the $lgg$ between both clauses. The result is:

$$\leftarrow a(X), c(Y), r(X, Z), r(Z', Y)$$

Note that this pattern is not equivalent to say "all the lists having the subsequence *ac*". For this, it is enough to see that the list $l' = dcad$ is covered by the pattern above and does not satisfy the desired property. Basically,

$$l' \quad \equiv \quad d(c_1), c(c_2), a(c_3), d(c_4), r(c_1, c_2), r(c_2, c_3), r(c_3, c_4)$$

and just consider the substitution $\theta = \{X/c_3, Y/c_2, Z/c_4, Z'/c_1\}$. The property we want to express would correspond to

$$\leftarrow a(X), c(Y), r(X, Y)$$

but this latter pattern would never be inferred by means of $lgg$ over $l_1$ and $l_2$. We could think of using other generalisation operators such as the $V$-operator, but the high degree of indeterminism associated with this operator would make this task rather difficult.

The drawbacks above can be explained as follows. Given that lists (and in the same way, other data structures such as trees or graphs) have a complex and a little intuitive representation by means of first-order formulas, this implies that pattern over lists (trees, graphs, etc.) have also a complex representation. Consequently, even simple patterns are difficult to be obtained.

Now, the question is how to deal, in a general way, with specific (and appropiate) structured data representations?

As seen, most of the similarity-based methods have a learning schema which is independent on the data representation. But the downside is that no symbolic model is obtained in this case. This is so because a similarity function encodes in a single number interesting properties or patterns that data can share. For instance, properly weighting common sublists in the case of lists, common subgraphs in the case of graphs, and so on.

If we were able to come up with a general framework which allows us to recover the properties used by the similarity functions then we could develop methods to extract symbolic patterns from any data representation as long as a similarity function is defined over it.

This idea is developed in the next chapters where we present such a framework when data (structured or not) is embedded in a metric space. The only restriction is then that similarity functions must be distances. To this end and as we will see, our proposal is somehow inspired on ILP in that it introduces some definitions extending key ILP concepts such as generalisation operator and minimal generalisation.

# Part II

# Definition of the Framework

# Chapter 4

# Distance-based Generalisation Operators

This chapter focuses on the formalisation of the concept of distance-based generalisation operators. This concept is the keystone of our framework and refers to those operators which are able to generalise elements embedded in a metric space in a consistent way w.r.t. the distance function employed.

## 4.1 The Gap between Distance and Generalisation

As said in previous chapters, this work is motivated by the observation that two of the most important concepts in machine learning, namely distance and generalisation, do not work well together. Before further discussing on this divergence, we will briefly review the use of distance and generalisation in machine learning.

Distances in machine learning are used profusely since they are an adequate formalisation of dissimilarity and there are many distances available for every datatype: numeric data (e.g. Euclidean-like metrics [2]), symbolic data (e.g. Value Difference Metric (VDM) and variants [157]), strings/lists (e.g. edit distances), sets, graphs, ... [65]. Many (but not all) distance-based methods are usually associated to instance-based methods (e.g. methods based on the nearest-neighbour rule [22]) or the so-called "local models" techniques [3] (e.g. k-means, self-organising maps, locally weighted regression, etc.).

The great advantage of distance-based methods is that the same algorithm can be applied to whatever kind of datatype provided that there is a distance defined for that datatype. In this sense, we can cluster or classify integers, tuples of integers, strings, text documents, etc. However, the main disadvantage of distance-based methods is that the pattern (if any) is defined in terms of the distance to one or more prototypes or centroids. This makes distance-based methods too cryptical for many applications. This is especially the case for problems with complex data (graphs, strings, trees). We extract no knowledge of "patterns" such as "elements which are near to graph $x$ and graph $y$".

In order to obtain more useful patterns, there exist other methods that rely on the idea of generalisation. Basically, this consists of expressing regularities in data by means of patterns or

models. Although generalisation is also ubiquitous in machine learning (see e.g. [3, 113]), not every learning method that uses the concept of generalisation is known as a generalisation-based method (e.g. $SVM$). According to [113] (Chapter 2), generalisation-based methods define a lattice (or a direct graph) over a set of symbolic patterns/models (by symbolic we mean patterns close to natural language such as patterns based on logic, regular expressions, if-then-else rules, numerical intervals, etc.) and generalisation/specialisation operators from which the search through this lattice is performed. For instance, most decision tree and rule learners, ILP methods, grammar learning methods and conceptual clustering are generalisation-based methods.

Therefore, the great advantage of generalisation-based methods is that the result is expressed in a pattern language that is intuitive for the datatype at hand. The main disadvantage of using generalisation is that, although many algorithms share similar ideas, each algorithm has to be completely redesigned if the datatype is changed. For instance, generally speaking, grammar and first-order inference have a similar learning schema but they differ in the way the main ideas are put into practise.

From the short description above, the possibility of combining the best of both techniques arises. In the recent literature [27, 29, 69, 132, 150], we find approaches that combine distance-based (often instance-based) methods with model-based learners in order to minimise the problems of each technique by taking the best of both. However, this connection is made at a shallow level (e.g. running a model-based learner in post-processing after executing an instance-based learner), and the patterns that are obtained may have no relation with the underlying distance or similarity function used in the instance-based part.

By a connection between distance and generalisation, we mean that the way in which generalisation is defined takes the underlying distance into consideration (or at least the two are consistent). For instance, a first attempt in this line could be as follows: when we generalise a set of examples $E$, we expect that objects in the space which are close to the elements in $E$ will be inside the generalisation and objects which are far away will not be inside the generalisation. However, this idea has many possible interpretations, even in very simple metric spaces. E.g., given $E = \{2, 10\}$ in the metric space of integers and distance as the absolute difference. We could consider the generalisation $G = [1..3] \cup [9..11]$. Although this generalisation includes elements around 2 and 10, the generalisation $G$ appears not to be too much intuitive in the sense that some of the elements in between are outside of $G$. That is, it does not seem to be reasonable that if $G$ is a generalisation consistent with the distance, $G$ excludes part of the "segment of elements" which precisely make up the distance between the elements 2 and 10. Thus, we need to strength our initial proposal and this will be done by adding the notion of connection. By connection we mean that when a pattern generalises a pair of elements $e_1$ and $e_2$, we expect to go from one to other though similar (close) elements without living out of the generalisation. For instance, in the example above, we expect to go from 2 to 10 through the path $2 \to 3 \to \ldots \to 9 \to 10$. Or considering a more practical case, if we want to generalise a "black crow" and a "brown dove" we expect a "brown crow"/"black dove" to be included in the generalisation. This would mean that, inside the metric space, we can gradually move from a black crow to a brown dove through a brown crow (or a black dove).

However, while the notion of 'gradually' or 'close' is something inherent to the underlying metric space, we have that this is disregarded in the the generalisation-order space. A consequence of this is that most generalisation-operators and distances originate incompatible spaces. For instance, the pattern $*cc*$ for the strings $aaacc$ and $ccaaa$ excludes the shortest-distance transformation $aaacc \to aaac \to aaa \to caaa \to ccaaaa$. In fact, this transformation explains why the distance between $aaacc$ and $ccaaa$ is 4. Therefore, the intuitive idea behind the concept of distance-based generalisation is

that the elements in between (in terms of the distance) should be included in the generalisation. This will be done by transporting the triangle inequality of distances to generalisation operators.

Another reason for the gap between the concepts of distance and generalisation is that distances are always defined between two objects whereas generalisations can (and must) be defined over a set of an arbitrary number of elements. This rises the question about how a generalisation can explain the distances between the elements of a set when the number of elements in this set is greater than two. For instance, if we have the elements *abc*, *cab* and *ac*, which pattern $*c*$ or $*a*$ is compatible with the distance among these elements?

Finally, another important "disagreement" between distance and generalisation is the notion of fitting. In a metric space, the concepts of underfitting and overfitting are usually understood in terms of the attraction function and the distances to the reference, original or prototype elements (the closer the border is to these elements, the more fitting it is). In a generalisation-ordered space, the concept of underfitting and overfitting (and hence minimality) are understood in terms of coverage. If both spaces are divergent, one can see overfitting in one space while seeing underfitting in the other (or viceversa). For instance, consider the alphabet of symbols $\Sigma = \{a, b, c\}$, the metric space $X$ made up of all the 3-length sequences built from $\Sigma$ where $d$ is the Hamming distance such that $d(a, b) = d(a, c) = 5$ and $d(b, c) = 1$. Now, $E = \{abc, aba\}$ and let $p_1 = aV_1V_2$ and $p_2 = V_1bV_2$ be two patterns covering $E$. Regarding the symbolic description $p_1$ and $p_2$ look equally general since both patterns provide the same "amount of information" of $E$. That is, $p_1$ says that all the elements in $E$ have the symbol $a$ in common and $p_2$ says that all the elements in $E$ have the symbol $b$ in common. However, regarding the distance, we have that $p_1$ becomes more general than $p_2$ because $p_1$ covers a greater "area" than $p_2$. That is, the nearest elements to $E$ not covered by $p_1$ are *bbc* and *bba* which are both at distance 5. Taking $p_2$ the nearest uncovered elements are *acc* and *aca* which are both at distance 2.

Given the previous analysis of the relationship between distances and generalisation or, more properly, their deficient connection, there can be several ways to bridge this "gap". We base our proposal on three notions: the notion of *reachability*, the notion of *intrinsicality* and the notion of *minimality*.

Our notion of reachability means that given two elements and their generalisation, we should be able to reach both elements from each other by making small "steps" to other elements which must also be in the generalisation. The concept of short step must be understood in the sense of the distance. In the previous example about the integer numbers, we concluded that a generalisation as $G_1 = [2..10]$ of $E = \{2, 10\}$ was specially preferable to $G_2 = [2..3] \cup [9, 10]$ because a path of close elements connecting them, namely $\{3, 4, 5, 6, 7, 8, 9\}$, is included in $G_1$ but not in $G_2$. Or for the pair of sequences $E = \{aaacc, ccaaa\}$, the pattern $*cc*$ is worse than the pattern $*aa*$, since the latter connects the sequences through a smoother path $aaacc \to aaac \to aaa \to caaa \to ccaaaa$ than the former which only permits $aaacc \to cc \to ccaaa$. Note that in a continuous metric space this property can be understood as having a curve that connects both objects inside the generalisation.

As can be observed, the previous notion forces any generalisation to include short-stepped paths which connect the elements that led to the generalisation, but this path can be curved and completely deviated to a short path. To avoid this, a second notion, which we call intrinsicality is introduced. A intrinsic generalisation means that given two objects, their generalisation should cover all the elements that are between them in terms of the underlying distance, i.e. the distance is intrinsic to the generalisation. This means that all the elements in the shortest path between two elements (in many metric spaces, the straight segment which connects them) must be included in the generalisation.

Figure 4.1 shows examples of both properties in $\mathbb{R}^2$: neither reachability nor intrinsicality are

satisfied in generalisation $G1$; in $G2$ the elements $A$ and $B$ are reachable but not by means of an intrinsic path because the elements in between are excluded; finally, reachability and intrinsicality are satisfied in $G3$ and $G4$.

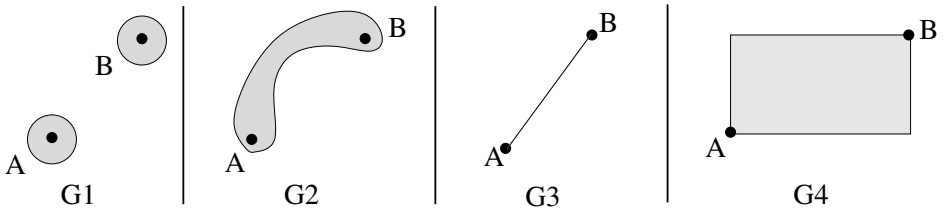


Figure 4.1: Generalising the elements $A$ and $B$.

Although the notions above are informal, it seems clear that for two elements and continuous spaces, the condition of ensuring that every intermediate point is in the set (intrinsicality) implies the notion of reachability (since it defines a path from both elements).

However, for more than two elements, the relation between the two notions is not so clear. Here we again see that distances are binary functions while generalisations are not necessarily binary. Both the notion of reachability and intrinsicality for more than two elements can be understood in many ways, e.g. all the elements must be connected pairwise, all the elements must be connected in some way, etc.fig-curves2

The notion of intrinsicality is the one which allows a generalisation to explain the distance. Since distances are defined for two elements, we will impose the notion of intrinsicality only for some pair of elements. Consequently, for sets of more than two elements, the pairs of elements that will have to comply with the intrinsicality property will be set by a path or connection graph which we will call *nerve*. The nerve does not need to be the minimum spanning tree, although this could be a typical choice. The result is that we ensure that all the elements in the set are reachable from any of them by moving from one element to another through direct (intrinsical) paths.

Finally, the third notion we will introduce is related to the concept of minimality, which is understood not only in terms of coverage or the size of the set (i.e., semantic minimality) but also as the simplicity of the pattern (i.e., syntactic minimality).

Figure 4.2 (bottom) shows that some of the generalisations are more specific than others (semantics), and some are simpler than others (syntax). We have $G4$, which is a simple generalisation of the elements $A$, $B$ and $C$; $G5$ and $G6$ are two less simple but more specific and finally, $G7$ is a very simple and very specific generalisation. The notion of minimality that we will work with combines both of them and, additionally, bases the semantic part on the underlying distance as well.

## 4.2 Distance-based Generalisation Operators

The above discussion on the three notions (reachability, intrinsicality and minimality) has been made at an informal level in order to clarify the choice of the properties that we will establish to shape the notion of distance-based generalisations and minimal distance-based generalisations. In the rest of this section, we switch to a more formal level and we define precisely what a generalisation is and how the notions of local intrinsicality (through the triangle inequality) and the notion of global

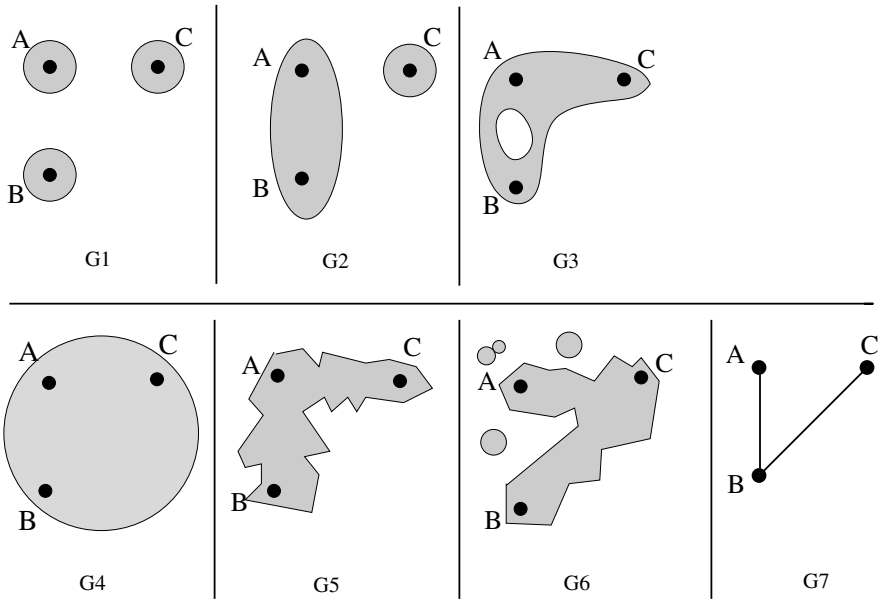


Figure 4.2: Generalising the elements $E = \{A, B, C\}$. **(Top)** Elements in $E$ are not reachable through a path of segments in generalisations $G1$, $G2$ and $G3$. **(Bottom)** For any two elements in $E$, generalisations include a path of segments connecting them.

reachability (through the notion of nerve) define the distance-based generalisation operator. We leave the notion of minimality for the following section.

Let us start with the definition of generalisation. Intuitively, a generalisation of a finite set of elements $E$ in a metric space $(X, d)$ could be extensionally defined as any set that contains $E$. But, from a predictive/explanatory point of view, a generalisation that is just expressed as a set of elements (e.g. $\{2, 4, 6, \ldots\}$) is much less useful than a generalisation that has an associated pattern (e.g. $\{x : odd(x)\}$). That is, a pattern $p \in \mathcal{L}$ can be considered as an intensional (and hopefully "comprehensible") manner of representing in a pattern language $\mathcal{L}$ a set of elements of $X$. We say that an element $x \in X$ is covered by a pattern $p$, if $x \in Set(p)$. In the same way, two patterns $p_1$ and $p_2$ overlap if $Set(p_1) \cap Set(p_2) \neq \emptyset$.

For instance, given the strings $abb$ and $abc$, and the regular pattern $ab*$, then $Set(ab*) = \{ab, abc, aba, abb, abaa, ...\}$, and we say that $ab*$ covers the elements $abb$ and $abc$ because $abb \in Set(ab*)$ and $abc \in Set(ab*)$. We can extend the well-known operations for sets to patterns. For example, we can say that a pattern $p_1$ is included in a pattern $p_2$ if $Set(p_1) \subseteq Set(p_2)$, or we can say that an element $x \in X$ belongs to the pattern $p$ (or is covered by $p$), if $x \in Set(p)$. Note that, one given set can be denoted or covered by several patterns. Furthermore, depending on our pattern language $\mathcal{L}$, some sets can be expressed as generalisations but others cannot. For instance, if our family of patterns in the metric space $\mathcal{R}^2$ is made of all the possible squares of size $1 \times 1$, the concept of a $2 \times 2$ square can not be expressed. For this reason, $\mathcal{L}$ will be defined according to the problem to be solved, and very specially, according to the kind of patterns that the user can understand.

Note that if there is no representation bias, $\mathcal{L}$ can even be defined to exhaustively cover $2^X$.

It can also happen that the pattern language is comprehensible for the user but not expressive enough for the problem at hand. Given a pattern language $\mathcal{L}$, its expressiveness can always be improved by combining patterns via logical operators. Pattern disjunction becomes quite useful in this sense, since we can express sentences such as "elements belonging to a pattern $p_1$ or a pattern $p_2$". In this work, pattern disjunction is denoted by the symbol + and so the expression $p_1 + p_2$ represents the set $Set(p_1) \cup Set(p_2)$. For simplicity, when a variable number of patterns $p_1, \ldots, p_n$ is used to define the pattern $p = p_1 + \ldots + p_n$, $p$ will be expressed as $p = \sum_{i=1}^{n} p_i$. In what follows, sometimes we will refer to terms $p_i$ as subpatterns.

**Definition 3. (Generalisation operator)** *Let $X$ be a space of elements and let $\mathcal{L}$ be a pattern language. For every finite set $E$ of elements in $X$, a generalisation operator $\Delta$ is a function such that $\Delta(E) = p$ where $p \in \mathcal{L}$ and $E \subset Set(p)$. If $E$ is restricted to sets of two elements $\{e_1, e_2\}$, then $\Delta$ is a binary generalisation operator and the mapping is written as $\Delta(e_1, e_2)$.*

Therefore, a generalisation operator simply maps sets of elements $E$ into patterns representing sets. Note that this definition says nothing about the nature of the resulting pattern, but that it must cover the original elements. Regarding Figure 4.1, the generalisation of two elements belonging to $\mathcal{R}^2$ could be something as simple as a straight line ($G3$) or something less intuitive such as $G2$. At this point, the most interesting aspect of this figure, as we also discussed in the previous section, is that the generalisation of $A$ and $B$ expressed as a straight line somehow explains the value of $d(A, B)$ since this generalisation covers those elements that are exactly placed between $A$ and $B$.

**Definition 4. (Intermediate element wrt. a distance)** *Given a metric space $(X, d)$ and two elements $e_1, e_2 \in X$, we say that an element $e_3 \in X$ is between $e_1$ and $e_2$, or is an intermediate element w.r.t. $d$, if $d(e_1, e_2) = d(e_1, e_3) + d(e_3, e_2)$.*

Now, we introduce the definition of binary distance-based pattern and binary distance-based generalisation operator.

**Definition 5.** ***(Binary distance-based pattern and binary distance-based generalisation operator)*** *Let $(X, d)$ be a metric space, $\mathcal{L}$ a pattern language, and a set of elements $E = \{e_1, e_2\} \subset X$. We say that a pattern $p \in \mathcal{L}$ is a binary distance-based (db) pattern of $E$ if $p$ covers all the elements between $e_1$ and $e_2$. Additionally, we say that $\Delta$ is a binary distance-based generalisation (dbg) operator if $\Delta$ always computes a binary distance-based pattern.*

The above definition is a simple formalisation of the notion of intrinsicality. Interestingly, this formalisation works well for continuous and discrete spaces. An example illustrating these definitions is introduced next.

**Example 4.** *Let us suppose that the elements $A$ and $B$ in Figure 4.1 are in the metric space $(\mathbb{R}^2, d)$, where $d$ is the Euclidean distance and $A(1, 1)$ and $B(3, 4)$. The intermediate elements are those in the segment $\overline{AB}$. Therefore, as said, only patterns representing the generalisations $G3$ and $G4$ are db. However, if $d$ is now the Manhattan distance[1], the intermediate elements are placed in the rectangle delimited by $A$ and $B$. Therefore, $G4$ is db but not $G3$.*

For the case of more than two elements to be generalised, the concept of "nerve" of a set of elements $E$ is needed to define distance-based generalisations. This corresponds to the notion of reachability.

Informally, a nerve of $E$ is simply a connected[2] graph whose vertices are the elements belonging

[1] Given two points $A(a_1, a_2)$ and $B(b_1, b_2)$, the Manhattan distance is defined as $d(A, B) = |a_1 - b_1| + |a_2 - b_2|$.

[2] Here, the term connected refers to the well-known property for graphs.

to $E$. Observe that if $E = \{e_1, e_2\}$, the only possible nerve corresponds to a graph with a unique edge. Formally,

**Definition 6.** ***(Nerve function)*** *Let $(X, d)$ be a metric space and let $S_G$ be the set of undirected and connected graphs over $X$, a nerve function $N : 2^X \to S_G$ maps every finite set $E \subset 2^X$ into a graph $G \in S_G$, such that each element $e$ in $E$ is inequivocally represented by a vertex in $G$ and vice versa. We say the obtained graph $N(E)$ is a nerve of $E$.*

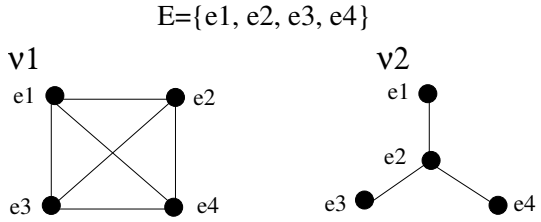


Figure 4.3: Some examples of nerves for the set $E = \{e_1, \ldots, e_4\}$. **(Left)** $\nu_1$ is a complete graph. **(Right)** $\nu_2$ is a 3-star graph.

**Definition 7.** ***(Skeleton)*** *Let $(X, d)$ be a metric space, $\mathcal{L}$ a pattern language, a set $E \subseteq X$, and $\nu$ a nerve of $E$. Then, the skeleton of $E$ wrt. $\nu$, denoted by skeleton($\nu$), is defined as a set which includes all the elements $z \in X$ between $x$ and $y$, for every $(x, y) \in \nu$.*

Consequently, we look for generalisations that include the skeleton. From here, we can define the notion of distance-based pattern wrt. a nerve.

**Definition 8.** ***(Distance-based pattern and distance-based pattern wrt. a nerve $\nu$)*** *Let $(X, d)$ be a metric space, $\mathcal{L}$ a pattern language, $E$ a finite set of examples. A pattern $p$ is a distance-based pattern of $E$ if there exists a nerve $\nu$ of $E$ such that skeleton($\nu$) $\subset$ Set($p$). If the nerve $\nu$ is known then we will say that $p$ is a distance-based pattern of $E$ wrt. $\nu$.*

And, from here, we have:

**Definition 9.** ***(Distance-based generalisation operator)*** *Let $(X, d)$ be a metric space and $\mathcal{L}$ a pattern language. Given a generalisation operator $\Delta$, we will say that $\Delta$ is a distance-based generalisation operator if for every $E \subseteq X$, $\Delta(E)$ is a distance-based pattern of $E$.*

The above definition can be characterised for one nerve function in particular.

**Definition 10.** ***(Distance-based generalisation operator wrt. a nerve function)*** *Let $(X, d)$ be a metric space and $\mathcal{L}$ a pattern language. Given a generalisation operator $\Delta$, we will say that $\Delta$ is a distance-based generalisation operator wrt. a nerve function $N$ if for every $E \subseteq X$ then $\Delta(E)$ is a distance-based pattern of $E$ wrt. $N(E)$.*

Some typical nerve functions are the ones which return a complete graph, a radial/star graph around a vertex (see Figure 4.3), the Minimum Spanning Tree nerve (the graph which connects groups to their nearest neighbour in a different group until all the elements are connected) and the

Travelling Salesman nerve (the graph which gives the smallest sum of distances). Only the latter two have to be derived using the underlying distance. Obviously, not all these nerves can be computed efficiently.

In some cases, we will be able to show that a generalisation operator is *db* for any nerve. In other cases, we will only require a nerve such that we can prove that the generalisation operator is *db*. In relation to this, it is possible to define $n$-ary operators from binary operators when the nerve function is given beforehand.

**Proposition 1.** *Let $\mathcal{L}$ be a pattern language endowed with the operation $+$ and let $\Delta^b$ be a binary distance-based operator in $\mathcal{L}$. Given a finite set of elements $E$ and a nerve function $N$. Then, the generalisation operator $\Delta_N(E)$ defined as*

$$\Delta_N(E) = \sum_{\forall(e_1,e_j)\in N(E)} \Delta^b(e_i, e_j)$$

*is distance-based w.r.t. $N$.*

*Proof.* It follows from the definition of distance-based operator. □

Let us see an example about *db* generalisations over more than two elements.

**Example 5.** *None of the generalisation at the top part of Figure 4.2 admits a nerve w.r.t. the generalisation is distance-based. As for the generalisation at the bottom part, $G4$ and $G5$ are distance-based wrt. the four nerves depicted in Figure 4.4; Note that there are elements "in the middle" of generalisation $G4$ which are excluded, but this central area does not exclude or avoid the straight paths between the three elements. Generalisation $G5$ is distance-based wrt. $N2$ only. Generalisation $G6$ is distance-based wrt. $N3$ only.*


Figure 4.4: Four different nerves.

# Chapter 5

# Minimal Distance-based Generalisation Operators

The notion of minimal generalisation is an important concept in order to avoid problems related to overfitting and overgeneralising in the learnt patterns. In this chapter we present a definition of minimality for *dbg* operators. Unlike other possible definitions of minimal or least generalisation used in machine learning (e.g. in ILP) based on the notion of subsumission, the one we propose here follows the MDL/MML principle and measures both the complexity and the fitness of the patterns, being the fitness of the patterns not expressed by subsumission but in terms of the underlying distance.

## 5.1 Minimal Distance-based Generalisations

Given the definition of *dbg* operator in the previous section, we can now guarantee that a pattern obtained by a *dbg* operator from a set of elements ensures that all the original elements are reachable inside the pattern through intrinsic (direct) paths. However, as we discussed, the generalisation can contain many other paths. More precisely, Figure 4.2 shows that a generalisation can be arbitrarily general ($G4$) and/or arbitrarily complex and whimsical ($G3$ and $G4$).

Hence, in general, we can proceed as follows. On the one hand, once we know that all the generalisations will be distance-based, we can ignore the issues of generality and complexity formally and let each specific machine-learning algorithm address the issues of underfitting, overfitting and model complexity. On the other hand, another possibility is to define a general criterion to determine the fitness and/or the simplicity of a generalisation. Given this criterion, we could set an order relation and compute the minimal (hence optimal) generalisation. This typical approach has given interesting generalisation operators in the past. The *lgg* [129] in the field of (ILP) [117] is the best example of this. However, Plotkin's *lgg* operator works well because the generality relation for atoms ensures a minimum and also because the pattern language is restricted to atoms and it cannot become arbitrarily (or infinitely) complex from finite evidence.

However, for other representation languages and pattern languages, this may not be the case. Let us see how to define a general criterion that makes it possible to obtain a minimal generalisation in general. Let us start with the concept of generality/fitness, which will also leads us to the concept

of simplicity.

First, consider that we establish a criterion to determine, given two patterns computed by two different dbg operators $\Delta$ and $\Delta'$, which one is less general. Then, the least general distance-based generalisation operator $\Delta$ will be that one such that for every set $E$ and for every distance-based operator $\Delta'$, the pattern $\Delta(E)$ is less general than $\Delta'(E)$. The least general $\Delta$ might not be unique. Instead, we will use the term minimal.

In order to formalise this idea, we could utilise the inclusion operation between sets ($\subset$) as a "mechanism" to compare how general two generalisations are. That is, a pattern $p$ for $E$ computed by a *dbg* operator $\Delta$ would be less general than a pattern $p'$ for $E$ computed by $\Delta'$, if $Set(\Delta(E)) \subset Set(\Delta'(E))$. However, this has several weakness.

- The inclusion operator between sets ignores the underlying distance: in most cases, generalisations are not comparable via the inclusion operation since, given $\Delta(E) = p_1$ and $\Delta'(E) = p_2$, it is possible that neither $Set(p_1) \subseteq Set(p_2)$ nor vice versa. However, it is also possible that for these two non-comparable generalisations of $E$, $Set(p_1)$ fits $E$ better than $Set(p_2)$ does as Figure 5.1 shows. Observe that the two patterns are not comparable via the inclusion operator. However, taking into consideration the distances from the elements in $E$ to the border[1] of $Set(p_1)$ and $Set(p_2)$ (the arrows labelled as $d_{ij}$) respectively, we can tell that $p_1$ fits $E$ better than $p_2$ does.

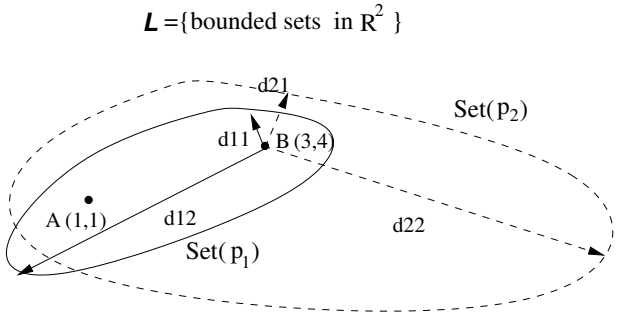


Figure 5.1: Two patterns $p_1$ and $p_2$ generalising $E = \{A, B\}$. The pattern $p_1$ fits $E$ in a better way. However, the two patterns are not comparable via the inclusion operator.

- The inclusion operator ignores the complexity of the pattern: this would lead to overfitting or, in some cases, to the unexistence of a minimum. For instance, consider $\mathbb{R}^2$ with the Euclidean distance and the pattern language $\mathcal{L}$ as the set of finite unions of rectangles in $\mathbb{R}^2$. Then, the generalisation of $n$ points belonging to $\mathbb{R}^2$ could be something as simple as a rectangle containing them (see left Figure 5.2) or, in some contexts, it could be preferable to obtain a more elaborated generalisations. as the one one depicted on the right of Figure 5.2.

  However, we must take into consideration that the more expressive a pattern language is, the higher the chance of overfitting. For instance, if we look at Figure 5.3, the pattern $p_0$, can be a reasonable *db* generalisation of the elements $A$ and $B$. But a pattern such as $p_1$ is also

[1]Let $(X, d)$ be a metric space and $B(e, r)$ the closed ball centred on $e$ with radius $r$. We will say that an element $e$ belonging to set $A \subseteq X$ is a border point, if for every $\epsilon > 0$, $B(e, \epsilon)$ is not totally included in $A$. According to standard notation, the border of a set $A$ will be denoted by $\partial A$.

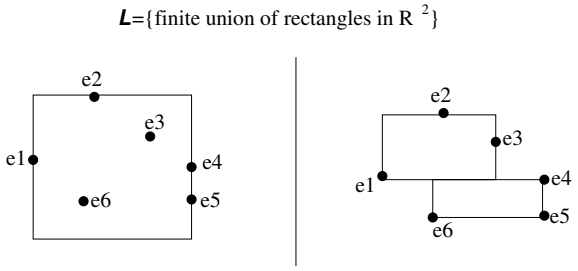


Figure 5.2: **(Left)** A simple generalisation of the set of elements $E = \{e_1, \ldots, e_6\}$. **(Right)** A more elaborated generalisation of $E$.

*db* and $Set(p_1) \subset Set(p_0)$, and the pattern $p_2$ is *db* as well and $Set(p_2) \subset (p_1)$. However, the complexity of $p_1$ and $p_2$ is unnecessary for the data set at hand. Furthermore, note that, in this case the minimum will never exist because for any *db* pattern we can always find another *db* pattern included in the previous one.

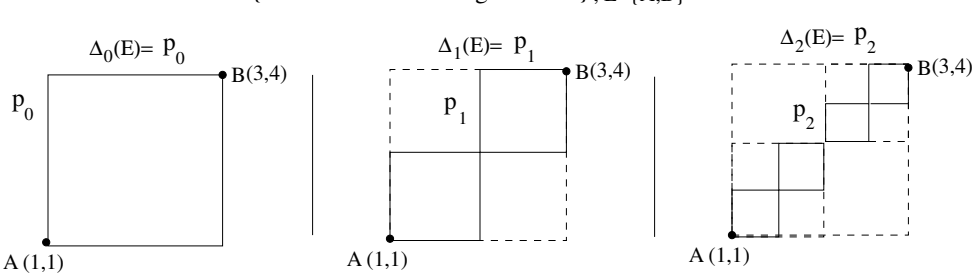


Figure 5.3: Generalising $E = \{A(1,1), B(3,4)\}$ by means of $\Delta_i$.

Therefore, these two drawbacks suggest the definition of the notion of optimality based on a more abstract principle. An abstract well-founded and widely-used principle that connects the notions of fitness and simplicity is the well-known $MDL/MML$ principle [144, 165]. This principle says that a high level of "complexity" of a pattern is reasonable only if a sufficient number of examples justify it. In other words, any simpler model would fit the data much worse. In the example above, a pattern made of thousand small rectangles becomes too complex, even if it fits the evidence $E$ really well. In general, the optimality of a model is evaluated by a trade-off between the fitness and the simplicity of the model. Our proposal reformulates this principle, especially the fitness part, in terms of the underlying distance (thus being the first distance-based formulation of the $MDL/MML$ principle). This results in a more generic quality criterion that can be applied over any datatype embedded in a metric space.

For this purpose, we introduce a cost function, denoted by $k(E, p)$, which takes into account both the complexity of the pattern $p$ and how well the pattern fits $E$. As Figure 5.3 indicates, the cost function $k(E, p)$ should be able to discriminate those unnecessary complicated patterns. Hence, the optimality of a generalisation will be defined in terms of a cost function instead of the inclusion between sets.

**Definition 11. (Cost function)** *Let $(X, d)$ and $\mathcal{L}$ be a metric space and a pattern language, respectively. We will say that the mapping $k : 2^X \times \mathcal{L} \to \mathbb{R}^+ \cup \{0\}$ is a cost function, if for every $E \in 2^X$, where $E$ is finite and every pattern $p \in \mathcal{L}$ such that $E \subset Set(p)$, if $Set(p) \neq X$ then there exits a constant $c > 0$ such that $k(E, p) < c$.*

As usual in MDL/MML approaches, most of the $k(E, p)$ functions that we are going to use will be expressed as the sum of a complexity (syntactic) function $c(p)$ (which measures how complicated the pattern is) and a fitness function $c(E|p)$ (which measures how the pattern fits the data $E$). The most novel and remarkable point in this approach is that this latter function will be expressed in terms of the distance employed.

The function $c(p)$ follows the tradition of a practical MDL/MML principle. As $c(p)$ measures how complex a pattern is, this function will strongly depend on the sort of data and the pattern space $\mathcal{L}$ we are dealing with. For instance, if the generalisation of two real numbers is a closed interval containing them, then a simple choice for $c(p)$ would be the length of the interval. If we consider atoms, its complexity could be given by the number of symbols in an atom. Of course, more sophisticated symbols can be found with a more genuine understanding of the MDL/MML principle (see Table 5.1).

| Sort of data | $\mathcal{L}$ | $c(p)$ | Example |
|---|---|---|---|
| Numerical | Closed intervals | Length of the interval | $c([a, b]) = \|a - b\|$ |
| Finite lists over an alphabet of symbols $\Sigma$ | Patterns built from an alphabet $\Sigma$ and a special alphabet $V$ of variables. | Number of items in the pattern | $c(V_0 a b V_1 b) = 5)$ |
| First order predicates | Herbrand base with variables | Number of different variables | $c(p(V_0, V_0, V_1, a)) = 2$ |
| | | Number of symbols | $c(p(V_0, V_0, V_1, a)) = 5$ |
| Any | Any | Constant function | $\forall p \in \mathcal{L}$, $c(p) = k$ |

Table 5.1: Some definitions of the function $c(p)$ for several sorts of data.

Now, let us see some definitions of $c(E|p)$. As stated above, all of them must be based on the underlying distance. In fact, all the definitions we present here are based on or inspired by the well-known concept of border of a set. Intuitively, if a pattern $p_1$ fits $E$ better than a pattern $p_2$, then the border of $p_1$ ($\partial p_1$) will somehow be nearer to $E$ than the border of $p_2$ ($\partial p_2$) (see Figure 5.4).

As the concept of border of a set is something intrinsic to metric spaces, the function $c(E|p)$ will be much more independent of the sort of data we are handling with than $c(p)$ is. Therefore, several general definitions of $c(E|p)$ can be given independently from the datatype as shown in Table 5.2.

In general, the functions $c(p)$ and $c(E|p)$ can be combined to obtain many possibilities for $k(E, p)$. Nonetheless, in many cases we will use cost functions such that the fitness part of the cost function (i.e $c(E|p)$) is not only defined by using distances but it is also consistent with the pristine notion of "more general than", in terms of the inclusion operator. The following definitions formalise cost functions of this kind. These definitions will be useful in some other sections in this work.

**Definition 12. (Inclusion-preserving semantic cost function)** *Let $c(\cdot|\cdot)$ be the semantic (fitness) part of a cost function. We say that $c$ is inclusion-preserving iff for every $E \subset X$ and for*

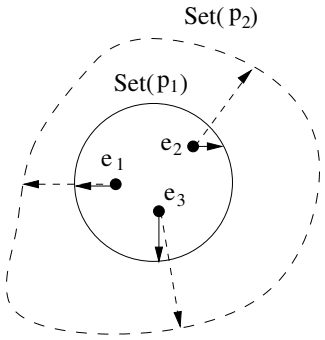


Figure 5.4: The pattern $p_1$ fits $E = \{e_1, e_2, e_3\}$ better than $p_2$, consequently, $\partial p_1$ is "nearer" to $E$ than $\partial p_2$.

| $\mathcal{L}$ | $c(E\|p)$ | Description |
|---|---|---|
| Any | $\sum_{\forall e \in E} r_e$<br>$r_e = \inf_{r \in R} B(e, r_e) \not\subset Set(p)$ | Infimum of uncovered elements<br>(top-left picture in Figure 5.5) |
| Any | $\sum_{\forall e \in E} r_e$<br>$r_e = \sup_{r \in R} B(e, r_e) \subset Set(p)$ | Supremum of covered elements<br>(top-right picture in Figure 5.5) |
| Any | $\sum_{\forall e \in E} \min_{e' \in \partial Set(p)} d(e, e')$ | Minimum to the border<br>(bottom-left picture in Figure 5.5) |
| $Set(p)$ is a bound set | $\sum_{\forall e \in E} \min_{e' \in \partial Set(p)} d(e, e')$<br>$+ \max_{e'' \in \partial Set(p)} d(e, e'')$ | Minimum and maximum to the border<br>(bottom-right picture in Figure 5.5) |

Table 5.2: Some definitions of the function $c(E|p)$.

*every two patterns $\Delta(E) = p$ and $\Delta'(E) = p'$, if $Set(p) \subset Set(p')$, then $c(E|p) \leq c(E|p')$.*

It is easy to see that all the functions described in Table 5.2 are inclusion-preserving. It is also trivial to see that if the syntactic part of the cost function $c(\cdot)$ is constant, then if the semantic (fitness) part $c(\cdot|\cdot)$ is inclusion-preserving, the whole cost function will be inclusion-preserving.

Now, we can introduce the definition of minimal distance-based generalisation operator and minimal distance-based generalisation operator relative to one nerve function.

**Definition 13. (Minimal distance-based generalisation operator and minimal distance-based generalisation operator relative to one nerve function $N$)** *Let $(X, d)$ and $N$ be a metric space and a nerve function, and let $\Delta$ be a distance-based generalisation operator wrt. $N$ defined in $X$ using a pattern language $\mathcal{L}$. Given a finite set of elements $E \subset X$ and a cost function $k$, we will say that $\Delta$ is a minimal distance-based generalisation operator (mdbg) for $k$ in $\mathcal{L}$ relative to $N$, if for every distance-based operator $\Delta'$ wrt. $N$,*

$$k(E, \Delta(E)) \leq k(E, \Delta'(E)), \text{for every finite set } E \subset X. \tag{5.1}$$

*In similar terms, we say that a dbg operator $\Delta$ wrt. a nerve function $N$ is a mdbg operator relative to $N$ if the expression (5.1) holds for every dbg operator $\Delta'$ wrt. $N$.*

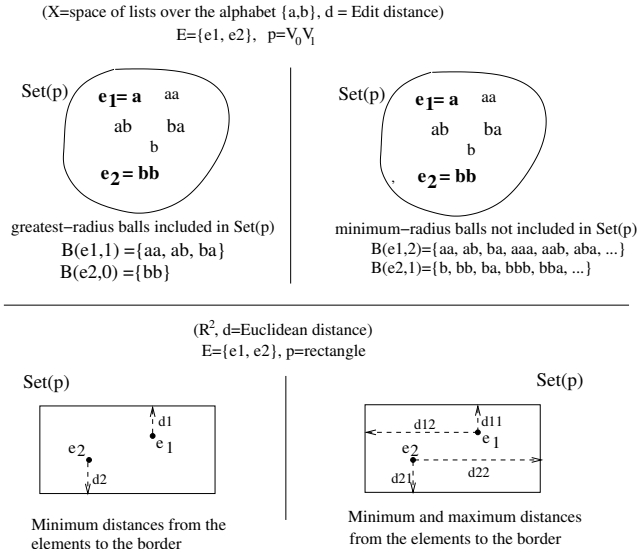


Figure 5.5: Pictures illustrating each definition of $c(E|p)$ collected in Table 5.2.

The previous definition says nothing about how to compute the *mdbg* operator, and as we will see later, this is far from being immediate. A way to proceed is to first try to simplify the optimisation problem as much as possible in order to find the *mdbg* operator. A first attempt might be the following:

**Definition 14. (Naive generalisation operator)** *Let $(X, d)$ be a metric space. The naive generalisation operator $\Delta^0$ is defined for every set $E \subset X$ as follows:*

$$\Delta^0(E) = \underset{\forall p \in \mathcal{L}: E \subset Set(p)}{\operatorname{argmin}} \; k(E, p)$$

The naive generalisation operator returns the simplest pattern (in terms of $k$) which covers the evidence. The advantage is that if $k$ is inclusion-preserving, $\Delta^0$ can be computed easily in most pattern languages $\mathcal{L}$. However, we cannot ensure beforehand that the resulting generalisation is *db*.

Another attempt is the following.

**Definition 15. (Skeleton generalisation operator w.r.t. a nerve function $N$)** *Let $(X, d)$ be a metric space and $N$ a nerve function. The skeleton generalisation operator $\bar{\Delta}_N$ is defined for every set $E \subset X$ as follows:*

$$\bar{\Delta}_N(E) = \underset{\forall p \in \mathcal{L}: skeleton(N(E)) = Set(p)}{\operatorname{argmin}} \; k(E, p)$$

which means the simplest pattern that covers the skeleton of the evidence (given a nerve) and nothing more. Clearly, it is a *dbg* operator because it includes the skeleton, but it might not exist because it cannot be expressed. And now, we can obtain the following specific but positive result:

**Proposition 2.** *Let $(X, d)$ be a metric space, a set $E \subset X$, $N$ a nerve function and $\mathcal{L}$ a pattern language that can express any skeleton$(N(E))$. If $k$ is inclusion-preserving and $\bar{\Delta}_N$ exists, then $\bar{\Delta}_N$ is a mdbg operator relative to $N$.*

*Proof.* Since $\bar{\Delta}_N$ exists, we can set $p = \bar{\Delta}_N(E)$. For any other *db* pattern $p'$ of $E$, by definition of $\bar{\Delta}_N$, we necessarily have $Set(p) \subset Set(p')$. Since $\bar{\Delta}_N$ is *db* and $k$ is inclusion-preserving then $\bar{\Delta}_N$ is a *mdbg* operator wrt. $N$. □

For some datatypes (such as nominal, numerical or first-order objects as shown in the following sections), the previous techniques to obtain a minimal distance-based generalisation operator will be sufficient. However, this will not be applicable in many other cases, either because the pattern language is not expressive enough or because of the complexity of the datatypes.

The following chapters are devoted to defining *db* and *mdbg* operators for the most common data types. We start with simple nominal and numerical data and then move to more complicated data types such as sets, lists, first-order objects, graphs and tuples.

# Part III

# (Minimal) Distance-based Operators for Common Data Types and Distances

# Chapter 6

# Generalising Nominal and Numerical Data Lying in a Metric Space

In this chapter, we present (minimal) distance-based generalisation operators for nominal and numerical data embedded in metric spaces. As for nominal data, we consider two kind of distances, (the discrete distance and a distance induced from a partial order relation) with a pattern language defined over each metric space. As for numerical data, we work with the absolute difference and consider two different pattern languages.

## 6.1 Nominal Data

Given the discussion and definitions introduced in the previous section, it is time to apply the setting to several datatypes. We start with the simplest datatype, the nominal, categorical or enumeration datatype. Along with numerical datatypes and tuples, this is one of the main components of what is called propositional learning, i.e., flat data that is expressed in terms of attributes and instances. We will address numerical data and tuples in subsequent sections.

Nominal (categorical) attributes can express a set of possibilities (e.g. colours, qualities, genders, etc.). A Boolean datatype is just a special case where there are only two possibilities. Therefore, in general, the metric space here is composed of a set $X$ which is just a finite set of values and a distance $d$.

Although the datatype is very simple, many distances can be defined on it. Some of the most-commonly used distances are the discrete (overlapping) distance (which returns 1 when both values matches and 0 otherwise), the VDM (Value Difference Metric) distance [157] and many others. Distances over nominal data can also be defined from a relation order previously defined over the set of nominal data.

### 6.1.1 Extensional Pattern Language and Simple Cost Functions

In many applications, nominal attributes are used in patterns in the form of conditions such as "at1 = black" or "at1 $\neq$ black". All of them can be expressed as "at1 $\in$ S", where $S$ is any subset of $X$, since $X$ is finite. Usually, the pattern is just represented by the set $S$. So, formally, the pattern language $\mathcal{L}$ is defined by the set $2^X$.

Once the pattern language is fixed, we use two cost functions $k_1$ and $k_2$, with $k_1 = c_1(p) + c(E|p)$ (where $c_1(p) = 0$ and $c(E|p)$ is any inclusion-preserving function) and $k_2 = c_2(p) + c(E|p)$ where $c_2(p) = |Set(p)|$. Note that, for this pattern language, $k_2$ is inclusion-preserving as well.

Under the conditions above:

**Proposition 3.** *Given a set of values $X$, any distance $d$, any nerve function $N$, the pattern language $\mathcal{L} = 2^X$, and any of the cost functions $k_1$ and $k_2$, then $\bar{\Delta}_N$ exists and is the minimal distance-based generalisation operator relative to $N$.*

*Proof.* For every $E \subset X$, there is only one way to express $skeleton(N(E))$ then $\Delta^0_N$ exists. In addition, as $k_1$ and $k_2$ are inclusion-preserving and $skeleton(N(E))$ can be expressed in $\mathcal{L}$, then by Proposition 2, $\Delta^0_N$ is the *mdbg* operator relative to $N$. □

It is also easy to see that if the distance is the discrete distance (1 when both values matches and 0 otherwise), then $\bar{\Delta}_N = E$ and thus $\bar{\Delta}_N = \Delta^0$.

**Example 6.** *Let $X = \{vhigh, high, medium, low, vlow\}$ be an ordered set such that the distance between two elements is the absolute difference of their positions in the order. For instance, $d(vhigh, high) = 1$ and $d(vhigh, vlow) = 4$. Given $E = \{vhigh, medium, low\}$ and any possible nerve for these elements, the minimal distance-based pattern is the one computed by $\bar{\Delta}_N$, which, in this case, is $\bar{\Delta}_N(E) = \{vhigh, high, medium, low\}$. Note that $\Delta^0$, although minimal, is not distance-based.*

### 6.1.2 Hierarchical Pattern Language and a Simple Cost Function

The previous example defines a distance from a total order relation. However, we can work on cases where there is a partial order relation. For instance, consider the case where there is a hierarchy of elements, such that $x\,R\,y$ if $x$ is a $y$. For instance, $Fish\ R\ Vertebrate$ since a fish is a verbrate. Figure 6.1 shows a tree hierarchy where elements that are directly connected are at a distance 1 and the rest of the distances are just derived as the composition (sum) of distances of the shortest path that connects them. We call this distance "the distance induced by $R$".

The pattern language $\mathcal{L}$ can be defined as $\{is\_a(v)\}_{\forall v \in X}$ which means covering any element $w$ such that $w\ R\ v$. For instance, given the previous Figure 6.1, we can say that the pattern $is\_a(vertebrate)$ covers the elements Vertebrate, Mammal, Dog, Cat, Fish, Herring and Shark.

Once the pattern language is fixed, we use one cost function $k = c(p) + c(E|p)$ where $c(p) = 0$ (we only allow a single $is\_a$ expression and i.e. all patterns will have the same complexity) and $c(E|p)$ which is any inclusion-preserving cost function.

Under the conditions above:

**Proposition 4.** *Given a set of values $X$, a distance $d$ induced from a partial order relation over $X$, a nerve function $N$, the pattern language $\mathcal{L}$ and the cost function $k_1$. If $\Delta(E) = is\_a(u)$ is the minimal distance-based generalisation operator relative to $N$ for $(\mathcal{L}, k_1)$, where $u$ is the minimum upper bound of $E$.*

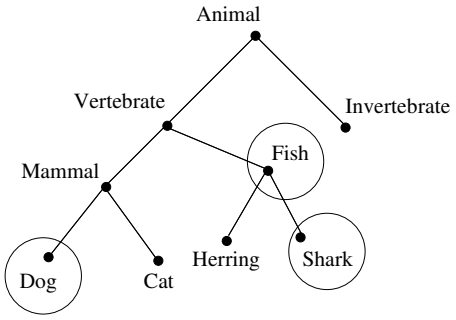


Figure 6.1: A set of nominal elements with a hierarchical relation from which we can infer a distance The circled values can be seen in a possible evidence $E$.

*Proof.* It is easy to show that $\Delta(E) = \bar{\Delta}_N(E)$ (see that $skeleton(N(E))) = Set(is\_a(u))$). Hence, any $skeleton(N(E))$ can be expressed in $\mathcal{L}$ and, given that $k_1$ is inclusion-preserving by Proposition 2 $\Delta$ is the *mdbg* relative to $N$. □

In the previous example, if we have the evidence $E = \{Dog, Fish, Shark\}$, the generalisation operator defined above would compute $is\_a(vertebrate)$, which clearly covers any intermediate element and is also minimal.

Many other kinds of languages, distances and cost functions might be analysed for nominal datatypes, but as the previous examples show, a *mdbg* operator is easy to find in these cases and it has an intuitive behaviour in terms of the notion of minimal generalisation and also in terms of the underlying distance. Let us now look at more interesting (and complex) datatypes.

## 6.2 Numerical Data

Numerical information is present in a wide variety of real-world features: temperature, income, weight, height, etc. For this reason, coping with numerical information becomes crucial in some contexts and, hence, some learning algorithms have been specially designed/upgraded for this purpose. Real numbers with the distance defined as the absolute difference of two real numbers ($d(e_i, e_j) = |e_i - e_j|$) is a metric space. In this section we show that the well-known concept of mathematical interval is a distance-based pattern for a set of numerical values.

### 6.2.1 Single Interval Pattern Language

The first pattern language $\mathcal{L}_0$ we consider is just the set of all the finite closed intervals in $\mathcal{R}$. We use two cost functions: $k_0$ is any inclusion-preserving cost function with $c_0(p) = 0$ and $k_1(E, p) = c_1(p) + c(E|p)$, where $c_2(p) = |b - a|$ being $p = [a, b]$. Note that $k_1$ is also inclusion-preserving.

The following proposition shows that intervals are minimal distance-based operators.

**Proposition 5.** *For every finite set of elements $E = \{e_1, \cdots, e_n\}$ such that $e_i \leq e_{i+1}$ $(i = 1, \ldots, n-1)$, then the mapping $\Delta(E) = [e_1, e_n]$ is a minimal distance-based generalisation operator for $(\mathcal{L}_0, k_0)$ and $(\mathcal{L}_0, k_1)$.*

*Proof.* It is easy to see that $skeleton(N(E)) = [e_1, e_n]$ can always be expressed in $\mathcal{L}_0$ and $\Delta(E) = \bar{\Delta}_N(E)$. Since both $k_0$ and $k_1$ are inclusion-preserving then by Proposition 2, $\Delta$ is the *mdbg* operator relative to $N$ for $(\mathcal{L}_0, k_0)$ and $(\mathcal{L}_0, k_1)$. □

It is easy to see that for this pattern language and cost function $k_1$, the naive generalisation operator does not exist except from an evidence with just one element. Consider, for instance, the following evidence $E = \{2, 3\}$. According to Definition 15, the naive generalisation operator should be the minimal pattern $p$ such that $Set(p) = \{2, 3\}$. However, any interval that covers $E$ (for instance the minimal interval $[2, 3]$) is different from $E$. Hence, there is no pattern $p$ such that $Set(p) = E$.

On the other hand, as mentioned in the previous chapters, given a metric space and a language pattern, not every generalisation operator that can be defined is distance-based. This is illustrated in the next subsection by using a more expressive pattern language than $\mathcal{L}_0$.

### 6.2.2 Multiple Interval Pattern Language

The possibility of working with several intervals, as the pattern language $\mathcal{L}_1$ allows us to do, is useful in tasks such as attribute discretisation, numerical partitions in decision tree learning and numerical clustering.

Let $\mathcal{L}_1$ be the pattern language of multiple intervals in which a pattern is expressed as the union of intervals. For instance, given the evidence $E = \{3, 4, 7, 8, 15, 16, 18, 20\}$, a pattern $p$ covering $E$ could be $[3, 8] \cup [15, 20]$. In this case, the trivial generalisation operator w.r.t. $\mathcal{L}_1$ and the cost function $k_1$ exists and is defined as $\Delta^0(E) = \bigcup_{e \in E}[e, e]$. For the previous evidence, $\Delta^0(E) = [3, 3] \cup [4, 4] \cup [7, 7] \cup [8, 8] \cup [15, 15] \cup [16, 16] \cup [18, 18] \cup [20, 20]$. Obviously, $\Delta^0$ is minimal w.r.t. $k_1$ since, for any other pattern $p$ covering $E$, $Set(p)$ contains $Set(\Delta^0(E))$ because $Set(\Delta^0(E)) = E$. Hence, as $k_1$ is inclusion-preserving, by Definition 12, $\Delta^0$ is minimal ($k_1(E, \Delta^0(E)) \leq k_1(E, p)$). However, this operator is not distance-based since for any nerve $\nu$ of $E$, $skeleton_\nu(E) \not\subset set(\Delta^0(E))$. This is an example of a minimal generalisation operator which is not distance-based. Moreover, $\Delta^0$ is not minimal w.r.t. many other cost functions. For instance, let $k_2 = c_2(p) + c_2(E|p)$ be the cost function where $c_2(p) = n^2$ with $n$ being the number of intervals in $p$ and, with $c_2(E|p)$ as the function number 3 in Table 5.2. Now, the trivial operator is not minimal w.r.t. $k_2$. For instance, continuing with the example, the pattern $p_1 = [3, 8] \cup [15, 20]$ has a lower cost than $\Delta^0(E)$ being $k_2(E, \Delta^0(E)) = 64 + 0 = 64$ and $k_2(E, p_1) = 4 + 5 = 9$. In fact, for this example and cost function, $p_1$ also has a lower cost than the pattern $p_2 = [3, 20]$ ($k_2(E, p_2) = 1 + 21 = 22$).

## 6.3 Discussion

In this chapter, the framework has been instantiated for nominal and numerical data embedded in metric spaces. Two distances have been analysed for nominal data, the discrete and the order-induced distance. In the first case, it is easy to show that the well-known union of sets is enough to obtain distance-based generalisation operators. This result does not hold when an order-induced distance is used, being necessary to adapt the pattern language to define distance-based operators in this new metric space.

As for numerical data, when the absolute difference is used as a metric function, a distance-based generalisation operator is the one returning an interval containing the elements to be generalised.

Therefore, these operators (union of sets and intervals) match the generalisation mechanisms implicit in most of the symbolic propositional learners (e.g. conceptual clustering, decision trees, etc.).

The whole generalisation of tuples of constants or numbers, important for symbolic propositional learning in metric spaces, is an issue which will be addressed in Chapter 10.

# Chapter 7

# Generalising Finite Sets of Elements Lying in a Metric Space

A set is a collection of elements without any particular order. This sort of data is quite common in structured learning. Namely, the concept of bag of words or key words in text or web mining, respectively, can be modelled by means of sets of words (or multi-sets if repeated words are taken into account). Besides, other data types such as trees or graphs can also be represented using sets. Examples of this can be found elsewhere, e.g., in the representation of phylogenetic trees derived from DNA data where trees are viewed as sets of labelled edges.

In this chapter, we will define distance-based generalisation operators for sets, and then we will study their minimality. As usual, we need to define a distance function as well as some pattern languages and some cost functions.

## 7.1 Distance, Pattern Languages and Cost Functions

The distance we will employ is the symmetric difference between two sets, i.e., $d(e_i, e_j) = |(e_i - e_j) \cup (e_j - e_i)|$ where $e_i$ and $e_j$ are any two finite sets whatever and $|\cdot|$ represents the cardinality of a set. In this way, the distance between two sets depends on the number of elements that they have in common. As mentioned in Chapter 3, this distance assumes that the 0-1 distance has previously been defined over the elements in sets and that, in some contexts, it might be more convenient to use another more sophisticated distance for sets such as the J. Ramon's distance [137]. This option is explored in next Chapter 9, for the moment, we will work with the symmetric difference.

With regard to patterns, two different pattern languages, $\mathcal{L}_0$ and $\mathcal{L}_1$, are considered. The patterns in $\mathcal{L}_0$ are sets from the alphabet $\Sigma = A \cup V$ where $A = \{a_1, a_2, a_3, \ldots\}$ is the set of ground symbols from which the sets to be generalised are built and $V = \{V_1, V_2, \ldots\}$ is a set of variables. An element $e$ is covered by a pattern $p$ if there exists a substitution $\sigma$ over the variables in $p$, such that $e \in \sigma p$. The function $Gr(p)$ defined over a pattern $p \in \mathcal{L}_0$ returns a set with the ground symbols in $p$. For example, if $p = \{a_1, a_2, V_1, V_2\}$ then $Gr(p) = \{a_1, a_2\}$. Consequently, no pattern except from $\emptyset$ covers the empty set. For example, $p = \{a_1, a_2, V_1, V_2, V_3\}$ covers $\{a_1, a_2, a_3\}$ with (among others) $\sigma = \{V_1/a_3, V_2/a_3, V_3/a_2\}$. The function $Gr(p)$ defined over a pattern $p_0 \in \mathcal{L}_0$ returns the set of all the ground symbols in $p$. For instance, if $p = \{a_1, a_2, V_1, V_2\}$ then $Gr(p) = \{a_1, a_2\}$. Finally, the

pattern language $\mathcal{L}_1$ is obtained from $\mathcal{L}_0$ by means of the operation + (see Chapter 4).

**Example 7.** *Given* $A = \{a_1, a_2, a_3, a_4\}$ *and the patterns* $p_1 = \{a_1, a_2, V_1\}$, $p_2 = \{a_3, V_1, V_2\}$ *and* $p_3 = p_1 + p_2$, *then,* $p_1$ *represents all the sets whose cardinality is* 2 *or* 3 *and contain the elements* $a_1$ *and* $a_2$. *That is,*

$$Set(p_1) = \{\{a_1, a_2\}, \{a_1, a_2, a_3\}, \{a_1, a_2, a_4\}\}$$

*In a similar way,* $p_2$ *represents all the sets whose cardinality ranges between* 1 *and* 3 *and contain the element* $a_3$.

$$Set(p_2) = \{\{a_3\}, \{a_1, a_3\}, \{a_2, a_3\}, \{a_3, a_4\}, \{a_2, a_3, a_4\}, \{a_1, a_2, a_3\}, \{a_1, a_3, a_4\}\}$$

*Finally,* $p_3$ *represents all the sets covered by the patterns* $p_1$ *and* $p_2$.

$$Set(p_3) = \{\{a_1, a_2\}, \{a_1, a_2, a_3\}, \{a_1, a_2, a_4\}, \{a_3\}, \{a_1, a_3\}, \{a_2, a_3\}, \ldots, \{a_1, a_2, a_3\}, \{a_1, a_3, a_4\}\}$$

If we are working with $\mathcal{L}_0$, and the sets to be generalised have no elements in common, then we can only compute patterns informing about the cardinality of these sets, that is, patterns which are made up of variable symbols only. This low expressivity shown by $\mathcal{L}_0$ can be improved if we use $\mathcal{L}_1$.

**Example 8.** *Let* $e_1 = \{a_1, a_2\}$, $e_2 = \{a_1, a_3\}$ *and* $e_3 = \{a_2, a_3\}$ *be three examples to be generalised. Using* $\mathcal{L}_0$, *the only possible solution is a pattern containing variables, for example,* $p = \{V_1, V_2\}$. *This pattern represents all the sets containing two or fewer symbols. On the contrary, using* $\mathcal{L}_1$, *we can write* $p = \{a_1, V_1\} + \{a_3, V_1\}$ *as a generalisation of* $\{e_1, e_2, e_3\}$. *That is, all the sets having one or two symbols and being one of them* $a_1$ *or* $a_3$.

Finally, two different cost functions are introduced. On the one hand, we use the inclusion-preserving cost function $k_0 = c(E|p)$ with $c(E|p)$ being the first function in Table 5.2. On the other hand, we define $k_1(E, p) = c_1(p) + c(E|p)$ where $c_1(p)$ is the number of both ground and variable symbols occurring in the pattern $p$. Note that when $\mathcal{L}_1$ is involved, a pattern generalising $E$ might be unnecessarily complicated. Hence, it is of interest to control both the complexity of the pattern and how well the pattern fits the data. This is what $k_1$ aims to do.

**Example 9.** *Let* $E = \{e_1, e_2, e_3\}$ *where* $e_1 = \{a_1, a_2\}$, $e_2 = \{a_2, a_3, a_4\}$ *and* $e_3 = \{a_3, a_4, a_5\}$ *are the elements to be generalised. Given the following patterns generalising* $E$*:*

$$\begin{array}{lcl} p_1 & = & \{V_1, V_2, V_3\} \\ p_2 & = & \{a_2, V_1, V_2\} + \{a_3, a_4, V_1\} \\ p_3 & = & \{a_1, V_1\} + \{a_2, V_1, V_2\} + \{a_5, V_1, V_2\} \end{array}$$

*Let us calculate* $k_0(E, \cdot)$ *and* $k_1(E, \cdot)$ *for all the patterns above. We denote by* $e'_1$, $e'_2$ *and* $e'_3$ *the nearest elements to* $e_1$, $e_2$ *and* $e_3$, *respectively, which are not covered by a given pattern. The table below shows the possible instances for* $e'_1$, $e'_2$ *and* $e'_3$ *w.r.t. a concrete pattern as well as the computation of the cost functions.*

The example above raises the question about how to compute the semantic function for this case. Things are immediate when we work with $\mathcal{L}_0$ since $c(\cdot|\cdot)$ can directly be computed from the pattern syntax. The intuition behind this is as follows. For a pattern $p \in \mathcal{L}_0$ having only variables, an element $e$ is not covered by $p$ if the cardinality of $e$ exceeds the number of variables in $p$. Back

| | | | | | | |
|---|---|---|---|---|---|---|
| $p_1$ | $e'_1 = \{a_1, a_2, a_3, a_4\}$<br>$e'_2 = e'_1$<br>$e'_3 = \{a_1, a_3, a_4, a_5\}$ | $c(\{e_1\|p_1\}) = d(e_1, e'_1) = 2$<br>$c(\{e_2\|p_1\}) = d(e_2, e'_2) = 1$<br>$c(\{e_3\|p_1\}) = d(e_3, e'_3) = 1$ | $c(E\|p_1) = 4$ | $k_0(E, p_1) = 4$ | $c_1(p_1) = 3$ | $k_1(E, p_1) = 7$ |
| $p_2$ | $e'_1 = \{a_1\}$<br>$e'_2 = \{a_1, a_2, a_3, a_4\}$<br>$e'_3 = \{a_3, a_5\}$ | $c(\{e_1\|p_2\}) = d(e_1, e'_1) = 1$<br>$c(\{e_2\|p_2\}) = d(e_2, e'_2) = 1$<br>$c(\{e_3\|p_2\}) = d(e_3, e'_3) = 1$ | $c(E\|p_2) = 3$ | $k_0(E, p_2) = 3$ | $c_1(p_2) = 6$ | $k_1(E, p_2) = 9$ |
| $p_3$ | $e'_1 = \{a_1, a_2, a_3, a_4\}$<br>$e'_2 = \{a_3, a_4\}$<br>$e'_3 = e'_2$ | $c(\{e_1\|p_3\}) = d(e_1, e'_1) = 2$<br>$c(\{e_2\|p_3\}) = d(e_2, e'_2) = 1$<br>$c(\{e_3\|p_3\}) = d(e_3, e'_3) = 1$ | $c(E\|p_3) = 4$ | $k_0(E, p_3) = 4$ | $c_1(p_3) = 8$ | $k_1(E, p_3) = 12$ |

Table 7.1: Computation of the cost functions $k_0$ and $k_1$ for patterns $p_1$, $p_2$ and $p_3$.

to our example, taking $e_1$ and $p_1$, one of the elements nearest to $e_1$ which is not covered by $p_1$ is $\{a_1, a_2, a_3, a_4\}$. For a pattern $p$ having ground symbols, if $e$ is not covered by $p$ then $e$ does not have all the ground symbols and/or its cardinality exceeds the cardinality permitted by $p$. In the example above, taking $e_1$ and $p_2$, $e'_1$ could be, among others, $\{a_1\}$. Put it in a more general way, for any pattern $p$ of the form $\{a_1, \ldots, a_n, V_1, \ldots, V_m\}$ $(n > 0)$ and for any element $e = \{a_1, \ldots, a_n, \ldots\}$ covered by this pattern, the element $e' = e - \{a_i\}$ $(1 \leq i \leq n)$ is not covered by $p$ and $d(e, e') = 1$, which is the lowest distance possible. Hence, $e'$ is the nearest element to $e$ not covered by $p$.

The previous comments lead to the expression below, which gives a quick way to calculate $c(\cdot|p)$.

**Proposition 6.** ***(Straight calculation for*** $c(E|p)$ ***in*** $\mathcal{L}_0$***)*** *Given the set of elements $E = \{e_1, \ldots, e_n\}$ and a pattern $p \in \mathcal{L}_0$ such that $E \subset Set(p)$. By decomposing $c(E|p) = \sum_{i=1}^{n} c(\{e_i\}|p)$, we can state that:*

$$c(\{e_i\}|p) = \begin{cases} 1, & \text{if } p = \{a_1, \ldots, a_m, V_{m+1}, \ldots, V_q\} \ (m > 0) \\ q - |e_i| + 1, & \text{if } p = \{V_1, \ldots, V_q\} \end{cases}$$

*Proof.* Trivial □

According to the equality above, we can ensure that $c(E|\cdot) \geq |E|$, and $k_0(E, \cdot) \geq |E|$.

Unfortunately, things are not as easy when working with $\mathcal{L}_1$. The reason is that an element $e$ can be simultaneously covered by more than one subpattern. Consequently, the reasoning we made in $\mathcal{L}_0$ to obtain the nearest not covered elements must be extended in $\mathcal{L}_1$ to take this circumstance into account. Let us see an example of this:

**Example 10.** *Given the element $e = \{a, b, c\}$ and the pattern $p = p_1 + p_2 + p_3$ where $p_1 = \{a, b, V_1, V_2, V_3\}$, $p_2 = \{a, c, V_1, V_2, V_3\}$ and $p_3 = \{b, c, V_1, V_2, V_3\}$. Clearly, $e_1$ is covered by all the three subpatterns $p_1$, $p_2$ and $p_3$. Focusing on $p_1$, the nearest element to $e$ not covered by $p_1$ is $e' = \{a, c\}$ or $e' = \{b, c\}$. Both possibilities for $e'$ are covered by $p_2$ and $p_3$, respectively. Hence, we should iterate this process in order to obtain the correct $e'$. That is, choosing the element $\{a, c, \}$, a possible nearest element to $\{a, c\}$ which is not covered by $p_2$ is $\{a\}$ which, in turn, is not covered by any of the subpatterns. Likewise, the nearest element to $\{b, c\}$ which is not covered by $p_3$ is $\{c\}$ which, in turn, is not covered by any of the subpatterns. Both $\{a\}$ and $\{c\}$ are at distance 2 of $e = \{a, b, c\}$, and any of them (among others) is the nearest element to $e$ not covered by $p$.*

## 7.2 Distance-based Operators in $\mathcal{L}_0$

First, we define a particular generalisation operator over $\mathcal{L}_0$ called $\uparrow$-transformation. This is just a bottom-up operator which permits us to move through $\mathcal{L}_0$.

**Definition 16.** ***($\uparrow$-transformation)*** *Given a finite set of patterns $\{p_1, \ldots, p_n\} \in \mathcal{L}_0$, we define the mapping $\uparrow(\cdot) : 2^{\mathcal{L}_0} \rightarrow \mathcal{L}_0$ as*

$$\uparrow(\{p_1, \ldots, p_n\}) = \{a_1, \ldots, a_p, V_1, \ldots, V_q\}$$

*such that*

$$\begin{array}{ll} (1) & G = \{a_1, \ldots, a_p\} = \cap_{i=1}^{n} Gr(p_i) \\ (2) & q = max\{|p_i| : \forall 1 \leq i \leq n\} - |G| \end{array}$$

It follows from the definition that $\uparrow(\{p\}) = p$. Additionally, it is clear that the $\uparrow$-transformation is a generalisation operator since given a collection of patterns $p_1, \ldots, p_n$, the $\uparrow$-transformation returns a pattern $p$ such that $Set(p_i) \subset Set(p)$, for every pattern $p_i$. However, this generalisation concerns the syntactic aspects of the patterns while disregards the underlying distance.

Note that the $\uparrow$-transformation can also be used to move through $\mathcal{L}_1$. Roughly speaking, given $p = \sum_{i=1}^{n} p_i$, we can always apply the $\uparrow$-transformation over any subset of patterns taking part in $p$ and obtain a more general pattern $p'$. For instance, we can define the pattern $p'$ from $p$ by doing $p' = \uparrow(\{p_1, p_2, p_3\}) + \sum_{i=4}^{n} p_i$ and we would have that $Set(p) \subset Set(p')$. Let us see a more specific example.

**Example 11.** *Given $p_1 = \{a_1, a_2, V_1\}$, $p_2 = \{a_1, a_2, V_1, V_2\}$ and $p_3 = \{a_1, a_3, a_4\}$ then $\uparrow(\{p_1, p_2, p_3\}) = \{a_1, V_1, V_2, V_3\}$. If we define the pattern $p' = p_1 + p_2 + p_3$, we can obtain a more general pattern in $\mathcal{L}_1$ by doing, $p'' = \uparrow(\{p_1, p_2\}) + p_3 = \{a_1, a_2, V_1, V_2\} + \{a_1, a_3, a_4\}$.*

Finally, we need to introduce a property about the $\uparrow$-transformation which will be used later.

**Proposition 7.** *Given three patterns $p_1$, $p_2$ and $p_3$ belonging to $\mathcal{L}_0$, if $Set(p_1) \subset Set(p_3)$ and $Set(p_2) \subset Set(p_3)$, then $Set(\uparrow(\{p_1, p_2\})) \subset Set(p_3)$.*

*Proof.* If $Set(p_1) \subset Set(p_3)$ (the same result is achieved by using the pattern $p_2$ instead of the pattern $p_1$) then $Gr(p_3) \subset Gr(p_1)$ and $|p_3| \geq |p_1|$. Setting $p' = \uparrow(\{p_1, p_2\})$, we have by definition of the $\uparrow$-transformation that $Gr(p') = Gr(p_1) \cap Gr(p_2)$ and $|p'| = max(|p_1|, |p_2|)$. Thus,

$$Gr(p_3) \subset Gr(p_1) \cap Gr(p_2) = Gr(p'),$$

and

$$|p_3| \geq max(|p_1|, |p_2|) = |p'|$$

Therefore, $Set(p') \subset Set(p_3)$. □

The $\uparrow$-transformation will provide us with the intuition for the definition of distance-based operators. The reasoning is straightforward. Given two sets $e_1$ and $e_2$, if a pattern $p \in \mathcal{L}_0$ generalises $e_1$ and $e_2$, then necessarily $Gr(p)$ must be a subset of $e_1$ and $e_2$, and, additionally, $|p| \geq max\{|e_1|, |e_2|\}$. Therefore, the goal consists of proving whether for any element $e_3$ between $e_1$ and $e_2$, $e_3$ contains

$Gr(p)$ and, if so, how many variables in $p$ are needed in order to cover $e_3$. This is what the next two propositions below state.

First, we focus on characterising the class of distance-based binary operators. Then, we will see how to extend the definition for more than two elements.

**Proposition 8.** ***(Binary distance-based generalisation operator)****. For every pair of elements $e_1$ and $e_2$, a binary generalisation operator $\Delta^b$ is distance-based iff for every pair of elements $e_1$ and $e_2$, $\Delta^b(e_1, e_2) = p \in \mathcal{L}_0$, $Gr(p) \subset e_1 \cap e_2$ and $|p| \geq |e_1 \cup e_2|$.*

*Proof.* ($\Rightarrow$) As $\Delta^b(e_1, e_2)$ is a generalisation operator then necessarily $Gr(p) \subset e_1 \cap e_2$. Otherwise, $e_1$ or $e_2$ would not be covered by $p$. Let us see that the element $e_3 = e_1 \cup e_2$ is between $e_1$ and $e_2$.

$$\begin{aligned} d(e_1, e_3) + d(e_3, e_2) &= |e_1 - (e_1 \cap e_3)| + |e_3 - (e_1 \cap e_3)| + |e_3 - (e_2 \cap e_3)| + |e_2 - (e_2 \cap e_3)| \\ &= |e_3 - e_1| + |e_3 - e_2| \\ &= |(e_1 \cup e_2) - e_1| + |(e_1 \cup e_2) - e_2| \\ &= |e_2 - (e_1 \cap e_2)| + |e_1 - (e_1 \cap e_2)| = d(e_1, e_2) \end{aligned}$$

Since $p$ is a distance-based pattern of $e_1$ and $e_2$, then $e_3 \in Set(p)$ and necessarily the number of symbols in $p$ must be enough to cover $e_1 \cup e_2$. Hence, $|p| \geq |e_1 \cup e_2|$.
($\Leftarrow$) For every three elements $e_1$, $e_2$ and $e_3$, if $e_3$ lies between $e_1$ and $e_2$ then $e_1 \cap e_2 \subset e_3$. If $e_3$ in an intermediate element, then the following relations hold:

$$d(e_1, e_3) + d(e_3, e_2) = d(e_1, e_2)$$

$$\Downarrow \text{ (by the definition of } d)$$

$$(\mid e_1 \mid + \mid e_3 \mid -2 \mid e_1 \cap e_3 \mid) + (\mid e_2 \mid + \mid e_3 \mid -2 \mid e_2 \cap e_3 \mid) = \mid e_1 \mid + \mid e_2 \mid -2 \mid e_1 \cap e_2 \mid \qquad (7.1)$$

$$\Updownarrow \text{ (by simplification)}$$

$$\mid e_3 \mid - \mid e_1 \cap e_3 \mid - \mid e_2 \cap e_3 \mid = - \mid e_1 \cap e_2 \mid$$

On the other hand, we have that

$$\mid e_3 \mid \geq \mid e_1 \cap e_3 \mid + \mid e_2 \cap e_3 \mid - \mid e_1 \cap e_2 \cap e_3 \mid \qquad (7.2)$$

By replacing (7.2) in the last expression in (7.1), we obtain

$$\begin{gathered} - \mid e_1 \cap e_2 \mid \geq - \mid e_1 \cap e_2 \cap e_3 \mid \\ \Updownarrow \\ \mid e_1 \cap e_2 \mid \leq - \mid e_1 \cap e_2 \cap e_3 \mid \end{gathered}$$

But the expression above only holds if $e_1 \cap e_2 \subset e_3$. Therefore,

$$Gr(p) \subset e_1 \cap e_2 \subset e_3 \qquad (7.3)$$

Additionally, something about the cardinality of the elements placed between $e_1$ and $e_2$ must be known to ensure that all of them are covered by $\Delta^b(e_1, e_2)$. In fact, we are going to see that if $e_3$ is an element between $e_1$ and $e_2$ then $|e_3| \leq |e_1 \cup e_2|$. We will proceed by contradiction.

$$\begin{aligned} d(e_1,e_3)+d(e_3,e_2) &= |e_1-(e_1\cap e_3)|+|e_3-(e_1\cap e_3)|+|e_3-(e_3\cap e_2)|+|e_2-(e_3\cap e_2)| \\ &> |e_3-(e_1\cap e_3)|+|e_3-(e_3\cap e_2)| \\ &= |e_3-e_1|+|e_3-e_2| \\ &> |(e_1\cup e_2)-e_1|+|(e_1\cup e_2)-e_2| \\ &= d(e_1,e_1\cup e_2)+d(e_1\cup e_2,e_2)=d(e_1,e_2) \end{aligned} \tag{7.4}$$

Hence,

$$\forall e_3 \text{ between } e_1 \text{ and } e_2 \Rightarrow |e_3| \leq |e_1 \cup e_2| \tag{7.5}$$

Combining (7.3) and (7.5) we have that for every element $e_3$ between $e_1$ and $e_2$, then $e_3$ is covered by $\Delta^b(e_1,e_2)$ and the generalisation operator is distance-based.

□

In $\mathcal{L}_0$, the difference between a pattern $p$ computed by a distance-based generalisation operator and a pattern $p'$ computed by any other generalisation operator (e.g. the $\uparrow$-transformation) relies on their cardinality. This is because more variables are needed in the distanced-based patterns to capture the intermediate elements.

**Example 12.** *Given the elements $e_1=\{a_1,a_2\}$ and $e_2=\{a_1,a_3\}$ and the patterns $p_1=\{a_1,V_1\}$, $p_2=\{V_1,V_2\}$ and $p_3=\{a_1,V_1,V_2\}$. The patterns $p_1$ and $p_2$ are not distance-based since the element $\{a_1,a_2,a_3\}$ is between $e_1$ and $e_2$ and is not covered either by $p_1$ or $p_2$. Nevertheless, the pattern $p_3$ is distance-based.*

The characterisation of binary distance-based generalisation operators has an immediate extension for $n$-ary operators if we previously fix a nerve function $N$.

**Proposition 9.** ***($n$-ary distance-based generalisation operator w.r.t. a nerve function $N$).*** *A generalisation operator $\Delta$ is distance-based w.r.t. to $N$ iff for every finite set of elements $E=\{e_1,\ldots,e_n\}$ $\Delta(E)=p\in\mathcal{L}_0$, $Gr(p)\subset\cap_{i=1}^n e_i$ and $|p|\geq max\{|e_i\cup e_j|:\forall(e_i,e_j)\in N(E)\}$.*

*Proof.* ($\Leftarrow$) Fer every pair $(e_i,e_j)$ in $N(E)$, let $e$ be an element between $e_i$ and $e_j$. From the proof in Proposition 8, we know that $e_i\cap e_j\subset e$. Hence,

$$Gr(p)\subset\cap_{k=1}^n e_k\subset e_i\cap e_j\subset e \tag{7.6}$$

Additionally, from the proof in Proposition 8 we can affirm that $|e|\leq|e_i\cup e_j|$. Hence,

$$|e|\leq|e_i\cup e_j|\leq max\{|e_i\cup e_j|:\forall(e_i,e_j)\in N(E)\}=|p| \tag{7.7}$$

Therefore, by (7.6) and (7.7), $e\in Set(p)$ and $\Delta$ is a distance-based generalisation operator.
($\Rightarrow$) As $\Delta(E)$ is a generalisation of $E$ then $E\subset Set(p)$ and necessarily for every $e_i\in E$, $Gr(p)\subset e_i$. In addition, for every $(e_i,e_j)\in N(E)$, we define

$$\Delta^b(e_i,e_j)=\Delta(E)=p \tag{7.8}$$

Now, according to Proposition 8, we can write

$$\forall(e_i,e_j)\in N(E),\ |p|\geq|e_i\cup e_j| \tag{7.9}$$

which is equivalent to

$$|p| \geq max\{|e_i \cup e_j| : \forall (e_i, e_j) \in N(E)\} \tag{7.10}$$

□

The relationship between binary and $n$-ary *dbg* operators is given below.

**Corollary 1.** *Let $\Delta$ be a $n$-ary distance-based generalisation operator w.r.t. a nerve function $N$, then there exists a binary distance-based generalisation operator $\Delta^b$ such that*

$$\Delta(E) = \uparrow (\{\Delta^b(e_i, e_j) : \forall (e_i, e_j) \in N(E)\})$$

*Proof.* It directly follows from the definition of $\uparrow$-transformation and Proposition 8. □

## 7.3 Minimal Distance-based Generalisation Operators in $(\mathcal{L}_0, k_0)$

After characterising the family of *dbg* operators for sets, we will study those that are minimal. A first attempt to obtain a *mdbg* operator consists of checking whether the naive generalisation operator ($\Delta^0$) is distance-based. The following proposition says that this is not possible.

**Proposition 10.** *The naive generalisation operator $\Delta^0$ for sets is not distance-based in general.*

*Proof.* Given $E = \{e_1, \ldots, e_n\}$, let us consider the case when $\cap_{i=1}^n e_i = \emptyset$. As $\Delta^0(E) = p$ is minimal, then according to the closed form of $c(E|p)$, the number of variables in $p$ is equal to $max\{|e_i| : \forall e_i \in E\}$. But taking Proposition 9 into account, $p$ is not distance-based since more variables are required. □

Basically, the condition of being a *mdbg* operator or not depends on the number of variables in the pattern computed by the operator. If $\cap_{i=1}^n e_i \neq \emptyset$, the number of variables in $\Delta(E)$ is enough to be greater than $max\{|e_i \cup e_j| : \forall e_i, e_j \in E\}$ (see Proposition 9). However, the number of variables required remains unclear when $\cap_{i=1}^n e_i = \emptyset$. Let us see an example.

**Example 13.** *$E = \{e_1, e_2, e_3, e_4\}$ where $e_1 = \{a_1, a_2\}$, $e_2 = \{a_2, a_3\}$, $e_3 = \{a_3, a_4\}$ and $e_4 = \{a_4, a_5\}$. Obviously, $\cap_{i=1}^4 e_i = \emptyset$. The patterns $p_1 = \{V_1, V_2, V_3\}$ and $p_2 = \{V_1, V_2, V_3, V_4\}$ are both distance-based of $E$. Namely, according to the Proposition 9, $p_1$ is distance-based w.r.t. nerve $N_1$ (see left-side Figure 7.1) whereas $p_2$ is distance-based w.r.t. nerve $N_2$ (see right-side Figure 7.1). But as we could expect, $k_0(E, p_1) = 8 < k_0(E, p_2) = 12$.*

Therefore, the hardest case in order to compute a *mdbg* generalisation in this context is given when $\cap_{i=1}^n e_i = \emptyset$. Obtaining the minimum number of required variables is equivalent to check if there exists a nerve w.r.t. the pattern $\Delta(E)$ is distance-based or not. However, this is not worth the computational effort required to return a pattern which contains only variable symbols. Given that this problem vanishes when the nerve is fixed beforehand, it turns out more convenient to compute the *mdbg* operator w.r.t. a nerve function. This is shown below.

**Proposition 11.** ***(Minimal distance-based generalisation operator relative to a nerve function N)*** *Given a finite set of elements $E = \{e_1, \ldots, e_n\}$, a nerve function $N$ and a generalisation operator $\Delta$. If for every set $E$, $\Delta$ satisfies the following conditions,*

1. *$Gr(\Delta(E)) \subset \cap_{i=1}^n e_i$*

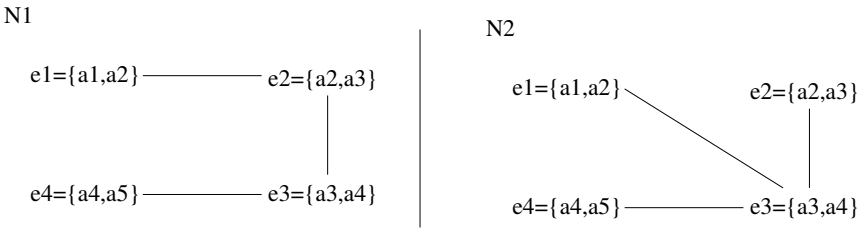


Figure 7.1: Two possible nerves $N_1$ **(left)** and $N_2$ **(right)** for the set $E$. The pattern $p_1 = \{V_1, V_2, V_3\}$ is distance-based w.r.t. $N_1$ and the pattern $p_2 = \{V_1, V_2, V_3, V_4\}$ is distance-based w.r.t. $N_2$.

2. $|\Delta(E)| = max\{|e_i \cup e_j| : \forall (e_i, e_j) \in N(E)\}$
3. *If* $\cap_{i=1}^{n} e_i = \emptyset$ *then* $Gr(\Delta(E)) = \emptyset$.

*then* $\Delta$ *is a minimal distance-based generalisation operator relative to* $N$.

*Proof.* According to Proposition 9, $\Delta$ is distance-based. Let us see that $\Delta$ is also minimal. If $\cap_{i=1}^{n} e_i \neq \emptyset$ then, by hypothesis, $Gr(\Delta(E)) \neq \emptyset$ and by definition of $c(E|p)$ for $\mathcal{L}_0$, $k_0(E, \Delta(E)) = |E|$ which is the lowest value the function $k_0(E, \cdot)$ can attain. On the contrary, if $\cap_{i=1}^{n} e_i = \emptyset$ then $\Delta(E)$ is only made up of variables, but according to Proposition 9, $\Delta(E)$ contain the minimum number of variables required to be distance-based. Therefore, $\Delta(E)$ is minimal. □

Given that every $n$-ary *dbg* operator can be built from an appropriate binary *dbg* operator (see Corollary 1), it would be interesting to set what conditions a binary operator must satisfy so that the $n$-ary *dbg* operator generated is minimal.

**Proposition 12.** *Given a finite set of elements* $E = \{e_1, \ldots, e_n\}$ *and a nerve function* $N$. *Let* $\Delta^b$ *be a binary distance-based generalisation operator which satisfies*

1. $\forall (e_i, e_j) \in N(E)$ *then* $Gr(\Delta^b(e_i, e_j)) = e_i \cap e_j$.
2. $\forall (e_i, e_j) \in N(E)$ *then* $|\Delta^b(e_i, e_j)| = |e_i \cup e_j|$

*Then, the* $n$*-ary distance-based generalisation operator* $\Delta$ *defined from* $\Delta^b$ *as Corollary 1 states, is minimal w.r.t. the nerve function* $N$.

*Proof.* From Condition 1 and by definition of the $\uparrow$-transformation, we have that $Gr(\Delta(E)) = \cap_{i=1}^{n} e_i$. Automatically, we also have that $Gr(\Delta(E)) = \emptyset$ if $\cap_{i=1}^{n} e_i = \emptyset$. Finally, from Condition 2 and again by definition of the $\uparrow$-transformation, we have that $|\Delta(E)| = max\{|e_i \cup e_j| : \forall (e_i, e_j) \in N(E)\}$. Therefore $\Delta$ satisfies the three conditions from Proposition 11 and i.e. $\Delta$ is minimal w.r.t. the nerve function $N$. □

Observe that although $\Delta^b$ is a binary *mdbg* operator, this does not ensure that the *dbg* operator $\Delta$ generated from $\Delta^b$ is minimal. An example of this is shown next.

**Example 14.** *Given the elements* $e_1 = \{a_1, a_2, a_3\}$, $e_2 = \{a_1, a_2\}$ *and* $e_3 = \{a_1, a_3\}$ *and the nerve* $\nu = \{(e_1, e_2), (e_1, e_3)\}$. *We define a binary dbg operator* $\Delta^b$ *such that* $\Delta^b(e_1, e_2) = \{a_2, V_1, V_2\}$ *and* $\Delta^b(e_1, e_3) = \{a_3, V_1, V_2\}$. *Clearly, both are db and minimal patterns of* $E$ *w.r.t.* $\nu$. *However, the pattern* $\Delta(E) = \uparrow (\{a_2, V_1, V_2\}, \{a_3, V_1, V_2\}) = \{V_1, V_2, V_3\}$ *which is db but not minimal. Note that the pattern* $\{a_1, V_1, V_2\}$ *is db of* $E$ *w.r.t.* $\nu$ *and* $k_0(E, \{a_1, V_1, V_2\}) = 3$ *while* $k_0(E, \Delta(E)) = 5$.

Also observe that the *mdbg* operator relative to a nerve function is not unique. This is because the maximum number of variables in a pattern computed by the operator $\Delta$ introduced in Proposition 11 is a sufficient but not a necessary condition. Let us see an example:

**Example 15.** *Given* $E = \{e_1, e_2\}$ *where* $e_1 = \{a_1, a_2, a_3\}$ *and* $e_2 = \{a_1, a_2, a_4\}$ *and given the following distance-based patterns of* $E$ $p_1 = \{a_1, a_2, V_1, V_2\}$, $p_2 = \{a_1, V_2, V_3\}$ *and* $p_3 = \{a_1, V_1, V_2, V_3, V_4, V_5\}$. *It easy to see that*

$$k_0(E, p_1) = k_0(E, p_2) = k_0(E, p_3) = |E|$$

*and in consequence all the patterns are minimal.*

The cost function $k_0$ does not completely match our intuition about what a minimal pattern in $\mathcal{L}_0$ is. Namely, we understand the more ground symbols a pattern generalising $E$ has, the better the pattern fits $E$. But this is not exactly the case because $c(E|p)$ takes only the nearest elements not covered by the patterns into account. If we complete $c(E|p)$ by also taking the furthest elements covered by the patterns, then $k_0$ will rank worse those patterns having more variables than they need.

## 7.4 Distance-based Operators in $\mathcal{L}_1$

As we mentioned before, the motivation behind this pattern language comes from the poor descriptions provided by $\mathcal{L}_0$ when the elements to be generalised have nothing in common altogether but they still have common traits for subsets. Now, and as we previously did, we have to define *dbg* operators in $\mathcal{L}_1$. Proposition 1 in Chapter 4 establishes a way to do this. According to this proposition, we need to define binary *dbg* operators in $\mathcal{L}_1$ to obtain $\Delta_N$ in $\mathcal{L}_1$. However, giving a characterisation of all the binary distance-based operators in $\mathcal{L}_1$, as we did in $\mathcal{L}_0$, is not immediate due to the expressiveness of $\mathcal{L}_1$ (see the example below).

**Example 16.** *Let* $e_1 = \{a_1, a_2\}$ *and* $e_2 = \{a_2, a_3\}$. *As we know,* $p = \{a_2, V_1, V_2\}$ *is a db pattern of* $e_1$ *and* $e_2$. *But*

$$p' = \{a_3, V_1, V_2\} + \{V_1, V_2\}$$

*and*

$$p'' = \{a_2\} + \{a_3, V_1, V_2\} + \{a_1, V_1\}$$

*are db as well since all the elements between* $e_1$ *and* $e_2$, *namely* $\{a_2\}$ *and* $\{a_1, a_2, a_3\}$, *are covered. And it is easy to see that many other patterns can be found.*

A feasible solution for using Proposition 1 in this context, consists of using binary operators defined in $\mathcal{L}_0$. In what follows, this will be the sort of binary *dbg* operators we are working with (see the example below). Concretely, $\Delta_N$ will be the *dbg* operator build, as the Proposition 1 indicates, from the binary *dbg* operator $\Delta^b$ defined in Proposition 12.

**Example 17.** *Given* $E = \{e_1, e_2, e_3\}$ *where* $e_1 = \{a_1, a_2, a_3, a_4, a_5\}$, $e_2 = \{a_1, a_2\}$, $e_3 = \{a_1, a_3\}$, $e_4 = \{a_3, a_4\}$ *and* $e_5 = \{a_3, a_5\}$ *and the nerve* $N(E)$ *depicted below. Then,*

$$\begin{array}{rcl} \Delta^b(e_1, e_2) & = & \{a_1, a_2, V_1, V_2, V_3\} \\ \Delta^b(e_1, e_3) & = & \{a_1, a_3, V_1, V_2, V_3\} \\ \Delta^b(e_1, e_4) & = & \{a_3, a_4, V_1, V_2, V_3\} \\ \Delta^b(e_1, e_5) & = & \{a_3, a_5, V_1, V_2, V_3\} \end{array}$$

*Now,* $\Delta_N(E)$ *is just*

$$\Delta_N(E) = \{a_1, a_2, V_1, V_1, V_3\} + \{a_1, a_3, V_1, V_2, V_3\} + \{a_3, a_4, V_1, V_2, V_3\} + \{a_3, a_5, V_1, V_2, V_3\}$$

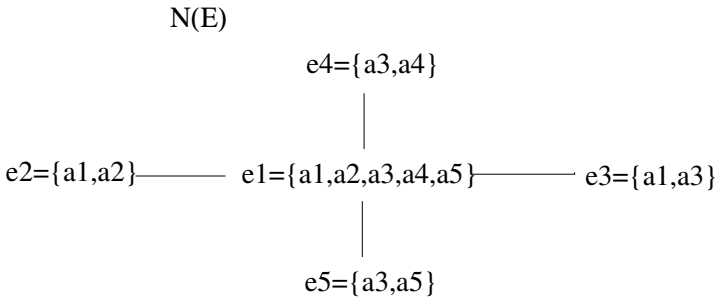


Figure 7.2: Fixing a nerve in order to define the operator $\Delta_N$ for sets.

### 7.4.1 Minimal Distance-based Generalisation Operators in $(\mathcal{L}_1, k_1)$

We are going to study how well a distance-based operator defined in $\mathcal{L}_1$ fits $E$. The first thing will be to see whether the naive generalisation operator in $\mathcal{L}_1$ for sets is distance-based or not.

**Proposition 13.** *The naive generalisation operator* $\Delta^0$ *in* $\mathcal{L}_1$ *for sets is not distance-based.*

*Proof.* Given any pair of elements $e_1$ and $e_2$ such that $e_1 \cap e_2 \neq \emptyset$ and neither $e_1 \not\subset e_2$ nor $e_2 \not\subset e_1$. If $\Delta^0(E)$ minimises $k_1(E, \cdot)$ then the pattern $\Delta^0(\{e_1, e_2\}) = p$ must satisfy

$$c(\{e_1, e_2\}|p) = 2$$

and

$$|p| = max\{|e_1|, |e_2|\}.$$

which is the minimum possible value for the function $k_1(\{e_1, e_2, \cdot\})$. Otherwise, $\Delta^0(E)$ would not be minimal because we can always define a pattern $p'$ such that $Gr(p') = e_1 \cap e_2$ and $|p'| = max\{|e_1|, |e_2|\}$ which satisfies both conditions above and $k_1(E, p') < k_1(E, p)$. Additionally, note that $p \in \mathcal{L}_0$ and according to Proposition 8 cannot be distance-based.

Now, it remains to check that there does not exist any distance-based pattern $p''$ of $\{e_1, e_2\}$ such that $k_1(E, p'') = k_1(E, p)$. Only two cases can be given:

1. $p'' \in \mathcal{L}_0$: as $p''$ is distance-based, then from Proposition 8, $c(p'') > c(p)$

2. $p'' \in \mathcal{L}_1 - \mathcal{L}_0$: we can write $p'' = \sum_{i=1}^{n} p''_i$ $(\forall p''_i \in \mathcal{L}_0)$, and there will always exist a pair of patterns $p''_i$ and a $p''_j$ such that $e_1 \in Set(p''_i)$ and $e_2 \in Set(p''_j)$. Hence, $c(p'') > c(p)$.

Summing up, for every distance-based pattern $p''$ of $\{e_1, e_2\}$, we have that $c(\{e_1, e_2\}|p'') \geq c(\{e_1, e_2\}|p)$ and $c(p'') > c(p')$ and in consequence $k_1(\{e_1, e_2\}, p'') > k_1(\{e_1, e_2\}, p)$. □

Thus, we have to find another way to define *mdbg* operators. As we only know how to define distance-based operators w.r.t. a previously fixed nerve, it makes sense to relax the problem and focus on the *mdbg* operators relative to one nerve function. For instance, we could see how the operator $\Delta_N$ behaves w.r.t. the cost function $k_1$.

**Example 18.** *Let $E$ and $N(E)$ be the set of elements and the nerve from Example 17. The pattern $p_1$ is the one computed by $\Delta_N$ (see Example 17) and the pattern $p_2$ is obtained from the $p_1$ by means of the $\uparrow$-transformation, as we indicate below.*

$$p_1 = \{a_1, a_2, V_1, V_1, V_3\} + \{a_1, a_3, V_1, V_2, V_3\} + \{a_3, a_4, V_1, V_2, V_3\} + \{a_3, a_5, V_1, V_2, V_3\}$$

$$\begin{aligned} p_2 &= \{a_1, a_2, V_1, V_1, V_3\} + \uparrow (\{\{a_1, a_3, V_1, V_2, V_3\} + \{a_3, a_4, V_1, V_2, V_3\} + \{a_3, a_5, V_1, V_2, V_3\}\}) \\ &= \{a_1, a_2, V_1, V_1, V_3\} + \{a_3, V_1, V_2, V_3, V_4\} \end{aligned}$$

*Obviously, both patterns are distance-based w.r.t. $N(E)$. Let us calculate $k_1(E, p_1)$ and $k_1(E, p_2)$. The table below summarises the computations.*

| | | | |
|---|---|---|---|
| $p_1$ | $e'_1 = \{a_1, \ldots, a_6\}$ | $c(\{e_1\|p_1\}) = d(e_1, e'_1) = 1$ | $k_1(E, p_1) = 25$ |
| | $e'_2 = \{a_1\}$ | $c(\{e_2\|p_1\}) = d(e_2, e'_2) = 1$ | |
| | $e'_3 = \{a_3\}$ | $c(\{e_3\|p_1\}) = d(e_3, e'_3) = 1$ | |
| | $e'_4 = \{a_4\}$ | $c(\{e_4\|p_1\}) = d(e_4, e'_4) = 1$ | |
| | $e'_5 = \{a_5\}$ | $c(\{e_5\|p_1\}) = d(e_5, e'_5) = 1$ | |
| $p_2$ | $e'_1 = \{a_1, \ldots, a_6\}$ | $c(\{e_1\|p_2\}) = d(e_1, e'_1) = 1$ | $k_1(E, p_2) = 15$ |
| | $e'_2 = \{a_1\}$ | $c(\{e_2\|p_2\}) = d(e_2, e'_2) = 1$ | |
| | $e'_3 = e'_2$ | $c(\{e_3\|p_2\}) = d(e_3, e'_3) = 1$ | |
| | $e'_4 = \{a_4\}$ | $c(\{e_4\|p_2\}) = d(e_4, e'_4) = 1$ | |
| | $e'_5 = \{a_5\}$ | $c(\{e_5\|p_2\}) = d(e_5, e'_5) = 1$ | |

Table 7.2: The elements $e'_1, e'_2, \ldots$ represent the nearest element to $e_1, e_2, \ldots$ respectively, not covered by the pattern.

*Although $c(E|p_1) = c(E|p_2)$, the pattern $p_1$ has a higher cost than $p_2$ because of its complexity.*

As we can read in the table, in general, the patterns computed by $\Delta_N$ tends to overfit $E$. Intuitively, the more elements $E$ contains, the more symbols the pattern $\Delta_N$ usually requires. To avoid this, and also illustrated in the previous example, a post-process of $\Delta_N$ guided by the $\uparrow$-transformation can help obtain a better pattern in terms of $k_1$.

The question now is, if there exists a method based on the $\uparrow$-transformation such that when applied over $\Delta_N$ the *mdbg* operator relative to $N$ can be obtained (see Figure 7.3). However, the above-mentioned method does not exists neither for the $\Delta^b$ we have used to define $\Delta_N$ nor for other binary *dbg* operators $\Delta'^b$ such that $Set(\Delta^b(e_i, e_j)) \subset Set(\Delta'^b(e_i, e_j))$. This is shown in the following example.

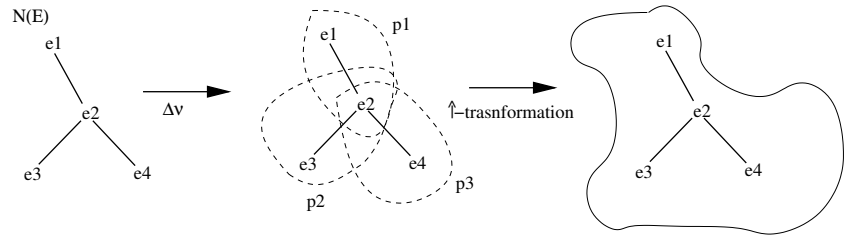


Figure 7.3: Computing a *mg* generalisation from $\Delta_N$ and the $\uparrow$-transformation.

**Example 19.** *Given* $E = \{e_1, e_2, e_3\}$ *where* $e_1 = \{a_1, a_2, a_3\}$, $e_2 = \{a_1, a_6, a_7\}$ *and* $e_3 = \{a_2, a_4, a_5\}$ *being* $N(E) = \{(e_1, e_2), (e_1, e_3)\}$. *Then,*

$$\Delta_N(E) = \{a_1, V_1, V_2, V_3, V_4\} + \{a_2, V_1, V_2, V_3, V_4\}$$

*But the pattern*

$$p = \{a_1, V_1, V_2, V_3, V_4\} + \{a_2, V_1, V_2, V_3\}$$

*has one symbol less than* $\Delta_N(E)$ *and is distance-based. Namely, the only element between* $e_1$ *and* $e_2$, *which is not covered by* $\{a_2, V_1, V_2, V_3\}$, *is* $e_1 \cup e_2$ *but this is covered by* $\{a_1, V_1, V_2, V_3, V_4\}$. *Doing some computations (the nearest elements not covered by the patterns are the same for both patterns, namely,* $e'_1 = \{a_3\}$, $e'_2 = \{a_6, a_7\}$ *and* $e'_3 = \{a_4, a_5\}$*).*

$$\begin{array}{lcl} k_1(E, \Delta_N(E)) & = & c(\Delta_N(E)) + c(E|\Delta_N(E)) = 10 + 4 = 14 \\ k_1(E, p) & = & c(p) + c(E|p) = 9 + 4 = 13 \end{array}$$

*Since* $p$ *has a variable symbol less than* $\Delta_N(E)$, *this means that* $p$ *can never be obtained by means of a* $\uparrow$*-transformation applied over* $\Delta^N(E)$.

It seems that if we want to use the $\uparrow$-transformation to define *mdbg* operators, we need to start from generalisations of $E$ more specific than the one computed by $\Delta_N$. In this case, the only possibility is to apply the $\uparrow$-transformation over the skeleton of the nerve.

**Proposition 14.** *Given a finite set of elements* $E = \{e_1, \ldots, e_n\}$ *and a nerve function* $N$. *Then there exists a partition* $P$ *of the set* $S = skeleton(N(E))$ *such that the pattern*

$$p = \sum_{\forall P_i \in P} \uparrow (P_i)$$

*is a minimal distance-based pattern of* $E$.

*Proof.* Let us suppose that $p' = \sum_{i=1}^{n} p'_i$ (where $p'_i \in \mathcal{L}_0$) is one of the possible minimal distance-based patterns patterns of $E$ w.r.t. $N(E)$. This pattern $p'$ induces a partition over $S$. We organise into a table which elements in $S$ are covered by each pattern $p'_i$ $(1 \leq i \leq n)$ (see Table 7.3).

We denote the $i$-th column at the table by $C_i$. Let us remove the repetitions of a symbol $a_j$ occurring at different columns. After this, we rename the columns as $C'_i$. Clearly, $C'_i \cap C'_j = \emptyset$ for every $i \neq j$.

| $C_1$ | $C_2$ | ... | $C_n$ |
|---|---|---|---|
| $a_{i_{1,1}}$ | $a_{i_{2,1}}$ | ... | $a_{i_{n,1}}$ |
| $a_{i_{1,2}}$ | $a_{i_{2,2}}$ | ... | $a_{i_{n,2}}$ |
| ... | ... | ... | ... |
| $a_{i_{1,m_1}}$ | $a_{i_{2,m_2}}$ | ... | $a_{i_{n,m_n}}$ |

Table 7.3: The $i$-th column contains those elements in $S$ which are covered by the pattern $p'_i$ $(1 \leq i \leq n)$.

According to Proposition 7 we can affirm that

$$Set(\uparrow (C'_i)) \subseteq Set(p'_i),\ \forall 1 \leq i \leq n$$

Now, we let

$$p = \sum_{i=1}^{n} \uparrow (C'_i)$$

Clearly, $p$ is distance based since $S \subset Set(p)$ and minimal since $c(p) \leq c(p')$ and $c(E|p) \leq c(E|p')$. □

**Corollary 2.** *Let $E$ be a finite set of elements, let $N$ be a nerve function and let $\mathcal{P}$ be the family of all the possible partitions of the set $S = skeleton(N(E))$. Then, the generalisation operator $\Delta(E)$ defined as*

$$\Delta(E) = \sum_{\forall P_i \in P} \uparrow (P_i),\ \textit{such that}\ P = argmin_{P \in \mathcal{P}}\ k_1(E, \sum_{\forall P_i \in P} \uparrow (P_i))$$

*is a mdbg operator relative to $N$.*

*Proof.* It directly follows from Proposition 14. □

The problem, which we address now, is devoted to the real applicability of the corollary above. According to this one, the calculation of $\Delta$ requires to compute first $S = skeleton(N(E))$ and then to search for the optimal partition of $S$. Given the computational cost of these two operations and given that we do not know other method to compute *mdbg* operators, we can conclude it would be more reasonable to approximate $\Delta$. The method we propose consists of precisely approximating these two time-consuming tasks. Thus, instead of $skeleton(\cdot)$, we turn to $\Delta^b$ which is minimal in $(\mathcal{L}_1, k_1)$ (see Proposition 15). Instead of searching for the the optimal partition of $skeleton(N(E))$, the patterns return by $\Delta^b$ are generalised via the $\uparrow$-transformation. The patterns to be generalised are determined by means of a greedy search guided by the cost function $k_1$. This idea is formally expressed in the Algorithm 1 which defines the distance-based generalisation operator $\tilde{\Delta}_N(E)$ in order to approximate the *mdbg* operator.

**Proposition 15.** *For every two elements $e_i$ and $e_j$, we define Then, $\Delta^b$ is a binary minimal distance-based generalisation operator in $(\mathcal{L}_1, k_1)$.*

*Proof.* We distinguish two cases:
**case 1:** Now, $\Delta^b(e_i, e_j) = \{V_1, \ldots, V_q\}$. We set $q = |e_i \cup e_j|$, $q_i = |e_i|$ and $q_j = |e_j|$. Next, let us denote by $F$ the set of all the elements which are between $e_i$ and $e_j$ both included (i.e. the skeleton of $(e_i, e_j)$ and let $F_{e_i,e_j} \subseteq F$ the set defined as follows:

$$F_{e_i,e_j} \equiv \{e^{ik} = e_i \cup A_k : \forall A_k \subseteq e_j\} \cup \{e^{jk} = e_j \cup B_k : \forall B_k \subseteq e_i\},$$

A pattern $p \in \mathcal{L}_1$ covering $F_{e_i,e_j}$ could be

$$p = (e_i \cup \{V_1, \ldots, V_{q_j-1}\}) + (e_j \cup \{V_1, \ldots, V_{q_i}\})$$

Let us see that there does not exist any other pattern $p' = \sum_{i=1}^{n} p'_i$ shorter (with fewer symbols) than $p$ covering $F_{e_i,e_j}$ such that $p'_i \neq \{V_1, \ldots, V_q\}$ for every $1 \leq i \leq n$. We will proceed by contradiction. Suppose that $p'$ exists. Then, there will exist a pattern $p'_i$ such that $e_i \cup e_j \in Set(p'_i)$ and i.e. $|p_i| \geq q$. Additionally $p'_i$ will contain at least one non-variable symbol, otherwise $p'_i = \Delta^b(e_i, e_j)$ which is not possible.

Hence, we can always build a set $e \in F_{e_i,e_j}$ by taking elements from $e_i$ and $e_j$ such that $e \not\subset Set(p'_i)$ and $|e| = q-1$. Hence, there will exist a pattern $p'_j$ such that $p'_j \neq p'_i$ and $p'_j$ coin polynomial timevers $e$. Then, $|p'_j| \geq |e| = q - 1$ which leads to the number of symbols in $p'$ is equal to or greater than the number of symbols in $p$. Therefore, a pattern such as $p'$ cannot exist.

This statement leads to the fact that there is no pattern $p' = \sum_{i=1}^{n} p'_i \in \mathcal{L}_1$ shorter than $p$ such that $p'_i \neq \{V_1, \ldots, V_q\}$ for every $1 \leq i \leq n$ and $p'$ covers $F$. If so, given that $F_{e_i,e_j} \subset F$, then there would exist a pattern $p'$ covering $F_{e_i,e_j}$ shorter than $p$ and it is not possible. Hence, for every distance-based pattern $p_F$ of $E$:

$$k_1(E, p_F) > 2 \cdot q - 1 \geq 2 + |e_i| + |e_j| = k_1(\{e_i, e_j\}, \Delta^b(e_i, e_j)),\ \forall q \geq 2$$

and $\Delta^b$ is a binary $mg$ operator.
**case 2:** If $\Delta^b(e_i, e_j) = \{a_1, \ldots, a_p, V_1, \ldots, V_q\}$, then $c(\{e_i, e_j\}|\Delta(e_i, e_j)) = 2$. Therefore, if there exists a distance-based pattern $p'$ such that $k_1(\{e_i, e_j\}, p') < k_1(\{e_i, e_j\}, \Delta^b(e_i, e_j))$ then the only possibility is that $c_2(p') < c_2(\Delta^b(e_i, e_j))$, but this is not possible since $p'$ would have fewer symbols than $|e_i \cup e_j|$ and would not be a distance-based pattern.

□

Observe that both $\Delta^b$ and $k_1$ are, in essence, two parameters of the algorithm. This means that they can be changed if this is required. In fact, recall that we do not have an efficient method to compute $c(E|p)$, unless $p \in \mathcal{L}_0$. Hence, in order to put Algorithm 1 into practise, we would have to approximate $k_1$ as well. For instance, a naive proposal could be cost a function based on a direct extension of the closed form of $c(E|p)$ in $\mathcal{L}_0$ for $\mathcal{L}_1$. We denote this extension by $c'(E|p)$. Basically,

$$c'(E|p = \sum_{i=1}^{n} p_i) = \begin{cases} |E - E_1| + c(E_1|V_1 \ldots V_j),\ \exists p_k = V_1 \ldots V_j \text{ and } E_1 = \{e \in E : |e| \leq j\} \\ |E|,\ otherwise. \end{cases}$$

That is, we assume that for every $e$ in $E$ covered by supatterns consisting of only variables, its nearest element $e'$ not covered by $p$ is obtained by adding symbols to $e$ until exceeding the number of variables in the subpattern. If $e$ is only covered by subpatterns with ground symbols, we assume that $d(e, e')$ is always 1.

A trace of the algorithm using $c'(E|p)$ instead of $c(E|p)$ is given next.

**Input**: $E = \{e_1, \ldots, e_n\}$, $\Delta^b$ (binary *mdbg* operator in $(\mathcal{L}_1, k_1)$) and $N$ (nerve function)
**Output**: A pattern which approximates a *mdb* pattern of $E$ w.r.t. $N(E)$

1 $\tilde{\Delta}_N(E)$
2 **begin**
3 $k \leftarrow 1$;
4 **for** $(e_i, e_j) \in N(E)$ **do**
5 $p_k \leftarrow \Delta^b(e_i, e_j)$;
6 $k \leftarrow k + 1$;
7 **end**
8 $p = \sum_{k=1}^{n} p_k$;
9 **do**
10 $k_p \leftarrow k_1(E, p)$;
11 $p' \leftarrow argmin\{k_1(E, p_{ij}) : \forall 1 \leq i, j, \leq n, p_{ij} =\uparrow (\{p_i, p_j\}) + (p - p_i - p_j)\}$;
12 $k'_p \leftarrow k_1(E, p')$;
13 **if** $k_{p'} < k_p$ **then** $p \leftarrow p'$;
14 **while** $k_{p'} < k_p$
15 return $p$;
16 **end**
17 //The notation $p - p_i - p_j$ employed in the algorithm means all the patterns in $p$ except $p_i$ and $p_j$.;

**Algorithm 1**: A greedy-based algorithm which approximates the *mdbg* operator.

**Example 20.** *Let $E$ and $N(E)$ be the finite set of elements and the nerve depicted in Figure 7.4. Then,*

$$\begin{array}{rcccl} \Delta^b(e_1, e_2) & = & p_1 & = & \{a_1, a_4, V_1, V_2\} \\ \Delta^b(e_2, e_3) & = & p_2 & = & \{a_1, a_5, V_1, V_2\} \\ \Delta^b(e_3, e_4) & = & p_3 & = & \{a_2, a_4, V_1, V_2\} \\ \Delta^b(e_4, e_5) & = & p_4 & = & \{a_2, a_5, V_1, V_2\} \\ \Delta^b(e_5, e_6) & = & p_5 & = & \{a_3, a_5, V_1, V_2\} \\ \Delta^b(e_6, e_7) & = & p_6 & = & \{a_3, a_4, V_1, V_2\} \end{array}$$

*and*

$$p = \{a_1, a_4, V_1, V_2\} + \{a_1, a_5, V_1, V_2\} + \{a_2, a_4, V_1, V_2\} + \{a_2, a_5, V_1, V_2\} + \{a_3, a_5, V_1, V_2\} + \{a_3, a_4, V_1, V_2\}$$

*See lines* 4-8 *in the algorithm. Now, we have to combine the subpatterns via the $\uparrow$-transformation. In each iteration, we search for the pair of subpatterns such that when generalised the value of the cost function is reduced most. If there are several possibilities, we choose one of them. For instance, initially*

$$c(p) + c'(E|p) = 20 + 7 = 27$$

*At the first iteration, (among others) the pair of subpatterns to be generalised via the $\uparrow$-transformation are $p_1$ and $p_2$, which gives the new pattern*

$$p' = \{a_1, V_1, V_2, V_3\} + \{a_2, a_4, V_1, V_2\} + \{a_2, a_5, V_1, V_2\} + \{a_3, a_5, V_1, V_2\} + \{a_3, a_4, V_1, V_2\}$$

*where*

$$c(p') + c'(E|p') = 16 + 7 = 23$$

*See lines* 10-13 *in the Algorithm. After iterating this process until the pattern cannot be improved more, the algorithm returns*

$$\tilde{\Delta}_N(E) = \{a_1, V_1, V_2, V_3\} + \{a_2, V_1, V_2, V_3\} + \{a_3, V_1, V_2, V_3\}$$

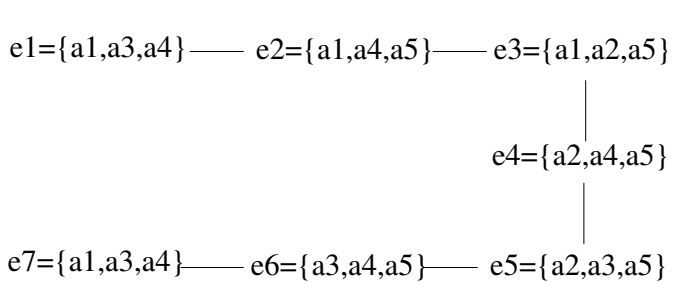


Figure 7.4: A finite set of elements $E$ and a nerve $N(E)$ as input arguments of the greedy-based algorithm.

## 7.5 On the Complexity of Computing Minimal Patterns for Sets

This section attempts to give a more precise perspective about the complexity of computing minimal patterns for sets in languages endowed with the operation $+$. In this line, first, we will see that the *mdbg* operator defined in Corollary 2 is an instance of a $NP$-Hard optimisation problem. Then, we will see that a natural simplification of the language $\mathcal{L}_1$ leads to the computation of minimal patterns in this new language is a $NP$-Hard optimisation problem.

**Lemma 1.** *Let $\mathcal{P}$ be the set of all the binary partitions*[1] *of a finite set $S$. Given a non-negative weight function $w : \mathcal{P} \to \mathcal{N}$, then the optimisation problem*

$$\operatorname*{argmin}_{P \in \mathcal{P}} w(P)$$

*is a $NP$-*Hard *problem.*

*Proof.* Easily, we are going to see that SAT-problem [127] (which is an $NP$-Hard optimisation problem) can be expressed in terms of optimal binary partitions. Thus, let $f(x_1, \ldots, x_n)$ be a Boolean function defined over $n$ Boolean variables. Consider the set of elements $S = \{s_0, s_1, \ldots, s_n\}$. Each binary partition of $S$ can be encoded by a $n+1$ dimensional vector of the form $(0, y_1, \ldots, y_n)$ where

[1] Given a set $S$, we consider that $S \cup \emptyset$ is a binary partition of $S$ as well.

$y_i$ is a Boolean variable informing the subset of the partition the element $s_i$ belongs to. We assume that the partition containing the element $s_0$ is always represented by 0. For instance, imagine the concrete case $S = \{s_0, s_1, s_2, s_3\}$, the partition $P = \{s_0, s_3\} \cup \{s_1, s_2\}$ would be encoded as follows:

$$\begin{array}{rcl} s_0 & \rightarrow & 0 \\ s_1 & \rightarrow & 1 \\ s_2 & \rightarrow & 1 \\ s_3 & \rightarrow & 0 \end{array}$$

In this way, recovering the vector which encodes any partition $P$ of $S$ (this vector is denoted by $\phi(P)$) can be done in polynomial time, since it is enough to see in which subset the element $s_0$ is. Namely, all the elements in this subset are represented by 0 components in the vector while elements in the remaining subset are represented by 1.

Next, we define the cost function $w(\phi(P)) = 1 - f(y_1, \ldots, y_n)$ over the family of partitions of $S$. If $P'$ is an optimal partition, where $\phi(P') = (0, y'_1, \ldots, y'_n)$, then $f(y'_1, \ldots, y'_n) = 1$ and $(y'_1, \ldots, y'_n)$ is a solution for the SAT-problem.

Therefore, solving the optimal binary partition problem is, at least, as hard as solving the SAT-problem. □

**Corollary 3.** *Let $\mathcal{P}$ be the set of all the possible partitions of a finite set $S$. Given a non-negative weight function $w : \mathcal{P} \rightarrow \mathcal{N}$, then the optimisation problem*

$$\operatorname*{argmin}_{P \in \mathcal{P}} w(P)$$

*is a $NP$-*Hard *problem.*

*Proof.* Let $(S, \mathcal{P}_b, w')$ be an instance of the optimal binary partition problem. Now, we consider the instance $(S, \mathcal{P}, w)$ for the optimal $n$-ary partition problem such that $w(P) = w'(P)$, if $P$ is a binary partition, and $w(P) = \infty$ otherwise. The solution of $(S, \mathcal{P}, w)$ is a binary partition. Therefore, solving the optimal $n$-ary partition problem is, at least, as hard as solving the optimal binary partition problem. □

**Corollary 4.** *The computation of the minimal distance-based generalisation operator defined in Corollary 2 is a concrete case of the optimal $n$-ary partition problem.*

*Proof.* Let us see that this computation can be polynomially reduced to the optimal $n$-ary partition problem. First, we set $S = \mathit{skeleton}(N(E))$. Let $\mathcal{P}$ be the set of all the possible partitions of $S$, then we define the non-negative weight function for every partition $P$ as:

$$w(P) = k_1(E, \sum_{\forall P_i \in P} \uparrow (P_i))$$

(see the proof of the Proposition 14). □

Unfortunately, the Corollary 4 cannot ensure if there is a polynomial solution for the computation of the so-mentioned *mdbg* operator. We address this question in a simplified version of the language $\mathcal{L}_1$.

The simplification of $\mathcal{L}_1$ deals with removing the variable symbols from the patterns. Namely, given $A = \{a_1, a_2, a_3, \ldots\}$ the set of ground symbols, we define the pattern language $\mathcal{L}'$ as follows,

$$\mathcal{L}' = \{\{\text{all the sets having the elements } A_1\}, \{\text{all the sets having the elements } A_2\}, \ldots\}$$

where $A_i$ is any finite subset of $A$. Keep in mind that $\mathcal{L}'$ is also endowed with the operation $+$, thus we can define patterns of the form

$$p = \{\text{all the sets having the elements } A_1\} + \{\text{all the sets having the elements } A_2\} + \ldots$$

The cost function $k_1$, which has been employed so far, can easily be adapted for this new pattern language. Given $p = \sum_{i=1}^{n} p_i \in \mathcal{L}'$, $c(p) = \sum_{i=1}^{n} |A_i|$ while the semantic cost remains identical.

Before introducing the main result, we need the following lemma.

**Lemma 2.** *Given a finite set of elements $S$, a family $\mathcal{S}= \{S_1, \ldots, S_k\}$ of subsets of covering $S$ (that is, $S \subset \cup_{i=1}^{k} S_i$) and the non-negative weight function $w(S_i) = 1 + |S_i|$, then the optimisation problem*

$$\underset{\forall C \in 2^{\mathcal{S}} : C \text{ covers } S}{\operatorname{argmin}} \sum_{\forall S_i \in C} w(S_i)$$

*is a NP-*Hard *problem.*

*Proof.* We will reduce the Minimum Set Covering problem ($MSC$ in short) [127], which is a *NP-Hard* problem, to our problem. The $MSC$ problem consists of finding a covering of $S$ using the minimum number of sets in $\mathcal{S}$. In this way, our problem is identical to the $MSC$ problem except for the weight function where $w_{MCS}(S_i) = 1$.

Given the set $S$ and $\mathcal{S}$, let $i$ and $j$ be the highest and the lowest cardinalities of the sets in $\mathcal{S}$. Next, we introduce the set of artificial symbols $\{\diamond_1, \ldots, \diamond_r\}$ where $r = i - j$ and define the transformation $\phi$ over $\mathcal{S}$ as follows:

$$\forall S_k \in \mathcal{S}, \phi(S_k) = S_k \cup \{\diamond_1, \ldots, \diamond_n\} \text{ where } |S_k| + n = i$$

The problem $(S, \mathcal{S}, w_{MCS})$ is an instance of the problem $(S, \phi(\mathcal{S}), w)$. Regarding the second problem, if $C$ is a covering of $S$ then we can write that $w(C) = (1 + i) \cdot |C|$ since all the sets have $i$ elements. Hence, if $C$ is a solution of this problem then, we can affirm that $C$ is the covering of $S$ with the minimum number of sets, otherwise $C$ would not be a solution. As for the first problem, we can conclude that $\phi^{-1}(C)$ is the covering of $S$ with the minimum number of sets.

Therefore, solving the $(S, \mathcal{S}, w)$ problem is, at least, as hard as solving the $MCS$ problem. □

Next we show the impossibility of computing minimal patterns in $(\mathcal{L}', k_1)$ in polynomial time.

**Proposition 16.** *Given a finite set of elements $E$, the optimisation problem*

$$\underset{\forall p \in \mathcal{L}' : E \subset Set(p)}{\operatorname{argmin}} k_1(E, p)$$

*is $NP$-*Hard.

*Proof.* We define a polynomial reduction over the optimisation problem $(S, \mathcal{S}, w)$ introduced in the previous lemma and the optimisation problem $(E, \mathcal{L}', k_1)$ at hand.

$$\begin{array}{rccl} \phi_1: & S & \rightarrow & E \\ & s_i & \rightarrow & e_i = \{''symbol''_{S_j} : \forall S_j \in \mathcal{S} \text{ such that } s_i \in S_j\} \end{array}$$

$$\begin{array}{rccl} \phi_2: & \mathcal{S} & \rightarrow & \mathcal{L}' \\ & S_i & \rightarrow & p_i = \{\text{all the elements having } ''symbol''_{S_i}\} \end{array}$$

Let us see that if $C = \{S_{i_1}, \ldots, S_{i_t}\}$ is a covering of $S$ then $w(C) = k_1(E, \phi_2(C))$ where

$$\phi_2(C) = \{\text{all the elements having } ''symbol''_{S_{i_1}}\} + \ldots + \{\text{all the elements having } ''symbol''_{S_{i_t}}\}$$

On the one hand we have,

$$w(C) = t + \sum_{j=1}^{t} |S_{i_j}|$$

On the other hand, setting $p = \phi_2(C)$, we can write

$$c(p) = t$$

and for every $e_i \in E$, $c(e|p)$ is equal to the number of subpatterns in $p$ covering $e$. For instance, if $e$ is only covered by $\{\text{all the elements having } ''symbol''_{S_{i_1}}\}$ and $\{\text{all the elements having } ''symbol''_{S_{i_2}}\}$ then $c(e|p) = 2$ because it is enough to remove $''symbol''_{S_{i_1}}$ and $''symbol''_{S_{i_2}}$ from $e_i$ in order to obtain the nearest element not covered by $p$. Additionally, if

$$e_i \in Set(\{\text{all the elements having } ''symbol''_{S_{i_1}}\})$$

then, as $e_i = \phi_1(s_i)$, we have that $s_i \in S_{i_1}$ and consequently,

$$c(E|p) = \sum_{j=1}^{t} |S_{i_j}|$$

Hence, $w(C) = k_1(E, \phi_2(C))$. Now, it is clear that if $E \subset Set(p)$ where

$$p = \{\text{all the elements having } ''symbol''_{S_{i_1}}\} + \ldots + \{\text{all the elements having } ''symbol''_{S_{i_t}}\}$$

then $\{S_{i_1}, \ldots, S_{i_t}\}$ is a covering of $S$. We call this, the covering associated to a pattern. Therefore, let $p$ be a minimal pattern of $E$ and let $C$ be the covering associated to $p$, since $w(C) = k_1(E, p)$ then $C$ is the minimal covering of $S$.

Therefore, solving our optimisation problem is, at least, as hard as solving the optimisation problem stated in the previous lemma. □

## 7.6 Discussion

In this chapter, we have seen how to obtain (minimal) distance-based generalisation operators for sets of elements when the distance employed is the symmetric difference for sets. An interesting question would be how to extend these results to cope with multi-sets. Intuitively, this does not seem very difficult, since all the results should be reformulated to take the cardinality of the common elements into account.

# Chapter 8

# Generalising Lists Lying in a Metric Space

After the previous section on sets, we move on to a more complex structure, lists, where order is relevant and repetition can take place.

The examples to be generalised are finite lists defined over an alphabet of symbols. This sort of data is widely used in structured learning. For instance, in bioinformatics, compounds such as amino-acids have a direct representation as sequences of symbols. Furthermore, other much more complex molecules can also be described in terms of sequences by using the so-called 1-D or SMILE representation [**?**]. Another example is found in text mining where documents are usually transformed into sequences[1] of words.

In the section below, we state first the distance function, the pattern languages and the cost functions to be used. Then, we study how to define (minimal) distance-based operators for some particular combinations of pattern language and cost-function.

## 8.1 Distance, Pattern Languages and Cost Functions

As mentioned in Chapter 3, several distance functions for lists can be found in the literature. Here, we use the edit distance [100] in such a way that only insertions and deletions are allowed, or equivalently, an edit distance setting where a substitution can be viewed as a deletion followed by an insertion or vice-versa.

As for pattern description, we work with two different languages $\mathcal{L}_0$ and $\mathcal{L}_1$. The patterns in $\mathcal{L}_0$ are lists that are built from the extended alphabet $\Sigma' = \{\lambda\} \cup \Sigma \cup V$ where $\lambda$ denotes the empty list, $\Sigma = \{a, b, c, \ldots\}$ is the alphabet (also called ground symbols) from which the lists to be generalised are defined, and $V = \{V_1, V_2, \ldots\}$ is a set of variables. The same variable cannot appear twice in a pattern. Each variable in a pattern represents a symbol from $\{\lambda\} \cup \Sigma$. The pattern language $\mathcal{L}_1$ is defined from $\mathcal{L}_0$ by means of the operation $+$ (see Chapter 4). In this way, given a pattern $p$ in $\mathcal{L}_0$ or in $\mathcal{L}_1$ and an element $e$, we say that $e \in Set(p)$ if there exists a substitution $\sigma$ (where $\sigma$ is a

[1] In this section, the terms list and sequence will be used indistinctly.

set of pairs variable/value saying which variables in $p$ must be instantiated by a symbol) such that $e = p\sigma$. Let us see an example of this:

**Example 21.** *Given the pattern $p = V_1 aa V_2 V_3$ and the substitution $\sigma = \{V_1/b, V_3/\lambda\}$ then $p\sigma = baaV_2$. Additionally, the element aab belongs to $Set(p)$ because there exists a substitution, namely $\sigma = \{V_1/\lambda, V_2/b, V_3/\lambda\}$, such that $e = p\sigma$. For the same reason the element aba is not covered by p.*

**Example 22.** *Let $\Sigma = \{a, b\}$ be the alphabet of ground symbols. The pattern language $\mathcal{L}_0$ contains, among others, the following patterns:*

$$\mathcal{L} = \{a, b, aa, ab, ba, bb, aaa, \ldots, V_1, V_1V_2, V_1V_2V_3, \ldots, aV_1, aV_1V_2, \ldots, bV_1, bV_1V_2, \ldots\}$$

*Given the patterns $p_1 = aV_1V_2$ and $p_2 = bV_1V_2b$ and the element $e = aab$, then $e \in Set(p)$ because setting $\sigma = \{V_1/a, V_2/b\}$, $e = p_1\sigma$. Additionally,*

$$Set(p_1) = \{aaa, aab, aba, abb, aa, ab, a\}$$

*and*

$$Set(p_2) = \{baab, babb, bbab, bbbb, bab, bbb, bb\}.$$

*The pattern $p_1$ denotes all those words headed by the symbol a whose length ranges between 1 and 3. In a similar way, $p_2$ contains all the lists headed and ended by b whose length ranges between 2 and 4. Remember that patterns with repeated variables, that is, patterns such as $V_1aV_1$, $bV_1V_2aV_2$, etc. are not admitted in $\mathcal{L}_0$.*

*Now, from $p_1$ and $p_2$, we can define the pattern $p_3 = p_1 + p_2$ in $\mathcal{L}_1$ where*

$$Set(p_3) = Set(p_1) \cup Set(p_2) = \{aaa, aab, aba, abb, aa, ab, a, baab, babb, bbab, bbbb, bab, bbb, bb\}$$

The pattern language $\mathcal{L}_1$ aims to improve the expressiviness provided by $\mathcal{L}_0$. Thus, if the lists to be generalised have no symbols in common, a pattern computed by a a generalisation operator defined over $\mathcal{L}_0$ will simply contain variables (see Example 23). In this case, $\mathcal{L}_0$ can just inform about the length of the lists to be generalised. However, we can find patterns in $\mathcal{L}_1$ which might give more information than simply the length of the lists.

**Example 23.** *Given $e_1 = ab$, $e_2 = ac$ and $e_3 = cb$, as three examples to be generalised. Using $\mathcal{L}_0$, a pattern which covers all of them is made up of variables. For instance, we could take the pattern $V_1V_2$ which says that all the lists have a length equal to or less than 2. Nevertheless, using $\mathcal{L}_1$ we can express the pattern $aV_1 + cV_1$ which informs about the length of the examples as well as some ground symbols they have.*

With regard to the cost function, two different definitions, each one for each pattern language, are introduced. First, $k_0$ (the inclusion-preserving cost function where $k(E, p) = c(E|p)$ with $c(E|p)$ being the first function in Table 5.2) will be used to measure how well a pattern in $\mathcal{L}_0$ fits a set of examples. Secondly, we define $k_1(E, p) = c_1(p) + c'(E|p)$ where $c_1(p)$ measures the complexity of a pattern $p \in \mathcal{L}_1$ by counting both the ground and variable symbols in $p$, and $c'(E|p)$ is an approximation of the semantic cost function used in $k_0$ that we will describe next. The reason why we approximate $c(E|p)$ is due to the computational complexity of calculating $c(E|p)$ in any of the two languages. Let us illustrate this by means of an example.

**Example 24.** *Let $E = \{e_1, e_2\}$ be a binary set where $e_1 = ccaba$ and $e_2 = ac$. Given the patterns generalising $E$:*

$$\begin{aligned} p_1 &= V_1V_2V_3V_4V_5 \\ p_2 &= V_1V_2V_3V_4aV_5V_6V_7V_8 \\ p_3 &= V_1V_2V_3V_4aV_5V_6V_7V_8 + V_1V_2 \end{aligned}$$

*Let us calculate $c(E, \cdot)$ for all the patterns above. In principle, for every pattern $p_i$, we have to find the nearest elements to $e_1$ and $e_2$ which are not covered by the pattern. We denote these elements by $e'_1$ and $e'_2$, respectively.*

*Like in sets, when a pattern contains only variables (e.g. $p_1$), the solution is immediate. Basically, $e'_1$ and $e'_2$ are two supersequences of $e_1$ and $e_2$, respectively, whose length is equal to the number of variables in the pattern plus one (see Table 8.1). However, when a pattern includes ground symbols, the things are not as direct as in the previous case. This is because a subsequence can occur several times in the same sequence, and all these repetitions must be taken into account in order to compute $c(E|\cdot)$. For instance, let us consider the pattern $p_2$. If an element $e$ is not covered by $p_2$, this is because either the symbol $a$ does not occur in $e$ (e.g. $e = ccb$) or the number of symbols before or after each occurrence of $a$ in $e$ is greater than 4 (e.g. $e = ccbbbaba$). From these two possibilities, the nearest element to $e_1$ not covered by $p_2$ is $e'_1 = b$ which is at distance 2 (see Table 8.1). Observe that $e'_1$ is obtained by removing all the symbols $a$ in $e_1$. If we only remove one symbol $a$ the resulting sequence is still covered by the pattern. Additionally, in $\mathcal{L}_1$, we must consider all the subpatterns in order to determine the nearest uncovered elements. For instance, the nearest uncovered element to $e_2$, when considering the pattern $p_2$, is $e'_2 = c$. But this is not valid when considering $p_3$ since $c$ is covered by the subpattern $V_1V_2$ (see Table 8.1).*

| | | | |
|---|---|---|---|
| $p_1$ | $e'_1 = ccabac$<br>$e'_2 = acbbbb$ | $c(\{e_1\|p_1\}) = d(e_1, e'_1) = 1$<br>$c(\{e_2\|p_1\}) = d(e_2, e'_2) = 4$ | $c(E\|p_1) = 5$ |
| $p_2$ | $e'_1 = ccb$<br>$e'_2 = c$ | $c(\{e_1\|p_2\}) = d(e_1, e'_1) = 2$<br>$c(\{e_2\|p_2\}) = d(e_2, e'_2) = 1$ | $c(E\|p_2) = 3$ |
| $p_3$ | $e'_1 = b$<br>$e'_2 = cbb$ | $c(\{e_1\|p_3\}) = d(e_1, e'_1) = 2$<br>$c(\{e_2\|p_3\}) = d(e_2, e'_2) = 3$ | $c(E\|p_3) = 3$ |

Table 8.1: The nearest elements to $E$ not covered by their respective patterns.

Putting the reasoning above into a more general way, we can say that, from a computational point of view, the calculus of $c(e|p)$ ($p \in \mathcal{L}_0$) can be as complicated as determining the number of times a sequence $s_1$ occurs in a sequence $s_2$. That is, if $s_p$ is the sequence of ground symbols in $p$ and $e'$ is the nearest element to $e$ not covered by $p$, then $e'$ will be a supersequence or a subsequence of $e$ which will be obtained by modifying (deletions or insertions), potentially, all the occurrences of $s_p$ in $e$ (e.g. the element $e_1$ and the pattern $p_2$ from the previous example). Of course, as for the general form $c(E|p)$ where $p \in \mathcal{L}_1$, this operation must be repeated for all the elements in $E$ and for all the subpatterns in $p$.

Therefore, if the learning problem requires the use of a cost function (e.g. because we are interested in minimal generalisations), it might be more convenient to approximate $c(E|p)$, instead of handling the original definition. Next, we propose a naive but intuitive approximation of $c$ inspired on the one we introduced for sets:

$$c'(E|p = \sum_{i=1}^{n} p_i) = \begin{cases} |E - E_1| + c(E_1|V_1 \dots V_j), \exists p_k = V_1 \dots V_j \text{ and } E_1 = \{e \in E : \text{length of } e \leq j\} \\ |E|, \text{ otherwise.} \end{cases}$$

The justification is as follows. If there exists a pattern $p_k = V_1 \dots V_j$ in $p$, then it is immediate that for very element $e$ such that its length $l$ is equal to or less than $i$, its nearest element not covered by $p$ is, at least, at a distance $j - l + 1$, which is the value computed by $c(e|V_1 \dots V_j)$. Otherwise, we assume that the nearest element of $e$ is, at least, at a distance of 1. Implicitly, we are assuming that the nearest element to $e$ can be obtained by removing (or adding) one specific ground symbol from (to) $e$. Table 8.2 illustrate this using the elements and patterns from the previous example.

| | | |
|---|---|---|
| $p_1$ | $c'(\{e_1\|p_1\}) = 1$<br>$c'(\{e_2\|p_1\}) = 4$ | $c'(E\|p_1) = 5$ |
| $p_2$ | $c'(\{e_1\|p_2\}) = 1$<br>$c'(\{e_2\|p_2\}) = 1$ | $c'(E\|p_2) = 2$ |
| $p_3$ | $c'(\{e_1\|p_3\}) = 1$<br>$c'(\{e_2\|p_3\}) = 1$ | $c'(E\|p_3) = 2$ |

Table 8.2: Approximating $c(E|p)$ by $c'(E|p)$.

Although the assumptions behind $c'$ might be quite strong, they make sense in contexts where subsequences hardly occur more than once in a sequence, such as the practical example we address in Chapter 12. In any case, the analysis of obtaining minimal patterns is undertaken for both the original definition of the semantic cost function over $\mathcal{L}_0$ and the approximated version over $\mathcal{L}_1$. Notation and definitions required throughout the rest of this chapter are introduced next.

## 8.2 Notation and Previous Definitions

The function $Seq(\cdot)$ defined over a pattern $p \in \mathcal{L}_0$ returns the sequence of ground symbols in $p$. For example, setting $p = V_1 a a V_2 b$, then $Seq(p) = aab$. The bar notation $|\cdot|$ denotes the length of a sequence (here a sequence can be an element, a pattern, etc.). For instance, in the previous case, $|p| = 5$. The $i$-th symbol in a sequence $s$ is denoted by $s(i)$. Sequences are indexed starting from 1. Following with the example, $p(1) = V_1$, $p(2) = a$, $\dots, p(5) = b$. The set of all the indices of $p$ is denoted by $I(p)$. Thus, $I(p) = \{1, 2, 3, 4, 5\}$.

We sometimes use superscript as a shorthand notation to write sequences and patterns in a shorter way. For instance, $V^5 a^3 V^2$ is equivalent to $V_1 \dots V_5 a a a V_6 V_7$ and $V^2 (ab)^3 c$ is the same as $V_1 V_2 a b a b a b c$.

Finally, we will often introduce mappings which will be defined from one sequence to another. By $Dom(\cdot)$ and $Im(\cdot)$ we denote the domain and the image, respectively, of a mapping.

The next concept is the widely-used in biological motif discovery and musical information retrieval.

**Definition 17. (Maximum common subsequence)** *Given a set of sequences $E = \{e_1, \dots, e_n\}$, and according to [161], the maximum common subsequence (mcs, to abbreviate) is the longest (not necessarily continuous) subsequence of all the sequences in $E$.*

This concept is already widely used in pattern recognition. Note that the *mcs* of a group of sequences is not necessarily unique, as the following example shows:

**Example 25.** *Given $E = \{e_1, e_2, e_3\}$ where $e_1 = baabbac$, $e_2 = acaca$ and $e_3 = ababac$, the mcs of $E$ is the (not continuous) subsequence aaa but also the (not continuous subsequence) aac is also possible.*

The following definitions will allow us to handle common subsequences in a more algebraic fashion. This level of abstraction will help us to develop the proofs of the main results of this chapter.

**Definition 18. (Alignment)** *Given two elements $e_1$ and $e_2$, we say that the mapping $M_{e_2}^{e_1} : I(e_1) \rightarrow I(e_2)$ is an alignment of $e_1$ with $e_2$ if:*

*i)* $\forall i \in Dom(M_{e_2}^{e_1}),\ e_1(i) = e_2(M_{e_2}^{e_1}(i))$
*ii)* $M_{e_2}^{e_1}(i)$ *is a strictly increasing function in* $Dom(M_{e_2}^{e_1})$.

**(Remark 1)** If $Dom(M_{e_2}^{e_1}) = \emptyset$, we say that $M_{e_2}^{e_1}$ is the empty alignment of $e_1$ with $e_2$. Thus, for every pair of elements we can affirm that there is always at least one an alignment between them.

**(Remark 2)** Note that the alignment definition it is also valid when $e_1 = e_2$.

We call $e_1(i) = e_2(M_{e_2}^{e_1}(i))$ a (symbol) matching. We denote as $Rang(M_{e_2}^{e_1})$ the number of matchings between $e_1$ and $e_2$ captured by $M_{e_2}^{e_1}$. The function $Seq(M_{e_2}^{e_1})$ returns the sequence implicitly represented by the matchings. Since $I(e_1)$ and $I(e_2)$ are finite sets, an alignment $M_{e_2}^{e_1}$ can be written as a $2 \times n$ matrix where $n = Rang(M_{e_2}^{e_1})$. Hence,

$$M_{e_2}^{e_1} = \begin{pmatrix} a_{11} & \dots & a_{1n} \\ a_{21} & \dots & a_{2n} \end{pmatrix}$$

where $e_1(a_{1i}) = e_2(a_{2i})$ forall $1 \leq i \leq n$ (condition $i$) from the Definition 18) and $a_{1j} < a_{1(i+1)}$ and $a_{2i} < a_{2(i+1)}$ forall $1 \leq i \leq n-1$) (condition $ii$) from the Definition 18). An element of $M_{e_2}^{e_1}$ placed at the row $i$ and the column $j$ is denoted by $(M_{e_2}^{e_1})_{ij}$. Let us illustrate all these ideas by means of an example.

**Example 26.** *Given the elements $e_1 = caabbc$ and $e_2 = aacd$ where $I(e_1) = \{1,2,3,4,5,6\}$ and $I(e_2) = \{1,2,3\}$. We define the alignment $M_{e_2}^{e_1}$ ($M$ in short) as*

$$M = \begin{pmatrix} 2 & 6 \\ 1 & 3 \end{pmatrix} \equiv \begin{array}{cccccc} c & a & a & b & b & \\ & a & & & & a \end{array} \quad \begin{array}{ccc} & & c \\ a & c & d \end{array}$$

*Note that $M$ satisfies both conditions from Definition 18. Following with $M$, we have that $Dom(M) = \{2,6\}$, $Im(M) = \{1,3\}$, $Rang(M) = 2$ and $Seq(M) = ac$. The mappings $P_{e_2}^{e_1}$ and $Q_{e_2}^{e_1}$ defined below are not alignments.*

$$P = \begin{pmatrix} 1 & 2 \\ 1 & 3 \end{pmatrix} \quad Q = \begin{pmatrix} 1 & 3 & 2 \\ 3 & 1 & 2 \end{pmatrix}$$

*The condition i) (from Definition 18) is not satisfied in $P$ whereas the condition ii) (from Definition 18) is not satisfied in $Q$.*

We are strongly interested in those alignments which have the greatest number of matches between $e_1$ and $e_2$. We call them optimal alignments.

**Definition 19. (Optimal alignment)** *Given two elements $e_1$ and $e_2$ in $X$ and let $M_{e_2}^{e_1}$ be an alignment of $e_1$ with $e_2$, we say that $M_{e_2}^{e_1}$ is an optimal alignment if $Seq(M_{e_2}^{e_1})$ is a mcs of $e_1$ and $e_2$.*

**Example 27.** *Again, $e_1 = caabbc$ and $e_2 = aacd$. The alignment $N_{e_2}^{e_1}$ ($N$ in short) defined as*

$$N = \begin{pmatrix} 2 & 3 & 6 \\ 1 & 2 & 3 \end{pmatrix} \equiv \begin{array}{cccccc} c & a & a & b & b & c \\ & a & a & & & c & d \end{array}$$

*is an optimal alignment. However, the alignment $M$ defined in the previous example is not an optimal alignment.*

Given that different optimal alignments can be defined over two elements $e_1$ and $e_2$, we might be interested in obtaining a concrete optimal alignment. To do this, we define a total order over the set of optimal alignments in order to specify the optimal alignment we want.

**Definition 20.** ***(Total order for optimal alignments)*** *Given two elements $e_1$ and $e_2$ and given the optimal alignments $M_{e_2}^{e_1}$ ($M$ in short) and $N_{e_2}^{e_1}$ ($N$ in short) defined as*

$$M = \begin{pmatrix} a_{11} & \dots & a_{1n} \\ a_{21} & \dots & a_{2n} \end{pmatrix} \quad N = \begin{pmatrix} b_{11} & \dots & b_{1n} \\ b_{21} & \dots & b_{2n} \end{pmatrix}$$

*we say that $M < N$ iff $(a_{11}, \dots, a_{1n}, a_{21}, \dots, a_{2n}) <_{LO} (b_{11}, \dots, b_{1n}, b_{21}, \dots, b_{2n})$ where $<_{LO}$ is the Lexicographical Order for numerical tuples.*

**Example 28.** *Given $e_1 = aab$ and $e_2 = ab$, we define the optimal alignments*

$$M_{e_2}^{e_1} = \begin{pmatrix} 1 & 3 \\ 1 & 2 \end{pmatrix} \quad N_{e_2}^{e_1} = \begin{pmatrix} 2 & 3 \\ 1 & 2 \end{pmatrix}$$

*Then $M_{e_2}^{e_1} < N_{e_2}^{e_1}$.*

Every alignment between two elements $e_1$ and $e_2$ induces a pattern $p$ in $\mathcal{L}_0$ which covers both $e_1$ and $e_2$. This pattern is unique and we call it the pattern associated to an alignment.

**Definition 21. (Pattern associated to an alignment)** *Let $e_1$ and $e_2$ be two elements and let $M_{e_2}^{e_1}$ ($M$ in short) be an alignment of $e_1$ with $e_2$. We say that a pattern $p \in \mathcal{L}_0$ is a pattern associated to the alignment $M$ (and denoted by $p_M$), if*
*i) $Seq(M) = Seq(p)$*
*ii) Letting $n = Rang(M)$, $l_1 = |e_1|$ and $l_2 = |e_2|$, the variable symbols in $p$ are distributed as follows:*

- *The number of variables in the pattern $p$ before the first ground symbol is equal to*

$$((M)_{11} - 1) + ((M)_{21} - 1)$$

- *The number of variables between whatever two ground symbols $p(i)$ and $p(j)$ $(i < j)$ in $Seq(p)$ such that there does not exists $i < k < j$ where $p(k)$ is a ground symbol, is equal to*

$$(M)_{1(i+1)} - (M)_{1i} - 1) + ((M)_{2(i+1)} - (M)_{2i} - 1)$$

- *The number of variables after the last ground symbol in $p$ is equal to*

$$(l_1 - (M)_{1n}) + (l_2 - (M)_{2n})$$

**(Remark 1)** If $M$ is the empty alignment then $p_M = V^{l_1+l_2}$ and $Seq(M) = \lambda$.
Again, we are interested in those patterns associated to the optimal alignments.

**Definition 22.** ***(Optimal alignment pattern)*** *If $M_{e_2}^{e_1}$ is an optimal alignment of $e_1$ with $e_2$, we say that $p_{M_{e_2}^{e_1}}$ is an optimal alignment pattern.*

The concepts of alignment pattern and optimal alignment pattern are shown in the following example.

**Example 29.** *Given the elements $e_1 = caabbc$ and $e_2 = cabdcd$ we define the optimal alignment $M_{e_2}^{e_1}$ ($M$ in short) between $e_1$ and $e_2$ as*

$$M_{e_2}^{e_1} = \begin{pmatrix} 1 & 2 & 4 & 6 \\ 1 & 2 & 3 & 4 \end{pmatrix} \equiv \begin{array}{cccccccc} c & a & a & b & b & & c & \\ c & a & & b & & d & c & d \end{array}$$

*The pattern associated to the alignment $M$ is $p_M = caVbV^2cV$. Since $M$ is an optimal alignment then $p_M$ is an optimal alignment pattern. Note that the following patterns $p_1 = VcaVbV^2cV$, $p_2 = caVbVcV$ and $p_3 = caVbV^2cV^2$ cover $e_1$ and $e_2$, but they are not associated to $M$ because of the distribution of the variables.*

We need to introduce two theoretical results related to the alignments which will be used in a posterior proposition.

**Lemma 3.** *Given the elements $e_1$, $e_2$ and $e_3$ and the alignments $M_{e_2}^{e_1}$ and $M_{e_3}^{e_2}$. Then, the mapping $M_{e_3}^{e_1} : I(e_1) \to I(e_3)$ defined as the composition of $M_{e_2}^{e_1}$ and $M_{e_3}^{e_2}$, and denoted by $M_{e_3}^{e_1} = M_{e_2}^{e_1} \circ M_{e_3}^{e_2}$ is an alignment of $e_1$ with $e_3$.*

*Proof.* The proof is direct and we will simply show that the properties $i$) and $ii$) from the Definition 18 hold for $M_{e_3}^{e_1}$.
**(1st property)** From the definition of mapping composition, we know that for every, $k_1 \in Dom(M_{e_3}^{e_1})$, there exist $k_2 \in Dom(M_{e_3}^{e_2})$ such that $M_{e_2}^{e_1}(k_1) = k_2$. Consequently, $M_{e_3}^{e_1}(k_1) = M_{e_3}^{e_2}(M_{e_2}^{e_1}(k_1))$. Now, we can write

$$e_1(k_1) = e_2(M_{e_2}^{e_1}(k_1)) = e_2(k_2) = e_3(M_{e_3}^{e_2}(k_2)) = e_3(M_{e_3}^{e_2}(M_{e_2}^{e_1}(k_1)) = e_3(M_{e_3}^{e_1}(k_1))$$

and the first property holds.
**(2nd property)** We have that,

$$\forall k_1 < k_1' \in Dom(M_{e_3}^{e_1}),\ M_{e_2}^{e_1}(k_1) = k_2 < M_{e_2}^{e_1}(k_1') = k_2' \text{ and } M_{e_3}^{e_2}(k_2) < M_{e_3}^{e_2}(k_2').$$

Then,

$$\forall k_1 < k_1' \in Dom(M_{e_3}^{e_1}),\ M_{e_3}^{e_2}(M_{e_2}^{e_1}(k_1)) < M_{e_3}^{e_2}(M_{e_2}^{e_1}(k_1'))$$

□

**(Remark 1)** Note that $Dom(M_{e_3}^{e_1}) \subset Dom(M_{e_2}^{e_1})$ and $Im(M_{e_3}^{e_1}) \subset Im(M_{e_3}^{e_2})$. The above result leads to the following equalities:

$$|Dom(M_{e_3}^{e_1})| = |Im(M_{e_3}^{e_1})| = |Im(M_{e_2}^{e_1}) \cap Dom(M_{e_3}^{e_2})|$$

**Lemma 4.** *Given the elements $e_1, e_2$ and the alignment $M_{e_2}^{e_1}$ then the inverse function $(M_{e_2}^{e_1})^{-1}$ is an alignment of $e_2$ with $e_1$ and this is denoted by $M_{e_1}^{e_2}$.*

*Proof.* Given that $M_{e_2}^{e_1}$ is a strictly increasing function, then $M_{e_2}^{e_1}$ is a bijection between $Dom(M_{e_2}^{e_1})$ and $Im(M_{e_2}^{e_1})$. In consequence, the inverse function exists and is, in turn, a strictly increasing function over $Im(M_{e_2}^{e_1})$. Trivially we have that,

$$\forall i \in Dom((M_{e_2}^{e_1})^{-1}) \Rightarrow e_1((M_{e_2}^{e_1})^{-1}(i)) = e_2(i)$$

Hence, $(M_{e_2}^{e_1})^{-1}$ is an alignment as well. □

**(Remark 1)** Observe that for a given alignment $M_{e_2}^{e_1}$ then $M_{e_1}^{e_2}(M_{e_2}^{e_1}(i)) = i$.
The concepts of alignment composition and alignment inversion are shown in the following example:

**Example 30.** *Let $e_1 = caabbc$, $e_2 = cabcd$ $e_3 = abd$ be three elements and let $M_{e_2}^{e_1}$ and $M_{e_3}^{e_2}$ be the alignments of $e_1$ with $e_2$ and $e_2$ with $e_3$ defined as*

$$M_{e_2}^{e_1} = \begin{pmatrix} 1 & 2 & 4 & 6 \\ 1 & 2 & 3 & 4 \end{pmatrix} \quad M_{e_3}^{e_2} = \begin{pmatrix} 2 & 3 & 5 \\ 1 & 2 & 3 \end{pmatrix}$$

*The alignment $M_{e_3}^{e_1}$ resulting from the composition $M_{e_2}^{e_1} \circ M_{e_3}^{e_2}$ is:*

$$M_{e_3}^{e_1} = \begin{pmatrix} 2 & 4 \\ 1 & 2 \end{pmatrix}$$

*Finally, the inversion of this latter alignment $M_{e_3}^{e_1}$ is:*

$$(M_{e_3}^{e_1})^{-1} = M_{e_1}^{e_3} = \begin{pmatrix} 1 & 2 \\ 2 & 4 \end{pmatrix}$$

The alignment and optimal alignment concepts (Definitions **??** and **??**) can be easily extended to cope with patterns in $\mathcal{L}_0$. Given two patterns $p_1$ and $p_2$, $M_{p_2}^{p_1}$ is an alignment of $p_1$ with $p_2$ if $M_{p_2}^{p_1}$ satisfies the conditions $i$) and $ii$) in Definition 18 and additionally

$$\forall i \in Dom(M_{p_2}^{p_1}),\ p_1(i) \text{ and } p_2(M_{p_2}^{p_1}(i)) \text{ are ground symbols}$$

Analogously, $M_{p_2}^{p_1}$ is an optimal alignment if $Seq(M_{p_2}^{p_1})$ is *mcs* of $Seq(p_1)$ and $Seq(p_2)$.

**Example 31.** *Given the patterns $p_1 = V_1abcV_2$ and $p_2 = aV_1c$ where $I(p_1) = \{1,2,3,4,5\}$ and $I(p_2) = \{1,2,3\}$, then and alignment $M$ of $p_1$ with $p_2$ can be*

$$(M \begin{pmatrix} 2 & 4 \\ 1 & 3 \end{pmatrix}$$

*Since $p_1(2) = p_2(M(2)) = a$ and $p_1(4) = p_2(M(4)) = c$. Additionally, $M$ is an optimal alignment because $ac$ is a mcs of $Seq(p_1)$ and $Seq(p_2)$.*

To conclude, as we did in the previous chapter for sets, we present a bottom-up generalisation operator defined over $\mathcal{L}_0$, which allows us to move through the pattern language. By analogy with the generalisation operator defined for sets, this will also be called $\uparrow$-transformation. However, in this case the difference is that the $\uparrow$-transformation for lists is a binary operator. Thus:

**Definition 23.** *Given two patterns $p_1$ and $p_2$ in $\mathcal{L}_0$ we define the binary mapping*

$$\begin{array}{rccl} \uparrow(\cdot,\cdot): & \mathcal{L}_0 \times \mathcal{L}_0 & \rightarrow & \mathcal{L}_0 \\ & (p_1, p_2) & \rightarrow & \uparrow(p_1, p_2) = p, \end{array}$$

*such that if $M_{p_2}^{p_1}$ ($M$ in short) is the minimum ($<_{LO}$) optimal alignment of $p_1$ with $p_2$ then*

1. *$Seq(p) = Seq(M)$*
2. *$p = V^{max\{|p_1|,|p_2|\}}$ if $Seq(M) = \lambda$. Otherwise, the distribution of the variables in $p$ is as follows:*
   - *Before the first ground symbol in $p$, the number of variable is equal to:*
   $$max\{(M)_{11} - 1, (M)_{21} - 1\}$$
   - *Between two consecutive ground symbols in $p$, the number of variables is equal to:*
   $$max\{(M)_{1(i+1)} - (M)_{1i} - 1, (M)_{2(i+1)} - (M)_{2i} - 1\}$$
   - *After the last ground symbol in $p$, the number of variables is equal to (letting $n = Rang(M)$, $l_1 = |p_1|$ and $l_2 = |p_2|$):*
   $$max\{l_1 - (M)_{1n}, l_2 - (M)_{2n}\}$$

**Example 32.** *Given the patterns $p_1 = abcV_1$, $p_2 = V_1abcccV_2$ and $p_3 = dV_1$ then $\uparrow(p_1, p_2) = VabcV^3$ and $\uparrow(p_1, p_3) = V^4$.*

**(Remark 1)** The $\uparrow$-transformation is **not commutative**. For instance, given $p_1 = acV$ and $p_2 = caV$, then $\uparrow(p_1, p_2) = VaV^2$ and $\uparrow(p_2, p_1) = VcV^2$.

**(Remark 2)** The $\uparrow$-transformation is **not associative**. For instance, given $p_1 = caV$, $p_2 = acV$ and $p_3 = aV$. Then, $\uparrow(\uparrow(p_1, p_2), p_3) = V^4$ and $\uparrow(p_1, \uparrow(p_2, p_3)) = VaV^3$.

**(Remark 3)** Note that this $\uparrow$-transformation cannot be extended for $n$ elements since its computation turns into an $NP$-Hard problem. Basically, the computation of the $\uparrow$-transformation involves the computation of the *mcs* which is an $NP$-Hard problem for an arbitrary number $n$ of lists [161].

**Proposition 17.** *Given a set of patterns $\{p_1, \ldots, p_n\}$ in $\mathcal{L}_0$ such that $Seq(p_i) = Seq(p_j)$ $(\forall i \neq j)$ then the $\uparrow$-transformation is commutative and associative over this set of patterns.*

*Proof.* For any pattern $p_i$, we set $Seq(p_i) = s_1 \ldots s_n$. Since all the patterns have the sequence $s_1 \ldots s_n$, the pattern obtained by applying the $\uparrow$-transformation over the set will also have this sequence, independently of the chosen order. Then, the resulting pattern is of the form $V^{i_1} s_1 \ldots V^{i_{n-1}} s_n V^{i_n}$ where $i_j$ $(1 \leq j \leq n)$ is necessarely equal to the length of the longest sequence of variables between the ground symbols $s_{j-1}$ and $s_j$ of a pattern $p_i$. As the numbers $i_j$ are also independent of the chosen order, the $\uparrow$-transformation is associative. The same reasoning can be applied in order to prove the commutativity of the transformation. □

**(Remark 1)** For simplicity, if the $\uparrow$-transformation is commutative and associative over a set of patterns $P$ then we directly write $\uparrow(P)$.

**Proposition 18.** *For every pair of patterns $p_1$ and $p_2$ in $\mathcal{L}_0$, if $p = \uparrow(p_1, p_2)$ then $Set(p_1) \subset Set(p)$ and $Set(p_2) \subset Set(p)$.*

*Proof.* It is direct from the definition of the $\uparrow$-transformation. □

Since the $\uparrow$-transformation is in essence a way of generalising, one could wonder if the following property, concerning the degree of generalisation performed by the $\uparrow$-transformation, holds:

$$\forall p_1, p_2 \in \mathcal{L}_0 \Rightarrow \nexists p' \in \mathcal{L}_0 : Set(p_i) \subset Set(p') \subset Set(\uparrow(p_1, p_2)) \ \ (i = 1, 2) \tag{8.1}$$

However, this is not the case, as the example below shows.

**Example 33.** *Given $p_1 = V^{10}abcV$ and $p_2 = VababcV^{10}$ and according to the definition of $\uparrow$-transformation we have that $\uparrow(p_1, p_2) = V^{10}abV^2cV^{10}$, since the minimum mcs of $Seq(p_1)$ and $Seq(p_2)$ is:*

$$\begin{array}{lcccccc} Seq(p_1) & = & a & b & & & c \\ Seq(p_2) & = & a & b & a & b & c \end{array}$$

*However, we have the pattern $p' = V^{10}abcV^{10}$ which corresponds to the optimal alignment*

$$\begin{array}{lcccccc} Seq(p_1) & = & & & a & b & c \\ Seq(p_2) & = & a & b & a & b & c \end{array}$$

*is different to $\uparrow(p_1, p_2)$ and $Set(p_i) \subset Set(p') \subset Set(\uparrow(p_1, p_2))$.*

Although the property expressed in (8.1) would lead to an alternative definition of the $\uparrow$-transformation (a transformation looking for a better fitness of the original patterns), it is shown next that the main theoretical results are independent of whether the $\uparrow$-transformation satisfies Property 8.1 or not. Additionally, a method implementing a transformation satisfying this property has a high computational cost because all the possible alignments between $Seq(p_1)$ and $Seq(p_2)$ must be considered in order to choose the correct one.

As it happens with the $\uparrow$-transformation for sets, we must point out that the $\uparrow$-transformation for lists also concerns the syntactic aspects of the patterns while missing the distance. This means that if we compute $\uparrow(e_1, e_2)$ the obtained pattern is not distance-based in general (see Example 34).

**Example 34.** *Given $e_1 = ab$ and $e_2 = ac$ then we have that $p = \uparrow(e_1, e_2) = aV$ but the element $abc$ is between $e_1$ and $e_2$ and is not covered by $p$.*

Next, let us see how to define (minimal) distance-based operators for the different pattern languages and cost functions.

## 8.3 Distance-based Operators in $\mathcal{L}_0$

One would expect that an operator $\Delta(E)$ which computes a pattern $p$ such that $Seq(p)$ is a *mcs* of the lists in $E$ is a distance-based generalisation operator. However, we find that this operator is not, in general, distance-based. The following example illustrates this:

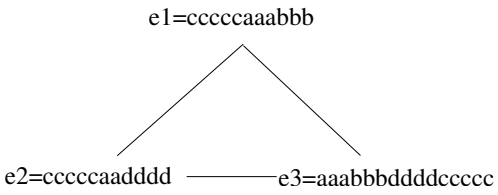


Figure 8.1: A complete nerve for the elements to be generalised.

**Example 35.** *Let $E = \{e_1, e_2, e_3\}$ where $e_1 = c^5a^3b^3$, $e_2 = c^5a^2d^4$ and $e_3 = a^3b^3d^4c^5$ are the elements to be generalised. Initially, we are going to fix a nerve for these elements, namely, the complete nerve (see Figure 8.1).*

*The pattern $p = V^{10}c^5V^6$ generalises $E$, and $Seq(p)$ is a mcs of the lists in $E$. However, this pattern is not a distance-based pattern of $E$ since, for example, the element $a^3b^3$ (which is between $e_1$ and $e_3$) and the element $a^2d^4$ (which is between $e_2$ and $e_3$) are not covered by $p$. As a matter of fact, none pattern containing the ground symbol $c$ will be distance-based and this result is independent of the nerve we choose.*

The explanation for this apparently counterintuitive result is based on how the distance between the different pairs of elements $e_i$ and $e_j$ is calculated. Although all the lists in $E$ have the subsequence $c^5$ in common, this subsequence is never taken into account to compute the distance $d(e_i, e_j)$, for any pair $(e_i, e_j)$ in $N(E)$.

Therefore, we need to find an alternative way to define $n$-ary distance based operators, which is not exclusively based on the concept of *mcs*. The definition we propose next not only use the concept of *mcs* but also uses others such as the $\uparrow$-transformation and the concept of nerve which ensures the condition of being distance-based. To this end, we will first solve the problem for binary generalisations, and then we extend it for an arbitrary $n$-ary generalisations.

Initially, for any two elements $e_1$ and $e_2$ to be generalised, we need to know what conditions a pattern in $\mathcal{L}_0$ must satisfy in order to cover $e_1$, $e_2$ and the intermediate elements. Note that every pattern in $\mathcal{L}_0$ fixes a maximum length over the elements that the pattern covers. Thus, we could start with fixing two elements and see the maximum length their intermediate elements can achieve. This is equivalent to adapt Proposition 8 for sets to lists. The result is the proposition below.

**Proposition 19.** *Given the elements $e_1$, $e_2$, and $e_3$, if $e_3$ is between $e_1$ and $e_2$, then $e_3$ has a subsequence $s$ such that $s$ is a mcs of $e_1$ and $e_2$, and $|e_3| \leq |e_1| + |e_2| - |s|$.*

*Proof.* According to Proposition 5 in [43], $e_3$ necessarily has a subsequence, namely $s$, which matches one of the *mcs* of $e_1$ and $e_2$. Next, let us demonstrate that the length of $e_3$ satisfies the inequality, otherwise $e_3$ would not be in between. At the moment, we can write the following equalities:

$$\begin{array}{lcl} d(e_1, e_2) & = & |e_1| + |e_2| - 2 \cdot |s| \\ d(e_1, e_3) & = & |e_1| + |e_3| - 2 \cdot |s'| \\ d(e_3, e_2) & = & |e_3| + |e_2| - 2 \cdot |s''| \end{array} \tag{8.2}$$

where $|s'|$ and $|s''|$ are the length of the *mcs* of the pairs $e_1$ and $e_3$, and $e_3$ and $e_2$, respectively. Sure, $|s'| \leq |e_1|$ and $|s''| \leq |e_2|$. Now, let us suppose that the inequality stated by the proposition does not hold, that is, $|e_3| > |e_1| + |e_2| - |s|$. Then,

$$\begin{array}{rcl} d(e_1,e_3)+d(e_3,e_2) & = & |e_1|+|e_2|+2\cdot|e_3|-2\cdot|s'|-2\cdot|s''| \\ & > & |e_1|+|e_2|+2\cdot|e_3|-2\cdot|e_1|-2|e_2| \\ & > & |e_1|+|e_2|+2\cdot(|e_1|+|e_2|-|s|)-2\cdot|e_1|-2\cdot|e_2| \\ & = & |e_1|+|e_2|-2|s| \\ & = & d(e_1,e_2) \end{array} \tag{8.3}$$

But $d(e_1,e_3)+d(e_3,e_2) > d(e_1,e_2)$ is not possible since $e_3$ is between $e_1$ and $e_2$. □

Unfortunately, this latter proposition is not restrictive enough since a lot of patterns must still be examined in order to define a distance-based operator. Let us consider the following example:

**Example 36.** *Let $e_1 = cacab$ and $e_2 = d^2abd^2$ be the elements to be generalised. The sequence $ab$ is the mcs of $e_1$ and $e_2$. According to Proposition 19, those elements between $e_1$ and $e_2$ have the subsequence $ab$ and contain 9 or less symbols. In consequence, those patterns which are candidate to cover them are:*

$$abV^7,\ aVbV^6,\ aV^2bV^5,\ldots,V^7ab$$

*However, all of these patterns are not necessary. For instance, at first sight, we find that the pattern $abV^7$ is completely useless since neither $e_1$ nor $e_2$ are headed by the sequence $ab$. Indeed, from the previous set of patterns all the elements between $e_1$ and $e_2$ are only covered by the patterns $V^5abV^2$ and $V^3aV^2bV^2$.*

However, if we want to have more practical distance-based operators, it is convenient to figure out how to discard as many unnecessary patterns as possible. Note that the huge amount of potential patterns suggested by Proposition 19 is because this proposition says nothing about the position of the variables in the patterns. If we take the position of the variables in the pattern, as illustrated in the example above, more unhelpful patterns can easily be recognised. This observation leads us to introduce a more restrictive result for lists based on the concept of optimal alignment patterns.

**Proposition 20.** *Given the elements $e_1$, $e_2$ and $e$, if $e$ is between $e_1$ and $e_2$ then there exists an optimal alignment pattern $p$ of $e_1$ and $e_2$ such that $e \in Set(p)$.*

*Proof.* Let $M_e^{e_1}$ and $M_{e_2}^{e}$ be the optimal alignments of $e_1$ with $e$ and $e$ with $e_2$, respectively. We define the mapping $M$ between $e_1$ and $e_2$ as

$$M = M_e^{e_1} \circ M_{e_2}^{e} \tag{8.4}$$

The goal is to prove first that $M$ is an optimal alignment of $e_1$ with $e_2$ and then, see that the pattern $p_M$ covers $e$.

**(1st part)** According to Lemma 3, we know that the mapping $M$ is an alignment of $e_1$ with $e_2$. Thus, it remains to prove that $M$ is optimal. We start expressing the length of $e_1$ and $e_2$ in terms of $M$. That is,

$$\begin{array}{rcl} |e_1| & = & |Dom(M_e^{e_1}) - Dom(M)| + |Dom(M)| + r_1 \\ |e_2| & = & |Im(M_{e_2}^{e}) - Im(M)| + |Im(M)| + r_2 \end{array} \tag{8.5}$$

where $r_1$ (respectively, $r_2$) is the number of symbols in $e_1$ (respectively, $e_2$) which do not belong to $Dom(M_e^{e_1})$ (respectively, $Im(M_{e_2}^{e})$). We set the following equalities:

$$\begin{array}{lcl} s & = & |mcs(e_1, e_2)| \\ s_1 & = & s - |Dom(M_e^{e_1}) - Dom(M)| \\ s_2 & = & s - |Im(M_{e_2}^e) - Im(M)| \\ h & = & |Dom(M)| = |Im(M)| \end{array} \tag{8.6}$$

According to Equations (8.5) and (8.6), the distance between $e_1$ and $e_2$ can be expressed as,

$$\begin{array}{rcl} d(e_1, e_2) & = & |e_1| + |e_2| - 2 \cdot s \\ & = & -s_1 - s_2 + r_1 + r_2 + 2 \cdot h \end{array} \tag{8.7}$$

In a similar way, since $M_e^{e_1}$ and $M_e^{e_2}$ are optimal alignments, the distances $d(e_1, e)$ and $d(e, e_2)$ can be written as,

$$\begin{array}{lcl} d(e_1, e) & = & r_1 + r + |Dom(M_{e_2}^e) - Im(M_e^{e_1})| \\ d(e, e_2) & = & r_2 + r + |Im(M_e^{e_1}) - Dom(M_{e_2}^e)| \end{array} \tag{8.8}$$

where $r$ is the number of symbols in $e$ whose indices belong neither $Im(M_e^{e_1})$ nor $Dom(M_{e_2}^e)$. Given that $e$ is an element between $e_1$ and $e_2$ and after operating from (8.7) and (8.8), we obtain

$$2 \cdot h = 2 \cdot r + (s_1 + |Im(M_e^{e_1}) - Dom(M_{e_2}^e)|) + (s_2 + |Dom(M_{e_2}^e) - Im(M_e^{e_1})|) \tag{8.9}$$

We are going to simplify this latter expression. To do that, we must prove both equalities below:

$$s = s_1 + |Im(M_e^{e_1}) - Dom(M_{e_2}^e)| \tag{8.10}$$

$$s = s_2 + |Dom(M_{e_2}^e) - Im(M_e^{e_1})| \tag{8.11}$$

**Proving Equality (8.10):** it is equivalent to show that

$$|Dom(M_e^{e_1}) - Dom(M)| = |Im(M_e^{e_1}) - Dom(M_{e_2}^e)| \tag{8.12}$$

which is obtained by plugging (8.6) into (8.10). Since $|Dom(M_e^{e_1})| = |Im(M_e^{e_1})|$, it is enough to show that

$$|Dom(M_e^{e_1}) \cap Dom(M)| = |Im(M_e^{e_1}) \cap Dom(M_{e_2}^e)| \tag{8.13}$$

which is immediate. Recall that $Dom(M) = Dom(M_e^{e_1}) \cap Dom(M)$ and $|Im(M)| = |Im(M_e^{e_1}) \cap Dom(M_{e_2}^e)|$ (see Remark 1 in Lemma 3).

**Proving Equality (8.11)**: the proof is similar to the previous one. Thus, proving the equality (8.11) is equivalent to prove

$$|Im(M_{e_2}^e) - Im(M)| = |Dom(M_{e_2}^e) - Im(M_e^{e_1})| \tag{8.14}$$

which is obtained by plugging (8.6) into (8.11). Again, since $|Im(M_{e_2}^e)| = |Dom(M_{e_2}^e)|$, it is enough to show that

$$|Im(M_{e_2}^e) \cap Im(M)| = |Dom(M_{e_2}^e) \cap Im(M_e^{e_1})| \tag{8.15}$$

Note that $Im(M) = Im(M^e_{e_2}) \cap Im(M)$ and $|Dom(M)| = |Dom(M^e_{e_2}) \cap Im(M^{e_1}_e)|$ (see Remark 1 in Lemma 4). Now, substituting (8.10) and (8.11) into (8.9) gives

$$h = s + r \tag{8.16}$$

Necessarily $r$ vanishes ($r = 0$), otherwise $s$ would not be the length of the *mcs* of $e_1$ and $e_2$. Therefore, $h = s$ and $M$ is optimal.
**(2nd part)** The last step consists of showing that $e \in Set(p_{_M})$. We must distinguish two independent cases depending on the value of $s$:

- **Case 1 ($s = 0$):** it means that $M$ is the empty alignment and $p_{_M} = V^{|e_1|+|e_2|}$ (see Definition 21). According to Proposition 20, as $e$ is between $e_1$ and $e_2$, then $|e| \leq |e_1| + |e_2|$ and in consequence $e \in Set(p_{_M})$.
- **Case 2 ($s > 0$):** first, we map every symbol $Seq(p_{_M})(i)$ to $e(M^{e_1}_e((M)_{1i}))$. Then, we have to prove that the number and the distribution of the variable symbols in $p_{_M}$ is the correct one to cover those symbols in $e$ not taking part in the previous mapping. Since $r = 0$, it means that

$$\forall 1 \leq j \leq |e|,\ e(j) \in Im(M^{e_1}_e) \text{ or } e(j) \in Dom(M^e_{e_2}) \tag{8.17}$$

Now, depending on the value of $j$, only three cases can be given:

**First**, there exist $1 \leq i < Rang(M)$ such that

$$M^{e_1}_e((M)_{1i}) \leq j \leq M^{e_1}_e((M)_{1(i+1)}) \tag{8.18}$$

If $j \in Im(M^{e_1}_e)$ and applying the inverse alignment $M^e_{e_1}$ (see Lemma 4) over this latter inequality, we have that

$$(M)_{1i} \leq M^e_{e_1}(j) \leq (M)_{1(i+1)} \tag{8.19}$$

Additionally, note that (8.18) can also be rewritten in an equivalent way as

$$M^{e_2}_e((M)_{2i}) \leq j \leq M^{e_2}_e((M)_{2(i+1)}) \tag{8.20}$$

As the remaining possibility is that $j \in Dom(M^e_{e_2})$, in a similar way, by applying $M^e_{e_2}$ over (8.20) we have that

$$(M)_{2i} \leq M^e_{e_2}(j) \leq (M)_{2(i+1)} \tag{8.21}$$

Now, combining (8.19) and (8.21)

$$M^{e_1}_e((M)_{1(i+1)}) - M^{e_1}_e((M)_{1i}) \leq (M_{1(i+1)} - M_{1i}) + (M_{2(i+1)} - M_{2i}) \tag{8.22}$$

which implies that the number of symbols between $e(M^{e_1}_e((M)_{1i}))$ and $e(M^{e_1}_e((M)_{1(i+1)}))$ is equal to or less than the number of variable symbols between the ground symbols $Seq(p_{_M})(i)$ and $Seq(p_{_M})(i+1)$ in $p_{_M}$.

The **second** and **third** cases correspond to:

$$1 \leq j < M_e^{e_1}(M_{11}) \tag{8.23}$$

and

$$M_e^{e_1}(M_{1Rang(M)}) < j \leq |e|, \tag{8.24}$$

respectively. Reasoning in the same way leads to the number of symbols from $e(1)$ to $e(j)$ (respectively from $e(j)$ to the last symbol in $e$) is equal to or less than the number of variable symbols before the first (last) ground symbol in $p_M$. Hence, there exists a substitution $\sigma$ such that $e = p_{_M}\sigma$.

□

We will use the proposition above along with the $\uparrow$-transformation in order to define binary distance-based operators.

**Corollary 5.** *Given the elements $e_1$ and $e_2$, if $\{p_i\}_{i=1}^n$ is the set of all the optimal alignment patterns of $e_1$ and $e_2$, then the binary generalisation operator defined as*

$$\Delta^b(e_1, e_2) = \uparrow (p_1, \uparrow (p_2, \ldots \uparrow (p_{n-1}, p_n)) \ldots)$$

*is distance-based.*

*Proof.* For every optimal alignment pattern, we know from Corollary 18, that

$$Set(p_i) \subset Set(\Delta^b(e_1, e_2)) \tag{8.25}$$

Then, from Proposition 20, we can write that

$$\forall \text{ element } e \text{ between } e_1 \text{ and } e_2 \ \Rightarrow \exists p_i : e \in Set(p_i) \tag{8.26}$$

Now, combining (8.25) and (8.26), we can affirm that

$$\forall \text{ element } e \text{ between } e_1 \text{ and } e_2 \ \Rightarrow e \in Set(\Delta^b(e_1, e_2)) \tag{8.27}$$

Hence, the generalisation operator is distance-based. □

**Example 37.** *Let $e_1 = cacab$ and $e_2 = d^2abd^2$ be the elements used in the Example 36. According to that example, a huge amount of pattern should be considered in order to defined a distance-based generalisation of $e_1$ and $e_2$. However, according to Corollary 5, the only patterns to be considered are the two optimal alignment patterns*

$$p_1 = V^5abV^2 \text{ and } p_2 = V^3aV^2bV^2.$$

*Taking the patterns in the order above, we would have that $\Delta^b(e_1, e_2) = \uparrow (p_1, p_2) = V^5aV^2bV^2$.*

Note that Corollary 5 simply states a sufficient condition to define distance-based operators. This means that other operators, different to the one defined in this Corollary, can also be distance-based. This is shown in the next two examples.

**Example 38.** *It is not always necessary to consider all the optimal alignment patterns to obtain distance-based patterns. For example, given* $e_1 = ab$ *and* $e_2 = abab$*, only the three next optimal alignment patterns are possible:*

$$p_1 = abV_1V_2,\ p_2 = aV_1V_2b \text{ and } p_3 = V_1V_2ab.$$

*The elements between* $e_1$ *and* $e_2$ *are:*

$$e'_1 = aba,\ e'_2 = abb,\ e'_3 = aab \text{ and } e'_4 = bab.$$

*Observe that* $e'_1, e'_2 \in Set(p_1)$ *and* $e'_3, e'_4 \in Set(p_3)$*. Therefore a pattern* $p \in \mathcal{L}_0$ *such that* $Set(p_1) \cup Set(p_3) \subset Set(p)$*, for instance,* $p = \uparrow (p_1, p_2) = V_1V_2abV_3V_4$*, would be a distance-based pattern of* $e_1$ *and* $e_2$ *and observe that* $Set(p_2) \not\subset Set(p)$*.*

As shown, not all the optimal alignment patterns are necessary to define distance-based patterns. Consequently, another possibility that we can try to characterise binary *dbg* operators is if distance-based patterns are equal to or more general than a pattern obtained via $\uparrow$-transformation from a (sub)set of minimal patterns. However, this is not generally true either.

**Example 39.** *Let* $e_1 = ab^2ab^3$ *and* $e_2 = cb^2ac^2a$ *be the elements to be generalised. The only two possible optimal alignments are* $p_1 = V^2bbaV^6$ *and* $p_2 = V^2bbV^3aV^3$*. Neither* $p_1$ *nor* $p_2$ *are distance-based. That is, the element* $b^2ab^3c^2a \in Set(p_1)$ *is between* $e_1$ *and* $e_2$ *but does not belong to* $Set(p_2)$*. On the contrary, the element* $b^2c^2a \in Set(p_2)$ *is between* $e_1$ *and* $e_2$ *but does not belong to* $Set(p_1)$*.*

*In principle, both patterns* $p_1$ *and* $p_2$ *should be taken into account to define a distance-based pattern. Thus,* $p = \uparrow (p_1, p_2) = V^7bbV^3aV^6$*. But if we set* $p' = V^{10}bbV^{10}aV^5$*, it is easy to see that* $p$ *is distance-based and* $Set(p) \not\subset Set(p')$ *because* $Set(p_1) \subset Set(p)$ *but* $Set(p_1) \not\subset Set(p')$ *(e.g.* $bbab^6 \in Set(p_1)$ *but not in* $Set(p)$*). The explanation is given next* [2]*.*

*Since* $Set(p_2) \subset Set(p)$*, we have to show that the elements between* $e_1$ *and* $e_2$ *which are exclusively covered by* $p_1$ *are also covered by* $p$*. Focusing on how* $p_1$ *and* $p$ *are built, the elements covered by* $p_1$ *which could not be covered by* $p$ *are those requiring an instantiation of the last six variable symbols in* $p_1$*:*

$$bbab^3c^2a,\ bbac^2ab^3,\ b^2ab^2c^2ab, \ldots$$

*All these elements have the subsequence* $bbaa$ *(not necessarily continuous) in common. Matching the second* "a" *symbol in the common subsequence with the* "a" *symbol in* $p$*, then the second block of ten consecutive variables along with the last block of variables in* $p$ *could be instantiated in order to cover all these elements. Thus,* $p'$ *is distance-based.*

These two previous examples show that not only the distance-based operators can be defined in a different way that the one proposed by Corollary 5, but also that a characterisation of these operators (as we did for sets) is neither intuitive nor immediate. For this reason, the definition of $n$-ary distance-based generalisation operators is exclusively based on extending Corollary 5.

**Corollary 6.** *Given a finite set of elements* $E$ *and a nerve function* $N$*, the* $n$*-ary generalisation operator* $\Delta$ *defined in Algorithm 2 (where* $\Delta^b$ *is the binary distance-based operator defined in Corollary 5) is distance-based w.r.t.* $N$

[2] This example can even be more extreme, since we can find pairs of elements $e_1$ and $e_2$ where there exists a distance-based pattern $p$ of $e_1$ and $e_2$ such that, for every optimal alignment pattern $p_i$ of $e_1$ and $e_2$, $Set(p_i) \not\subset Set(p)$ (see Example 42).

*Proof.* For every $(e_i, e_j) \in N(E)$, $Set(\Delta^b(e_i, e_j)) \subset Set(\Delta(E))$ by the definition of the $\uparrow$-transformation. Therefore, for every finite set $E$, $\Delta(E)$ is distance-based w.r.t. $N(E)$. □

**Input**: $E = \{e_1, \ldots, e_n\}$, $\Delta^b$ (binary distance-based operator) and $N$ (nerve function)
**Output**: Distance-based pattern of $E$ w.r.t. $N(E)$

```
1  Δ(E)
2  begin
3      k ← 0;
4      L ← []/ * emptylist * /;
5      for (e_i, e_j) ∈ N(E) do
6          L[k] ← Δ^b(e_i, e_j);
7          k ← k + 1;
8      end
9      Sort L : ∀0 < i ≤ k, |Seq(L[i − 1])| ≤ |Seq(L[i])|;
10     p ← L[0];
11     for i = 1..k do  p ← ↑ (p, L[i]);
12     return p;
13 end
```

**Algorithm 2**: An algorithm to compute a distance-based generalisation of a set of lists $E$ w.r.t. a nerve function $N$.

Basically, Algorithm 2 returns a pattern $p$ such that for every pair of elements $(e_i, e_j) \in N(E)$, $Set(\Delta^b(e_i, e_j)) \subset Set(p)$, by iteratively applying the $\uparrow$-transformation over all the patterns $\Delta^b(e_i, e_j)$. Line number 9 in the algorithm is important since it ensures that $Seq(p) \neq \lambda$, if all the sequences $Seq(\Delta^b(e_1, e_j))$ have a subsequence in common. Let us illustrate this by means of an example:

**Example 40.** *Given $E = \{e_1, e_2, e_3, e_4\}$ where $e_1 = abc$, $e_2 = cabcd$, $e_3 = c$, $e_4 = cab$ and the nerve $N(E) = \{(e_1, e_2), (e_2, e_3), (e_2, e_4)\}$ depicted in Figure 8.2. The binary distance-based generalisations (lines 5-7 in the algorithm) are:*

$$\begin{array}{rcl} L[0] & = & \Delta^b(e_1, e_2) = VabcV \\ L[1] & = & \Delta^b(e_2, e_3) = V^3cV^4 \\ L[2] & = & \Delta^b(e_2, e_4) = VcabV \end{array}$$

*If we applied the $\uparrow$-transformation in any arbitrary order over the set of binary patterns, we could obtain for example:*

$$p = \uparrow (\uparrow (VabcV, VcabV), V^3cV^4) = \uparrow (V^2abV^2, V^3cV^4) = V^8$$

*However, if we sort the binary patterns as the algorithm indicates (line 9),*

$$\begin{array}{rcl} L[0] & = & V^3cV^4 \\ L[1] & = & VabcV \\ L[2] & = & VcabV \end{array}$$

*and then apply the $\uparrow$-transformation (lines 10-11), we will have:*

$$p = \uparrow (\uparrow (V^3cV^4, VabcV), VcabV) = \uparrow (V^3cV^4, VcabV) = V^3cV^4$$

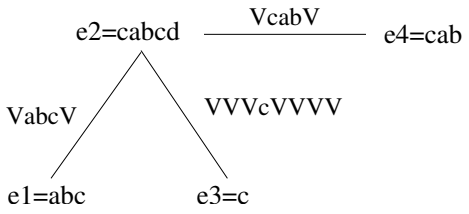


Figure 8.2: The nerve for the elements in $E$. Each edge is labelled with the binary distance-based generalisation of the elements in the vertices.

### $Mg$ operators in $(\mathcal{L}_0, k_0)$

Unlike what happened with $(\mathcal{L}_0, k_0)$ for sets, the determination of *mdbg* operators in this context is not trivial at all, even for the case of binary operators. Leaving aside the fact that the operator which minimises the cost function is not distance-based in general, we come across some other deeper problems which makes this task specially hard.

Basically, all the difficulties arise because the only way we know to define a distance-based operator is based on the notion of the $\uparrow$-transformation and the optimal alignment pattern (see Corollary 5). What we need to see then, is if the *mdbg* operators can be expressed in terms of these two concepts. However, this does not seem possible, as we will show next:

- The $\uparrow$-transformation is not adequate to guide the search: keeping in mind the pair $(\mathcal{L}_0, k_0)$ for sets, we could expect that a distance-based pattern of a pair of elements $\{e_1, e_2\}$ which contains as many ground symbols as possible, should be one of the minimal patterns. But this is not case. Indeed, the most surprising thing is that the value $k_0(\{e_1, e_2\}, p)$ (where $p$ is the so-mentioned pattern) can be greater than $k_0(\{e_1, e_2\}, p')$ for some distance-based pattern $p'$ having less ground symbols than $p$. Let us see an example:

  **Example 41.** *Given $e_1 = abcabab$ and $e_2 = d^{10}cabcd^{10}$ their optimal alignments are shown in the Table 8.3:*

| | | | | | | | | | | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| $e_1$ | = | | | a | b | c | | a | b | a | b | | $e_1$ | = | | a | b | c | a | b | a | b | | |
| $e_2$ | = | $d^{10}$ | c | a | b | c | $d^{10}$ | | | | | | $e_2$ | = | $d^{10}$ | | | c | a | b | | | c | $d^{10}$ |
| $e_1$ | = | | a | b | c | a | b | a | b | | | | $e_1$ | = | | a | b | c | a | b | a | b | | |
| $e_2$ | = | $d^{10}$ | | | c | a | | | b | c | $d^{10}$ | | $e_2$ | = | $d^{10}$ | | | c | | | a | b | c | $d^{10}$ |

Table 8.3: The four possible optimal alignments of $e_1$ and $e_2$.

*The correspondent optimal alignment patterns are:*

$$\begin{aligned} p_1 &= V^{11}abcV^{14} \\ p_2 &= V^{12}cabV^{13} \\ p_3 &= V^{12}caV^2bV^{11} \\ p_4 &= V^{12}cV^2abV^{11} \end{aligned}$$

*Let us have a look at the following patterns generalising* $\{e_1, e_2\}$*:*

$$\begin{aligned} p'_1 &= V^{15}aV^{16} \\ p'_2 &= V^{16}bV^{15} \\ p'_3 &= V^{14}cV^{16} \\ p'_4 &= V^{15}aV^2bV^{15} \\ p'_5 &= V^{28} \end{aligned}$$

*where*

$$\begin{aligned} k_0(\{e_1, e_2\}, p'_1) &= 4 \\ k_0(\{e_1, e_2\}, p'_2) &= 4 \\ k_0(\{e_1, e_2\}, p'_3) &= 3 \\ k_0(\{e_1, e_2\}, p'_4) &= 5 \\ k_0(\{e_1, e_2\}, p'_5) &= 27 \end{aligned}$$

*According to Corollary 5, all the patterns* $p'_i$ $(i = 1, \ldots, 5)$ *are distance-based because each one of them subsumes all the optimal alignment patterns of* $e_1$ *and* $e_2$ *(*$\{p_1, \ldots, p_4\}$*).*

*Now, let us see that there does not exist any other distance-based pattern* $p'$ *such that* $Seq(p') = ab$*, or* $Seq(p') = a$ *or* $Seq(p') = b$ *and* $c(\{e_1, e_2\}|p') < 4$*. We focus on the first case* $Seq(p') = ab$*. Necessarily, the pattern must fit the layout* $V^n aV^m bV^p$ *where* $n, p \geq 10$ *(*$e_2$ *is headed and ended by* 10 *ground symbols different to the symbols* $a$ *and* $b$*) and* $m \geq 0$*, otherwise* $p'$ *would not be distance-based. Now, the distance from* $e_1$ *to its nearest element* $e'_1$ *which is not covered by* $p'$ *is greater than or equal to* 3*, because all the elements placed nearer are covered by* $p'$*. For instance, we could set* $e'_1 = bcbb$*. Regarding* $e'_2$*, its distance w.r.t.* $e_2$ *is equal to* 1*. Altogether we have* $k_0(\{e_1, e_2\}, p') \geq 4$*. A similar reasoning can be applied for the two remaining cases, that is, when* $Seq(p') = a$ *and* $Seq(p') = b$*.*

*In conclusion, no distance-based pattern* $p'$ *of* $e_1$ *and* $e_2$ *such that* $Seq(p') = ab$ *(or* $Seq(p') = a$ *or* $Seq(p') = b$*) can be minimal. Thus, if* $p'$ *is a mdb pattern of* $e_1$ *and* $e_2$ *then* $Seq(p') = c$*.*

The explanation to this is as follows. In comparison with the subsequences $a$, $b$ and $ab$, the subsequence $c$ is the common subsequence of $e_1$ and $e_2$ which occurs fewer number of times. This means that, if we use $p'_3$, we have to remove fewer symbols from $e_1$ (respectively, $e_2$) in order to obtain the elements $e'_1$ (respectively, $e'_2$), and in consequence, the value $c(\{e_1, e_2\}|p'_3)$ is lower than $c(\{e_1, e_2\}, p'_1)$, $c(\{e_1, e_2\}, p'_2)$, $c(\{e_1, e_2\}, p'_4)$.

The fact that a subsequence can occur in a sequence more than once, causes that a pattern containing the longest sequence of ground symbols is not necessarily the minimal. For instance, the pattern $p'_3 = V^{14}cV^{16}$. This even happens for enhanced semantic cost functions such as the one taking also the furthest elements covered by the patterns into account.

A negative consequence of this is that the *mdb* pattern might not be reachable via the $\uparrow$-transformation neither over the optimal patterns nor the skeleton of the elements because

the ↑-transformation always recovers one of the *mcs* of two given sequences. For instance, observe that there is no manner to obtain a pattern such as $p = V^n cV^m$ employing the ↑-transformation neither over the set of optimal alignment patterns nor the skeleton of $e_1$ and $e_2$. Additionally, this limitation would even persist if our definition ↑-transformation satisfied the property 8.1.

Since the ↑-transformation turns out to be unsuitable for the *mdb* computation and we do not know other way to define distance-based operators without using the ↑-transformation, we have no option but performing an exhaustive-search to obtain a *mdb* operator. Obviously, to do that, we need to know what kind of patterns conforms the search space. This leads to the following problem:

- Few knowledge about which patters are in the search space: since we have to carry out an exhaustive search, we are interested in narrowing this search space as much as possible. In this sense, we could wonder if there always exists a *mdb* pattern which subsumes a group of optimal alignment patterns. If so, we could define a bottom-up search starting from the optimal alignment patterns! Unfortunately, we do not know whether this property is satisfied or not. We conjecture that this might not be possible because there exist distance-based patterns (although not minimal) not satisfying this condition (see previous Example 39). Here, we present a more extreme example in that the distance-based pattern does not subsume any of the optimal alignment patterns.

  **Example 42.** *Given $e_1 = ab^2ab^3$ and $e_2 = cb^2ac^2a$. The two possible alignment patterns are shown in the Table 8.4: The corresponding optimal alignment patterns are $p_1 = V^2bbVaV^6$ and*

| | |
|---|---|
| $e_1$ = a b b a b b b<br>$e_2$ = c b b a c c a | $e_1$= a b b a b b b<br>$e_2$= c b b a c c a |

Table 8.4: The two possible optimal alignments of $e_1 = ab^2ab^3$ and $e_2 = cb^2ac^2a$.

*$p_2 = V^2bbV^3aV^3$. The pattern $p = V^8bV^2bV^6$ is distance-based but neither $Set(p_1) \subset Set(p)$ nor $Set(p_2) \subset Set(p)$ (e.g. $e = b^2a^7$ is covered by $p_1$ and by $p_2$ but not by $p$). Note that $p$ covers all the elements between $e_1$ and $e_2$. The problem can be simplified in such a way that only few elements, from all the elements in between, must be analysed. These elements are those having more than 6 symbols after the first subsequence bb, since only 6 variables contain the pattern $p$ after the subsequence bb. Concretely, the worst case is $b^2ac^2ab^3$, but it is covered if we consider the substitution:*

$$\begin{array}{rcccccccccccc} p & = & V & V & V & V & V & V & b & V & V & b & V^6 \\ & & b & b & a & c & c & a & b & b & & b & \end{array}$$

*Hence, $p$ is a distance-based pattern of $e_1$ and $e_2$ such that $Set(p_1) \not\subset Set(p)$ and $Set(p_2) \not\subset Set(p)$.*

In any case, the uncertainty about the relationship between the *mdbg* patterns and the optimal alignment patterns makes the search space broader. This drawback is considered by the top-down

algorithm that we propose next (see Algorithm 3). Although this algorithm runs a top-down search for the *mdbg* pattern of a set $E$, we must point out that a bottom-up search can also be defined by taking the skeleton as the starting point for the exploration as we propose for the case $(\mathcal{L}_1, k_1)$.

```
   Input: E = {e_1, ..., e_n}
   Output: mdb pattern of E
 1 Δ(E)
 2 begin
 3     l ← length of the maximum common subsequence of E;
 4     k ← max{|e_i| + |e_j| : ∀e_i, e_j ∈ E};
 5     toExplore ← {V^{l+k}} //Set of patterns;
 6     searchSpace ← ∅;
 7     while toExplore ≠ ∅ do
 8         Remove a pattern p from the set toExplore;
 9         searchSpace ← searchSpace ∪ {p};
10         for each variable symbol V in p do
11             for s ∈ {λ} ∪ Σ do
12                 p' ← p{V/s};
13                 if p' is distance-based then toExplore ← toExplore ∪ {p'};
14             end
15         end
16     end
17     return argmin_{∀p∈searchSpace} k_0(E, p);
18 end
```

**Algorithm 3**: An exhaustive-search top-down algorithm to compute the *mdb* pattern of a set of elements $E$.

In order to prove that the algorithm above finds the *mdb* pattern of a set of elements $E$, it is enough to check that, at least, one of the *mdb* patterns is not missed during the search. Put in different words, we have to see that there exists a *mdb* pattern $p$ of $E$ such that there exists a substitution $\sigma$ where $p = V^{l+k}\sigma$.

**Lemma 5.** *Algorithm 3 finds the mdb pattern of a set of elements $E$.*

*Proof.* If $p$ is a *mdb* pattern of $E$, then $p$ has the form of:

$$p = V^{n_1} s_1 \dots V^{n_r} s_r V^{n_{r+1}} \tag{8.28}$$

where

$$r \leq l \text{ and } s_i \in \{\lambda\} \cup \Sigma, \forall 1 \leq i \leq k \tag{8.29}$$

According to Proposition 19, every continuous sequence of variables in $p$ has necessarily a length less than $max\{|e_i| + |e_j| : \forall e_i, e_j \in E\}$ and in consequence

$$|p| \leq l + k \tag{8.30}$$

Combining (8.29) and (8.30), we have that there exists a substitution $\sigma$ such that $p = V^{l+k}\sigma$. □

Let us illustrate briefly how this algorithm works:

**Example 43.** *Given $e_1 = ab$ and $e_2 = bc$ over the alphabet $\Sigma = \{a, b, c\}$. Clearly, the length of the greatest common subsequence is 1. The search space is made up of:*

$$searchSpace = \{V^5, aV^4, bV^4, cV^4, VaV^3, VbV^3, VcV^3, \ldots, V^3, \ldots, VaV, VbV, VcV\}$$

*The patterns with the form $V^ibV^j$ where $(i, j \geq 1)$ are just the mg patterns of $E$ since $k_0(E, p) = 2$ and some of them (those whose length is less than $l + k = 5$) are included in the search space. Hence, one of the mg patterns is returned by the algorithm.*

Of course, the algorithm above is not feasible from a computational point of view. We could mention, among other immediate reasons, that the search space has a exponential size with the length of the longest sequences in $E$ and the number of the ground symbols in the alphabet, the operation in line 3 is *NP-Hard*, etc.

Therefore, the only possibility to put this schema into practise is to approximate the *mdb* operator instead of obtaining an exact definition. In this sense, if we restrict ourselves to approximate the *mdbg* operator relative to a nerve function and if we consider that a reasonable approximation to the *mdb* pattern could be a distance-based pattern having the longest common subsequence of all the optimal patterns, then Algorithm 2 can be viewed as proposal. On the contrary, if we want a method which explicitly takes the cost function into account, we can always propose a greedy search algorithm guided by the cost function in a similar way we did for sets.

## 8.4 Distance-based Operators in $\mathcal{L}_1$

In order to define distance-based operators in $\mathcal{L}_1$, we adapt the operator $\Delta_N$ in $\mathcal{L}_1$ for lists by using the binary operator $\Delta^b$ in $\mathcal{L}_0$ for lists (see Corollary 5). An example about how this operator works is shown next:

**Example 44.** *Given a finite set of elements $E = \{e_1, e_2, e_3, e_4\}$ where $e_1 = a^2b^2d$, $e_2 = da^2c^2$, $e_3 = c^2db^2$ and $e_4 = ad$ and the nerve depicted below.*

$$\begin{array}{lcccl} \Delta^b(e_1, e_2) & = & p_1 & = & Va^2V^5 \\ \Delta^b(e_1, e_3) & = & p_2 & = & V^5b^2 \\ \Delta^b(e_1, e_4) & = & p_3 & = & VaV^3d \end{array}$$

*Finally,*

$$\Delta_N(E) = Va^2V^5 + V^5b^2 + VaV^3d$$

Observe that the previous example consists of a slight variation of Example 35 where the only admissible distance-based patterns in $\mathcal{L}_0$ of $E$ are just made up of variable symbols. Next, let us see how to obtain *mdbg* operators in $\mathcal{L}_1$.

### *Mdbg* operators in $(\mathcal{L}_1, k_1)$

Since the only way we know to define a distance-based operator in $\mathcal{L}_1$ consists of fixing a nerve beforehand, it is reasonable to study how to define *mdbg* operators relative to a nerve function. The

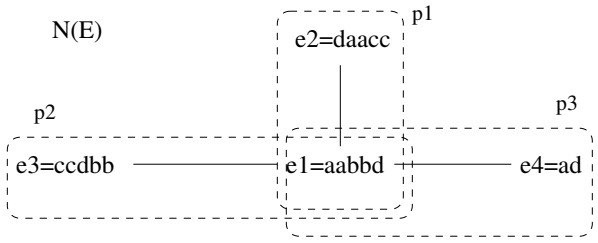


Figure 8.3: Fixing a nerve and a binary operator to define $\Delta_N(E)$.

calculus of the *mdbg* operator is not easy at all. Basically, the same problems that we found in order to define *mdbg* operators in $(\mathcal{L}_0, k_0)$, are present in $(\mathcal{L}_1, k_1)$ as well.

In this line, the first thing we can affirm is that if a generalisation operator $\Delta$ minimises $k_1(E, \cdot)$, then $\Delta$ is not distance-based. A counterexample is given next:

**Example 45.** *Given $e_1 = abba$ and $e_2 = acca$ the shortest patterns which cover both elements are $p_1 = aV^2a$, $p_2 = aV^3$, $p_3 = V^3a$ and $p_4 = V^4$. We have that $k_1(E, p_i) = 6$ $(i = 1, 2, 3, 4)$. None of them is distance-based since the element $e = abbcca$ is between $e_1$ and $e_2$ and is not covered by any of the latter patterns.*

*Let us check that there does not exist any distance-based pattern $p$ of $e_1$ and $e_2$ in $\mathcal{L}_1$ such that $k_1(E, p) \leq 64$. Setting $p = \sum_{i=1}^{n} p_i$, by definition $c'(E|p) \geq 2$ and since $e \in Set(p)$, there will exist a subpattern $p_i$ such that $e \in Set(p_i)$. Hence $|p_i| \geq 6$ and $k_1(E, p) > 6$.*

Thus, we need another way to undertake the problem. Let us see how to proceed. Observe that an operator like $\Delta_N$ tends to overfit $E$. In other words, $\Delta_N$ will compute, in general, patterns with a high $k_1(E, p)$ score. However, if we use the $\uparrow$-transformation over $\Delta_N$, we might obtain a new distance-based operator which attains a better value for $k_1(E, \cdot)$ as the following Example 46 shows.

**Example 46.** *Let $E$ and $\Delta_N(E)$ be the set and the generalisation operator employed in Example 44. We obtain a new pattern $p$ by applying the $\uparrow$-transformation over $\Delta_N(E)$ as follows:*

$$p = \uparrow (Va^2V^5, VaV^3d) + V^5b^2 = VaV^6 + V^5b^2$$

*Clearly $p$ is distance-based and $k_1(E, p) = 19$ which is lower than $k_1(E, \Delta_N(E)) = 25$.*

Now, the question is whether the *mdbg* operators relative to a nerve function $N$ can be defined in terms of $\Delta_N$ and the $\uparrow$-transformation. However, this result seems hard to be established. On the one hand, we ignore how to explicitly define most of the $\Delta^b$ operators (since Corollary 5 is just a sufficient condition) and on the other hand, we must take into consideration the inherent limitations of the $\uparrow$-transformation:

1. The *mdb* pattern might not be found by applying the $\uparrow$-transformation over $\Delta_N$ if this one uses the binary operator $\Delta^b$ defined in the Corollary 5: we will illustrate this by means of an example which is the result of adapting to lists the Example 19 employed for sets in $\mathcal{L}_1$.

   **Example 47.** *Given the set $E = \{e_1, e_2, e_3\}$ where $e_1 = a_1a_2a_3$, $e_2 = a_1a_6a_7$ and $e_3 = a_2a_4a_5$ being $N(E) = \{(e_1, e_2), (e_1, e_3)\}$. The optimal alignment patterns which are associated to*

*$(e_1, e_2)$ and $(e_1, e_3)$, respectively, are $a_1V^4$ and $Va_2V^3$. Then $a_1V^4$ is a db pattern of $(e_1, e_2)$ and $Va_2V^3$ is a db pattern of $(e_2, e_3)$ and i.e. the pattern $p = a_1V^4 + Va_2V^3$ is db w.r.t. $N(E)$. However, pattern*

$$p' = a_1V^4 + a_2V^3$$

*is distance-based (the only element between $e_1$ and $e_2$, which is not covered by $a_2V^3$, is $a_1a_2a_4a_5$ but this is covered by $a_1V^4$) but $Set(p') \not\subset Set(p)$. The mdb pattern for $E$ will have $|p'|$ or even fewer symbols and this will never be achieved by the $\uparrow$-transformation over the optimal alignment patterns.*

Therefore, given that $\Delta^b$ is defined from the concept of optimal alignment patterns and $\Delta_N$ is defined from $\Delta_b$, it is not possible that the *mdbg* operator can be expressed in terms of the $\uparrow$-transformation and $\Delta_N$.

2. The *mdbg* pattern might not be found by applying the $\uparrow$-transformation over $skeleton(N(E))$: from the previous point, we could think that the *mdb* pattern cannot be found because the optimal alignment patterns are excessively general. But this is not completely true. If it was so, it would mean that starting the search from something extremely specific, namely the skeleton, the *mdb* pattern should be found. However, this is not true as the next example reveals:

   **Example 48.** *Given $E = \{e_1, e_2, e_3, e_4, e_5\}$ where $e_1 = ac^3b^2$, $e_2 = ab^2$, $e_3 = ab^2ce$, $e_4 = d$ and $e_5 = fgh$ and the nerve depicted below:*

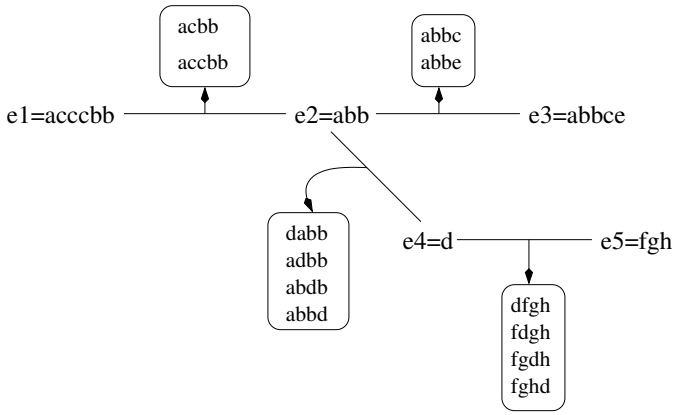


Figure 8.4: A naive generalisation of the set $E$ w.r.t. the nerve N(E). Circled elements are the intermediate elements.

*If we group the elements according to its similarity and then apply the $\uparrow$-transformation over the different groups, the pattern obtained would attain a lower value for $k_1(E, \cdot)$. Taking this strategy into account, we can distinguish several meaningful grouping criteria. For instance, those elements which contain the subsequence abb ($G_1$) and those which do not ($G_2$). That is,*

$$\begin{aligned} G_1 &= \{ac^3b^2, acb^2, ac^2b^2, ab^2, ab^2c, ab^2e, ab^2ce, dab^2, adb^2, abdb, ab^2d\} \\ G_2 &= \{dfgh, fdgh, fgdh, fghd\} \end{aligned}$$

*In this particular case, it does not matter how the elements in the groups are ranked in order to apply the ↑-transformation since the final result remains invariable. Thus, we can write*

$$p_1 = \uparrow(G_1) + \uparrow(G_2) = VaV^3bVbV^2 + VfVgVhV$$

*For any other binary splitting, we would have elements having no subsequence in common in the same group (e.g. abb and dfgh). The shortest patterns we would be*

$$\begin{array}{rcl} p_2 & = & aV^3b^2V^2 + V^4 \\ p_3 & = & V^6 \end{array}$$

*Using three groups, another interesting possibility can be explored. For instance, $G_1 = \{fgh\}$, those elements containing the subsequence d ($G_2$) and the remaining ones ($G_3$). Depending on the order of the elements in $G_2$ we could obtain*

$$\begin{array}{rcl} p_4 & = & fgh + \uparrow(\uparrow(\uparrow(abbd, dabb), d), \ldots,) + \uparrow(G_3) = V^5 + aV^3b^2V^2 \\ p_5 & = & fgh + \uparrow(\uparrow(abbd, d), \ldots,) + \uparrow(G_3) = V^3dV^3 + aV^3b^2V^2 + fgh \end{array}$$

*Finally, it is not worth to use more than three groups because of the excessive length of the pattern obtained. Evaluating the different patterns, we have that:*

$$\begin{array}{rcllcl} k_1(E, p_1) & = & c(p_1) + (c'(e_1|p_1) + \ldots c'(e_5|p)) & = & 17 + (1+1+1+1+1) & = & 22 \\ k_1(E, p_2) & = & c(p_2) + (c'(e_1|p_1) + \ldots c'(e_5|p)) & = & 12 + (1+2+1+4+2) & = & 22 \\ k_1(E, p_3) & = & c(p_3) + (c'(e_1|p_1) + \ldots c'(e_5|p)) & = & 6 + (1+4+2+6+4) & = & 23 \\ k_1(E, p_4) & = & c(p_4) + (c'(e_1|p_1) + \ldots c'(e_5|p)) & = & 13 + (1+3+1+5+3) & = & 26 \\ k_1(E, p_5) & = & c(p_5) + (c'(e_1|p_1) + \ldots c'(e_5|p)) & = & 18 + (1+1+1+1+1) & = & 23 \end{array}$$

*But the following patterns are also distance-based of E:*

$$\begin{array}{rcl} p_6 & = & V^3cV^2 + V^4 \\ p_7 & = & aV^5 + V^4 \end{array}$$

*where*

$$\begin{array}{rcllcl} k_1(E, p_6) & = & c(p_6) + (c'(e_1|p_1) + \ldots c'(e_5|p)) & 10 + (1+2+1+4+2) & = & 20 \\ k_1(E, p_7) & = & c(p_7) + (c'(e_1|p_1) + \ldots c'(e_5|p)) & 10 + (1+2+1+4+2) & = & 20 \end{array}$$

*However, neither $p_6$ nor $p_7$ can be derived from a ↑-transformation since this tends to extract the longest common subsequence. Observe that all the elements which have the subsequence c or a also contain the subsequence abb in common.*

From this previous analysis, we can conclude that the ↑-transformation is not enough in itself to explore the search space. We need a generalisation tool which is not based on the concept of the longest common subsequence. For this purpose, we introduce the so-called inverse substitution.

**Definition 24.** ***(Inverse substitution)*** *Given a pattern p in $\mathcal{L}_0$ or in $\mathcal{L}_1$ an inverse substitution $\sigma^{-1}$ is a set of indices where each index denotes a ground symbol in p to be changed by a variable. Thus, $p\sigma^{-1}$ represents the new pattern which is obtained by applying $\sigma^{-1}$ over p.*

Basically, an inverse substitution just changes ground symbols by variables. For example, given $p = VaabV$ and $\sigma^{-1} = \{2, 4\}$ then $p\sigma^{-1} = V^2aV^2$. Now, we are in conditions to introduce the next proposition:

**Proposition 21.** *Given a finite set of elements $E = \{e_1, \ldots, e_n\}$ and a nerve function $N$. If we set $S = skeleton(N(E))$ then there exists a partition $P$ of the set $S$ and a collection of inverse substitutions $\{\sigma_1^{-1}, \ldots, \sigma_n^{-1}\}$ such that the pattern*

$$p = \sum_{\forall P_i = \{e_{k_i}\}_{k_i=1}^{m_i} \in P} \uparrow (\{e_{k_i}\sigma_{k_i}^{-1}\}_{k_i=1}^{m_i})$$

*is a mdb pattern of $E$ relative to $N(E)$.*

*Proof.* The proof is nearly identical to the proof of the Proposition 14 for sets. Once we have the table of elements induced by a *mdb* pattern $p' = \sum_{i=1}^{n} p'_i$ (see Table 8.5) and the repeated elements have been removed, we have to see that there exists a set of inverse substitutions $\{\sigma_{i_j,1}^{-1}, \ldots, \sigma_{i_j,m_j}^{-1}\}$ for the elements in the column $C_j$ such that

$$\forall 1 \leq j \leq n,\ Set(\uparrow (\{a_{i_j,1}\sigma_{i_j,1}^{-1}, \ldots\})) \subset Set(p'_j)$$

| $C_1$ | $C_2$ | $\ldots$ | $C_n$ |
|---|---|---|---|
| $a_{i_{1,1}}$ | $a_{i_{2,1}}$ | $\ldots$ | $a_{i_{n,1}}$ |
| $a_{i_{1,2}}$ | $a_{i_{2,2}}$ | $\ldots$ | $a_{i_{n,2}}$ |
| $\ldots$ | $\ldots$ | $\ldots$ | $\ldots$ |
| $a_{i_{1,m_1}}$ | $a_{i_{2,m_2}}$ | $\ldots$ | $a_{i_{n,m_n}}$ |

Table 8.5: The $i$-th column contains those elements in $S$ which are covered by the pattern $p'_i$ $(1 \leq i \leq n)$.

Given that $a_{i_j,k} \in Set(p'_j)$ then there exists a substitution $\sigma_{i_j,k}$ such that $a_{i_j,k} = p'_j\sigma_{i_j,k}$. The inverse substitution $\sigma_{i_j,k}^{-1}$ is derived by obtaining the indices of the ground symbols in $a_{i_j,k}$ which appear at the right-hand side of the pairs in $\sigma_{i_j,k}^{-1}$. Since all the set of patterns $\{a_{i_j,k}\sigma_{i_j,k}^{-1}\}$ satisfy Proposition 17, the order of the patterns can be ignored and we can write

$$p''_j = \uparrow (\{a_{i_j,1}\sigma_{i_j,1}^{-1}, a_{i_j,2}\sigma_{i_j,2}^{-1}, \ldots\})$$

Given that

$$Seq(p'_j) = Seq(p''_j),$$

if we set

$$p'_j = V^{q_1}s_1 \ldots V^{q_{n-1}}s_nV^{q_n}$$

then

$$p''_j = V^{r_1}s_1 \ldots V^{r_{n-1}}s_nV^{r_n}$$

where $q_t \geq r_t$ (for every $1 \leq t \leq n$) otherwise there would exist a pattern $a_{i_j,k}\sigma_{i_j,k}^{-1}$ which would not be covered by $p'_j$ and consequently there would exists an element $a_{i_j,k}$ not be covered by $p'_j$, but this is a contradiction.

Finally, we have that $c(p''_j) \leq c(p'_j)$ and since $c'(\cdot|\cdot)$ is an inclusion-preserving cost function then $c'(E|p''_j) \leq c'(E|p'_j)$. Given that $p'$ is a *mdb* pattern of $E$ relative to $N$ then we can conclude that $p''_j$ is also a *mdb* of $E$ relative to $N$. □

**(Remark 1)** Note that this proposition holds for $(\mathcal{L}_0, k_0)$ as well because $c(\cdot|\cdot)$ is an inclusion-preserving function and $c(E|p''_j) \leq c(E|p'_j)$.

This latter proposition leads to an exhaustive search algorithm in order to compute the *mbdg* operator. This algorithm turns out to be useless in general due to the size of the search space (the number of different possibilities for the partition of $skeleton(N(E))$ and substitutions). Hence, an alternative option is to approximate the calculus of the *mdb* patterns. To do this, it is immediate to adapt the greedy search schema proposed for sets to lists by using the binary operator $\Delta^b$ defined in Corollary 5 and the cost function $k_1$ for lists, as we show below.

**Input**: $E = \{e_1, \ldots, e_n\}$, $\Delta^b$ (binary *dbg* operator) and $N$ (nerve function)
**Output**: A pattern which approximates a *mdb* pattern of $E$ w.r.t. $N(E)$

1 $\tilde{\Delta}_N(E)$
2 **begin**
3  $k \leftarrow 1$;
4  **for** $(e_i, e_j) \in N(E)$ **do**
5  $p_k \leftarrow \Delta^b(e_i, e_j)$;
6  $k \leftarrow k + 1$;
7  **end**
8  $p = \sum_{k=1}^{n} p_k$;
9  **do**
10  $k_p \leftarrow k_1(E, p)$;
11  $p' \leftarrow argmin\{k_1(E, p_{ij}) : \forall 1 \leq i, j, \leq n, p_{ij} =\uparrow (\{p_i, p_j\}) + (p - p_i - p_j)\}$;
12  $k'_p \leftarrow k_1(E, p')$;
13  **if** $k_{p'} < k_p$ **then** $p \leftarrow p'$;
14  **while** $k_{p'} < k_p$
15  return $p$;
16 **end**
17 //The notation $p - p_i - p_j$ employed in the algorithm means all the patterns in $p$ except $p_i$ and $p_j$.;

**Algorithm 4**: A greedy-based algorithm which approximates the *mdbg* operator.

**Example 49.** *Let $E$ and $N(E)$ be the set of examples and the nerve employed in Example 44. Remember that,*

$$\begin{aligned} p_1 = \Delta^b(e_1, e_2) &= Va^2V^5 \\ p_2 = \Delta^b(e_1, e_3) &= V^5b^2 \\ p_3 = \Delta^b(e_1, e_4) &= VaV^3d \end{aligned}$$

*and*

$$p = Va^2V^5 + V^5b^2 + VaV^3d$$

*see lines* 4-8 *in the algorithm. Next, we have to apply the $\uparrow$-transformation over each pair of binary generalisations and we choose the one which attains a lower value of $k_1(E, \cdot)$ (see lines* 9-14*). In*

*our case, we must consider two possibilities:*

$$\begin{array}{rcl} p_1 & = & \uparrow (Va^2V^5, V^5b^2) + VaV^3d \quad = \quad V^8 + VaV^3d = V^8 \\ p_2 & = & \uparrow (Va^2V^5, VaV^3d) + V^5b^2 \quad = \quad VaV^6 + V^5b^2 \end{array}$$

*Since $k_1(E, p_2) = 19$ is less than $k_1(E, p_1) = 27$, we choose the pattern $p_2$. The process stops when the pattern cannot be further improved. Note that next iteration leads to*

$$\uparrow (VaV^6, V^5b^2) = V^8$$

*which performs worse than $p_2$. Therefore, the algorithm returns $p_2$.*

## 8.5 On the Complexity of Computing Minimal Patterns for Lists

This section attempts to analyse the complexity of computing minimal pattern for lists in languages endowed with the + operation. As we did for sets, we will work with a simplied version of $L_1$ (denoted by $\mathcal{L}'$) consisting of removing the variable symbols from the patterns. Namely, given $A = \{a_1, a_2, a_3, \ldots\}$ the set of ground symbols, we define the pattern language $\mathcal{L}'$ as follows,

$$\mathcal{L}' = \{\{\text{all the lists having the subsequence } A_1\}, \{\text{all the lists having the subsequence } A_2\}, \ldots\}$$

where $A_i$ is any finite sequence built from symbols in $A$.
Next we show the impossibility of computing minimal patterns in $(\mathcal{L}', k_1)$ in polynomial time.

**Proposition 22.** *Given a finite set of elements $E$, the optimisation problem*

$$\underset{\forall p \in \mathcal{L}' : E \subset Set(p)}{\operatorname{argmin}} \quad k_1(E, p)$$

*is $NP$-*Hard*.

*Proof.* We define a polynomial reduction from the Minimum Set Covering Problem (MSC in short) [127], which is a *NP-Hard* problem, to the optimisation problem $(E, \mathcal{L}', k_1)$ at hand.

Let $(S, \mathcal{S}, w)$ be an instance of the $MSC$ problem where $S$ is a finite set of elements, $\mathcal{S}$ a family of sets covering $S$ and $w$ is a weight function such that $w(S) = 1$ for every $S \in \mathcal{S}$. We endow $\mathcal{S}$ with a total order $\leq_{\mathcal{S}}$ and define the mappings:

$$\begin{array}{rccl} \phi_1 : & S & \rightarrow & E \\ & s_i & \rightarrow & e_i ='' \, symbol''_{S_{j_1}} \ldots'' \, symbol''_{S_{j_i}} \end{array}$$

where for every $S \in \mathcal{S}$ exists $r$ $(j_i \leq r \leq j_i)$ such that $S = S_{j_r}$ and for every $r$ $(j_1 \leq r \leq j_i)$ $S_{j_{r-1}} \leq_{\mathcal{S}} S_{j_r}$.

$$\begin{array}{rccl} \phi_2 : & \mathcal{S} & \rightarrow & \mathcal{L}' \\ & S_i & \rightarrow & p_i = \{\text{all the lists having } ''symbol''_{S_i}\} \end{array}$$

Let us see that if $C = \{S_{i_1}, \ldots, S_{i_t}\}$ is a covering of $S$ then $w(C) = k_1(E, \phi_2(C)) - |E|$ where

$$\phi_2(C) = \{\text{all the lists having } ''symbol''_{S_{i_1}}\} + \ldots + \{\text{all the lists having } ''symbol''_{S_{i_t}}\}$$

On the one hand we have,

$$w(C) = t$$

On the other hand, setting $p = \phi_2(C)$, we can write

$$c(p) = t$$

By definition $c'(E|p) = |E|$, hence $w(C) = k_1(E, \phi_2(C)) - |E|$. Now, it is clear that if $E \subset Set(p)$ where

$$p = \text{all the lists having } ''symbol''_{S_{i_1}} + \ldots + \text{all the lists having } ''symbol''_{S_{i_t}}$$

then $\{S_{i_1}, \ldots, S_{i_t}\}$ is a covering of $S$. We call this, the covering associated to a pattern. Therefore, if $p$ is a minimal pattern of $E$ then $c(p)$ is also minimal and the covering $C$ associated to $p$ is the minimal covering of $S$.

Therefore, solving our optimisation problem is, at least, as hard as solving the MSC problem. □

## 8.6 Discussion

In this chapter we have seen how to define (minimal) distance-based generalisation operators for sequences embedded in a metric space generated by a setup of the edit distances where substitutions are treated as a combination of insertions and deletions. The pattern language we have used is a subset of the regular languages. A very interesting extension would be the definition of these operators over the whole set of the regular languages. Roughly speaking, this could be done by adapting traditional grammar inference algorithms, for instance, by controlling their induction mechanism by means of the cost function $k$. In this way, we could obtain regular grammars consistent with the distances between the sequences we want to generalise, something which is not ensured if we apply traditional grammar learners directly.

# Chapter 9

# Generalising First-Order Objects Lying in a Metric Space

A great effort has been made in the field of inductive logic programming to establish the notion of generalisation on a formal basis as well as to study its applicability (generalisation/specialisation operators, order relations, refinement operators, etc.) in the development of inductive inference algorithms. However, and as mentioned in previous chapters, there is not much work concerning the relationship between generalisation and distances over first-order objects, except from the research done in [139] (where a distance for atoms is obtained from $lgg$).

In this chapter, we use our framework to precisely explore the connection between generalisation and distances in the context of first-order logic and to develop distance-based operators for several first-order representations (atoms and clauses).

## 9.1 Atoms

The space of atoms is defined over a signature $\langle \mathcal{C}, \mathcal{F}, \Pi, \mathcal{X} \rangle$, where $\mathcal{C}$ is a set of constants, $\mathcal{F}$ (and respectively $\Pi$) is a family that is indexed on $\mathbb{N}$ (non negative integers) with $\mathcal{F}_n$ ($\Pi_n$) being a set of $n-$adic function (predicate) symbols and $\mathcal{X}$ is a (infinite denumerable set of variable symbols). In the case of no ambiguity, both predicate and function symbols are referred to as symbols, and variable symbols are referred to as variables. $f/n$ (and respectively $p/n$) denotes a function symbol $f \in \mathcal{F}_n$ (and respectively $p \in \Pi_n$).

The goal of this section is to compute $mdbg$ operators for atoms embedded in a particular metric space. As usual, to do this, a distance function, a pattern language and a cost function are fixed next.

### 9.1.1 Distance, Pattern Language and Cost Function

The distance function we are going to employ is the one defined in [139] (the reader can find a brief description in Section 3.4.1 of Chapter 3). Recall that this distance returns an ordered pair of integer values $(i, j)$ instead of a single value. This is because differences between two atoms are expressed in terms of function symbols (first component) and variable symbols (second component).

Distances between atoms can be compared (e.g. we can say something like "atoms are nearer or further than...") thanks to a total relation order (a lexicographic order, indeed) defined over the set of pairs. Consequently, pairs of numbers can be treated as though they were single numbers and for this reason, all the definitions of our framework can be automatically extended for this special case.

In what follows, $(X_a, d_a)$ denotes the metric space of atoms where $X_a$ is the Herbrand base with variables induced by the signature, and $d_a$ denotes the distance mentioned above. The pattern language $\mathcal{L}$ to be employed matches the space of examples, that is, $\mathcal{L} = X_a$. In this way, given a pattern $p$, $Set(p)$ denotes all the atoms in $X_a$ which are instances of $p$.

**Example 50.** *Let $\mathcal{C} = \{a, b\}$ be a set of constants, $\mathcal{F} = \{f/1\}$ a set of function symbols, $\mathcal{V} = \{V_1, V_2, \ldots\}$ a denumerable set of variables and $\Pi = \{p/2, q/2\}$ a set of predicate symbols. Then,*

$$\mathcal{L} = \{p(a, V_1), p(V_1, a), p(V_1, V_2), p(f(a), b), \ldots q(a, X_1), q(X_1, a), \ldots\}$$

*Setting $p_1 = p(a, V_2)$,*

$$Set(p_1) = \{p(a, a), p(a, f(a)), p(a, f(f(a))), \ldots\}$$

Finally, the function $k$ will be any inclusion-preserving cost function. For instance, $k$ could be defined, among other possibilities, as $k(E, p) = c(p) + c(E|p)$ where $c(p) = (0, 0)$ and $c(E|p)$ is the first function in Table 5.2.

### 9.1.2 (Minimal) Distance-based Operators in $\mathcal{L}$

Some previous definitions and intermediate results are required before defining distance-based generalisation operators.

First, we need a manner to refer to subterms in a term/atom. For this purpose, a term/atom can be viewed as a labelled tree where each symbol in the term/atom corresponds to a node in the tree. Then, the position of a symbol in the term/atom is denoted by a sequence of natural numbers such as $\gamma = n_1.n_2.\cdots.n_k$ which identifies the corresponding node of the symbol in the tree. In addition, if $\gamma$ is a position in a term/atom $e$, then $e|^t_\gamma$ denotes the subterm $t$ in $e$ whose root functor symbol is placed at position $\gamma$ in $e$ while $e|^s_\gamma$ is just the root functor. For instance, if $e = p(f(a, b), c)$ then $e|^t_1 = f(a, b)$ and $e|^s_1 = f$. Following with notation, the $lgg$ between two elements $e_1$ and $e_2$ is denoted in this section (just for simplicity) by $L_{e_1,e_2}$.

Next, we introduce the symbol $\cdot$ as an artificial symbol which represents an anonymous variable in a term/atom. In other words, each dot in a term/atom denotes a variable different from the rest. This notation is adopted to indicate that we do not need to know the name of (some) variables. For instance, we could have atoms such as $p(\cdot, \cdot)$ or even $q(\cdot, V_1, V_1)$, etc. This is just syntactic sugar because $p(\cdot, \cdot)$ is equivalent to $p(V_2, V_1)$ and $q(\cdot, V_2, V_2)$ is the same that $q(V_1, V_2, V_2)$). Additionally, $lgg$ is well defined for atoms having dots. For instance, $lgg(p(\cdot, a, b, c, c), p(a, \cdot, a, b, b)) = p(\cdot, \cdot, V_1, V_2, V_2)$ which is equivalent to $p(V_1, V_2, V_3, V_4, V_4)$.

**Definition 25.** ***(Schema of an atom)*** *Let $e$ be an atom in $X_a$, the schema of $e$ (denoted by $sc(e)$) is a new atom obtained from $e$ by replacing each variable symbol in $e$ by a dot.*

For instance, given the atom $e = p(a, f(V_1), g(h(V_1)))$ then, $sc(e) = p(a, f(\cdot), g(h(\cdot)))$.

From this latter definition, we can see a schema as an atom whose all variable symbols are just denoted by dots. In what follows, we can speak of schemata without the need of specifying the atom the schema is from. When this happens, we refers s by $sc_1, sc_2, \ldots$

**Definition 26.** ***(Intersection of two schemata)*** *Given two schemata $sc_1$ and $sc_2$, the intersection of two schemata (denoted by $sc_1 \cap sc_2$) is $sc(lgg(sc_1, sc_2))$. If the predicate symbol in $sc_1$ and $sc_2$ is different then $sc_1 \cap sc_2 = \top$.*

For instance, given $sc_1 = p(a, \cdot, g(b))$ and $sc_2 = p(b, \cdot, g(b))$ then $sc_1 \cap sc_2 = p(\cdot, \cdot, g(b))$.

**Definition 27.** ***(Subschema)*** *Let $sc_1$ and $sc_2$ be two schemata. Then, we will say that the $sc_2$ is a subschema of $sc_1$ (denoted by $sc_2 \subset sc_1$), if $sc_2 = sc_2 \cap sc_1$. In the same way, the schemata $sc_1$ and $sc_2$ are equal (denoted by $sc_1 = sc_2$) if $sc_1 \subset sc_2$ and vice versa.*

For instance, the schema $sc_2 = p(a, \cdot, g(\cdot))$ is a subschema of $p(a, f(\cdot), g(h(\cdot)))$. Finally, say that symbols and subterms in a schema will also be referred using the same notation that the one used for terms/atoms.

Next result says how the schemata of three atoms $e_1$, $e_2$ and $e_3$, where $e_3$ is an intermediate element of $e_1$ and $e_2$, are related.

**Proposition 23.** *Let $e_1$, $e_2$ and $e_3$ be three elements in $X_a$. If $e_3$ is an intermediate element of $e_1$ and $e_2$ then:*
***i)*** *$sc(L_{e_1,e_2}) \subset sc(e_3)$.*
***ii)*** *For every subterm $t$ at a position $\gamma$ in $e_3$ there exists a subterm $t'$ at position $\gamma$ in $e_1$ or in $e_2$ such that $t = t'$ (modulo variable renaming).*

*Proof.* Let $d_a(e_1, e_2) = (F_1, V_1)$, $d_a(e_2, e_3) = (F_2, V_2)$ and $d_a(e_1, e_3) = (F_3, V_3)$ be the distances between these elements. The function $F$, which returns the number of non-variable symbols in an atom (see Subsection in 3.4.1), can perfectly be applied over schemata as well in such a way that dot symbols are not counted by $F$. According to the definition of $d_a$, we can write:

$$\begin{array}{rcl} F_1 & = & F(e_1) + F(e_2) - 2F(L_{e_1,e_2}) \\ F_2 & = & F(e_1) + F(e_3) - 2F(L_{e_1,e_3}) \\ F_3 & = & F(e_2) + F(e_3) - 2F(L_{e_2,e_3}) \end{array} \tag{9.1}$$

Given that $e_3$ is an intermediate element then,

$$\begin{array}{rcl} F_1 & = & F_2 + F_3 \Leftrightarrow \\ -F(L_{e_1,e_2}) & = & F(e_3) - F(L_{e_2,e_3}) - F(L_{e_1,e_3}) \end{array} \tag{9.2}$$

The second expression in (9.2) is obtained by plugging (9.1) into the first expression and doing some basic mathematics. Observe that common symbols in $L_{e_2,e_3}$ and $L_{e_1,e_3}$, that is, symbols in $sc(L_{e_2,e_3}) \cap sk(L_{e_1,e_3})$ are counted twice. To show this clearer, we rewrite the second expression in (9.2) as:

$$-F(L_{e_1,e_2}) = (F(e_3) - F(sc(L_{e_2,e_3}) \cap sc(L_{e_2,e_3})) - N - N') - F(sc(L_{e_2,e_3}) \cap sc(L_{e_2,e_3})) \tag{9.3}$$

where

$$N = F(sc(L_{e_2,e_3})) - F(sc(L_{e_2,e_3}) \cap sc(L_{e_1,e_3}))$$

counts non-variable symbols in $L_{e_2,e_3}$ which are not in $sc(L_{e_2,e_3}) \cap sc(L_{e_1,e_3})$, and

$$N' = F(sc(L_{e_1,e_3})) - F(sc(L_{e_2,e_3}) \cap sc(L_{e_1,e_3}))$$

counts non-variable symbols in $L_{e_1,e_3}$ which are not in $sc(L_{e_2,e_3}) \cap sc(L_{e_1,e_3})$. Given that $sc(L_{e_2,e_3}) \cap sc(L_{e_1,e_3}) \subset sc(L_{e_1,e_2})$, then

$$F(sc(L_{e_2,e_3}) \cap sc(L_{e_1,e_3})) \leq F(L_{e_1,e_2}) \equiv -F(sc(L_{e_2,e_3}) \cap sc(L_{e_1,e_3})) \geq -F(L_{e_1,e_2})$$

From definition of $N$ and $N'$ and since $sc(L_{e_2,e_3}) \cap sc(L_{e_1,e_3}) \subset sc(e_3)$, it is clear that

$$F(e_3) - F(sc(L_{e_2,e_3}) \cap sc(L_{e_1,e_3})) - N - N' \geq 0$$

Consequently, the only way the equation (9.3) holds, is that

$$F(sc(L_{e_2,e_3}) \cap sc(L_{e_1,e_3})) = F(L_{e_1,e_2}) \tag{9.4}$$

and

$$F(e_3) - F(sc(L_{e_2,e_3}) \cap sc(L_{e_1,e_3})) - N - N' = 0 \tag{9.5}$$

From Equation (9.4), we have that $sc(L_{e_1,e_2}) = sc(L_{e_1,e_3}) \cap sc(L_{e_2,e_3}) \subset sc(e_3)$ and i.e. statement $i$) is proved.

From Equation (9.5) and paying attention about how $N$ and $N'$ are defined, we have necessarily that $e_3|^t_\gamma = e_1|^t_\gamma$ or $e_3|^t_\gamma = e_2|^t_\gamma$ for every position $\gamma$ in $e_3$. This is just what statement $ii$) affirms. □

Next corollary, although immediate, will help us to understand the proof of the next proposition better.

**Corollary 7.** *Let $e_1$, $e_2$ and $e_3$ be three elements in $X_a$. If $e_3$ is an intermediate element of $e_1$ and $e_2$ then for every position $\gamma$ such that $sc(L_{e_1,e_2})|^s_\gamma$ is a dot then either $e_3|^t_\gamma = e_1|^t_\gamma$ or $e_3|^t_\gamma = e_2|^t_\gamma$.*

*Proof.* From Proposition 23 (statement $ii$)), it is known that $e_3$ is built up from terms in $e_1$ or in $e_2$. If $sc(L_{e_1,e_2})|^s_\gamma$ is a dot then this necessarily implies that $e_1|^t_\gamma \neq e_2|^t_\gamma$ and trivially, either $e_3|^t_\gamma = e_1|^t_\gamma$ or $e_3|^t_\gamma = e_2|^t_\gamma$. □

Once we know that if $e_3$ is an intermediate element of $e_1$ and $e_2$ then $e_3$ is necessarily built up from terms in $e_1$ or in $e_2$, it would be interesting to know whether there exists a relationship between terms in $e_3$ and variables in $L_{e_1,e_2}$ in such a way that identical variables in $L_{e_1,e_2}$ correspond to identical terms in $e_3$. This is shown in the next proposition.

**Proposition 24.** *Let $e_1$, $e_2$ and $e_3$ be three atoms such that $e_3$ is an intermediate element of $e_1$ and $e_2$. Let $Z$ be a variable with $n$ occurrences in $L_{e_1,e_2}$ at positions $\{\gamma_i\}_{i=1}^n$ then $e_3|^t_{\gamma_1} = \ldots = e_3|^t_{\gamma_n}$.*

To avoid any confusion with the weight function $V$ for variables on which the distance $d_a$ is based (recall that $V(t)$ returns the sum of the squared number of occurrences of each variable in the term/atom $t$), in the next proof variables are denoted by subscripted uppercase letters (e.g. $X_1, X_2, \ldots, Y_1, Y_2, \ldots, Z_1, Z_2, \ldots$).

*Proof.* Variables in $e_1$ are $\{X_1, \ldots, X_q\}$ and variables in $e_2$ are $\{Y_1, \ldots, Y_r\}$. Table 9.1 shows which terms in $e_1$ and $e_2$ are substituted by a variable when the $lgg$ between $e_1$ and $e_2$ is computed.

According to this table, $L_{e_1,e_2}$ has $p$ different variables $\{Z_1, \ldots, Z_p\}$ such that variable $Z_i$ occurs $n_i$ times in $L_{e_1,e_2}$ at positions $\{\gamma_{ij}\}_{j=1}^{n_i}$. Therefore,

$$V(L_{e_1,e_2}) = n_1^2 + \ldots + n_p^2$$

| Variables in $L_{e_1,e_2}$ | Number of occurrencies | Positions in $L_{e_1,e_2}$ | Terms in $e_1$ | Terms in $e_2$ |
|---|---|---|---|---|
| $Z_1$ | $n_1$ | $\{\gamma_{1j}\}_{j=1}^{n_1}$ | $t_1$ | $t'_1$ |
| $Z_2$ | $n_2$ | $\{\gamma_{2j}\}_{j=1}^{n_2}$ | $t_2$ | $t'_2$ |
| | | $\vdots$ | | |
| $Z_p$ | $n_p$ | $\{\gamma_{2j}\}_{j=1}^{n_p}$ | $t_p$ | $t'_p$ |

Table 9.1: Variable symbols in $L_{e_1,e_2}$ and terms in $e_1$ and $e_2$ which causes every variable occurrence in $L_{e_1,e_2}$.

For simplicity, we denote the number of occurrences of a variable in a term/atom by the following notation $\langle Variable, term\rangle$ (e.g. $\langle X, p(a,X,X,Y)\rangle = 2$ and $\langle Y, p(a,X,X,Y)\rangle = 1$).

Since $e_3$ is an intermediate element, we know from Proposition 23 (statement $i$)) and Corollary 7 that $e_3$ is built up from the schema $sc(L_{e_1,e_2})$ in such a way that dots in the schema are substituted by terms either in $e_1$ or in $e_2$. In this case, these terms necessarily are $\{t_1, t'_1, \ldots, t_p, t'_p\}$ and i.e. variables in $e_3$ are exclusively provided by these terms (or subterms of these terms).

Equations (9.6) express the number of times a term $t_i$ (equivalently $t'_i$) occurs in $e_3$. That is, the term $t_i$ appears $a_{i1}$ times in $e_3$ while $t'_i$ appears $a_{2i}$ times in $e_3$.

$$\begin{array}{rcl} n_1 & = & a_{11} + a_{12} \\ n_2 & = & a_{21} + a_{22} \\ & \vdots & \\ n_p & = & a_{p1} + a_{p2} \end{array} \tag{9.6}$$

Therefore, we can write

$$V(e_3) = \sum_{j=1}^{q} \Big( \sum_{i=1}^{p} a_{i1} \langle X_j, t_i \rangle \Big)^2 + \sum_{j=1}^{r} \Big( \sum_{i=1}^{p} a_{i2} \langle Y_j, t'_i \rangle \Big)^2 \tag{9.7}$$

According to how $e_3$ is built (Proposition 23) we have that,

$$V(L_{e_1,e_3}) = \sum_{i=1}^{p} a_{1i}^2 + \sum_{j=1}^{r} \Big( \sum_{i=1}^{p} a_{i2} \langle Y_j, t'_i \rangle \Big)^2 \tag{9.8}$$

and

$$V(L_{e_3,e_2}) = \sum_{j=1}^{q} \Big( \sum_{i=1}^{p} a_{i1} \langle X_j, t_i \rangle \Big)^2 + \sum_{i=1}^{p} a_{2i}^2 \tag{9.9}$$

Given that $e_3$ is an intermediate element and according how $d_a$ is defined, we can write

$$V(e_1) + V(e_2) - 2V(L_{e_1,e_2}) = V(e_1) + V(e_3) - 2V(L_{e_1,e_3}) + V(e_3) + V(e_2) - 2V(L_{e_3,e_2})$$

which is equivalent to

$$-V(L_{e_1,e_2}) = V(e_3) - V(L_{e_1,e_3}) - V(L_{e_3,e_2}) \tag{9.10}$$

Plugging Equations (9.7), (9.8) and (9.9) into (9.10), we have that

$$\sum_{i=1}^{p}(a_{i1}+a_{i2})^2 = \sum_{i=1}^{p} a_{i1}^2 + a_{i2}^2$$

which is possible only if

$$a_{i1}=0 \text{ and } n_i = a_{i2} \text{ or } a_{i2}=0 \text{ and } n_i = a_{i1},\ \forall 1 \leq i \leq p$$

This latter result allow us to affirm that

$$\forall 1 \leq i \leq p \text{ and } \forall \gamma \in \{\gamma_{ij}\}_{j=1}^{n_i},\ \text{then either } e_3|_{\gamma}^{t} = t_i \text{ or } e_3|_{\gamma}^{t} = t_i'$$

and consequently

$$\forall \gamma, \gamma' \in \{\gamma_{ij}\}_{j=i}^{n_i} \text{ then } e_3|_{\gamma}^{t} = e_3|_{\gamma'}^{t}$$

□

**Corollary 8.** *Given three elements $e_1$, $e_2$ and $e_3$ such that $e_3$ is an intermediate element of $e_1$ and $e_2$, then $e_3 \in Set(L_{e_1,e_2})$.*

*Proof.* From Proposition 23 (statement *i*)), $sc(L_{e_1,e_2}) \subset sc(e_3)$ and from Proposition 24, if $L_{e_1,e_2}|_{\gamma_1}^{s}$, $L_{e_1,e_2}|_{\gamma_2}^{s} \in \mathcal{V}$ and $L_{e_1,e_2}|_{\gamma_1}^{s} = L_{e_1,e_2}|_{\gamma_2}^{s}$ then $e_3|_{\gamma_1}^{t} = e_3|_{\gamma_2}^{t}$. Therefore, there exists a substitution $\theta$ such that $L_{e_1,e_2}\theta = e_3$ and i.e. $e_3 \in Set(L_{e_1,e_2})$. □

**Proposition 25.** *Given a finite set of elements $E \subset X_a$, the operator $\Delta(E) = lgg(E)$ is a minimal distance-based generalisation operator in $(\mathcal{L}, k)$.*

*Proof.* First, we prove that $\Delta$ is distance-based. Let $N$ be a nerve function. From Corollary 8, we have that,

$$\forall (e_i, e_j) \in N(E) \text{ and } \forall e_k \text{ intermediate element of } e_i \text{ and } e_j, e_k \in Set(L_{e_i,e_j})$$

Since $Set(L_{e_i,e_j}) \subset Set(lgg(E))$ then $e_k \in Set(lgg(E))$ and $\Delta$ is distance-based.

Second, let us see that $\Delta$ is a minimal. This is direct because for every pattern $p$ covering $E$, we have that $Set(lgg(E)) \subset Set(p)$ and given that $k$ is inclusion-preserving then $k(E, lgg(E)) \leq k(E, p)$ and i.e. $\Delta$ is minimal. □

**(Remark 1)** Note that $\Delta$ is a *mdbg* operator independent of the nerve function we take.

This latter result does not necessarily hold for every distance, cost function or pattern language. Things can be different when one of these elements is changed. Next two examples show that the *lgg* operator is not necessarily distance-based when a different metric is considered and is not necessarily minimal either when other cost functions are employed.

**Example 51.** *Let $d$ be the distance function for ground terms and atoms introduced in [125] and explained in Section 3.4.1 from Chapter 3 of this thesis. Given $e_1 = p(a,a)$ and $p(b,b)$, we know from the example reported in the explanation of this distance that $d(e_1, e_2) = 1/2$. Now, if we set $e_3 = p(a,b)$ we have that $d(e_1, e_3) = 1/4$ and $d(e_2, e_3) = 1/4$, thus $d(e_1, e_2) = d(e_1, e_3) + d(e_3, e_2)$ and we can affirm that $e_3$ is an element between $e_1$ and $e_2$ which is not covered by $lgg(e_1, e_2) = p(X, X)$. Therefore, lgg is not distance-based w.r.t. d. This makes sense since lgg performs an unnecessarily more restricted generalisation than the one induced by the distance because repeated term occurrences in an atom are disregarded by d.*

**Example 52.** *If we set $k_2(E,p) = c_2(E) + c(E|p)$ where $c_2(p) = (0, v)$ (denoting $v$ the number of different variable symbols in $p$. E.g. $c_2(p(X,Y,X)) = 2$) and $c(E|p)$ is the same function used above, then $lgg(E)$ is not a mdbg operator.*

*To show this, let us consider the set $E = \{e_1, e_2\}$ where $e_1 = p(f(a,b),a)$, $e_2 = p(f(b,a),a)$. From here, we have that $p = lgg(e_1, e_2) = p(f(X,Y),a)$. The nearest element $e_1'$ to $e_1$ (respectively to $e_2$) is the one obtained by changing a ground symbol by a new variable symbol. If we set $e_1' = p(f(a,b),X)$, the distance between $e_1'$ and $e_1$ is $d_a(e_1, e_1') = (1,-1)$. Given that there does not exist any other atom $e$ such that $d_a(e_1, e) < d(e_1, e_1')$ and $e_1'$ is not covered by $p$ then we can conclude that $c(e_1|p) = (1,-1)$. An identical reasoning leads to $c(e_2|p) = (1,-1)$. Therefore, $k_2(E,p) = (0,2) + (2,-2) = (2,0)$.*

*Now, let us consider the pattern $p' = p(X,a)$ which also covers $E$. Again, $e_1'$ is not covered by $p'$. Hence, we can affirm that $c(e_1|p') = c(e_2|p') = (1,-1)$ and consequently $k_2(E) = (0,1) + (2,-2) = (2,-1)$.*

*Clearly, $p'$ is a db pattern but $k_2(E,p') < k(E,p)$. Therefore the lgg operator, although distance-based, is not minimal in this case.*

## 9.2 First Order Clauses

Clauses allow us to express real-world objects more accurately than single literals do. A set of literals can represent not only the different parts of an object but also the relationships among them.

This section is devoted to determine *mdbg* operators for clauses. As usual, we need to establish a distance function, a pattern language and a cost function for this sort of data.

### 9.2.1 Distance, Pattern Language and Cost Function

As suggested in [137], if clauses are viewed as finite sets of literals (e.g. the clause $class(X,c1) :- molec(X) \wedge atom(X,h)$ would correspond to the set $C = \{class(X,c_1), \neg molec(X), \neg atom(X,h)\}$) one could think of adapting generic distances for sets to the specific case of clauses. Precisely, the so-mentioned work proposes a distance between finite sets of elements (assuming that there exists a distance defined over the space of elements) based on the idea of optimal matchings[1]. The cost of a matching is expressed in terms of the distance between the elements (see Section 3.4.1 in this dissertation for a more detailed description). Therefore, the authors of [137] propose $d_a$ as the distance for the elements of sets (the atoms), obtaining in this way a distance $d_m$ for clauses.

Since $d_a$ is defined over atoms, and clauses are made up of literals, for a correct definition of $d_m$, $d_a$ must be extended over literals. This extension is trivial, since $p(\ldots)$ and $\neg p(\ldots)$ are considered incompatible literals. Hence it is like treating them as different predicates. The extended version of $d_a$ is denoted here by $d_l$. Another aspect to be mentioned is that the distance in [137], and consequently also $d_m$, depends on a constant $M$ which must be equal to or greater than the maximal distance between two elements (the elements from which sets are built up) so that the distance for sets satisfies all the axioms of a metric. This restriction forces us to bound (restrict) the space of literals. Then the restriction will consist of setting a threshold for the number of symbols in a literal because this affects on the distance between two literals. This restriction is not a serious problem since a real-life object representation always requires a finite number of symbols.

[1] Recall that a matching is an injective mapping between two sets $A$ and $B$ which is not necessarily defined over all the elements in $A$.

In what follows, $(\bar{X}, d_l)$ denotes the bounded space of literals, $M = (M_1, M_2)$ is the bound of $\bar{X}$ (recall that $d_l$ returns an ordered pair of values) and $(2^{\bar{X}_l}, d_m)$ is the metric space of sets (clauses). Next, we introduce the notation to refer to sets (clauses), elements in sets (literals) and matchings between sets because it will be needed for the proofs below. Namely, sets are denoted by uppercase letters $(A, B, C, \ldots)$, elements in sets by (subscripted) lowercase letters $(a_1, a_2, \ldots, b_1 \ldots, c_1 \ldots)$ and a matching between to sets $A$ and $B$ by $\alpha_{A,B}$. We will say that an ordered pair of elements $(a, b)$ belongs to the matching $\alpha_{A,B}$ if $\alpha_{A,B}(a) = b$. By $D(\alpha_{A,B})$, we denote the domain of the matching, that is, $D(\alpha_{A,B}) = \{a \in A : \exists (a, b) \in \alpha_{A,B}\}$, and by $\alpha_{A,B}(A) = \{b \in B | (a, b) \in \alpha_{A,B} \wedge a \in A\}$ we denote the codomain of the matching.

**Example 53.** *Given the sets* $A = \{a_1 = p(g(a), e), a_2 = p(f(a), f(b)), a_3 = p(a, a)\}$ *and* $B = \{b_1 = p(f(b), f(a)), b_2 = p(f(a), e))\}$ *and according to the distances between all the atoms, the optimal matching* $\alpha_{A,B}$ *is* $\{(a_1, b_2), (a_2, b_1)\}$ *where* $D(\alpha_{A,B}) = \{a_1, a_2\}$ *and* $\alpha_{A,B}(A) = \{b_1, b_2\}$. *The distance between the sets is given by* $d_m(A, B) = d(a_1, b_2) + d(a_2, b_1) + \frac{1}{2}(M_1, M_2) = (8, -6) + (\frac{M_1}{2}, \frac{M_2}{2})$.

Regarding the pattern language, we define $\mathcal{L}$ as the set of all the logic programs we can define given a signature. Some examples of patterns could be,

$$\begin{array}{rcl} p_1 & \equiv & class(X, c_1) : -molec(X), atom(X, Y, h). \\ & & class(X, c_1) : -molec(X), atom(X, Y, o) \\ p_2 & \equiv & class(X, c_2) : -molec(Y), atom(Y, Z, c) \end{array}$$

The pattern $p_1$ says that a molecule belongs to the class/cluster $c_1$ if it has an atom of hydrogen or oxygen (something similar for the pattern $p_2$). As defined, a pattern can also be viewed as a set of clauses. For example,

$$\begin{array}{rcl} p_1 & = & \{C_{11} = \{class(X, c_1), \neg molec(X), \neg atom(X, h)\}, C_{12} = \{class(X, c_1), \neg molec(X), \neg atom(X, o)\}\} \\ p_2 & = & \{C_{21} = \{class(X, c_2), \neg molec(Y), \neg atom(Y, c))\}\} \end{array}$$

Given a pattern $p \in \mathcal{L}$, $Set(p)$ represents all those clauses in the metric space $2^{\bar{X}_l}$ which are a logical consequence of $p$. For instance, the clause $\{class(m_1, c_1), \neg molec(m_1), \neg atom(m_1, h)\}$ belongs to $Set(p_1)$.

Observe that each clause $C$ in a pattern $p$ is a pattern as well. To denote that a clause $C$ works as a pattern we write $\{C\}$. Additionally, there is no doubt that patterns can be combined by means of the + operation. Thus, it is easy to see that the pattern $p_3 = p_1 + p_2$ is equivalent to $p_3 = \{C_{11}, C_{12}, C_{21}\}$.

Finally, $k$ is an inclusion-preserving cost function. For instance, $k$ could be defined as $k(E, p) = c(p) + c(E|p)$ where $c(p) = (0, 0)$ and $c(E|p)$ is the first function in Table 5.2.

### 9.2.2 Computing Minimal Distance-based Generalisation Operators

Let us first see that $\Delta(E) = lgg(E)$ (where $lgg$ is the *least general generalisation* for clauses [129]) is not a *db* operator in this metric space. Thus, given two clauses $A$ and $B$ there exists a clause $C$ such that $d(A, C) + d(C, B) = d(A, B)$ and $C$ is not covered by $lgg(A, B)$ (see Example 54).

**Example 54.** *Given the sets* $A = \{\neg p(g(a), e), \neg p(f(a), f(b))\}$, $B = \{\neg p(f(b), f(a)), \neg p(f(a), e)\}$ *and* $C = \{\neg p(f(b), f(b)), \neg p(g(a), e)\}$. *The optimal mappings from A to C and from C to B, respectively, are depicted below.*

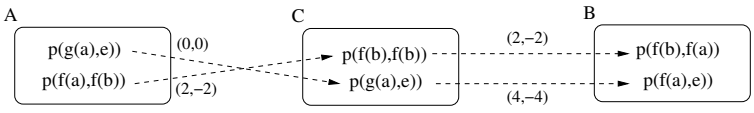


Figure 9.1: The arrows indicate the optimal mappings between the different pair of sets.

*We can easily see that* $d_m(A,B) = (8,-8) = d_m(A,C) + d_m(C,B)$. *However,*

$$\begin{aligned} lgg(A,B) = \quad & \{\neg p(f(X),f(Y)), \neg p(Z,e), \neg p(f(a),T)\} \equiv \\ & :-p(f(X),f(Y)), p(Z,e), p(f(a),T), \end{aligned}$$

*but* $C \equiv :-p(f(b),f(b)), p(g(a),e)$ *is not a logical consequence of* $lgg(A,B)$.

Unfortunately, computing *mdbg* operators in this space will not be as intuitive as it was in the previous section. We undertake the problem in a different way. First, we focus on determining binary *mdbg* operators. Then, we study whether by combining these operators, we can obtain $n$-ary *mdbg* operators. To define binary operators, we need to know, given two clauses $A$ and $B$, what conditions a clause $C$ must satisfy to be an intermediate clause of $A$ and $B$. This is stated next.

**Proposition 26.** *Let A,B and C three finite sets of elements in* $(2^{\bar{X}}, d_m)$. *If C is an intermediate element of A and B, then there exists an optimal mapping* $\alpha'_{A,B}$ *such that for every pair of elements* $(a_i, b_j)$ *in* $\alpha'_{A,B}$ *there exists an element* $c_k \in C$ *satisfying* $d_l(a_i,b_j) = d_l(a_i,c_k) + d_l(c_k,b_j)$.

*Proof.* Let $\alpha_{A,C}$, $\alpha_{C,B}$ be the optimal matchings used for the computation of $d_m(A,C)$ and $d_m(C,B)$, respectively. We can write,

$$\begin{aligned} d_m(A,C) &= \textstyle\sum_{\forall (a_i,c_j)\in\alpha_{A,C}} d_l(a_i,c_j) + \frac{M}{2}\cdot k_{\alpha_{A,C}} \\ d_m(C,B) &= \textstyle\sum_{\forall (c_i,b_j)\in\alpha_{C,B}} d_l(c_i,b_j) + \frac{M}{2}\cdot k_{\alpha_{C,B}} \end{aligned}$$

where $k_{\alpha_{A,C}}$ (respectively, $k_{\alpha_{C,B}}$) denotes the number of elements which do not belong to $\alpha_{A,C}$ (respectively, $\alpha_{C,B}$).

Next, we define the matching $\alpha'_{A,B}$ as the composition of the matchings $\alpha_{A,C}$ and $\alpha_{C,B}$. That is, $\alpha'_{A,B} = \alpha_{C,B}(\alpha_{A,C}(A))$. Keeping $\alpha'_{A,B}$ in mind, the sum $d_m(A,C) + d_m(C,B)$ can be written as,

$$\begin{aligned} d_m(A,C) + d_m(C,B) \quad = \quad & \textstyle\sum_{\forall (a_i,b_j)\in\alpha'_{A,B}} d_l(a_i,\alpha_{A,C}(a_i)) + d_l(\alpha_{A,C}(a_i),b_j) \\ & + \textstyle\sum_{\forall a_i \in D(\alpha_{A,C}) - D(\alpha'_{A,B})} d_l(a_i,\alpha_{A,C}(a_i)) \\ & + \textstyle\sum_{\forall c_i \in D(\alpha_{C,B}) - \alpha_{A,C}(A)} d_l(c_i,\alpha_{C,B}(c_i)) \\ & + \frac{M}{2}\cdot k_{\alpha_{A,C}} + \frac{M}{2}\cdot k_{\alpha_{C,B}} \end{aligned} \tag{9.11}$$

The first term on the right-hand side of 9.11 considers all the ordered pairs belonging to the matchings that share an element $c_i \in C$. The second and third terms concern those ordered pairs in $\alpha_{A,C}$ and $\alpha_{C,B}$ (respectively), which were not taken into account by the first term. Finally, the two last terms come from those unmatched elements.
Next, the chain of inequalities shown in Equation (9.12) can be derived as follows. First we apply the triangle inequality over the first term on the right-hand side of Expression 9.11. Second, we

remove the second and the third terms. Third, we apply $k_{\alpha_{A,C}} + k_{\alpha_{C,B}} \geq k_{\alpha'_{A,B}}$. And finally, the last inequality is a direct consequence of the $d_m$ definition.

$$\begin{array}{rcl} d_m(A,C) + d_m(C,B) & \geq & \sum_{\forall (a_i,b_j) \in \alpha'_{A,B}} d_l(a_i,b_j) \\ & & + \sum_{\forall a_i \in D(\alpha_{A,C}) - D(\alpha'_{A,B})} d_l(a_i, \alpha_{A,C}(a_i)) \\ & & + \sum_{\forall c_i \in D(\alpha_{C,B}) - \alpha_{A,C}(A)} d_l(c_i, \alpha_{C,B}(c_i)) \\ & & + \frac{M}{2} \cdot k_{\alpha_{A,C}} + \frac{M}{2} \cdot k_{\alpha_{C,B}} \\ & \geq & \sum_{\forall (a_i,b_j) \in \alpha'_{A,B}} d_l(a_i,b_j) \\ & & + \frac{M}{2} \cdot (k_{\alpha_{A,C}} + k_{\alpha_{C,B}}) \\ & \geq & \sum_{\forall (a_i,b_j) \in \alpha'_{A,B}} d_l(a_i,b_j) + \frac{M}{2} \cdot (k_{\alpha'_{A,B}}) = d(\alpha'_{A,B}, A, B) \\ & \geq & d_m(A,B) \end{array} \tag{9.12}$$

The expression $d(\alpha'_{A,B})$ (third inequality in (9.12)) stands for the distance between the sets $A$ and $B$ according to the matching $\alpha'$ (see Section 3.4.1 in this dissertation).

The equality $d(A,C) + d(C,B) = d(A,B)$ holds only if all the inequalities ($\geq$) on the right-hand-side of Equation (9.12) becomes an equality. Among all of these transformations, only the first and the last one are fundamental in order to prove the proposition. The first inequality turns into an equality if the element $c_k = \alpha_{A,C}(a_i)$ in the first term on the right-hand-side of (9.11) satisfies $d(a_i,b_j) = d(a_i,c_k) + d(c_k,b_j)$, for every pair $(a_i,b_j)$ in $\alpha'_{A,B}$. The proposition is automatically proved if $\alpha'_{A,B}$ is an optimal matching, and if this occurs then the last inequality is transformed into an equality. □

Another intermediate result we also need before defining distance-based operators for clauses (by the fact that clauses are built up from literals) is to prove that *lgg* for literals is also distance-based, or even better, minimal distance-based.

**Proposition 27.** *The Proposition 25 holds for the metric space* $(\bar{X}_l, d_l)$.

*Proof.* Given $e_i, e_j \in E \subset \bar{X}_l$, then $lgg(e_i,e_j) = \top$ or $lgg(e_i,e_j) \neq \top$. Regarding the first possibility, if $e_k$ is an intermediate element of $e_1$ and $e_j$ then $e_k \in Set(\top) = lgg(E)$. Regarding the second possibility, this implies that $e_i$ and $e_j$ are either positive or negative atoms headed by the same predicate symbol. Hence, we can apply Corollary 8 to conclude that any intermediate element $e_k$ of $e_i$ and $e_j$ satisfies that $e_k \in Set(lgg(e_i,e_j)) \subset Set(lgg(E))$. Therefore $lgg(E)$ is *db* in this space.

As for its minimality, for every pattern $p$ such that $E \subset Set(p)$ we also have that $Set(lgg(E)) \subset Set(p)$. Hence *lgg* is a *mdbg* operator for any inclusion preserving cost function. □

Proposition 26 along with the fact that *lgg* for literals is distance-based suggests a strategy to define binary *dbg* operators: given two clauses $A$ and $B$ in $2^{\bar{X}_l}$, we could initially define $\Delta(A,B) = \{\{lgg(a_i,b_j) : (a_i,b_j) \in \alpha_{A,B}\}\}$ where $\alpha_{A,B}$ is an optimal matching. A distance-based operator must compute a pattern covering the elements $C$ between $A$ and $B$. Proposition 26 states that if $C$ is between $A$ and $B$, then $C$ contains literals $c_k$ which are also between $a_i$ and $b_j$, for every $(a_i,b_j)$ in an optimal $\alpha_{A,B}$. But as *lgg* for literals is distance-based for the distance $d_l$, if $c_k$ is between the literals $a_i$ and $b_j$, then $c_k \in Set(lgg(a_i,b_j))$ and i.e. $C \in Set(\Delta(A,B))$. At first glance, this definition of $\Delta$ seems to be distance-based. However, two important details must be considered.

1. Variables occurring in the different $lgg(a_i, b_j)$ must be independent (i.e. never repeated). Otherwise, the corresponding pattern might not be distance-based. Note that this makes sense since the distance $d_m$ does not capture the semantic of repeated variables occurring in different atoms in the same clause. For instance, given the sets $A = \{p(a), q(a)\}$, $B = \{p(b), q(b)\}$, $C = \{p(c), q(d)\}$, we have that $d_m(A, B) = d_m(A, C)$ when intuitively $B$ is more similar to $A$ than $C$ is, and i.e., $B$ should be nearer to $A$ than $C$ is.

2. More than one optimal matching can be given for $d_m(A, B)$. Of course, if the matchings lead to different patterns $p_1$ and $p_2$ such as $Set(p_1) \neq Set(p_2)$, there will be elements between $A$ and $B$ which do not belong to $Set(p_1)$ but also $Set(p_2)$ or vice versa. Hence, all the optimal matchings must be taken into account. Of course, this has a negative effect on the efficiency of computing distance-based operators.

Taking both observations above into account, the next Propositions 28 and 29 characterise the family of all the distance-based binary generalisation operators in $(2^{\bar{X}_l}, d_m)$. Generally speaking, they show that a binary generalisation operator $\Delta(A, B)$ is distance-based iff $Set(\Delta^*(A, B)) \subset Set(\Delta(A, B))$, where $\Delta^*(A, B)$ represents the union of all the patterns $p_i$ obtained by taking all the optimal matchings between $A$ and $B$ into account.

**Proposition 28.** *Given two elements $A$ and $B$ in $(2^{\bar{X}_l}, d_l)$, the binary generalisation operator $\Delta^*(A, B)$ defined as*

$$\Delta^*(A, B) = \bigcup_{\forall \text{ optimal } \alpha_{A,B}} \{lgg(a_i, b_j) : \forall (a_i, b_j) \in \alpha_{A,B}\},$$

*where the repeated variables occurring in different $lgg(a_i, b_j)$ are independent, is distance-based.*

*Proof.* From Proposition 26, we know that if an element $D$ is between $A$ and $B$, then there exists an optimal matching $\alpha_{A,B}$ such that for every $(a_i, b_j) \in \alpha_{A,B}$ there exists a $d_k \in D$ with $d_k$ being between $a_i$ and $b_j$. As $lgg$ for literals is distance-based (Proposition 25), necessarily $d_k \in Set(lgg(a_i, b_j))$ and i.e. $D \in Set(\{lgg(a_i, b_j) : \forall (a_i, b_j) \in \alpha\})$. Since all the optimal matchings are taken into consideration, for every element $D$ between $A$ and $B$, $D \in Set(p)$ and therefore, $\Delta^*(A, B)$ is distance-based. □

**Proposition 29.** *A mapping $\Delta : 2^{\bar{X}_l} \times 2^{\bar{X}_l} \to \mathcal{L}$ is distance-based iff for every pair of elements $A$ and $B$ in $2^{\bar{X}_l}$, $Set(\Delta^*(A, B)) \subset Set(\Delta(A, B))$ with $\Delta^*$ being the distance-based operator defined in Proposition 28.*

*Proof.* ($\Rightarrow$) If $\Delta(A, B)$ is distance-based, then it means that for every $D$ between $A$ and $B$, $D \in Set(\Delta(A, B))$. Then, for every optimal matching $\alpha_{A,B}$, we define $D_{\alpha_{A,B}}$ as

$$D_{\alpha_{A,B}} = \{lgg(a_i, b_j) : \forall (a_i, b_j) \in \alpha_{A,B}\}$$

where $D_{\alpha_{A,B}}$ is clearly an element between $A$ and $B$. Then, for every optimal matching $\alpha_{A,B}$ , $D_{\alpha_{A,B}} \in Set(\Delta(A, B))$, i.e. $\Delta^*(A, B) \in Set(\Delta(A, B))$ and by definition of $Set(\cdot)$, $Set(\Delta^*(A, B)) \subset Set(\Delta(A, B))$.

($\Leftarrow$) We have that $Set(\Delta^*(A, B))$ is a subset of $Set(\Delta(A, B))$. As $\Delta^*(A, B)$ is distance-based, automatically $\Delta(A, B)$ is distance-based. □

As a direct consequence from Proposition 29 we have that $\Delta^*$ is the binary *mdbg* operator.

**Corollary 9.** *If $k$ is an inclusion-preserving cost function defined over $(2^{\bar{X}_l}, \mathcal{L})$ then the binary distance-based operator $\Delta^*$ is a binary minimal distance-based generalisation operator.*

*Proof.* For every distance-based operator $\Delta$, and for every pair of elements $A$ and $B$ in $2^{\bar{X}_l}$, we know from Proposition 29 that $Set(\Delta^*(A,B)) \subset Set(\Delta(A,B))$ and given that $k$ is inclusion-preserving then $\Delta^*(A,B)$ is a binary *mdbg* operator. □

Once we know how to define binary *dbg* operators and given that $\mathcal{L}$ is endowed with the $+$ operation we can easily define $n$-ary *dbg* operators w.r.t. a nerve function $N$ via Proposition 1. To do this, it is enough to choose a binary *dbg* operator $\Delta^b$ and a nerve function $N$ in order to define $\Delta_N$. But the most remarkable thing here is that, although $\Delta_N$ is not in general minimal operator relative to $N$, if we set $\Delta^b = \Delta^*$ then $\Delta_N$ is a *mdbg* operator relative to $N$.

**Proposition 30.** *Let $k$ be an inclusion-preserving cost function and let $N$ be a nerve function. If we set $\Delta^b = \Delta^*$ then the operator $\Delta_N$ is the mdbg operator relative to $N$ for $(k, \mathcal{L})$.*

*Proof.* We will proceed by contradiction. Let us suppose that $\Delta_N$ is not minimal. Then there exists a finite set of elements $E$ and a distance-based generalisation operator $\Delta'$ wrt. to $N$ such that $k(E, \Delta'(E)) < k(E, \Delta_N(E))$. We define from a binary distance-based operator $\Delta''$ restricted to all the pairs $(e_i, e_j) \in N(E)$ in such a way that $\Delta''(e_i, e_j) = \Delta'(E)$. But according to Proposition 29 as $\Delta''$ is distance-based then

$$\forall (e_i, e_j) \in N(E),\ Set(\Delta^*(e_i, e_j)) \subseteq Set(\Delta''(e_i, e_j)) = Set(\Delta'(E))$$

As it happens for every $(e_i, e_j) \in N(E)$, then we have that $Set(\Delta_N(E)) \subseteq Set(\Delta'(E))$, and given that $k$ is inclusion preserving $k(E, \Delta_N(E)) \leq k(E, \Delta'(E))$. □

## 9.3 Discussion

In this chapter, we have seen that Plotkin's *lgg* for atoms is a particular case of this setting because the *lgg* is a *mdbg* operator considering the metric space defined in [139] and any inclusion-preserving cost function. Furthermore, we have suggested that different *mdbg* operators for atoms can be obtained by changing the cost function, which can be an alternative which would allow us to redesign existing ILP methods or to derive new ones. For instance, imagine a problem whose examples are described by atoms where each atom contains a lot of information. The *lgg* of two near atoms will probably have again a lot of information and we could be interested in obtaining slightly more general meaningful generalisations that the one returned by the *lgg*. Thus, we need to change terms in the atom returned by the *lgg* by variables. But the question now is, which terms should be changed? We could consider a cost function $k$ which takes complexity of the atoms into account in such a way that minimising $k$ will result in atoms less complex that one obtained via *lgg*.

As for clauses, Plotkin's *lgg* has been shown not to be a *mdbg* operator for the particular metric space derived from the distance introduced by [137]. Due to the complexity of this space, a new *mdbg* operator relative to one specific nerve function has been introduced.

Finally, another interesting observation is that some adaptations of the ILP algorithms can be viewed as distance-based operators in our setting. For instance, instead of taking ground evidence

as input, a bottom-up ILP inference algorithm could be redesigned to take as input the pattern (clauses) computed by an *mdbg* operator. The output of the algorithm, a pattern which is more general than the input pattern, would also be a distance-based pattern. Thus, we could think of ILP algorithms as particular distance-based operators.

# Chapter 10

# Generalising Tuples Lying in a Metric Space

## 10.1 Tuples and Composability

A tuple is a widely-used structure for knowledge representation. Indeed, in classical machine learning, examples are tuples of nominal and numerical data. Structured learning usually requires more expressive representation languages. However, in general, it might be useful to integrate both simple and structured data. Imagine a problem dealing with protein classification where the primary structure along with some chemical traits are known from each protein. In this case, an individual could be represented by means of a tuple, where one of the components would be a list of amino acids and the other could be numerical information corresponding to its chemical features. Hence, if other data types other than nominal or numerical data (e.g. lists, sets, etc.) are permitted, then we will have a nested representation language which will be able to cope with arbitrarily complex structures.

With regard to our framework, we do not restrict ourselves to tuples of nominal or numerical data; every data type can be included for the tuple definition. However, the most appealing aspect arises when the basic data types are embedded in a metric space. By assuming this, some theoretical results can be given to define distance-based generalisation operators for tuples from distance-based operators that are defined for these basic data types. In other words, we deal with the composability of distance-based operators for nested metric spaces. This is what we show next.

## 10.2 Distances, Pattern Languages and Cost Functions

Unless we state otherwise, we denote those objects that will be the components of a tuple by means of superscript symbols followed by a closing parenthesis . Thus, the space $X$ we are working with corresponds to $(X^{1)}, \ldots, X^{n)})$ where every $(X^{i)}, d_i)$ $(i = 1, \ldots, n)$ is a metric space endowed with a pattern language $\mathcal{L}^{i)}$ and a cost function $k_i$. Of course, an element $e$ in $X$ is written as $e = (e^{1)}, \ldots, e^{n)})$ and is called a $n$-tuple.

Although we assume that $X^{i)}$ is a space with a metric, a pattern language and a cost function,

note that nothing has yet been said about the metric, the pattern language and the cost function that the space $X$ will be endowed with. Therefore, the next step will consist of defining these items in $X$. In contrast to previous chapters, no particular distance function, pattern language or cost function will be introduced for this purpose; these will be obtained from the distance, the pattern language and the cost function defined over each $X^{i)}$.

Regarding the distance, thanks to some well-known mathematical results concerning the product of metric spaces [90], we have that the following expressions

$$\begin{array}{lcl} d(x,y) & = & \sum_{i=1}^{n} d_i(x^{i)}, y^{i)}) \text{ (Manhattan distance).} \\ d(x,y) & = & \sum_{i=1}^{n} \alpha_i \cdot d_i(x^{i)}, y^{i)}) \text{ (Weighted Manhattan distance).} \\ d(x,y) & = & max_{1 \leq i \leq n} d_i(x^{i)}, y^{i)}) \text{ (Box distance).} \end{array} \tag{10.1}$$

are distance functions in $X$.

**Example 55.** *Let $X^{1)}$ be the real numbers with the usual distance and let $X^{2)}$ be the space of lists over an alphabet with the edit distance employed in Chapter 8. Given the examples $e_1 = (-1, aaabb)$ and $e_2 = (9, aaa)$, according to the Manhattan distance we have*

$$d(e_1, e_2) = |-1-9| + d_{edit\ distance}(aaabb, aaa) = 10 + 2 = 12$$

*Now, using the Box distance we have*

$$d(e_1, e_2) = max\{|-1-9|, d_{edit\ distance}(aaabb, aaa)\} = max\{10, 2\} = 10$$

Secondly, as for the pattern language, we define the basic pattern language $\mathcal{L}_0$ as follows:

$$\mathcal{L}_0 = (\mathcal{L}^{1)}, \ldots, \mathcal{L}^{n)})$$

Thus, a pattern $p \in \mathcal{L}_0$ can be unfolded as $p = (p^{1)}, \ldots, p^{n)})$ where $p^{i)} \in \mathcal{L}^{i)}$. According to this, it is reasonable to define the mapping $Set(\cdot)$ over $\mathcal{L}_0$ as:

$$e \in Set(p) \Leftrightarrow \ e^{i)} \in Set(p^{i)}), \forall i = 1, \ldots, n$$

Let us illustrate this with the following example.

**Example 56.** *In regard with the previous example, let us consider that $\mathcal{L}^{1)}$ and $\mathcal{L}^{2)}$ are the pattern languages introduced for real numbers and lists, respectively. That is, $\mathcal{L}^{1)}$ and $\mathcal{L}^{2)}$ are the language $\mathcal{L}_0$ defined for numbers and lists in their respective chapters.*

*Given the pattern $p = ([-10, 10], aaV^5)$, it is clear that the elements $e_1$ and $e_2$ belong to $Set(p)$, but the element $e_3 = (0, bbb)$ is not covered since $bbb \notin Set(aaV^5)$.*

It does not matter how complex the pattern language $\mathcal{L}^{i)}$ is. Any pattern language can be used to define $\mathcal{L}_0$. For instance, suppose that every $\mathcal{L}^{i)}$ from the example above is, in turn, endowed with the + operation. Then, we would have patterns such as $p = ([0, 1] + [8, 11], aaaV^4 + VbbV^2)$ or $p = ([-7, 7], VbbV^2 + aaV + V)$. For our purposes, we will assume that every $\mathcal{L}^{i)}$ is endowed with the operation +.

Finally, the cost function over $X$ and $\mathcal{L}_0$ must be defined. For a set of elements $E = \{e_i\}_{i=1}^{n}$ and a pattern $p$ covering $E$, we can consider two interesting possibilities:

$$\begin{array}{lcl} K_0(E,p) & = & \sum_{j=1}^{n} k_j(\{e_i^{j)}\}_{i=1}^{n}, p^{j)}) \\ K_1(E,p) & = & min\{k_j(\{e_i^{j)}\}_{i=1}^{n}, p^{j)}) : \ \forall 1 \leq j \leq n\} \end{array}$$

It is easy to see that both $K_0$ and $K_1$ satisfy the condition to be a cost function (see Lemma 6). In this case, we use uppercase letters to differentiate them from the cost functions $k_i$ associated to $X^{i)}$.

**Lemma 6.** *The functions $K_0$ and $K_1$ are cost functions for $X$ and the pattern language $\mathcal{L}_0$.*

*Proof.* Trivia. Given that every $k_j$ $(j = 1, \ldots, n)$ is a cost function, the for every set of elements $\{e_i^{j)}\}_{i=1}^n$ and pattern $p^{j)}$ covering it such that $Set(p^{j)}) \neq X^{j)}$, there exists a constant $c_j$ such that $k_j(\{e_i^{j)}\}_{i=1}^n, p^{j)}) < c_j$. Thus, for every set of elements $E$ and pattern $p$ covering $E$ such that $Set(p) \neq X$, we have that

$$\begin{array}{rcl} K_0(E,p) & = & \sum_{j=1}^n k_j(\{e_i^{j)}\}_{i=1}^n, p^{j)}) \leq \sum_{j=1}^n c_j = C_j \\ K_1(E,p) & = & min\{k_j(\{e_i^{j)}\}_{i=1}^n, p^{j)}) : \ \forall 1 \leq j \leq n\} \leq min\{c_j\}_{j=1}^n \end{array}$$

hence, $K_0$ and $K_1$ are cost functions. □

Let us see a concrete example about how the cost functions $K_0$ and $K_1$ work.

**Example 57.** *Let $X^{1)}$ be the metric space of real numbers and let $\mathcal{L}^{1)}$ and $k_1$ be the pattern language and the cost function ($k_0$) explained in Subsection **??** from Chapter 6.*

*We use the Manhattan distance as a metric for the real plane $X = X^{1)} \times X^{1)}$. Now, given the set of elements $E = \{e_1(2,1), e_2(1,3)\}$ and the pattern $p = [0,4] \times [0,3]$ covering $E$ (see Figure 10.1), we have that:*

$$\begin{array}{rcl} K_0(E,p) & = & k_1(\{2,1\},[0,4]) + k_1(\{1,3\},[0,3]) = 3 + 2 = 5 \\ K_1(E,p) & = & min\{k_1(\{2,1\},[0,4]),\ k_1(\{1,3\},[0,3])\} = min\{3,2\} = 2 \end{array}$$

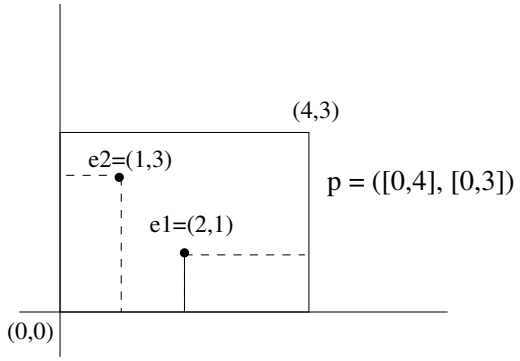


Figure 10.1: Computing cost functions $K_0$ and $K_1$ for the set $E = \{e_1, e_2\}$ and the pattern $p = [0,4] \times [0,3]$.

Before studying how to define possible distance-based and minimal distance-based generalisation operators for the different metric spaces of tuples, we need to introduce the following two definitions.

**Definition 28.** ***(Set projection)*** *Given a set of $m$-tuples $E = \{e_i\}_{i=1}^n \subset X$, then the projection of $E$ over the $j$-th component is just the set*

$$E^{j)} = \{e_i^{j)}\}_{i=1}^n.$$

**Definition 29.** ***(Nerve function projection)*** *Let $E$ be a finite set of $m$-tuples. Given a nerve function $N$, we define $N^{k)}$ as the nerve function defined over the set $E^{k)}$ such that*

$$(e_i^{k)}, e_j^{k)}) \in N^{k)}(E^{k)}) \Leftrightarrow (e_i, e_j) \in N(E) \tag{10.2}$$

Note that this definition only makes sense when $X$ is a space of tuples. Figure 10.2 illustrates the projection of a nerve given by a nerve function when the space $X$ is the real plane.

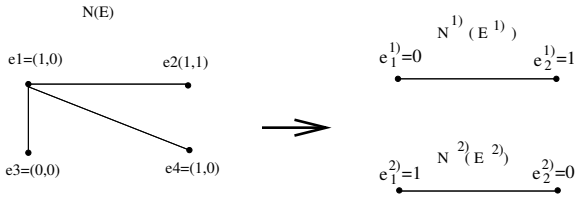


Figure 10.2: Projection of the nerve $N(E)$. Observe that $e_1^{1)} = e_2^{1)} = e_4^{1)}$ and $e_1^{2)} = e_3^{2)} = e_4^{2)}$.

## 10.3 Distance-based Operators Using the Weighted Manhattan Distance

In this section, $d$ denotes the weighted Manhattan distance. The reason why the weighted version is employed instead of the original one, comes by the fact that all the results stated for the weighted version will also hold for the original one.

In order to define distance-based operators, we introduce the next preliminary result, which characterises those elements that lie between two other elements.

**Proposition 31.** *Given the $m$-tuples $e_1$, $e_2$, $e_3 \in X$, the element $e_3$ is between $e_1$ and $e_2$ iff $e_3^{i)}$ is between $e_1^{i)}$ and $e_2^{i)}$, for every $1 \leq i \leq n$.*

*Proof.* ($\Rightarrow$): We will proceed by contradiction. Thus, $e_3$ is between $e_1$ and $e_2$ and there exists an index $i$ such that $1 \leq i \leq n$ and $e_3^{i)}$ is not between $e_1^{i)}$ and $e_2^{i)}$. Given that $d$ is a distance, we can write:

$$\begin{array}{l} d_j(e_1^{j)}, e_2^{j)}) = d_j(e_1^{j)}, e_3^{j)}) + d_j(e_3^{j)}, e_2^{j)}) \ \ \forall j, 1 \leq j \neq i \leq n \\ d_i(e_1^{i)}, e_2^{i)}) < d_i(e_1^{i)}, e_3^{i)}) + d_i(e_3^{i)}, e_2^{i)}) \end{array} \tag{10.3}$$

If we sum inequalities above indexed by the index $j$ and $i$, we have:

$$\begin{array}{lcl} \sum_{i=1}^m d_i'(e_1^{i)}, e_2^{i)}) & < & \sum_{i=1}^m d_i'(e_1^{i)}, e_3^{i)}) + d_i(e_3^{i)}, e_2^{i)}) \Leftrightarrow \\ d(e_1, e_2) & < & d(e_1, e_3) + d(e_3, e_2) \end{array} \tag{10.4}$$

which is not possible since $e_3$ is between $e_1$ and $e_2$. Therefore, $e_3^{i)}$ must lie between $e_1^{i)}$ and $e_2^{i)}$.

($\Leftarrow$): It is direct. Given that

$$d_i(e_1^{i)}, e_2^{i)}) = d_i(e_1^{i)}, e_3^{i)}) + d_i(e_3^{i)}, e_2^{i)}) \ \ \forall i, 1 \leq i \leq n \tag{10.5}$$

If we sum inequalities above, we automatically have $d(e_1, e_2) = d(e_1, e_3) + d(e_3, e_2)$. □

The following result says how to define a distance-based generalisation operator in $X$ from the distance-based generalisation operators in $X^{i)}$.

**Proposition 32.** *Given a finite set of m-tuples $E$, a nerve function $N$ and a set of generalisation operators $\{\Delta_i\}_{i=1}^n$ defined over $X^{i)}$. The generalisation operator $\Delta : X \to \mathcal{L}_0$ defined as*

$$\Delta(E) = (\Delta_1(E^{1)}), \ldots, \Delta_n(E^{n)}))$$

*is a distance-based generalisation operator in $X$ w.r.t. $N$ if $\Delta_i$ is a distance-based generalisation operator in $(X^{i)}, d_i)$ w.r.t. the nerve $N^{i)}$, for every $i = 1, \ldots, n$.*

*Proof.* For every $(e_1, e_2) \in N(E)$, if the element $e_3 \in X$ is between $e_1$ and $e_2$ then, according to Proposition 31 we have

$$d_i(e_1^{i)}, e_2^{i)}) = d_i(e_1^{i)}, e_3^{i)}) + d_i(e_3^{i)}, e_2^{i)}) \ \ \forall i, 1 \leq i \leq n \tag{10.6}$$

By the definition of nerve projection (see Definition 29) we can write

$$(e_1^{i)}, e_2^{i)}) \in N^{i)}(E^{i)}) \ \ \forall i, 1 \leq i \leq n \tag{10.7}$$

As $\Delta_i$ is a distance-based operator w.r.t. $N^{i)}$ then

$$e_3^{i)} \in Set(\Delta_i(E^{i)})) \ \ \forall i, 1 \leq i \leq n \tag{10.8}$$

Hence, we can conclude that $e_3 \in Set(\Delta(E))$.

□

**(Remark 1)** If $\Delta_i$ is not distance-based w.r.t. $N^{i)}(\cdot)$, then $\Delta$ is not distance-based w.r.t. $N(E)$.

Next, we illustrate this result by means of a simple example.

**Example 58.** *Let $X$ be the real plane with the Manhattan distance and the pattern language defined in the Example 57. The set of examples $E$ and the nerve $N(E)$ are those introduced in Figure 10.2.*

*If $p$ is a distance based-pattern of $E$ w.r.t. $N(E)$, then $Set(p)$ must include the rectangle $[0,1] \times [0,1]$. Namely, the elements between $e_1$ and $e_2$ are those in the pattern $[0,1] \times [1,1]$, the elements between $e_1$ and $e_3$ correspond to the pattern $[0,0] \times [0,1]$ and finally the elements between $e_1$ and $e_4$ are those in $[0,1] \times [0,1]$. Hence, if a pattern $p$ is distance w.r.t. to $N(E)$, necessarily must include the pattern $[0,1] \times [0,1]$.*

*Let us see that this result can be obtained by applying Proposition 32. According to this proposition, we have to define a distance-based pattern $p_1$ of $E^{1)} = \{0,1\}$ w.r.t. the nerve $N(E^{1)})^{1)} = \{(0,1)\}$ as well as a distance-based pattern of $E^{2)} = \{0,1\}$ w.r.t. $N(E^{2)})^{2)} = \{(0,1)\}$. As we know, $[0,1] \subset Set(p_1)$ and $[0,1] \subset Set(p_2)$ and so $[0,1] \times [0,1] \subset Set((p_1, p_2))$.*

Keeping this latter example in mind, we know that every pattern containing the rectangle $[0,1] \times [0,1]$ is a distance-based pattern of $E$ w.r.t. $N(E)$. What we want to know next is whether the pattern $p = ([0,1], [0,1])$ is minimal in $(\mathcal{L}_0, K_0)$.

### 10.3.1 Minimal Distance-Based Generalisation Operators via $K_0$

As a distance-based generalisation operator $\Delta$ in $X$ is built up from the distance-based operators $\Delta_i$ defined in $X^{i)}$, we could intuitively expect that if $\Delta_i$ is a minimal distance-based generalisation operator then $\Delta$ will be minimal as well. This is shown next.

**Corollary 10.** *Let $\Delta$ be the distance-based generalisation operator defined in Proposition 32 and let $\Delta_i$ be the distance-based operators introduced in the same proposition. If for every $i$, $\Delta_i$ is a minimal distance-based generalisation operator relative to $N^{i)}$ and $k_i$, then $\Delta$ is a minimal distance-based generalisation operator relative to $N$ and $K_0$.*

*Proof.* We proceed by contradiction. Let us suppose that $\Delta$ is not minimal. Then, there exists a finite set of elements $E$ and a distance-based pattern $p = (p_1, \ldots, p_n) \in \mathcal{L}$ of $E$ w.r.t. $N(E)$ such that $K_0(E, p) < K_0(E, \Delta(E))$. As $K_0$ is the sum of $n$ positive functions $(k_i)$, then necessarily

$$\exists i,\ 1 \leq i \leq n : k_i(E^{i)}, p^{i)}) < k_i(E^{i)}, \Delta_i(E^{i)})), \tag{10.9}$$

but this is impossible because $p^{i)}$ is a distance-based pattern of $E^{i)}$ w.r.t. $N^{i)}(E^{i)})$ and $\Delta_i$ is a minimal distance-based generalisation operator relative to $N^{i)}$. □

Therefore, taking Example 58 again, the pattern $p = ([0,1],[0,1])$ is a minimal distance-based pattern of $E$ relative to $N(E)$ since, as we have seen, the interval $[0,1]$ is a minimal distance-based pattern of the set $\{0,1\}$.

## 10.4 Distance-based Operators Using the Box Distance

In this section $d$ denotes the Box distance. As we did before, we need a property informing about the elements which are between two given elements $e_1$ and $e_2$ in $X$. Then, we will use this result to define distance-based operators. Let us see.

**Proposition 33.** *Let $e_1$, $e_2$ and $e_3$ be three $m$-tuples in $X$. If the element $e_3$ is between $e_1$ and $e_2$ then*

$$\forall i \text{ such that } d_i(e_1^{i)}, e_2^{i)}) = d(e_1, e_2) \Rightarrow e_3^{i)} \text{ is between } e_1^{i)} \text{ and } e_2^{i)}).$$

*Proof.* We will proceed by contradiction. Let us suppose that there exists an index $i$ $(1 \leq i \leq n)$ such that

$$d(e_1, e_2) = d_i(e_1^{i)}, e_2^{i)}) \tag{10.10}$$

and

$$d_i(e_1^{i)}, e_2^{i)}) < d_i(e_1^{i)}, e_3^{i)}) + d_i(e_3^{i)}, e_2^{i)}). \tag{10.11}$$

According to the definition of the Box distance, there exist two indices $j$ and $k$ $(1 \leq j, k \leq n)$ such that,

$$\begin{array}{rcl} d(e_1, e_3) & = & d_j(e_1^{j)}, e_3^{j)}) \\ d(e_3, e_2) & = & d_k(e_3^{k)}, e_2^{k)}) \end{array} \tag{10.12}$$

Again, the next two inequalities below are followed by the definition of the Box distance:

$$\begin{array}{l} d_j(e_1^{j)}, e_3^{j)}) \geq d_i(e_1^{i)}, e_3^{i)}) \\ d_k(e_3^{k)}, e_2^{k)}) \geq d_i(e_3^{i)}, e_2^{i)}) \end{array} \tag{10.13}$$

By combining the expressions (10.10), (10.11), (10.12) and (10.13) we can write:

$$d(e_1, e_3) + d(e_3, e_2) \geq d_i(e_1^{i)}, e_3^{i)}) + d_i(e_3^{i)}, e_2^{i)}) > d(e_1, e_2) \tag{10.14}$$

and $e_3$ is not between $e_1$ and $e_2$ which is not possible. □

**(Remark 1)** The inverse implication does not hold in general. It is enough to consider the following example in $R^3$. Let $e_1 = (0, 0, 0)$, $e_2 = (4, 3, 4)$ and $e_3 = (1, 7, 2)$. The first and the third components of $e_3$ are both between 0 and 4 but $e_3$ is not in between.

Now, let us see how to define distance-based operators in $X$ from those defined in $X_i$.

**Proposition 34.** *Given a finite set of m-tuples $E$, a nerve function $N$. The generalisation operator $\Delta : X \to \mathcal{L}_0$ defined as*

$$\Delta(E) = \begin{cases} (X^{1)}, \ldots, X^{k-1)}, \Delta_k(E^{k)}), X^{k+1)}, \ldots, X^{n)}), d(e_i, e_j) = d_k(e_i^{k)}, e_j^{k)}),\ \forall (e_i, e_j) \in N(E) \\ X,\ \textit{otherwise.} \end{cases}$$

*is a distance-based generalisation operator w.r.t. $N$ if $\Delta_k$ is a distance-based generalisation operator w.r.t. $N^{k)}$.*

*Proof.* Let us suppose that the condition

$$d(e_i, e_j) = d_k(e_i^{k)}, e_j^{k)}), \forall (e_i, e_j) \in N(E) \tag{10.15}$$

holds. If an element $e$ is between $e_i$ and $e_j$ then, according to Proposition 33, $e^{k)}$ is between $e_i^{k)}$ and $e_j^{k)}$. Additionally, as $(e_i^{k)}, e_j^{k)})$ belongs to $N^{k)}(E^{k)})$ and $\Delta_k(E^{k)})$ is distance-based w.r.t. $N^{k)}(E)$ then

$$e^{k)} \in Set(\Delta_k(E^{k)}))$$

Hence,

$$e \in Set((X^{1)}, \ldots, X^{k-1)}, \Delta_k(E^{k)}), X^{k+1)}, \ldots, X^{n)})) \in Set(\Delta(E))$$

If the condition (10.15) above does not hold, the pattern computed by $\Delta(E)$ is equal to $X$ and trivially distance-based. □

From this Proposition 34 we can see that the pattern language $\mathcal{L}_0$ is not really adequate for the Box distance since the condition (10.15), from which a non trivial pattern (a pattern different to $X$) can be obtained, is so restrictive that will rarely hold. The most usual thing is that different components (not exclusively the $k$-th component) are involved when computing the distances among the elements in $E$. For instance, setting $E = \{e_1, e_2, e_3\}$ and $N(E) = \{(e_1, e_2), (e_2, e_3)\}$, the $k$-th component is involved when computing $d(e_1, e_2)$, that is, $d(e_1, e_2) = d_k(e_1^{k)}, e_2^{k)})$ but, however, a different component can be involved in $d(e_2, e_3)$, namely, $d(e_2, e_3) = d_i(e_2^{i)}, e_3^{i)})$ where $k \neq i$. In this case, $\Delta(E) = X$ which is not really useful. To avoid this, we would need to handle patterns such as $(\ldots, \Delta_k, \ldots) + (\ldots, \Delta_i, \ldots)$, and i.e., a pattern language endowed with the operation $+$.

The problem of minimality for $K_1$ will not be analysed in detail since, as we have just pointed out, the pattern language $\mathcal{L}_0$ is not really adequate for the Box distance. We will simply focus on showing an example illustrating how intricate this process could be.

Given the space $X = X^{1)} \times X^{1)}$, where $X^{1)}$ is the space of lists. We want two generalise the elements $e_1 = (a^3, ab)$ and $e_2 = (b^3, a)$. Since,

$$d(e_1, e_2) = d_1(a^3, b^3) = 6$$

According to Proposition 34, a possible distance-based pattern generalising $e_1$ and $e_2$ is $p = (V^6, X^{1)})$. But the second component in $p$, $X^{1)}$, is extremely general and it is reasonable to think that there might exist other patterns which fits $E$ better. However, Proposition 34 says nothing about these other alternative patterns. The uncertainty makes the search space larger, and i.e., makes the problem more difficult.

## 10.5 Discussion

In this chapter we have studied the composability of distance-based generalisation operators when tuples are involved. However, it would be interesting to study other forms of composability which could be really useful, for instance, for tuples of lists of symbols, lists of lists of symbols, sets of lists of symbols, etc. The idea, as we did for tuples, is to study whether it is possible to state a generic framework which allows us to systematically compose distance-based generalisation operators defined over non-nested structured datatypes to obtain generalisation operators for nested data.

# Chapter 11

# Generalising Graphs Lying in a Metric Space

Graphs represents a powerful formalism for knowledge representation. For instance, labelled graphs are excellent representations for molecules where each atom corresponds to a tagged vertex and each bound to an edge. But apart from this prototypical scenario, graphs appears in other classical problems such as image recognition, robotics, networking as well as in other more recent challenges (e.g. social and biological network analysis, web mining, etc.) which have strongly attracted the attention of the machine learning community.

In this section, we introduce possible definitions of (minimal) distance-based operators for graphs embedded in a particular metric space along with an analysis about the computational complexity of these operators. Next, we briefly review some basic concepts about graphs.

## 11.1 Preliminaries

Let $\Sigma$ be an alphabet of symbols. A graph is a 3-tuple $(V, E, l)$ where $V$ is a finite set of vertices, $E$ is a set of edges (each one denoted as a pair of vertices belonging to $V \times V$), and $l$ is a function which assigns labels from $\Sigma$ to both vertices and edges. Concretely, in all the examples used, vertices are labelled by uppercase letters and edges by lowercase letters. We denote by $\mathcal{U}$ the universe of all the connected undirected graphs defined over $\Sigma$ and we denote by the (indexed or superscripted)-lowercases letters $e$ (e.g. $e'$, $e''$, $e_1$, $e_2$,...) or $g$ (e.g. $g'$, $g''$, $g_1$, $g_2$,...) graphs from $\mathcal{U}$. The number of nodes of a graph $e = (E, V, l)$ is given by $|V|$ and the number of edges by $|E|$. The degree of a vertex $v$ (denoted by $deg(v)$) is the number of edges incident to the vertex.

Given two graphs $e_1 = (V_1, E_1, l_1)$ and $e_2 = (V_2, E_2, l_2)$, we say that $e_1$ is a *subgraph* of $e_2$ if there exists an injective function $f : V_1 \to V_2$ where for every pair of vertices $v_i, v_j \in V_1$ such that $(v_i, v_j) \in E_1$ then $(f(v_i), f(v_j)) \in E_2$ and $f$ preserve both the vertex and edge labels. This is denoted by $e_1 \subset e_2$ or equivalently by $e_1 \subset_f e_2$ if we want to stand out the underlying function.

We say that $e_1$ is a *vertex-induced subgraph* of $e_2$ if there exists an injective function $f : V_1 \to V_2$ such that if $(f(v_i), f(v_j)) \in E_2$ then $(v_i, v_j) \in E_1$. Additionally, we say that $e_1$ and $e_2$ are *isomorphic* if $f$ is a bijection preserving both the vertex and the edge labels such that for every pair of vertices $v_i, v_j \in V_1$, $(v_i, v_j) \in E_1$ if and only if $(f(v_i), f(v_j)) \in E_2$.

Given a third graph $e_3 = (V_3, E_3, l)$, we say that $e_3$ is a *common subgraph* of $e_1$ and $e_2$ if $e_3 \subset e_1$ and $e_3 \subset e_2$. In fact, $e_3$ is the **maximum common subgraph** of $e_1$ and $e_2$ (denoted by $mcs(e_1, e_2)$) if there is not other common subgraph $e$ of $e_1$ and $e_2$ such that $e_3 \subset e$. Note that the concept of common subgraph and *mcs* can trivially be extended for sets containing more than two graphs.

In a similar way, we can also talk of the *common vertex-induced subgraph* and the *maximum common vertex-induced subgraph* (*mcvis*, in short). We say that $e_3$ is a common vertex-induced subgraph of $e_1$ and $e_2$ if $e_3$ is vertex-induced subgraph of $e_1$ and $e_2$. Additionally, if there does not exists other vertex-induced subgraph $e$ of $e_1$ and $e_2$ such that $e_3 \subset e$ then we say that $e_3$ is the *mcvis* of $e_1$ and $e_2$.

Unlike the *mcs*, the *mcvis* might not be unique. In this line, Figure 11.1 illustrates the difference between the concepts of *mcs* and *mcvis*.

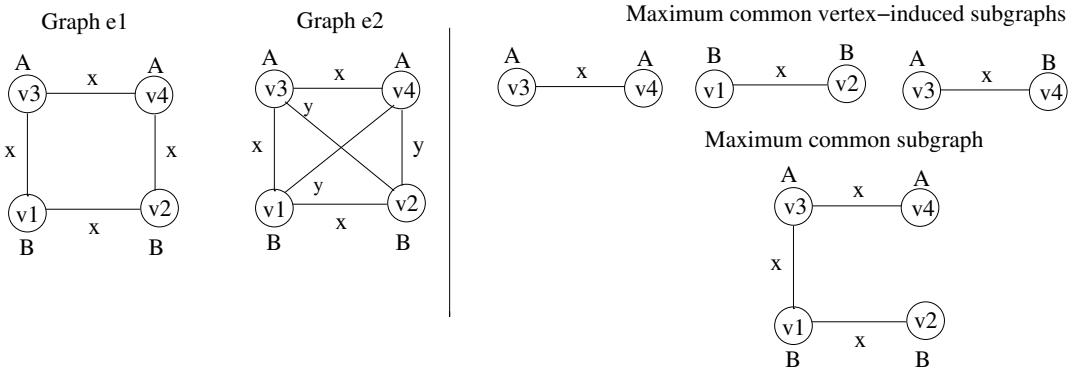


Figure 11.1: **(Left)** Graphs to be compared. **(Top right)** All the possible $mcvis(e_1, e_2)$. **(Bottom right)** The $mcs(e_1, e_2)$.

Besides knowing whether $e' \subset e$, that is, if a graph $e'$ occurs in graph $e$, we are also interested in determining the exact number of these occurrences. We denote this number by $O_{e',e}$. What it can happens now is that $e'$ occurs more than once in $e$ but some of these occurrences overlap. We define $O^{dis}_{e',e}$ as the greatest number of occurrences of $e'$ in $e$ such that none of them overlaps. We refer these occurrences as disjoint occurrences. This is shown in the next example:

**Example 59.** *Given the graphs $e_1$ and $e_2$ depicted below, we have that $O_{e_1,e_2} = 4$, $O^d_{e_1,e_2} = 2$.*

The last operation over graphs we are interested in is the difference. This is a binary operation defined over two graphs $e_1$ and $e_2$ and denoted by the usual symbol $-$. Suppose that $e_1 \subset_f e_2$ then $e_2 -_f e_1$ is the graph $(V_2 - f(V_1), E_2 - \{(u, v) : u \text{ or } v \in f(V_1)\})$. If $O_{e_1,e_2} = 1$ it is not necessary to specify $f$.

**Example 60.** *Let $e_1(V_1, E_1)$ and $e_2(V_2, E_2)$ be the graphs in Figure 11.2 and the function*

$$\begin{aligned} f \quad : \quad & V_1 \to V_2 \\ & v_1 \mapsto v_3 \\ & v_2 \mapsto v_4 \end{aligned}$$

*Then the graph $e_2 -_f e_1$ is the one depicted in Figure 11.3.*

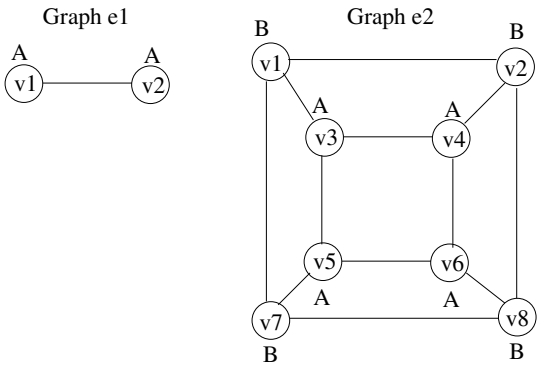


Figure 11.2: Counting the number of times the graph $e_1$ occurs in $e_2$.

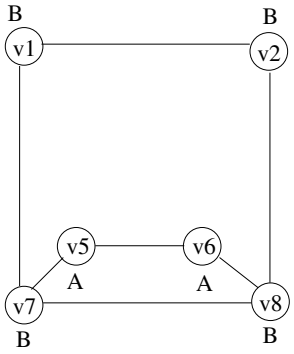


Figure 11.3: Computing the difference $e_2 -_f e_1$.

## 11.2 Distance, Pattern Languages and Cost Functions

The distance to be used in this chapter is a direct extension of the distance presented in [19] (Bunke's distance). In fact, our proposal results from a particular modification of the so-called *error correcting graph matching* (*ecgm*, in short) and the *cost of an ecgm*, which are the core concepts introduced in the referred work. Recall that an *ecgm* specifies a set of elementary operations (substitutions, insertions and deletions) in order to transform one graph into other. When every operation has assigned a cost, then we speak of the cost of an *ecgm* (see Subsection 3.4.1 in Chapter 3 for the formal definition).

In our case, the *ecgm* definition is not only given by the sets $\hat{V}_1 \subset V_1$, $\hat{V}_2 \subset V_2$ and by the bijective mapping $f : V_1 \to \hat{V}_2$ but we also need to specify the sets of edges $\hat{E}_1 \subset \hat{V}_1 \times \hat{V}_1$ and $\hat{E}_2 \subset \hat{V}_2 \times \hat{V}_2$ where $|\hat{E}_1| = |\hat{E}_2|$ to be transformed by $f$.

Having modified the definition of *ecgm*, we need to adapt the definition of the cost of an *ecgm* in order to take this change into account. Like in H. Bunke's work, we consider that the graphs are completely connected where the missing edges correspond to edges labelled with the special symbol

*null*.

$$\begin{aligned} Cost(e_1, e_2; f, \hat{V}_1, \hat{V}_2, \hat{E}_1, \hat{E}_2) &= |V_1| - |\hat{V}_1| + |V_2| - |\hat{V}_2| + \sum_{\forall v \in \hat{V}_1} c_f(u) + \sum_{\forall (u,v) \in \hat{E}_1} c_f((u,v)) \\ &\quad + \sum_{\forall (u,v) \in E_1 - \hat{E}_1} c_f((u,v)) + \sum_{\forall (u,v) \in E_2 - \hat{E}_2} c_f((u,v)) \end{aligned} \tag{11.1}$$

where

$$\forall u \in \hat{V}_1,\ c_f(u) = \begin{cases} 0 & , \quad l(u) = l(f(u)) \\ +\infty & , \quad \text{otherwise.} \end{cases}$$

$$\forall (u,v) \in \hat{E}_1,\ c_f((u,v)) = \begin{cases} 0 & , \quad l((u,v)) = l((f(u), f(v))) \\ +\infty & , \quad \text{otherwise.} \end{cases}$$

and

$$\forall (u,v) \in E_i - \hat{E}_i,\ c_f((u,v)) = \begin{cases} 0 & , \quad l((u,v)) = null \\ 1 & , \quad \text{otherwise.} \end{cases}$$

Next, we give an example dealing with the computation of the cost of several *ecgm*.

**Example 61.** *Let $e_1$ and $e_2$ be the graphs depicted in Figure 11.4. We are going to compute the cost of three different ecgm from $e_1$ to $e_2$.*

1. *$\hat{V}_1 = \{v_1, v_2\}$, $\hat{E}_1 = \{(v_1, v_2)\}$, $\hat{V}_2 = \{v_3, v_4\}$, $\hat{E}_2 = \{(v_3, v_4)\}$, $f(v_1) = v_3$ and $f(v_2) = v_4$. Then,*
$$c(e_1, e_2; f, \hat{V}_1, \hat{V}_2, \hat{E}_1, \hat{E}_2) = 1 + 2 + 2 + 3 + 0 + 0 = 8$$
2. *$\hat{V}_1$ and $\hat{E}_1$ are defined as before, $\hat{V}_2 = \{v_3, v_1\}$, $\hat{E} = \{(v_3, v_1)\}$ and $f(v_1) = v_3$ and $f(v_4) = v_1$. Then,*
$$c(e_1, e_2; f, \hat{V}_1, \hat{V}_2, \hat{E}_1, \hat{E}_2) = 1 + 2 + 2 + 3 + \infty + \infty = \infty$$
3. *$\hat{V}_1 = V_1$, $\hat{E}_1 = E_1$, $\hat{V}_2 = \{v_1, v_3, v_4\}$, $\hat{E}_2 = \hat{V}_2 \times \hat{V}_2$ where $f(v_1) = v_3$, $f(v_2) = v_4$ and $f(v_3) = v_1$.*
$$c(e_1, e_2; f, \hat{V}_1, \hat{V}_2, \hat{E}_1, \hat{E}_2) = 1 + 2 + 2 + 3 + 0 + \infty = \infty$$

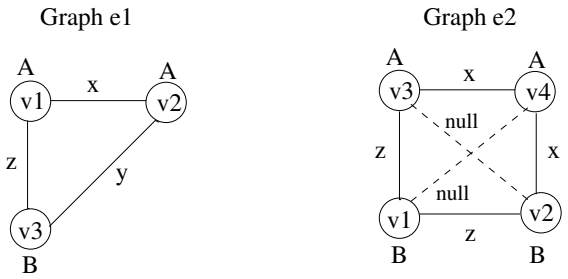


Figure 11.4: Transforming the graph $e_1$ into the graph $e_2$ using different *ecgm*.

Like in [19], the similarity between two graphs $e_1$ and $e_2$ is defined as the minimum cost taken over all *ecgm* from $e_1$ to $e_2$. Let us see that, in our case, this is equivalent to compute the $mcs(e_1, e_2)$ as the following proposition states.

**Proposition 35.** *Given two graphs $e_1 = (V_1, E_1)$ and $e_2 = (V_2, E_2)$ and let $e = (V, E)$ be the $mcs(e_1, e_2)$ then the equality below holds.*

$$\min_{\forall f, \hat{V}_1, \hat{V}_2, \hat{E}_1, \hat{E}_2} Cost(e_1, e_2; f, \hat{V}_1, \hat{V}_2, \hat{E}_1, \hat{E}_2) = (|V_1| + |V_2| - 2|V|) + (|E_1| + |E_2| - 2|E|)$$

*Proof.* Let $f$, $\hat{V}_1$, $\hat{V}_2$, $\hat{E}_1$, $\hat{E}_2$ be such that the minimum of the function *Cost* is attained. Then we necessarily have

$$Cost(e_1, e_2; f, \hat{V}_1, \hat{V}_2, \hat{E}_1, \hat{E}_2) < \infty$$

which leads to

$$\sum_{\forall v \in \hat{V}_1} c_f(v) = \sum_{\forall (u,v) \in \hat{E}_1} c_f((u, v)) = 0$$

By definition of the function *Cost*, we can ensure that

$$\forall v \in \hat{V}_1,\ l(v) = l(f(v))$$

and

$$\forall (u, v) \in \hat{E}_1,\ l((u, v)) = l(f((u, v)))$$

Given that $f$ is a bijection from $\hat{V}_1$ to $\hat{V}_2$ and $(v_i, v_j) \in \hat{E}_1$ iff $(v_i, v_j) \in \hat{E}_2$ then necessarily the graphs $(\hat{V}_1, \hat{E}_1)$ and $(\hat{V}_2, \hat{E}_2)$ are isomorphic. Therefore,

$$(\hat{V}_1, \hat{E}_1) = (\hat{V}_1, \hat{E}_1) \subset e$$

It is easy to see that if $(\hat{V}_1, \hat{E}_1) \subsetneq e$ then the minimum of the function *Cost* would not be achieved, thus $(\hat{V}_1, \hat{E}_1) = (\hat{V}_2, \hat{E}_2) = e$. Finally, doing some computations in the definition of the function *Cost* (see Equation 11.1 in this Section):

$$Cost(e_1, e_2; f, \hat{V}_1, \hat{V}_2, \hat{E}_1, \hat{E}_2) = Cost(e_1, e_2; f, V, V, E, E) = (|V_1| + |V_2| - 2|V|) + (|E_1| + |E_2| - 2|E|)$$

□

**Example 62.** *Let $e_1$ and $e_2$ be the graphs from the Example 61, the ecgm with minimum cost is given by $\hat{V}_1 = V_1$, $\hat{E}_1 = \{(v_1, v_2), (v_1, v_3)\}$, $\hat{V}_2 = \{v_1, v_3, v_4\}$, $\hat{E}_2 = \{(v_2, v_4), (v_3, v_1)\}$ and $f(v_1) = v_3$, $f(v_2) = v_4$ and $f(v_3) = v_1$ where*

$$Cost(e_1, e_2; f, \hat{V}_1, \hat{V}_2, \hat{E}_1, \hat{E}_2) = 0 + 1 + 0 + 0 + 1 + 2 = 4$$

*Observe that $(\hat{V}_1, \hat{E}_1)$ is the $mcs(e_1, e_2)$.*

Next, we will show that the *ecgm* with minimum cost induces a distance function over the universe of graphs.

**Proposition 36.** *Given two graphs $e_1 = (V_1, E_1)$ and $e_2 = (V_2, E_2)$ and let $e = (V, E)$ be the $mcs(e_1, e_2)$, then the similarity measure*

$$d(e_1, e_2) = (|V_1| + |V_2| - 2|V|) + (|E_1| + |E_2| - 2|E|)$$

*is a distance function for graphs.*

*Proof.* We have to check that the three properties of a distance hold.

1. **(identity)** $d(e, e) = 0$, for any graph $e$: Trivial from Proposition 35. Given two elements $e_1$ and $e_2$ whatever, if $d(e_1, e_2) = 0$ then necessarily $mcs(e_1, e_2) = e_1$ and $mcs(e_1, e_2) = e_2$ which implies that $e_1 = e_2$. Additionally, if $e_1 = e_2$ then $e_1 = e_2 = mcs(e_1, e_2)$ and $d(e_1, e_2) = 0$. Therefore, $d(e_1, e_2) = 0$ iff $e_1 = e_2$.

2. **(reflexive)** $d(e_1, e_2) = d(e_2, e_1)$, for any pair of graphs: Trivial from the Proposition 35 since $mcs(e_1, e_2) = mcs(e_2, e_1)$.

3. **(triangular inequality)** $d(e_1, e_2) \leq d(e_1, e_3) + d(e_3, e_1)$, for any three graphs. Initially, we set

   $$\begin{array}{l} e_i = (V_i, E_i),\ (i = 1, 2, 3) \\ (V_{13}, E_{13}) = mcs(e_1, e_3) \\ (V_{23}, E_{23}) = mcs(e_2, e_3) \\ (V_{13,23}, E_{13,23}) = mcs(mcs(e_1, e_2), mcs(e_2, e_3)) \end{array}$$

   Rewriting the sum $d(e_1, e_3) + d(e_3, e_2)$ by using the definition of the distance stated by the Proposition 35, we obtain:

   $$-|V_{12}| - |E_{12}| \leq (|V_3| - |V_{13}| - |V_{23}|) + (|E_3| - |E_{13}| - |E_{23}|) \tag{11.2}$$

   Thus, proving the triangle inequality is equivalent to prove the inequality (11.2). As the sets $V_{13}$ and $V_{23}$ can be viewed as subsets of $V_3$, and the set $V_{13,32}$ as a subset of both $V_{13}$ and $V_{23}$, then we can write:

   $$|V_3| - |V_{13}| - |V_{23}| = (|V_3| - |V_{13} - V_{13,23}| - |V_{23} - V_{13,23}| - |V_{13,23}|) - |V_{13,23}| \tag{11.3}$$

   Since

   $$(V_{13} - V_{13,32}) \cup (V_{13} - V_{13,32}) \cup V_{13,32} \subset V_3 \tag{11.4}$$

   then

   $$|V_3| - |V_{13} - V_{13,23}| - |V_{23} - V_{13,23}| - |V_{13,23}| \geq 0 \tag{11.5}$$

   and in consequence

   $$-|V_{13,32}| \leq |V_3| - |V_{13}| - |V_{23}| \tag{11.6}$$

   An identical reasoning over the set of edges leads to

   $$-|E_{13,32}| \leq |E_3| - |E_{13}| - |E_{23}| \tag{11.7}$$

   Thus, combining (11.6) and (11.9) we have

   $$-|V_{13,32}| - |E_{13,32}| \leq (|V_3| - |V_{13}| - |V_{23}|) + (|E_3| - |E_{13}| - |E_{23}|) \tag{11.8}$$

Given that $(V_{13,32}, E_{13,32})$ is a subgraph of $(V_{12}, E_{12})$ then

$$\begin{array}{rcl} |V_{12}| & \leq & |V_{13,32}| \\ |E_{12}| & \leq & |E_{13,32}| \end{array} \tag{11.9}$$

Putting (11.6) and (11.9) together,

$$-|V_{12}| - |E_{12}| \leq -|V_{13,32}| - |E_{13,32}| \leq (|V_3| - |V_{13}| - |V_{23}|) + (|E_3| - |E_{13}| - |E_{23}|) \tag{11.10}$$

□

**(Remark 1)** For any distance $d$ for graphs, we can say that obtaining $d(e_1, e_2)$ is, at least, as difficult as solving the isomorphic graph problem [62] because $e_1$ is isomorphic to $e_2$ iff $d(e_1, e_2) = 0$.

According to the proposition above, the distance between the graphs $e_1$ and $e_2$ in Example 62 is 4. Note that the distance between whatever two graphs $e_1$ and $e_2$ is the minimal number of removal and insertion operations in order to transform $e_1$ into $e_2$. This observation will permit us to define the distance function in a equivalent way. Thus, let $S(e_1) = (r_1, \ldots, r_p, i_1, \ldots, i_q)$ be a sequence of removal and insertion operations over a graph $e_1$. Each operation can act on edges or vertices and has a well-defined cost. Namely, every edge removal or insertion costs 1 and every vertex removal or insertion has cost $1 + deg(\cdot)$. It is easy to see that the distance between $e_1$ and $e_2$ can be put in terms of the cost of $S(e)$. Therefore,

$$d(e_1, e_2) = \min_{\forall S(e_1) = e_2} cost(S)$$

We will turn later to this particular way of expressing the distance when we start the study of the *mdbg* operators.

Regarding the pattern languages, we will use $\mathcal{L}_0$ and $\mathcal{L}_1$ where a pattern $p$ in $\mathcal{L}_0$ expressed by $[g]$ being $g$ a (connected or disconnected) graph built from the alphabet $\Sigma$ and $Set([p]) = \{g' \in \mathcal{U} : g \subset g'\}$. Additionally, we introduce a special symbol $\lambda$ which denotes the empty graph and in consequence $Set([\lambda]) = \mathcal{U}$. As usual, the pattern language $\mathcal{L}_1$ is obtained from $\mathcal{L}_0$ by means of the operation $+$.

As we can already forsee, the advantages of working with $\mathcal{L}_1$ arise when the elements to be generalised have no subgraph in common. In this case, the only possible generalisation in $\mathcal{L}_0$ corresponds to $[\lambda]$. This such an extreme generalisation can be soft when we move to $\mathcal{L}_1$ (see example below).

**Example 63.** *Given the graphs depicted below, the only possible generalisation in $\mathcal{L}_0$ is $[\lambda]$. In $\mathcal{L}_1$ other generalisations can be admissible, for instance,* "all the graphs having the vertex label $A$ or all the graphs having the vertex label $C$".

Finally, as for the cost functions, we will use $k_0 = c(E|p)$ in $\mathcal{L}_0$ where $c(E|p)$ is the first function in Table 5.2 and $k_1(E, p) = c_1(p) + c(E|p)$ in $\mathcal{L}_1$, where $c_1(p)$ counts both the number of vertices and edges of the subpatterns involved in $p$ to measure the complexity of the pattern. An example of this is given next.

**Example 64.** *Given the graphs $e_1$ and $e_2$ in Example 61 and the patterns $p_1 = [g_1]$ and $p_2 = [g_1] + [g_2]$ where $g_1$ and $g_2$ are the graphs depicted below (right hand-side picture). Let us calculate*

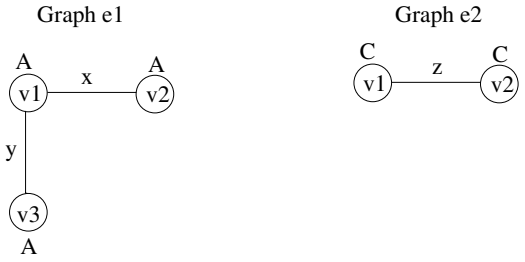


Figure 11.5: Two graphs $e_1$ and $e_2$ with no common subgraphs.

| | | | | |
|---|---|---|---|---|
| $p_1 = [g_1]$ | $e'_1 = e$<br>$e'_2 = e$ | $c(e_1\|p_1) = d(e_1, e'_1) = 3$<br>$c(e_2\|p_1) = d(e_2, e'_2) = 5$ | $k_0(E, p_1) = 8$ | $k_1(E, p_1) = 9$ |
| $p_2 = [g_1] + [g_2]$ | $e'_1 = e$<br>$e'_2 = e$ | $c(e_1\|p_2) = d(e_1, e'_1) = 3$<br>$c(e_2\|p_2) = d(e_2, e'_2) = 5$ | $k_0(E, p_2) = 8$ | $k_1(E, p_2) = 12$ |

Table 11.1: The nearest elements to $E$ not covered by the respective patterns.

*$k_0(E, \cdot)$ and $k_1(E, \cdot)$ for both patterns. In principle, the computation of $c(E|p)$ requires to find the elements $e'_1$ and $e'_2$ which are the nearest elements to $e_1$ and $e_2$, respectively, not covered by the pattern. This is summarised in Table 11.1.*

The computation of $c(\cdot|\cdot)$ shows the same problem that we had for lists. That is, we have that a subgraph can occur several times in a graph, and somehow, all the occurrences of the subgraph must be taken into account in order to compute $c(\cdot|\cdot)$. For instance, considering the pattern $p_1$ from the latter example, the subgraph $g_1$ occurs twice in $e_2$ and this circumstance increases the value $c(E|p_1)$ since the element $e'_2$ are placed further. Furthermore, unless $P = NP$, there is no efficient algorithm to compute $c(\cdot|\cdot)$.

**Corollary 11.** *Given a set of elements $E$ and a pattern $p$ covering $E$, the computation of $c(E|p)$ is a $NP$-*Complete *problem for both $\mathcal{L}_0$ and $\mathcal{L}_1$.*

*Proof.* The computation of $c(\cdot|\cdot)$ is equivalent to the subgraph isomorphism problem [62]. Given two graphs $e$ and $g$, note that if $e \notin Set([g])$ then by definition $c(e|[g]) = 0$ since the nearest element to $e$ not covered by $[g]$ is $e$ and $d(e, e, ) = 0$. Therefore, we can affirm that $g \subset e$ iff $c(e|[g]) > 0$. If $c(\cdot|\cdot)$ can be computed in polynomial time then the subgraph isomorphism problem can be solve in polynomial time as well. But the subgraph isomorphism problem is $NP$-Complete. □

## 11.3 (Minimal) Distance-based Generalisation Operators in $\mathcal{L}_0$

The simplicity of $\mathcal{L}_0$ will allow us to introduce a strong result for the distance-based operators characterisation. Before this, we need the next lemma:

**Lemma 7.** *Given the elements $e_1$, $e_2$ and $e_3$, if $e_3$ is between $e_1$ and $e_2$ then $mcs(e_1, e_2) \subset e_3$.*

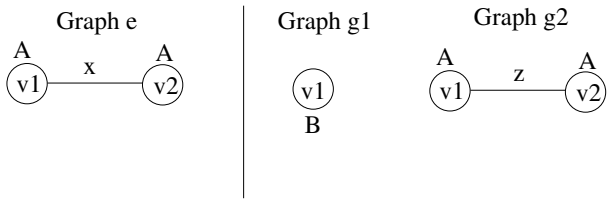


Figure 11.6: **(Left)** The nearest element to $e_1$ and $e_2$ not covered by any of the patterns **(Right)** Graphs involved in the definition of the patterns.

*Proof.* Again, we set $e_i = (V_i, E_i)$ where $(i = 1, 2, 3)$ and $(V_{13,23}, E_{13,23}) = mcs(mcs(e_1, e_3), mcs(e_2, e_3))$. The proof is directly based on the proof of the triangular inequality of the proposed distance. Concretely, we will focus on the Expression (11.10) from Proposition 36. Recall that this expression is equivalent to the triangular inequality. Therefore, if the triangular inequality turns into an equality, this implies that

$$-|V_{12}| - |E_{12}| = -|V_{13,32}| - |E_{13,32}| = (|V_3| - |V_{13}| - |V_{23}|) + (|E_3| - |E_{13}| - |E_{23}|) \tag{11.11}$$

Given that $(V_{13,23}, E_{13,23}) \subset mcs(e_1, e_2)$ and according to the equality above, we can affirm that

$$mcs(e_1, e_2) = (V_{13,23}, E_{13,23}) \subset (V_3, E_3) \tag{11.12}$$

□

The characterisation is stated next.

**Proposition 37.** *Any generalisation operator $\Delta : 2^{\mathcal{U}} \rightarrow \mathcal{L}_0$ is distance-based.*

*Proof.* Let $E$ be a set of finite elements to be generalised, $N$ a nerve function and $[g]$ the pattern computed by $\Delta(E)$. Since $E \subset Set([g])$, the only possibility is that $g \subset mcs(E)$. For any nerve $N(E)$, we can write:

$$\forall (e_i, e_j) \in N(E),\ g \subset mcs(E) \subset mcs(e_i, e_j)$$

which is equivalent to

$$\forall (e_i, e_j) \in N(E),\ Set([mcs(e_i, e_j)]) \subset Set([mcs(E)]) \subset Set([g])$$

and according to Lemma 7, $\Delta$ is distance-based generalisation operator. □

Despite the apparently generality of this latter result, it cannot be interpreted as an inherent property in graphs but a consequence of the distance and the pattern language employed. This means that the proposition does not necessarily hold if we change the distance function or even the pattern language as the next two examples show.

**Example 65.** ***(Preserving the pattern language while changing the distance)*** *Let $d$ be the Bunke's distance and let $e_1$ and $e_2$ be the graphs from Example 61. None of the generalisations of $e_1$ and $e_2$ expressed by the patterns $[g_1]$, $[g_2]$, $[g_3]$ (where $g_1$, $g_2$ and $g_3$ are the graphs depicted in Figure 11.7) is distance-based. Namely,*

1. *if we take* $[g_1]$*, the graph* $g_2$ *is between* $e_1$ *and* $e_2$ *but it is not covered by the pattern.*
2. *if we take* $[g_2]$*, the graph* $g_1$ *is between* $e_1$ *and* $e_2$ *but it is not covered by the pattern.*
3. *if we take* $[g_3]$*, neither* $g_1$ *nor* $g_2$ *are covered by the pattern.*

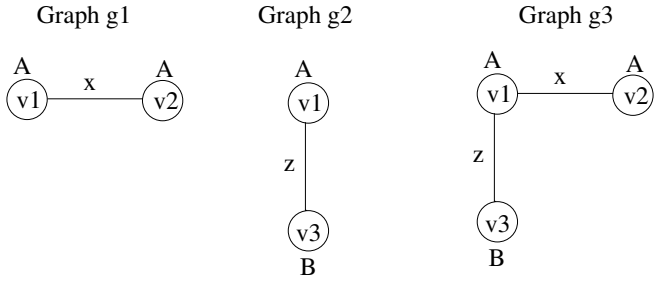


Figure 11.7: Graphs to be used as patterns.

**Example 66.** ***(Preserving the distance while changing the pattern language)*** *Now we introduce a new pattern language* $\mathcal{L}$ *consisting of all the undirected connected graphs built from the extended alphabet* $\Sigma \cup V$ *where* $\Sigma$ *contains the vertex and the edge labels and* $V = \{V_1, V_2, \ldots\}$ *is a numerable set of variables. Each variable represents a vertex or an edge label. If* $V$ *is* $\lambda$ *then the edge does not exist or the vertex with its incident edges do not exist. Let us see that a patter built from the mcs of two graphs is not always distance-based.*

*Given the graphs* $e_1$ *and* $e_2$ *and the pattern* $p$ *depicted in Figure 11.8, although* $p$ *generalises* $e_1$ *and* $e_2$ *and* $p$ *is built up from the* $mcs(e_1, e_2)$*,* $p$ *is not distance-based since the graph* $e$ *is between* $e_1$ *and* $e_2$ *and this is not covered by* $p$*.*

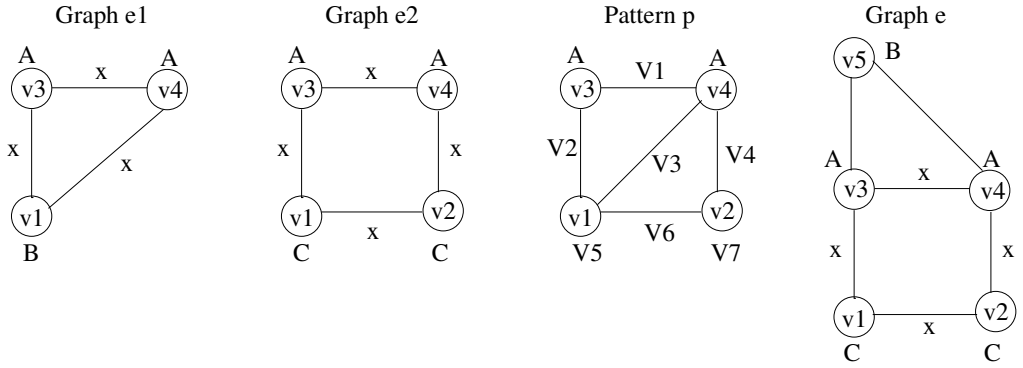


Figure 11.8: A generalisation of $e_1$ and $e_2$ based on the $mcs(e_1, e_2)$ which is not distance-based.

*Observe that this happens because when variables are included in the pattern language (as we did in sets and lists) then, in order to define distance-based operators we must also take into account how the variables appear in the pattern.*

The main advantage derived from the proposed distance is that, although there might not be an efficient algorithm for its computation, there exists distance-based operators which can be implemented in polynomial time. For instance, $\Delta$ can be defined by means of any polynomial-time algorithm which approaches the *mcs* [7, 130, 71]. Unfortunately, things are not so direct as for the Bunke's distance because according to [44], $\Delta(E) = [g]$ is distance-based iff $g \subset mcs(g_1, \ldots, g_n)$ where $\{g_i\}_{i=1}^n$ is the set of all the possible *mcvis* of $E$. In other words, not all the common subgraphs yield a distance-based generalisation. This has already been illustrated in Example 65. Indeed, the only admissible distance-based generalisation for this example is "*all the graphs containing a vertex labelled with the tag A*".

Having characterised the distance-based operators, the next step copes with identifying which of them are minimal. Since all the operators are distance-based, this time, the operator which minimises the cost function ($\Delta^0$) will surely be the minimal. An attempt in this line is as follows:

**Proposition 38.** *Let $E$ be the set of elements to be generalised, if $\Delta(E) = mcs(E)$ then $\Delta$ is a mg operator.*

*Proof.* Let $\Delta' \neq \Delta$ be a distance-based operator. By the definition of *mcs*, we have that $Set(\Delta(E)) \subset Set(\Delta'(E))$ and given that $k_0$ is the semantic preserving cost function then $k_0(E, \Delta(E)) \leq k_0(E, \Delta'(E))$. □

**(Remark 1)** The computation of $\Delta$ is a *NP-Complete* problem because it is equivalent to the *mcs* problem [62].

The *mdbg* operator is not unique, for instance, taking the graphs from Example 61, the subgraph of $e_1$ defined as $(V = \{v_1, v_2\}, E = \{(v_1, v_2)\})$ yields a *mg* generalisation different to the one stated by the previous proposition. This suggests the possibility of investigating how to define other *mdbg* operators and see whether they can be obtained in polynomial time. To this end, we need the following definition along with some preliminary theoretical results.

**Definition 30.** ***(mcs-representative)*** *Given a set of elements $E$, we will say that a graph $g$ is a mcs-representative graph of $E$, if $O^{dis}_{g,mcs(E)} = 1$ and $g \not\subset e_i - mcs(E)$ for every $e_i \in E$.*

Let us see that *mdbg* operators can be defined in terms of the *mcs*-representative graphs. Before, we need the following lemma.

**Lemma 8.** *Let $g$, $e = (V, E)$ and $e' = (V', E')$ be three graphs where $g \subset e$. If $c(e|[g]) = d(e, e')$ then $e'$ is the maximum subgraph of $e$ such that $g \not\subset e'$.*

*Proof.* Let $e'$ be the nearest element to $e$ not covered by $[g]$ and let $(f : e \to e', \hat{V}, \hat{V}', \hat{E}, \hat{E}')$ the *ecgm* with minimal cost from $e$ to $e'$. As $f$ is a bijective function, then for every occurrence $g$ in $e$, $g \not\subset (\hat{V}, \hat{E})$, otherwise we would have $g \subset (\hat{V}', \hat{E}') \subset e'$. Now, suppose that $(\hat{V}', \hat{E}') \subsetneq e'$. This means that the graph $(\hat{V}', \hat{E}')$ does not contain any occurrence of $g$, as we have already seen, and that $d(e, (\hat{V}', \hat{E}')) < d(e, e')$ which is not possible by definition of $e'$. Therefore, $e' \subset e$. □

This leads to the following corollary.

**Corollary 12.** *Let $g$, $e$ and $e'$ be three graphs where $g \subset e$. If $c(e|[g]) = d(e, e')$ then there exist a sequence $S = (d_1, \ldots, d_k)$ of deletion operations such that $S(e) = s'$, $O^{dis}_{g,e} \leq k \leq O_{g,e}$ and $d(e, e') = cost(S)$.*

*Proof.* From the Lemma above, this sequence of operation exists since $e'$ is a subgraph of $e$. It is also obvious that at least one deletion operation is needed for every disjoint occurrence of $g$ in $e$, and that, at most, we need as many deletion operations as occurrences of $g$ in $e$. Finally, $d(e, e') = cost(S)$ by definition of the distance. □

**Proposition 39.** *Given a set of elements $E$ and a generalisation operator $\Delta$, if $\Delta$ is minimal then $\Delta(E) = [g]$ where $g$ is a mcs-representative graph of $E$.*

*Proof.* Given $e$ in $E$, we set $e'$ and $e''$ such that $c(e|[mcs(E)]) = d(e, e')$ and $c(e|[g]) = d(e, e'')$. We proceed by contradiction. Thus suppose that $g$ is not a *mcs*-representative graph of $E$. Then, two non-exclusive cases can be given:
**(case 1)** $O^{dis}_{g,mcs(E)} > 1$: according to Corollary 12 there exist two sequences of removal operations $S'$ and $S''$ such that

$$S'(e) = e' \text{ and } d(e, e') = cost(S')$$

and

$$S''(e) = e'' \text{ and } d(e, e'') = cost(S'')$$

Since $O^{dis}_{g,mcs(E)} > 1$ there will exist a sequence of deletion operations $\hat{S} \subsetneq S''$ such that $S''(e)$ yields a new graph which does not contain any occurrence of $mcs(E)$. This leads to:

$$d(e, e'') = cost(S'') > cost(\hat{S}) \geq cost(S') = d(e, e')$$

A similar reasoning can be used in the following case.
**(case 2)** There exists $e \in E$ such that $O_{g,e-mcs(E)} \geq 1$: it means that $O^{dis}_{g,e} > O_{mcs(E),e}$ and then according to Corollary 12 the number of deletion operations in order to obtain $e''$ from $e$ will be greater than the number of deletion operations to obtain $e'$ from $e$ which leads to $d(e, e'') > d(e, e')$. □

There is a strong intuition that the computation of any minimal operator might be a *NP-Complete* problem in $\mathcal{L}_0$. This is based on the fact that the minimal operators are defined in terms of the *mcs*-representative graph, which in turn, relies on the concept of *mcs*. Despite this argumentation, we have not been able to provide a full proof of this. What we have is a sketch of a proof based on the idea that if one *mcs*-representative graph of two graphs can be computed in polynomial time then we could compute the *mcs* of these two graphs in polynomial time as well.

**Conjecture 1.** *Let $\Delta$ be a mg operator. Unless $P = NP$, $\Delta$ cannot be computed in polynomial time.*

*Proof.* **(sketch)** We proceed by contradiction. We will focus on binary generalisations. Thus for every $e_1$ and $e_2$, if $\Delta(e_1, e_2) = [g]$ can be computed in polynomial time then we will outline that the $mcs(e_1, e_2)$ could be computed in polynomial time as well.

The assumption is made over the algorithm $\mathcal{A}$ implementing $\Delta$. Namely, $\mathcal{A}$ must somehow explore both $e_1$ and $e_2$ in order to obtain $g$. $\mathcal{A}$ could also return the functions $f_1$ and $f_2$ from which $g \subset_{f_1} e_1$ and $g \subset_{f_2} e_2$. If it happens, the strategy to obtain the *mcs* is the next one:

1. Store $(g, f_1, f_2)$
2. Let $e'_1 = e_1 -_{f_1} g$ and $e'_2 = e_2 -_{f_2} g$.

3. Compute $\Delta(e_1', e_2') = [g']$ and store $(g', f_1', f_2')$.

4. Iterate the process until $\Delta(e_1', e_2') = \lambda$.

5. Construct $mcs(e_1, e_2)$ from $(g, f_1, f_2)$, $(g', f_1', f_2')$,...

   (a) Obtain the edges linking whatever $g^{i)}$ and $g^{j)}$ which takes place in $mcs(e_1, e_2)$.

Since $\Delta$ has a polynomial cost, the number of iterations depends sublinearly on the number of vertices in $e_1$ and $e_2$ and step $5a$ has a sublinear cost w.r.t. the number of edges in $e_1$ and $e_2$. Then the $mcs(e_1, e_2)$ could be obtained in polynomial time. □

## 11.4 (Minimal) Distance-based Generalisation Operators in $\mathcal{L}_1$

As we proceed for the rest of data types, the distance-based operators, that we define for $\mathcal{L}_1$, result from the adaption of $\Delta_N$ for graphs. To do this it is enough to define a binary distance-based operator for graphs and plug it in $\Delta_N$. The problem of using those binary operators defined for $\mathcal{L}_0$ is that they yield a not useful generalisation when the elements involved have nothing in common. We can overcome this by means of the following schema.

**Proposition 40.** *Let $\Delta'$ be a distance-based operator in $\mathcal{L}_0$. Then the generalisation operator*

$$\Delta^b(e_1, e_2) = \begin{cases} \Delta(\{e_1, e_2\}), & mcs(e_1, e_2) \neq \lambda \\ e_1 + e_2, & otherwise. \end{cases}$$

*is a binary distance-based operator in $\mathcal{L}_1$.*

*Proof.* It is enough to see that if $mcs(e_1, e_2) = \lambda$ then there is not any element between $e_1 = (V_1, E_1)$ and $e_2 = (V_2, E_2)$. Given a graph $e_3 = (V_3, E_3)$, we will proceed by cases:
**(Case 1)** $mcs(e_1, e_3) = mcs(e_3, e_2) = \lambda$: Applying the Proposition 36.

$$d(e_1, e_3)+d(e_3, e_1) = |V_1|+|V_3|+|E_1|+|E_3|+|V_3|+|V_2|+|E_3|+|E_2| > |V_1|+|E_1|+|V_2|+|E_2| = d(e_1, e_2) \tag{11.13}$$

**(Case 2)**: $mcs(e_1, e_3) \neq \lambda$ or $mcs(e_3, e_2) \neq \lambda$: we set $(V_{13}, E_{13}) = mcs(e_1, e_3)$ and $(V_{23}, E_{23}) = mcs(e_2, e_3)$. Let us prove that

$$(|V_3| - |V_{13}| - |V_{23}|) + (|E_3| - |E_{13}| - |E_{23}|) > 0 \tag{11.14}$$

Since $(V_{13}, E_{13}) \subset (V_3, E_3)$, $(V_{23}, E_{23}) \subset (V_3, E_3)$ then we can write

$$|V_3| - |V_{13}| - |V_{23}| \geq 0 \tag{11.15}$$

Additionally $mcs((V_{13}, E_{13}), (V_{23}, E_{23})) = \lambda$ because $e_1$ and $e_2$ have nothing in common and given that $e_3$ is a connected graph, there will be extra edges connecting both subgraphs $(V_{13}, E_{13})$ and $(V_{23}, E_{23})$ in the graph $(V_3, E_3)$. This motivates that

$$|E_3| - |E_{13}| - |E_{23}| > 0 \tag{11.16}$$

Thus the Expression (11.14) holds. Again, applying Proposition 36.

$$\begin{array}{rcl} d(e_1,e_3)+d(e_3,e_2) & = & |V_1|+|V_3|-2|V_{13}|+|E_1|+|E_3|-2|E_{13}| \\ & + & |V_2|+|V_3|-2|V_{23}|+|E_2|+|E_3|-2|E_{23}| \\ & > & |V_1|+|E_1|+|V_2|+|E_2| = d(e_1,e_2) \end{array} \tag{11.17}$$

□

Having arrived to this point, we have seen that in the language $\mathcal{L}_0$ those algorithms approaching the *mcs* could be utilised to implement distance-based operators. The next proposition can be regarded as an attempt to extend this in $\mathcal{L}_1$. The idea is that in the literature we can find several algorithms which extract subgraphs from a graph data base. (e.g. frequent subgraph mining algorithms [170, 77]). What we want is to find a property these algorithms must satisfy in order to be considered distance-based operators. The best result we have found is the next one:

**Proposition 41.** *Given a set of elements $E$ to be generalised and given the pattern $p' = \sum_{i=1}^{n} p'_i$ returned by a frequent subgraph mining algorithm. Let us define a binary relation $R$ among the pattern $p'_i$ in the following way:*

1. *$p'_i R p'_j$ iff $E \cap Set(p'_i) \cap Set(p'_j) \neq \emptyset$*
2. *$p'_i R p'_j$ iff $\exists e_i \in Set(p'_i), e_j \in Set(p'_j)$ such that $mcs(e_i, e_j) = \lambda$.*

*then $p'$ is a distance-based generalisation of $E$ if the graph corresponding to the binary relation $R$ is connected.*

*Proof.* Trivial. First we construct a nerve from the binary relation $R$. Let $P_1 \subset E$ containing all the elements covered by $p'_1$. According to Proposition 37 $p'_1$ is distance-based w.r.t. any nerve defined over $P_1$. We call this nerve $N(P_1)$. We repeat the process over $E$ until all the patterns $p'_i$ have been explored. What we have at the end is a sequence of nerves $N(P_1), \ldots, N(P_n)$. If $p'_i R p'_j$ (case 1) then there exists and edge linking the nerves $N(P_i)$ and $N(P_j)$. Otherwise we add the link between the elements $e_i \in P_i$ and $e_j \in P_j$. Since the graph of the relation $R$ is connected we can build up a nerve of $E$ from all the nerves $N(P_i)$. Hence $p'$ is distance-based w.r.t. $E$. □

Regarding the minimality of the distance-based operators, we know those operators such as $\Delta_N$ tend to overfit the data. Thus we need a method which permit us locate the *mg*. The best result we can provide is an exhaustive search algorithm which starts from a concrete $\Delta_N$ operator.

**Proposition 42.** *Let $E$ be a set of elements to be generalised and let $N(E)$ be a nerve. Given the binary operator $\Delta^b$ defined in Proposition 40 which is defined from the binary operator in $\mathcal{L}_0$ $\Delta'_{e_i,e_j} = mcs(e_i, e_j)$. Then the generalisation computed by $\Delta_N$ using $\Delta^b$ as a binary operator, satisfies*

$$Set(\Delta_N(E)) \subset Set(p = \sum_{i=1}^{n} p_i)$$

*where $p$ is the mg of $E$ w.r.t. $N$.*

*Proof.* If $\Delta(e_i, e_j) \neq [\lambda]$ we know that $mcs(e_i, e_j)$ is between $e_i$ and $e_j$. As $p$ is distance-based then there exists a $p_i$ such that $mcs(e_i, e_j) \in Set(p_i)$. This implies that

$$Set(\Delta(e_i, e_j)) = Set([mcs(e_i, e_j)]) \subset Set(p_i)$$

Otherwise, $\Delta(e_i, e_j) = [e_i] + [e_j]$ and as $p$ is distance-based there exist the patterns $p_k$ and $p_l$ where $e_i \in Set(p_k)$ and $e_j \in Set(p_l)$ which is equivalent to

$$Set(\Delta(e_i, e_j)) = Set([e_i]) \cup Set([e_j]) \subset Set(p_k) \cup Set(p_l)$$

□

## 11.5 Discussion

In this chapter, we propose a distance for graphs consisting of a slightly variation of H. Bunke's distance published in [19]. This modification will allow us to relate the definition of distance-based operators to algorithms existing in the literature used to approximate the maximum common subgraph or those ones used for mining data bases of graphs. In other words, algorithms dealing with the computation of the maximum common subgraph and algorithms used for mining data bases of graphs can be used as implementations of *dbg* operators. This generic result does not hold when the original H. Bunke's distance is employed.

# Part IV

# Distance-based Generalisation Operators in Unsupervised and Supervised Learning: Practical Examples

# Chapter 12

# An Unsupervised Learning Example: Distance-based Generalisation Operators for Conceptual Clustering

In the previous chapters we have defined distance-based generalisation operators for the most common data types and distances. In this chapter we present experiments over a toy data set in order to illustrate how our framework works for multiple data representations, pattern languages and distances. Concretely, we propose a symbolic unsupervised method capable to deal with different data representations. This method is just a hierarchical clustering algorithm endowed with a distance-based operator. Therefore, if the data representation is changed, the same algorithm can be used by simply providing an appropriate distance and an appropriate distance-based generalisation operator. Finally, we conclude this chapter with a discussion about the patterns obtained from the different data representations.

## 12.1 Description of the Experimental Setting

The data set we are working with contains 44 titles corresponding to some of the Machine Learning Journal editor's publications from the DBLP bibliography server (as queried in April 2007). We are interested in finding clusters in these publications according to the words contained in the titles. Our purpose is twofold: on the one hand, we aim to extract distance-based patterns describing those clusters computed by an agglomerative hierarchical clustering algorithm; on the other hand, we want to compare the different cluster descriptions that are obtained when several distance functions, pattern languages and data representations are employed.

First, we need to prepare the data. The preprocessing task simply focuses on removing the stop words ("a","an", "the", etc.) from the data set. To find the groups in the data, we employ a clustering algorithm that builds the hierarchy from the individual elements by progressively merging clusters. One of the key steps of the algorithm is to determine what clusters must be joined. This is

given by the so-called *average linkage criterion*. Basically, each cluster is represented by a prototype (the element in the cluster whose average distance to the rest is lowest), then, the pair of clusters whose prototypes are nearer will be joined. We employ the following stopping criterion. We define the radius of a cluster as the average distance from all the elements of the cluster to its prototype. Then, the two clusters $C_1$ and $C_2$ that have the nearest prototypes will be joined, if the radius of $C_1$ plus the radius of $C_2$ is equal to or greater than $l * d$, where $d$ is the distance between the prototypes and $l$ is a coefficient in $]0, 1]$ (see Figure 12.1). The algorithm finishes when no pair of clusters satisfies this condition.

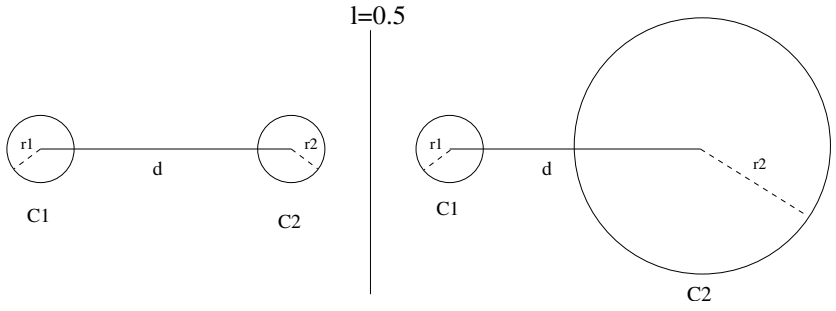


Figure 12.1: By fixing $l = 0.5$, the clusters $C1$ and $C2$ on the left side will not be joined. The clusters on the right side will be joined.

Since the clustering algorithm is based on the proximity between the prototypes, the star nerve has been set for all the experiments. With this simple and general clustering method, we will see how different representation languages and distances will provide different patterns. There are several appropriate representations for data of this kind. We, then consider three possibilities: lists of words, tuples, and sets of words. All these patterns, however, will have something in common: all of them will be distance-based.

## 12.2 List of Words

In this case, the edit distance is used to measure the similarity between two given titles, where each word in the title is treated as a single symbol. Consequently, if we have the titles $l_1 = [word_1]$ and $l_2 = [word_2]$ such that $word_1 \neq word_2$, then $d(l_1, l_2) = 2$.

We first employ the pattern language $\mathcal{L}_0$ along with the generalisation operator defined in Corollary 5 (binary version of the operator) and Corollary 6 ($n$-ary version of the operator) respectively. The obtained results are reported in Tables 12.1 and 12.2:

Since the titles are quite different, the data are uniformly distributed and seldom lie in well-separated clusters. Consequently, titles with no words in common are likely to be grouped in the same cluster, and its corresponding pattern only contains variable symbols.

In order to change those patterns made up of only variables for more informative ones, we introduce the pattern language $\mathcal{L}_1$ (see Section 8.1 from Chapter 8) along with an extension of the generalisation operator defined in Corollary 6 for the pattern language $\mathcal{L}_1$ (see Example 49 in Chapter 8). This latter operator requires a cost function. Thus, we define $k(E, p) = c(p) + c'(E|p)$,

| Number of clusters | | | | |
|---|---|---|---|---|
| $l = 0.5$<br>1 clus. | $l = 0.75$<br>20 clus. | $l = 0.85$<br>40 clus. | $l = 0.95$<br>40 clus. | $l = 1.0$<br>40 clus. |
| | 1 clus. with 24 elem.<br>1 clus. with 2 elem.<br>18 clus. with 1 elem. | 4 clus. with 2 elem.<br>36 clus. with 1 elem. | | |

Table 12.1: Clusters obtained by using a list representation

| | |
|---|---|
| $l = 0.5$ | (clus. with 44 elem.) $p_1$ =[$V^{24}$] |
| $l = 0.75$ | (clus. with 24 elem.) $p_1$ =[$V^{12}$]<br>(clus with. 2 elem.) $p_2$ =[$V^3$, active, feature, value, acquisition, $V^2$] |

Table 12.2: Patterns covering more than two element obtained using $\mathcal{L}_0$ for list pattern representation.

where the syntactic part $c(p)$ is equal to the number of both ground and variable symbols in $p$, and $c'(E|p)$ is the function defined in Section 8.1 from Chapter 8.

Recall that this semantic cost function is an approximation of the the first cost function defined in Table 5.2 because the computation of this latter cost function is rather difficult and computational expensive. Also remember that the simplified version relied on the fact that the sequence of ground symbols in a pattern comes up only once in the lists covered by this pattern. This assumption makes sense in this case because it is unlikely that a sequence of words occurs more than once in a title. Now, we repeat the experiments with $\mathcal{L}_1$ and the above-mentioned cost function. The patterns obtained are:

| | |
|---|---|
| $l = 0.5$ | (clus. with 44 elem.) $p_1$=[$V^3$, learning, $V^8$] (7 elem.) + [$V^{12}$, 'inductive', $V^{11}$] (7 elem.)+<br>[$V^3$, active,$V^7$] (4 elem.) + [$V^2$, distributed, $V^9$] (3 elem.) + ... |
| $l = 0.75$ | (clus. with 24 elem.) $p_1$=[$V^3$, learning, $V^8$] (8 elem.) + [$V^{12}$, 'inductive', $V^{11}$] (8 elem.)+<br>[$V^3$, probability, $V$, ranking, $V^2$] (3 elem.)+<br>[$V$, confidence, bands, $V^3$, empirical, $V^3$] (2 elem.)+<br>[$V^2$, fraud, detection, $V^4$] (2 elem.)+ ...<br>(clus. with 2 elem.) $p_2$ =[$V^3$, active, feature, value, acquisition, $V^2$] |

Table 12.3: Patterns covering more than one element obtained using $\mathcal{L}_1$ for list pattern representation.

Since the distance between two titles depends on their respective length, it might occur that the distance between two titles with different topics is lower than the distance between two titles with the same topic. For instance, given the following titles:

title 1 $\equiv$ [Web, Mining, Web, Semantic Web]
title 2 $\equiv$ [Organizational, Data, Mining, Leveraging, Enterprise, Data Resources, Optimal, Performance]
title 3 $\equiv$ [G, protein, coupled, receptors, mutant, database]

We have that the titles 1 and 2 belong to the same area but the distance between them (12) is

greater than the distance between the titles 1 and 3 (11). To avoid this problem, a possible solution (which is widely used in text mining) is to make the title length equal by adding an artificial word. Next, we report the results obtained by repeating the experiments above using this (see Tables 12.4, 12.5 and 12.6)[1] .

| Number of clusters | | | | |
|---|---|---|---|---|
| $l = 0.5$<br>1 clus. | $l = 0.75$<br>1 clus. | $l = 0.85$<br>1 clus. | $l = 0.95$<br>25 clus. | $l = 1.0$<br>25 clus. |
| | | | 1 clus. with 5 elem.<br>1 clus. with 4 elem.<br>3 clus. with 3 elem.<br>6 clus. with 2 elem.<br>14 clus. wiht 1 elem. | |

Table 12.4: Clusters obtained by using a list representation and by making the title lengths equal.

| | |
|---|---|
| $l = 0.5, \ldots, 0.85$ | (clus. with 44 elem.) $p_1 = [V^{21}]$ |
| $l = 0.95$ | (clus. with 5 elem.) $p_1 = [V^1 7]$<br>(clus. with 4 elem.) $p_2 = [V^3$, data, mining, $V^9]$<br>(clus. with 3 elem.) $p_3 = [V^3$, machine, learning, $V^5]$<br>(clus. with 3 elem.) $p_4 = [V^2$, scaling, inductive, $V^5]$<br>(clus. with 3 elem.) $p_5 = [V^6$, relational, $V^8]$<br>(clus. with 2 elem.) $p_6 =$[inductive, policy, $V^3]$<br>...<br>(clus. with 2 elem.) $p_{11} = [V^3$, active, feature, value, acquisition, $V^2]$ |

Table 12.5: Patterns covering more than one element obtained using $\mathcal{L}_0$ for list pattern representation.

Note that the location of the words in a title might not be a meaningful enough property in order to group them. Therefore, the next two data representations mainly focus on the words in a tittle rather than on their positions.

## 12.3 Tuple of Words

This representation relies on the well-known vector-space model for texts. In this model, each title is represented by a vector of word frequencies. In our case, we simplify this representation by using $0-1$ vectors, where 0 and 1 means absence or not of a word in a title, respectively.

Now, if we view each vector as a tuple built from $X = \{0, 1\}$ and if we can fix a distance function, a pattern language and a generalisation operator in $X$ for each attribute, then we can run a generalisation process on the tuples based on the Proposition 32 from Chapter 10. Thus, we define $d(\cdot, \cdot)$ as the discrete distance and introduce the pattern language $\mathcal{L}= \{0, 1, V\}$, where $V$ is a variable symbol representing 0 or 1. The distance-based generalisation operator is then:

[1]The different number of variables between the pattern generalising the whole data set with equal-length titles and the pattern generalising the whole data set with variable-length titles is due to the prototype is different in each case.

| | |
|---|---|
| $l = 0.5, \ldots, 0.85$ | (clus. with 44 elem.) $p_1$ = [$V^4$, learning, $V^4$] (8 elem.)+ [$V^7$, inductive, $V^4$] (6 elem.)+ [$V^3$, probability, $V$, ranking, $V^2$] (3 elem.) + [$V$, confidence, bands, $V^3$, empirical, $V^3$] (2 elem.) + ... |
| $l = 0.95$ | (clus. with 5 elem.) $p_1$ =[$V^5$, learning, $V^{11}$]+[$V^5$, probability, $V^2$, ranking]<br>(clus. with 4 elem.) $p_2$ =[$V^3$, data, mining, $V^9$]<br>(clus. with 3 elem.) $p_3$ =[$V^3$, machine, learning, $V^5$]<br>(clus. with 3 elem.) $p_4$ =[$V^2$, scaling, inductive, $V^5$]<br>(clus. with 3 elem.) $p_5$ =[$V^6$, relational, $V^8$]<br>(clus. with 2 elem.) $p_6$ =[inductive, policy, $V^3$]<br>...<br>(clus. with 2 elem.) $p_{11}$ =[$V^3$, active, feature, value, acquisition, $V^2$] |

Table 12.6: Patterns covering more than one element obtained using $\mathcal{L}_1$ for list pattern representation.

$$\Delta(e_1, e_2) = \begin{cases} e_1 & , e_1 = e_2 \\ V & , otherwise \end{cases}$$

According to Proposition 32, a pattern over the space of tuples is a $n$-dimensional tuple over $\{0, 1, V\}$, such that 1 stands for all the titles in $E$ that have this word in common and 0 stands for when the titles do not have this word in common. In contrast, the symbol $V$ denotes that this word exists only in some titles but not in all.

For example, given the dictionary $\{inductive, logic, programming\}$, the pattern $(1, V, 0)$ represents all the titles containing the word "inductive" but not the word "programming". We use the tokens "WITH" and "WITHOUT" for 1 and 0, respectively. Next, we show the results obtained (Tables 12.7 and 12.8). The patterns that are identified match many of the patterns discovered by the list representation, but important differences are also found.

| Number of clusters | | | | |
|---|---|---|---|---|
| $l = 0.5$<br>1 clus. | $l = 0.75$<br>29 clus. | $l = 0.85$<br>39 clus. | $l = 0.95$<br>39 clus. | $l = 1.0$<br>39 clus. |
| | 1 clus. with. 11 elem.<br>5 clus. with 2 elem.<br>23 clus. with 1 elem. | 5 clus. with 2 elem.<br>34 clus. with 1 elem. | | |

Table 12.7: Clusters obtained by using the discrete distance for tuples.

## 12.4 Set of Words

When we represent a title as a set of words, the repetitions of a word must be removed. Note that in this case, the expressiveness for instances are similar to the previous case. Proceeding like in the list representation, the similarity between two titles is given by the symmetric difference where each word in the set is treated as a single symbol. Hence, if we have the sets $s_1 = \{word_1\}$ and $s_2 = \{word_1, word_2\}$ such that $word_1 \neq word_2$, then $d(s_1, s_2) = 1$. We employ the pattern

| | |
|---|---|
| $l = 0.5$ | (clus. with 44 elem.) $p_1$ =WITH:<br>WITHOUT: |
| $l = 0.75$ | (clus. with 11 elem.) $p_1$ =WITH:<br>WITHOUT: accuracy, acquisition,..., weakening<br>(clus. with 2 elem.) $p_2$ =WITH: distributed, learning, machine, scaling<br>WITHOUT: accuracy, acquisition,..., weakening<br>(clus. with 2 elem.) $p_3$ =WITH: detection, fraud<br>WITHOUT: accuracy, acquisition,..., weakening<br>(clus. with 2 elem.) $p_4$ =WITH: active, class, estimation, probability, ranking<br>WITHOUT: accuracy, action,..., weakening<br>(clus. with 2 elem.) $p_5$ =WITH: bands, confidence, empirical, roc<br>WITHOUT: accuracy, action,..., weakening<br>(clus. with 2 elem.) $p_6$ =WITH: acquisition, active, feature, value<br>WITHOUT: accuracy, action,..., weakening |
| $l = 0.85$ | (clus. with 2 elem.) $p_1$ =WITH: inductive, policy<br>WITHOUT: accuracy, acquisition,..., weakening<br>(clus. with 2 elem.) $p_2$ =WITH: algorithms, inductive, scaling<br>WITHOUT: accuracy, acquisition,..., weakening<br>(clus. with 2 elem.) $p_3$ =WITH: classification, environments, imprecise, robust<br>WITHOUT: accuracy, acquisition,..., weakening<br>(clus. with 2 elem.) $p_4$ =WITH: active, class, estimation, probability, ranking<br>WITHOUT: accuracy, acquisition,..., weakening<br>(clus. with 2 elem.) $p_5$ =WITH: bands, confidence, empirical, roc<br>WITHOUT: accuracy, acquisition,..., weakening |

Table 12.8: Patterns covering more than one element obtained by using a tuple representation.

language $\mathcal{L}_0$ defined in Section 7.1 from Chapter 7 along with the generalisation operator defined in Proposition 9 from the same chapter. The obtained results are shown in Tables 12.9 and 12.10. Although the examples have a representation that is biunivocal w.r.t. the tuple representation, the language $\mathcal{L}_0$ and the distance make the results slightly different to the tuple case, in the sense that patterns in tuples provide more information than patterns in sets. This is so because in the case of tuples, the words in the dictionary which are not used by two titles take also part in the pattern (key word 'WITHOUT') while this does not happen in sets. However, if left out this part of the patterns in tuples, patterns over tuples and sets would be equivalent.

| Number of clusters | | | | |
|---|---|---|---|---|
| $l = 0.5$<br>1 clus. | $l = 0.75$<br>29 clus. | $l = 0.85$<br>39 clus. | $l = 0.95$<br>39 clus. | $l = 1.0$<br>39 clus. |
| | 1 clus. with. 11 elem.<br>5 clus. with 2 elem.<br>23 clus. with 1 elem. | 5 clus. with 2 elem.<br>34 clus. with 1 elem. | | |

Table 12.9: Clusters obtained by using the symmetric distance for sets.

| | |
|---|---|
| $l = 0.5$ | (clus. with 44 elem.) $p_1 = \{V^{21}\}$ |
| $l = 0.75$ | (clus. with 11 elem.) $p_1 = \{V^9\}$<br>(clus. with 2 elem.) $p_2 = \{$scaling, machine, distributed, learning, $V^4\}$<br>(clus. with 2 elem.) $p_3 = \{$detection, fraud, $V^5\}$<br>(clus. with 2 elem.) $p_4 = \{$ranking, probability, active, estimation, class, $V^2\}$<br>(clus. with 2 elem.) $p_5 = \{$bands, confidence, roc, empirical, $V^4\}$<br>(clus. with 2 elem.) $p_6 = \{$active, value, feature, acquisition, $V^4\}$ |
| $l = 0.85$ | (clus. with 2 elem.) $p_1 = \{$policy, inductive, $V^3\}$<br>(clus. with 2 elem.) $p_2 = \{$scaling, algorithms, inductive, $V^3\}$<br>(clus. with 2 elem.) $p_3 = \{$robust, imprecise, environments, classification, $V\}$<br>(clus. with 2 elem.) $p_4 = \{$ranking, probability, active, estimation, class, $V^2\}$<br>(clus. with 2 elem.) $p_5 = \{$bands, confidence, roc, empirical, $V^4\}$ |

Table 12.10: Patterns covering more than one element obtained using a set representation.

## 12.5 Discussion

This example, although simple, allows us to show some relevant traits of our framework. As expected, data representation has a great influence on the patterns obtained. This is not much surprising since when data representation changes, metric also changes and i.e. data tends to be grouped differently, thus affecting the patterns. This is observed in pattern $p_1$ in Table 12.6 ($l = 0.95$) where the subpattern $[V^5, probability, V^2, ranking]$ covers the titles:

$$\begin{array}{rcl} t_1 & \equiv & \text{Active Sampling for Class Probability Estimation and Ranking} \\ t_2 & \equiv & \text{Tree Induction for Probability-Based Ranking} \\ t_3 & \equiv & \text{Active Learning for Class Probability Estimation and Ranking} \end{array}$$

However, when titles are viewed as sets (or tuples), $t_2$ is not grouped together $t_1$ and $t_3$, which results in the pattern $\{ranking, probability, active, estimation, class, V^2\}$ (see pattern $p_4$ in Table 12.10 with $l = 0.85$ or pattern $p_4$ in Table 12.8 with $l = 0.85$).

Nonetheless, what it is novel here is that patterns are also influenced by the distance itself. Obviously, given that patterns are distance-based, when distance is changed, the patterns can also change. For instance, the titles below

$$\begin{array}{rcl} t_1 & \equiv & \text{Confidence Bands for ROC Curves: Methods and an Empirical Study} \\ t_2 & \equiv & \text{ROC confidence bands: an empirical evaluation} \end{array}$$

are put in the same cluster in all the representations (lists, tuples and sets), however, the pattern for lists (see Table 12.6 with $l = 0.5$) differs from the pattern for tuples and sets (see Tables 12.8 and 12.10 with $l = 0.85$), with these last two being equivalent. That is,

$$\begin{array}{lcl} \text{lists} & = & [V, \text{confidence, bands, } V^3, \text{empirical, } V^3] \\ \text{tuples} & = & \text{WITH: bands, confidence, empirical, roc} \\ \text{sets} & = & \{\text{bands, confidence, roc, empirical, } V^4\} \end{array}$$

Even though both titles have the word *ROC*, this word is disregarded when computing the distance between the lists representing the titles. Thus, the pattern that we obtain are in accordance with the distance in that it reveals traits that are taken into account by the distance. The consequence

is that a pattern saying *"all the title having the words ROC and ..."* would be admissible for the elements in the cluster but not for the distance. However, things are different when we move to tuples or sets because the word *ROC* is now considered by the distance.

Another aspect to be mentioned is the role played by $\mathcal{L}_1$ for pattern extraction from clusters. As pointed out, data is quite uniformly distributed, which makes cluster discovering a hard task. In fact, the clusters returned by the algorithm are rather rough since some of them are large and contain titles with no words in common. Thus, the use of distance-based operators defined in $\mathcal{L}_1$ can be viewed as a refinement step which overcomes the clustering algorithm limitations by "detecting" subclusters of titles with common words inside large clusters. Something which cannot be considered when $\mathcal{L}_0$ is employed.

# Chapter 13

# A Supervised Learning Example: a Distance-based Decision Tree

The previous chapter outlines what possibilities our setting can offer for model extraction in unsupervised learning. In connection with this, it would also be useful to study how our framework could be applied for supervised learning. An immediate proposal would be run a (lazy) distance-based method for classification and then apply *dbg* operators in post processing for model extraction.

However, as seen in Chapter 3, although widely known approaches are capable of handling complex (structured, multi-relational) data in supervised contexts, not most of them are distance-based. The most popular method is $k$-NN but its predictions rely on local analysis leading this to multiple local symbolic models rather than a single global model. Thus, we would need a distance-based method which can provide a more suitable input for *dbg* operators in order to obtain a global model from data.

This chapter contains a preliminary study about a novel method which fits the requirement above. To this end, we try a flexible distance-based decision tree learning algorithm applicable either to classical attribute-value problems, to complex data scenarios or, more importantly, to mixtures of both situations.

## 13.1 Decision Trees and Distances

As mentioned in Chapter 2, decision tree learning methods could be classified according to those datatypes they can deal with. The first decision tree learning systems, such as ID3 [131], only dealt with nominal attributes. Later, other systems such as CART [18] and C4.5 [131] introduced numerical attributes as well. Since then, many variants and extensions have been introduced. Most remarkable are those concentrated on handling structured data such as FOIL [134], TILDE [15], MRDT [6], higher-order decision trees, etc.

However, those methods which handle complex data, and very specially those coming from relational learning, tend to manage numerical (and sometimes nominal data) not quite properly. In order to design a flexible decision tree learner, there is the possibility to consider a different split construction depending on the type at hand. In fact, the most popular decision tree learning system, C4.5, treats nominal and numerical attributes in a completely different way. Extending

the philosophy of C4.5 to other datatypes, such as sets, lists, multisets, graphs, etc., would require defining specific splits for each data type. This would turn decision tree learning methods into patchy and ad-hoc systems.

In the following preliminary work, we explore what advantages distances and similarity functions in general can offer in order to design more flexible decision trees. Thus, we present a distance-based decision tree learning method which performs splits on any datatype based on the definition of class prototypes. This method is then able to deal with virtually any datatype provided we can define a distance on it (and this is usually the case for virtually any datatype, since many distance metrics have been defined in the literature, as seen in Chapter 3). It is also important to highlight that, as a consequence of this, the method returns a tree of class prototypes where each class prototype represents a node in the tree. This means that an example drops in one given node depending on its proximity to the class prototype and i.e. examples are not required for classification, being the model (the tree of prototypes) enough for this purpose.

Unfortunately, the flexibility of the method affects the comprehensibility of the model obtained because this is not as intelligible as conventional decision trees are. That is, in the conventional case we have rules of the form *"if the attribute i-th is equal to... and the attribute j-th is lower than... then the example belongs to class c"* while the rules in the model which is return by DBDT are of the form *"if the value of the attribute i-th is nearer to the prototype $p_i$ and the value of the attribute j-th is nearer... then the example belongs to class c"*. However, we must also say that the tree of prototypes rises the possibility of representing distance-based splits into more familiar symbolic-splits by applying *dbg* operators in post-processing. This possibility, though not implemented yet, will be illustrated in this chapter by means of a toy example. At the moment, we sketch the intuitive idea behind this. For this purpose, it is enough to define *dbg* operators for the different pairs of data representations and distances employed in the problem. Next, we have only to apply over every node in the tree the proper operator. By doing this, we obtain a tree of symbolic patterns (instead of prototypes) describing consistently the nodes induced by the distance-based splits. However, we cannot ensure beforehand that the patterns, which come from the same split, do not overlap. Consequently, the use of *mdbg* operators is recommended as an attempt to mitigate this problem.

## 13.2 The Method

Our proposal is based on the use of distances in a quite similar way as centre splitting does. Centre splitting [162] is a distance-based technique used for the classification of numerical attribute-valued examples, and can be seen as a divide-and-conquer extension of a linear discriminant. Basically, this method consists of dividing the metric evidence space in different regions, where each region is represented by a special point called centre. In every iteration of the process, a centre is calculated for every different class presented in an area. The coordinates of the centre may match to an existing example or not. Then, every example is associated to its nearest centre. Since at the first split usually generates impure regions, the process is iterated for each impure region. The process stops when the obtained regions are totally pure, i.e., the examples placed in one region belong to the same class.

Although the method is efficient and flexible, the major inconvenience is that examples are always managed as a whole. That is, it does not look for components of the tuple which would help to solve the problem in an adequate manner as decision tree does. This appreciation leads us to propose a decision tree inference strategy where partitions are made only taking into account one

attribute at a time. In this way centroids are computed considering only the values of one attribute. This fact allows the centre splitting and decision tree learning techniques to be joined in an elegant way. We call this approach distance-based decision tree learning. Additionally, the fact of using only one attribute at each split, will allow a post-processing to provide symbolic conditions such as $value_1 \leq attribute \leq value_2$.

For this purpose, a similarity space is associated to every attribute. The split is performed in a similar way, but only considering the distances and class distribution of the selected attribute. As usual, an attribute is selected depending on a heuristic function (gain ratio, GINI index, etc.).
The result of this adaptation of centre splitting is not very different from classical decision trees, when attributes are either nominal and numeric. The interesting point comes when we have structured data types for some attributes. In order to handle these types appropriately, we have to determine how the centres are computed. If we want one centre per class, the first idea would be to compute the centre which minimises the distance to the rest of examples for each class. It is important, however, to note that for many data types, the "centre" might not necessarily be inside the original data type. For instance, given a type of lists of constants, the centre of the set $\{[a,a],[a,a,a],[a,a,a,a,a]\}$ is a list containing 3.33 $a$'s, which does not belong to the original data type. Consequently, it is much more reasonable to find a representative example, prototype or median. In the previous case, the most appropriate value would be the list $[a,a,a]$.

### 13.2.1 Algorithm

With this idea of attribute prototypes, we can extend decision tree learning to handle complex structures and data types. Before introducing the algorithm, we require some notation. First, $Attr_{x_j}(e)$ returns the $j$-th attribute of example $e$. Similarly, $Class(e)$ returns the class of example $e$. Additionally, two important functions to reduce the complexity of computing the distances are needed: $Values(S, x_j)$ returns the number of different values for attribute $x_j$ in the sample $S$, and $Card(v, S, x_j)$ returns the number of occurrences of value $v$ in the attribute $x_j$ in the sample $S$, namely, $|\{e \in S : Attr_{x_j}(e) = v\}|$. The function $dist(x, y)$ computes the distances of values $x$ and $y$, attributes which are of the same type. It is assumed that these distances are pre-calculated.

The next algorithm is the main procedure of our method. The inputs are a training dataset and a constant $m$, which is a parameter that limits the number of children per split (if we set $m = 2$ we only have binary partitions):
The previous algorithm is just a typical decision tree learning algorithm which, in this case, determines, for each attribute, a ranked list of prototypes, which will lead to a set of children for each node. The main difference with classical decision tree learners lies in two functions: `Compute_Prototypes` and `Attracts`, which we describe next.

The function `Attracts` just determines which prototype is assigned with a new example. Consequently, this function is not only applied when learning the decision tree but also when using the decision tree to predict new examples, i.e. as the decision rule. Several options are possible here (e.g. considering the density or not), but the `Attracts` function below implements a very simple one. It just returns the closest prototype. In case of a tie, it returns the rightmost prototype.
As suggested at the end of the previous section, *dbg* operators can be used at this point in order to obtain symbolic descriptions of the nodes instead of rules expressed in terms of distances to a prototype. Let us illustrate this possibility by means of an easy example. Imagine that $x$ is a numerical attribute and two prototypes $p_1 = 2$ and $p_2 = 5$ are given. The values of $x$ the examples of this node $N$ are $Values(N, x) = \{-1.5, 1, 3, 4, 5.2, 6\}$ then, after successively invoking

**Input**: $S$ is a set of examples of the form: $(x_1, \ldots, x_n), n \geq 1$,$m$ is the maximum # of children per node.

**begin**
  $C \leftarrow \{Class(e) : e \in S\}$;
  **if** $|C| < 2$ **then** EXIT;
  **foreach** *attribute* $x_j$ **do**
    **if** $Values(x_j, S) < 2$ **then** CONTINUE;
    $ProtList \leftarrow Compute\_Prototypes(x_j, S, m, C)$;
    **if** $Size(ProtList) \leq 1$ **then** EXIT;
    $Split_j \leftarrow \emptyset$;
    **for** $i = 1..length(ProtList)$ **do**
      $\hat{S}_i \leftarrow \{e \in S : i = Attracts(e, ProtList, x_j)\}$ // examples attracted by $i$;
      $Split_j \leftarrow Split_j \cup \hat{S}_i$ // Add a new child;
    **end**
  **end**
  $BestSplit \leftarrow Argmax_{Split_j}(Optimality(Split_j))$ //Gain ratio, GINI, etc. ;
  **foreach** *set* $S_k \in BestSplit$ **do** DBDT($S_k$,$n$);
**end**

**Procedure** `DBDT(`$S$,$m$`)`

**Input**: $e$ an example, $Protlist$ a ranked list of prototypes and $x_j$ a chosen attribute
**Output**: Index of prototype that attracts $e$

**begin**
  **for** $i = 1, .., lenght(ProtList)$ **do**
    $v \leftarrow attr_{x_j}(e)$;
    **if** $\forall k > i, distance(attr_{x_j}(ProtList[i]), v) < distance(attr_{x_j}(ProtList[k]), v)$ **then**
      RETURN $i$;
    **end**
  **end**
**end**

**Function** `Attracts(`$e$,$ProtList$,$x_j$`)`

the function *Attracts* for every value in the set $Values(N, x)$ and the list of prototypes $[p_1, p_2]$, the set $Values(N, x)$ is split into the subsets $N_1 = \{-1.5, 1, 2, 3\}$ and $N_2 = \{4, 5, 5.2, 6\}$. Finally, using the *dbg* operator for numerical data defined in Proposition 5 from Chapter 6 we have the descriptions $\Delta(N_1) = [1.5, 3]$ and $\Delta(N_2) = [4, 6]$.

Finally, the function `Compute_Prototypes` is the most important one. This function can be performed in many different ways. Below, we show one of these possibilities, which just selects the best prototype (in the way that the distances to the prototype for the elements of the same class are minimised), removes its class and the value of the attribute and looks for the next best prototype of a different class and value for the attribute, and so on, until the limit $m$ or the number of classes or attribute values is reached.

**Input**: $x_j$ is the attribute, $S$ is the dataset, $m$ is the maximum # of prototypes, $C$ is the set of classes
**Output**: A ranked list of prototypes

**begin**
  **foreach** *class* $c \in C$ **do**
    $S_c \leftarrow \{e \in S : class(e) = c\}$ // example of class $c$;
    **if** $S_c \neq \emptyset$ **then**
      $V_c \leftarrow Values(x_j, S_c)$;
      **foreach** *element* $v \in V_c$ **do** $MeanDistance_c[v] \leftarrow \frac{\sum_{i \in V_c} distance(v,i) \times Card(i, S_c, x_j)}{|S_c|}$;
    **end**
  **end**
  $UV \leftarrow \emptyset$ // values for the attribute used ;
  $ProtList \leftarrow \emptyset$ // list of prototypes ;
  $RC \leftarrow C$ // remaining classes ;
  $Values \leftarrow Values(x_j, S)$ // different values in $S$ ;
  $Prots \leftarrow min(|C|, m, Values)$ // prototypes of the split ;
  **for** $k = 1, .., Prots$ **do**
    $BestProt \leftarrow Argmin_e \{MeanDistance_c[Attr_{x_j}(e)]\}_{c \in RC, e \in S, Attr_{x_j}(e) \notin UV}$ ;
    $RC \leftarrow RC - Class(BestProt)$;
    $ProtList \leftarrow append(ProtList, BestProt)$;
    $UV \leftarrow UVAttr_{x_j}(BestProt)$;
  **end**
  RETURN $ProtList$;
**end**

**Function** `Compute_Prototypes`($x_j$,$S$,$m$,$C$)

The previous function is the core of the algorithm and it is (jointly with the splitting criterion) the point where efficiency is crucial. The complexity of the previous function is mostly determined by the first part (the first $for$), which is in $\mathcal{O}(|C| \cdot |V_c| \cdot |V_c|)$ where $V_c$ is the number of different values for the attribute. Although this order is cubic, the cardinality of $V_c$ is, in general, small, especially for nominal and for some structured attributes. In the case of numeric data types and types with many different possible values (e.g. sets), the distances can be pre-calculated before calling the procedure `DBDT`. The second part of the function `Compute_Prototypes` (the second $for$) has an *argmin* which,

overall, can be done in $\mathcal{O}(\#Prots \cdot |C| \cdot |V_c|)$, which, in general, is lower than the first part.

It is important to note that distances are computed between attribute values and not between examples. This issue is not only important for obtaining attribute-based decision trees but also, as we have seen, is crucial for efficiency.

### 13.2.2 An Illustrative Example

In this section, we illustrate our proposal by means of an artificial classification problem. Suppose we have a set of three different classes of molecules where each molecule consists of a variable chain of atoms, headed by a molecular group. This group is chosen from a set of four possible molecular groups and they are represented by the capital letters $X$,$Y$,$Z$ and $T$.

The breaking temperature (i.e, the temperature at which the chain is broken) is known as well. Ideally, we would want to learn a set of rules or a classification function which allow us to identify, in a comprehensible way, every group of molecules. Table 13.1 shows the set of examples which are described by means of a numerical attribute (Temp.($C^\circ$)) and a structured attribute (Molecule).

| id | Temp.($C^\circ$) | Molecule | Class |
|---|---|---|---|
| 1 | 30 | $xc^5$ | 1 |
| 2 | 31 | $xc^{35}b$ | 1 |
| 3 | 29 | $xc^{35}bc^5bc^5$ | 1 |
| 4 | 33 | $yc^{27}bc^{13}bc^{10}$ | 1 |
| 5 | 34 | $yc^{33}bc^7$ | 1 |
| 6 | 31 | $yc^5$ | 1 |
| 7 | 51 | $zc^5$ | 2 |
| 8 | 53 | $tc^{12}$ | 2 |
| 9 | 46 | $zc^{10}$ | 2 |
| 10 | 46 | $zc^7$ | 2 |
| 11 | 47 | $zc^9$ | 2 |
| 12 | 49 | $tc^{14}$ | 2 |
| 13 | 55 | $tc^{30}bc^{10}bc^{10}$ | 3 |
| 14 | 49 | $zc^{25}$ | 3 |
| 15 | 50 | $zc^{29}b$ | 3 |
| 16 | 48 | $tc^{30}bc^7$ | 3 |
| 17 | 52 | $tc^{31}b$ | 3 |
| 18 | 53 | $tc^{28}bc^{12}$ | 3 |

Table 13.1: Examples for the molecule problem.

The attribute 'temperature' is a numerical data type and the attribute molecule is just a finite list of symbols belonging to the alphabet $\{b, c, x, y, z, t\}$. The next step is to formulate a metric space associated to every attribute of the problem. We will consider the absolute difference between two real numbers for the temperature attribute, whereas the distance between two molecules will be given by the edit distance, since a molecule is modelled as a list of symbols. For the splitting criterion, we use a criterion based on accuracy.

Initially, both attributes, temperature and molecule, are candidates to be selected for splitting the root node. It is necessary to calculate the proximity criterion in both cases. According to the algorithm, we compute centroids for every class. In this case, the root, we get three centroids per attribute. Next, the most accurate split is selected. In this case, the best partition is given by the first attribute, temperature. The algorithm continues until the decision tree has only pure nodes, i.e. leaves. The result of the process is shown in Figure 13.1.

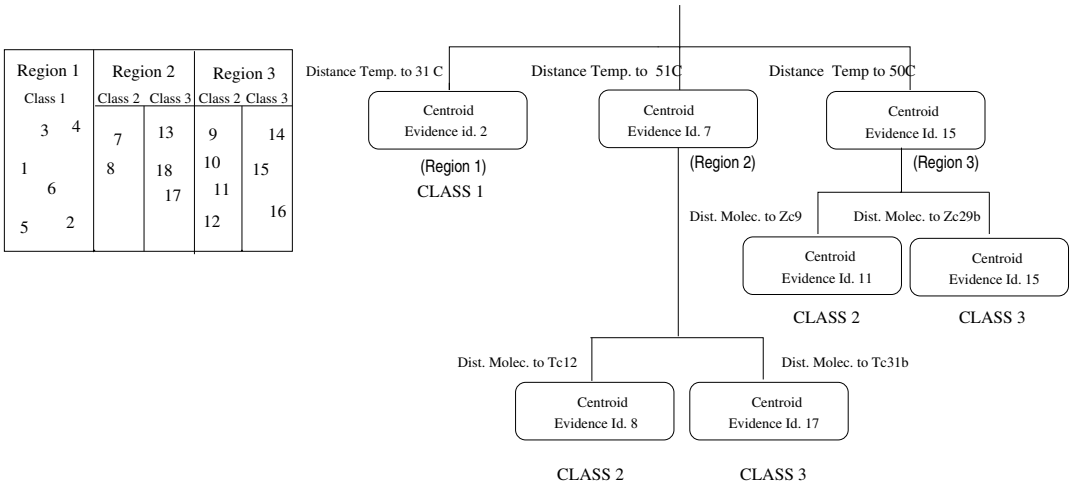


Figure 13.1: The induced distance-based decision tree.

Now, if we were interested in transforming the tree of prototypes into a tree of symbolic patterns, we could apply *dbg* (or if possible, *mdbg*) operators in post-processing. Let us illustrate how this stage could be carried out. First, we would have to define *dbg* (or *mdbg*) operators for all the pairs of data representation and distance involved during the inference of the tree. In our case, we distinguish two combinations: numerical data with the absolute difference (attribute *temperature*) and sequences with the edit distance. For the former combination (attribute *molecule*), we will use the *mdbg* operator which have been employed in the previous section to illustrate the function *Attracts*. This is denoted here as $\Delta_1$. For the latter one, we use the *dbg* operator defined in Corollary 6 from Chapter 8, which is referred here as $\Delta_2$. Since examples are assigned to nodes by a direct comparison with the prototypes, the nerve we fix, in order to generalise the elements in a node, is the start graph with the prototype of the node being the central vertex of the graph. Figure 13.2 shows, the splitting attribute, the prototype in each node of the tree, which one of both operators is applied in each node, over which examples is applied and the pattern which is return.

The rules that can be read by traversing the tree of patterns are:

- IF $46 \leq\ Temp.\ \leq 50$ THEN $CLASS$ 1.
- IF $51 \leq\ Temp.\ \leq 55$ AND $Molec. \in Set(Vc^5V^7)$ THEN $CLASS$ 2.
- IF $51 \leq\ Temp.\ \leq 55$ AND $Molec. \in Set(tc^{28}V^3bV^{21})$ THEN $CLASS$ 2.
- IF $46 \leq\ Temp.\ \leq 50$ AND $Molec. \in Set(Vc^7V^7)$ THEN $CLASS$ 2.
- IF $46 \leq\ Temp.\ \leq 50$ AND $Molec. \in Set(Vc^{25}V^{10})$ THEN $CLASS$ 3.

With this invented example we show how our approach can deal with very different data types in a homogeneous way, learning a decision tree based on the defined distances and how a post-processing can produce attribute patterns which provide an explanation of the classification. Since

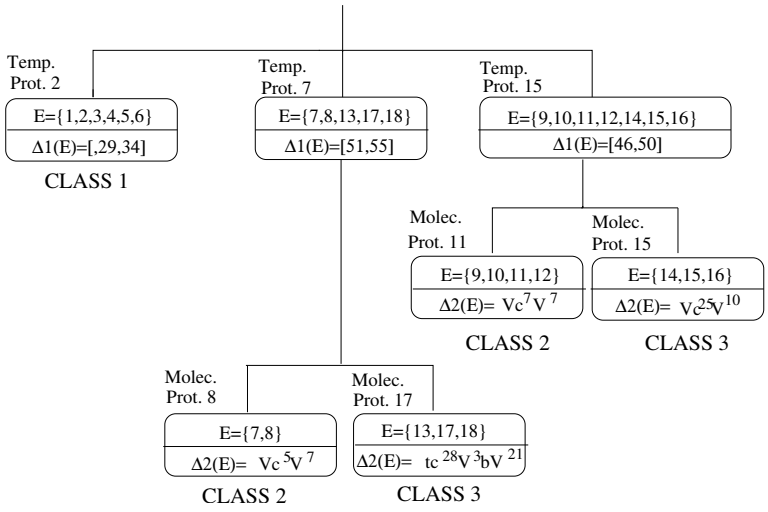


Figure 13.2: Symbolic decision tree obtained by applying *dbg* operators in post-processing.

this latter possibility has not been developed yet, in the next section, we present how our distance-based method behaves (in terms of accuracy) with real problems.

## 13.3 Experimental Evaluation

The procedure and functions above are implemented in WEKA. Running the current DBDT system requires two stages. The first one deals with preprocessing the data set file in order to obtain the distance matrix associated to each attribute (this task is done by means of *python* scripts specially written for each dataset). In the second step, the DBDT algorithm is executed taking both the dataset and the distance matrices file as input parameters (see Figure 13.3).

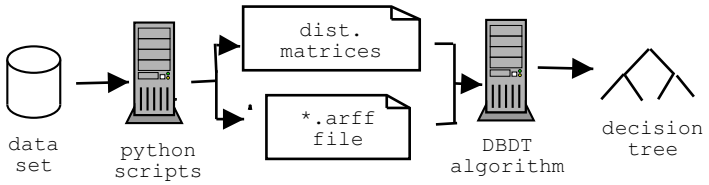


Figure 13.3: The current architecture of the DBDT system.

The aim of this section is to illustrate that the algorithm presented here can be used as a general purpose machine learning algorithm for both non-structured and structured data problems. All the results have been obtained using 10x10 fold cross validation.

With regard to the parameters of the DBDT, one prototype per class is computed and the

information gain heuristic is employed as splitting criterion.

We have utilised several distance and pseudo-distance functions. As for nominal and numerical data, we have used the discrete and absolute value difference, respectively. As for structured data, we have used the Hausdorff distance for sets (see [138]); a family of syntax-driven kernel-based pseudodistances [65] (*sdk* in short) has been employed for tuples, sets, multisets and trees; we have also used a kernel-based pseudodistance for a set-based representation of MI problems (see [64]); finally, we have used a kernel-based pseudo distance for text sequences (see [103]).

### 13.3.1 Non-Structured Data Problems

The first evaluation has been performed over twenty three attribute-value data sets extracted from the UCI repository, with all the attributes being nominal or numerical data types. The results have been compared to $J48$ with its default parameters in WEKA. Pruning is disabled for $J48$ and DBDT but a minimum of two examples per node are forced in each node (see Table 13.2).

| Data Set | $J48$ | DBDT |
|---|---|---|
| Balance Scale | 78.4 | 76.0 |
| B. cancer wdbc | 93.4 | 92.1 |
| B. cancer wisc | 94.3 | 94.3 |
| B. cancer wpbc | 72.1 | 69.5 |
| Cmc | 49.5 | 46.6 |
| Dermatology | 93.4 | 92.6 |
| Diabetes | 73.6 | 68.6 |
| Ecoli | 81.5 | 76.2 |
| Haber. Breast | 70.0 | 66.2 |
| Heart dis. | 51.2 | 48.1 |
| House votes | 94.5 | 93.9 |
| Ionosphere | 89.8 | 88.7 |
| Iris | 94.7 | 93.8 |
| Letter | 87.96 | 86.3 |
| Liver | 65.0 | 61.1 |
| Monks 1 | 95.1 | 96.1 |
| Monks 2 | 62.0 | 96.5 |
| Monks 3 | 98.5 | 97.6 |
| Tic Tac Toe | 78.9 | 74.8 |
| Pima | 73.7 | 68.6 |
| Post operative | 58.8 | 61.7 |
| Wave form | 76.39 | 74.3 |
| Sonar | 73.16 | 75.9 |
| Mean | 75.3 | 75.1 |

Table 13.2: Classification accuracy (%) on UCI data sets.

As we can observe, DBDT performs competitively. However, if we leave aside the data sets with only nominal attributes (monks 1, monks 2, monks 3 and tic tac toe), the accuracy decreases about 1.8 points. The explanation could be that $J4.8$ implements sophisticated and well-studied techniques for numerical data in front of the general treatment carried out by DBDT (the boundary point is just the median between the two prototypes). More elaborated (density or distribution based) ways of computing the boundary point could be developed once the prototypes are determined. Nevertheless the results are reasonably good for a decision tree learner that treats nominal and numerical attributes in the same way.

| | lumo and logp (R1) | Indicators (R2) | Atoms (R3) |
|---|---|---|---|
| *J*48 | 79.9 | 89.2 | 88.5 |
| DBDT | 77.1 | 84.2 | 84.9 |

Table 13.3: Accuracy (%) performed on non-structured version of Mutagenesis.

### 13.3.2 Structured Data Problems

We evaluate the performance of our approach over four real-world structured datasets. The first two datasets contain biomedical information whereas the remaining ones refer to WWW data. We also include performance achieved by the best-known learning systems over some of these domains.

#### The Mutagenesis data set

This problem aims at distinguishing between mutagenic and non-mutagenic compounds [156]. We have used the "regression friendly" version (188 compounds). Several experiments have been done in order to investigate how the data representation and the distances employed affect the performance of the learner. Concretely, the experiments can be divided into three groups. The first group of experiments (see Table 13.3) are focused on studying the performance achieved by the algorithm when a non-structured representation (propositionalisation) is used. In the second and third group of experiments (Tables 13.4 and 13.5 and Table 13.6, respectively), we begin with an initial structured representation, and then, non-structured information is incorporated.

In the first group of experiments, the first experiment (denoted by R1) treats only with the numerical attributes *lumo* and *logp* which represent certain chemical properties of the molecule. In the second experiment (denoted by R2), the Boolean indicators *ind1* and *inda*, which represent certain molecular substructures in a molecule (e.g. *ind1* is set 1 for those molecules having three or more fused rings), are added. Finally, in the third experiment (denoted by R3) the atomic composition of the molecule is captured in a naive way by incorporating in $R2$ as numerical attributes as the number of different atoms existing in the problem. Thus, each one of these new attributes represents the number of occurrences of a symbol of an atom in a molecule.

In the second group of experiments, we start with representing molecules by means of one structured attribute (R4). This attribute is a multiset of 3-tuples, where every tuple contains information referring to atoms in a molecule. More concretely, the first component of the tuple refers to the symbol of the atom (oxygen, carbon, etc.), the second component refers to the type of element (numerical) and the third component refers to the electrical charge (numerical). For instance, the first molecule in the data set is represented as:

$$\begin{aligned}\text{molecule } 1 \quad = \quad & \{(9;['c', 22.0, -0.117]), (9;['h', 3.0, 0.142]), \ldots, \\ & (2;['o', 40.0, -0.38])\}\end{aligned}$$

which means that this molecule has a group of nine carbon atoms of type 22.0 with an electric charge of 0.117, a group of nine hydrogen atoms of type 3.0 with an electric charge of 0.142, etc.

In order to use the Hausdorff distance and the pseudodistance (*sdk*) over this representation, the multiset of 3-tuples is essentially treated as a set of 4-tuples whose first component just indicates the number of occurrences of a 3-tuple in the multiset. For instance, given the 4-tuple $(a, b, c, d)$,

| | Multiset of 3-tuples (R4) | + lumo and logp (R5) | + Indicators (R6) |
|---|---|---|---|
| (*sdk*) pseudo-dist. | 82.6 | 80.9 | 83.9 |
| Hausdorff dist | 80.5 | 79.5 | 84.3 |

Table 13.4: Accuracy performed by DBDT algorithm using two different (pseudo-) distances for the structured attribute.

$a$ indicates the number of times the 3−tuple $(b, c, d)$ occurs in the multi-set. Recall that $b$ is the symbol of the atom, $c$ is the type of atom and $d$ is the electric charge of the atom.

The distance over 4-tuples that we need to set the Hausdorff distance for sets is:

$$d((a,b,c,d),(a',b',c',d')) = \frac{|a-a'|}{1+|a-a'|} + \delta(b,b') + \frac{|c-c'|}{1+|c-c'|} + \frac{|d-d'|}{1+|d-d'|}$$

The kernel behind the pseudodistance is directly obtained by applying a syntax-driven scheme for kernel definition over structured data introduced in [65]. Therefore, given two sets of 4-tuples $S_1$ and $S_2$,

$$k(S_1,S_2) = \sum_{\forall s_1 \in S_1, \forall s_2 \in S_2} a \cdot a' \cdot k'((b,c,d),(b',c',d'))$$

where $s_1 = (a,b,c,d)$, $s_2 = (a',b',c',d')$ and $k'$ is the kernel for 3-tuples defined as

$$k'((b,c,d),(b',c',d')) = \delta(b,b') + (c \cdot c') + (d \cdot d')$$

Next two experiments are done by progressively adding non-structured attributes. First the numerical attributes *lumo* and *logp* (R5), and finally, the Boolean indicators (R6).

Next we repeat the process but using this time a more complex structured attribute. In this new setting (Table 13.5) we capture not only the physical properties concerning each atom but also the type of bounds each atom is linked to. Thus, the multiset of 3-tuples from the previous experiment becomes here a multiset of 5-tuples where the last two components in the extended tuple are, in turn, multisets of outgoing and ingoing bounds. For instance, the first molecule is represented as:

$$\begin{aligned} \text{molecule 1} \quad = \quad & \{(9;['c',22.0,-0.117,\{(1;[1]),(1;[7])\},\{(1;[7])\}]), \\ & (9;['h',3.0,0.142,\{\},\{(1;[1])\}]),\ldots, \\ & (2;['o',40.0,-0.38,\{\},\{(1;[2])\}])\} \end{aligned}$$

which means that the first molecule contains a group of nine carbon atoms with identical physical properties and identical outgoing and ingoing bounds. In this case, these carbon atoms have two outgoing bounds of type 1 and 7 and one ingoing bound of type 7.

The kernel for multisets is the same that the one we have used above. Nonetheless, we have to extend the kernel $k'$ to deal with 5-tuples. Again, according to [65], given two 5-tuples $s_1 = (b,c,d,e,f)$ and $s_2 = (b',c',d',e',f')$

$$k'(s_1,s_2) = \delta(b,b') + (c \cdot c') + (d \cdot d') + k''(e,e') + k''(f,f')$$

where $k''$ is the kernel for the multisets of outgoing and ingoing bounds.

| | Multiset of 5-tuples | + lumo and logp | + Indicators |
|---|---|---|---|
| ($sdk$) pseudo-dist. | 83.1 | 80.6 | 84.1 |

Table 13.5: Accuracy performed by DBDT algorithm.

In the last group of experiments (Table 13.6), along with the multiset of 3-tuples, a new structured attribute is added. This attribute is a multiset of 1-depth trees. For every tree, its root corresponds to a chemical symbol of an atom in a molecule and its descendants are the atoms in this molecule which are directly connected to atom at the root of the tree. The kernel for trees is also obtained by applying the syntax-driven schema defined in [65]. To this end, we set the constructor *Tree root List Tree*. For instance, *Tree a [Tree b [], Tree c [], Tree d []]* represents a tree whose root is the symbol $a$ while the symbols $b$, $c$ and $d$ are the first-level nodes.

| | Multiset of 3-tuples and multiset of trees | + lumo and logp | + Indicators |
|---|---|---|---|
| ($sdk$) pseudo-dist. | 84.6 | 81.0 | 84.1 |

Table 13.6: Accuracy performed by DBDT algorithm.

Some interesting observations can be extracted from the last experiments. When the numerical attributes are added, in general, the algorithm performs worse. But the accuracy ratio is again improved when the indicators are used. This is so because the indicators encode information about the number of rings in a molecule. Thus, the indicators complement the structural information about the molecule contained in the structured attribute. However, we can notice that this improvement is not attained when more complex structured attributes are used. In fact, $J48$ seems to be a really good option for mutagenesis. DBDT performs similarly when a pure non-structured version and a pure structured version of the data set is given. Consequently, DBDT can compute feasible classifiers without using relevant information, such as the indicators attributes, provided by an expert.

In order to extract more reliable conclusions about the performance achieved by DBDT, some other known results obtained by ILP algorithms are summarised in Table 13.7.

| Alg. | $B0$ | $B1$ | $B2$ | $B3$ | $B4$ |
|---|---|---|---|---|---|
| Progol | 79.0 | 86.0 | 86.0 | 88.0 | 89.0 |
| Progol | 76.0 | 81.0 | 83.0 | 88.0 | ?? |
| FOIL | 61.0 | 61.0 | 83.0 | 82.0 | ?? |
| Tilde | 75.0 | 79.0 | 85.0 | 86.0 | ?? |
| MRDTL | 67.0 | 87.0 | 88.0 | ?? | ?? |

Table 13.7: Accuracy (%) performed by ILP systems on friendly mutagenesis. The *??* stands for an unknown result. The table has two entries for the PROGOL system because different results can be found in the literature.

The accuracy achieved by the different ILP algorithms is clearly improved as the background knowledge is populated with more useful predicates. In our case, no background knowledge is used for the best results in Table 13.6.

### The Musk data set

As mentioned in Chapter 3, this dataset is one of the most representative examples of the so-called MI problem. Several techniques have been proposed to cope with the multi-instance problem. On the one hand, we find algorithms specially designed or adapted for this purpose like APR [25], and some upgraded techniques such as nearest neighbours (Bayesian-Knn and Citation-KNN [166]), neural nets (MI-NN [142]), decision tree learners (RELIC), support vector machine (MI-SVM and mi-svm [5]). On the other hand, we distinguish classical approaches, which ignore the MI setting during training (NeuralNets, C4.5) but not during testing, or general-purpose approaches able to face the MI problem without changing its learning scheme (IBL[138],KeS[65]). The DBDT algorithm is situated in this latter group, as a general-purpose system.

In our setting, a molecule will be represented by only one structured attribute, that is, a set of tuples. We try both the Hausdorff distance and a pseudo-distance function from a MI-specific kernel proposed in [64]. Best known results are reported in Table 13.8.

| Id. | Algorithm | Accu.(%) 10CV |
|---|---|---|
| 1 | Iter. APR | 92.4 |
| 2 | GFS positive APR | 83.7 |
| 3 | Citat.-kNN | 92.4 |
| 4 | MI-SVM | 89.0 |
| 5 | mi-SVM | 84.0 |
| 6 | MI-nn | 88 |
| 7 | RELIC | 84.3 |
| 8 | IBL-Hausdorff | 82.0 |
| 9 | DBDT-Hausdorff | 81.6 |
| 10 | KeS | 81.0 |
| 11 | DBDT-pseudo-distance | 75.7 |
| 12 | nn | 75.0 |
| 13 | C4.5 | 68.5 |

Table 13.8: Classification accuracy (%) on musk1 data.

This table is organised as follows. The first seven rows contains the accuracy of some of the specific-MI tools reported in the literature. The next four values are performed by general-purpose algorithms. We must stand out that the kernel method ($KeS$) achieves an optimal result (86.4%) when a parameter is fixed to an optimal value. The last two results correspond to algorithms ignoring the MI setting in training. The DBDT algorithm performs 81.6% and 75.7% for the Hausdorff distance and the pseudo-distance functions respectively. Thus, the experiments show that our approach performs competitively to general purpose methods.

### WWW data

In this section, we perform experiments with two datasets containing web data. Initially, each *html* document is described by a collection of unique words (the so-called summary) which are drawn from the title and the body sections. Of course, *html* tags or script instructions are not included in the summary. Additionally, prepositions and adverbs are discarded as well. Each word in the summary is selected according to a significance measure which quantifies how important it is for classification. Several significance measures can be found in the literature [172]. All the experiments have been performed using an entropy-based measure.

Two datatypes (a set of words and a sequence of words) are used for summary representation. With regard to sets, the Hausdorff distance is applied with the distance for words being the edit distance. Regarding sequences, we use a pseudodistance based on a kernel introduced in [103] and successfully applied in text classification (see also Subsection 3.4.2 from Chapter 3 for a brief description).

The general setting for all experiments below is as follows. We start with summaries containing the most 50 significant words, adding 25 words in each setting.

Two tasks are addressed:

1. Classifying web sites by topic: at this point, we tackle the classification of *html* documents. For this purpose, we have collected a total of 83 *html* documents downloaded directly from the Internet. The documents are grouped into two different topics: mathematics (biographies, technical pages, personal web sites, lectures, etc.) and sports (biographies, news, events, championships, etc.). Thus, the goal is to learn a classifier which tells about the topic of a new unclassified document. Additionally, we want to study the influence of using more information rather than the summary. Taking the sequence representation for summaries, we add two extra attributes: a list of keywords (using also the same pseudo-distance that in the summary) and the number of pictures appearing in the web page. The obtained results (accuracy %) are reported below.

| Number of words | Summary (set) | Summary (seq.) | Summary (seq.) + Key words (seq.) + Number of pictures |
|---|---|---|---|
| 50 | 93.6 | 100.0 | 100.0 |
| 75 | 91.5 | 98.1 | 99.7 |
| 100 | 91.4 | 95.9 | 100.0 |
| 125 | 94.8 | 98.4 | 100.0 |
| 150 | 92.5 | 97.6 | 91.4 |

Table 13.9: Accuracy (%) achieved varying the number of words included in the summary, the summary representation and the number of attributes.

From the experimental results, we can see that more accurate results are obtained when sequences (i.e. the specific-purpose pseudo-distance) are employed. In connection with this, we cannot miss the fact that Hausdorff distance is quite sensitive to outliers and these might affect the performance of the method when sets are employed. Secondly, as we can appreciate, except for summaries of 150 words length, the accuracy is slightly increased by incorporating these two extra attributes.

2. Learning user profiles: The *Syskill & Webert* dataset [128] is a repository of web documents organised into several topics: clinical information (*Biomedical*), music events (*Bands*), biochemistry (*Proteins*), etc. Each document is ranked, according to the preferences of the user, in "hot", "medium" or "cold". A document labelled with "hot" stands for a high interesting web page, being uninteresting when it is labelled with "cold". Therefore, learning a user profile leads to learning a classifier which labels unseen pages with the proper tag.

   In order to compare the obtained results to those reported in [128], only "hot" and "cold" documents will be taken into account and these will be represented by only their respective

summary. All the experimental process is focused on *Bands* and *Biomedical* repositories. In these experiments, the summary is modelled by a sequence of words using the pseudo-distance derived from the subsequence kernel (see Table below).

| Num.words | *Bands* Acc. % | *Biomedical* Acc. % |
|---|---|---|
| 50 | 74.5 | 71.5 |
| 75 | 79.7 | 81.3 |
| 100 | 77.9 | 83.0 |
| **125** | **82.0** | **84.0** |
| 150 | 81.7 | 79.6 |
| 200 | 82.1 | 82.0 |

Table 13.10: Accuracy achieved on *Bands* and *Biomedical* data sets varying the number or words included in the summary.

As [128] points out and as we can also observe from the results above, it seems that the optimal number of words is around 125. In contrast, fixing a low number or a high number of words can motivate that some really meaningful words are not included in the summary, or that noisy words take part of it. Keeping on the same work, a collection of learning techniques (ID3, Bayesian methods, neural networks, among others) to address the proposed problem are compared. Although the evaluation method used in the so-mentioned work is slightly different to cross validation, the best accurate results on both data sets is performed by a Bayesian classifier (78.2% and 74.6% for *Biomedical* and *Bands* respectively). In addition, the authors take a further step by trying to improve the performance by incorporating background knowledge. It consists of a set of key words, in this case, explicitly introduced by the user. Using this strategy, the Bayesian classifier performs $82-83\%$ and $78-79\%$ for *Biomedical* and for *Bands*. Furthermore, according to the outcomes reached by ID3, (remember that DBDT can be viewed as an extension of it) they are still lower, obtaining $70.2-75.9\%$ and $68.6-70.7\%$ for *Biomedical* and for *Bands*.

## 13.4 Discussion

In this chapter we have presented an algorithm that combines decision tree learning and distances, resulting in an algorithm which can deal with any data type as long as a distance function has been defined on it. This is so because, unlike classical decision trees where conditions are expressed by means of relational tests, here, the conditions are expressed in terms of distances to prototypes.

The advantage is that all the conditions (independently of the data type) are performed equally, which means that all the splits are handled equally. Note that this not happens in many decision-tree learning algorithms, such as C4.5, which treats nominal and numerical data very differently. However, the downside is that this splitting criterion affects the comprehensibility of the model obtained. This limitation could be overcome, as we have suggested by means of a toy example, by applying proper distance-based *dbg* operators and, very specially, *mdbg* operators, over each one of the nodes in the tree.

The second part of this chapter includes an experimental evaluation of the method. More concretely, we focus on studying the performance of our method, in terms of the accuracy, over propositional problems extracted from the UCI repository, and with datasets that contain structured

attributes. Although in both cases there are other approaches that obtain better results, our system shows that can be employed in all these scenarios with a satisfactory performance. Probably, the inclusion of well-known refinements that improve decision trees (pruning, smoothing, etc.) could significantly increase the performance of our system.

As future work we are studying how to convert the distance-based splits for structured data types into pattern-based splits, in order to make them more comprehensible using the *dbg* operators. Note that, if we can do this, we would obtain a global symbolic model from data and a collection of distances.

# Chapter 14

# Conclusions and Future Work

This final chapter outlines the obtained results and points out the main open issues that follow from this research.

## 14.1 Conclusions

There is little doubt about the importance that information has in modern times. The daily routine in companies, governments and research centres consists of analysing huge amounts of data in order to obtain usable knowledge. For instance, research centres go through experimental data to corroborate theories or enlighten new lines of work; companies look into their databases to find out sales patterns or to come up with cost-cutting strategies while maintaining production levels; governments perform their data analysis on security, fraud detection, economy prediction, discovery of new social trends, etc.

Machine learning is a discipline that is focused on developing computational methods that obtain patterns hidden in data repositories. The methods to be employed depend on the nature of the problem at hand. Thus, two important aspects that must be considered in order to choose between one method or another concern data and hypothesis representations. Usually, different representations require different learning algorithms, and a multiple variety of methods are needed since both data and hypotheses can be represented in various ways (e.g. data descriptions can be provided as tuples, lists, sets, graphs, or nested forms of them). However, most algorithms that have been developed for different data representations have similar core ideas. This makes the possibility of designing learning methods to be independent from data representation a relevant contribution.

Following this idea, we find that the so-called distance-based methods (or more generally, similarity-based methods) can systematically be adapted for different data representations. These methods rely on the assumption that near (similar) cases require similar solutions. The notion of proximity between the examples is formalised by a distance. Thus, these methods can be used for any data representation as long as a distance (or a similarity) function has been previously defined. Indeed, these have successfully been applied over propositional and structural real-world domains thanks to the fact that there is a wide variety of distance functions for the most common data types. Unfortunately, a serious drawback is that these methods do not capture possible patterns in data. Or to be more precise, the "patterns" (to say something) are expressed in terms of the distance-function. This

makes it difficult for the user to obtain any relevant knowledge from these methods. For instance, the algorithms classify or group examples according to criteria such as "example $e$ belongs to class $c$ because its nearest prototype is in class $c$".

In contrast, we have symbolic learners (which have also been successfully tested over propositional and structured domains), which work differently from the family of methods described above. They focus on expressing patterns into a language which is easily comprehensible to the user. Unfortunately, the problem is that symbolic methods are designed from the concept of generalisation (the process of obtaining a pattern), which is understood or used differently depending on the data representation. This limits symbolic learners with regard to the data representations they can handle.

Consequently, if the problem to be solved requires a comprehensible solution, we must find a symbolic learner that is capable of dealing with the data representation we are given. Of course, in case a method like this is not available, there is no other option but to upgrade an existing method or design a new one. For this reason, it seems useful to investigate how to obtain learning systems that could deal with or could be systematically adapted to any data representation (as distance-based methods do) and would also be able to return a symbolic model (as symbolic-based learners do). This is precisely the goal of this thesis.

We have explored the idea of transforming the output of distance-based methods into symbolic descriptions. The most intuitive way to do this is to plug a "generalisation procedure" into the distance-based method. However, this must be done in such a way that this "procedure" can systematically be adapted to different distance-based methods and to different data representations so that it returns symbolic descriptions which are consistent with the output of the distance-based method. This latter issue is crucial since inconsistencies may arise when the distance and the generalisation are defined separately. Note that both concepts (distances and generalisations) are in essence two ways to compare examples. Also note that inconsistencies will show up if we introduce a generalisation process that compares examples in a completely different way than the distance does.

### Contributions

The main contributions of this thesis are:

#### i) A formal approach for the integration of two complementary learning paradigms

The main contribution of this thesis is the definition of a formal framework that allows us to design algorithms which can transform the output of distance-based methods into symbolic-based patterns while preserving the consistency between the two. This framework relies on a thorough analysis of the fundamental notions of distance, pattern and generalisation, which results in the core definitions of a binary and $n$-ary distance-based generalisation operator and a novel definition of minimal generalisation. The flexibility of the framework comes from the fact that these previous definitions are valid independently of the data representation and the distance employed, which means that this framework can be applied to obtain symbolic patterns for any combination of data representation and distance.

We first provide a definition of binary distance-based generalisation as a generalisation of two elements that is consistent with the underlying distance. The consistency is understood in terms of the reachability of these two elements by means of straight paths that are included in the generalisation.

The concept of binary distance-based operator, however, is just an initial step towards achieving

our goal. Obviously, distance-based methods (like any other group of learning methods) rarely work with two examples only. This means that we need to extend this concept if we want to deal with an arbitrary number of examples. Hence, another important issue that has been addressed in this thesis consists in defining the concept of $n$-ary distance-based generalisation operators. This extension is not immediate since the notion of binary distance-based operators strongly relies on the conception of the distance as a binary function. Thus, the extended definition must be adapted to this circumstance, and an $n$-ary distance-based generalisation is the one that locally (for some pairs of examples) behaves as a binary distance-based generalisation. The pairs of examples for which the pattern is distance-based are given by the context. For instance, if elements are assigned to a cluster by a direct comparison with a prototype, it seems reasonable that a symbolic pattern describing this cluster is distance-based w.r.t. all the pairs of the form (element,prototype), but not necessarily for others. This is what the notion of nerve formalises.

The concept of least or minimal generalisation is also crucial when learning models since this keeps us from overgeneralising data and also helps to control overfitting. This concept has been extensively studied in the field of ILP. In this thesis, we introduce a more flexible notion of minimal generalisation than the one used in ILP since the definition we propose here is partially independent of the data representation. Unlike ILP where the notion of generalisation is exclusively of a semantic nature ($\theta$-subsumption), in our framework, minimality is expressed by means of a cost function (with a syntactic part and a semantic distance-based part). The syntactic part (if not set to a constant value) is a useful tool to help in the ordering of the search space (in the tradition of optimal search and learnability in complex pattern spaces derived from the MML/MDL principle and Kolmogorov complexity, Muggleton's U-learnability, etc.). Its formulation depends on the pattern language used. However, the semantic part is exclusively defined in terms of the underlying distance and measures how much a pattern fits a set of data. By doing this, the notion of fitness (or generality) is independent of the data representation and is also consistent with the distance. This is important because a notion of generality directly stated over the pattern language (as ILP does) might be different to the notion of generality implicit in the distance (this being understood as the sparseness of the "area" formed by the elements in the metric space that are covered by a pattern). For instance, given two equally general patterns (equally general according to the idea of generality stated in the pattern language), one of these patterns covers a "greater area" (further elements) than the other, resulting in a clear contradiction.

Thus, the major benefit obtained by integrating these two components of a pattern in the same schema is that this formalism allows us to select patterns that achieve a trade-off between complexity and generalisation. Patterns like these are useful when generalising a set of elements because the generic definition of $n$-ary distance-based operator that we handle tends to overfit the data when the pattern language is too expressive. Thus, to avoid this problem, we set an appropriate cost function and search for the distance-based pattern which minimises this cost function (minimal distance-based generalisation operators).

The notion of cost function, which has been introduced in this thesis, can be seen as the first formulation of the MDL/MML criterion in terms of distances. It completes the connection between the concepts of distance, pattern and generalisation that we have attempted to achive.

**ii) (Minimal) distance-based generalisation operators for common datatypes**

The framework has been instantiated for different data representations embedded in metric spaces.

This provides theoretical results for computing the distance-based generalisation operators. The combinations of distances and data representations which have been studied are: absolute difference and real numbers, discrete distances and categorical data, symmetric difference and sets, edit distance and lists, edit distance and graphs, and J. Ramon's distances and first-order objects (atoms and clauses).

An immediate consequence from this work is that, given a distance, not every pattern represents a generalisation that is compatible with the distance. A representative case of this comes up when studying the above-mentioned combination of distances and first-order objects. As for atoms, we conclude that classification/clustering using the distance function in [139] can be accompanied by a model obtained from Plotkin's $lgg$ operator. However, this result does not hold for other metric spaces over atoms. For instance, if we consider the distance defined in [125], the $lgg$ is not distance-based. Likewise, for clauses, the $lgg$ is not suitable when the distance [137] is involved, which is the only distance known for this sort of data representation. Another curious case occurs when working with sequences or graphs. In this case the concept of the longest common subsequence that is widely-used in bioinformatics as a generalisation for sequences is not distance-based for a usual setting of the edit distance. Additionally the concept of common subgraph, which is a concept used by graph mining techniques, is not a distance-based generalisation if the distance proposed by H. Bunke in [19] is utilised. In fact, due to the importance of this concept, we redefine Bunke's distance so that common subgraphs become distance-based patterns.

As said, this thesis not only undertakes the question related to what generalisations work well with a given distance, but also deals with the question of the underfitting/overffiting of the generalisations obtained for the above-mentioned distances and data representations. In this work, we also explore the relationship between the notion of classical minimality proposed for the different data/pattern representations and the notion of minimality that we state here. In some contexts, both notions match. For instance, for simple data representations such as numerical data, the lowest-length interval covering a set of numbers is minimal w.r.t. a well-defined family of cost functions. This intuitive result can be extended to other more interesting scenarios such as first-order atoms where the $lgg$ operator is also minimal w.r.t. our definition. Of course, these results depend on the cost function employed. For instance, for other functions, we show that the $lgg$ is not necessarily minimal. This suggests that our framework could be used as a well-founded mechanism to define novel generalisation operators for atoms. These alternative operators would allow us to redesign existing ILP methods or to derive new ones.

For some other contexts, we have shown that the calculus of the minimal generalisation can be computationally expensive (specially for highly expressive pattern languages). It would be more convenient to approximate this by means of a greedy search guided by the cost function itself.

### iii) Composability of the distance-based generalisation operators

Another point that has been dealt with in this work concerns the "composability" of distance-based operators to be used over nested data types. Here, we have analysed the most intuitive variant of composability, that is, tuple-composability. This is also the most natural extension of propositional learning towards dealing with structured data since this language allows more complex constructions such as tuples of lists, tuples of sets, etc. Tuples of structured data can easily be handled by distance-based learners since a distance for tuples can be obtained from the distance defined over each one of the components of the tuple. In this thesis, we show that if the distance employed for

the space of tuples is the Manhattan distance, a nice result is obtained since the Cartesian product of the distance-based generalisation operators is a distance-based generalisation operator in the space of tuples. Additionally, if the operators employed are minimal then the operator for tuples is also minimal. This result allows us to extract symbolic patterns from quite complex structures in an easy way since a distance-based generalisation operator can be defined for the different sorts of data involved in the definition of the space of tuples. However, we have also shown that the result concerning the composability of the distance-based operators does not hold when other distances for tuples are employed (e.g. the Box distance).

**iv) Transforming distance-based learners into symbolic learners**

An immediate application of (minimal) $n$-ary operators is the possibility of redefining any distance-based clustering algorithm into a conceptual-clustering algorithm. This can be done easily since we only have to apply the generalisation operator to every cluster of elements. This provides a symbolic description of each cluster that is consistent with the underlying distance. Furthermore, since this can be done for any data representation (propositional or structured) that is endowed with a distance function, what we actually have is a conceptual clustering algorithm that can systematically be adapted to any data representation because only the generalisation operator involved must be changed. Therefore, unlike traditional conceptual clustering algorithms that have mainly been conceived for propositional representations, we can derive conceptual clustering algorithms which can be used over different data representations. We illustrate how to do this by transforming a hierarchical clustering algorithm into a conceptual-clustering algorithm. This is done by applying several $n$-ary distance-based generalisation operators, each of which is defined over a different representation of the problem.

In a similar way, classification constitutes another interesting scenario where distance-based generalisation operators could be applied. The problem this time is that the most commonly used distance-based method for classification, $k$-NN, bases its predictions on a local analysis (a comparison of the unseen example with its $k$ nearest examples) which would lead to multiple symbolic models if generalisations operators were applied. For this reason, we have developed another distance-based method for classification that provides a better input for the generalisation operators. This method consists of a decision tree learner whose splits are expressed in terms of distances to prototypes. Thus, the output is in fact a tree where each node is represented by a prototype that attracts its nearest elements. The prototypes can be changed by symbolic patterns by applying generalisation operators to every node. The learner has been tested over propositional and structured domains and the transformation process (to convert a tree of prototypes into a tree of patterns) has been illustrated by means of a toy example.

**Implications**

The contributions and results obtained may have many implications and connections in the field of machine learning:

- The most direct consequence of this work is the development of systematically adaptable symbolic learners for data representations that are endowed with a distance function.
- It provides new tools for the problem of learning from complex data, which is one of the main

challenges in machine learning (e.g. distance-based and kernel-based methods for structured data [65]). At a general level, our framework can be seen as attempt to obtain comprehensible descriptions that capture or mimic the semantics hidden in "distance-based models" that are learnt from both propositional and complex data. This viewpoint is to some extent related to mimicking techniques that are used in propositional learning to obtain symbolic descriptions from black-box models that are learnt from propositional data [41, 14].

- Learning from complex data while preserving comprehensibility is a challenge that has mainly been addressed in the area of ILP [114]. This work allows the ILP to be extended to many other data representation paradigms and pattern languages.

- Our framework can improve existing symbolic learners if we assume that the data these learners use is embedded in a metric space. That is to say, we can redefine ILP methods, grammar inference algorithms, etc. by plugging distance-based operators and/or cost functions into them. For instance, in ILP, when the $lgg$ operator is applied over clauses, it yields a new clause with more literals than the original ones. This creates the need for special methods to be run in postprocessing in order to remove redundant and little significant literals. In contrast, the generalisation operator for clauses that we propose in this work tends to yield a more general clause with fewer literals than the one returned by the original $lgg$. Although this must still be refined, this is a promising result which could be considered for (re)designing (existing) new ILP learners based on $lgg$ for clauses.

- In this framework, distance and generalisation are presented as dual concepts. Here, we have mainly investigated this relationship in the direction of going from distances to generalisation operators. However, we could also explore the opposite direction and determine which distance could be defined for a generalisation operator to be distance-based. This is suggested by J. Ramon's distance definition for atoms, and this is precisely what we have done for graphs by modifying an existing distance so that distance-based generalisation operators could be defined and computed easier.

## 14.2 Future Work

This work has established a foundation which makes it possible to integrate learning techniques based on such different concepts as distance and generalisation. The main contributions and implications have been enumerated in the previous section. However, the course of this research has shown up new related issues which have not yet been addressed and which can be considered in future work.

- **Handling normalised distances**: the definition we have presented in this thesis about what a distance-based generalisation could be has some limitations when distances are normalised. A quite common transformation for distance normalisation is $d' = \frac{d}{1+d}$ since it is known that $d'$ is, in turn, a well-defined distance. When $d'$ is used instead of $d$, it is likely that for every two given elements $x$ and $y$, there are no elements lying between them. In this case, every generalisation operator in $(X, d')$ would be distance-based. This is not totally suitable since the elements that are between $x$ and $y$ in $(X, d)$ will also lie near $x$ and $y$ (though not exactly in between) in $(X, d')$. Thus, if there are no elements between $x$ and $y$ it would, at least, be preferable to cover those elements that are fairly near to them. This suggests extending the

current framework to properly deal with normalised distances. One proposal could be to relax the metric condition referring to distance-based operators so that the elements in "between" are now those satisfying an expression like $d(x,z) + d(z,y) < k \cdot d(x,y)$ where $k \geq 1$ (or other similar mathematical expressions) and to introduce what could be called the $k$-generalisation operators. As 1-generalisations in $(X, d')$ would make little sense for normalised distances, this undoubtedly requires finding $k$-generalisation operators such that $k > 1$. To make this task easier, if the relation between $d$ and $d'$ is known, we could determine if 1-generalisations in $(X, d)$ are $k$-generalisations in $(X, d')$ with $k > 1$.

- **Defining weak distance-based generalisation operators**: one of the specific issues that must be addressed for any new distance-based generalisation operator is, logically, its efficiency. In some metric spaces, distance-based generalisation operators are computationally expensive. Representative cases of this are distance-based generalisation operators in the space of lists and in the space of clauses. In both, all the optimal matchings (between two lists or two clauses) should be found in order to compute distance-based generalisation operators. Given that this makes the process slower, it may be more interesting to achieve a trade-off between the amount of information captured by the operator and its efficiency. This is the idea behind what we call weak distance-based generalisation operators, which are essentially operators whose generalisations cover some elements in between (not necessarily all).

- **Defining generalisation operators for data embedded in pseudo-metric spaces**: as mentioned above, distances can be computationally expensive and sometimes there is no other option but to use relaxed forms of distances such as pseudo-distances and semi-distances. This is quite common in graph learning where the reported distances cannot be computed in polynomial time, and consequently, distance-based methods become inefficient when using them. However, kernels provide a rich repertory of pseudo-distances over graphs (and other structures) which can be computed in polynomial time. Thus, in order to take advantage of this and to extract symbolic models when using pseudo-distances, it is necessary to adapt our framework to deal with similarity functions of this type.

- **Handling nested data-types**: in this thesis, we have studied how to define distance-based operators from datatypes built from tuples. However, the possibilities when composing datatypes are enormous: lists of lists of symbols, sets of lists of symbols, lists of sets of symbols, etc. Obviously, the goal is to define distance-based operators in such complex spaces. As this task seems to be hard, it would be necessary to find out an algebraic way of nesting distance-based operators defined in basic spaces in order to obtain distance-based operators in the complex space.

- **Defining distance-based generalisation operators w.r.t. background knowledge**: there is no doubt about the importance of background knowledge in machine learning. In [109], a thorough study about the need of background knowledge in inductive learning is presented. Background knowledge is important because some problems could hardly be solved or their solution would be of low quality without it. For instance, in molecule clustering, some patterns might make sense from a semantic point of view (e.g. molecules having an aromatic ring) but not from others points of view (e.g. molecules that have only a part of a ring and a simple bound in common). In [129], Plotkin defines how to make use of background knowledge while generalising ($rlgg$ operator). However, distance-based generalisation operators disregard

background knowledge. Thus, we want to know how background knowledge can be plugged into distance-based generalisation operators.

- **Studying new ILP and grammar inference applications**: nice possibilities arise when using distance-based generalisation operators for first-order objects and lists. For first-order objects, we could study the derivation of some other generalisations for atoms and clauses by changing the pattern languages, the cost function and the distances. Furthermore, since we have derived a generalisation operator for clauses that is different from $lgg$, we could explore what possibilities this new operator brings along with a proper cost function in order to obtain new ILP algorithms. A similar thing happens when working on lists. We could explore how to use distance-based operators for lists together with cost functions to obtain new grammar inference algorithms.

- **Controlling overlapping in conceptual clustering**: the framework gives us support to obtain consistent conceptual descriptions from clusters of elements that are learnt via distance-based clustering algorithms. However, these descriptions are performed focusing on one cluster only while disregarding the rest. Although we have tried to obtain descriptions that fit the clusters as much as possible according to the cost function, we cannot ensure a priori whether these descriptions overlap or not, or at least, how much these descriptions overlap. An interesting proposal would be to integrate this new control degree in the cost function. For instance, we could add a new term of the form $c(E^-|p)$, measuring how far the elements that do not belong to the cluster ($E^-$) are from the border of the pattern $p$. This more powerful schema would also be very useful for the *dbdt* algorithm since we could obtain descriptions that better separate elements from different classes.

- **Defining supervised distance-based generalisation operators**: our framework is currently not designed to perform optimally in classification. Basically, this is so because the patterns computed by the distance-based generalisation operators fit the cluster of data, but we do not know whether or not, the patterns fit the border separating different groups of data because the class label information is disregarded. An interesting line of work would be to extend the current framework for the supervised case. If successful, a promising result would be a framework that could help us to develop bottom-up classifiers that are independent of the data representation. Note that this problem subsumes the previous one in that elements belonging to different clusters can be considered as elements from different classes. However, this problem is more ambitious since, in a classification problem, elements from one class could be distributed in different groups of data (a class could be represented by more than one cluster), class distributions could overlap, etc.

- **Improving the dbdt algorithm**: dbdt is different from all existing decision-tree learners in that the same splitting condition (a splitting condition based on the distance) is used for any datatype. Therefor, it can handle any sort of data that is endowed with a distance function. However, there are several aspects to be considered in order make dbdt competitive. Basically, all of them concern the performance and the quality of the learnt model. Although experiments have shown that *dbdt* performs adequately when compared with other methods, we also found that the decision tree learnt had a high amount of leaf nodes with few examples inside. This has two negative consequences. On the one hand, this shows that the learnt model tends to overfit the data, which might negatively affect its performance. On the other hand, multiple

leaf nodes also mean multiple rules, which makes the comprehensibility of the model worse. It is known that pruning techniques have been shown to be effective for problems of this kind. Then, it would be necessary to study whether or not standard pruning techniques are also effective in *dbdt* or if we have to develop new ways to solve this problem. Finally, *dbdt* does not return symbolic model. Although we have manually illustrated how distance-based generalisation operators could be used to transform the model, this characteristic has not been implemented yet. Thus, it would be interesting to extend *dbdt* functionality by adding this possibility and also studying the quality of the symbolic models obtained.

This work establishes a link between apparently unconnected notions and learning paradigms, providing new understanding of how to use machine learning techniques in an integrated way. Hence, this thesis is in line with some contributions and proposals whose goal is to endow this discipline with a more unified view.

To the best of our knowledge, this work constitutes the first attempt to formalise and understand the process of integrating two important learning paradigms that are as different from each other as are distance-based learning and symbolic learning. The results reported here give us insights about how to carry out this process and how to start to develop learners that inherit the best from both paradigms. However, this is not a closed work. In this thesis, we have just begun to bridge the gap between them.

# Bibliography


[1] R. Agrawal, H. Mannila, R. Srikant, H. Toivonen, and A.I. Verkamo. Fast discovery of association rules. In *Advances in Knowledge Discovery and Data Mining*, pages 307–328. American Association for Artificial Intelligence, 1996.

[2] D. W. Aha, D. Kibler, and M. K. Albert. Instance-based learning algorithms. *Machine Learning*, 6(1):37–66, 1991.

[3] E. Alpaydin. *Introduction to Machine Learning*. MIT Press, 2004.

[4] R.A. Amar, D.R. Dooly, S. A. Goldman, and Q. Zhang. Multiple-instance learning of real-valued data. In *Proc. of the 18th International Conference on Machine Learning (ICML'2001)*, pages 3–10. Morgan Kaufmann, 2001.

[5] S. Andrews, I. Tsochantaridis, and T. Hofmann. Support vector machines for multiple-instance learning. In *Proc. of 16th Int. Conf. on Advances in Neural Information Processing Systems (NIPS'2002)*, pages 561–568, 2002.

[6] A. Atramentov, H. Leiva, and V. Honavar. A multi-relational decision tree learning algorithm - implementation and experiments. In *Proc. of the 13th Int. Conference on Inductive Logic Programming (ILP'2003)*, Lecture Notes in Computer Science, pages 38–56. Springer, 2003.

[7] S. Bachl, F.J. Brandenburg, and D. Gmach. Computing and drawing isomorphic subgraphs. *Journal of Graph Algorithms and Applications*, 8(2):215–238, 2004.

[8] E. Van Baelen and L. De Raedt. Analysis and prediction of piano performances using inductive logic programming. In *Proc. of the 6th Int. Workshop on Inductive Logic Programming (ILP'1996)*, Lecture Notes in Computer Science, pages 55–71. Springer, 1996.

[9] R. Baeza-Yates. Introduction to data structures and algorithms related to information retrieval. In W.B. Frakes and R. Baeza-Yates, editors, *Information Retrieval: Data Structures and Algorithms*, pages 13–27. Prentice-Hall, 1992.

[10] P. Baldi and S. Brunak. *Bioinformatics: The Machine Learning Approach*. MIT Press, 2001.

[11] F. Bergadano and D. Gunetti. *Inductive Logic Programming*. MIT Press, 1995.

[12] P. Berkhin. Survey of clustering data mining techniques. Technical report, Accrue Software, San Jose, CA., 2002.

[13] G. Bisson. Conceptual clustering in a first-order logic representation. In *Proc. of the 10th European Conference on Artificial Intelligence (ECAI'1992)*, pages 1–10. John Wiley & Sons, Inc., 1992.

[14] R. Blanco-Vega, J. Hernández-Orallo, and M. J. Ramírez-Quintana. Analysing the trade-off between comprehensibility and accuracy in mimetic models. In *Proc. of the 7th Int. Conference on Discovery Science (DS'2004)*, pages 338–346. Springer, 2004.

[15] H. Blockeel. *Top-Down Induction of First-Order Logical Decision Trees.* PhD thesis, Katholieke Universiteit Leuven, Belgium, 1998.

[16] H. Blockeel, L. De Raedt, and J. Ramon. Top-down induction of clustering trees. In *Proc. of the 15th International Conference on Machine Learning (ICML'1998)*, pages 55–63. Morgan Kaufmann, 1998.

[17] A.F. Bowers, C. G. Giraud-Carrier, and J. W. Lloyd. Classification of individuals with complex structure. In *Proc. of the 17th International Conference on Machine Learning (ICML'2000)*, pages 81–88. Morgan Kaufmann, 2000.

[18] L. Breiman, J.H. Friedman, R.A. Olshen, and C.J. Stone. *Classification and Regression Trees.* Chapman and Hall, 1984.

[19] H. Bunke. On a relation between graph edit distance and maximum common subgraph. *Pattern Recognition Letters*, 18(8):689–694, 1997.

[20] R.M. Cameron-Jones and J.R. Ross Quinlan. Avoiding pitfalls when learning recursive theories. In *Proc. of the 13th International Joint Conference in Artifical Intelligence (IJCAI'1993)*, pages 1050–1057, 1993.

[21] P. Clark and R. Boswell. Rule induction with cn2: Some recent improvements. In *Proc. of the European Working Session on Learning on Machine Learning (EWSL'1991)*, pages 151–163. Springer-Verlag, 1991.

[22] T.M. Cover and P.E. Hart. Nearest neighbour pattern classification. *IEEE Transactions on Information Theory*, IT-13:21–27, 1967.

[23] C. de la Higuera. A bibliographical study of grammatical inference. *Pattern Recognition*, 38(9):1332–1348, 2005.

[24] R.C. de Vrijer, J.W. Klop, and M. Bezem. *Term rewriting systems.* Cambridge University Press, 2003.

[25] T. G. Dietterich, R. H. Lathrop, and T. Lozano-Perez. Solving the multiple instance problem with axis-parallel rectangles. *Journal of Artificial Intelligence*, 89(1-2):31–71, 1997.

[26] T.G. Dietterich. An experimental comparison of three methods for constructing ensembles of decision trees: Bagging, boosting and randomisation. *Machine Learning*, 40(2):130–157, 2000.

[27] P. Domingos. Unifying instance-based and rule-based induction. *Machine Learning*, 24(2):141–168, 1996.

[28] P. Domingos and M. Pazzani. On the optimality of the simple bayesian classifier under zero-one loss. *Machine Learning*, 29(2-3):103–130, 1997.

[29] K. Driessens and S. Dzeroski. Combining model-based and instance-based learning for first order regression. In *Proc. of the 22nd International Conference on Machine Learning (ICML'2005)*, pages 193–200. Morgan Kaufmann, 2005.

[30] R.O. Duda, P.E. Hart, and D.G. Stork. *Pattern Classification*. John Wiley and Sons, 2001.

[31] S. Dzeroski and N. Lavrac, editors. Springer-Verlag.

[32] S. DZeroski and N. Lavrac. *Relational data mining*. Springer-Verlag, 2001.

[33] G. A. Edgar. *Measure, Topology and Fractal Geometry*. Springer-Verlag, 1990.

[34] T. Eiter and H. Mannila. Distance measures for point of sets and their computation. *Acta Informatica*, 34, 1997.

[35] W. Emde and D. Wettschereck. Relational instance based learning. In *Proc. of the 13th International Conference on Machine Learning (ICML'1996)*, pages 122 – 130. Morgan Kaufmann, 1996.

[36] S. España and V. Estruch. A memoizing semantics for functional logic languages. In *Proc. of the 13th European Symposium on Programming, Programming Languages and Systems (ESOP'2004)*, pages 109–123. Springer, 2004.

[37] F. Esposito, D. Malerba, and G. Semeraro. A comparative analysis of methods for pruning decision trees. *IEEE Transactions of Pattern Analysis and Machine Intelligence*, 19(5):476–491, 1997.

[38] V. Estruch, C. Ferri, J. Hernández-Orallo, and M.J. Ramírez-Quintana. Shared ensemble learning using multi-trees. In *Proc. of the 8th Ibero-American Conference on Artificial Intelligence (IBERAMIA'02)*, pages 204–213. Springer, 2002.

[39] V. Estruch, C. Ferri, J. Hernández-Orallo, and M.J. Ramírez-Quintana. Smiles: A multi-purpose learning system. In *Proc. of Logics in Artificial Intelligence, European Conference (JELIA'2002)*, pages 529–532. Springer, 2002.

[40] V. Estruch, C. Ferri, J. Hernández-Orallo, and M.J. Ramírez-Quintana. Beam search extraction and forgetting strategies on shared ensembles. In *Proc. of the 4th Int. Workshop on Multiple Classifier Systems (MCS'2003)*, LNCS, pages 206–216. Springer, 2003.

[41] V. Estruch, C. Ferri, J. Hernández-Orallo, and M.J. Ramírez-Quintana. Simple mimetic classifiers. In *Proc. of the 3rd Int. Conference on Machine Learning and Data Mining (MLDM'2003)*, pages 156–171. Springer, 2003.

[42] V. Estruch, C. Ferri, J. Hernández-Orallo, and M.J. Ramírez-Quintana. Bagging decision multi-trees. In *Proc. of the 4th Int. Workshop on Multiple Classifier Systems (MCS'2004)*, LNCS, pages 41–51. Springer, 2004.

[43] V. Estruch, C. Ferri, J. Hernández-Orallo, and M.J. Ramírez-Quintana. Distance based generalisation (best student paper award). In *Proc. of the 15th Int. Conference on Inductive Logic Programming (ILP'2005)*, volume 3625, pages 87–102. Springer, 2005.

[44] V. Estruch, C. Ferri, J. Hernández-Orallo, and M.J. Ramírez-Quintana. Distance based generalisation for graphs (short paper). In *Proc. of the 3rd Int. Workshop on Mining and Learning with Graphs, (MLG'06)*, 2006.

[45] V. Estruch, C. Ferri, J. Hernández-Orallo, and M.J. Ramírez-Quintana. Minimal distance-based generalisation operators for first-order objects. In *Proc. of the 16th Int. Conference on Inductive Logic Programming (ILP'2006)*, pages 169–183. Springer, 2006.

[46] V. Estruch, C. Ferri, J. Hernández-Orallo, and M.J. Ramírez-Quintana. Web categorisation using distance-based decision trees. *Electronic Notes in Theoretical Computer Science*, 157(2):35–40, 2006.

[47] V. Estruch, C. Ferri, J. Hernández-Orallo, and M.J. Ramírez-Quintana. Bridging the gap between distance and generalisation. *Machine Learning (submitted)*, 2008.

[48] V. Estruch, C. Ferri, J.Hernández-Orallo, and M.J. Ramírez-Quintana. Similarity functions for structured data. An application to decision trees. *Revista Iberoamericana de Inteligencia Artificial*, 10(29):109–121, 2006.

[49] C. Ferri, P. A. Flach, and J. Hernández-Orallo. Improving the AUC of probabilistic estimation trees. In *In Proc. of the 14th European Conference on Machine Learning, ECML'03*, Lecture Notes in Computer Science, pages 121–132. Springer, 2003.

[50] C. Ferri, J. Hernández-Orallo, and M. J. Ramírez-Quintana. Learning MDL-guided decision trees for conctructor based languages. In *Work in progress track in the 11th International Conference on Inductive Logic Programming,ILP'01*, volume 3625 of *LNCS*, pages 87–102. Springer, 2001.

[51] D. H. Fisher. Knowledge acquisition via incremental conceptual clustering. *Machine Learning*, 2(2):139–172, 1987.

[52] P. A. Flach. Knowledge representation for inductive learning. In *Proc. of the 5th European Conference on Symbolic and Quantitative Approaches to Reasoning and Uncertainty (ECSQARU'1999)*, pages 160–167. Springer-Verlag, 1999.

[53] P. A. Flach. The use of functional and logic languages in machine learning. In *Proc. of the 9th Int. Workshop on Functional and Logic Programming (WFLP'2000)*, pages 225–237. Universidad Politècnica de València, 2000.

[54] P. A. Flach. The geometry of ROC space: Understanding machine learning metrics through ROC isometrics. In *Proc. of the 20th International Conference on Machine Learning (ICML'2003)*, pages 194–201, 2003.

[55] P. A. Flach, C. Giraud-Carrier, and J. W. Lloyd. Strongly typed inductive concept learning. In *Proc. of the 8th Int. Conference on Inductive Logic Programming (ILP'1998)*, volume 1446, pages 185–194. Springer-Verlag, 1998.

[56] P. A. Flach, E. Gyftodimos, and N. Lachiche. Probabilistic reasoning with terms. volume 7. Electronic Articles in Computer and Information Systems, 2004.

[57] P. A. Flach and N. Lachiche. Confirmation-guided discovery of first order rules with Tertius. *Machine Learning*, 42(1/2):61–95, 2001.

[58] P. A. Flach and N. Lavrac. The role of feature construction in inductive rule learning. In *Proc. of the Int. Workshop on Attribute-Value and Relational Learning: crossing the boundaries. (in conjunction with ICML'2000)*, pages 1–11. 17th International Conference on Machine Learning, 2000.

[59] P.A. Flach and N. Lachiche. Naive bayesian classification of structured data. *Machine Learning*, 57:233–269, 2004.

[60] P. Flener and S. Yilmaz. Inductive synthesis of recursive logic programs: achievements and prospects. *Journal of Logic Programming*, 41(2-3):141–195, 1999.

[61] B. Ganter, G. Stumme, and R. Wille, editors. *Formal Concept Analysis, Foundations and Applications*, volume 3626 of *Lecture Notes in Computer Science*. Springer, 2005.

[62] M.R. Garey and D.S. Johnson. *Computers and Intractability: A Guide to NP Completeness*. Freeman, 1979.

[63] T. Gärtner. A survey of kernels for structured data. *SIGKDD Explorations*, 5(1):49–58, 2003.

[64] T. Gärtner, P.A. Flach, A. Kowalczyk, and A. Smola. Multi-instance kernels. In *Proc. of the 19th International Conference on Machine Learning (ICML'2002)*, pages 179–186. Morgan Kaufmann, 2002.

[65] T. Gärtner, J.W. Lloyd, and P.A. Flach. Kernels and distances for structured data. *Machine Learning*, 57(3):205–232, 2004.

[66] L. Getoor, N. Firedman, D. Koller, and A. Pfeffer. Learning probabilistic relational models. In *Proc. of Relational Data Mining*. Springer-Verlag, 2001.

[67] M. A. Gluck and J. E. Corter. Information, uncertainty and the utility of categories. In *Proc. of the 7th Annual Conference of the Cognitive Science Society*, pages 283–287, 1985.

[68] D.E. Goldberg. *Genetic Algorithms in Search, Optimisation and Machine Learing*. Addison-Wesley, 1989.

[69] A. Golding and P. Rosenbloom. Improving rule-based systems through case-based reasoning. In *National Conference on Artificial Intelligence*, pages 22–27, 1991.

[70] R. W. Hamming. Error detecting and error correcting codes. *Bell System Technical Journal.*, 26(2):147–160, 1950.

[71] R. Hassin and S. Rubinstein. Approximations for the maximum acyclic subgraph problem. *Information Processing Letters*, 51(3):133–140, 1994.

[72] D. Haussler. Convolution kernels on discrete structures. Technical report, Department of Computer Science, University of California at Santa Cruz, 1999.

[73] H.Blockeel and L. De Raedt. Top-down induction of first-order logical decision trees. *Artificial Intelligence*, 101(1-2):285–297, 1998.

[74] J. Hernández-Orallo and M. J. Ramírez-Quintana. A strong complete schmema for inductive functional logic programming. In *Proc. of the 9th Int. Conference on Inductive Logic Programming (ILP'1999)*, pages 116–127, 1999.

[75] J. Hernández-Orallo, M.J. Ramírez-Quintana, and C. Ferri. *Minería de datos*. Pearson/Prentice Hall, 1995.

[76] J.H. Holland. *Adaptation in Natural and Artificial Systems*. University of Michigan Press, 1975.

[77] T. Horváth, J. Ramon, and S. Wrobel. Frequent subgraph mining in outerplanar graphs. In *Proc. of the 12th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 197–206. ACM, 2006.

[78] T. Horváth and G. Turán. Learning logic programs with structured background knowledge. *Artificial Intelligence*, 128(1-2):31–97, 2001.

[79] T. Horváth, S. Wrobel, and U. Bohnebeck. Relational instance-based learning with lists and terms. *Machine Learning*, 43(1/2):53–80, 2001.

[80] A. Hutchinson. Metrics on terms and clauses. In *Proc. of the 9th European Conference on Machine Learning (ECML'1997)*, pages 138–145. Springer-Verlag, 1997.

[81] A. K. Jain, M. N. Murty, and P. J. Flynn. Data clustering: a review. *ACM Comp. Surveys*, 31(3):264–323, 1999.

[82] A. Juan and E. Vidal. Fast -means-like clustering in metric spaces. *Pattern Recognition Letters*, 15(1):19–25, 1994.

[83] k. Sycara, R. Guttal, J. Koning, S. Narasimhan, and D. Navinchandra. Cadet: A case-based synthesis tool for engineering design. *International Journal of Expert Systems*, 4(2).

[84] M. Kearns and Y. Mansour. On the boosting ability of top-down decision tree learning algorithms. In *Proc. of the 28th Annual ACM Symposium on Theory of Computing (STOC'1996)*, pages 459–468. ACM, 1996.

[85] K. Kersting and L. De Raedt. Towards combining inductive logic programming with bayesian networks. In *Proc. of the 11th Int. Conference on Inductive Logic Programming (ILP'2001)*, pages 118–131. Springer-Verlag, 2001.

[86] B. King. Step-wise clustering procedures. *Journal of American Statistical Association*, 69:86–101, 1967.

[87] A. J. Knobbe, A. Siebes, and D. van der Wallen. Multi-relational decision tree induction. In *Principles of Data Mining and Knowledge Discovery*, pages 378–383, 1999.

[88] T. Kohonen. *Self-organization and associative memory (3rd edt.)*. Springer-Verlag, 1989.

[89] R. Kosala and Hendrik Blockeel. Web mining research: A survey. *SIGKDD Explorations*, 2(1):1–15, 2000.

[90] J. Nagata K.P. Hart and J.E. Vaughan. *Encyclopedia of General Topology*. Elsevier, 2003.

[91] S. Kramer. *Relational learning vs. propositionalisation. Investigations in Inductive Logic Programming and Propositional Machine Learning.* PhD thesis, Technishen Universitat Wien, 1999.

[92] S. Kramer, N. Lavrac, and P.A. Flach. Propositionalisation approaches to relational data mining. In N. Lavrac and S. Dzeroski, editors, *Relational Data Mining*, pages 262–291. Springer, 2001.

[93] M.A. Krogel, S. Rawles, F. Zelezný, P.A. Flach, N. Lavrac, and S. Wrobel. Comparative evaluation of approaches to propositionalisation. In *Proc. of the 13th Int. Conference on Inductive Logic Programming (ILP'2003)*, pages 197–214. Springer, 2003.

[94] M.A. Krogel and S. Wrobel. Transformation-based learning using multirelational aggregation. In *Proc. of the 11th Int. Conference on Inductive Logic Prgramming (ILP'2001)*, pages 142–155. Springer, 2001.

[95] N. Lachiche and P.A. Flach. A first-order representation for knowledge discovery and bayesian classification on relational data. In *Int. Workshop on Data Mining, Decision Support, Meta-learning and ILP: Forum for Practical Problem Presentation and Prospective Solutions (PKDD00)*, pages 49–60. 4th Int. Conference on Principles of Data Mining and Knowledge Discovery, 2000.

[96] W. Van Laer and L. De Raedt. How to upgrade propositional learners to first order logic: a case study. In *Machine Learning and its Applications*, pages 102–126, 2001.

[97] N. Lavrač and S. Džeroski. *Inductive Logic Programming: Techniques and Applications*. Ellis Horwood, 1994.

[98] H.A. Leiva. MRDTL: A multi-relational decision tree learning algorithm, 2002.

[99] P. Leslie, M.L. Littman, and A.W. Moore. Reinforcement learning: A survey. *Journal of Artificial Intelligence Research*, 4:235–285, 1996.

[100] V. I. Levenshtein. Binary codes capable of correcting deletions, insertions, and reversals. *Soviet Physics Doklady.*, 10:707–710, 1966.

[101] J. W. Lloyd. Programming in an integrated functional and logic language. *Journal of Functional and Logic Programming*, 1999(3), 1999.

[102] J. W. Lloyd. Learning comprehensible theories from structured data. In *Advanced lectures on machine learning*, pages 203–225. Springer-Verlag, 2003.

[103] H. Lodhi, C. Saunders, J. Shawe-Taylor, N. Cristianini, and C. Watkins. Text classification using string kernels. *Journal Machine Learning Research*, 2:419–444, 2002.

[104] J. B. MacQueen. Some methods for classification and analysis of multivariate observations. In *Proc. of the 5th Berkeley Symposium on Mathematical Statistics and Probability*, pages 281–297. University of California Press, 1967.

[105] J. Marcinkowski and L. Pacholski. Undecidability of the horn clause implication problem. In *Proc. of the 33rd Int. IEEE Symposium on Foundations of Computer Science*, pages 354–362. IEEE Computer Society Press, 1992.

[106] J. McClelland, J. Rumelhart, and The PDP Research Group. *Parallel Distributed Processing*, 1 & 2, 1986.

[107] K. McKusick and K. Thompson. Cobweb/3: A portable implementation. Technical report, NASA, Ames Research Center, 1990.

[108] R. S. Michalski and R. E. Stepp. Learning from observation: conceptual clustering. In J. G. Carbonell, R. S. Michalski, and T. M. Mitchell, editors, *Machine Learning: An Artificial Intelligence Approach*, pages 331–363. Morgan Kaufmann, 1983.

[109] R.S. Michalski. A theory and methodology of inductive learning. In R.S. Michalski, J. Carbonell, and T. Mitchell, editors, *Machine Learning*. Tioga Publishing Company, 1983.

[110] R.S. Michalski. Toward a unified theory of learning: an outline of basic ideas (invited paper). In *Proc. of the 1st World Conference on the Fundamentals of Artificial Intelligence*, 1991.

[111] D. Michie and D.J. Spiegelhalter. *Machine learning, neural and statistical classification*. Ellis Horwood Series in Artificial Intelligence, 1994.

[112] N. Mishra, D. Ron, and R. Swaminathan. A new conceptual clustering framework. *Machine Learning*, 56(1-3):115–151, 2004.

[113] T. M. Mitchell. *Machine Learning*. McGraw-Hill, 1997.

[114] S. Muggleton. Inductive Logic Programming. *New Generation Computing*, 8(4):295–318, 1991.

[115] S. Muggleton. Inverse entailment and Progol. *New Generation Computing*, 13(3&4):245–286, 1995.

[116] S. Muggleton. Stochastic logic programs. In *Advances in Inductive Logic Programming*, pages 254–264. IOS Press, 1995.

[117] S. Muggleton. Inductive logic programming: Issues, results, and the challenge of learning language in logic. *Artificial Intelligence*, 114(1–2):283–296, 1999.

[118] S. Muggleton. Scientific knowledge discovery using Inductive Logic Programming. *Communications of the ACM*, 42(11):42–46, 1999.

[119] S. Muggleton and W. L. Buntine. Machine invention of first order predicates by inverting resolution. In *In Proc. of the 5th International Conference on Machine Learning (ICML'1998)*, pages 339–352. Morgan Kaufmann, 1988.

[120] S. Muggleton and C. Feng. Efficient induction of logic programs. In *Proc. of the 1st Conference on Algorithm Learning Theory (ALT'1990)*, 1990.

[121] S. Muggleton, R.D. King, and M.J.E. Sternberg. Protein secondary structure prediction using logic. In *Proc. of the 2nd Int. Workshop on Inductive Logic Programming (ILP'1992)*, pages 228–259. Springer, 1992.

[122] S.K. Murthy, S. Kasif, and S. Salzberg. A system for induction of oblique decision trees. *Journal of Artificial Intelligence Research*, 2:1–32, 1994.

[123] G. Nagy. State of the art in pattern recognition. In *Proc. of the IEEE*, pages 836–862, 1968.

[124] K.S. Ng and J. W. Lloyd. Predicate selection for structural decision trees. In *Proc. of the 15th. Int. Conference on Inductive Logic Programming (ILP'2005)*, pages 264–278. Springer, 2005.

[125] S-H. Nienhuys-Cheng. Distance between Herbrand interpretations: A measure for approximations to a target concept. In *Proc. of the 7th Int. Workshop on Inductive Logic Programming (ILP'1997)*, pages 213–226. Springer-Verlag, 1997.

[126] S.H. Nienhuys-Cheng and R. de Wolf. *Foundations of Inductive Logic Programming*, volume 1228. 1997.

[127] C. Papadimitriou. *Computational Complexity*. Addison-Wesley, 1994.

[128] M.J. Pazzani and D. Billsus. Learning and revising user profiles: The identification of interesting web sites. *Machine Learning*, 27(3):313–331, 1997.

[129] G. Plotkin. A note on inductive generalisation. *Machine Intelligence*, 5:153–163, 1970.

[130] T. Poranen. Two new approximation algorithms for the maximum planar subgraph problem. *Acta Cybernetica*, To Appear, 2006.

[131] J. R. Quinlan. *C4.5: Programs for Machine Learning*. Morgan Kaufmann, San Mateo, CA, 1993.

[132] J. R. Quinlan. Combining instance-based and model-based learning. In *Proc. of the 10th International Conference on Machine Learning, ICML'93*, pages 236–243. Morgan Kaufmann, 1993.

[133] J.R. Quinlan. Induction of decision trees. In *Reading in Machine Learning*. Morgan Kaufmann, 1990.

[134] J.R. Quinlan and R.M. Cameron Jones. Foil: A midterm report. In *Proc. of the European Conference on Machine Learning (ECML'1993)*, pages 3–20. Springer, 1993.

[135] J.R. Quinlan and R.M. Cameron Jones. Foil: A midterm report. Technical report, Basser Department of Computer Science, University of Sydney, 2006.

[136] Luc De Raedt. Attribute-value learning versus inductive logic programming: The missing links (extended abstract). In *Int. Conference of Inductive Logic Programming (ILP'1998)*, pages 1–8, 1998.

[137] J. Ramon and M. Bruynooghe. A framework for defining distances between first-order logic objects. In *Proc. of the 8th Int. Conference on Inductive Logic Programming (ILP'1998)*, pages 271–280. Springer, 1998.

[138] J. Ramon and M. Bruynooghe. A polynomial time computable metric between point sets. *Acta Informatica*, 37(10):765–780, 2001.

[139] J. Ramon, M. Bruynooghe, and W. Van Laer. Distance measures between atoms. In *CompulogNet Area Meeting on Computational Logic and Machine Learing*, pages 35–41. University of Manchester, UK, 1998.

[140] J. Ramon, T. Francis, and H. Blockeel. Learning a go heuristic with TILDE. In *Computers and Games*, pages 151–169, 2000.

[141] J. Ramon and T. Gärtner. Expressivity versus efficiency of graph kernels. In *Proc. of the 1st Int. Workshop on Mining Graphs, Trees and Sequences (MTGS'2003)*, pages 65–74. ECML/PKDD'03 workshop proceedings, 2003.

[142] J. Ramon and L. De Raedt. Multiple-instance neural networks. In *Attribute-Value and Relational Learning: Crossing the Boundaries. A Workshop at 17th International Conference on Machine Learning (ICML'2000).*

[143] Y. Reich and S. Fenves. The formation and use of abstract concepts in design. In *Concept Formation: Knowledge and Experience in Unsupervised Learning*, pages 323–353. Morgan Kaufmann, 1991.

[144] J. Rissanen. An introduction to the MDL principle. Technical report, Helsinkin Institute for Information Technology, 2005.

[145] J. A. Robinson. A machine-oriented logic based on the resolution principle. *Journal of ACM*, 12(1):23–41, 1965.

[146] F. Rosenblatt. *Principles of Neurodynamics*. Spartan Books, 1962.

[147] C. Rouveirol. Extensions of inversion of resolution applied to theory completion. In *Proc. of the 5th Int. Conference on Inductive Logic Programming (ILP'1992)*, pages 63–92. AP, 1992.

[148] C. Rouveirol and J.F. Puget. Beyond inversion of resolution. In *Proc. of the 7th Int. Conference on Machine Learning (ICML'1990)*, pages 122–130. Porter and Money, 1990.

[149] Sajama and A. Orlitsky. Estimating and computing density-based distance metrics. In *Proc. of the 22nd International Conference on Machine Learning (ICML'2005)*, pages 760–767. ACM, 2005.

[150] S. Salzberg. A nearest hyperrectangle learning method. *Machine Learning*, 6:251–276, 1991.

[151] B. Scholkopf and A.J. Smola. *Learning with Kernels*. MIT Press, 2002.

[152] M. Sebag. Distance induction in first order logic. In *Proc. of the 7th Int. Workshop on Inductive Logic Programming (ILP'1997)*, LNCS, pages 264–272. Springer, 1997.

[153] W.D. Seeman and R. S. Michalski. The cluster3 system for goal-oriented conceptual clustering: method and preliminary result. In *Data Mining VII: Data, Text and Web Mining and their Business Applications*. 2006.

[154] E. Y. Shapiro. *Algorithmic Program Debugging*. MIT Press, 1983.

[155] P.H.A. Sneath and R.R. Sokal. *Numerical Taxonomy*. Freeman,London, Uk, 1973.

[156] A. Srinivasan, S. Muggleton, R. King, and M. Sternberg. Mutagenesis: ILP experiments in a non-determinate biological domain. In *In Proc. of the 4th Int. Workshop on Inductive Logic Programming (ILP'1994)*, pages 217–232. Gesellschaft fur Mathematik und Datenverarbeitung, 1994.

[157] A. Stanfill and D. Waltz. Toward memory-based resoning. *Communications of the ACM.*, 29:1213–1228, 1986.

[158] R. E. Stepp and R. S. Michalski. Conceptual clustering: Inventing goal oriented classifications of structured objects. In J. G. Carbonell, R. S. Michalski, and T. M. Mitchell, editors, *Machine Learning: An Artificial Intelligence Approach, Volume II*, pages 471–498. Morgan Kaufmann, 1986.

[159] P.J. Tan and D.L. Dowe. MML inference of decision graphs with multi-way joins. In *Proc. of the Australian Joint Conference on Artificial Intelligence*, pages 131–142, 2002.

[160] P.J. Tan and D.L. Dowe. MML inference of oblique decision trees. In *Australian Conference on Artificial Intelligence*, pages 1082–1088, 2004.

[161] R. Rivest T.H. Cormen, C. Leiserson and C. Stein. *Introduction to Algorithms*. MIT Press, 2000.

[162] Chris Thornton. *Truth from Trash: How Learning Makes Sense*. The MIT Press, 2000.

[163] S. Tong and D. Koller. Support vector machine active learning with applications to text classification. In *Proc. of the International Conference of Machine Learning (ICML'2000)*, pages 999–1006. ACM, 2000.

[164] E. Vidal. An algorithm for finding nearest neighbours in (approximately) constant average time. *Pattern Recognition Letters*, 4(3):145–157, 1986.

[165] C. S. Wallace and D. L. Dowe. Minimum message length and Kolmogorov complexity. *The Computer Journal*, 42(4):270–283, 1999.

[166] J. Wang and J.D. Zucker. Solving multiple-instance problem: A lazy learning approach. In *Proc. of the 17th International Conference on Machine Learning (ICML'2000)*, pages 1119–1125, 2000.

[167] J. Wneck, K. Kaufman, E. Bloedorn, and R.S. Michalski. Inductive learning system aq15c: The method and user's guide. Technical report, Machine Learning and Inference Laboratory, George Mason University, 1995.

[168] S. Wrobel. An algorithm for multi-relational discovery of subgroups. In *Proc. of the Principles of Data Mining and Knowledge Discovery (PKDD'1997)*, pages 78–87. Springer, 1997.

[169] R. Xu and D. Wunsch. Survey of clustering algorithms. *IEEE Transactions on Neural Networks.*, 16(3), 2005.

[170] X. Yan and J. Han. gspan: Graph-based substructure pattern mining. In *Proc. of the 2nd International Conference on Data Mining (ICDM'2002)*, pages 721–724. IEEE Computer Society, 2002.

[171] Q. Yang and X. Wu. 10 machine learning problems in data mining research. *International Journal in Information Technology and Decision Making*, 5(4):597–604, 2006.

[172] Y. Yang and J.O. Pedersen. A comparative study on feature selection in text categorization. In *Proc. of 14th International Conference on Machine Learning (ICML'1997)*, pages 412–420. Morgan Kaufmann, 1997.

[173] Y. Yuan and M.J. Shaw. Induction of fuzzy decision trees. *Fuzzy Sets and Systems*, 69(2):125–139, 1995.

[174] F. Zelezný and N. Lavrac. Propositionalisation-based relational subgroup discovery. *Machine Learning*, 62(1-2):23–63, 2006.

[175] X. Zhu. Semi-supervised learning literature survey. Technical report, University of Wisconsin, 2006.