Chapter 1
A Theory of Inductive Query Answering

Luc De Raedt, Manfred Jaeger, Sau Dan Lee, and Heikki Mannila

Abstract We introduce the Boolean inductive query evaluation problem, which is concerned with answering inductive queries that are arbitrary Boolean expressions over monotonic and anti-monotonic predicates. Boolean inductive queries can be used to address many problems in data mining and machine learning, such as local pattern mining and concept-learning, and actually provides a unifying view on many machine learning and data mining tasks. Secondly, we develop a decomposition theory for inductive query evaluation in which a Boolean query Q is reformulated into k sub-queries $Q_i = Q_A \wedge Q_M$ that are the conjunction of a monotonic and an anti-monotonic predicate. The solution to each sub-query can be represented using a version space. We investigate how the number of version spaces k needed to answer the query can be minimized and define this as the dimension of the solution space and query. Thirdly, we generalize the notion of version spaces to cover Boolean queries, so that the solution sets form a closed Boolean-algebraic space under the usual set operations. The effects of these set operations on the dimension of the involved queries are studied.

Luc De Raedt
Dept. of Computer Science, Katholieke Universiteit Leuven, e-mail: luc.deraedt@cs.kuleuven.be

Manfred Jaeger
Dept. of Computer Science, Aalborg Universitet, e-mail: jaeger@cs.aau.dk

Sau Dan Lee
Dept. of Computer Science, University of Hong Kong, e-mail: sdlee@cs.hku.hk

Heikki Mannila
Dept. of Information and Computer Science, Helsinki University of Technology, e-mail: mannila@cs.helsinki.fi

1.1 Introduction

Many data mining and learning problems address the problem of finding a set of patterns, concepts or rules that satisfy a set of constraints. Formally, this can be described as the task of finding the set of patterns or concepts $Th(Q, \mathcal{D}, \mathcal{L}) = \{\varphi \in \mathcal{L} \mid Q(\varphi, \mathcal{D})\}$ i.e., those patterns and concepts φ satisfying query Q on a data set $\mathcal{D}$. Here $\mathcal{L}$ is the language in which the patterns or concepts are expressed, and Q is a predicate or constraint that determines whether a pattern or concept φ is a solution to the data mining task or not [20]. This framework allows us to view the predicate or the constraint Q as an *inductive query* [7]. It is then the task of machine learning or data mining system to efficiently generate the answers to the query. This view of mining and learning as a declarative querying process is also appealing as the basis for a theory of mining and learning. Such a theory would be analogous to traditional database querying in the sense that one could study properties of different pattern languages $\mathcal{L}$, different types of queries (and query languages), as well as different types of data. Such a theory could also serve as a sound basis for developing algorithms that solve inductive queries.

It is precisely such a theory that we introduce in this chapter. More specifically, we study inductive queries that are Boolean expressions over monotonic and anti-monotonic predicates. An example query could ask for molecular fragments that have frequency at least 30 percent in the active molecules or frequency at most 5 percent in the inactive ones [15]. This type of Boolean inductive query is amongst the most general type of inductive query that has been considered so far in the data mining and the machine learning literature. Indeed, most approaches to constraint based data mining use either single constraints (such as minimum frequency), e.g., [2], a conjunction of monotonic constraints, e.g., [24, 10], or a conjunction of monotonic and anti-monotonic constraints, e.g., [8, 15]. However, [9] has studied a specific type of Boolean constraints in the context of association rules and itemsets. It should also be noted that even these simpler types of queries have proven to be useful across several applications, which in turn explains the popularity of constraint based mining in the literature. Inductive querying also allows one to address the typical kind of concept-learning problems that have been studied within computational learning theory [14] including the use of queries for concept-learning [3]. Indeed, from this perspective, there will be a constraint with regard to every positive and negative instance (or alternatively some constraints at the level of the overall dataset), and also the answers to queries to oracle (membership, equivalence, etc.) can be formulated as constraints.

Our theory of Boolean inductive queries is first of all concerned with characterizing the solution space $Th(Q, \mathcal{D}, \mathcal{L})$ using notions of convex sets (or version spaces [12, 13, 22]) and border representations [20]. This type of representations have a long history in the fields of machine learning [12, 13, 22] and data mining [20, 5]. Indeed, within the field of data mining it has been re-

alized that the space of solutions w.r.t. a monotone constraint is completely characterized by its set (or border) of maximally specific elements [20, 5]. This property is also exploited by some effective data mining tools, such as Bayardo's MaxMiner [5], which output this border set. Border sets have an even longer history in the field of machine learning, where Mitchell recognized as early as 1977 that the space of solutions to a concept-learning task could be represented by two borders, the so-called S and G-set (where S represents the set of maximally specific elements in the solution space and G the set of maximally general ones). These data mining and machine learning viewpoints on border sets have been unified by [8, 15], who introduced the level-wise version space algorithm that computes the S and G set w.r.t. a conjunction of monotonic and anti-monotonic constraints.

In the present chapter, we build on these results to develop a decomposition approach to solving arbitrary Boolean queries over monotonic and anti-monotonic predicates. More specifically, we investigate how to decompose arbitrary queries Q into a set of sub-queries Q_k such that $Th(Q, \mathcal{D}, \mathcal{L}) = \bigcup_i Th(Q_i, \mathcal{D}, \mathcal{L})$, and each $Th(Q_i, \mathcal{D}, \mathcal{L})$ can be represented using a single version space. This way we obtain a query plan, in that to obtain the answer to the overall query Q all of the sub-queries Q_i need to be answered. As these Q_i yield version spaces, they can be computed by existing algorithms such as the level-wise version space algorithm of [8]. A key technical contribution is that we also introduce a canonical decomposition in which the number of needed subqueries k is minimal.

This motivates us also to extend the notion of version spaces into generalized version spaces (GVSes) [18] to encapsulate solution sets to such general queries. It is interesting that GVSes form an algebraic space that is closed under the usual set operations: union, intersection and complementation. We prove some theorems that characterize the effect on the dimensions of such operation. Because GVSes are closed under these operations, the concept of GVSes gives us the flexibility to rewrite queries in various forms, find the solutions of subqueries separately, and eventually combine the solutions to obtain the solution of the original query. This opens up many opportunites for query optimization.

This chapter is organized as follows. In Section 1.2, we define the inductive query evaluation problem and illustrate it on the pattern domains of strings and itemsets. We model the solution sets with GVSes, which are introduced in Section 1.3. In Section 1.4, we introduce a decomposition approach to reformulate the original query in simpler sub-queries. Finally, we give our conclusions in Section 1.6.

1.2 Boolean Inductive Queries

We begin with describing more accurately the notions of patterns and pattern languages, as we use them in this chapter. We always assume that datasets consist of a list of data items from a set $\mathcal{U}$, called the *domain* or the *universe* of the dataset.

A *pattern* or *concept* ϕ for $\mathcal{U}$ is some formal expression that defines a subset ϕ_e of $\mathcal{U}$. When $u \in \phi_e$ we say that ϕ *matches* or *covers* u. A *pattern language* $\mathcal{L}$ for $\mathcal{U}$ is a formal language of patterns. The terminology used here is applicable to both concept-learning and pattern mining. In concept-learning, $\mathcal{U}$ would be the space of examples, $2^{\mathcal{U}}$ the set of possible concepts (throughout, we use 2^X to denote the powerset of X), and $\mathcal{L}$ the set of concept-descriptions. However, for simplicity we shall throughout the chapter largely employ the terminology of pattern mining. It is, however, important to keep in mind that it also applies to concept-learning and other machine learning tasks.

Example 1.2.1 Let $\mathcal{I} = \{i_1, \ldots, i_n\}$ be a finite set of possible items, and $\mathcal{U}_\mathcal{I} = 2^\mathcal{I}$ be the universe of itemsets over $\mathcal{I}$. The traditional pattern language for this domain is $\mathcal{L}_\mathcal{I} = \mathcal{U}_\mathcal{I}$. A pattern $\phi \in \mathcal{L}_\mathcal{I}$ covers the set $\phi_e := \{\mathcal{H} \subseteq \mathcal{I} \mid \phi \subseteq \mathcal{H}\}$.

Instead of using $\mathcal{L}_\mathcal{I}$ one might also consider more restrictive languages, e.g., the sublanguage $\mathcal{L}_{\mathcal{I},k} \subseteq \mathcal{L}_\mathcal{I}$ that contains the patterns in $\mathcal{L}_\mathcal{I}$ of size at most k.

Alternatively, one can also use more expressive languages, the maximally expressive one being the language $2^{\mathcal{U}_\mathcal{I}}$ of all subsets of the universe, or as is common in machine learning the language of conjunctive concepts, $\mathcal{L}_{\overline{\mathcal{I}}}$, which consists of all conjunctions of literals over $\mathcal{I}$, that is, items or their negation. This language can be represented using itemsets that may contain items from $\overline{\mathcal{I}} = \mathcal{I} \cup \{\neg i | i \in \mathcal{I}\}$. It is easy to see that the basic definitions for itemsets carry over for this language provided that the universe of itemsets is $\mathcal{U}_{\overline{\mathcal{I}}}$. □

Example 1.2.2 Let Σ be a finite alphabet and $\mathcal{U}_\Sigma = \Sigma^*$ the universe of all strings over Σ. We will denote the empty string with ϵ. The traditional pattern language in this domain is $\mathcal{L}_\Sigma = \mathcal{U}_\Sigma$. A pattern $\phi \in \mathcal{L}_\Sigma$ covers the set $\phi_e = \{\sigma \in \Sigma^* \mid \phi \sqsubseteq \sigma\}$, where $\phi \sqsubseteq \sigma$ denotes that ϕ is a substring of σ. □

One pattern ϕ for U is *more general* than a pattern ψ for U, written $\phi \succeq \psi$, if and only if $\phi_e \supseteq \psi_e$. For two itemset patterns $\phi, \psi \in \mathcal{L}_\mathcal{I}$, for instance, we have $\phi \succeq \psi$ iff $\phi \subseteq \psi$. For two conjunctive concepts $\phi, \psi \in \mathcal{L}_\mathcal{C}$, for instance, we have $\phi \succeq \psi$ iff $\phi \models \psi$. For two string patterns $\phi, \psi \in \mathcal{L}_\Sigma$ we have $\phi \succeq \psi$ iff $\phi \sqsubseteq \psi$. A pattern language $\mathcal{L}'$ is *more expressive* than a pattern language $\mathcal{L}$, written $\mathcal{L}' \succeq \mathcal{L}$, iff for every $\phi \in \mathcal{L}$ there exists $\phi' \in \mathcal{L}'$ with $\phi_e = \phi'_e$.

A pattern *predicate* defines a primitive property of a pattern, often relative to some data set D (a set of examples). For any given pattern or concept, a pattern predicate evaluates to either *true* or *false*. Pattern predicates are the

basic building blocks for building inductive queries. We will be mostly interested in *monotonic* and *anti-monotonic* predicates. A predicate p is monotonic, if $\mathsf{p}(\phi)$ and $\psi \succeq \phi$ implies $\mathsf{p}(\psi)$, i.e., p is closed under generalizations of concepts. Similarly, anti-monotonic predicates are defined by closure under specializations.

1.2.1 Predicates

We now introduce a number of pattern predicates that will be used for illustrative purposes throughout this chapter. Throughout the section we will introduce predicates that have been inspired by a data mining setting, in particular by the system MolFea [15], as well as several predicates that are motivated from a machine learning perspective, especially, by Angluin's work on learning concepts from queries [3].

Pattern predicates can be more or less general in that they may be applied to patterns from arbitrary languages $\mathcal{L}$, only a restricted class of languages, or perhaps only are defined for a single language. Our first predicate can be applied to arbitrary languages:

- $\mathsf{minimum\_frequency}(p,n,D)$ evaluates to true iff p is a pattern that occurs in database D with frequency at least $n \in \mathbb{N}$. The frequency $f(\phi, D)$ of a pattern ϕ in a database D is the (absolute) number of data items in D covered by ϕ. Analogously, the predicate $\mathsf{maximum\_frequency}(p, n, D)$ is defined. $\mathsf{minimum\_frequency}$ is a monotonic, $\mathsf{maximum\_frequency}$ an anti-monotonic predicate.

These predicates are often used in data mining, for instance, when mining for frequent itemsets, but they can also be used in the typical concept-learning setting, which corresponds to imposing the constraints $\mathsf{minimum\_frequency}(p,|P|,P) \wedge \mathsf{maximum\_frequency}(p,0,N)$ where P is the set of positive instances and N the set of negative ones, that is, all positive examples should be covered and none of the negatives ones.

A special case of these frequency related predicates is the predicate

- $\mathsf{covers}(p, u) \equiv \mathsf{minimum\_frequency}(p, 1, \{u\})$, which expresses that the pattern (or concept) p covers the example u. covers is monotonic.

This predicate is often used in a concept-learning setting. Indeed, the result of a membership query (in Angluin's terminology) is a positive or negative example and the resulting constraint corresponds to the predicate covers or its negation.

The next predicate is defined in terms of some fixed pattern ψ from a language $\mathcal{L}$. It can be applied to other patterns for $\mathcal{U}$.

- is_more_general(p,ψ) is a monotonic predicate that evaluates to true iff p is a pattern for U with $p \succeq \psi$. Dual to the is_more_general predicate one defines the anti-monotonic is_more_specific predicate.

The is_more_general(p,ψ) predicate only becomes specific to the language $\mathcal{L}$ for the fixed universe $\mathcal{U}$ through its parameter ψ. By choice of other parameters, the predicate is_more_general becomes applicable to any other pattern language $\mathcal{L}'$. This type of predicate has been used in a data mining context to restrict the patterns of interest [15] to specify that patterns should be sub- or superstrings of a particular pattern. In a concept learning context, these predicates are useful in the context of learning from queries [3]. This is a framework in which the learner may pose queries to an oracle. The answers to these queries then result in constraints on the concept. There are several types of queries that are considered in this framework and that are related to the is_more_general and the is_more_specific predicates:

- a subset, respectively superset query [3], determines whether a particular concept must cover a subset, respectively a superset, of the positive examples or not. The answers to these queries directly correspond to constraints using the predicates is_more_specific and is_more_general.
- an equivalence query determines whether a particular concept-description ϕ is equivalent to the target concept or not. This can be represented using the predicate equivalent(c, ϕ), which can be defined as follows:

$$\mathsf{equivalent}(c, \phi) \equiv \mathsf{is\_more\_general}(c, \phi) \wedge \mathsf{is\_more\_specific}(c, \phi)$$

- a disjointness query determines whether or not a particular concept ϕ overlaps with the target concept c, that is, whether there are elements in the universe which are covered by both ϕ and c. This can be represented using the anti-monotonic predicate disjoint(c, ϕ), which evaluates to true iff $c_e \cap \phi_e = \emptyset$. It can be defined in terms of generality in case the language of concepts $\mathcal{L}$ is closed under complement:

$$\mathsf{disjoint}(c, \phi) \equiv \mathsf{is\_more\_specific}(c, \neg\phi)$$

- an exhaustiveness query determines whether a particular concept ϕ together with the target concept c covers the whole universe; this can be written using the monotonic predicate exhausts(c, ϕ), which evaluates to true iff $c_e \cup \phi_e = \mathcal{U}$. It can be defined in terms of generality in case the language of concepts $\mathcal{L}$ is closed under complement:

$$\mathsf{exhausts}(c, \phi) \equiv \mathsf{is\_more\_general}(c, \neg\phi)$$

The next pattern predicate is applicable to patterns from many different languages $\mathcal{L}$. It is required, however, that on $\mathcal{L}$ the length of a pattern is defined.

- length_at_most(p,n) evaluates to true for $p \in \mathcal{L}$ iff p has length at most n. Analogously the length_at_least(p,n) predicate is defined.

We apply the length_at_most-predicate mostly to string patterns, where the length of a pattern is defined in the obvious way (but note that e.g., for itemset patterns $\phi \in \mathcal{L}_\mathcal{I}$ one can also naturally define the length of ϕ as the cardinality of ϕ, and then apply the length_at_most-predicate).

1.2.2 Illustrations of Inductive Querying

Let us now also look into the use of these predicates for solving a number of machine learning and data mining problems. First, we look into association rule mining, for which we introduce a pattern predicate that is applicable only to itemset patterns $\phi \in \mathcal{L}_\mathcal{I}$ for some fixed $\mathcal{I}$. The dependence on $\mathcal{I}$ again comes through the use of a parameter, here some fixed element $i_j \in \mathcal{I}$.

- association(p,i_j,D) evaluates to true for $p \in \mathcal{L}_\mathcal{I}$ iff $p \Rightarrow i_j$ is a valid association rule in D, i.e., for all data items $d \in D$: if $p \subseteq d$ then $i_j \in d$. association is anti-monotonic.

The predicate association—as defined above—allows only valid association rules, i.e., association rules that have a confidence of 100%. It could also be applied to string patterns. Then the condition would be that $p \Rightarrow i$ is valid iff for all strings $d \in D$: if $d \in \phi_p$ then $d \in \phi_{pi}$, where pi denotes the concatenation of the string p and the character i.

Secondly, let us investigate the use of constraints in clustering, where must-link and cannot-link constraints have been used in machine learning. We can phrase a clustering problem as a concept learning task in our framework by interpreting a clustering of a set of objects $\mathcal{O}$ as a binary relation $cl \subset \mathcal{O} \times \mathcal{O}$, where $cl(o, o')$ means that o and o' are in the same cluster. Thus, with $\mathcal{U} = \mathcal{O} \times \mathcal{O}$, a clustering is just a pattern in our general sense (one may use any suitable pattern language that provides a unique representation for clusterings). According to our general notion of generality of patterns, a clustering c is more general than another clustering c' if $c_e \supseteq c'_e$, i.e., if more pairs of objects belong to the same cluster in c as in c', which, in turn, means that c can be obtained from c' by merging of clusters. Furthermore, the most specific generalizations of c are just the clusterings obtained by merging two clusters of c, whereas the most general specializations are the clusterings obtained by splitting one cluster of c into two.

We can now express as a concept learning task the problem of retrieving all possible clusterings that satisfy certain constraints.

The first two kinds of useful constraints represent generally desirable properties of clusterings:

- clusters_atmost(cl, k) evaluates to true if the clustering cl consists of at most k clusters. This predicate is monotonic.
- within_cluster_distance_atmost(cl, r) evaluates to true if no two objects with distance $> r$ are in the same cluster. This predicate is anti-monotonic.

Specific constraints as used in constraint based clustering are now:

- must_link(cl, o_1, o_2), which evaluates to true when the two objects o_i are in the same cluster (monotonic).
- must_not_link(cl, o_1, o_2), which evaluates to true when the two objects o_i are in different clusters (anti-monotonic)

Using these predicates, we could for a given dataset retrieve with the query

$$\text{clusters\_atmost}(cl, 5) \wedge \text{within\_cluster\_distance\_atmost}(cl, 0.5) \wedge \text{must\_link}(cl, o_1, o_2)$$

all possible clusterings of at most 5 clusters, such that no clusters contain points farther apart than 0.5 (in the underlying metric used for the dataset), and such that the two designated objects o_1, o_2 are in the same cluster.

Finally, machine learning has also devoted quite some attention to multi-instance learning. In multi-instance learning examples consist of a set of possible instances and an example is considered covered by a concept, whenever the concept covers at least one of the instances in the set. One way of formalizing multi-instance learning within our framework is to adapt the notion of coverage to have this meaning. Alternatively, a multi-instance learning example could be represented using the predicate minimum-frequency(c,1,e) where c is the target concept and e is the example, represented here as a set of its instances. A negative example then corresponds to the negation of this expression or requiring that maximum-frequency(c,0,e) holds.

1.2.3 A General Framework

In all the preceding examples the pattern predicates have the form pred(p,*params*) or pred(p,D,*params*), where *params* is a tuple of parameter values, D is a data set and p is a pattern variable.

We also speak a bit loosely of pred alone as a pattern predicate, and mean by that the collection of all pattern predicates obtained for different parameter values *params*.

We say that pred(p,D,*params*) is a *monotonic* predicate, if for all pattern languages $\mathcal{L}$ to which pred(p,D,*params*) can be applied, and all $\phi, \psi \in \mathcal{L}$:

$$\phi \succeq \psi \;\Rightarrow\; \text{pred}(\psi, D, params) \rightarrow \text{pred}(\phi, D, params)$$

We also say that pred is monotonic, if pred(p,D,*params*) is monotonic for all possible parameter values *params*, and all datasets D. Analogously, we define

anti-monotonicity of a predicate by the condition

$$\phi \succeq \psi \;\Rightarrow\; \mathsf{pred}(\phi, D, params) \rightarrow \mathsf{pred}(\psi, D, params).$$

A pattern predicate $\mathsf{pred}(p,D,params)$ that can be applied to the patterns from a language $\mathcal{L}$ defines the *solution set*

$$Th(\mathsf{pred}(p, D, params), \mathcal{L}) = \{\phi \in \mathcal{L} \mid \mathsf{pred}(\phi, D, params) = true\}.$$

Furthermore, for monotonic predicates $\mathsf{m}(\ldots)$ these sets will be *monotone*, i.e., for all $\phi \succeq \psi \in \mathcal{L} : \psi \in Th(\mathsf{m}(\ldots), \mathcal{L}) \rightarrow \phi \in Th(\mathsf{m}(\ldots), \mathcal{L})$. Similarly, anti-monotonic predicates define anti-monotone solution sets.

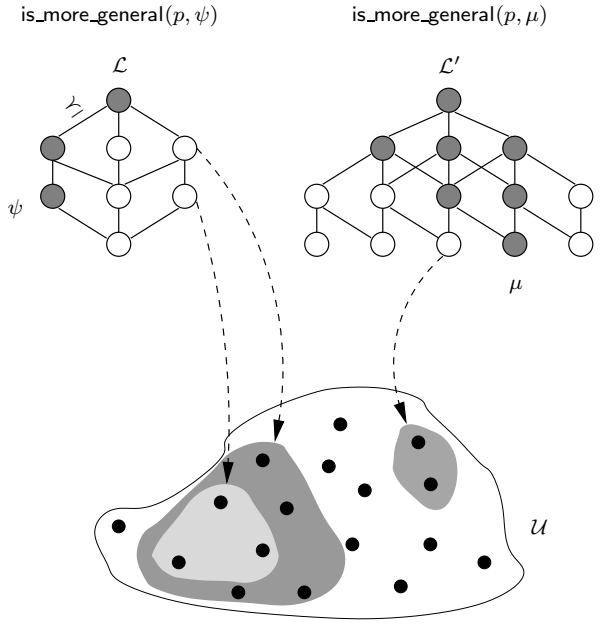

Fig. 1.1 Pattern languages and pattern predicates

Figure 1.1 illustrates the definitions given so far. It gives a schematic representation of a universe $\mathcal{U}$ and two pattern languages $\mathcal{L}, \mathcal{L}'$ for $\mathcal{U}$. The $\succeq$ relation between patterns is represented by lines connecting immediate neighbors in the $\succeq$ relation, with the more general patterns being above the more specific ones. For two patterns from $\mathcal{L}$ and one pattern from $\mathcal{L}'$ the subsets of the universe covered by the patterns are indicated. For the pattern $\psi \in \mathcal{L}$ and $\mu \in \mathcal{L}'$ the figure shows the interpretation of the pattern predicates $\mathsf{is\_more\_general}(p, \psi)$, respectively $\mathsf{is\_more\_general}(p, \mu)$ by filled nodes corresponding to patterns for which these predicates are true.

Example 1.2.3 Consider the string data set $D = \{\mathtt{abc}, \mathtt{abd}, \mathtt{cd}, \mathtt{d}, \mathtt{cd}\}$. Here we have pattern frequencies $f(\mathtt{abc}, D) = 1$, $f(\mathtt{cd}, D) = 2$, $f(\mathtt{c}, D) = 3$, $f(\mathtt{d}, D) = 4$, $f(\mathtt{abcd}, D) = 0$. And trivially, $f(\epsilon, D) = |D| = 5$. Thus, the following predicates evaluate to true: $\mathsf{minimum\_frequency}(\mathtt{c}, 2, D)$, $\mathsf{minimum\_frequency}(\mathtt{cd}, 2, D)$, $\mathsf{maximum\_frequency}(\mathtt{abc}, 2, D)$.

The pattern predicate $\mathsf{m} := \mathsf{minimum\_frequency}(p, 2, D)$ defines $Th(\mathsf{m}, \mathcal{L}_\Sigma) = \{\epsilon, a, b, c, d, ab, cd\}$, and the predicate $\mathsf{a} := \mathsf{maximum\_frequency}(p, 2, D)$ defines the infinite set $Th(\mathsf{a}, \mathcal{L}_\Sigma) = \mathcal{L}_\Sigma \setminus \{\epsilon, c, d\}$. □

The definition of $Th(\mathsf{pred}(\mathrm{p}, D, params), \mathcal{L})$ is extended in the natural way to a definition of the solution set $Th(Q, \mathcal{L})$ for Boolean combinations Q of pattern predicates: $Th(\neg Q, \mathcal{L}) := \mathcal{L} \setminus Th(Q, \mathcal{L})$, $Th(Q_1 \vee Q_2, \mathcal{L}) := Th(Q_1, \mathcal{L}) \cup Th(Q_2, \mathcal{L})$. The predicates that appear in Q may reference one or more data sets $D_1, \ldots, D_n$. To emphasize the different data sets that the solution set of a query depends on, we also write $Th(Q, D_1, \ldots, D_n, \mathcal{L})$ or $Th(Q, \mathcal{D}, \mathcal{L})$ for $Th(Q, \mathcal{L})$.

Example 1.2.4 Let D_1, D_2 be two datasets over the domain of itemsets $\mathcal{U}_\mathcal{I}$. Let $i \in I$, and consider the query

$$Q = \mathsf{association}(p, i, D_1) \wedge \mathsf{minimum\_frequency}(p, 10, D_1) \\ \wedge \neg \mathsf{association}(p, i, D_2).$$

The solution $Th(Q, D_1, D_2, \mathcal{L}_\mathcal{I})$ consists of all $p \in \mathcal{L}_\mathcal{I} = 2^\mathcal{I}$ for which $p \Rightarrow i$ is a valid association rule with support at least 10 in D_1, but $p \Rightarrow i$ is not a valid association rule in D_2. □

We are interested in computing solution sets $Th(Q, \mathcal{D}, \mathcal{L})$ for Boolean queries Q that are constructed from monotonic and anti-monotonic pattern predicates. As anti-monotonic predicates are negations of monotonic predicates, we can, in fact, restrict our attention to monotonic predicates. We can thus formally define the *Boolean inductive query evaluation problem* addressed in this chapter.

Given

- a language $\mathcal{L}$ of patterns,
- a set of monotonic predicates $\mathcal{M} = \{\mathsf{m}_1(p, D_1, params_1), \ldots, \mathsf{m}_n(p, D_n, params_n)\}$,
- a query Q that is a Boolean expression over the predicates in $\mathcal{M}$,

Find

the set of patterns $Th(Q, D_1, \ldots, D_n, \mathcal{L})$, i.e., the solution set of the query Q in the language $\mathcal{L}$ with respect to the data sets $D_1, \ldots, D_n$. Moreover,

the representation of the the solution set should be optimized with regard to understandability and representation size.

1.3 Generalized Version Spaces

We next investigate the structure of solution sets $Th(Q, \mathcal{D}, \mathcal{L})$ based on the classic notion of version spaces [22][12].

Definition 1.3.1 Let $\mathcal{L}$ be a pattern language, and $I \subseteq \mathcal{L}$. If for all $\phi, \phi', \psi \in \mathcal{L}$ it holds that $\phi \preceq \psi \preceq \phi'$ and $\phi, \phi' \in I$ implies $\psi \in I$, then I is called a *version space* (or a convex set). The set of all version spaces for $\mathcal{L}$ is denoted $\mathcal{VS}^1(\mathcal{L})$.

A *generalized version space (GVS)* is any finite union of version spaces. The set of all generalized version spaces for $\mathcal{L}$ is denoted by $\mathcal{VS}^{\mathbb{Z}}(\mathcal{L})$.

The *dimension* of a generalized version space I is the minimal k, such that I is the union of k version spaces.

Version spaces are particularly useful when they can be represented by boundary sets, i.e., by the sets $G(Q, \mathcal{D}, \mathcal{L})$ of their maximally general elements, and $S(Q, \mathcal{D}, \mathcal{L})$ of their most specific elements. Generalized version spaces can then be represented simply by pairs of boundary sets for their convex components. Our theoretical results do not require boundary representations for convex sets. However, in most cases our techniques will be more useful for pattern languages in which convexity implies boundary representability. This is guaranteed for finite languages [12].

Definition 1.3.2 The dimension of a query Q is the dimension of the generalized version space $Th(Q, \mathcal{D}, \mathcal{L})$.

Example 1.3.3 Let $\Sigma = \{a, b\}$ and $\mathcal{L}_\Sigma$ as in Example 1.2.2. Let

$$Q = \mathsf{length\_at\_most}(p, 1) \vee (\mathsf{is\_more\_specific}(p, ab) \wedge \mathsf{is\_more\_general}(p, ab)).$$

When evaluated over $\mathcal{L}_\Sigma$, the first disjunct of Q gives the solution $\{\epsilon, a, b\}$, the second $\{ab\}$, so that $Th(Q, \mathcal{L}_\Sigma) = \{\epsilon, a, b, ab\}$, which is convex in $\mathcal{L}_\Sigma$. Thus, $\dim(Q) = 1$ (as Q does not reference any datasets, the maximization over $\mathcal{D}$ in the definition of dimension here is vacuous).

$Th(Q, \mathcal{L}_\Sigma)$ can be represented by $S(Q, \mathcal{L}_\Sigma) = \{ab\}$ and $G(Q, \mathcal{L}_\Sigma) = \{\epsilon\}$. □

With the following definitions and theorem we provide an alternative characterization of dimension k sets.

Definition 1.3.4 Let $I \subseteq \mathcal{L}$. Call a chain $\phi_1 \preceq \phi_2 \preceq \cdots \preceq \phi_{2k-1} \subseteq \mathcal{L}$ an *alternating chain (of length k) for I* if $\phi_i \in I$ for all odd i, and $\phi_i \notin I$ for all even i.

Definition 1.3.5 Let $I \subseteq \mathcal{L}$. We define two operators on I,

$$I^- = \{\phi \in I \mid \exists \psi \in \mathcal{L} \setminus I, \phi' \in I : \ \phi \preceq \psi \preceq \phi'\}$$
$$I^+ = I \setminus I^-.$$

Thus, I^- is constructed from I by removing all elements that only appear as the maximal element in alternating chains for I. I^+ is the set of such removed elements. Note that since $I^- \subset I$ by definition, we have $I = I^+ \cup I^-$ and $I^+ \cap I^- = \emptyset$.

Theorem 1.3.6 Let I be a generalized version space. Then $\dim(I)$ is equal to the maximal k for which there exists in $\mathcal{L}$ an alternating chain of length k for I.

Proof: By induction on k: if I only has alternating chains of length 1, then $I \in \mathcal{VS}^1$ and $\dim(I) = 1$ by definition. Assume, then, that $k \geq 2$ is the length of the longest alternating chain for I. As there are chains of length ≥ 2, both I^- and I^+ are nonempty.

It is clear from the definition of I^- that I^- has alternating $\mathcal{L}$-chains of length $k-1$, but not of length k. By induction hypothesis, thus $\dim(I^-) = k-1$. The set I^+, on the other hand, has dimension 1. It follows that $\dim(I = I^+ \cup I^-)$ is at most k. That $\dim(I)$ is at least k directly follows from the existence of an alternating chain $\phi_1 \preceq \phi_2, \preceq \cdots \preceq \phi_{2k-1}$ for I, because $\phi_1, \phi_3, \ldots, \phi_{2k-1}$ must belong to distinct components in every partition of I into convex components. □

The operator I^+ allows us to define a *canonical decomposition* of a generalized version space. For this, let I be a generalized version space of dimension k. Define

$$I_0 = I^+$$
$$I_i = (I \setminus I_0 \cup \cdots \cup I_{i-1})^+ \quad (1 \leq i \leq k)$$

The version spaces I_i then are convex, disjoint, and $I = \cup_{i=1}^k I_i$.

Our results so far relate to the structure of a fixed generalized version space. Next, we investigate the behavior of GVSs under set-theoretic operations. Hirsh [13] has shown that $\mathcal{VS}^1$ is closed under intersections, but not under unions. Our following results show that $\mathcal{VS}^{\mathbb{Z}}$ is closed under all set-theoretic operations, and that one obtains simple bounds on growth in dimension under such operations.

Theorem 1.3.7 Let $V \in \mathcal{VS}^{\mathbb{Z}}(\mathcal{L})$. Then $\mathcal{L} \setminus V \in \mathcal{VS}^{\mathbb{Z}}(\mathcal{L})$, and $\dim(V) - 1 \leq \dim(\mathcal{L} \setminus V) \leq \dim(V) + 1$.

Proof: Any alternating chain of length k for $\mathcal{L} \setminus V$ defines an alternating chain of length $k-1$ for $\mathcal{L}$. It follows that $\dim(\mathcal{L} \setminus V) \leq \dim(V) + 1$. By a symmetrical argument $\dim(V) \leq \dim(\mathcal{L} \setminus V) + 1$. □

By definition, generalized version spaces are closed under finite unions, and $\dim(V \cup W) \leq \dim(V) + \dim(W)$. Combining this with the dimension bound for complements, we obtain $\dim(V \cap W) = \dim((V^c \cup W^c)^c) \leq \dim(V) + \dim(W) + 3$. However, a somewhat tighter bound can be given:

Theorem 1.3.8 Let $V, W \in \mathcal{VS}^{\mathbb{Z}}(\mathcal{L})$. Then $\dim(V \cap W) \leq \dim(V) + \dim(W) - 1$.

Proof: Let $\phi_1 \preceq \phi_2, \preceq \cdots \preceq \phi_{2k-1}$ be an alternating chain for $V \cap W$. Let $I_V := \{i \in 1, \ldots k-1 \mid \phi_{2i} \notin V\}$, and $I_W := \{i \in 1, \ldots k-1 \mid \phi_{2i} \notin W\}$. Then $I_V \cup I_W = \{1, \ldots, k-1\}$. Deleting from the original alternating chain all ϕ_{2i}, ϕ_{2i-1} with $i \notin I_V$ gives an alternating chain of length $|I_V| + 1$ for V. Thus, $|I_V| \leq \dim(V) - 1$. Similarly, $|I_W| \leq \dim(W) - 1$. The theorem now follows with $|I_V| + |I_W| \geq k - 1$. □

1.4 Query Decomposition

In the previous section we have studied the structure of the solution sets $Th(Q, \mathcal{D}, \mathcal{L})$. We now turn to the question of how to develop strategies for the computation of solutions so that, first, the computations for complex Boolean queries can be reduced to computations of simple version spaces using standard level-wise algorithms, and second, the solutions obtained have a parsimonious representation in terms of the number of their convex components, and/or the total size of the boundaries needed to describe the convex components.

A first approach to solving a Boolean query Q using level-wise algorithms is to transform Q into disjunctive normal form (DNF). Each disjunct then will be a conjunction of monotonic or anti-monotonic predicates, and thus define a convex solution set. The solution to the query then is simply the union of the solutions of the disjuncts. This approach, however, will often not lead to a parsimonious representation: the number of disjuncts in Q's DNF can far exceed the dimension of Q, so that the solution is not minimal in terms of the number of convex components. The solutions of the different disjunctions also may have a substantial overlap, which can lead to a greatly enlarged size of a boundary representation.

In this section we introduce two alternative techniques for decomposing a Boolean query into one-dimensional sub-queries. The first approach is based on user-defined query plans which can improve the efficiency by a reduction to simple and easy to evaluate convex sub-queries. The second approach, which we call the canonical decomposition, is fully automated and guaranteed to lead to solutions given by convex components that are minimal in number, and non-overlapping.

1.4.1 Query plans

The solution set $Th(Q, \mathcal{D}, \mathcal{L})$ can be constructed incrementally from basic convex components using algebraic union, intersection and complementation operations. Using Theorems 1.3.7 and 1.3.8 one can bound the number of convex components needed to represent the final solution. For any given query, usually multiple such incremental computations are possible. A query plan in the sense of the following definition represents a particular solution strategy.

Definition 1.4.1 A query plan is a Boolean formula with some of its subqueries marked using the symbol $\underbrace{\quad}$. Furthermore, all marked subqueries are the conjunction of a monotonic and an anti-monotonic subquery.

Example 1.4.2 Consider the query

$$Q_1 = (\mathsf{a}_1 \vee \mathsf{a}_2) \wedge (\mathsf{m}_1 \vee \mathsf{m}_2).$$

Since this is a conjunction of a monotonic and an anti-monotonic part, it can be solved directly, and $\underbrace{(\mathsf{a}_1 \vee \mathsf{a}_2) \wedge (\mathsf{m}_1 \vee \mathsf{m}_2)}$ is the corresponding query plan.

A transformation of Q_1 into DNF gives

$$(\mathsf{a}_1 \wedge \mathsf{m}_1) \vee (\mathsf{a}_1 \wedge \mathsf{m}_2) \vee (\mathsf{a}_2 \wedge \mathsf{m}_1) \vee (\mathsf{a}_2 \wedge \mathsf{m}_2),$$

for which $\underbrace{(\mathsf{a}_1 \wedge \mathsf{m}_1)} \vee \underbrace{(\mathsf{a}_1 \wedge \mathsf{m}_2)} \vee \underbrace{(\mathsf{a}_2 \wedge \mathsf{m}_1)} \vee \underbrace{(\mathsf{a}_2 \wedge \mathsf{m}_2)}$ is the only feasible query plan, which now requires four calls to the basic inductive query solver. □

For any inductive query Q, we can rewrite it in many different forms. One can thus construct a variety of different query plans by annotating queries that are logically equivalent to Q. The question then arises as to which query plan is optimal, in the sense that the resources (i.e., memory and cpu-time) needed for computing its solution set are as small as possible. A general approach to this problem would involve the use of cost estimates that for each call to a conjunctive solver and operation. One example of a cost function for a call to a conjunctive solver could be *Expected Number of Scans of Data* $\times$ *Size of Data Set*. Another one could be the *Expected Number of Covers Tests*. In this chapter, we have studied the query optimization problem under the assumption that each call to a conjunctive solver has unit cost and that the only set operation allowed is union. Under this assumption, decomposing a query Q into k subqueries of the form $Q_{a,i} \wedge Q_{m,i}$ (with $Q_{a,i}$ anti-monotonic and $Q_{m,i}$ monotonic) and $\dim(Q) = k$ is an optimal strategy. We will leave open the challenging question as to which cost-estimates to use in practice. However, what should be clear is that given such cost-estimates, one could optimize inductive queries by constructing all possible query plans and then

selecting the best one. This is effectively an optimization problem, not unlike the query optimization problem in relational databases.

The optimization problem becomes even more interesting in the light of interactive querying sessions [4], which should be quite common when working with inductive databases. In such sessions, one typically submits a rough query to get some insight in the domain, and when the results of this query are available, the user studies the results and refines the query. This often goes through a few iterations until the desired results are obtained.

1.4.2 Canonical Decomposition

As in the simple DNF decomposition approach, Q will be decomposed into k sub-queries Q_i such that Q is equivalent to $Q_1 \vee \cdots \vee Q_k$, and each Q_i is convex. Furthermore, the Q_i will be mutually exclusive.

We develop this technique in two stages: in the first stage we do not take the concrete pattern language $\mathcal{L}$ into account, and determine Q_i such that $Th(Q, \mathcal{D}, \mathcal{L}) = \cup Th(Q_i, \mathcal{D}, \mathcal{L})$ for all $\mathcal{L}$ to which the predicates in Q can be applied. This step only uses the monotonicity of the predicates and the Boolean structure of Q. In a second step we refine the approach in order to utilize structural properties of $\mathcal{L}$ that can reduce the number of components Q_i needed to represent $Th(Q, \mathcal{D}, \mathcal{L})$.

Applied to the query from Example 1.3.3, for instance, the first step will result in a decomposition of Q into two components (essentially corresponding to the two disjuncts of Q), which yields a bound of 2 for the dimension of $Th(Q, \mathcal{D}, \mathcal{L})$ for all $\mathcal{L}$. The second step then is able to use properties of $\mathcal{L}_\Sigma$ in order to find the tighter bound 1 for the dimension of $Th(Q, \mathcal{L}_\Sigma)$.

The idea for both stages of the decomposition is to first evaluate Q in a reduced pattern language $\mathcal{L}'$, so that the desired partition $\vee Q_i$ can be derived from the structure of $Th(Q, \mathcal{L}')$. The solution set $Th(Q, \mathcal{L}')$ does not depend on the datasets $\mathcal{D}$ that Q references, and the complexity of its computation only depends on the size of Q, but not on the size of any datasets.

In the following we always assume that Q is a query that contains n distinct predicates $\mathsf{m}_1, \ldots, \mathsf{m}_n$, and that the m_i are monotonic for all pattern languages $\mathcal{L}$ for which Q can be evaluated (recall that we replace anti-monotonic predicates by negated monotonic ones).

Definition 1.4.3 Let $\mathcal{M}(Q) = \{\mathsf{m}_1, \ldots, \mathsf{m}_n\}$, $\mathcal{L}_{\mathcal{M}(Q)} = 2^{\mathcal{M}(Q)}$, and for $\mu \in \mathcal{L}_{\mathcal{M}(Q)}$:

$$\mu_e := \{M \subseteq \mathcal{M}(Q) \mid M \subseteq \mu\}.$$

The predicates m_i are interpreted over $\mathcal{L}_{\mathcal{M}(Q)}$ as

$$Th(\mathsf{m}_i, \mathcal{L}_{\mathcal{M}(Q)}) := \{\mu \in \mathcal{L}_{\mathcal{M}(Q)} \mid \mathsf{m}_i \in \mu\}.$$

Thus, the definitions of $\mathcal{L}_{\mathcal{M}(Q)}$ and μ_e are similar to the ones for itemsets with $\mathcal{M}(Q)$ the set of possible items (cf. Example 1.2.1). Alternatively, each $\mu \in \mathcal{L}_{\mathcal{M}(Q)}$ can be viewed as an interpretation for the propositional variables $\mathcal{M}(Q)$ (see also Section 1.5). However, the inclusion condition in the definition of μ_e here is the converse of the inclusion condition in Example 1.2.1. In particular, here, $\mu' \succeq \mu$ iff $\mu' \supseteq \mu$. The predicates m_i are interpreted with respect to $\mathcal{L}_{\mathcal{M}(Q)}$ like the predicates $\mathsf{is\_more\_general}(p, \{\mathsf{m}_i\})$. By the general definition, with $Th(\mathsf{m}_i, \mathcal{L}_{\mathcal{M}(Q)})$ $(1 \leq i \leq k)$ also $Th(Q, \mathcal{L}_{\mathcal{M}(Q)})$ is defined.

Theorem 1.4.4 Let $\mathcal{L}$ be a pattern language for which the predicates m_i in Q are monotone. The dimension of $Th(Q, \mathcal{D}, \mathcal{L})$ is less than or equal to the dimension of $Th(Q, \mathcal{L}_{\mathcal{M}(Q)})$.

Proof: Let $\mathcal{L}$ be given and $\mathcal{D}$ be any dataset. Define a mapping

$$\begin{aligned} h_{\mathcal{D}} : \mathcal{L} &\to \mathcal{L}_{\mathcal{M}(Q)} \\ \phi &\mapsto \{\mathsf{m} \in \mathcal{M}(Q) \mid \phi \in Th(\mathsf{m}, \mathcal{D}, \mathcal{L})\} \end{aligned} \tag{1.1}$$

First we observe that $h_{\mathcal{D}}$ is order preserving:

$$\phi \succeq \psi \quad \Rightarrow \quad h_{\mathcal{D}}(\phi) \succeq h_{\mathcal{D}}(\psi). \tag{1.2}$$

This follows from the monotonicity of the predicates m, because $\phi \succeq \psi$ and $\psi \in Th(\mathsf{m}, \mathcal{D}, \mathcal{L})$ implies $\phi \in Th(\mathsf{m}, \mathcal{D}, \mathcal{L})$, so that $h_{\mathcal{D}}(\phi)$ is a superset of $h_{\mathcal{D}}(\psi)$, which, in the pattern language $\mathcal{L}_{\mathcal{M}(Q)}$ just means $h_{\mathcal{D}}(\phi) \succeq h_{\mathcal{D}}(\psi)$.

Secondly, we observe that $h_{\mathcal{D}}$ preserves solution sets:

$$\phi \in Th(Q, \mathcal{D}, \mathcal{L}) \quad \Leftrightarrow \quad h_{\mathcal{D}}(\phi) \in Th(Q, \mathcal{L}_{\mathcal{M}(Q)}). \tag{1.3}$$

To see (1.3) one first verifies that for $i = 1, \ldots, n$:

$$\phi \in Th(\mathsf{m}_i, \mathcal{D}, \mathcal{L}) \quad \Leftrightarrow \quad \mathsf{m}_i \in h_{\mathcal{D}}(\phi) \quad \Leftrightarrow \quad h_{\mathcal{D}}(\phi) \in Th(\mathsf{m}_i, \mathcal{L}_{\mathcal{M}(Q)}).$$

Then (1.3) follows by induction on the structure of queries Q constructed from the m_i.

Now suppose that $\phi_1 \preceq \cdots \preceq \phi_{2k-1} \subseteq \mathcal{L}$ is an alternating chain of length k for $Th(Q, \mathcal{D}, \mathcal{L})$. From (1.2) it follows that $h_{\mathcal{D}}(\phi_1) \preceq \cdots \preceq h_{\mathcal{D}}(\phi_{2k-1}) \subseteq \mathcal{L}_{\mathcal{M}(Q)}$, and from (1.3) it follows that this is an alternating chain of length k for $Th(Q, \mathcal{L}_{\mathcal{M}(Q)})$. From Theorem 1.3.6 it now follows that the dimension of $Th(Q, \mathcal{D}, \mathcal{L})$ is at most the dimension of $Th(Q, \mathcal{L}_{\mathcal{M}(Q)})$. □

Example 1.4.5 Let $\Sigma = \{a, b, \ldots, z\}$. Let

$$\begin{aligned} \mathsf{m}_1 &= \mathsf{not\text{-}is\_more\_specific}(p, ab) \\ \mathsf{m}_2 &= \mathsf{not\text{-}is\_more\_specific}(p, cb) \\ \mathsf{m}_3 &= \mathsf{not\text{-}length\_at\_least}(p, 4) \\ \mathsf{m}_4 &= \mathsf{minimum\_frequency}(p, 3, D) \end{aligned}$$

These predicates are monotonic when interpreted in the natural way over pattern languages for the string domain. The first three predicates are the (monotonic) negations of the (anti-monotonic) standard predicates introduced in Section 1.2 (note that e.g., not-is_more_specific is distinct from is_more_general). Let

$$Q = \neg \mathsf{m}_1 \wedge \neg \mathsf{m}_2 \wedge (\neg \mathsf{m}_3 \vee \mathsf{m}_4). \tag{1.4}$$

Figure 1.2 (a) shows $\mathcal{L}_{\mathcal{M}(Q)}$ for this query. The solution set $Th(Q, \mathcal{L}_{\mathcal{M}(Q)})$ is $\{\emptyset, \{\mathsf{m}_4\}, \{\mathsf{m}_3, \mathsf{m}_4\}\}$, which is of dimension 2, because $\emptyset \preceq \{\mathsf{m}_3\} \preceq \{\mathsf{m}_3, \mathsf{m}_4\}$ is a (maximal) alternating chain of length 2. □

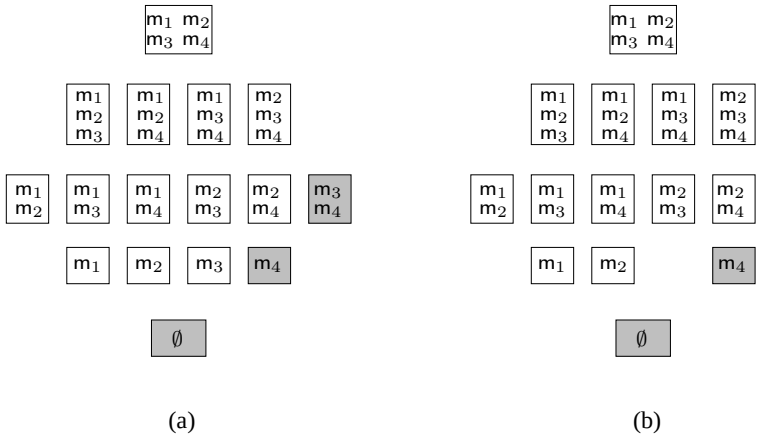

Fig. 1.2 Pattern languages $\mathcal{L}_{\mathcal{M}(Q)}$ and $\mathcal{L}_{\mathcal{M}(Q),\mathcal{L}}$

Given Q we can construct $\mathcal{L}_{\mathcal{M}(Q)}$ and $Th(Q, \mathcal{L}_{\mathcal{M}(Q)})$ in time $O(2^n)$. We can then partition $Th(Q, \mathcal{L}_{\mathcal{M}(Q)})$ into a minimal number of convex components $I_1, \ldots, I_k$ by iteratively removing from $Th(Q, \mathcal{L}_{\mathcal{M}(Q)})$ elements that are maximal in alternating chains. More precisely, let:

$$\begin{array}{ll} R_0 & := Th(Q, \mathcal{L}_{\mathcal{M}(Q)}) \\ I_{i+1} & := R_i^+ \\ R_{i+1} & := R_i^- \end{array} \tag{1.5}$$

for $i = 1, 2, \ldots, k$. The I_h are defined by their sets of maximal and minimal elements, $G(I_h)$ and $S(I_h)$, and define queries Q_h with the desired properties:

Theorem 1.4.6 Let $Th(Q, \mathcal{L}_{\mathcal{M}(Q)}) = I_1 \cup \cdots \cup I_k$ with convex I_h.

Given an element μ from $\mathcal{L}_{\mathcal{M}(Q)}$ define

$$Q_{M,\mu} = \bigwedge_{\mathsf{m} \in \mu} \mathsf{m} \quad \text{and} \quad Q_{A,\mu} = \bigwedge_{\mathsf{m} \notin \mu} \neg \mathsf{m}$$

For $h = 1, \ldots, k$ let

$$Q_{h,M} = \bigvee_{\mu \in S(I_h)} Q_{M,\mu} \quad \text{and} \quad Q_{h,A} = \bigvee_{\mu \in G(I_h)} Q_{A,\mu}.$$

Finally, let $Q_h = Q_{h,M} \wedge Q_{h,A}$.

Then $Th(Q_h, \mathcal{D}, \mathcal{L})$ is convex for all $\mathcal{L}$ and $\mathcal{D}$, and $Th(Q, \mathcal{D}, \mathcal{L}) = Th(\vee_{h=1}^{k} Q_h, \mathcal{D}, \mathcal{L}) = \cup_{h=1}^{k} Th(Q_h, \mathcal{D}, \mathcal{L})$.

Proof: The Q_h are constructed so that

$$Th(Q_h, \mathcal{L}_{\mathcal{M}(Q)}) = I_h \quad (h = 1, \ldots, k).$$

Using the embedding $h_{\mathcal{D}}$ from the proof of Theorem 1.4.4 we then obtain for $\phi \in \mathcal{L}$:

$$\begin{aligned} \phi \in Th(Q, \mathcal{D}, \mathcal{L}) &\Leftrightarrow h_{\mathcal{D}}(\phi) \in Th(Q, \mathcal{L}_{\mathcal{M}(Q)}) \\ &\Leftrightarrow h_{\mathcal{D}}(\phi) \in Th(\vee_{h=1}^{k} Q_h, \mathcal{L}_{\mathcal{M}(Q)}) \\ &\Leftrightarrow \phi \in Th(\vee_{h=1}^{k} Q_h, \mathcal{D}, \mathcal{L}) \\ &\Leftrightarrow \phi \in \cup_{h=1}^{k} Th(Q_h, \mathcal{D}, \mathcal{L}) \end{aligned}$$

□

Example 1.4.7 (continued from Example 1.4.5) Using (1.5) we obtain the partition of $Th(Q, \mathcal{L}_{\mathcal{M}(Q)})$

$$I_1 = \{\{\mathsf{m}_4\}, \{\mathsf{m}_3, \mathsf{m}_4\}\}, \; I_2 = \{\emptyset\},$$

so that

$$\begin{array}{ll} G(I_1) = \{\{\mathsf{m}_3, \mathsf{m}_4\}\} & S(I_1) = \{\{\mathsf{m}_4\}\} \\ G(I_2) = \{\emptyset\} & S(I_2) = \{\emptyset\}. \end{array}$$

These boundary sets define the queries

$$\begin{aligned} Q_1 &= (\neg \mathsf{m}_1 \wedge \neg \mathsf{m}_2) \wedge \mathsf{m}_4 \\ Q_2 &= (\neg \mathsf{m}_1 \wedge \neg \mathsf{m}_2 \wedge \neg \mathsf{m}_3 \wedge \neg \mathsf{m}_4) \end{aligned}$$

When we view Q, Q_1, Q_2 as propositional formulas over propositional variables $\mathsf{m}_1, \ldots, \mathsf{m}_4$, we see that $Q \leftrightarrow Q_1 \vee Q_2$ is a valid logical equivalence. In fact, one can interpret the whole decomposition procedure we here developed as a method for computing a certain normal form for propositional formulas. We investigate this perspective further in Section 1.5. □

Example 1.4.8 (continued from Example 1.4.2) Introducing $\tilde{\mathsf{m}}_1 = \neg \mathsf{a}_1$, $\tilde{\mathsf{m}}_2 = \neg \mathsf{a}_2$, we can express query Q_1 from Example 1.4.2 using monotone predicates only as

$$\tilde{Q}_1 = (\neg \tilde{\mathsf{m}}_1 \vee \neg \tilde{\mathsf{m}}_2) \wedge (\mathsf{m}_1 \vee \mathsf{m}_2)$$

The boundary sets of the single convex component I of $\tilde{Q}_1$ are

$$G(I) = \{\{\tilde{\mathsf{m}}_1, \mathsf{m}_1, \mathsf{m}_2\}, \{\tilde{\mathsf{m}}_2, \mathsf{m}_1, \mathsf{m}_2\}\} \quad S(I) = \{\{\mathsf{m}_1\}, \{\mathsf{m}_2\}\}.$$

The query construction of Theorem 1.4.6 yields the sub-queries $Q_A = \neg\tilde{\mathsf{m}}_1 \vee \neg\tilde{\mathsf{m}}_2$ and $Q_M = \mathsf{m}_1 \vee \mathsf{m}_2$, i.e., the same decomposition as given by the query plan in Example 1.4.2.

Now consider the query

$$Q' = \mathsf{m}_1 \vee (\mathsf{m}_2 \wedge \neg\mathsf{m}_3).$$

This query has dimension two, and can be solved, for example, using the query plan $\underbrace{\mathsf{m}_1} \vee \underbrace{(\mathsf{m}_2 \wedge \neg\mathsf{m}_3)}$.

The canonical decomposition gives the following boundary sets for two convex components:

$$\begin{array}{ll} G(I_1) = \{\{\mathsf{m}_1, \mathsf{m}_2, \mathsf{m}_3\}\} & S(I_1) = \{\{\mathsf{m}_1\}\} \\ G(I_2) = \{\{\mathsf{m}_2\}\} & S(I_2) = \{\{\mathsf{m}_2\}\}, \end{array}$$

which leads to the sub-queries

$$Q'_1 = \mathsf{m}_1, \quad Q'_2 = \neg\mathsf{m}_1 \wedge \neg\mathsf{m}_3 \wedge \mathsf{m}_2.$$

Thus, the we obtain a different solution strategy than from the simple query plan, and the solution will be expressed as two disjoint convex components, whereas the query plan would return overlapping convex components. □

When we evaluate the query Q from Example 1.4.5 for $\mathcal{L}_\Sigma$, we find that $Th(Q, \mathcal{D}, \mathcal{L}_\Sigma)$ actually has dimension 1. The basic reason for this is that in $\mathcal{L}_\Sigma$ there does not exist any pattern that satisfies $\neg\mathsf{m}_1 \wedge \neg\mathsf{m}_2 \wedge \mathsf{m}_3 \wedge \neg\mathsf{m}_4$, i.e., that corresponds to the pattern $\{\mathsf{m}_3\} \in \mathcal{L}_{\mathcal{M}(Q)}$ that is the only "witness" for the non-convexity of $Th(Q, \mathcal{L}_{\mathcal{M}(Q)})$. To distinguish patterns in $\mathcal{L}_{\mathcal{M}(Q)}$ that we need not take into account when working with a pattern language $\mathcal{L}$, we introduce the concept of $\mathcal{L}$-admissibility:

Definition 1.4.9 Let $\mathcal{L}$ be a pattern language. A pattern $\mu \in \mathcal{L}_{\mathcal{M}(Q)}$ is called $\mathcal{L}$-*admissible* if there exists $\phi \in \mathcal{L}$ and datasets $\mathcal{D}$ such that $\mu = h_\mathcal{D}(\phi)$, where $h_\mathcal{D}$ is as defined by (1.1). Let $\mathcal{L}_{\mathcal{M}(Q),\mathcal{L}} \subseteq \mathcal{L}_{\mathcal{M}(Q)}$ be the language of $\mathcal{L}$-admissible patterns from $\mathcal{L}_{\mathcal{M}(Q)}$. As before, we define

$$Th(\mathsf{m}_i, \mathcal{L}_{\mathcal{M}(Q),\mathcal{L}}) = \{\mu \in \mathcal{L}_{\mathcal{M}(Q),\mathcal{L}} \mid \mathsf{m}_i \in \mu\}.$$

An alternative characterization of admissibility is that μ is $\mathcal{L}$-admissible if there exists $\mathcal{D}$ such that

$$Th(\bigwedge_{\mathsf{m}_i \in \mu} \mathsf{m}_i \wedge \bigwedge_{\mathsf{m}_j \notin \mu} \neg\mathsf{m}_j, \mathcal{D}, \mathcal{L}) \neq \emptyset.$$

Theorem 1.4.10 Let $\mathcal{L}$ and Q be as in Theorem 1.4.4. The dimension of $Th(Q, \mathcal{D}, \mathcal{L})$ is less than or equal to the dimension of $Th(Q, \mathcal{L}_{\mathcal{M}(Q),\mathcal{L}})$.

Proof: The proof is as for Theorem 1.4.4, by replacing $\mathcal{L}_{\mathcal{M}(Q)}$ with $\mathcal{L}_{\mathcal{M}(Q),\mathcal{L}}$ throughout. We only need to note that according to the definition of admissibility, the mapping $h_{\mathcal{D}}$ defined by (1.1) actually maps $\mathcal{L}$ into $\mathcal{L}_{\mathcal{M}(Q),\mathcal{L}}$, so that $h_{\mathcal{D}}$ still is well-defined. □

Example 1.4.11 (continued from Example 1.4.7) Consider the language $\mathcal{L}_\Sigma$. Of the patterns in $\mathcal{L}_{\mathcal{M}(Q)}$ two are not $\mathcal{L}_\Sigma$-admissible: as $\mathsf{is\_more\_specific}(p, ab) \wedge \mathsf{is\_more\_specific}(p, cb)$ implies $\mathsf{length\_at\_least}(p, 4)$, we have that $Th(\neg \mathsf{m}_1 \wedge \neg \mathsf{m}_2 \wedge \mathsf{m}_3 \wedge \mathsf{m}_4, \mathcal{D}, \mathcal{L}_\Sigma) = Th(\neg \mathsf{m}_1 \wedge \neg \mathsf{m}_2 \wedge \mathsf{m}_3 \wedge \neg \mathsf{m}_4, \mathcal{D}, \mathcal{L}_\Sigma) = \emptyset$ for all $\mathcal{D}$, so that the two patterns $\{\mathsf{m}_3\}$ and $\{\mathsf{m}_3, \mathsf{m}_4\}$ from $\mathcal{L}_{\mathcal{M}(Q)}$ are not $\mathcal{L}_\Sigma$-admissible.

Figure 1.2 (b) shows $\mathcal{L}_{\mathcal{M}(Q),\mathcal{L}_\Sigma}$. Now $Th(Q, \mathcal{L}_{\mathcal{M}(Q),\mathcal{L}_\Sigma}) = \{\emptyset, \{\mathsf{m}_4\}\}$ has dimension 1, so that by Theorem 1.4.10 $Th(Q, \mathcal{D}, \mathcal{L}_\Sigma)$ also has dimension 1.

With Theorem 1.4.6 we obtain the query

$$Q_1 = \neg \mathsf{m}_1 \wedge \neg \mathsf{m}_2 \wedge \neg \mathsf{m}_3$$

with $Th(Q, \mathcal{D}, \mathcal{L}_\Sigma) = Th(Q_1, \mathcal{D}, \mathcal{L}_\Sigma)$.

□

Table 1.1 The schema of the Query Decomposition Approach

Input:
• Query Q that is a Boolean combination of monotone predicates: $\mathcal{M}(Q) = \{\mathsf{m}_1, \ldots, \mathsf{m}_n\}$ • datasets $\mathcal{D}$ • pattern language $\mathcal{L}$
Step 1: Construct $\mathcal{L}_{\mathcal{M}(Q),\mathcal{L}}$: for each $\mu \subseteq \mathcal{M}(Q)$ decide whether μ is $\mathcal{L}$-admissible.
Step 2: Construct $Th(Q, \mathcal{L}_{\mathcal{M}(Q),\mathcal{L}})$: for each $\mu \in \mathcal{L}_{\mathcal{M}(Q),\mathcal{L}}$ decide whether $\mu \in Th(Q, \mathcal{L}_{\mathcal{M}(Q),\mathcal{L}})$.
Step 3: Determine convex components: compute partition $Th(Q, \mathcal{L}_{\mathcal{M}(Q),\mathcal{L}}) = I_1 \cup \cdots \cup I_k$ into a minimal number of convex components.
Step 4: Decompose Q: compute queries $Q_1, \ldots, Q_k$.
Step 5: For $i = 1, \ldots, k$ determine $Th(Q_i, \mathcal{D}, \mathcal{L})$ by computing the boundaries $G(Q_i, \mathcal{D}, \mathcal{L})$ and $S(Q_i, \mathcal{D}, \mathcal{L})$.

Table 1.1 summarizes the decomposition approach to inductive query evaluation as derived from Theorems 1.4.10 and 1.4.6. A simplified procedure based on Theorems 1.4.4 and 1.4.6 can be used simply by omitting the first step.

Assuming that $\mathcal{L}$-admissibility is decidable in time exponential in the size s of the query (for the pattern languages and pattern predicates we have considered so far this will be the case), we obtain that steps 1–4 can be performed naively in time $O(2^s)$. This exponential complexity in s we consider

uncritical, as the size of the query will typically be very small in comparison to the size of $\mathcal{D}$, i.e., $s \ll |\mathcal{D}|$, so that the time critical step is step 5, which is the only step that requires inspection of $\mathcal{D}$.

Theorem 1.4.10 is stronger than Theorem 1.4.4 in the sense that for given $\mathcal{L}$ it yields better bounds for the dimension of $Th(Q, \mathcal{D}, \mathcal{L})$. However, Theorem 1.4.4 is stronger than Theorem 1.4.10 in that it provides a uniform bound for all pattern languages for which the m_i are monotone. For this reason, the computation of $Th(Q, \mathcal{D}, \mathcal{L})$ using the simpler approach given by Theorems 1.4.4 and 1.4.6 can also be of interest in the case where we work in the context of a fixed language $\mathcal{L}$, because the solutions computed under this approach are more robust in the following sense: suppose we have computed $Th(Q, \mathcal{D}, \mathcal{L})$ using the decomposition provided by Theorems 1.4.4 and 1.4.6, i.e., by the algorithm shown in Table 1.1 omitting step 1. This gives us a representation of $Th(Q, \mathcal{D}, \mathcal{L})$ by boundary sets $G(Q_h, \mathcal{D}, \mathcal{L}), S(Q_h, \mathcal{D}, \mathcal{L})$. If we now consider any refinement $\mathcal{L}' \succeq \mathcal{L}$, then our boundary sets still define valid solutions of Q in $\mathcal{L}'$, i.e., for all $\psi \in \mathcal{L}'$, if $\phi \preceq \psi \preceq \phi'$ for some $\phi \in S(Q_h, \mathcal{D}, \mathcal{L}), \phi' \in G(Q_h, \mathcal{D}, \mathcal{L})$, then $\psi \in Th(Q, \mathcal{D}, \mathcal{L}')$ (however, the old boundary may not completely define $Th(Q, \mathcal{D}, \mathcal{L}')$, as the maximal/minimal solutions of Q_h in $\mathcal{L}$ need not be maximal/minimal in $\mathcal{L}'$). A similar preservation property does not hold when we compute $Th(Q, \mathcal{D}, \mathcal{L})$ according to Theorem 1.4.10.

1.5 Normal forms

In this section we analyze some aspects of our query decomposition approach from a propositional logic perspective. Central to this investigation is the following concept of certain syntactic normal forms of propositional formulas.

Definition 1.5.1 Let Q be a propositional formula in propositional variables $\mathsf{m}_1, \ldots, \mathsf{m}_n$. We say that Q belongs to the class Θ_1 if Q is logically equivalent to a formula of the form

$$\left(\bigvee_{i=1}^{h} M_i\right) \wedge \left(\bigvee_{j=1}^{k} A_j\right), \tag{1.6}$$

where M_i's are conjunctions of positive atoms and A_j's are conjunctions of negative atoms. We say that Q belongs to the class Θ_k if Q is equivalent to the disjunction of k formulas from Θ_1.

The formulas Q_h defined in Theorem 1.4.6 are in the class Θ_1 when read as propositional formulas in $\mathsf{m}_1, \ldots, \mathsf{m}_n$, and were constructed so as to define convex sets. The following theorem provides a general statement on the relation between the Θ_k-normal form of a query and its dimension. As such,

every formula belongs to some Θ_k, as can easily be seen from the disjunctive normal forms.

Theorem 1.5.2 Let Q be a query containing pattern predicates $\mathsf{m}_1, \ldots, \mathsf{m}_n$. The following are equivalent:

(i) When interpreted as a Boolean formula over propositional variables $\mathsf{m}_1, \ldots, \mathsf{m}_n$, Q belongs to Θ_k.
(ii) The dimension of Q with respect to any pattern language $\mathcal{L}$ for which $\mathsf{m}_1, \ldots, \mathsf{m}_n$ are monotone is at most k.

Proof: (i)⇒(ii): We may assume that Q is written in Θ_k-normal form, i.e., as a disjunction of k subformulas of the form (1.6). As both unions and intersections of monotone sets are monotone, we obtain that the left conjunct of (1.6) defines a monotone subset of $\mathcal{L}$ (provided the m_i define monotone sets in $\mathcal{L}$). Similarly, the right conjunct defines an anti-monotone set. Their conjunction, then, defines a convex set, and the disjunction of k formulas (1.6) defines a union of k convex sets.

(ii)⇒(i): This follows from the proofs of Theorems 1.4.4 and 1.4.6: let $\mathcal{L} = \mathcal{L}_{\mathcal{M}(Q)}$. We can view $\mathcal{L}_{\mathcal{M}(Q)}$ as the set of all truth assignments to the variables m_i by letting for $\mu \in \mathcal{L}_{\mathcal{M}(Q)}$:

$$\mu: \quad \mathsf{m}_i \ \mapsto \begin{cases} \mathit{true} & \mathsf{m}_i \in \mu \\ \mathit{false} & \mathsf{m}_i \notin \mu \end{cases}$$

Then for all μ and all Boolean formulas $\tilde{Q}$ in $\mathsf{m}_1, \ldots, \mathsf{m}_n$:

$$\mu \in \mathit{Th}(\tilde{Q}, \mathcal{L}_{\mathcal{M}(Q)}) \ \Leftrightarrow \ \mu: \ \tilde{Q} \mapsto \ \mathit{true}.$$

Therefore

$$\begin{aligned} \mu: \ Q \mapsto \ \mathit{true} \ &\Leftrightarrow \ \mu \in \mathit{Th}(Q, \mathcal{L}_{\mathcal{M}(Q)}) \\ &\Leftrightarrow \ \mu \in \cup_{h=1}^{k} \mathit{Th}(Q_h, \mathcal{L}_{\mathcal{M}(Q)}) \\ &\Leftrightarrow \ \mu: \ \vee_{h=1}^{k} Q_h \mapsto \ \mathit{true} \end{aligned}$$

□

In the light of Theorem 1.5.2 we can interpret the decomposition procedure described by Theorems 1.4.4 and 1.4.6 as a Boolean transformation of Q into Θ_k-normal form. This transformation takes a rather greedy approach by explicitly constructing the exponentially many possible truth assignments to the propositional variables in Q. It might seem possible to find a more efficient transformation based on purely syntactic manipulations of Q. The following result shows that this is unlikely to succeed.

Theorem 1.5.3 The problem of deciding whether a given propositional formula Q belongs to Θ_1 is co-NP complete.

Proof: The class Θ_1 is in co-NP: let Q be a formula in variables $\mathsf{m}_1, \ldots, \mathsf{m}_n$. From Theorems 1.4.4 and 1.5.2 we know that $Q \notin \Theta_1$ iff the dimension of $Th(Q, \mathcal{L}_{\mathcal{M}(Q)})$ is at least 2. This, in turn, is equivalent to the existence of an alternating chain of length 2 for $Th(Q, \mathcal{L}_{\mathcal{M}(Q)})$. The existence of such a chain can be determined in nondeterministic polynomial time by guessing three elements $\mu_1, \mu_2, \mu_3 \in \mathcal{L}_{\mathcal{M}(Q)}$, and checking whether $\mu_1 \succeq \mu_2 \succeq \mu_3$ and $\mu_1, \mu_3 \in Th(Q, \mathcal{L}_{\mathcal{M}(Q)})$ and $\mu_2 \notin Th(Q, \mathcal{L}_{\mathcal{M}(Q)})$.

To show co-NP hardness we reduce the satisfiability problem to the complement of Θ_1. For this let F be a propositional formula in propositional variables $\mathsf{m}_1, \ldots, \mathsf{m}_k$. Define

$$Q := (F \wedge \neg \mathsf{x}_1 \wedge \neg \mathsf{y}_1) \vee (\mathsf{m}_1 \wedge \mathsf{m}_2 \wedge \cdots \wedge \mathsf{m}_k \wedge \mathsf{x}_1 \wedge \mathsf{y}_1),$$

(where $\mathsf{x}_1, \mathsf{y}_1$ are new propositional variables). Then $Th(Q, \mathcal{L}_{\mathcal{M}(Q)})$ has dimension ≥ 2 (i.e., $Q \notin \Theta_1$) iff F is satisfiable: If F is not satisfiable, then $(F \wedge \neg \mathsf{x}_1 \wedge \neg \mathsf{y}_1)$ is not satisfiable. So, Q can only be satisfied when all variables m_i, x_1, y_1 are true. Consequently, $Th(Q, \mathcal{L}_{\mathcal{M}(Q)})$ has only one element, namely $\{\mathsf{m}_1, \ldots, \mathsf{m}_k, \mathsf{x}_1, \mathsf{y}_1\}$ and $\dim(Th(Q, \mathcal{L}_{\mathcal{M}(Q)})) = 1$. On the other hand, if F is satisfiable, then $Th(Q, \mathcal{L}_{\mathcal{M}(Q)})$ contains a set $\phi \subseteq \{\mathsf{m}_1, \ldots, \mathsf{m}_k\}$, and then $\phi \subseteq \phi \cup \{\mathsf{x}_1\} \subseteq \{\mathsf{m}_1, \ldots, \mathsf{m}_k, \mathsf{x}_1, \mathsf{y}_1\}$ is an alternating chain of length 2, because $\phi \cup \{\mathsf{x}_1\} \notin Th(Q, \mathcal{L}_{\mathcal{M}(Q)})$.

□

The sub-queries to which the original query Q is reduced not only are known to have convex solution sets $Th(Q_h, \mathcal{D}, \mathcal{L})$, they also are of a special syntactic form $Q_h = Q_{h,M} \wedge Q_{h,A}$, where $Q_{h,M}$ defines a monotone set $Th(Q_{h,M}, \mathcal{D}, \mathcal{L})$, and $Q_{h,A}$ defines an anti-monotone set $Th(Q_{h,A}, \mathcal{D}, \mathcal{L})$. This factorization of Q_h facilitates the computation of the border sets $G(Q_h, \mathcal{D}, \mathcal{L})$ and $S(Q_h, \mathcal{D}, \mathcal{L})$, for which the level wise version space algorithm [8, 15] can be used.

1.6 Conclusions

We have described an approach to inductive querying, which generalizes both the pattern discovery problem in data mining and the concept-learning problem in machine learning. The method is based on the decomposition of the answer set to a collection of components defined by monotonic and anti-monotonic predicates. Each of the components is a convex set or version space, the borders of which can be computed using, for instance, the level wise version space algorithm or—for the pattern domain of strings—using the VSTmine algorithm [17], which employs a data structure called the version space tree.

The work presented is related to several research streams within data mining and machine learning. In machine learning, there has been an interest in version spaces ever since Tom Mitchell's seminal Ph.D. thesis [21]. The key complaint about standard version spaces was for a long time that it only allowed one to cope with essentially conjuctive concept-learning, that is, the induced concepts or patterns need to be conjuctive (which holds also for item sets). There has been quite some work in the machine learning literature on accomodating also disjunctive concepts (as is required in a general rule- or concept-learning setting), for instance, [26, 27]. While such disjunctive version space techniques sometimes also work with multiple borders set and version spaces, the way that this is realized differs from our approach. Indeed, in a disjunctive version space, a single solution consists of a disjunction of patterns, of which each pattern must belong to a single version space in the traditional sense. This differs from our approach in which each member of a single version space is a solution in itself. Algebraic properties of version spaces have also been investigated in the machine learning literature by, for instance, Haym Hirsh [11] who has investigated the properties of set theoretic operations on version spaces, and [16] who have developed a special version space algebra to represent functions, and used it for programming by demonstration.

In data mining, the structure on the search space was already exploited by early algorithms for finding frequent itemsets and association rules [1] leading soon to the concept of border sets [20]. Whereas initially the focus was on the use of the most specific borders, this was soon extended towards using also the most general borders to cope with multiple data sets, with conjunctive inductive queries or emerging patterns [8, 19]. The resulting version space structure was further analyzed by, for instance, Bucila et al. [6] for the case of itemsets. Each of these developments has resulted in new algorithms and techniques for finding solutions to increasingly complex inductive queries. The contribution of our work is that it has generalized this line of ideas in accomodating also non-convex solution sets, that is, generalized version spaces. It also allows one to cope with arbitrary Boolean inductive queries. Finally, as already mentioned in the introduction, the present paper also attempts to bridge the gap between the machine learning and data mining perspective both at the task level – through the introduction of Boolean inductive querying – and at the representation level – through the introduction of generalized version spaces to represent solution sets.

The results we have presented in this chapter are by no means complete, a lot of open problems and questions remain. A first and perhaps most important question is as to an experimental evaluation of the approach. Although some initial results in this direction have been presented in [17, 25, 18], the results are not yet conclusive and a deeper and more thorough evaluation is needed. A second question is concerned with further extending the framework to cope with other primitives, which are neither monotonic nor anti-monotonic. A typical example of such primitives are the questions that ask

for the top-k patterns w.r.t. a particular optimization function such as χ^2. This is known as correlated pattern mining [23]. A third question is how to perform a more quantitative query optimization, which would estimate the resources needed to execute particular query plan.

Although there are many remaining questions, the authors hope that the introduced framework provides a sound theory for studying these open questions.

Acknowledgements

This work was partly supported by the European IST FET project cInQ.

References

1. R. Agrawal, H. Mannila, R. Srikant, H. Toivonen, and A.I. Verkamo. Fast discovery of association rules. In U. Fayyad, G. Piatetsky-Shapiro, P. Smyth, and R. Uthurusamy, editors, *Advances in Knowledge Discovery and Data Mining*, pages 307–328. MIT Press, 1996.
2. R. Agrawal, T. Imielinski, and A. N. Swami. Mining association rules between sets of items in large databases. In Peter Buneman and Sushil Jajodia, editors, *Proceedings of the 1993 ACM SIGMOD International Conference on Management of Data*, pages 207–216, Washington, D.C., U.S.A., 25–28 May 1993.
3. D. Angluin. Queries and concept-learning. *Machine Learning*, 2:319–342, 1987.
4. E. Baralis and G. Psaila. Incremental refinement of mining queries. In M. K. Mohania and A. Min Tjoa, editors, *Proceedings of the First International Conference on Data Warehousing and Knowledge Discovery (DaWaK'99)*, volume 1676 of *Lecture Notes in Computer Science*, pages 173–182, Florence, Italy, August 30–September 1 1999. Springer.
5. R. Bayardo. Efficiently mining long patterns from databases. In *Proceedings of ACM SIGMOD Conference on Management of Data*, 1998.
6. C. Bucilă, J. Gehrke, D. Kifer, and W. White. Dualminer: A dual-pruning algorithm for itemsets with constraints. *Data Min. Knowl. Discov.*, 7(3):241–272, 2003.
7. L. De Raedt. A perspective on inductive databases. *SIGKDD Explorations: Newsletter of the Special Interest Group on Knowledge Discovery and Data Mining, ACM*, 4(2):69–77, January 2003.
8. L. De Raedt and S. Kramer. The levelwise version space algorithm and its application to molecular fragment finding. In *IJCAI01: Seventeenth International Joint Conference on Artificial Intelligence*, August 4–10 2001.
9. B. Goethals and J. Van den Bussche. On supporting interactive association rule mining. In *Proceedings of the Second International Conference on Data Warehousing and Knowledge Discovery*, volume 1874 of *Lecture Notes in Computer Science*, pages 307–316. Springer, 2000.
10. J. Han, L. V. S. Lakshmanan, and R. T. Ng. Constraint-based multidimensional data mining. *IEEE Computer*, 32(8):46–50, 1999.
11. H. Hirsh. *Incremental Version-Space Merging: A General Framework for Concept Learning*. Kluwer Academic Publishers, 1990.

12. H. Hirsh. Theoretical underpinnings of version spaces. In *Proceedings of the Twelfth International Joint Conference on Artificial Intelligence (IJCAI91)*, pages 665–670. Morgan Kaufmann Publishers, 1991.
13. H. Hirsh. Generalizing version spaces. *Machine Learning*, 17(1):5–46, 1994.
14. M. Kearns and U. Vazirani. *An Introduction to Computational Learning Theory*. MIT, 1994.
15. S. Kramer, L. De Raedt, and C. Helma. Molecular feature mining in HIV data. In *KDD-2001: The Seventh ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*. Association for Computing Machinery, August 26–29 2001. ISBN: 158113391X.
16. T. Lau, S. A. Wolfman, P. Domingos, and D. S. Weld. Programming by demonstration using version space algebra. *Mach. Learn.*, 53(1-2):111–156, 2003.
17. S. D. Lee. *Constrained Mining of Patterns in Large Databases*. PhD thesis, Albert-Ludwigs-University, 2006.
18. S. D. Lee and L. De Raedt. An algebra for inductive query evaluation. In X. Wu, A. Tuzhilin, and J. Shavlik, editors, *Proceedings of The Third IEEE International Conference on Data Mining (ICDM'03)*, pages 147–154, Melbourne, Florida, USA, November 19–22 2003. Sponsored by the IEEE Computer Society.
19. J. Li, K. Ramamohanarao, and G. Dong. The space of jumping emerging patterns and its incremental maintenance algorithms. In *ICML '00: Proceedings of the Seventeenth International Conference on Machine Learning*, pages 551–558, San Francisco, CA, USA, 2000. Morgan Kaufmann Publishers Inc.
20. H. Mannila and H. Toivonen. Levelwise search and borders of theories in knowledge discovery. *Data Mining and Knowledge Discovery*, 1(3):241–258, 1997.
21. T.M. Mitchell. *Version Spaces: An Approach to Concept Learning*. PhD thesis, Stanford University, 1978.
22. T.M. Mitchell. Generalization as search. *Artificial Intelligence*, 18(2):203–226, 1980.
23. S. Morishita and J. Sese. Traversing itemset lattice with statistical metric pruning. In *Proceedings of the 19th ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems*, pages 226–236. ACM Press, 2000.
24. R. T. Ng, L. V. S. Lakshmanan, J. Han, and A. Pang. Exploratory mining and pruning optimizations of constrained associations rules. In *Proceedings ACM-SIGMOD Conference on Management of Data*, pages 13–24, 1998.
25. L. De Raedt, M. Jaeger, S. D. Lee, and H. Mannila. A theory of inductive query answering. In *Proceedings of the 2002 IEEE International Conference on Data Mining*, pages 123–130, 2002.
26. G. Sablon, L. De Raedt, and M. Bruynooghe. Iterative versionspaces. *Artificial Intelligence*, 69:393–409, 1994.
27. M. Sebag. Delaying the choice of bias: A disjunctive version space approach. In *Proceedings of the 13th International Conference on Machine Learning*, pages 444–452, 1996.