Expert-Driven Validation of Set-Based Data Mining Results

by

Gediminas Adomavicius

A dissertation submitted in partial fulfillment of the requirements for the degree

of Doctor of Philosophy

Department of Computer Science

New York University

September 2002

Alexander Tuzhilin

Ernest Davis

To my parents, and to Laima

Acknowledgments

First and foremost, I would like to thank Prof. Alexander Tuzhilin, my advisor, for his guidance and support throughout these years.

I owe much gratitude to the Courant Institute of Mathematical Sciences and to the Computer Science Department. Special thanks to Prof. Marsha Berger, Prof. Richard Cole, Prof. Ernest Davis, Prof. Zvi Kedem, Prof. Alan Siegel, and Anina Karmen, our graduate division coordinator.

Many thanks to my friends and fellow students Fangzhe Chang, Hseu-Ming Chen, Raoul-Sam Daruwala, Deepak Goyal, Allen Leung, Ninghui Li, Niranjan Nilakantan, Toto Paxia, Archisman Rudra, Jiae Shin, David Tanzer, Marc Waldman, Zhe Yang, and Roman Yangarber for making my graduate school years enjoyable and memorable.

I would also like to thank my parents and my sister Dalia for always having faith in me.

Most of all, I would like to thank my wife Laima and my children for their unconditional love, encouragement, support, and for making it all possible.

Doctoral Dissertation Abstract

This dissertation addresses the problem of dealing with large numbers of set-based patterns, such as association rules and itemsets, discovered by data mining algorithms. Since many discovered patterns may be spurious, irrelevant, or trivial, one of the main problems is how to validate them, e.g., how to separate the "good" rules from the "bad." Many researchers have advocated the explicit involvement of a human expert in the validation process. However, scalability becomes an issue when large numbers of patterns are discovered, since the expert cannot perform the validation on a pattern-by-pattern basis in a reasonable period of time. To address this problem, this dissertation describes a new expert-driven approach to set-based pattern validation.

The proposed validation approach is based on validation sequences, i.e., we rely on the expert's ability to iteratively apply various *validation operators* that can validate multiple patterns at a time, thus making the expert-based validation feasible. We identified the class of scalable set predicates called *cardinality predicates* and demonstrated how these predicates can be effectively used in the validation process, i.e., as a basis for validation operators. We examined various properties of cardinality predicates, including their expressiveness. We also have developed and implemented the set validation language (SVL) that can be used for manual specification of cardinality predicates by a domain expert. In addition, we have proposed and developed a scalable algorithm for set and rule grouping that can be used to generate cardinality predicates automatically.

The dissertation also explores various theoretical properties of sequences of validation operators and facilitates a better understanding of the validation process. We have also addressed the problem of finding optimal validation sequences and have shown that certain formulations of this problem are NP-complete. In addition, we provided some heuristics for addressing this problem.

Finally, we have tested our rule validation approach on several real-life applications, including personalization and bioinformatics applications.

Contents

List of Figures

List of Tables

Part I

Expert-Driven Validation of Set-Based Data

Chapter 1

Introduction

1.1 Motivations

The research area of data mining, often also called knowledge discovery in data (KDD), deals with the discovery of useful information in large collections of data. In recent years, association rules [8] emerged as one of the most popular types of data mining patterns, and they are being discovered in a variety of applications. One fundamental advantage of association rule discovery from a given dataset is the *completeness* of data mining results. That is, association rule mining methods usually find *all* rules (i.e., not only the ones with predetermined dependent variables, as in various classification methods) that satisfy minimum support and confidence requirements specified by the user.

However, this advantage at the same time can become a significant limitation, since the number of discovered rules often can be huge (e.g., measured in tens of millions or more). This is particularly common in "dense" datasets [14] or the datasets with highly correlated attributes. Another very common criticism of many association rule discovery algorithms is that they produce not only too many rules, but also that many of the discovered rules are spurious, trivial, or simply irrelevant to the application at hand [34, 66, 46, 71, 73, 20, 79, 78, 14].

To address this problem, most previous approaches have focused on developing various measures of rule interestingness that could be used to prune the non-interesting rules. Alternatively, these measures could be directly incorporated into association rule discovery algorithms in order to generate only interesting rules. Excellent surveys of statistical (or objective) rule interestingness measures can be found in [42, 81]. Other approaches to deal with large numbers of discovered rules include multi-level organization and summarization [56], as well as introducing the *subjective* measures of rule interestingness, such as unexpectedness [60, 61, 54, 79] and actionability [73, 1].

Many authors advocate the direct involvement of the user (e.g., domain expert) in the process of post-analysis (or validation) of data mining results, and the rule validation problem in the post-analysis stage of the data mining process has been addressed before. In particular, there has been work done on specifying filtering constraints that select only certain types of rules from the set of all the discovered rules; examples of this research include [46, 53, 55]. In these approaches the user specifies constraints but does not do it *iteratively.* In contrast to this, it has been observed by several researchers, e.g. [18, 32, 72, 67, 50, 2, 69], that knowledge discovery should be an iterative and interactive process that involves an explicit participation of the domain expert. In our research we have followed the latter approach and applied it to the rule validation process.

Note, that the "quality" of discovered rules can be defined in several ways. In particular, rules can be “good” because they are:

1. statistically valid;
2. acceptable to a human expert in a given application;
3. “effective” in the sense that they result in certain benefits obtained in an application.

In our research, we have focused on the first two aspects, i.e., statistical validity and acceptability to an expert. The third aspect of rule quality is a more complex issue, and we do not address it in this dissertation, leaving it as a topic for future research.

1.2 Validation Problem: The Proposed Approach

Before discussing the rule validation problem, we would like to formulate a more general validation problem and describe our proposed approach to handling it.

Let's assume that we have a finite set $\mathcal{E}$ that contains all possible data points. Then a *dataset* D is simply a set of data points, i.e., $D \subseteq \mathcal{E}$. Let's assume that the domain expert has to “validate” some dataset D. Here the concept of validation is understood as follows.

The domain expert has a set $\mathcal{L}$ of possible labels. Let's denote these labels L_1, L_2, …, L_n. Then, given dataset D, the goal of the validation process is to “label” each input element $e \in D$ with one of the labels from $\mathcal{L}$. In other words, the goal is to split the input set D into $n+1$ pairwise disjoint sets V_1, V_2, …, V_n, and U, where each V_i represents the subset of D that was labeled with L_i and U denotes the subset of D that remains unlabeled after the validation process (i.e., some input elements may remain unvalidated).

For example, in a data mining application, the domain expert may want to validate the discovered association rules by "accepting" the rules that are relevant to him/her and rejecting the ones that are irrelevant. Therefore, in this case $\mathcal{L}$ would be defined as $\mathcal{L} := \{\texttt{Accept}, \texttt{Reject}\}$.

We assume that dataset D contains a large number of data points, because otherwise the domain expert would be able to validate (i.e., to label) all data points manually, i.e., in a one-by-one manner. However, with many data points to be validated, their individual validation by an expert is not feasible. To make the expert-driven validation process feasible, we propose to use *validation operators*, i.e., tools that allow the expert to validate multiple rules at a time. More specifically, instead of validating (labeling) data points one at a time, we propose to use *predicates* to specify a *class* of data points that should be labeled with a particular label.

In particular, let's denote $\mathcal{P}$ to be the set of all possible predicates for validating input data from $\mathcal{E}$, i.e., let $\mathcal{P}$ contain all predicates p of the form:

$$p : \mathcal{E} \longrightarrow \{\text{True}, \text{False}\} \tag{1.1}$$

Then, the validation operator is defined as follows.

Definition 1 (Validation Operator) *A tuple (l, p), where $l \in \mathcal{L}$ and $p \in \mathcal{P}$, is called a* validation operator.

In other words, given an unvalidated dataset D and an expert-specified validation operator $o = (l, p)$, all data points $e \in D$ for which $p(e) = \texttt{True}$ are labeled

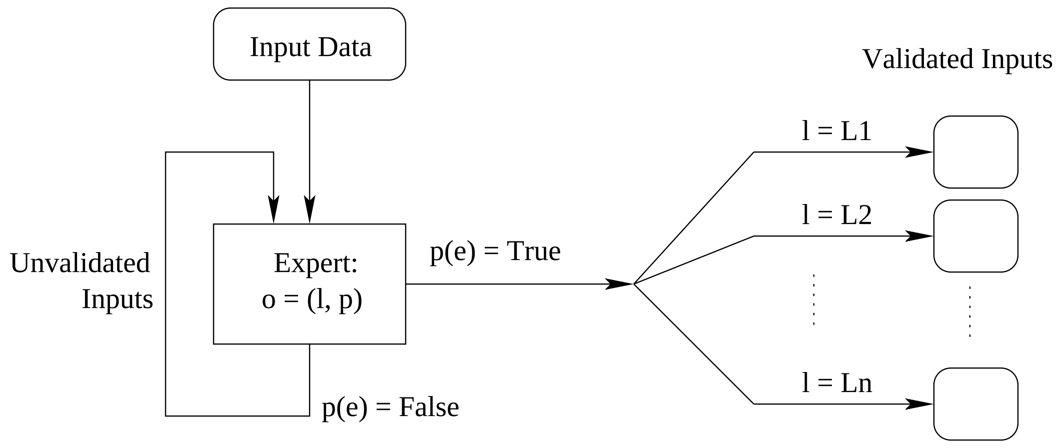


Figure 1.1: Expert-driven validation process.

with l and are considered validated. Data points $e \in D$ for which $p(e) = \texttt{False}$ remain unvalidated. Therefore, validation is an iterative process, where in each iteration the domain expert can specify a new validation operator that validates another portion of input dataset D. In other words, the validation process can be described as a *sequence* of validation operators.

Definition 2 (Validation Sequence) *A sequence of validation operators is called a validation sequence. We will denote the validation sequence as* $< o_1, o_2, \ldots, o_k >$*, where* $o_i = (l_i, p_i)$*,* $l_i \in \mathcal{L}$ *and* $p_i \in \mathcal{P}$*.*

The schematic description of the validation process is presented in Figure 1.1. Here we have defined the general validation problem. In the next chapter we will discuss the specifics of validation operators when dealing with set-based data (i.e., sets and rules).

1.3 Contributions

The contributions of the research presented in this dissertation are the following.

We have introduced a novel expert-driven approach to validating large numbers of rules (or itemsets) discovered by association rule mining methods. In fact, our approach can be straightforwardly generalized to validating large collections of any set-based data (e.g., collections of sets or rules), and not necessarily only those collections that were discovered by data mining algorithms. This approach is based on validation sequences, i.e., the expert's ability to iteratively apply various validation operators that can validate multiple rules at a time, thus making the expert-based validation feasible.

We identified the class of set predicates called *cardinality predicates*. We demonstrated how these predicates can be effectively used in the validation process, i.e., as a basis for validation operators. We examined various properties of cardinality predicates, including their expressiveness. We also showed that they can be efficiently implemented. In other words, the proposed cardinality predicates scale well, which is crucial in the post-analysis of data mining results, where the number of discovered rules can be very large.

We have developed and implemented the set validation language (SVL), which is an intuitive and user-friendly language that allows the domain expert to specify validation operators.

We have developed and implemented a novel algorithm for grouping sets and rules. We also showed that this algorithm is scalable.

We explored various theoretical properties of validation sequences. In particular, we have introduced the concepts of sequence permutation, sequence equivalence and strong equivalence, sequence optimality, and some others. We have proved various propositions about these concepts, which provides for better understanding of validation process. We have also addressed the problem of validation sequence optimization. We have shown that certain formulations of this problem are NP-complete and provided some heuristics for addressing this problem.

In addition, we have tested our rule validation approach in several real-life applications. In particular, we have addressed the problem of constructing individual user profiles in personalization applications. We have proposed to augment the traditional factual user profiles with the behavioral component (i.e., rules). We used our validation approach to enable the domain expert (i.e., marketing analyst) to validate the individual user profiles after the rules have been discovered by data mining algorithms.

We have also applied the rule validation ideas in the area of bioinformatics. More specifically, we have proposed to use association rule methods to discover rules in the biological microarray data. Since the numbers of rules discovered in such data collections are usually huge, we showed how to use the validation tools to deal with this issue. In other words, we have adapted our validation tools (mainly the set validation language and the rule grouping algorithm) so that the domain expert (geneticist/biologist) can analyze these large numbers of rules in a convenient manner.

Chapter 2

Validation Operators

2.1 Set Predicates

In our research, we have focused on the validation of set-based data, or more precisely, on set-based data mining results (i.e., rules and itemsets). For the remainder of this dissertation we will assume that there exists a discrete and finite set $\mathcal{I} = \{i_1, i_2, \dots, i_n\}$. We will call $\mathcal{I}$ a *base set*. Also, the elements of $\mathcal{I}$ are usually called *items*. For example, in a supermarket application $\mathcal{I}$ could be the set of all possible products.

Any set I, such that $I \subseteq \mathcal{I}$, is usually referred to as an *itemset*. Furthermore, an *association rule* is defined as an implication $A \Rightarrow C$, where A and C are itemsets (i.e., $A \subseteq \mathcal{I}$, $C \subseteq \mathcal{I}$) and $A \cap C = \emptyset$. In other words, an association rule can be represented by an ordered pair of two itemsets, i.e., (A, C). Here A is called an antecedent (or body, left-hand-side) of the rule, and C is called a consequent (or head, right-hand-side) of the rule.

The general validation process, described in the previous chapter, obviously directly applies to the validation of sets and rules. In the case of set validation, set $\mathcal{E}$ (i.e., set of all possible data points to be validated) is simply a set of all possible itemsets. That is, $\mathcal{E}_{set} = Powerset(\mathcal{I})$. In case of rule validation, naturally, $\mathcal{E}_{rule} = \{(A, C) : A \subseteq \mathcal{I}, C \subseteq \mathcal{I}, A \cap C = \emptyset\}$, i.e., the set of all possible rules. Most of the research presented in this dissertation applies for both of these cases. Furthermore, it can also be extended to other validation applications, e.g., where $\mathcal{E}$ consists of relational data tuples.

As mentioned in the previous chapter, we propose to use *predicates* to specify a class of data points to be validated at once, rather than validating data points on a one-by-one basis. Since we are dealing with set-based data, we will be using *set predicates* when specifying validation operators. That is, we will be using the predicates of the form:

$$p : Powerset(\mathcal{I}) \longrightarrow \{\text{True}, \text{False}\} \tag{2.1}$$

Set predicates can be applied to sets and rules as follows. Assume, p is an arbitrary set predicate. We can apply this predicate directly to any itemset I, i.e., itemset I matches predicate p if and only if $p(I) = \texttt{True}$. Furthermore, let $R = (A, C)$ be an arbitrary rule, i.e., $A \subseteq \mathcal{I}$, $C \subseteq \mathcal{I}$, $A \cap C = \emptyset$. Then p can be applied to R in several ways, each of which corresponds to a certain itemset of R:

- p can be applied to the body (antecedent) of rule. We will denote it as predicate $p[body]$. That is, rule $R = (A, C)$ matches $p[body]$ if and only if itemset A (i.e., the body of R) matches predicate p;

- p can be applied to the head (consequent) of rule. We will denote it as predicate $p[head]$. That is, rule $R = (A, C)$ matches $p[head]$ if and only if itemset C (i.e., the head of R) matches predicate p;

- p can be applied to the whole rule. We will denote it as predicate $p[rule]$. That is, rule $R = (A, C)$ matches $p[rule]$ if and only if itemset $A \cup C$ (i.e., the whole rule R) matches predicate p.

An important question is what kind of set predicates we should use for set and rule validation. When developing a query language or a programming language, it often comes down to a tradeoff between expressiveness of the language and its computational complexity. That is, we may choose an elaborate language that allows the domain expert to express validation decisions in a highly intuitive manner. However, while giving much expressive freedom to the expert, such language is likely to be computationally expensive, i.e., in many cases it may not be able to evaluate quickly whether a given set or rule satisfies a predicate. On the other hand, we may choose a language that works very fast, which obviously is important in the validation of large numbers of rules discovered by data mining algorithms. Moreover, such validation is an interactive and iterative process, therefore we definitely would like to have the language that is as computationally inexpensive as possible, so that the domain expert does not have to wait for long periods of time in order to see the results of the previously applied validation operator. However, such language is likely to allow only very simple and straightforward predicates, and therefore the expressive power of the domain expert could be significantly limited.

Having in mind this tradeoff between the expressiveness and the computational complexity, we propose to use a class of set predicates, called *cardinality predicates*, which is a compromise between the two issues. In other words, these predicates are both scalable and expressive. We describe *cardinality predicates* in the next section.

2.2 Cardinality Predicates

2.2.1 Definition and Basic Properties

Given any itemset $X \subseteq \mathcal{I}$, we will define the set $[X]$ as: $[X] := \{0, 1, 2, \ldots, |X|\}$.

Definition 3 (Cardinality Predicate) *Given the sets X and S, where $X \subseteq \mathcal{I}$ and $S \subseteq [X]$, the cardinality predicate C_X^S is defined as*

$$C_X^S(I) = \begin{cases} \texttt{True}, & \text{if } |X \cap I| \in S \\ \texttt{False}, & \text{otherwise} \end{cases} \tag{2.2}$$

where I is any itemset, i.e., $I \subseteq \mathcal{I}$. Such predicate is called a cardinality predicate, or c-predicate for short.

In other words, given itemset I, the cardinality predicate $C_X^S(I)$ evaluates to `True` if and only if the size of the set $X \cap I$ is among the sizes specified in S. We will call I an *input* set, X a *comparison* set, and S a *cardinality* set.

Below is an example of a cardinality predicate.

> **Example 1** Assume $\mathcal{I} = \{a, b, c, d, e\}$. Then, cardinality predicate $C_{a,b}^1$ would match any input itemset $I \subseteq \mathcal{I}$ that has either a or b item present, but not both. For example, let $I_1 = \{a, c, d\}$ and $I_2 = \{a, b, e\}$. Then, $C_{a,b}^1(I_1) = \texttt{True}$ and $C_{a,b}^1(I_2) = \texttt{False}$[1].

[1]Note, that according to the definition of cardinality predicates we should write $C_{\{a,b\}}^{\{1\}}(I_1)$, but we use the $C_{a,b}^1(I_1)$ notation here and in the rest of the dissertation for the sake of better readability.

Sample Input Itemsets (I)	Cardinality Predicates (C_X^S)				
	$C^1_{a,b}$	C^0_c	$C^{0,2}_{b,e}$	$C^3_{a,b,c}$	$C^{0,1,3}_{b,c,d,e}$
$\{a\}$	`True`	`True`	`True`	`False`	`True`
$\{a, c, d\}$	`True`	`False`	`True`	`False`	`False`
$\{a, b, e\}$	`False`	`True`	`True`	`False`	`False`
$\{b, c\}$	`True`	`False`	`False`	`False`	`False`
$\{b, d, e\}$	`True`	`True`	`True`	`False`	`True`
$\{c, e\}$	`False`	`False`	`False`	`False`	`False`
$\{a, b, c, d\}$	`False`	`False`	`False`	`True`	`True`

Table 2.1: Examples of cardinality predicates.

Example 2 Assume $\mathcal{I} = \{a, b, c, d, e\}$. Cardinality predicate $C^1_{a,b}[body]$ then would match any rule $R = (A, C)$ that has either a or b item present in its body, but not both. For example, let $R_1 = (\{a, c, d\}, \{b\})$ and $R_2 = (\{a, b, e\}, \{c\})$. In other words, R_1 represents the association rule $a \wedge c \wedge d \Rightarrow b$, and R_2 represents the association rule $a \wedge b \wedge e \Rightarrow c$. Then, $C^1_{a,b}[body](R_1) =$ `True` and $C^1_{a,b}[body](R_2) =$ `False`.

More examples of cardinality predicates and their values given certain inputs can be found in Table 2.1 (assuming $\mathcal{I} = \{a, b, c, d, e\}$).

Here are some basic properties of cardinality predicates.

Lemma 1 *The following equalities hold for every itemset $I \subseteq \mathcal{I}$.*

$$C_X^{\emptyset}(I) = \texttt{False} \tag{2.3}$$

$$C_X^{[X]}(I) = \texttt{True} \tag{2.4}$$

$$\neg C_X^S(I) = C_X^{\overline{S}}(I), \quad \text{where } \overline{S} := [X] - S \tag{2.5}$$

$$C_X^{S_1}(I) \wedge C_X^{S_2}(I) = C_X^{S_1 \cap S_2}(I) \tag{2.6}$$

$$C_X^{S_1}(I) \vee C_X^{S_2}(I) = C_X^{S_1 \cup S_2}(I) \tag{2.7}$$

$$\bigwedge_{i=1}^{n} C_{X_i}^{|X_i|}(I) = C_{\cup X_i}^{|\cup X_i|}(I) \tag{2.8}$$

$$\bigwedge_{i=1}^{n} C_{X_i}^{0}(I) = C_{\cup X_i}^{0}(I) \tag{2.9}$$

► Follows directly from the definition of cardinality predicates. ◄

2.2.2 Combining Cardinality Predicates Using Boolean Operators

Several cardinality predicates (as defined in Section 2.2.1) can be combined into one more complex set predicate using standard Boolean operations such as conjunction ($\wedge$), disjunction ($\vee$), and negation ($\neg$). The matching semantics of such a predicate combination is defined in a standard manner. That is, $\neg C_X^S(I)$ is `True` if and only if $C_X^S(I)$ is `False`; $C_{X_1}^{S_1} \wedge C_{X_2}^{S_2}(I)$ is `True` if and only if both $C_{X_1}^{S_1}(I)$ and $C_{X_2}^{S_2}(I)$ are `True`; and $C_{X_1}^{S_1} \vee C_{X_2}^{S_2}(I)$ is `True` if and only if at least one of $C_{X_1}^{S_1}(I)$ and $C_{X_2}^{S_2}(I)$ is `True`.

In order to distinguish between the simple cardinality predicates of the form of the form C_X^S (as defined in Section 2.2.1) and the more complex predicates that combine several simple cardinality predicates using Boolean operators, we will call the former *atomic* cardinality predicates and the latter *composite* cardinality predicates.

2.3 Completeness of Cardinality Predicates

As mentioned earlier, we use *set predicates* for the purpose of set and rule validation. That is, any predicate p of the form described in Equation 2.1 can be used for validation purposes.

From elementary mathematics we know that if we have two finite sets M and N, the number of possible functions from M to N (i.e., functions $f : M \longrightarrow N$) is $|N|^{|M|}$. Therefore, the number of different possible set predicates described by Equation 2.1 is $2^{|Powerset(\mathcal{I})|}$. Furthermore, since the size of $Powerset(\mathcal{I})$ is 2^N (assuming $|\mathcal{I}| = N$), the total number of different set validation predicates is 2^{2^N}.

One way to estimate the expressive power of a particular class C of set predicates is to determine how many different predicates (out of possible 2^{2^N}) can be specified (expressed) using the predicates from class C. Let's consider the following simple class M of set predicates – the set *membership* predicates. In particular, given an input itemset I, the set membership predicate m_x is defined as follows:

$$m_x(I) = \begin{cases} \text{True}, & x \in I \\ \text{False}, & \text{otherwise} \end{cases} \tag{2.10}$$

More specifically, let's denote the class M as containing all possible membership predicates m_x ($x \in \mathcal{I}$) as well as their combinations using standard Boolean operations AND ($\wedge$), OR ($\vee$), and NOT ($\neg$).

Let's assume p is an arbitrary set validation predicate, i.e., p is a function described by (2.1). Let p_T be the set of all possible inputs for which p evaluates to `True`, i.e., $p_T = \{X \in Powerset(\mathcal{I}) | p(X)\}$. Since $Powerset(\mathcal{I})$ is finite and $p_T \subseteq Powerset(\mathcal{I})$ (by definition), p_T is also finite. Therefore, let's enumerate p_T, i.e., $p_T = \{X_1, X_2, \ldots, X_k\}$. Then predicate p can be expressed as follows (for any input set $I \subseteq \mathcal{I}$):

$$p(I) = \bigvee_{i=1}^{k} \left(\bigwedge_{x \in X_i} m_x(I) \wedge \bigwedge_{x \notin X_i} \neg m_x(I) \right) \tag{2.11}$$

Essentially, the above expression is analogous to how every Boolean function can be expressed using the disjunctive normal form.

Since *every set predicate can be expressed with a Boolean combination of membership predicates*, we say that class M is complete. By the same argument, the class of cardinality predicates is also complete, since $m_x = C_x^1$, i.e., any set predicate can be expressed using cardinality predicates with the same Boolean combination as in Equation 2.11, only replacing all m_x predicates with C_x^1.

2.4 Expressiveness of Cardinality Predicates

While the class of cardinality predicates is complete, as shown in the previous section, obviously not every set predicate can be expressed with cardinality predicates *concisely* by the domain expert. However, many widely used and universally accepted set predicates can indeed be expressed concisely using cardinality predicates and we demonstrate it in this section.

2.4.1 Expressing Simple Membership Predicates

Consider the following simple set membership predicates. INCLUDE_ALL_X is a set predicate that evaluates to `True` for those itemsets $I \subseteq \mathcal{I}$ that include all items from the given set X. Similarly, set predicate INCLUDE_NONE_X evaluates to `True` for all itemsets $I \subseteq \mathcal{I}$ that include none of the elements from the given set X. These two predicates can be expressed using cardinality predicates as follows:

$$\begin{aligned} \text{INCLUDE\_ALL}_X &= C_X^{|X|} \\ \text{INCLUDE\_NONE}_X &= C_X^{0} \end{aligned}$$

2.4.2 Expressing Subset/Superset Predicates

Consider the standard subset and superset predicates. In particular, set predicate SUPERSET_X evaluates to `True` for those itemsets $I \subseteq \mathcal{I}$ that are supersets of the given itemset X. Similarly, set predicate SUBSET_X evaluates to `True` for those itemsets $I \subseteq \mathcal{I}$ that are subsets of the given itemset X. Furthermore, let PROPER_SUPERSET_X and PROPER_SUBSET_X be the standard "proper superset" and "proper subset" predicates. All these predicates can be concisely expressed using cardinality predicates as follows:

$$\begin{aligned} \text{SUPERSET}_X &= \text{INCLUDE\_ALL}_X \\ &= C_X^{|X|} \\ \text{SUBSET}_X &= \text{INCLUDE\_NONE}_{\overline{X}} \\ &= C_{\overline{X}}^{0} \\ \text{PROPER\_SUPERSET}_X &= \text{SUPERSET}_X \wedge \neg\text{INCLUDE\_NONE}_{\overline{X}} \\ &= C_X^{|X|} \wedge \neg C_{\overline{X}}^{0} \\ &= C_X^{|X|} \wedge C_{\overline{X}}^{1,\ldots,|X|} \end{aligned}$$

$$\begin{aligned}\text{PROPER\_SUBSET}_X &= \text{SUBSET}_X \wedge \neg\text{INCLUDE\_ALL}_X \\ &= C^0_{\overline{X}} \wedge \neg C^{|X|}_X \\ &= C^0_{\overline{X}} \wedge C^{0,\ldots,|X|-1}_X\end{aligned}$$

Here set $\overline{X}$ denotes a complement of set X and is defined as $\overline{X} := \mathcal{I} - X$.

2.4.3 Expressing Set Equality/Inequality Predicates

Consider the standard set equality and inequality predicates. In particular, set predicate EQUAL_X evaluates to `True` for those itemsets $I \subseteq \mathcal{I}$ that are equal to the given itemset X. Similarly, set predicate NOT_EQUAL_X evaluates to `True` for those itemsets $I \subseteq \mathcal{I}$ that are not equal to the given itemset X. These predicates can be concisely expressed using cardinality predicates as follows:

$$\begin{aligned}\text{EQUAL}_X &= \text{SUBSET}_X \wedge \text{SUPERSET}_X \\ &= C^{|X|}_X \vee C^0_{\overline{X}} \\ \text{NOT\_EQUAL}_X &= \neg\text{EQUAL}_X \\ &= \neg(C^{|X|}_X \wedge C^0_{\overline{X}}) \\ &= \neg C^{|X|}_X \vee \neg C^0_{\overline{X}} \\ &= C^{0,\ldots,|X|-1}_X \vee C^{1,\ldots,|X|}_{\overline{X}}\end{aligned}$$

2.4.4 Expressing Set Size Restriction Predicates

Consider the following set size restriction predicates. In particular, set predicate SIZE^k_X evaluates to `True` for those itemsets $I \subseteq \mathcal{I}$ that have exactly k items from the given set X. Set predicate MAX_SIZE^k_X evaluates to `True` for sets that have at most k items from the given set X. Set predicate MIN_SIZE_X^k evaluates to `True` for inputs that have at least k items from the given set X. Finally, set predicate $\text{SIZE\_RANGE}_X^{a,b}$ evaluates to `True` for inputs that have at least a and at most b items from the given set X. These sets can be straightforwardly expressed using cardinality predicates:

$$\begin{aligned} \text{SIZE}_X^k &= C_X^k \\ \text{MAX\_SIZE}_X^k &= C_X^{0,\ldots,k} \\ \text{MIN\_SIZE}_X^k &= C_X^{k,\ldots,|X|} \\ \text{SIZE\_RANGE}_X^{a,b} &= C_X^{a,\ldots,b} \end{aligned}$$

If restrictions on *absolute set sizes* are wanted, simply replace X by $\mathcal{I}$ in the above equations. For example, set predicate SIZE^k evaluates to `True` for itemsets of size k. It can be expressed using cardinality predicates as follows:

$$\text{SIZE}^k = C_{\mathcal{I}}^k$$

The summary of the commonly used set predicates that can be concisely expressed using cardinality predicates is presented in Table 2.2.

2.4.5 Limitations of Cardinality Predicates

In this section we discuss several limitations of the expressivity of cardinality predicates.

Note, that we do not assume any specific properties about the base set $\mathcal{I}$ and its items. For example, in some application domains, all items in $\mathcal{I}$ may be numbers. In those applications the domain expert might be interested in using predicates

Set predicate	Expression using cardinality predicates
INCLUDE_ALL$_X$	$C_X^{\lvert X\rvert}$
INCLUDE_NONE$_X$	C_X^0
SUPERSET$_X$	$C_X^{\lvert X\rvert}$
SUBSET$_X$	$C_{\overline{X}}^0$
PROPER_SUPERSET$_X$	$C_X^{\lvert X\rvert} \wedge C_{\overline{X}}^{1,\ldots,\lvert X\rvert}$
PROPER_SUBSET$_X$	$C_{\overline{X}}^0 \wedge C_X^{0,\ldots,\lvert X\rvert-1}$
EQUAL$_X$	$C_X^{\lvert X\rvert} \vee C_{\overline{X}}^0$
NOT_EQUAL$_X$	$C_X^{0,\ldots,\lvert X\rvert-1} \vee C_{\overline{X}}^{1,\ldots,\lvert X\rvert}$
SIZE$_X^k$	C_X^k
MAX_SIZE$_X^k$	$C_X^{0,\ldots,k}$
MIN_SIZE$_X^k$	$C_X^{k,\ldots,\lvert X\rvert}$
SIZE_RANGE$_X^{a,b}$	$C_X^{a,\ldots,b}$

Table 2.2: Expressing various common set predicates using cardinality predicates.

with arithmetic capabilities, e.g., "match all itemsets where the sum (or *max*, *min*, *avg*, or some other function) of its elements is greater than k." Therefore, predicates that assume something about the "type" of items cannot be expressed using cardinality predicates. However, this can be viewed as a benefit as well, since by specifically selecting the class of *cardinality* predicates for the validation purposes, we stayed independent of specific application domains. Because of that, our validation approach can be easily adapted for such diverse application domains as personalization and bioinformatics, as will be shown in Chapters 5 and 6.

Another limitation of cardinality predicates C_X^S is that they use only one comparison set X. Therefore, it is not possible to express certain useful set predicates, even when combining several cardinality predicates using Boolean operations. One example of such potentially useful set predicate would be: "given comparison sets X and Y, match all input itemsets that have more items from set X than from set Y." Or more specifically, in the supermarket application: "match all shopping baskets that have more dairy products than meat products." Cardinality predicates that have more than one comparison set is one of the topics for our future research.

2.5 Computational Complexity of Cardinality Predicates

Given an atomic cardinality predicate C_X^S and an input itemset I, it is easy to show that the computational complexity of calculating the value of $C_X^S(I)$ is $O(|I|)$, if we store the sets X and S in data structures that allow to perform a set membership test operation in constant time. Obviously, there are many such data structures (e.g., bitmaps, lookup tables, hash structures), and the choice of a particular data

```
Input:   Comparison set $X$
         Cardinality set $S$
         Input set $I$
Output:  Value of $C_X^S(I)$, i.e., True or False.

begin
(1)  cnt := 0;
(2)  for each (i $\in I$)
(3)      if (i $\in X$) then cnt++;
(4)  if (cnt $\in S$) then return True;
(5)  else return False;
end
```

Figure 2.1: An implementation of a cardinality predicate.

structure is up to the programmer.

The straightforward algorithm for implementing a cardinality predicate is presented in Figure 2.1. Since the algorithm essentially consists of $|I| + 1$ set membership test operations, its computational complexity is $O(|I|)$.

In this analysis we ignored the cost for setting up the data structures for sets X and S. For example, using the lookup table-based data structures, the setup can be performed fairly efficiently in at most $|\mathcal{I}|$ steps. However, these setup costs occur only once, and subsequently large numbers of sets (or rules) can be validated using the same setup. Hence, we assume that the computational complexity of the validation of large dataset dominates the complexity of the one-time setup cost, and therefore we can say that computational complexity of cardinality predicates is linear with respect to the size of its input.

If we have a Boolean expression of t atomic cardinality predicates then in general its computational complexity given the input itemset I is $O(t|I|)$. However if we have a conjunction of t atomic cardinality predicates, i.e., $p := \bigwedge_{i=1}^{t} C_{X_i}^{S_i}$, then in certain special cases we can achieve computational complexity that is better than $O(t|I|)$, for example:

- If all comparison sets X_i are equal, then, based on Lemma 1, p can be expressed as a single atomic cardinality predicate $p = C_{X_i}^{\cap S_i}$. Therefore, its computational complexity would be $O(|I|)$ instead of $O(t|I|)$.
- If all comparison sets X_i are pairwise disjoint (i.e., $X_i \cap X_j = \emptyset$ when $i \neq j$), then obviously a single lookup table is sufficient to host all of the comparison sets, making $O(|I|)$ computational complexity possible.

2.6 Set Validation Language (SVL)

In this chapter, we present a language for the validation of set-based data – mainly for the validation of large numbers of itemsets and/or rules discovered by association rule mining algorithms. This language is called SVL (for Set Validation Language), and it allows the domain expert to specify various validation operators based on cardinality predicates. In other words, SVL is the language in which domain expert can specify what types of rules he or she wants to label as belonging to a certain class, e.g., the class of `Rejected` (unimportant) rules. After a template is specified, unvalidated rules are "matched" against it. Data (sets or rules) that match a template are labeled with the corresponding label (specified by the validation operator) and are considered validated. Rules that do not match a template remain unvalidated.

The problem of post-analysis of large numbers of discovered rules using filtering methods has been studied before in the KDD literature [3, 43, 46, 53, 57, 70, 77], and we utilize some of this work in our approach. In particular, Klemettinen et al [46] and Imielinski et al [43] present the methods for the users to specify classes of patterns in which they are interested by providing *pattern templates* expressed in a certain specification language. However, the pattern specification languages in most of the previous approaches are *ad hoc*, while SVL is based strictly on the class of cardinality predicates.

The formal BNF specification of the SVL syntax is presented in Figure 2.2. In the rest of this section we will briefly overview the SVL language and some of its features. Some examples of SVL templates can also be found in Chapters 5 and 6.

2.6.1 Basic SVL Syntax

SVL is based on cardinality predicates that were introduced earlier in this chapter. That is, SVL is a tool that allows the domain expert to specify the cardinality predicates in a intuitive and human-readable form. In its simplest form, any cardinality predicate C_X^S, where $X \subseteq \mathcal{I}$ and $S \subseteq [X]$, can be expressed using SVL as follows:

$$\mathit{InputSetIndicator}\ \ \mathbf{HAS}\ \ S\ \ \mathbf{FROM}\ \ X \tag{2.12}$$

Validation_Operator	::=	`Label :` *SVL_Template*
SVL_Template	::=	*PredicateConjunction* |
		PredicateConjunction **OR** *CompositeTemplate*
PredicateConjunction	::=	*AtomicPredicate* |
		AtomicPredicate **AND** *PredicateConjunction*
AtomicPredicate	::=	[**NOT**] *InputSetIndicator* **HAS** *CardinalitySet*
		FROM [**CO**] *ComparisonSet* [**ONLY**]
InputSetIndicator	::=	**ITEMSET** | **BODY** | **HEAD** | **RULE**
CardinalitySet	::=	*CardinalityItem* | *CardinalityItem*, *CardinalitySet*
ComparisonSet	::=	*ComparisonItem* | *ComparisonItem*, *ComparisonSet*
CardinalityItem	::=	*CardinalityValue* | *CardinalityRange*
ComparisonItem	::=	`Item` | `ItemGroup` | *ItemAndValues*
CardinalityRange	::=	ANY | NOTALL | NOTZERO | SOME |
		CardinalityValue − *CardinalityValue*
CardinalityValue	::=	NONE | ALL | `CardinalityNumber`
ItemAndValues	::=	`Item` *EqOp* { *ValueSet* }
EqOp	::=	= | !=
ValueSet	::=	`Value` | `Value`, *ValueSet*

Figure 2.2: The syntax of the template specification language (in BNF).

where *InputSetIndicator* can be of the following keywords: ITEMSET, BODY, HEAD, or RULE. It represents the part of the input element (i.e., set or rule) on which constraint is being placed. Keyword ITEMSET is used if and only if the input data consists of sets (itemsets). When dealing with input data consisting of rules, other three terms – BODY, HEAD, or RULE – should be used to indicate the part of the rule on which the constraints (restrictions) should be placed. More specifically, when *InputSetIndicator* is specified as BODY, the cardinality predicate represented by the above SVL template is applied to sets of items comprising the bodies (i.e., antecedents or left-hand-sides) of all the unvalidated rules in the input data. In other words, such template represents predicate $C_X^S[body]$. Similarly, if *InputSetIndicator* is HEAD, the SVL template represents predicate $C_X^S[head]$ and is applied to the sets of items from rule heads (i.e., consequents or right-hand-sides). Finally, if *InputSetIndicator* is RULE, the template represents predicate $C_X^S[rule]$ and is applied to the set of items comprising the whole rule, i.e., to the union set of rule body and head.

Furthermore, in Equation 2.12, X represents the comparison set of the underlying cardinality predicate, i.e., it represents the items against which the input data points should be compared. It is specified as a comma-separated list $x_1, x_2, \ldots, x_N$ (where $x_i \in \mathcal{I}$). Finally, S is a comma-separated list $s_1, s_2, \ldots, s_M$ (where $s_i \in [X]$) that represents the *cardinality set* of the cardinality predicate.

In addition, the atomic templates from Equation 2.12 can be combined into more complex templates using Boolean operations AND and OR.

Below we present some examples of cardinality predicates and the corresponding SVL statements, assuming $\mathcal{I} = \{a, b, c, d, e\}$.

Cardinality predicate	Corresponding SVL statement
$C^1_{a,b}$	ITEMSET **HAS** 1 **FROM** *a*,*b*
$C^{0,1,2}_{b,c,e}[body]$	BODY **HAS** 0,1,2 **FROM** *b*,*c*,*e*
$C^1_a[body] \wedge C^0_b[head]$	BODY **HAS** 1 **FROM** *a* **AND** HEAD **HAS** 0 **FROM** *b*

2.6.2 Extended SVL Syntax

To make SVL more user-friendly we have augmented the atomic SVL template, shown in 2.12, with the several useful features. Note, that all features described in this section are purely for the convenience of the domain expert. That is, they do not add any expressive power to the language – all SVL statements can still be expressed using cardinality predicates.

$$[\textbf{NOT}]\ \textit{InputSetIndicator}\ \textbf{HAS}\ \textit{CardinalitySet}\ \textbf{FROM}\ [\textbf{CO}]\ X \qquad (2.13)$$

Item Groups Instead of having to specify the comparison set by listing each item individually, the domain expert can first define a group of items, and then subsequently use the group name in SVL statements. For example, suppose we are dealing with the analysis of supermarket transactional data (i.e., all shopping baskets purchased over some period of time) and we need to validate all the frequent shopping baskets that were discovered using some association rule mining algorithm. Therefore, in this case $\mathcal{I}$ would be the set of all products sold in the store. Suppose, we want to find all the input rules that contain one or two milk products in their antecedents. One way to write this statement would be to list all the milk products individually, i.e.,

$$\text{BODY} \;\; \textbf{HAS} \;\; 1,2 \;\; \textbf{FROM} \;\; \text{SkimMilk}, \text{2\%Milk}, \text{WholeMilk}, \text{ButterMilk}, \ldots$$

On the other hand, the domain expert can define a group *MilkProducts* of items, i.e., MilkProducts = { SkimMilk, 2%Milk, WholeMilk, ButterMilk, ... }, and then rewrite the above SVL template as

$$\text{BODY} \;\; \textbf{HAS} \;\; 1,2 \;\; \textbf{FROM} \;\; \text{MilkProducts}$$

Defining item groups is a domain-specific issue and it is up to the expert that performs validation to define the groups that are useful to him/her. However, we do provide an expert with one pre-defined item group, named AllItems. This group contains all possible items for a given application, i.e., AllItems basically represents set $\mathcal{I}$. This group is often very useful when we want to place absolute size restrictions on input elements. For example, to match all the rules that have exactly 3 items in their antecedents, we would use the following template:

$$\text{BODY} \;\; \textbf{HAS} \;\; 3 \;\; \textbf{FROM} \;\; \text{AllItems}$$

Cardinality Synonyms For better readability and intuitiveness, the domain expert can use keywords NONE and ALL in SVL templates to represent cardinality numbers 0 and $|X|$, where X is a comparison set specified in the SVL template. Suppose, for example, that we would like to validate the shopping baskets that contain all of the following products (and possibly some others): orange juice, wheat bread, eggs, and bacon. We can state the corresponding SVL template as:

$$\text{ITEMSET} \;\; \textbf{HAS} \;\; \text{ALL} \;\; \textbf{FROM} \;\; \text{OrangeJuice}, \text{WheatBread}, \text{Eggs}, \text{Bacon}$$

Similarly, if we want to select the shopping baskets that have no milk products, we can employ the following SVL template that also uses the group *MilkProducts* as defined earlier:

ITEMSET **HAS** NONE **FROM** MilkProducts

Cardinality Ranges Sometimes the desired cardinality set contains several cardinality numbers that are consecutive. For example, among all frequent shopping itemsets we would like to find those that have either less than 3 or more than 5 milk products in them. Assuming there are 10 different milk products available, we could write the corresponding SVL template as:

ITEMSET **HAS** $0, 1, 2, 6, 7, 8, 9, 10$ **FROM** MilkProducts

On the other hand, we allow to specify a range of numeric values in a more straightforward manner, i.e.,

ITEMSET **HAS** $0 - 2, 6 - 10$ **FROM** MilkProducts

Furthermore, using the cardinality synonyms defined earlier, we could rewrite the above template as:

ITEMSET **HAS** NONE $- 2, 6 -$ ALL **FROM** MilkProducts

We also provide several "standard" ranges that can be specified using keywords ANY, NOTALL, NOTZERO, SOME. Here ANY stands for the range 0-ALL, NOTALL stands for 0-(ALL-1), NOTZERO stands for 1-ALL, and SOME stands for 1-(ALL-1). For example, the following template matches all itemsets that have at least one milk product:

ITEMSET **HAS** NOTZERO **FROM** MilkProducts

Complement of Comparison Set Suppose we would like have a template that matches all the itemsets containing at least three non-milk products. Assuming we have group MilkProducts defined, this template can be specified as:

$$\text{ITEMSET} \;\; \textbf{HAS} \;\; 3-\text{ALL} \;\; \textbf{FROM} \;\; \textbf{CO} \;\; \text{MilkProducts}$$

Here we are using the optional **CO** keyword which indicates that the actual comparison set for this template is the complement of the set indicated in the template. As defined earlier, the complement set $\overline{X}$ of any set of items X ($X \subseteq \mathcal{I}$) is defined as $\overline{X} := \mathcal{I} - X$.

Negation of the Atomic Template By specifying the optional keyword **NOT** in front of the atomic SVL template, we *negate* this template. The matching behavior of the negated template is defined in a standard way. That is, if input element e matches template **NOT** T if and only if e does not match T.

As mentioned before, each atomic SVL template represents some cardinality predicate C_X^S. From Lemma 1 we have that $\neg C_X^S = C_X^{\overline{S}}$, where $\overline{S} = [X] - S$. Therefore, the negation of a given atomic SVL template essentially implies the complementation of the specified cardinality set in this template. Therefore, for example, if we want to write a template that matches all itemsets except those that have exactly 2 milk products in them, we could write it as:

$$\textbf{NOT} \;\; \text{ITEMSET} \;\; \textbf{HAS} \;\; 2 \;\; \textbf{FROM} \;\; \text{MilkProducts}$$

Macro Capabilities of SVL As shown in Section 2.4, many useful set predicates (e.g., set equality, subset, superset predicates) can be expressed using cardinality predicates. For this purpose, we have implemented the mechanism for specifying simple textual macros using SVL templates. Some of the built in macros include:

$$\begin{aligned}
\text{SUBSET}(\mathit{InpSet}, X) &:= \mathit{InpSet}\ \textbf{HAS}\ \text{NONE}\ \textbf{FROM CO}\ X \\
\text{SUPERSET}(\mathit{InpSet}, X) &:= \mathit{InpSet}\ \textbf{HAS}\ \text{ALL}\ \textbf{FROM}\ X \\
\text{PR\_SUBSET}(\mathit{InpSet}, X) &:= \mathit{InpSet}\ \textbf{HAS}\ \text{NONE}\ \textbf{FROM CO}\ X\ \textbf{AND} \\
&\ \mathit{InpSet}\ \textbf{HAS}\ \text{NOTALL}\ \textbf{FROM}\ X \\
\text{PR\_SUPERSET}(\mathit{InpSet}, X) &:= \mathit{InpSet}\ \textbf{HAS}\ \text{ALL}\ \textbf{FROM}\ X\ \textbf{AND} \\
&\ \mathit{InpSet}\ \textbf{HAS}\ \text{NOTZERO}\ \textbf{FROM CO}\ X \\
\text{EQUAL}(\mathit{InpSet}, X) &:= \mathit{InpSet}\ \textbf{HAS}\ \text{ALL}\ \textbf{FROM}\ X\ \textbf{AND} \\
&\ \mathit{InpSet}\ \textbf{HAS}\ \text{NONE}\ \textbf{FROM CO}\ X \\
\text{SIZE}(\mathit{InpSet}, k) &:= \mathit{InpSet}\ \textbf{HAS}\ k\ \textbf{FROM}\ \text{AllItems}
\end{aligned}$$

More examples of SVL macros can be found in Chapter 6.

Chapter 3

Grouping of Set-Based Data

3.1 Motivations

As stated earlier, validation operators provide a way for the domain expert to examine multiple rules at a time. This examination process can be performed in the following two ways. First, the expert may already know some types of rules that he or she wants to examine and validate based on the prior experience. Therefore, it is important to provide capabilities allowing him or her to specify such types of rules. The set validation language SVL (described in the previous chapter) that allows the domain expert to specify validation operators based on cardinality predicates serves precisely this purpose. Second, the expert may not know all the relevant types of rules in advance, and it is important to provide methods that *group* discovered rules into classes that he or she can subsequently examine and validate. In this chapter we propose the similarity-based grouping method that can group sets or rules into groups according to some expert-specified grouping criteria.

Therefore, the proposed grouping method can be used in a couple of different ways. First, it can be used in conjunction with SVL in the process of set or rule validation. If, after specifying several validation operators using SVL, there are still many unvalidated rules remaining and the expert is not sure what validation operator to specify next, he or she can apply the grouping algorithm instead. In this case the groups generated by this method can be viewed as potential validation operators. That is, the domain expert can examine the groups and validate (label) some of them, as if he/she were specifying validation operators for these groups. And second, the proposed grouping method can be used in various data mining applications as a stand-alone post-analysis tool and not be a part of an interactive validation process.

In this chapter we will be mostly discussing the grouping of association rules. However, most of the ideas in this chapter can be straightforwardly applied to grouping itemsets as well. The main contribution of this chapter lies in that it proposes a grouping method for set-based data (i.e., sets and rules) that is flexible, scalable, intuitive and easy for the end-user to use in many data mining applications, especially in the validation process or simply where there is a need to evaluate large numbers of discovered rules.

3.2 Similarity-Based Grouping Approach

There can be many "similar" rules among all the discovered rules, and it would be useful for the domain expert to evaluate all these similar rules together rather than individually. In order to do this, some similarity measure that would allow grouping similar rules together needs to be specified.

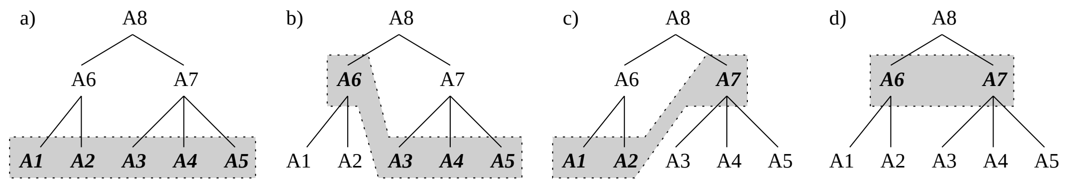


Figure 3.1: An example of an attribute hierarchy for similarity-based grouping.

3.2.1 Attribute Hierarchies

In this paper, we propose a method to specify such a similarity measure using *attribute hierarchies*. An attribute hierarchy is organized as a tree by the human expert in the beginning of the validation process.[1] The leaves of the tree consist of all the attributes of the data set to which rule discovery methods were applied, i.e., all the attributes that can potentially be present in the discovered rules. The non-leaf nodes in the tree are specified by the human expert and are obtained by combining several lower-level nodes into one parent node. For instance, Figure 3.1 presents an example of such a hierarchy, where nodes A1 and A2 are combined into node A6 and nodes A3, A4 and A5 into node A7, and then nodes A6 and A7 are combined into node A8. Another example of an attribute hierarchy is presented in Figure 3.5. We call non-leaf nodes of an attribute hierarchy *aggregated attributes*.

[1]In certain domains, e.g., groceries, such hierarchies may already exist, and some well-known data mining algorithms, such as [26, 76], explicitly assume the existence of attribute (or, more generally, feature) hierarchies. Alternatively, attribute hierarchies may possibly be constructed automatically in certain other applications. However, automatic construction of such hierarchies is beyond the scope of this paper.

3.2.2 Basic Steps of Grouping Algorithm

The attribute hierarchy is used for determining similar rules and grouping them together. More specifically, the semantics of the similarity-based grouping approach is defined as follows.

1. **Specifying rule aggregation level**. Rules are grouped by specifying the level of rule aggregation in the attribute hierarchy which is provided by the human expert. Such a specification is called a *cut*, and it forms a subset of all the nodes of the tree (leaf and non-leaf), such that for every path from a leaf node to the root, exactly one node on such path belongs to this subset. Therefore, given a cut, every leaf node has its corresponding *cut node*. Given a cut C, we define for any leaf node X_i its corresponding cut node $cut_C(X_i)$ as follows:

   $$cut_C(X_i) = \begin{cases} X_i, & if \;\; X_i \in C \\ cut_C(parent(X_i)), & otherwise \end{cases}$$

   Figure 3.1 presents several different cuts of an attribute hierarchy that are represented by shaded regions. For example, for the cut from Figure 3.1(c), $cut_{3c}(A2) = A2$ and $cut_{3c}(A3) = A7$. Moreover, the cut node of any leaf node can be calculated in constant time by implementing a straightforward lookup table for that cut.

2. **Aggregating rules**. Given a cut C, a rule $X_1 \wedge ... \wedge X_k \Rightarrow X_{k+1} \wedge ... \wedge X_l$ is *aggregated* by performing the following *syntactic* transformation:

$$\begin{aligned} cut_C(X_1 \wedge ... \wedge X_k \Rightarrow X_{k+1} \wedge ... \wedge X_l) &= \\ cut_C(X_1) \wedge ... \wedge cut_C(X_k) \Rightarrow cut_C(X_{k+1}) \wedge ... \wedge cut_C(X_l) \end{aligned}$$

where $cut_C(X_i)$ maps each leaf node of the attribute hierarchy into its corresponding cut node as described in Step 1 above. The resulting rule is called an *aggregated rule*.

Since several different leaf nodes can have the same cut node, sometimes after aggregating a rule we can get multiple instances of the same aggregated attribute in the body or in the head of the rule. In this case we simply eliminate those extra instances of an attribute. Consider, for example, the rule $A2 \wedge A3 \wedge A4 \Rightarrow A5$. By applying cut (c) from Figure 3.1 to this rule, we will get the aggregated rule $A2 \wedge A7 \wedge A7 \Rightarrow A7$, and by removing duplicate terms $A7$ in the body of the rule we finally get $A2 \wedge A7 \Rightarrow A7$.[2] Given a cut, the computational complexity of a single rule aggregation is linearly proportional to the size of the rule (i.e., total number of attributes in the rule), as will be described later.

3. **Grouping rules**. Given a cut C, we can group a set of rules S into groups by applying C to every rule in S as described in Step 2 above. When a cut is applied to a set of rules, different rules can be mapped into the same

[2]Note that, while the just obtained aggregated rule $A2 \wedge A7 \Rightarrow A7$ may look like a tautology, it is not. As mentioned above, aggregated rules are obtained from the originally discovered rules using purely syntactic transformations. Therefore, the above mentioned aggregated rule does not make any logical statements about the relationship between attributes $A2$ and $A7$ in the given data, but simply denotes the *class* of rules of the particular syntactic structure.

Initial rule set S	Rule groups obtained from rule set S using cuts :					
	cut 3(b)		cut 3(c)		cut 3(d)	
$A1 \Rightarrow A3$	$A6 \Rightarrow A3$	(3)	$A7 \Rightarrow A7$	(2)	$A6 \Rightarrow A7$	(3)
$A1 \wedge A2 \Rightarrow A3$	$A3 \wedge A6 \Rightarrow A5$	(2)	$A2 \wedge A7 \Rightarrow A1$	(2)	$A6 \wedge A7 \Rightarrow A7$	(3)
$A1 \wedge A2 \wedge A3 \Rightarrow A5$	$A3 \Rightarrow A5$	(1)	$A2 \wedge A7 \Rightarrow A7$	(2)	$A6 \wedge A7 \Rightarrow A6$	(2)
$A2 \wedge A3 \Rightarrow A4$	$A3 \wedge A5 \Rightarrow A4$	(1)	$A1 \Rightarrow A7$	(1)	$A7 \Rightarrow A7$	(2)
$A2 \wedge A3 \Rightarrow A5$	$A3 \wedge A6 \Rightarrow A4$	(1)	$A2 \Rightarrow A7$	(1)		
$A2 \Rightarrow A3$	$A4 \wedge A6 \Rightarrow A6$	(1)	$A1 \wedge A2 \Rightarrow A7$	(1)		
$A2 \wedge A4 \Rightarrow A1$	$A5 \wedge A6 \Rightarrow A6$	(1)	$A1 \wedge A2 \wedge A7 \Rightarrow A7$	(1)		
$A3 \Rightarrow A5$						
$A2 \wedge A5 \Rightarrow A1$						
$A3 \wedge A5 \Rightarrow A4$						

Figure 3.2: Grouping a Set of Rules Using Several Different Cuts from Figure 3.1 (the number of rules in groups is specified in parentheses).

aggregated rule. For example, consider rules $A2 \wedge A3 \wedge A4 \Rightarrow A5$ and $A2 \wedge A5 \Rightarrow A3$. After applying cut (c) from Figure 3.1 to both of them, they are mapped into the same rule $A2 \wedge A7 \Rightarrow A7$. More generally, we can group a set of rules based on the cut C as follows. Two rules R_1 and R_2 belong to the same group *if and only if* $cut_C(R_1) = cut_C(R_2)$. Naturally, two different aggregated rules represent two disjoint groups of rules. As an example, Figure 3.2 presents the results of grouping a set of rules based on the attribute hierarchy and several different cuts shown in Figure 3.1.

The grouping algorithm described above allows the user to group rules into sets of similar rules, where similarity is defined by the expert who selects a specific cut of the attribute hierarchy. Moreover, instead of examining and validating individual rules inside each group, the user can examine the group of these rules as a whole based on the aggregated rule (that is common for all the rules in the group) and decide whether to accept or reject *all* the rules in that group at once based on this aggregated rule.

In addition, note that aggregated rules produced by the grouping algorithm can be expressed as cardinality predicates. E.g., aggregated rule $A6 \Rightarrow A3$ (taken from Figure 3.2) can be expressed as:

$$C^{1,2}_{A6}[body] \wedge C^{0}_{\overline{A6}}[body] \wedge C^{1}_{A3}[head] \wedge C^{0}_{\overline{A3}}[head]$$

Therefore, grouping algorithm can be used as a *generator* of potential validation operators.

3.2.3 Extended Attribute Hierarchies

So far, we assumed that the leaves in the attribute hierarchies are specified by the attributes of the data set. However, we also consider the case when attribute hierarchies include *values* and *aggregated values* of attributes from the data set. For example, assume that a data set has attribute *Month*. Then Figure 3.3 presents an attribute hierarchy with 12 values as the leaves representing specific months of the year that are grouped together into four aggregated values: *winter*, *spring*, *summer*, and *fall*.

For these extended hierarchies, cuts can include not only attribute and aggre-

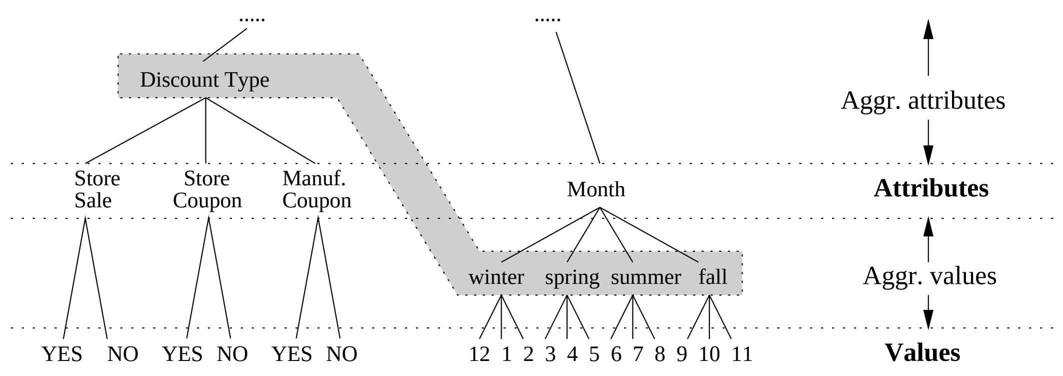


Figure 3.3: A fragment of attribute hierarchy which includes attribute values.

gated attribute nodes, but also value and aggregated value nodes. For example, consider the extended attribute hierarchy presented in Figure 3.3 that includes 12 values for the attribute *Month* and the boolean values for the attributes *StoreSale*, *StoreCoupon*, and *ManufCoupon*. Also consider the cut from Figure 3.3 specified with a shaded line, and the following three rules: (1) *Month*=3 $\Rightarrow$ *StoreSale*=YES, (2) *Month*=5 $\Rightarrow$ *ManufCoupon*=NO, (3) *Month*=10 $\Rightarrow$ *StoreSale*=YES. The cut presented in Figure 3.3 maps rules (1) and (2) into the same aggregated rule *Month=spring* $\Rightarrow$ *DiscountType*. However, rule (3) is mapped into a different aggregated rule *Month=fall* $\Rightarrow$ *DiscountType* by the cut. Therefore rule (3) will be placed into a different group than rules (1) and (2).

3.3 Implementation Issues and Computational Complexity

The grouping approach based on attribute hierarchies provides a flexible way for the expert to group rules according to the granularity important to that expert.

This provides the expert with the ability to evaluate larger or smaller number of groups of similar rules based on his or her preferences and needs. Moreover, an efficient algorithm that implements the grouping approach has been developed and is presented in Figure 3.4. The procedure `GROUP` performs the grouping using a single pass over the set of discovered rules (the `foreach` loop statement in lines 3-7 in Figure 3.4). For each rule r in the input rule set R (line 3) we compute its aggregated rule r' using the procedure `AGGR_ATTRS` (lines 5-6).

The procedure `AGGR_ATTRS` (lines 11-15) performs the aggregation of a set of attributes. Using the mapping cut_C, each element of an attribute set is aggregated in constant time. Moreover, since the attribute set `AttrSet` is implemented as a hash table, an insertion of an aggregated attribute into the resulting set A' (line 13, inside the loop) also takes constant time. Therefore, the total running time of the procedure `AGGR_ATTRS` is linear in the size of the attribute set.

As the result, the running time of a rule aggregation (lines 5-6) is linear in the size of the rule (i.e., total number of attributes in the body and the head of the rule). Also, since the group set `GroupSet` is implemented as a hash tree data structure (similar to the one described by [75]), an insertion of a group into the resulting group set G (line 7) is also linear in the size of the rule. Consequently, the running time of the whole grouping algorithm is linear in the total size of the rules to be grouped. Note also that, besides the computational space needed to store the resultant rule groups, the algorithm uses virtually no additional computational space (except for several local variables).

In summary, the grouping algorithm presented in Figure 3.4 scales up well, which is very important for personalization applications dealing with very large numbers

```
GROUP ( RuleSet $R$, Map $cut_C$ ) {
    GroupSet $G := \emptyset$;
    foreach $r$ from $R$ {
        $r'$ := new Rule;
        $r'.body$ := AGGR_ATTRS($r.body$, $cut_C$);
        $r'.head$ := AGGR_ATTRS($r.head$, $cut_C$);
        $G$ := $G \cup r'$;
    }
    return $G$;
}
AGGR_ATTRS( AttrSet $A$, Map $cut_C$ ) {
    AttrSet $A' := \emptyset$;
    foreach $a$ from $A$ { $A'$ := $A' \cup cut_C[a]$; }
    return $A'$;
}
```

Figure 3.4: Algorithm for similarity-based rule grouping.

of rules.

3.4 Benefits of the Proposed Grouping Method

The proposed rule grouping method has the following distinguishing features that make it useful in the data mining applications requiring post-analysis of large numbers of discovered rules. First, unlike the traditional clustering methods [28], where the user has only a limited control over the structure and sizes of resulting clusters, an expert has an explicit control over the granularity of the resulting rule groups in our approach. That is, a domain expert can specify different aggregation levels in the attribute hierarchy, which allows grouping the discovered rules according to the granularity important to that expert (i.e., depending on how he or she wants to explore the rules). This property is very useful in the applications dealing with very large numbers of discovered rules (e.g., millions), because traditional clustering methods may still generate an unmanageable number of clusters. Moreover, the proposed method allows the domain expert to incorporate the domain knowledge into the grouping process by specifying an attribute hierarchy.

Second, the rule groups (denoted by different aggregated rules) obtained by the proposed rule grouping method are *equivalence classes*, since any two rules R1 and R2 belong to the same rule class if and only if they are mapped into the same aggregated rule, given a specific gene aggregation level. This means that we can determine what rule class the particular rule R belongs to based solely on the structure of rule R. This property makes the proposed rule grouping method consistent and predictable, since the domain expert knows to what class a rule belongs *regardless* of what other discovered rules are. This is in contrast to some of the traditional distance-based clustering methods, where any two rules may or may not be in the same cluster depending on the other discovered rules.

Third, one of the limitations of some of the other rule grouping approaches (e.g., [83]), lies in that it is not clear how to describe concisely the resulting rule cluster to the end-user for the purpose of evaluation, since rules belonging to the same cluster may have substantially different structures. In contrast, in our proposed grouping approach that is based on attribute hierarchies, every rule cluster (group) is uniquely represented by its aggregated rule (common to all rules in that cluster), that is both concise and descriptive.

Finally, the proposed rule grouping method works with large numbers of attributes, both numerical and categorical. Also, it scales up well. In fact, using lookup tables for attribute hierarchies and hash table-based structures for storing aggregated rules, the grouping algorithm is linear in the total size of the rules to be grouped. This is especially important in applications dealing with very large numbers of rules.

3.5 Related Work

There have been related approaches to rule grouping proposed in the literature [51, 86] that consider association rules in which both numeric and categorical attributes can appear in the body and only categorical attributes in the head of a rule. However, [51] take a more restrictive approach by allowing only two numeric attributes in the body and one categorical attribute in the head of a rule, whereas [86] allow any combination of numeric and categorical attributes in the body and one or more categorical attributes in the head of a rule. Both of the approaches merge adjacent intervals of numeric values in a bottom-up manner, where [51] utilize a clustering approach to merging and [86] maximize certain interestingness measures during the merging process. It is interesting to observe that interval merging can also be supported in our rule grouping operator by letting a domain expert specify the cuts at the value and aggregated-value levels of the attribute hierarchy (as shown in Figure 3.3).

The two approaches [51] and [86] would work well for medium-sized problems having moderately large number of rules (e.g., 10's of thousands). However, in large-scale applications, such as personalization or bioinformatics applications (as described in Chapters 5 and 6), that can generate millions of rules, the two approaches would still produce too many clusters to be of practical use. Therefore, in order to allow the domain expert to validate very large numbers of rules within a reasonable amount of time, such applications require more powerful grouping capabilities that go beyond the interval merging techniques for attribute values. Therefore, our approach differs from [51, 86] in that it allows the grouping of rules with different structures, at different levels of the attribute hierarchy and also not only for numerical but for categorical attributes as well. Moreover, the domain expert has the flexibility to specify the relevant cuts in the attribute hierarchy, whereas the interval merging approaches do the merging automatically based on the built-in heuristics.

Still another related approach to grouping is proposed by [83] where a distance between two association rules is defined as the number of transactions on which two rules differ. Using this distance measure, [83] group all the rules into appropriate clusters. One of the limitations of this approach lies in that the distance measures selected for rule clustering are somewhat arbitrary. Moreover, it is not clear how to describe concisely the rule cluster to the user for the purpose of evaluation, since rules belonging to the same cluster may have substantially different structures. In contrast, in our proposed similarity-based grouping approach every rule cluster is uniquely represented by its aggregated rule (common to all rules in that cluster), that is concise and descriptive.

Still other approaches related to grouping are presented in [55, 36], where [55] describes a method for extracting "most important" (direction-setting) rules from large numbers of discovered rules, and [36] presents data mining methods using two-dimensional optimized association rules.

3.6 Experiments

We tested the proposed grouping method on a "real-life" personalization application that analyzes individual customer responses to various types of promotions, including advertisements, coupons, and various types of discounts. The application included data on 1903 households that purchased different types of non-alcoholic beverages over a period of one year. The data set contained 21 fields characterizing purchasing transactions. The whole data set contained 353,421 records (on average 186 records per household). We ran an association rule discovery algorithm on the transactional data for each of the 1903 households and generated 1,022,812 association rules in total.

	Cut 5(a)			Size
(1)	$Time\_related$	$\Rightarrow$	$Product$	4431
(2)	$Deal\_type$	$\Rightarrow$	$Product$	446
(3)	$Store$	$\Rightarrow$	$Deal\_type$	245
(4)	Ad_place	$\Rightarrow$	$Product$	38
(5)	$Store \wedge Ad\_place$	$\Rightarrow$	$Product$	24

	Cut 5(b)			Size
(6)	Day_of_week	$\Rightarrow$	$Product$	2682
(7)	$Store$	$\Rightarrow$	$Product$	2193
(8)	$Season$	$\Rightarrow$	$Product$	1435
(9)	$Store$	$\Rightarrow$	$Total\_price$	1143
(10)	$Day\_of\_week \wedge Season$	$\Rightarrow$	$Product$	314
(11)	$Store\_coupon$	$\Rightarrow$	$Store$	191

Table 3.1: Sample rule groups produced by the cuts from Figure 3.5.

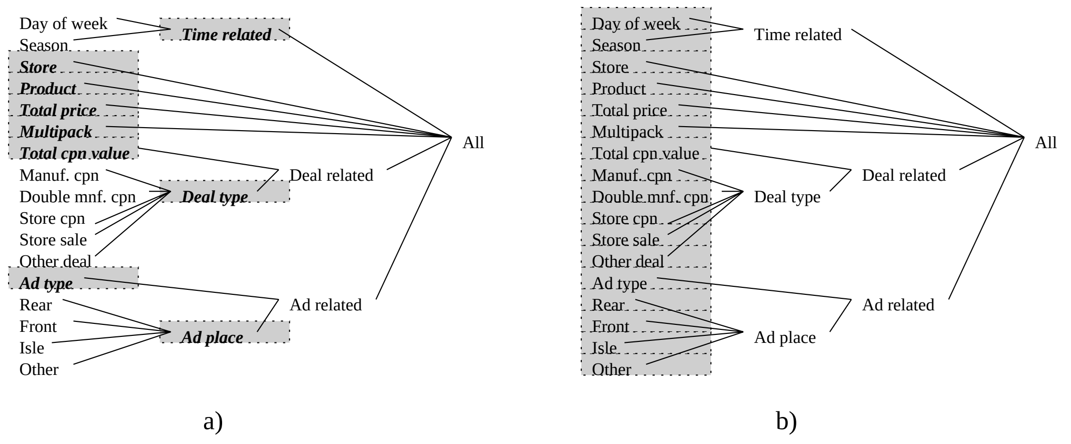


Figure 3.5: Fragment of an attribute hierarchy used in a marketing application.

Figure 3.5 shows the fragment of an attribute hierarchy[3] that was used in this application along with the two sample cuts that we used for rule grouping. Also, Table 3.1 presents some sample groups generated with these cuts. Moreover, Table 3.2 summarizes the grouping statistics for the same two cuts from Figure 3.5.

As we can see from Figure 3.2, taking the "coarser" of the two cuts (i.e., cut that is further from the leaf nodes - in this particular case it is cut (a)) reduces the number of groups substantially (2,742 vs. 24,920 groups). Moreover, the percentage of rules in the largest N groups also increases substantially (e.g. from 3.7% to 17.8% for $N = 10$) along with the sizes of the groups (e.g. 23,891 vs. 5,022 for the largest groups respectively). Therefore, when using the coarser cut, the user has to deal with fewer groups (e.g., browse through and evaluate them) than when using the finer cut. However, it is more difficult to validate coarser groups since their syntactic

[3]In order not to clutter the diagram in Figure 3.5, the attribute hierarchy does not contain levels representing attribute values and aggregate attribute values.

Various results	Cut 5(a)	Cut 5(b)
# of groups	2,742	24,920
# of rules in the biggest group	23,891	5,022
# of rules in 10 biggest groups	181,681 (17.8%)	38,336 (3.7%)
# of rules in 20 biggest groups	320,018 (31.3%)	68,241 (6.7%)
# of rules in 50 biggest groups	568,570 (55.6%)	151,890 (14.9%)

Table 3.2: Summary of the groupings produced by the cuts from Figure 3.5.

representations are not as informative as the ones of finer groups.

In addition, Figure 3.6 shows the percentage of all the rules contained in N largest groups for both cuts. As expected, the coarser cut produces groups containing many more rules in their largest N groups than the finer cut. For example, 100 largest groups (out of the total of 2,742 groups) produced by the coarser cut contain 75% of all the discovered rules. Also, Figure 3.7 illustrates how many groups of each particular size were generated, based on the same two cuts from Figure 3.5.

3.7 Discussion

In this chapter, we presented an approach to grouping and summarizing large numbers of discovered rules into classes based on user-specified attribute hierarchies. Then these groups of rules can be browsed by the user during the post-analysis stage of the data mining process in order to evaluate their relevance. This method allows the user to control the grouping process by selecting appropriate cuts in the attribute hierarchy. Moreover, it works well not only with the numeric data as the

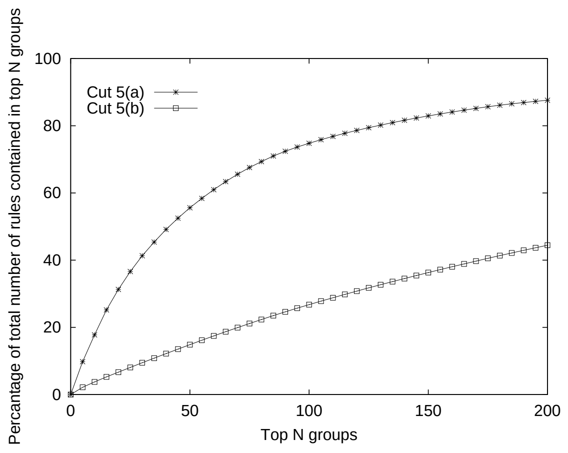


Figure 3.6: Comparison of the groups produced by two cuts from Figure 3.5.

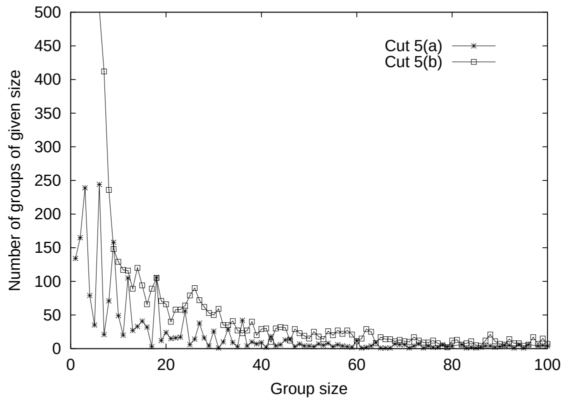


Figure 3.7: Comparison of the group sizes produced by two cuts from Figure 3.5.

clustering methods do, but equally well with the categorical data. Also, it can be determined from the structure of the rule itself to which group it belongs, making this method more consistent and predictable than some standard clustering methods. Finally, our method is efficient and scalable since it is linear in the total size of the rules to be grouped.

The user can apply the grouping method presented in this paper in an interactive manner during the post-analysis stage of the data mining process in order to evaluate relevance of discovered rules. Moreover, the user can recursively examine some of the large groups of similar rules by specifying finer cuts for these large groups. This process of iteratively specifying cuts, examining the resulting groups and specifying additional cuts for some of the selected groups of rules can continue until the user completes the examination process.

In its current version of our grouping method, we do not provide statistical properties of the formed groups of rules beyond simple counts. One of the interesting extensions of this work is to develop a set of statistics that would characterize these groups of rules in the ways meaningful to the end-users, and we are planning to work on supporting such statistics in the future.

Chapter 4

Optimizing Validation Process

When large amounts of data need to be validated, the performance of the validation process is crucial. Therefore, as described in previous chapters, in our research we have focused on making the validation tools and methods (e.g., cardinality predicates, grouping algorithm) as scalable as possible. In this chapter, however, we tried to look at a different aspect of validation performance, i.e., the performance of the validation sequence as a whole (as opposed to the performance of individual validation operators). This issue may be important in the applications where, for example, there is a continuous stream of data to be validated. In such cases, the domain expert may validate a sample of data, thus generating some validation sequence. After the sequence is generated, it could perform the subsequent validation of the continuous data stream by itself, i.e., without the domain expert. Then, the question is: can we optimize this sequence to be as efficient as possible, while retaining the exact validation behavior initially encoded by the domain expert? We address this issue in this chapter.

4.1 Sequence Optimization Problem Formulation

In this section, we will formulate the problem of validation sequence optimization. In fact, we will formulate this problem for general validation sequences and not just the ones that validate set-based data (i.e., sets or rules).

4.1.1 General Validation Sequences

As mentioned in Section 1.2, we define general validation sequences as follows. Let's assume that the domain expert has to validate input dataset D, where $D \subseteq \mathcal{E}$ is a set of certain input elements. We will assume that $\mathcal{E}$ is a finite set that contains all possible input elements, i.e., all possible inputs that may be present in datasets to be validated. For example, in the case of itemset validation, $\mathcal{E} = \text{Powerset}(\mathcal{I})$, where $\mathcal{I}$ is a base set (as described earlier).

Furthermore, the domain expert has a set $\mathcal{L}$ of possible labels (let's call them L_1, L_2, $\ldots$, L_n) to be used to label (validate) individual input elements $e \in D$. Also, let $\mathcal{P}$ be the set of all possible predicates for validating input data from $\mathcal{E}$, i.e., let $\mathcal{P}$ contain all predicates p of the form:

$$p : \mathcal{E} \longrightarrow \{\text{True}, \text{False}\} \tag{4.1}$$

As in the case of set and rule validation, the validation of $D \subseteq \mathcal{E}$ is achieved by the domain expert by producing a sequence of validation operators. Let's denote this sequence s. In other words, let $s =< o_1, o_2, \ldots, o_k >$, where $k > 0$ and each o_i represents a validation operator, i.e., $o_i = (l_i, p_i)$, $l_i \in \mathcal{L}$ and $p_i \in \mathcal{P}$. As a result of this validation, the input set D is divided into $n+1$ pairwise disjoint sets V_1, V_2, $\ldots$, V_n, and U, where each V_i represents the subset of D that was labeled with L_i, and U denotes the subset of D that remains unlabeled after the validation process.[1] We will denote $VI_s(D) := (V_1, V_2, \ldots, V_n)$ (validated input elements) and $UI_s(D) := U$ (unvalidated input elements).

Then, informally, the validation sequence optimization problem could be formulated as follows. Given an expert-specified sequence s, find the "best" sequence $s*$ among all the sequences "equivalent" to s (i.e., sequences that have exactly the same validation "behavior" as s). In other words, $s*$ should be the "best" sequence among those which produce the same results as s for any given dataset D. In the rest of Section 4.1 we will define precisely the notion of validation sequence equivalence as well as the sequence performance metric to be used in the optimization problem.

4.1.2 Equivalence of Validation Sequences

Here we introduce the notion of *equivalent* validation sequences.

Definition 4 (Equivalent Validation Sequences) *Validation sequences s and s' are equivalent if and only if $VI_s(D) = VI_{s'}(D)$ for every input D. If s and s' are equivalent, we will denote it as $s \sim s'$. If s and s' are not equivalent, we will denote it as $s \nsim s'$.*

Alternatively, the equivalence of a validation sequence can also be defined as follows.

Lemma 2 $s \sim s' \iff VI_s(\mathcal{E}) = VI_{s'}(\mathcal{E})$.

[1]That is, U contains the input elements that were not matched by any predicates in the validation sequence.

▶ Follows directly from the equivalence definition above, since $(\forall D \subseteq \mathcal{E})\ VI_s(D) = VI_{s'}(D) \iff VI_s(\mathcal{E}) = VI_{s'}(\mathcal{E})$. ◀

The equivalence of validation sequences can be viewed as a *binary relation* on the set of all possible validation sequences. Let's denote this relation $R_\sim$. Note, that $R_\sim$ is a *true* equivalence relation, since it is reflexive, symmetric, and transitive, as shown below.

Lemma 3 *Relation $R_\sim$ is a true equivalence relation.*

▶ *$R_\sim$ is reflexive.* Obviously, $s \sim s$ for all s, since $VI_s(\mathcal{E}) = VI_s(\mathcal{E})$.

$R_\sim$ is symmetric. For any validation sequences s and s', if $s \sim s'$ then $s' \sim s$, because $VI_s(\mathcal{E}) = VI_{s'}(\mathcal{E})$ obviously implies $VI_{s'}(\mathcal{E}) = VI_s(\mathcal{E})$.

$R_\sim$ is transitive. For any validation sequences s, s', and s'', if $s \sim s'$ and $s' \sim s''$, then $s \sim s''$, because $VI_s(\mathcal{E}) = VI_{s'}(\mathcal{E})$ and $VI_{s'}(\mathcal{E}) = VI_{s''}(\mathcal{E})$ implies $VI_s(\mathcal{E}) = VI_{s''}(\mathcal{E})$. ◀

Example 3 Assume that we are dealing with an application for validating frequent shopping itemsets (i.e., shopping baskets). That is, assume that $\mathcal{E}$ contains all possible shopping baskets, $\mathcal{P}$ contains all cardinality predicates, and $\mathcal{L} = \{\texttt{Accept}, \texttt{Reject}\}$. Let $s = <(\texttt{Accept}, C^1_{Bread})$, $(\texttt{Accept}, C^1_{Milk}) >$ and let $s' = <(\texttt{Accept}, C^1_{Milk})$, $(\texttt{Accept}, C^1_{Bread}) >$. Both sequences have two validation operators and clearly $s \neq s'$. In fact, s' is a reverse of s. However, $s \sim s'$, since clearly both s and s' will validate (i.e., label with `Accept`) the same inputs, given any input dataset.

4.1.3 Sequence Optimization Problem and Related Issues

Now that we have defined the notion of validation sequence equivalence, we can formulate the validation sequence optimization problem more precisely:

$$s* = \arg\min_{s' \sim s} cost(s') \tag{4.2}$$

where $cost(s)$ is some performance measure for validation sequences.

In order to be able to address the problem specified by Equation 4.2, we should address the following issues:

- Exactly how should the $cost(s)$ function be defined?
- How do we find the sequences that are equivalent to s? In other words, what is the *search space* of our optimization problem?
- How to find the optimal sequence efficiently? That is, even if we know how to determine all possible equivalent sequences and how to calculate the cost function for each of them, there may be too many of these sequences, and therefore, the exhaustive search may not be practical for solving the optimization problem (4.2).

4.1.4 Defining the Cost of Validation Sequence

One natural way to define the cost of a validation sequence would be through the performance of a validation sequence, e.g., the amount of time it takes to perform validation. However, we would like to note that the cost of validation sequence may highly depend on the input dataset that the sequence is being used on. For example, on the one hand, validation sequence s might be able to validate the entire dataset D_1 with its first validation operator (i.e., if all elements of D_1 match the predicate of the first operator), therefore the subsequent validation operators would not even have to be invoked. On the other hand, the same validation sequence s might not be able to validate any elements of dataset D_2. That is, all validation operators would have to be invoked on all input elements of D_2.

Therefore, we define the validation sequence optimization problem in the context of a specific dataset D. That is, given an expert-specified validation sequence s and an input dataset D, which of the sequences that are equivalent to s would have the best performance validating dataset D? Ideally, we would like to use the running time as the performance measure for validation sequences. While we generally would be able to compute the running time of the expert-specified sequence s (i.e., during the initial validation process), it obviously would be difficult to estimate running time of other sequences theoretically. However, we can overcome this difficulty with certain simplifying assumptions.

We will assume that it takes a certain *fixed* time for an arbitrary validation operator $o = (l, p)$ to validate an arbitrary input element $e \in D$, i.e., to check whether e satisfies predicate p. Based on this assumption, we can define the cost function (given a validation sequence s and input D) to be the total number of predicate/input satisfaction checks performed by s to validate input D. More specifically, assume that sequence s consists of k validation operators o_1, o_2, $\ldots$, o_k and each of the operators o_i validated n_i number of elements from input dataset D. That is, operator o_1 checked all $|D|$ inputs and validated n_1 of them. Subsequently, o_2 checked the remaining $|D| - n_1$ inputs and validated n_2 of them, and so on. Then, the cost of validating D using sequence s can be defined as:

$$\begin{aligned} cost(s,D) &= |D| + (|D| - n_1) + (|D| - n_1 - n_2) + \ldots + (|D| - \sum_{j=1}^{k-1} n_j) \\ &= k|D| \; - \; \sum_{i=1}^{k-1} (k-i) \; n_i \qquad (4.3) \end{aligned}$$

Furthermore, let's define $cost_0(s, D)$ to be the worst possible cost scenario, when not a single input from D is validated by s, i.e., all $n_i = 0$. Therefore, $cost_0(s, D) = k|D|$. Then, we can define *benefit* function as follows:

$$\begin{aligned} benefit(s,D) &= cost_0(s,D) \; - \; cost(s,D) \\ &= \sum_{i=1}^{k-1} (k-i) n_i \qquad (4.4) \end{aligned}$$

Based on the above definitions of cost and benefit functions, our optimization problem is now formulated as:

$$s* = \arg\min_{s' \sim s} cost(s', D) = \arg\max_{s' \sim s} benefit(s', D)$$

Now that we have defined the *cost* and *benefit* functions, we have to decide how we will be searching for equivalent sequences. An interesting question is: in what ways we are allowed to change sequence s, so that the newly obtained sequence s' remains equivalent to s? Example 3 from Section 4.1.2 can provide some intuition. In that example, we had two equivalent sequences s and s', where s' was a *permutation* of s. That is, s' had exactly the same operators as s, but not necessarily in the same order. Therefore, given validation sequence s, we will be searching for the solution to our optimization problem among the permutations s' of sequence s such that $s \sim s'$.

In the next section, we formally introduce the concept of validation sequence permutation.

4.1.5 Permutations of Validation Sequences

Definition 5 (Permutation) *Let s and s' be validation sequences, i.e., $s =< o_1, o_2, \ldots, o_k >$ and $s' =< o'_1, o'_2, \ldots, o'_k >$. Sequence s' is called a permutation of sequence s if and only if the sets $\{o_1, o_2, \ldots, o_k\}$ and $\{o'_1, o'_2, \ldots, o'_k\}$ are equal.*

In other words, s' is a permutation of s if it contains the exact same validation operators, but not necessarily in the same order.

Let $s =< o_1, o_2, \ldots, o_k >$ be a validation sequence and let u be some validation operator. We will say that s *contains* u (and denote $u \in s$) if there exists $x \in \{1, \ldots, k\}$ such that $u = o_x$. In such case we will say that s contains u at a position x and denote $pos_s(u) := x$.

Let u and v be validation operators contained in the validation sequence s, i.e., $u, v \in s$. Then we will say that u *precedes* v in sequence s if $pos_s(u) < pos_s(v)$. We will denote it as $u \prec_s v$.

Definition 6 (Permutation Distance) *Let s be a validation sequence, i.e., let $s =< o_1, o_2, \ldots, o_k >$. Let $s' =< o'_1, o'_2, \ldots, o'_k >$ be some permutation of s. We will define the distance between s and s' as the number of inversions between s and s', i.e., the number all distinct pairs of validation operators u, v, such that $u \prec_s v$ and $v \prec_{s'} u$. More precisely,*

$$dist(s, s') := |\{(u, v) : u \in s, v \in s, u \prec_s v, v \prec_{s'} u\}| \tag{4.5}$$

Definition 7 (Simple Permutation) *Let s be a validation sequence, and let s' be a permutation of s, such that $dist(s, s') = 1$. Then s' is called a simple permutation of s.*

In other words, validation sequence $s' =< o'_1, o'_2, \ldots, o'_k >$ is a simple permutation of $s =< o_1, o_2, \ldots, o_k >$ if and only if there exists $i \in \{1, \ldots, k-1\}$ such that $o_i = o'_{i+1}$ and $o_{i+1} = o'_i$, and for all j ($j \neq i$ and $j \neq i+1$) $o_j = o'_j$ holds true.

Lemma 4 *If s' is a permutation of s, then $UI_s(D) = UI_{s'}(D)$ for every input dataset D.*

▶ Let's assume otherwise, i.e., that there exists dataset D such that $UI_s(D) \neq UI_{s'}(D)$. Consequently, there must exist $e \in D$ such that $e \in UI_s(D)$ and $e \notin UI_{s'}(D)$. (The proof for the case $e \notin UI_s(D)$ and $e \in UI_{s'}(D)$ would be essentially the same.)

Since $e \notin UI_{s'}(D)$, e must be validated by some validation operator in s', say, operator u. However, because s' is a permutation of s, s contains all the operators that s' does, including u. Therefore, s contains a validation operator that validates e. Consequently, $e \notin UI_s(D)$. This is a contradiction, since we defined e to be such that $e \in UI_s(D)$.

Therefore, our initial assumption that there exists dataset D such that $UI_s(D) \neq UI_{s'}(D)$ was incorrect. ◀

Definition 8 (Safe Permutation) *Let s be a validation sequence. Then s' is a safe permutation of s if and only if s' is a permutation of s and $s \sim s'$.*

4.2 Equivalence of Sequence Permutations

In this section, we will describe several different equivalence relations on validation sequence permutations and will prove some of their fundamental properties.

4.2.1 Deriving Equivalence Criteria for Sequence Permutations

The following theorem states necessary and sufficient conditions for $s \not\sim s'$, where s' is a permutation of s.

Theorem 5 *Let $s =< (l_1, p_1), \ldots, (l_k, p_k) >$ and $s' =< (l'_1, p'_1), \ldots, (l'_k, p'_k) >$ be validation sequences, where s' is a permutation of s. Then $s \not\sim s'$ if and only if they contain a pair of validation operators $u = (l_u, p_u)$ and $v = (l_v, p_v)$ that satisfy <u>all</u> of the following conditions:*

1. *u precedes v in s, but v precedes u in s', i.e., $u \prec_s v$ and $v \prec_{s'} u$;*
2. *u and v have different labels, that is, $l_u \neq l_v$;*
3. *There exists an element $e \in \mathcal{E}$ such that the following Boolean expression is true:*

$$p_u(e) \quad \wedge \quad p_v(e) \quad \wedge \quad \bigwedge_{i=1}^{x-1} \neg p_i(e) \quad \wedge \quad \bigwedge_{j=1}^{y-1} \neg p'_j(e) \tag{4.6}$$

   where $x = pos_s(u)$ and $y = pos_{s'}(v)$.

▶ First, let's assume that validation sequences s and s' contain validation operators u and v that satisfy all three conditions described above. We will show that $s \not\sim s'$.

Based on the condition 3, there exists an input element e that satisfies the Boolean expression 4.6. Because this expression is a conjunction of several subexpressions, e satisfies each of these subexpressions. Based on this we derive the following.

Since both $p_u(e)$ and $\bigwedge_{i=1}^{x-1} \neg p_i(e)$ hold, we have that e satisfies predicate p_u (which is at the position x in s), but does not satisfy any of the predicates p_1, p_2,

$\dots$, p_{x-1}. These predicates are at positions 1, 2, $\dots$, $x-1$ respectively in sequence s, therefore all predicates that precede p_u in the validation sequence would not match e. Consequently, in the sequence s, e would be matched by predicate p_u and labeled with l_u. Obviously, $u \prec_s v$, since otherwise v (and not u, as we just showed) would match the input e in sequence s.

Similarly, since both $p_v(e)$ and $\bigwedge_{j=1}^{y-1} \neg p'_j(e)$ hold, we have that e satisfies predicate p_v (which is at the position y in s'), but does not satisfy any of the predicates p'_1, p'_2, $\dots$, p'_{y-1}. Therefore, in the sequence s', e would be matched by operator p_v and labeled with l_v. Obviously, $v \prec_{s'} v$, since otherwise u (and not v, as we just showed) would match the input e in sequence s'.

Since both $u \prec_s v$ and $v \prec_{s'} u$, condition 1 is satisfied automatically.

Based on the condition 2, $l_u \neq l_v$. Therefore, s would validate e differently than s'. Therefore, when $D = \{e\}$, we have $VI_s(D) \neq VI_{s'}(D)$. Hence, $s \not\sim s'$.

Now let's assume that $s \not\sim s'$. We will show that these sequences contain validation operators u and v that satisfy all three conditions mentioned above.

$s \not\sim s' \Rightarrow (\exists D)(VI_s(D) \neq VI_{s'}(D))$. Let's denote $VI_s(D) = (V_1, V_2, \dots, V_{|\mathcal{L}|})$ and $VI_{s'}(D) = (V'_1, V'_2, \dots, V'_{|\mathcal{L}|})$. Here V_i $(i = 1, \dots, |\mathcal{L}|)$ is a subset of D labeled with the label L_i by sequence s. Similarly, V'_i $(i = 1, \dots, |\mathcal{L}|)$ is a subset of D labeled with the label L_i by sequence s'. Since $VI_s(D) \neq VI_{s'}(D)$, we have that $(V_1, \dots, V_{|\mathcal{L}|}) \neq (V'_1, \dots, V'_{|\mathcal{L}|})$. Therefore, there exists i such that $V_i \neq V'_i$.

Since $V_i \neq V'_i$, let's assume (without loss of generality) that there exists an entity $e \in D$ such that $e \in V_i$, but $e \notin V'_i$. (It could also be $e \in V'_i$ and $e \notin V_i$, in which case the proof would be virtually the same as below.) Since $e \notin V'_i$, there exists $j \in \{1, \ldots, |\mathcal{L}|\}$ such that $i \neq j$ and $e \in V'_j$. Note that, e can not remain unvalidated by s', as shown in Lemma 4.

Because $e \in V_i$, there must exist a validation operator $u = (L_i, p_u)$ in the sequence s (say, at the position x, i.e., $pos_s(u) = x$) that validates e (i.e., $p_u(e)$ is `True`), but none of the preceding operators do (i.e., $\neg p_i(e)$ for all $i = 1, \ldots, x-1$). Therefore, both $p_u(e)$ and $\bigwedge_{i=1}^{x-1} \neg p_i(e)$ hold.

Similarly, because $e \in V'_j$, there must exist a validation operator $v = (L_j, p_v)$ in the sequence s' (say, at the position y, i.e., $pos_s(v) = y$) that validates e (i.e., $p_v(e)$ is `True`), but none of the preceding operators do (i.e., $\neg p'_j(e)$ for all $j = 1, \ldots, y-1$). Therefore, both $p_v(e)$ and $\bigwedge_{i=1}^{y-1} \neg p'_j(e)$ hold.

The previous two paragraphs combined show that condition 3 holds. Condition 2 also holds, since u and v operators described above have different labels (i.e., L_i and L_j, $i \neq j$). Finally, condition 1 holds as well, because the same input element e is validated by u in sequence s and by operator v in sequence s', which would be impossible when either of operators u and v precedes the other one in both sequences, since they both match e. ◀

The following corollary states necessary and sufficient conditions for $s \sim s'$, where s' is a permutation of s.

Corollary 6 *Let $s =< (l_1, p_1), \ldots, (l_k, p_k) >$ and $s' =< (l'_1, p'_1), \ldots, (l'_k, \ldots, p'_k) >$ be validation sequences, where s' is a permutation of s. Then $s \sim s'$ if and only if* <u>*every*</u> *possible pair of validation operators $u = (l_u, p_u)$ and $v = (l_v, p_v)$ (i.e., $u \in s$, $v \in s$, $u \neq v$) must satisfy* <u>*at least one*</u> *of the following conditions:*

1. *Either u precedes v in both s and s', or v precedes u in both s and s';*

2. *u and v have the same label, that is, $l_u = l_v$;*

3. *For all possible input elements $e \in \mathcal{E}$, the following Boolean expression is true:*

$$\neg(p_u(e) \quad \wedge \quad p_v(e)) \quad \vee \quad \bigvee_{i=1}^{x-1} p_i(e) \quad \vee \quad \bigvee_{j=1}^{y-1} p'_j(e) \tag{4.7}$$

where $x = pos_s(u)$ and $y = pos_{s'}(v)$.

▶ Follows directly from Theorem 5, which states conditions that are necessary and sufficient for s and s' to be non-equivalent. Simply by taking logical negation of these conditions, we obtain the necessary and sufficient conditions for $s \sim s'$. ◀

The following corollary states necessary and sufficient conditions for $s \sim s'$, where s' is a *simple* permutation of s.

Corollary 7 *Let $s =< o_1, \ldots, o_k >$ be a validation sequence where $o_i = (l_i, p_i)$, and let $s' =< o'_1, \ldots, o'_k >$ be a simple permutation of s. That is, $(\exists!\ x \in \{1, \ldots, k-1\})\ ((o_x = o'_{x+1}) \wedge (o_{x+1} = o'_x))$, and also $o_i = o'_i$ for all $i \in \{1, \ldots, k\}$ such that $i \neq x$ and $i \neq x+1$.*

Then $s \sim s'$ if and only if <u>at least one</u> of the following two conditions is satisfied:

1. *o_x and o_{x+1} have the same label, that is, $l_x = l_{x+1}$;*

2. *For all possible input elements $e \in \mathcal{E}$, the following Boolean expression is true:*

$$\neg(p_x(e) \quad \wedge \quad p_{x+1}(e)) \quad \vee \quad \bigvee_{i=1}^{x-1} p_i(e) \tag{4.8}$$

▶ Since s' is a permutation of s, Corollary 6 gives us the necessary and sufficient conditions for $s \sim s'$. We will show that when s' is a simple permutation of s, these conditions are equivalent to the two conditions above.

More specifically, when s' is a *simple* permutation of s, there is only one inversion of operator precedence in s' with respect to s. Therefore, all pairs of validation operators $u, v \in s$ except one (i.e., o_x and o_{x+1}) automatically satisfy condition 1 of Corollary 6. Consequently, $s \sim s'$ if and only if the remaining pair of validation operators, i.e., o_x and o_{x+1}, satisfies at least one of the remaining two conditions of Corollary 6.

Therefore, o_x and o_{x+1} must either have the same label, or the Equation 4.7 must be satisfied, i.e., $\neg(p_x(e) \wedge p_{x+1}(e)) \vee \bigvee_{i=1}^{x-1} p_i(e) \vee \bigvee_{j=1}^{y-1} p'_j(e)$ must be `True`, where $x = pos_s(o_x)$ and $y = pos_{s'}(o_{x+1})$. Because $o_x = o'_{x+1}$ and $o_{x+1} = o'_x$, we have that $y = pos_{s'}(o_{x+1}) = pos_{s'}(o'_x) = x$. We also know that $o_i = o'_i$ when $i < x$. Therefore Equation 4.7 in our case can be rewritten as $\neg(p_x(e) \wedge p_{x+1}(e)) \vee \bigvee_{i=1}^{x-1} p_i(e)$. ◀

4.2.2 "Connectedness" of Equivalent Sequence Permutations

In this section, we will prove some facts that will provide some understanding about the "space" of equivalent validation sequence permutations.

First, we will prove a simple lemma that will be useful in the subsequent proofs.

Lemma 8 *Let $s =< o_1, o_2, \ldots, o_k >$ be a validation sequence. Let s' be some permutation of s. Then, for every pair of validation operators o_i and o_j such that $o_i \prec_s o_j$ (i.e., $i < j$), but $o_j \prec_{s'} o_i$, there exists x such that $i \le x \le j-1$ and $o_{x+1} \prec_{s'} o_x$.*

▶ *Case 1:* $j - i = 1$. Let $x := i$. Therefore, $o_x = o_i$ and $o_{x+1} = o_j$. We have that $o_{x+1} \prec_{s'} o_x$, since $o_j \prec_{s'} o_i$.

Case 2: $j - i > 1$. Let's assume otherwise, that for every x such that $i \le x \le$

$j-1$, we have $o_x \prec_{s'} o_{x+1}$. Therefore we have, that in the sequence s': o_i precedes o_{i+1}, o_{i+1} precedes o_{i+2}, ..., o_{j-1} precedes o_j. Hence, by transitivity, $o_i \prec_{s'} o_j$. This is a contradiction, since o_j precedes o_i in s'. Therefore, our assumption that for every x such that $i \leq x \leq j-1$, $o_x \prec_{s'} o_{x+1}$ was incorrect. ◀

Theorem 9 *Let $s =< o_1, o_2, \ldots, o_k >$ be a validation sequence. Let s' be some safe permutation of s such that $dist(s, s') = d (d \geq 1)$. Then there exists a sequence s'' that is a safe simple permutation of s, such that $s' \sim s''$ and $dist(s', s'') = d - 1$.*

▶ Let x be the *largest* number from the set $\{1, \ldots, k\}$, such that $o_{x+1} \prec_{s'} o_x$. Then, let's construct the sequence $s'' =< o''_1, \ldots, o''_k >$ as follows. Let $o''_i := o_i$, for all i, such that $1 \leq i \leq x-1$ or $x+2 \leq i \leq k$. Also, let $o''_x = o_{x+1}$ and $o''_{x+1} = o_x$. Essentially, sequence s'' is the same as s except for o_x and o_{x+1} that are swapped. Obviously, s'' is a simple permutation of s, thus $dist(s, s'') = 1$.

Now we will show that $s \sim s''$. Since s'' is a simple permutation of s, Corollary 7 gives us two conditions to be satisfied in order to have $s \sim s''$.

Assume o_x and o_{x+1} have the same label, i.e., $l_x = l_{x+1}$, then the first condition from Corollary 7 is satisfied. Therefore, $s \sim s''$. In case o_x and o_{x+1} do not have the same label, the only way for $s \sim s''$ to be true is for o_x and o_{x+1} to satisfy the second condition from Corollary 7. For the remainder of this proof we will assume that o_x and o_{x+1} do not have the same label, and we will show that they satisfy the second condition from Corollary 7, i.e.,

$$\neg(p_x \wedge p_{x+1}) \vee \bigvee_{i=1}^{x-1} p_i \tag{4.9}$$

Let's go back to sequences s and s' for a moment. Since $s \sim s'$, from Corollary 6 we have that all pairs of operators from s, including o_x and o_{x+1}, must satisfy at least one of the three necessary and sufficient conditions for $s \sim s'$. Let's consider the pair o_x and o_{x+1}.

Since o_x precedes o_{x+1} in s, but o_{x+1} precedes o_x in s' (that's how we chose o_x in the beginning of the proof), the first condition is not satisfied by these two operators. These operators do not satisfy the second condition as well, since they do not have the same label (according to our assumption). Therefore, since $s \sim s'$, o_x and o_{x+1} must satisfy the third condition of Corollary 6, namely:

$$\neg(p_x \wedge p_{x+1}) \vee \bigvee_{i=1}^{x-1} p_i \vee \bigvee_{j=1}^{y-1} p'_j \tag{4.10}$$

where y is the position of o_{x+1} in s'. Therefore, $o_{x+1} = o'_y$. Also note, that by construction, none of o'_j ($j \in \{1, \ldots, y-1\}$) can be equal to o_x or o_{x+1}, since $o_{x+1} = o'_y$ and $o_{x+1} \prec_{s'} o_x$.

We will show that every o'_j ($j \in \{1, \ldots, y-1\}$) is from among $o_1, \ldots, o_{x-1}$. Suppose otherwise, there exists $j \in \{1, \ldots, y-1\}$ such that $o'_j = o_z$, where $x \leq z \leq k$. Since, as mentioned above, none of o'_j ($j \in \{1, \ldots, y-1\}$) can be equal to o_x or o_{x+1}, we can obtain an even tighter bound for z, i.e., $x + 1 < z \leq k$.

Then, consider validation operators o_{x+1} and o_z. o_{x+1} precedes o_z in s, because $x + 1 < z$. However, o_z precedes o_{x+1} in s', because $pos_{s'}(o_{x+1}) = y$ and $pos_{s'}(o_z) < y$. From Lemma 8 we have, that there exists t, $x + 1 \leq t \leq z - 1$, such that o_{t+1} precedes o_t in s'.

Thus, we showed that there exists $t \geq x + 1 > x$, such that o_{t+1} precedes o_t in s'. However, by definition x is the *largest* number, such that o_{x+1} precedes o_x in s' (i.e., we chose x to be the largest such number in the first paragraph of this proof). We derived a contradiction, therefore our assumption that there exists

$j \in \{1, \ldots, y-1\}$ such that $o'_j = o_z$, where $x \leq z \leq k$ is incorrect. This implies that every o'_j ($j \in \{1, \ldots, y-1\}$) is from among $o_1, \ldots, o_{x-1}$. Therefore, every p'_j ($j \in \{1, \ldots, y-1\}$) is from among $p_1, \ldots, p_{x-1}$. Consequently, the third necessary condition 4.10 of $s \sim s'$ in this case is equivalent to:

$$\neg(p_x \wedge p_{x+1}) \vee \bigvee_{i=1}^{x-1} p_i \tag{4.11}$$

Hence, based on the fact that $s \sim s'$, we proved that the Boolean expression 4.11 holds for every element $e \in \mathcal{E}$. However, this expression is exactly the same as the one described by Equation 4.9, which was needed to prove that $s \sim s''$ (when o_x and o_{x+1} do not have the same label). Therefore, o_x and o_{x+1} satisfy the second sufficient condition of Corollary 7 and, hence, $s \sim s''$.

Now we have $s \sim s'$ and $s \sim s''$. Because of the transitivity and the symmetry of the relation $R_\sim$ (see Lemma 3), $s' \sim s''$ is also true. Also, we know that $dist(s, s') = d$ and $dist(s, s'') = 1$. Because of how we constructed s'', s' has all the same precedence inversions with respect to s'' as with respect to s, except for one. More specifically, o_x and o_{x+1} have the same precedence in both s' and s''. Therefore, the distance between s' and s'' is one less than between s' and s, i.e., $dist(s', s'') = d-1$.

◀

Theorem 10 *Let $s =< o_1, o_2, \ldots, o_k >$ be a validation sequence. Let s' be some permutation of s such that $dist(s, s') = d \geq 1$). Then, $s \sim s'$ if and only if there exists $d+1$ validation sequences s_0, s_1, …, s_d, such that $s_0 = s$, $s_d = s'$, and s_i is a safe simple permutation of s_{i-1} for every $i = 1, \ldots, d$.*

▶ Assume $s \sim s'$. Let's denote $s_0 := s$ and $s_d := s'$. Based on Theorem 9, there exists sequence s_1, such that s_1 is a safe simple permutation of s_0, and also

$s_1 \sim s_d$, and $dist(s_1, s_d) = d - 1$. Repeat this process for s_1 and s_d to obtain s_2, etc. In general, when we have s_i, such that $s_i \sim s_d$ and $dist(s_i, s_d) = d - i$, we can obtain s_{i+1} (which is a safe simple permutation of s_i), such that $s_{i+1} \sim s_d$ and $dist(s_{i+1}, s_d) = d - i - 1$. Hence, there exists $d + 1$ validation sequences s_0, s_1, ..., s_d, such that $s_0 = s$, $s_d = s'$, and s_i is a safe simple permutation of s_{i-1} for every $i = 1, \ldots, d$.

Now assume, that there exists $d + 1$ validation sequences s_0, s_1, ..., s_d, such that s_i is a safe simple permutation of s_{i-1} for every $i = 1, \ldots, d$. In other words, for every $i = 1, \ldots, d$, $s_{i-1} \sim s_i$. By transitivity of the equivalence relation: $s_0 \sim s_d$. Hence, $s \sim s'$. ◀

The above results give us a better understanding about equivalent permutations of a given validation sequence.

Let's assume that s is an arbitrary validation sequence. We can visualize the space of *all* permutations of s by constructing a *permutation graph*, where each vertex corresponds to a different permutation of s. Furthermore, in this graph, two vertices will have an edge connecting them if one of them is a simple permutation of the other. An example of such graph for a sequence consisting of four validation operators is presented in Figure 4.1.

Obviously, any two permutations are connected by a path (in fact, multiple paths) in this permutation graph. Theorem 10 states that if two permutations, say $s1$ and $s2$, are equivalent, there exists a minimal path from $s1$ to $s2$ that goes only through "intermediate" permutations that are equivalent to $s1$ and $s2$, where "minimal paths" between $s1$ and $s2$ are the paths with the length (i.e., number of edges comprising the paths) equal to the permutation distance between $s1$ and $s2$.

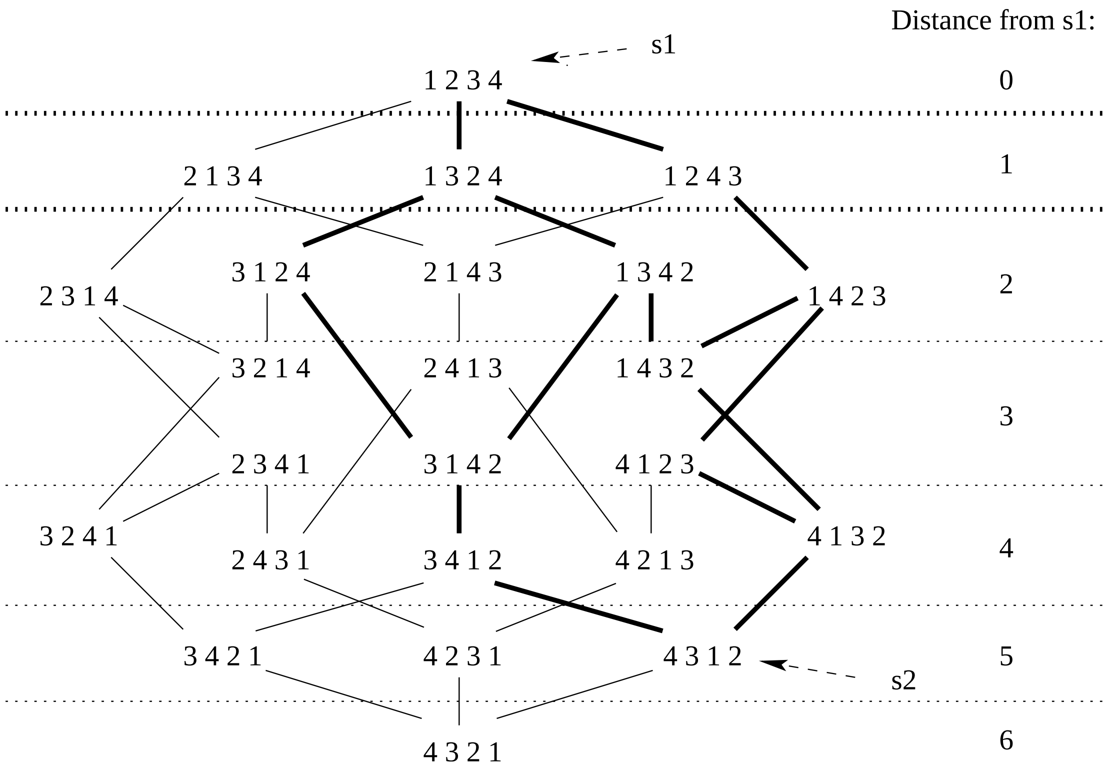


Figure 4.1: Permutation graph of a validation sequence.

Figure 4.1 illustrates this fact. More specifically, let $s1 := < o_1, o_2, o_3, o_4 >$. Also, let $s2$ be a permutation of $s1$ such that $s2 := < o_4, o_3, o_1, o_2 >$. The distance between $s1$ and $s2$ is 5, since there are 5 operator inversions in $s2$ with respect to $s1$, i.e., (o_4, o_3), (o_4, o_2), (o_4, o_1), (o_3, o_2), (o_3, o_1). Figure 4.1 highlights all the paths of length 5 between $s1$ and $s2$.

A useful fact about Theorem 10 is that it suggests a "search space" of equivalent validation sequence permutations, and we could try and take advantage of it in solving our sequence optimization problem.

Let's assume that we have an expert-specified validation sequence s. Let's also assume that we have a cost function defined for each permutation of s (we will address this issue in later sections). Then, we could search for the optimal permutation by traversing the permutation graph (such as depicted in Figure 4.1) by doing only *safe* simple permutations, starting from vertex s. Theorem 10 guarantees that we will encounter the optimal permutation $s*$ along the way.

We could use various graph traversal techniques, such as depth-first search or breadth-first search. However, if the validation sequence s has k validation operators, then the number of possible permutations is $k!$. When k is large, the exhaustive search techniques would not be scalable.

On the other hand, obviously, the largest possible distance between two permutations is $(k-1) + (k-2) + \ldots + 2 + 1 = k(k-1)/2$ (i.e., the largest possible number of inversions between two permutations – where everything is inverted). Therefore, there exists a path between the expert-specified validation sequence s and the optimal sequence $s*$ that is not longer than $k(k-1)/2$. Consequently, based on the specific cost function, we would like to find a greedy algorithm that allows us to choose the right path and not have to traverse the whole graph.

One obstacle with this approach is that we must be able to determine whether a given simple permutation is safe. Corollary 7 gives us two conditions – at least one of them has to be met by a simple permutation in order for it to be safe. Assume $s =< o_1, \ldots, o_k >$ be a validation sequence where $o_i = (l_i, p_i)$, and let $s' =< o'_1, \ldots, o'_k >$ be a simple permutation of s. That is, $(\exists! x \in \{1, \ldots, k - 1\})((o_x = o'_{x+1}) \wedge (o_{x+1} = o'_x))$, and also $o_x = o'_x$ for all $i \in \{1, \ldots, k\}$ such that $i \neq x$ and $i \neq x+1$. The first condition of Corollary 7, $l_x = l_{x+1}$, is easily verifiable. The second condition, i.e., whether the following expression holds for all possible inputs $e \in \mathcal{E}$

$$\neg(p_x(e) \quad \wedge \quad p_{x+1}(e)) \quad \vee \quad \bigvee_{i=1}^{x-1} p_i(e)$$

obviously is not easily solvable analytically.

We will later show that even when we restrict the validation sequence optimization problem, it remains very computationally complex. More specifically, we will show that one special case of our problem is NP-complete.

Therefore, we will try to simplify (restrict) the validation sequence optimization problem. For this purpose, we will introduce more restrictive concepts of permutation equivalence, i.e., *strong equivalence* and *very strong equivalence* of validation sequence permutations in the remainder of Section 4.2. But first, in the next section, we will introduce the notion of *predicate orthogonality*, which will be useful to us throughout this chapter.

4.2.3 Predicate Orthogonality

In this section, we define the concept of predicate orthogonality and prove some of its basic properties.

Definition 9 (Predicate Orthogonality) *Two predicates p and q are orthogonal if $(\forall e \in \mathcal{E})\ (\neg(p(e) \wedge q(e)))$. If p and q are orthogonal, we will denote it $p \perp q$. If p and q are not orthogonal, we will denote it $p \not\perp q$.*

Simply put, two predicates are orthogonal if they can never both match the same input element.

> **Example 4** By definition, cardinality predicates C^0_{Milk} and C^1_{Milk} would never match the same itemset. Hence, $C^0_{Milk} \perp C^1_{Milk}$.

Lemma 11 $(\forall p \in \mathcal{P})(p \perp \neg p)$.

▶ Follows directly from the definition of predicate orthogonality, since $\neg(p(e) \wedge \neg p(e)) = \neg p(e) \vee p(e)$ is `True` for any $e \in \mathcal{E}$. ◀

Predicate orthogonality can be viewed as a *binary relation* on the set of all predicates. Let's denote this relation $R_\perp$. $R_\perp$ has the following properties: symmetric, but neither reflexive nor transitive, as shown below.[2]

Lemma 12 *Binary relation $R_\perp$ is symmetric, but neither reflexive nor transitive.*

[2]These are the properties that one may intuitively expect an orthogonality relation to have, e.g., orthogonality relation on the set of straight lines on a two-dimensional plane has the exact same properties.

▶ $R_\perp$ *is not reflexive.* In order for $R_\perp$ to be reflexive, for every predicate p, $p \perp p$ must hold. Based on the predicate orthogonality definition, $p \perp p \iff (\forall e \in \mathcal{E})\ (\neg(p(e) \wedge p(e))) \iff (\forall e \in \mathcal{E})\ (\neg p(e))$. However, clearly there exists a set predicate that does not evaluate to `False` on every itemset.[3] Hence, $R_\perp$ is not reflexive.

$R_\perp$ *is symmetric.* For any predicates p and q, if $p \perp q$ then $q \perp p$. This follows directly from the definition of predicate orthogonality, since obviously $\neg(p(e) \wedge q(e)) = \neg(q(e) \wedge p(e))$.

$R_\perp$ *is not transitive.* For any predicates p, q, and r, if $p \perp q$ and $q \perp r$, it does not necessarily imply that $p \perp r$. For example, let a be an arbitrary predicate and let's define predicates p, q, r as follows: $p := a$, $q := \neg a$, and $r := a$. From Lemma 11, $p \perp q$ and $q \perp r$, since $a \perp \neg a$. However, p and r are not orthogonal, since $R_\perp$ is not reflexive, as shown above. ◀

Lemma 13 *Let $p_1, \ldots, p_m$ and q be predicates. Let $p := p_1 \wedge \ldots \wedge p_m$. If there exists $i \in \{1, \ldots, m\}$ such that $p_i \perp q$, then $p \perp q$.*

▶ Consider the following set predicate expression:

$$\begin{aligned}
\neg(p \wedge q) &= \neg(p_1 \wedge \ldots \wedge p_m \wedge q) \\
&= \neg p_1 \vee \ldots \vee \neg p_m \vee \neg q \\
&= (\neg p_1 \vee \neg q) \vee \ldots \vee (\neg p_m \vee \neg q) \\
&= \neg(p_1 \wedge q) \vee \ldots \vee \neg(p_m \wedge q)
\end{aligned}$$

Now assume $p_i \perp q$. By orthogonality definition, $\neg(p_i \wedge q)$ holds for all possible inputs. Therefore, $\bigvee_{j=1}^{m} \neg(p_j \wedge q)$ holds for all possible inputs. Based on the equalities above,

[3] E.g., the predicate that *always* returns `True` regardless of input.

$\neg(p \wedge q)$ holds for all inputs. Therefore, $p \perp q$. ◀

Corollary 14 *Let* $p := p_1 \wedge \ldots \wedge p_m$ *and* $q := q_1 \wedge \ldots \wedge q_n$*. If* $(\exists i \in \{1, \ldots, m\})$ $(\exists j \in \{1, \ldots n\})(p_i \perp q_j)$*, then* $p \perp q$*.*

▶ Assume $p_i \perp q_j$. From Lemma 13, $p_i \perp q$. Again from Lemma 13, $p \perp q$. ◀

Lemma 15 *Let* p_1*, ...,* p_m *and* q *be predicates. Let* $p := p_1 \vee \ldots \vee p_m$*. Then* $p \perp q$ *if and only if* $(\forall i \in \{1, \ldots, m\})$ $(p_i \perp q)$*.*

▶ Consider the following predicate expression:

$$\begin{aligned} \neg(p \wedge q) &= \neg((p_1 \vee \ldots \vee p_m) \wedge q) \\ &= \neg(p_1 \wedge q) \vee \ldots \vee (p_m \wedge q) \\ &= \neg(p_1 \wedge q) \wedge \ldots \wedge \neg(p_m \wedge q) \end{aligned}$$

Assume $p \perp q$. From orthogonality definition we have, that $\neg(p \wedge q)$ holds for all possible inputs. Then, based on the equalities above, we have that all the expressions $\neg(p_i \wedge q)$ (where $i \in \{1, \ldots, m\}$) must hold for all inputs. Consequently, $p_i \perp q$ for all $i \in \{1, \ldots, m\}$.

Now assume $p_i \perp q$ for all $i \in \{1, \ldots, m\}$. From orthogonality definition we have that $\neg(p_i \wedge q)$ holds on all inputs for all $i \in \{1, \ldots, m\}$. Based on the equalities above, $\neg(p \wedge q)$ always holds. Therefore, $p \perp q$. ◀

Corollary 16 *Let* $p := p_1 \vee \ldots \vee p_m$ *and* $q := q_1 \vee \ldots \vee q_n$*. Then* $p \perp q$ *if and only if* $(\forall i \in \{1, \ldots, m\})$ $(\forall j \in \{1, \ldots n\})(p_i \perp q_j)$*.*

▶ Follows directly from Lemma 15. ◀

4.2.4 Strongly Equivalent Permutations

Definition 10 (Strongly Equivalent Permutation)
Let $s =< (l_1, p_1), \ldots, (l_k, p_k) >$ and $s' =< (l'_1, p'_1), \ldots, (l'_k, p'_k) >$ be validation sequences, where s' is a permutation of s. s' is said to be a strongly equivalent permutation of s if and only if <u>every</u> possible pair of validation operators $u = (l_u, p_u)$ and $v = (l_v, p_v)$ (i.e., $u \in s$, $v \in s$, $u \neq v$) satisfies <u>at least one</u> of the following conditions:

1. *Either u precedes v in both s and s', or v precedes u in both s and s';*
2. *u and v have the same label, that is, $l_u = l_v$;*
3. *p_u and p_v are orthogonal, i.e., $p_u \perp p_v$.*

If s' is a strongly equivalent permutation of s, we will denote it as $s \approx s'$.

Lemma 17 $s \approx s' \Longrightarrow s \sim s'$

► Follows straightforwardly from the definition of strong equivalence and Corollary 6.

More specifically, since $s \approx s'$, every pair of validation operators u and v (i.e., $u \in s$ and $v \in s$) must satisfy one of the three conditions from Definition 10. Consequently, u and v satisfy the corresponding sufficient condition from Corollary 6. Hence, $s \sim s'$. ◄

The strong equivalence of validation sequence permutations can be viewed as a *binary relation* on the set of all possible permutations of a given validation sequence. Let's denote this relation $R_{\approx}$. Note, that $R_{\approx}$ is a *true* equivalence relation, since it is reflexive, symmetric, and transitive, as shown below.

Lemma 18 *Relation $R_{\approx}$ is a true equivalence relation.*

▶ *$R_{\approx}$ is reflexive.* Obviously, $s \approx s$ for all s, since no operators are permuted. That is, condition 1 of Definition 10 is satisfied for all pairs u, v of operators in sequence s.

$R_{\approx}$ is symmetric. For any two validation sequences s and s', where s' is a permutation of s, if $s \approx s'$ then $s' \approx s$, since all three conditions in Definition 10 treat sequences s and s' in a symmetric manner.

$R_{\approx}$ is transitive. For any validation sequences s, s', and s'', where s' and s'' are permutations of s, if $s \approx s'$ and $s' \approx s''$, then $s \approx s''$.

Assume otherwise, i.e., there exist sequences s, s', and s'', such that $s \approx s'$ and $s' \approx s''$, but $s \not\approx s''$. If $s \not\approx s''$, there exist validation operators $u = (l_u, p_u)$ and $v = (l_v, p_v)$ in sequence s that do not satisfy any of the three conditions in Definition 10. Without loss of generality, let's assume that $u \prec_s v$. More specifically, all of the following must be true: $u \prec_s v$, $v \prec_{s''} u$, $l_u \neq l_v$, and $p_u \not\perp p_v$.

Since $s \approx s'$, we have that all pairs of operators, including u and v, should satisfy the three conditions of strong equivalence of s and s'. As we have shown in the previous paragraph, $l_u \neq l_v$, and $p_u \not\perp p_v$, therefore, conditions 2 and 3 do not hold. Consequently, condition 1 must hold in order to have $s \approx s'$. Since $u \prec_s v$, then $u \prec_{s'} v$ (according to condition 1).

If we repeat the same argument for $s' \approx s''$, we would see that operators u and v again have to satisfy condition 1, because conditions 2 and 3 cannot hold for the same reasons as described in the previous paragraph. Since we derived earlier that $u \prec_{s'} v$, then $u \prec_{s''} v$ (according to condition 1 for $s' \approx s''$). We derived a contradiction, since $v \prec_{s''} u$ (assuming $R_{\approx}$ is not transitive). Hence, $R_{\approx}$ is a transitive relation. ◀

Also, we can show that for strongly equivalent sequence permutations there exist similar "connectedness" results as for safe (equivalent) sequence permutations. But first let's derive the necessary and sufficient conditions of strong equivalence for *simple* permutations.

Lemma 19 *Let $s =< o_1, \ldots, o_k >$ be a validation sequence where $o_i = (l_i, p_i)$, and let $s' =< o'_1, \ldots, o'_k >$ be a simple permutation of s. That is, $(\exists!\ x \in \{1, \ldots, k-1\})((o_x = o'_{x+1}) \wedge (o_{x+1} = o'_x))$, and also $o_i = o'_i$ for all $i \in \{1, \ldots, k\}$ such that $i \neq x$ and $i \neq x+1$.*

Then $s \approx s'$ if and only if <u>at least one</u> of the following two conditions is satisfied:

1. *o_x and o_{x+1} have the same label, that is, $l_x = l_{x+1}$;*

2. *$p_x \perp p_{x+1}$.*

▶ Definition 10 gives us three conditions that have to be satisfied in order to have $s \approx s'$. Since s' is a simple permutation of s, these conditions are equivalent to the two conditions above.

More specifically, when s' is a *simple* permutation of s, there is only one inversion of operator precedence in s' with respect to s. Therefore, all pairs of validation operators $u, v \in s$ except one (i.e., o_x and o_{x+1}) automatically satisfy condition 1 of Definition 10. Consequently, $s \sim s'$ if and only if the remaining pair of validation operators, i.e., o_x and o_{x+1}, satisfies at least one of the remaining two conditions. ◀

Theorem 20 *Let $s =< o_1, o_2, \ldots, o_k >$ be a validation sequence. Let s' be some strongly equivalent permutation of s (i.e., $s \approx s'$) such that $dist(s, s') = d \geq 1$). Then there exists a sequence s'' that is a* <u>*strongly*</u> *equivalent simple permutation of s (i.e., $dist(s, s'') = 1$ and $s \approx s''$), such that $s' \approx s''$ and $dist(s', s'') = d - 1$.*

▶ Let x be any number from the set $\{1, \ldots, k\}$, such that $o_{x+1} \prec_{s'} o_x$. Then, let's construct the sequence $s'' =< o''_1, \ldots, o''_k >$ as follows. Let $o''_i := o_i$, for all i, such that $1 \leq i \leq x - 1$ or $x + 2 \leq i \leq k$. Also, let $o''_x = o_{x+1}$ and $o''_{x+1} = o_x$. Essentially, sequence s'' is the same as s except for o_x and o_{x+1} that are swapped. Obviously, s'' is a simple permutation of s, thus $dist(s, s'') = 1$.

Now we will show that $s \approx s''$. Since s'' is a simple permutation of s, Lemma 19 gives us two conditions to be satisfied in order to have $s \approx s''$.

Assume o_x and o_{x+1} have the same label, i.e., $l_x = l_{x+1}$, then the first condition from Lemma 19 is satisfied. Therefore, $s \approx s''$. In case o_x and o_{x+1} do not have the same label, the only way for $s \approx s''$ to be true is for o_x and o_{x+1} to satisfy the second condition from Lemma 19, namely $p_x \perp p_{x+1}$.

However, since $s \approx s'$, from Definition 10 we have that all pairs of operators from s, including o_x and o_{x+1}, must satisfy at least one of the three conditions for $s \approx s'$. Let's consider the pair o_x and o_{x+1}.

Since o_x precedes o_{x+1} in s, but o_{x+1} precedes o_x in s' (that's how we chose o_x in the beginning of the proof), the first condition is not satisfied by these two operators. These operators do not satisfy the second condition as well, since they do not have the same label (according to our assumption). Therefore, since $s \approx s'$, o_x and o_{x+1} must satisfy the third condition of Definition 10, namely $p_x \perp p_{x+1}$, which is exactly what we needed to prove.

Now we have $s \approx s'$ and $s \approx s''$. Because of the transitivity and the symmetry of the relation $R_{\approx}$ (see Lemma 18), $s' \approx s''$ is also true. Also, we know that $dist(s, s') = d$ and $dist(s, s'') = 1$. Because of how we constructed s'', s' has all the same precedence inversions with respect to s'' as with respect to s, except for one – o_x and o_{x+1} have the same precedence in both s' and s''. Therefore, the distance between s' and s'' is one less than between s' and s, i.e., $dist(s', s'') = d - 1$. ◀

Theorem 21 *Let $s =< o_1, o_2, \ldots, o_k >$ be a validation sequence. Let s' be some permutation of s such that $dist(s, s') = d \geq 1$). Then, $s \approx s'$ if and only if there exists $d + 1$ validation sequences s_0, s_1, $\ldots$, s_d, such that $s_0 = s$, $s_d = s'$, and s_i is a strongly equivalent simple permutation of s_{i-1} for every $i = 1, \ldots, d$.*

▶ Assume $s \approx s'$. Let's denote $s_0 := s$ and $s_d := s'$. Based on Theorem 20, there exists sequence s_1, such that s_1 is a strongly equivalent simple permutation of s_0, $s_1 \sim s_d$, and $dist(s_1, s_d) = d - 1$. Repeat this process for s_1 and s_d to obtain s_2, etc. In general, when we have s_i, such that $s_i \approx s_d$ and $dist(s_i, s_d) = d - i$, we can obtain s_{i+1} (which is a strongly equivalent simple permutation of s_i), such that $s_{i+1} \approx s_d$ and $dist(s_{i+1}, s_d) = d - i - 1$. Hence, there exists $d + 1$ validation sequences s_0, s_1, $\ldots$, s_d, such that $s_0 = s$, $s_d = s'$, and s_i is a strongly equivalent simple permutation of s_{i-1} for every $i = 1, \ldots, d$.

Now assume, that there exists $d + 1$ validation sequences s_0, s_1, $\ldots$, s_d, such that s_i is a strongly equivalent simple permutation of s_{i-1} for every $i = 1, \ldots, d$. In other words, for every $i = 1, \ldots, d$, $s_{i-1} \approx s_i$. By transitivity of the strong equivalence relation: $s_0 \approx s_d$. ◀

Now that we have defined the notion of the strong equivalence for validation sequence permutations, we can formulate the restricted validation sequence optimization problem as follows:

$$\begin{aligned} s* &= \arg\min_{s' \approx s} cost(s', D) \\ &= \arg\max_{s' \approx s} benefit(s', D) \\ &= \arg\max_{s' \approx s} \sum_{i=1}^{k-1} (k-i)n'_i \end{aligned} \tag{4.12}$$

4.2.5 Very Strongly Equivalent Permutations

In Section 4.2.4, we introduced a subclass of equivalent (safe) permutations called strongly equivalent permutations. Let's restrict the class of strongly equivalent permutations even further and introduce the class of *very strongly equivalent permutations.* We will use this new class to prove NP-completeness of the restricted optimization problem.

Definition 11 (Very Strongly Equivalent Permutation)

Let $s =< (l_1, p_1), \ldots, (l_k, p_k) >$ *and* $s' =< (l'_1, p'_1), \ldots, (l'_k, p'_k) >$ *be validation sequences, where* s' *is a permutation of* s. s' *is said to be a* <u>*very strongly equivalent*</u> *permutation of* s *if and only if* <u>*every*</u> *possible pair of validation operators* $u = (l_u, p_u)$ *and* $v = (l_v, p_v)$ *(i.e.,* $u \in s$, $v \in s$, $u \neq v$*) satisfies* <u>*at least one*</u> *of the following conditions:*

1. *Either* u *precedes* v *in both* s *and* s', *or* v *precedes* u *in both* s *and* s'*;*

2. $p_u \perp p_v$.

If s' *is a very strongly equivalent permutation of* s, *we will denote it as* $s \approxeq s'$.

Lemma 22 $s \approxeq s' \Longrightarrow s \approx s'$

► Follows straightforwardly from the definitions of strong equivalence and very strong equivalence of validation sequence permutations. ◄

One could also prove that the relation of very strong equivalence is a true equivalence relation in the same manner as it was done for equivalent validation sequences and strongly equivalent validation sequences. The "connectedness" results for very strongly equivalent permutations can also be obtained in a straightforward manner using the same techniques as were used to prove the "connectedness" for both equivalent and strongly equivalent validation sequences in Section 4.2.2.

Let's further restrict the optimization problem:

$$s* = \arg\max_{s' \approx s} \mathit{benefit}(s', D)$$

and consider its special case:

$$s* = \arg\max_{s' \approxeq s} \mathit{benefit}(s', D)$$

We will show that this problem is equivalent to a known NP-complete problem. By inserting in the definition of the *benefit* function, we obtain that

$$s* = \arg\max_{s' \approxeq s} \sum_{i=1}^{k-1} (k-i) n'_i$$

where n'_i are the numbers of data points from D validated by each validation operator o'_i in sequence s'. Since, we know only the numbers n_i (i.e., numbers of data points from D validated by each validation operator o_i in sequence s), we will have to estimate n'_i theoretically.

Assume, s' is a very strongly equivalent permutation of s. Let's consider an arbitrary operator $o_i \in s$, i.e., $pos_s(o_i) = i$. Also, let's assume that in the permuted sequence s', o_i would be at some position j, i.e., $o_i = o'_j$ or $pos_{s'}(o_i) = j$. Finally, let's also assume that o_i validated n_i data points from dataset D. How many data points will o_i validate in the permuted sequence s' being at a position j? The answer is, it will validate exactly n_i points again, regardless of position j it was placed in the permuted sequence s', as demonstrated below.

Lemma 23 *Let $s =< o_1, \ldots, o_k >$ be a validation sequence and let $n_1, \ldots, n_k$ be the numbers of data points validated by each of the validation operators in s, given some dataset D. Let $s' =< o'_1, \ldots, o'_k >$ be a very strongly equivalent permutation of s (i.e., $s \approxeq s'$) and let $n'_1, \ldots, n'_k$ be the numbers of data points validated by each of the validation operators in s', given the same dataset D. Then for every $i \in \{1, \ldots, k\}$: $n_i = n'_j$ where $j = pos_{s'}(o_i)$.*

▶ Let's assume $s \approxeq s'$ and let's consider an arbitrary validation operator o_i from sequence s, i.e., $pos_s(o_i) = i$. Also, let $j = pos_{s'}(o_i)$. We have to show that $n_i = n'_j$. We will show this by showing that o_i validates exactly the same subset of D in both s and s'.

Assume otherwise, that there exists $e \in D$ such that either (1) o_i validates e in s but not in s', or (2) o_i validates e in s' but not in s. We will provide the proof for the first of these two situations. The proof for the second one is essentially identical.

Since there exists $e \in D$ such that o_i validates e in s but not in s', there must exist a validation operator o_x such that $o_x \prec_{s'} o_i$ and o_x validates e. However, $o_i \prec_s o_x$, because otherwise o_i would not be able to validate e in s (i.e., o_x would validate e before o_i). Therefore, we have two validation operators o_i and o_x such that $o_i \prec_s o_x$, $o_x \prec_{s'} o_i$, and $p_i \not\perp p_x$ (since there exists $e \in D$ that can be validated by both o_i and o_x). This is a contradiction, because by the definition of very strong equivalence all pairs of validation operators must satisfy one of two conditions (see Definition 11), whereas the pair o_i and o_x satisfies neither.

Therefore, given s and s', where s' is very strongly equivalent to s, each validation operator validates exactly the same subset of inputs from D in both s and s'. Hence, for all i: $n_i = n'_j$, where $j = pos_{s'}(o_i)$. ◀

Given validation sequence $s =< o_1, \ldots, o_k >$ and some permutation s', Definition 11 specifies several conditions that must be satisfied by every pair of validation operators $u \in s$ and $v \in s$, so that $s \approxeq s'$. We will show that these conditions are equivalent to specifying a certain partial order on the set of validation operators in s.

More specifically, let $G_s = (V, E)$ be a directed acyclic graph with k vertices, where each vertex $i \in V$ is associated with a different validation operator o_i ($i \in \{1, \ldots, k\}$). Furthermore, the set E of edges is defined as follows. For every pair of validation operators $o_i = (l_i, p_i)$ and $o_j = (l_j, p_j)$ such that $o_i \prec_s o_j$ (i.e., $i < j$), add an edge from vertex i to vertex j to set E if $p_i \not\perp p_j$. We will call this graph a *precedence* graph of sequence s.

Note, that if precedence graph G_s has an edge from i to j, then any permutation s' that is very strongly equivalent to s must have $o_i \prec_{s'} o_j$. If that were not the case, i.e., if there existed a very strongly equivalent permutation s' such that $o_j \prec_{s'} o_i$, then we would derive a contradiction, since validation operators o_i and o_j would not satisfy either of the two conditions from Definition 11 and it would imply that $s \not\approxeq s'$.

It is easy to see that G_s represents a partial order over the set of validation operators o_1, ..., o_k. As we showed above, those permutations of s that satisfy this partial order are very strongly equivalent to s, and the ones that do not satisfy this order are not very strongly equivalent to s. Also, since we have not placed any restrictions on what kind of predicates can be used in validation operators, the resulting precedence graph in general can represent any possible partial order.

Therefore, we have transformed the restricted validation sequence optimization problem

$$s* = \arg\max_{s' \approxeq s} \sum_{i=1}^{k-1} (k-i)n'_i$$

into the following validation operator "scheduling" problem:

$$s* = \arg\max_{s' \ \textit{satisfies} \ G_s} \sum_{i=1}^{k-1} (k-i)n'_i \tag{4.13}$$

where $n'_i = n_x$, if $pos_{s'}(o_x) = i$ (according to Lemma 23). In other words, the problem is to find a "scheduling" of operators o_1, ..., o_k such that it obeys the precedence graph G and the corresponding permutation of n_1, ..., n_k maximizes the *benefit* function.

Let's assume that we know how to efficiently compute whether two predicates are orthogonal (we will show how to do that for cardinality predicates later in this chapter). Therefore, let's assume that, given validation sequence s, we can efficiently construct precedence graph G_s. Then, we will show that the above scheduling problem (specified in Equation 4.13) is NP-complete, by showing that solving it is equivalent to solving a known NP-complete problem, described in the next section.

4.3 NP-Completeness of Restricted Optimization Problem

In this section, we will show that our restricted optimization problem (i.e., its search space restricted only to *very strongly equivalent* permutations) is equivalent to the NP-complete task sequencing problem, described below.

4.3.1 Task Sequencing to Minimize Weighted Completion Time

The following problem is often referred to as the problem of "Task Sequencing on a Single Processor to Minimize Weighted Completion Time" [37].

Assume, that a set of tasks T has to be sequenced for processing by a single machine. The sequencing of the tasks must obey the precedence constraints imposed by a given directed acyclic graph $G = (V, E)$, where each vertex $v \in V$ is associated with a different task (therefore, $|T| = |V|$). In other words, G imposes a partial order on T. Task $t' \in T$ must precede task $t'' \in T$ if there is a directed path from t' to t'' in G.

Furthermore, each task t is assigned a processing time $p(t) \in Z^+$ and a weight $w(t) \in Z$. Given a specific sequencing of T, e.g., $s =< t_1, \ldots, t_k >$, the completion time of each task t_i is denoted as $C(t_i)$ and can be calculated as

$$C(t_i) = \sum_{j=1}^{i} p(t_j) \tag{4.14}$$

where we assume that the processing of the first task begins immediately (i.e., at time 0) and there is no idle time between consecutive jobs.

The objective of the sequencing problem is to find the feasible sequence (i.e., that obeys the partial order imposed by G) $s =< t_1, \ldots, t_k >$ ($t_i \in T$) that minimizes the weighted total completion time $WTCT(s)$, defined as a weighted sum of individual completion times, i.e.,

$$WTCT(s) = \sum_{i=1}^{k} w(t_i)\ C(t_i) \tag{4.15}$$

Lawler [49] showed that the above problem is NP-complete. Furthermore, it was also shown that the above problem remains NP-complete even when all $w(t) = 1$.

Assuming $w(t) = 1$ for all $t \in T$ and using the definition of $C(t)$ from Equation 4.14, the weighted total completion time of sequence $s =< t_1, \ldots, t_k >$ can be expressed as

$$\begin{aligned} WTCT(s) &= \sum_{i=1}^{k} C(t_i) && (4.16)\\ &= \sum_{i=1}^{k}\sum_{j=1}^{i} p(t_j) && (4.17)\\ &= \sum_{i=1}^{k} (k+1-i)\ p(t_i) && (4.18) \end{aligned}$$

4.3.2 Equivalence of the Two Problems

As indicated above, the problem of finding the task sequence that obeys the specified partial order and *minimizes* the weighted total completion time is NP-complete. We will show that the problem of finding the task sequence that obeys the specified partial order and *maximizes* the weighted total completion time is also NP-complete.

Let $G = (V, E)$ be an acyclic directed graph representing the partial order to be imposed on tasks T. Then we will define a "reverse" graph $G' = (V', E')$ as follows. Let $V' = V$ and let E' contain the same edges as E, only each edge should point in the reverse direction. That is, $E' := \{(u, v) : (v, u) \in E\}$.

We will show that $s =< t_1, \ldots, t_k >$ minimizes $WTCT$ with respect to partial order G if and only if $s' =< t_k, \ldots, t_1 >$ (i.e., s' is the reversed sequence s)

maximizes $WTCT$ with respect to partial order G'.

Theorem 24 *$s =< t_1, \ldots, t_k >$ minimizes $WTCT$ with respect to partial order G $\iff$ $s' =< t_k, \ldots, t_1 >$ maximizes $WTCT$ with respect to partial order G'.*

▶ Let's assume that sequence $s =< t_1, \ldots, t_k >$ minimizes $WTCT$ with respect to partial order G. We will show that the reverse sequence $s' =< t'_1, \ldots, t'_k >$ then maximizes $WTCT$ with respect to partial order G'. Because s' is a reverse of s, we have that $t'_i = t_{k+1-i}$ for all $i \in \{1, \ldots, k\}$. Obviously, since s obeys the partial order of G, the reverse sequence s' obeys the partial order of the "reverse" graph G'.

Assume otherwise, i.e., s' does not maximize $WTCT$ with respect to G'. Therefore, there exists a different sequencing of tasks $z' =< u'_1, \ldots, u'_k >$, such that $z' \neq s'$, z' obeys G' and $WTCT(s') < WTCT(z')$. Therefore, we have

$$\begin{aligned} 0 &< WTCT(z') - WTCT(s') \\ &= \sum_{i=1}^{k}(k+1-i)\ p(u'_i) - \sum_{i=1}^{k}(k+1-i)\ p(t'_i) \\ &= \sum_{i=1}^{k}(k+1-i)\ (p(u'_i) - p(t'_i)) \end{aligned}$$

Let's denote $z =< u_1, \ldots, u_k >$ to be the reverse of z', i.e., $u'_i = u_{k+1-i}$ for all $i \in \{1, \ldots, k\}$. Since z' satisfies partial order constraints specified by G', z must satisfy the ones specified by G. Also, $s' \neq z'$ implies $s \neq z$. Then the above equations can be rewritten as:

$$\begin{aligned}
0 &< \sum_{i=1}^{k}(k+1-i)\ (p(u'_i) - p(t'_i)) \\
&= \sum_{i=1}^{k}(k+1-i)\ (p(u_{k+1-i}) - p(t_{k+1-i})) \\
&= \sum_{j=1}^{k} j\ (p(u_j) - p(t_j)) \qquad (4.19)
\end{aligned}$$

where in the last expression above, we simply changed the summation index from i to j, where $j = k + 1 - i$.

Note, that $s =< t_1, \ldots, t_k >$ and $z =< u_1, \ldots, u_k >$ are two different sequences of the same task set, therefore

$$\begin{aligned}
\sum_{j=1}^{k} p(t_j) &= \sum_{j=1}^{k} p(u_j) \quad \Rightarrow \\
\sum_{j=1}^{k}(p(u_j) - p(t_j)) &= 0 \quad \Rightarrow \\
(k+1)\sum_{j=1}^{k}(p(u_j) - p(t_j)) &= 0 \quad \Rightarrow \\
\sum_{j=1}^{k}(k+1)\ (p(u_j) - p(t_j)) &= 0 \qquad (4.20)
\end{aligned}$$

By puting together Equations 4.19 and 4.20 we have:

$$\begin{aligned}
& \sum_{j=1}^{k}(k+1)\ (p(u_j) - p(t_j)) < \sum_{j=1}^{k} j\ (p(u_j) - p(t_j)) \\
\Rightarrow\ & \sum_{j=1}^{k}(k+1-j)\ (p(u_j) - p(t_j)) < 0 \\
\Rightarrow\ & \sum_{j=1}^{k}(k+1-j)\ p(u_j) < \sum_{j=1}^{k}(k+1-j)\ p(t_j) \\
\Rightarrow\ & WTCT(z) < WTCT(s)
\end{aligned}$$

Therefore, we have showed that there exists sequence z ($z \neq s$) that obeys partial order specified by G and has smaller weighted total completion time than s. We derived a contradiction, because by definition s is the sequence that minimizes $WTCT$ with respect to G. Therefore, our assumption that s' does not maximize $WTCT$ with respect to G' was incorrect.

The proof in the other direction is essentially identical. ◀

The above theorem indicates that solving the problem of task sequencing to minimize weighted completion time subject to partial order constraints is equivalent to solving the problem of task sequencing to maximize weighted completion time subject to partial order constraints. Since the former problem has been shown to be NP-complete [49], consequently the latter problem is NP-complete as well. In addition, the latter problem is equivalent to our restricted validation sequence optimization (i.e., benefit maximization) problem (Equation 4.13), since in both cases we are searching for the sequence that satisfies the given partial order and maximizes essentially the same function.[4] Hence, our restricted optimization problem is NP-complete as well.

4.4 Greedy Heuristic for Validation Sequence Improvement

In the previous section we showed that, given validation sequence s, the problem of finding the optimal sequence among all the sequences that are very strongly equivalent to s is NP-complete. Note, that this problem is already NP-complete without even taking into account the computation of the precedence graph G. In

[4]Actually, the functions in two problems differ, but only by a constant, which does not depend on a particular sequencing and, therefore, does not affect the solution.

the case of the task scheduling problem, described in Section 4.3.1, the precedence graph is given (i.e., it is part of the input). However, in our problem, we have to calculate the precedence graph ourselves. In other words, we have to be able to calculate which pairs of operators must preserve their precedence in the permuted sequence, based on the orthogonality of their predicates.

Note, that the precedence graph calculation depends on the class of predicates used in validation operators. If the predicates are complex, it may be very difficult (or impossible) to show whether two given predicates are orthogonal or not. Since the problem is NP-complete even when the graph is already given, in this section we will present a general (i.e., independent of the class of predicates used) heuristic-based approach to improving the validation sequence when a precedence graph G is given as an input. We will separately address the issue of how to construct the precedence graph for cardinality predicate-based validation sequences later in this chapter.

4.4.1 Precedence Graph for Strongly Equivalent Permutations

For the purpose of proving the NP-completeness of the optimization problem, we showed earlier how to construct the precedence graph based on *very strong equivalence* constraints. For our heuristic approach, we will construct the precedence graph based on less restrictive equivalence – *strong equivalence* – constraints. This way, our heuristic will have a potentially larger search space to work with and, therefore, may generate permutations with better performance improvements.

Given validation sequence $s =< o_1, \dots, o_k >$ and some its permutation s', Definition 10 specifies several conditions that must be satisfied by every pair of validation operators $u \in s$ and $v \in s$ so that $s \approx s'$. We will show that these conditions are equivalent to specifying a certain partial order on the set of validation operators in s.

More specifically, let $G_s = (V, E)$ be a directed acyclic graph with k vertices, where each vertex $i \in V$ is associated with a different validation operator o_i ($i \in \{1, \ldots, k\}$). Furthermore, the set E of edges is defined as follows. For every pair of validation operators $o_i = (l_i, p_i)$ and $o_j = (l_j, p_j)$ such that $o_i \prec_s o_j$ (i.e., $i < j$), add an edge from vertex i to vertex j to set E if both $l_i \neq l_j$ and $p_i \not\perp p_j$.

Note, that if precedence graph G_s has an edge from i to j, then any permutation s' that is strongly equivalent to s must have $o_i \prec_{s'} o_j$. If that were not the case, i.e., if there existed a strongly equivalent permutation s' such that $o_j \prec_{s'} o_i$, then we would derive a contradiction, since validation operators o_i and o_j would not satisfy all three conditions from Definition 10 and it would imply that $s \not\approx s'$.

Let's assume that we know how to efficiently compute whether two predicates are orthogonal (we will show how to do that for cardinality predicates later in this chapter). Therefore, let's assume for now that, given validation sequence s, we can efficiently construct precedence graph G_s.

4.4.2 Sequence Improvement using a Simple Permutation

Let $s =< o_1, \ldots, o_k >$ be a validation sequence. Also, let G be a precedence graph based on sequence s. As mentioned earlier, the cost of s given specific input data D is:

$$cost(s, D) = k|D| \; - \; \sum_{i=1}^{k-1} (k - i) \; n_i$$

where n_i is the number of input data points from D validated (labeled) by validation operator o_i.

Also, let $s' =< o'_1, \ldots, o'_k >$ be a *simple* permutation of s. That is, $(\exists!\ x \in \{1, \ldots, k-1\})((o_x = o'_{x+1}) \wedge (o_{x+1} = o'_x))$, and also $o_i = o'_i$ for all $i \in \{1, \ldots, k\}$ such that $i \neq x$ and $i \neq x+1$. Also, let's assume that there is no precedence constraint between operators o_x and o_{x+1}, i.e., there is no edge from o_x to o_{x+1} in G. Therefore, $s \approx s'$. Consequently, $s \sim s'$ and therefore s' will produce the same validation results as s. What is the cost of s'? To be able to answer this, we have to estimate the numbers n'_i, i.e., the number of data points from dataset D that each validation operator o'_i (from permuted sequence) would validate.

First, it is clear that $n_i = n'_i$ for all $i < x$, since only the operators o_x and o_{x+1} are permuted. That is, first $x-1$ operators in both sequences s and s' are the same and will produce the same validation results.

It is also easy to see that $n_i = n'_i$ for all $i > x+1$. This is the case, because the set of first $x+1$ validation operators is the same in both sequences (not necessarily in the same order). Obviously, the exact same subset of input dataset D would remain unvalidated after $x+1$ operators in both.[5] In addition, $o_i = o'_i$ for $i > x+1$. Therefore, $n_i = n'_i$ for all $i > x+1$.

We still need to estimate n'_x and n'_{x+1}. Let's consider operators $o_x = (l_x, p_x)$ and $o_{x+1} = (l_{x+1}, p_{x+1})$. We know that $o'_x = o_{x+1}$ and $o'_{x+1} = o_x$. Since $s \approx s'$ and s' is a simple permutation of s, according to Lemma 19 we have one of the following two possibilities:

- $p_x \perp p_{x+1}$. This means that validation operators o_x and o_{x+1} can never both

[5]For more precise reasoning, consider the two sequences of length $x+1$ and see Lemma 4.

match the same input data point. Therefore, it does not matter whether o_x precedes o_{x+1} (as in sequence s) or o_{x+1} precedes o_x (as in sequence s'), they will still validate the same exact data points as before. Hence, $n'_x = n_{x+1}$ and $n'_{x+1} = n_x$.

- $l_x = l_{x+1}$. Since o_{x+1} will precede o_x in sequence s', obviously, it will be able to validate at least as many data points in s' as in s, therefore $n'_x \geq n_{x+1}$. As mentioned above, the set of first $x+1$ validation operators is the same in both sequences (not necessarily in the same order) and the exact same subset of input dataset D would remain unvalidated after $x+1$ operators in both. Therefore, $\sum_{i=1}^{x+1} n_i = \sum_{i=1}^{x+1} n'_i$. However, since $n_i = n'_i$ for $(i < x)$, we have that $n_x + n_{x+1} = n'_x + n'_{x+1}$. Furthermore, since $n'_x \geq n_{x+1}$ (as we have just shown), we have that $n'_{x+1} \leq n_x$.

Therefore, in both cases above it is true that $n_x + n_{x+1} = n'_x + n'_{x+1}$ and $n'_{x+1} \leq n_x$. Now, let's estimate how much different is the cost of sequence s from the cost of sequence s', when $s \approx s'$ and s' is a simple permutation of s.

In general, let's denote the *improvement* of sequence s' over sequence s as $\Delta_{s \to s'}$ and define it as follows:

$$\Delta_{s \to s'} := cost(s, D) - cost(s', D) \tag{4.21}$$

In other words, $\Delta_{s \to s'}$ specifies how much more efficient s' is with respect to s. Based on definitions of *cost* and *benefit* functions (Equations 4.3 and 4.4), it is obvious that:

$$\Delta_{s \to s'} = benefit(s', D) - benefit(s, D) \tag{4.22}$$

In the case where s' is a simple permutation of s, we get (by applying the above analysis and also by plugging in the definition of the *benefit* function from Equation 4.4):

$$
\begin{aligned}
\Delta_{s \to s'} &= \mathit{benefit}(s', D) - \mathit{benefit}(s, D) \\
&= \sum_{i=1}^{k-1} (k-i)\; n'_i \;-\; \sum_{i=1}^{k-1} (k-i)\; n_i \\
&= \sum_{i=1}^{k-1} (k-i)(n'_i - n_i) \\
&= (k-x)(n'_x - n_x) + (k-x-1)(n'_{x+1} - n_{x+1}) \\
&= (k-x)(n'_x + n'_{x+1} - n_x - n_{x+1}) + (n_{x+1} - n'_{x+1}) \\
&= n_{x+1} - n'_{x+1} \\
&\geq n_{x+1} - n_x
\end{aligned} \tag{4.23}
$$

where in the last equation we have an equality in the case when p_x and p_{x+1} are orthogonal.

Therefore, we have that whenever we perform a simple permutation that is permissible (i.e., allowed by the precedence graph), we are guaranteed to decrease the cost (or increase the benefit) of the validation sequence by at least $n_{x+1} - n_x$. Based on this simple idea, in the next section we propose a heuristic-based method for reducing the cost of validation sequences.

4.4.3 Greedy Heuristic Based on Simple Permutations

We will construct the improved validation sequence $s' =< o'_1, \ldots, o'_k >$ from $s =< o_1, \ldots, o_k >$ as follows. In the beginning, let $o'_i := o_i$ and $n'_i := n_i$ for each i. Also, let's initialize variable Δ_{Total} to 0. Then, the following steps should be performed.

1. Take all $k-1$ pairs of adjacent validation operators in s', i.e., (o'_1, o'_2), (o'_2, o'_3), $\ldots$, (o'_{k-1}, o'_k). Discard pairs (o'_i, o'_{i+1}) that have a precedence constraint between them, i.e., there is an edge from validation operator o'_i to operator o'_{i+1} in the precedence graph G. If all pairs are discarded at this point then we are done, since no more simple permutations are permissible.

2. Among the remaining pairs, choose the one with biggest $\delta_i := n'_{i+1} - n'_i$ value. Assume, we chose (o'_x, o'_{x+1}). If δ_x is negative or zero then we are done, since none of the remaining simple permutations would improve the total benefit (or decrease the total cost) of the sequence.

3. If in the previous step we found δ_x that is positive, then we exchange positions of o'_x and o'_{x+1} in s' (and, correspondingly, we also swap values of n'_x and n'_{x+1}). We also increase the value of Δ_{Total} by δ_x, i.e., $\Delta_{Total} := \Delta_{Total} + \delta_x$. That is, using a "greedy" approach, among all possible simple permutations that are permitted by the precedence graph we chose the one that improves the total benefit (or decreases total cost) the most. Then, repeat the same process from step 1.

The heuristic is not guaranteed to give us an optimal solution to the sequence optimization problem. However, it can be shown that $\Delta_{s \to s'} \geq \Delta_{Total}$, where Δ_{Total} is calculated by the heuristic. More precisely,

$$\Delta_{Total} = \sum_{i:\ (o_i \prec_s o_{i+1}) \wedge (o_{i+1} \prec_{s'} o_i)} (n_{i+1} - n_i) \tag{4.24}$$

To show that $\Delta_{s \to s'} \geq \Delta_{Total}$, one can straightforwardly extend the argument from Section 4.4.2 which showed that $\Delta_{s \to s'} \geq n_{x+1} - n_x$ for simple strongly equivalent permutations. Furthermore, it can be shown that the above lower bound is tight, i.e., $\Delta_{s\rightarrow s'} = \Delta_{Total}$, when *all* simple permutations performed by the above heuristic involved pairs of operators with orthogonal predicates, i.e., when all permutations are *very strongly equivalent.*

Consider, for example, a validation sequence $s =< o_1, o_2, o_3 >$. Assume, that this sequence was used to validate dataset D consisting of 1000 data elements, and that o_1 validated 200 (n_1), o_2 validated 100 (n_2), and o_3 validated 400 (n_3) of them. Thus, $cost(s, D) = 1000 + 800 + 700 = 2500$. Furthermore, let's assume that our heuristic produced the following equivalent permutation $s' =< o_3, o_1, o_2 >$ by first swapping operators o_2 and o_3, and then o_1 and o_3. According to the heuristic, $\Delta_{Total} = (n_3 - n_2) + (n_3 - n_1) = (400 - 100) + (400 - 200) = 500$. Therefore, we are guaranteed to have $cost(s', D) \leq 2000$.

Because at every iteration we perform a simple permutation that satisfies the precedence graph, the sequence on every iteration remains strongly equivalent to the original sequence. In addition, note that on every iteration we perform a simple permutation only if $\delta_x > 0$. Because of this, we are guaranteed not to "swap" the same two validation operators more than once. Therefore, the maximal number of iterations performed by the above heuristic is equal to the maximal number of "swaps" you can do in a sequence without "swapping" anything twice. Obviously, this number is equal to the number of possible validation operator inversions, i.e., $k(k-1)/2$ (assuming the validation sequence has k operators). Moreover, since the computational complexity of a single iteration is $O(k)$ (i.e., dealing with $k-1$ pairs of operators), the worst case computational complexity of the heuristic is $O(k^3)$.

This heuristic assumes that the precedence graph is already created. Creating a graph is a separate (predicate-specific) problem, and we address this problem for cardinality predicate-based validation operators in the next section.

4.5 Improving Itemset Validation Sequences That Are Based on Cardinality Predicates

In this section, we will show how to construct the precedence graph for the validation sequences that are based on cardinality predicates. After the graph is constructed, the heuristic from Section 4.4.3 can be directly used to generate the improved validation sequence.

4.5.1 Orthogonality of Atomic Cardinality Predicates with Singleton Cardinality Sets

Let's consider two cardinality predicates $C_{A_1}^{k_1}$ and $C_{A_2}^{k_2}$, where $A_1 \subseteq \mathcal{I}$, $A_2 \subseteq \mathcal{I}$, and k_1 and k_2 are cardinality numbers, i.e., $k_1 \in [A_1]$ and $k_2 \in [A_2]$. In other words, let's consider two cardinality predicates with singleton cardinality sets. The following theorem presents necessary and sufficient conditions for $C_{A_1}^{k_1} \perp C_{A_2}^{k_2}$.

Theorem 25 *Let $p_1 := C_{A_1}^{k_1}$ and $p_2 := C_{A_2}^{k_2}$. $p_1 \perp p_2$ if and only if either of the following inequalities hold:* [6]

$$|A_2 - A_1| < k_2 - k_1 \tag{4.25}$$

$$|A_1 - A_2| < k_1 - k_2 \tag{4.26}$$

[6]Obviously, (4.25) and (4.26) cannot both hold at the same time, otherwise $k_1 + |A_2 - A_1| < k_2 < k_1 - |A_1 - A_2| \Rightarrow k_1 < k_1$ (since both $|A_2 - A_1|$ and $|A_1 - A_2|$ are non-negative).

▶ Claim: if (4.25) holds, then $(p_1 \perp p_2)$.

Let's denote $A_{1-2} := A_1 - A_2$, $A_{2-1} := A_2 - A_1$, and $A_{12} := A_1 \cap A_2$. Note, that sets A_{1-2}, A_{2-1}, and A_{12} are pairwise disjoint.

Assume, (4.25) holds, i.e., $k_1 + |A_{2-1}| < k_2$. We want to prove that $p_1 \perp p_2$. Let's assume otherwise, that p_1 and p_2 are not orthogonal. Hence, by the definition of orthogonality, there must exist an itemset I such that both $p_1(I)$ and $p_2(I)$ are `True`. More specifically, $(\exists I \subseteq \mathcal{I})\ ((|I \cap A_1| = k_1) \wedge (|I \cap A_2| = k_2))$. Let's analyze $|I \cap A_1| = k_1$ and $|I \cap A_2| = k_2$ further.

By analyzing $|I \cap A_2| = k_2$ further, we have $k_2 = |I \cap A_2| = |I \cap (A_{2-1} \cup A_{12})| = |(I \cap A_{2-1}) \cup (I \cap A_{12})| = |I \cap A_{2-1}| + |I \cap A_{12}|$, since $A_{2-1} \cap A_{12} = \emptyset$. Therefore, $|I \cap A_{12}| = k_2 - |I \cap A_{2-1}|$. Furthermore, since $k_2 > k_1 + |A_{2-1}|$ and $|I \cap A_{2-1}| \leq |A_{2-1}|$, we have $|I \cap A_{12}| = k_2 - |I \cap A_{2-1}| > k_1 + |A_{2-1}| - |A_{2-1}| = k_1$. That is, $|I \cap A_{12}| > k_1$.

On the other hand, by analyzing $|I \cap A_1| = k_1$ further, we have $k_1 = |I \cap A_1| = |I \cap (A_{1-2} \cup A_{12})| = |(I \cap A_{1-2}) \cup (I \cap A_{12})| = |I \cap A_{1-2}| + |I \cap A_{12}| \geq |I \cap A_{12}|$, since $|I \cap A_{1-2}| \geq 0$. Hence, $|I \cap A_{12}| \leq k_1$.

Based on our assumption that p_1 and p_2 are not orthogonal, we derived both $|I \cap A_{12}| > k_1$ and $|I \cap A_{12}| \leq k_1$. Therefore, our assumption was incorrect. Hence, $p_1 \perp p_2$.

Note that, because of the symmetry of the orthogonality relation, the proof for (4.26) $\Rightarrow$ $(p_1 \perp p_2)$ is essentially the same as for (4.25) $\Rightarrow$ $(p_1 \perp p_2)$ described above, only cardinality predicates p_1 and p_2 need to switch places.

Claim: $(p_1 \perp p_2) \Rightarrow ((k_1 + |A_2 - A_1| < k_2) \vee (k_2 + |A_1 - A_2| < k_1))$. We will prove this by showing that if both (4.27) and (4.28) inequalities hold, then p_1 and p_2 are not orthogonal.

$$|A_2 - A_1| \geq k_2 - k_1 \tag{4.27}$$

$$|A_1 - A_2| \geq k_1 - k_2 \tag{4.28}$$

Let's consider 4 cases: (1) $k_1 \geq |A_{12}|$ and $k_2 \geq |A_{12}|$; (2) $k_1 \geq |A_{12}|$ and $k_2 < |A_{12}|$; (3) $k_1 < |A_{12}|$ and $k_2 \geq |A_{12}|$; (4) $k_1 < |A_{12}|$ and $k_2 < |A_{12}|$. We will consider each of these cases individually.

Case (1): $k_1 \geq |A_{12}|$ and $k_2 \geq |A_{12}|$.

By definition, $k_2 \leq |A_2|$. Therefore, $k_2 \leq |A_2| = |A_{2-1}| + |A_{12}| \leq |A_{2-1}| + k_1$. Hence, (4.27) holds automatically.

Similarly, $k_1 \leq |A_1| = |A_{1-2}| + |A_{12}| \leq |A_{1-2}| + k_2$. Hence, (4.28) holds automatically.

We will show that cardinality predicates p_1 and p_2 are not orthogonal by constructing an itemset I such that $|I \cap A_1| = k_1$ and $|I \cap A_2| = k_2$.

We will construct three sets, α, β, and γ, in the following manner. Let $\alpha = A_{12}$. Since $k_1 \geq |A_{12}|$, let β be any subset of A_{1-2} such that $|\beta| = k_1 - |A_{12}|$. Similarly, Since $k_2 \geq |A_{12}|$, let γ be any subset of A_{2-1} such that $|\gamma| = k_2 - |A_{12}|$.

Consider itemset $I := \alpha \cup \beta \cup \gamma$. $|I \cap A_1| = |I \cap (A_{12} \cup A_{1-2})| = |(I \cap A_{12}) \cup (I \cap A_{1-2})| = |I \cap A_{12}| + |I \cap A_{1-2}| = |\alpha| + |\beta| = k_1$. Therefore, I satisfies p_1. Similarly, $|I \cap A_2| = |I \cap (A_{12} \cup A_{2-1})| = |(I \cap A_{12}) \cup (I \cap A_{2-1})| = |I \cap A_{12}| + |I \cap A_{2-1}| = |\alpha| + |\gamma| = k_2$. Therefore, I satisfies p_2. Hence, p_1 and p_2 are not orthogonal.

Case (2): $k_1 \geq |A_{12}|$ and $k_2 < |A_{12}|$.

As in case (1), $k_2 \leq |A_2| = |A_{2-1}| + |A_{12}| \leq |A_{2-1}| + k_1$. Hence, (4.27) holds automatically.

Assume, (4.28) also holds, i.e., $k_1 \leq k_2 + |A_{1-2}|$.

We will construct two sets, α and β, in the following manner. Since $k_2 < |A_{12}|$, let α be any subset of A_{12} such that $|\alpha| = k_2$. Also, since $k_1 - k_2 \leq |A_{1-2}|$, let β be any subset of A_{1-2} such that $|\beta| = k_1 - k_2$.

Consider itemset $I := \alpha \cup \beta$. $|I \cap A_1| = |I \cap (A_{12} \cup A_{1-2})| = |(I \cap A_{12}) \cup (I \cap A_{1-2})| = |I \cap A_{12}| + |I \cap A_{1-2}| = |\alpha| + |\beta| = k_1$. Therefore, I satisfies p_1. Similarly, $|I \cap A_2| = |I \cap (A_{12} \cup A_{2-1})| = |(I \cap A_{12}) \cup (I \cap A_{2-1})| = |I \cap A_{12}| + |I \cap A_{2-1}| = |\alpha| = k_2$. Therefore, I satisfies p_2. Hence, p_1 and p_2 are not orthogonal.

Case (3): $k_1 < |A_{12}|$ and $k_2 \geq |A_{12}|$.

Because of the symmetry of the orthogonality relation, the proof for case (3) is essentially the same as for case (2).

Case (4): $k_1 < |A_{12}|$ and $k_2 < |A_{12}|$.

Assume $k_1 \geq k_2$ (without loss of generality). In this case, (4.27) holds automatically, because $k_2 \leq k_1 \leq k_1 + |A_{2-1}|$.

Assume, (4.28) also holds, i.e., $k_1 \leq k_2 + |A_{1-2}|$.

As in case (2), we will construct two sets, α and β, in the following manner. Since $k_2 < |A_{12}|$, let α be any subset of A_{12} such that $|\alpha| = k_2$. Also, since $0 \leq k_1 - k_2 \leq |A_{1-2}|$, let β be any subset of A_{1-2} such that $|\beta| = k_1 - k_2$.

Consider itemset $I := \alpha \cup \beta$. $|I \cap A_1| = |I \cap (A_{12} \cup A_{1-2})| = |(I \cap A_{12}) \cup (I \cap A_{1-2})| = |I \cap A_{12}| + |I \cap A_{1-2}| = |\alpha| + |\beta| = k_1$. Therefore, I satisfies p_1. Similarly, $|I \cap A_2| = |I \cap (A_{12} \cup A_{2-1})| = |(I \cap A_{12}) \cup (I \cap A_{2-1})| = |I \cap A_{12}| + |I \cap A_{2-1}| = |\alpha| = k_2$. Therefore, I satisfies p_2. Hence, p_1 and p_2 are not orthogonal. ◀

Therefore, in order two calculate whether $C_{A_1}^{k_1}$ and $C_{A_2}^{k_2}$ are orthogonal we only need to check whether one of Equations 4.25 and 4.26 holds. In order to perform this check we need to compute numbers $|A_2 - A_1|$ and $|A_1 - A_2|$. Obviously, this can be done in $O(|A_1| + |A_2|)$ time. Since $A_1 \subseteq \mathcal{I}$ and $A_2 \subseteq \mathcal{I}$, the worst case computational complexity is $O(|\mathcal{I}|)$.

Based on the above theorem we can also show that if sets A_1 and A_2 are disjoint then $C_{A_1}^{k_1}$ and $C_{A_2}^{k_2}$ are not orthogonal.

Corollary 26 *If $A_1 \cap A_2 = \emptyset$ then cardinality predicates $C_{A_1}^{k_1}$ and $C_{A_2}^{k_2}$ are not orthogonal.*

▶ If sets A_1 and A_2 are disjoint, then $|A_1 - A_2| = |A_1|$ and $|A_2 - A_1| = |A_2|$. Also, by definition, $0 \leq k_1 \leq |A_1|$ and $0 \leq k_2 \leq |A_2|$.

Assume, (4.25) holds. Then, $k_1 + |A_2| < k_2 \leq |A_2| \Rightarrow |A_2| < |A_2|$. Contradiction.

Similarly, assume that (4.26) holds. Then, $k_2 + |A_1| < k_1 \leq |A_1| \Rightarrow |A_1| < |A_1|$. Contradiction.

From Theorem 25 we have that $C_{A_1}^{k_1}$ and $C_{A_2}^{k_2}$ are not orthogonal, since neither (4.26) nor (4.25) holds. ◀

4.5.2 Orthogonality of Atomic Cardinality Predicates with Arbitrary Cardinality Sets

In this section we will present a method for checking whether two arbitrary cardinality predicates are orthogonal. Let $C_{X_1}^{S_1}$ and $C_{X_2}^{S_2}$, where $X_1 \subseteq \mathcal{I}$, $X_2 \subseteq \mathcal{I}$, $S_1 \subseteq [X_1]$, and $S_2 \subseteq [X_2]$.

Based on Lemma 1 (in Chapter 2), we can rewrite the above cardinality predicates as:

$$\begin{aligned} C_{X_1}^{S_1} &= C_{X_1}^{k_1} \vee C_{X_1}^{k_2} \vee \ldots \vee C_{X_1}^{k_{n_1}} \\ C_{X_2}^{S_2} &= C_{X_2}^{l_1} \vee C_{X_2}^{l_2} \vee \ldots \vee C_{X_2}^{l_{n_2}} \end{aligned}$$

where $S_1 = \{k_1, k_2, \ldots, k_{n_1}\}$ and $S_2 = \{l_1, l_2, \ldots, l_{n_2}\}$.

Furthermore, from Corollary 16 we have that $C_{X_1}^{S_1} \perp C_{X_2}^{S_2}$ if and only if $C_{X_1}^{k_i} \perp C_{X_2}^{l_j}$ for all $i \in \{1, \ldots, n_1\}$ and all $j \in \{1, \ldots, n_2\}$. Therefore, we can use the result from the previous section for checking the orthogonality of cardinality predicates with singleton cardinality sets (there will be $|S_1| \times |S_2|$ such checks). Note, that comparison sets X_1 and X_2 are the same in all checks. Therefore, we need to calculate numbers $|X_2 - X_1|$ and $|X_1 - X_2|$ only once. Therefore, computational complexity of the whole process is $O(|X_1| + |X_2| + |S_1| \times |S_2|)$, which in the worst case scenario could be $O(|\mathcal{I}|^2)$.

4.5.3 Orthogonality of Cardinality Predicate Conjunctions

In this section we will present a sufficient condition for determining whether two cardinality predicate conjunctions are orthogonal. Let $p_1 := C_{X_1}^{S_1} \wedge \ldots \wedge C_{X_{m_1}}^{S_{m_1}}$ and $p_2 := C_{Y_1}^{T_1} \wedge \ldots \wedge C_{Y_{m_2}}^{T_{m_2}}$, where $X_i \subseteq \mathcal{I}$, $S_i \subseteq [X_i]$, $Y_j \subseteq \mathcal{I}$, and $T_j \subseteq [Y_j]$.

From Corollary 14 we have that $p_1 \perp p_2$ if $(\exists i \in \{1, \ldots, m_1\})$ $(\exists j \in \{1, \ldots m_2\})$ $(C_{X_i}^{S_i} \perp C_{Y_j}^{T_j})$. Note, that this is only a sufficient condition, but not necessary. We can use the method provided in the previous section to check whether two given cardinality predicates are orthogonal. There would be $m_1 \times m_2$ such checks in the worst case (we can stop as soon as one of the checks gives a positive result).

4.5.4 Applying the Heuristic: Experiments

Now we have all we need to use the heuristic presented in Section 4.4.3 for cardinality predicate-based validation sequences for itemset validation. Let $s = < o_1, \ldots, o_k >$ be a validation sequence, where $o_i = (l_i, p_i)$ and p_i is a conjunction of atomic cardinality predicates. Also, let $n_1, \ldots, n_k$ be the numbers of itemsets from a given dataset D that were validated by validation operators $o_1, \ldots, o_k$ respectively.

Then, we can build a precedence graph G_s directly using the procedure described in Section 4.4.1. Subsequently, we can directly use the heuristic presented in Section 4.4.3 to generate sequence s' ($s' \approx s$) that improves the cost of validation sequence s.

We performed a simple experiment to illustrate how the proposed heuristic works. More specifically, we created a base set $\mathcal{I}$ that consisted of 50 elements, i.e., $|\mathcal{I}| = 50$. We generated the dataset of 5,000,000 random itemsets, where the smallest itemset was of size 1, and the largest itemset was of size 20 (average itemset size was between 7 and 8). Then we arbitrarily specified 40 validation operators. 15 of them were specified manually, i.e., using set validation language SVL. The remaining 25 were specified using our proposed grouping method, i.e., after performing various groupings, we chose to validate some of the groups discovered by the grouping algorithm. Each validation operator $o_i = (l_i, p_i)$ was specified using one of two labels, i.e., `Accept` or `Reject`. Also, each predicate p_i was specified as a conjunction of atomic cardinality predicates.

After initial validation, about 88% of the dataset was validated (i.e., accepted or rejected). More importantly, the value of the cost function for our initial sequence s was above 74,000,000. Using the proposed heuristic, we obtained sequence s', the value of the cost function for which was approximately 53,000,000. Therefore, while the heuristic obviously does not guarantee the optimal solution, it could still significantly (e.g., by about 28% in our case) improve the performance of the validation sequence.

4.6 Future Work

Note, that we illustrated the heuristic using a validation sequence for itemsets. The same can also be done for rules. Since most of the results derived in this chapter pertain to general validation sequences (i.e., not just itemset validation sequences), these results will also be true specifically for rule validation sequences. Furthermore, since we use cardinality predicates not only for itemset validation, but for rule validation as well, similar methods can be used for determining the orthogonality of rule validation predicates (for the purpose of building the precedence graph), as were used for itemset validation predicates (described in Sections 4.5.1, 4.5.2, and 4.5.3). However, in the case of rule predicates, some additional orthogonality results need to be derived, since each rule has several different itemsets (i.e., antecedent, consequent, whole rule) to which predicates can be applied. For example, the orthogonality of rule cardinality predicates $C_{X_1}^{S_1}[body]$ and $C_{X_2}^{S_2}[body]$ can be calculated simply by calculating the orthogonality of cardinality predicates $C_{X_1}^{S_1}$ and $C_{X_2}^{S_2}$ using the methods described in this chapter. The same obviously cannot be said about calculating the orthogonality of $C_{X_1}^{S_1}[body]$ and $C_{X_2}^{S_2}[rule]$. We plan to examine this issue in the future.

Our ability to improve the performance of validation sequences is in part due to the fact that we use a fairly simple class of predicates (i.e., cardinality predicates) in our validation operators. Because of their simplicity, we were able to analyze them and derive the orthogonality conditions, which allowed us to determine which permutations are “safe” and which are not. Therefore, another interesting area for future research would be to see if we could enhance the class of predicates used in validation operators (i.e., to give even more flexibility to the domain expert), but at the same time to still be able to derive “workable” orthogonality conditions.

Part II

Practical Applications of Expert-Driven Validation Techniques

Chapter 5

Validating Rule-Based User Models in Personalization Applications

5.1 Motivations and Related Work

In various e-commerce applications, ranging from dynamic Web content presentation, to personalized ad targeting, to individual recommendations to the customers, *personalization* has become an important business problem [63, 64]. For example, the personalized version of Yahoo (myYahoo) provides to its customers personalized content, such as local weather or interesting events in the area where the customer lives. As another example, Amazon.com and Moviecritic.com provide recommendations on what books to read and movies to see respectively. In general, there is a very strong interest in the industry in personalized (one-to-one) marketing applications [63, 10] and in recommender systems [23, 45, 12, 74] that provide personal recommendations to individual users for products and services that might be of interest to them. The advantages of these personalized approaches over more traditional segmentation methods are well documented in the literature [63, 64, 10].

One of the key issues in developing such e-commerce applications is the problem of constructing accurate and comprehensive *profiles* of individual customers that provide the most important information describing who the customers are and how they behave. This problem is so important for building successful e-commerce applications that some authors propose that companies treat customer profiles as key economic assets in addition to more traditional assets such as plant, equipment and human assets [39, 40]. Although some work on how to construct personal user profiles has been published in the academic literature (and we will review it below), a lot of work has been done in the industry as well.

There are two main approaches to addressing the profiling problem developed by different companies. In the first approach, taken by such companies as Engage Technologies [www.engage.com] and Personify [www.personify.com], profiles are constructed from the customers' demographic and transactional data and contain important *factual* information about the customers. Examples of such factual information include (a) demographic attributes, such as age, address, income and a shoe size of a customer, and (b) certain facts extracted from his or her transactional data, such as that the average and maximal purchase amounts of that customer over the last year were $23 and $127 respectively, or that the favorite newspaper of a particular Travelocity customer is the New York Times and her favorite vacation destination is Almond Beach Club in Barbados. This factual data comprises the profile of a customer and is typically stored in a relational table.

According to the other approach, taken by such companies as Art Technology Group [www.atg.com] and BroadVision [www.broadvision.com], customer profiles contain not only factual information but also *rules* that describe on-line behavioral activities of the customers. However, these rules are defined by experts (e.g., a marketing manager working on a particular marketing application). For example, a manager may specify that if a customer of a certain type visits the Web site of the on-line groceries shopping company ShopTillUStop.com on Sunday evenings, that customer should be shown the discount coupons for diapers. This approach differs from the previous approach in that the profiles contain behavioral rules in addition to the factual information about the customer. However, these behavioral rules are not constructed in a truly one-to-one manner since these rules are specified by the expert rather than learned from the data and are applicable only to *groups* of customers.

In addition to the developments in the industry, the profiling problem was also studied in the data mining academic community in [30, 31, 6, 2, 24]. In particular, [30, 31] studied this problem within the context of fraud detection in the cellular phone industry. This was done by learning rules pertaining to individual customers from the cellular phone usage data using the rule learning system RL [26]. However, these discovered rules were used not for the purpose of understanding the personal behavior of individual customers, but rather to instantiate generalized profilers that are applicable to several customer accounts for the purpose of learning fraud conditions.

[6] study the problem of on-line mining of customer profiles specified with asso-

ciation rules, where the body of a rule refers to the demographic information of a user, such as age and salary, and the head of a rule refers to transactional information, such as purchasing characteristics. Moreover, [6] present a multidimensional indexing structure for mining such rules. The proposed method provides a new approach to deriving association rules that *segment* users based on their transactional characteristics. However, it does not derive behavior of an *individual* user in a one-to-one fashion [63].

Still another approach to the profiling problem was presented by [24] in the context of providing personalized Web search. In this approach the user profile consists of a Web Access Graph summarizing Web access patterns by the user, and a Page Interest Estimator characterizing interests of the user in various Web pages. Although the approach presented by [24] goes beyond building simple factual profiles, these profiles are specialized to be used in specific Web-related applications, i.e., to provide personalized Web search. This means that they do not attempt to capture all aspects of the on-line behavior of individual users. One specific consequence of this specialization is that [24] does not use behavioral rules as a part of a user profile.

In [2], we presented an initial approach to the profiling problem. The expanded and improved version of our methodology is presented here (as well as in [3, 5]). In particular, we present a framework for building behavioral profiles of individual users. These behavioral profiles contain not only factual information about the users, but also capture more comprehensive behavioral information using association rules that are learned from user transactional histories using various data mining methods. However, there are caveats to this approach due to the nature of personalization applications. In particular, as mentioned in Chapter 1, the behavioral rules learned about individual users can be unreliable, irrelevant, or obvious. Therefore, post-analysis, including *rule validation*, becomes an important issue for building accurate personalized profiles of users. This validation process is performed by the domain expert who can iteratively apply various rule validation operators. In particular, we describe how different validation tools described in earlier chapters can be used to validate individual user profiles in personalization applications. Finally, in this chapter we also describe a case study of testing the proposed profile validation method on a marketing application.

5.2 A Proposed Approach to Profiling

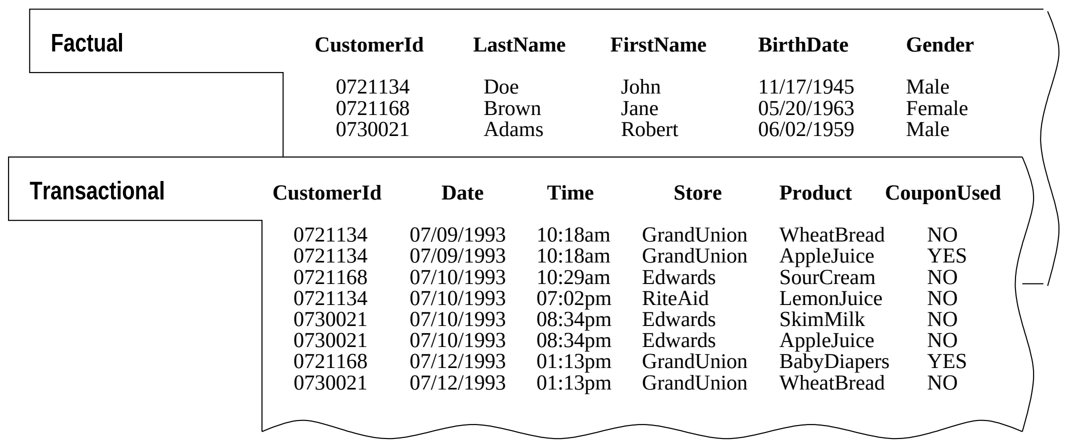
5.2.1 Defining User Profiles

In order to explain what user profiles are and how they can be constructed, we first focus on the data that is used for constructing these profiles.

Data Model

Various e-commerce personalization applications can contain different types of data about individual users. However, this data can be classified in many applications into two basic types – *factual* and *transactional.* Simply put, the factual data describes who the user *is* and the transactional data describes what the user *does.*

> **Example 5** In a marketing application based on purchasing histories of customers, the factual data could be the demographic data of customers, such as name, gender, birth date, address, salary, social security number,



Factual

CustomerId	LastName	FirstName	BirthDate	Gender
0721134	Doe	John	11/17/1945	Male
0721168	Brown	Jane	05/20/1963	Female
0730021	Adams	Robert	06/02/1959	Male

Transactional

CustomerId	Date	Time	Store	Product	CouponUsed
0721134	07/09/1993	10:18am	GrandUnion	WheatBread	NO
0721134	07/09/1993	10:18am	GrandUnion	AppleJuice	YES
0721168	07/10/1993	10:29am	Edwards	SourCream	NO
0721134	07/10/1993	07:02pm	RiteAid	LemonJuice	NO
0730021	07/10/1993	08:34pm	Edwards	SkimMilk	NO
0730021	07/10/1993	08:34pm	Edwards	AppleJuice	NO
0721168	07/12/1993	01:13pm	GrandUnion	BabyDiapers	YES
0730021	07/12/1993	01:13pm	GrandUnion	WheatBread	NO

Figure 5.1: Fragment of data in a marketing application.

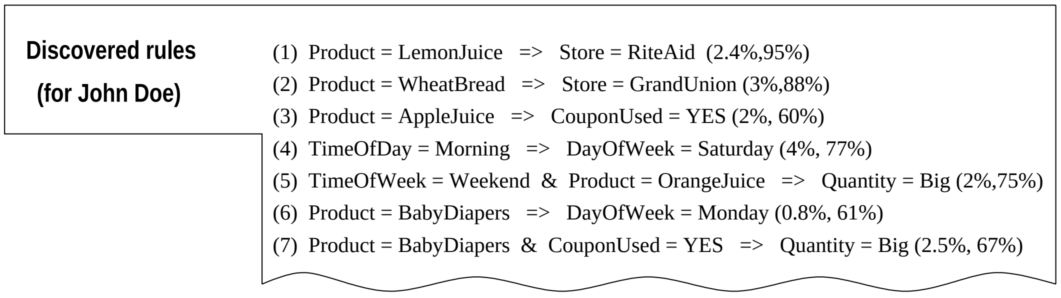
etc. The transactional data could consist of records of purchases the customer made over a specific period of time. A record of a purchase could include the following attributes: date of purchase, product purchased, amount of money spent, use or no use of a coupon, value of a coupon if used, discount applied, etc. An example of a fragment of such data is presented in Figure 5.1.

Profile Model

A profile is a collection of information that describes a user. One of the open issues in the profile construction process is what information should be included in a user profile. In their simplest form, user profiles contain *factual* information that can be described as a set of individual facts that, for example, can be stored in a record of a relational database table. These facts may include demographic information about the user, such as name, address, date of birth, and gender, that are usually taken from the user description data. The facts can also be derived from the transactional and item description data. Examples of such facts are *"the favorite beer of user ALW392 is Heineken"*, *"the biggest purchase made by ALW392 was for $237"*, *"the favorite movie star of ALW392 is Harrison Ford."* The construction of factual profiles is a relatively simple and well-understood problem, and keyword-based factual profiles have been extensively used in recommender systems.

A user profile can also contain a *behavioral* component that describes behavior of the user learned from his or her transactional history. One way to define user behavior is with a set of conjunctive rules, such as association [9] or classification rules [19]. Examples of rules describing user behavior are: *"when user ALW392 comes to the Web site Y from site Z, she usually returns back to site Z immediately"*, *"when shopping on the NetGrocer.com Web site on weekends, user ALW392 usually spends more than $100 on groceries"*, *"whenever user ALW392 goes on a business trip to Los Angeles, she stays there in expensive hotels."* More examples of the association rules discovered for a particular customer are presented in Figure 5.2.

The use of rules in profiles provides an intuitive, declarative and modular way to describe user behavior and was advocated in [31, 2]. These rules can either be defined by domain experts, as is done in systems developed by BroadVision and Art Technology Group, or derived from the transactional data of a user using various data mining methods. We describe this derivation process in the next section.



Discovered rules (for John Doe)

(1) Product = LemonJuice => Store = RiteAid (2.4%,95%)
(2) Product = WheatBread => Store = GrandUnion (3%,88%)
(3) Product = AppleJuice => CouponUsed = YES (2%, 60%)
(4) TimeOfDay = Morning => DayOfWeek = Saturday (4%, 77%)
(5) TimeOfWeek = Weekend & Product = OrangeJuice => Quantity = Big (2%,75%)
(6) Product = BabyDiapers => DayOfWeek = Monday (0.8%, 61%)
(7) Product = BabyDiapers & CouponUsed = YES => Quantity = Big (2.5%, 67%)

Figure 5.2: Sample of association rules discovered for an individual customer in a marketing application.

5.2.2 Profile Construction

Since we focus on personalization applications, rule discovery methods should be applied *individually* to the transactional data of *every* user, thus, capturing truly personal behavior of each user.

Such rules can be discovered using various data mining algorithms. For example, to discover association rules, we can use *Apriori* [9] and its numerous variations. Similarly, to discover classification rules, we can use *CART* [19], *C4.5* [68], or other classification rule discovery methods. We would like to point out that our approach *is not restricted* to any specific representation of data mining rules and their discovery methods.

Note that, since data mining methods discover conjunctive rules for *individual* customers, these methods work better for the applications containing many transactions for individual customers, such as credit card, grocery shopping, on-line browsing, and certain stock trading applications. In applications containing few transactions, such as car purchasing or air travel, individual rules are generated from relatively few transactions for most customers and tend to be statistically less reliable.

One of the serious problems with many rule discovery methods is that they tend to generate large numbers of patterns, and often many of them, while being statistically acceptable, are trivial, spurious, or just not relevant to the application at hand [66, 73, 53, 20, 78, 60, 61]. Therefore, post-analysis of discovered rules becomes an important issue, since there is a need to *validate* the discovered rules. For example, assume that a data mining method discovered the rule stating that, whenever customer ALW392 goes on a business trip to Los Angeles, she mostly stays in expensive hotels there. In particular, assume that ALW392 went to Los Angeles 7 times over the past 2 years and 5 out of 7 times stayed in expensive hotels. Before this rule can be placed into ALW392's profile, it needs to be validated, since it may not be immediately clear whether this rule really captures the behavior of ALW392, or whether it constitutes a spurious correlation or is simply not relevant to the application at hand.

Therefore, we focus on the two phases of the profile building process: rule discovery and validation (Figure 5.3). In the next section we present methods for validating behavioral rules in user profiles.

5.3 Validation of User Profiles

As we mentioned earlier, a common way to perform the post-analysis of data mining results is to let the *domain expert* perform this task, and several data mining

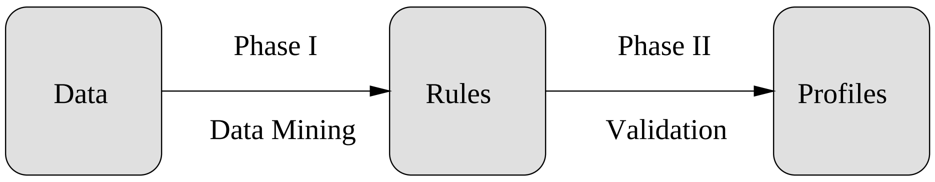


Figure 5.3: The simplified profile building process.

systems support this capability. For example, MineSet [22] provides a wide range of visualization techniques allowing the end-user to examine visually the results discovered by its data mining tools and thus evaluate the quality of these results.

In our approach, individual rules discovered during the data mining stage are validated by the expert, and, depending on how well they represent the actual behaviors of the users, some rules are "accepted" and some "rejected" by the expert. That is, in this particular application, the domain expert usually has the label set $\mathcal{L}$ that consists of two labels: `Accept` and `Reject`. After the validation process is complete, the accepted rules form the behavioral profiles of users.

One of the main issues about validating individual rules of users by a human expert is scalability. In many e-commerce personalization applications the number of users tends to be very large. For example, the number of registered users at major Web sites is measured in millions. If we discover a hundred rules per customer on average, then the total number of rules for such sites would be measured in hundreds of millions. Therefore, it would be impossible for a human expert to validate all the discovered rules on a one-by-one basis in such applications.

We address this problem by providing a framework allowing the human expert

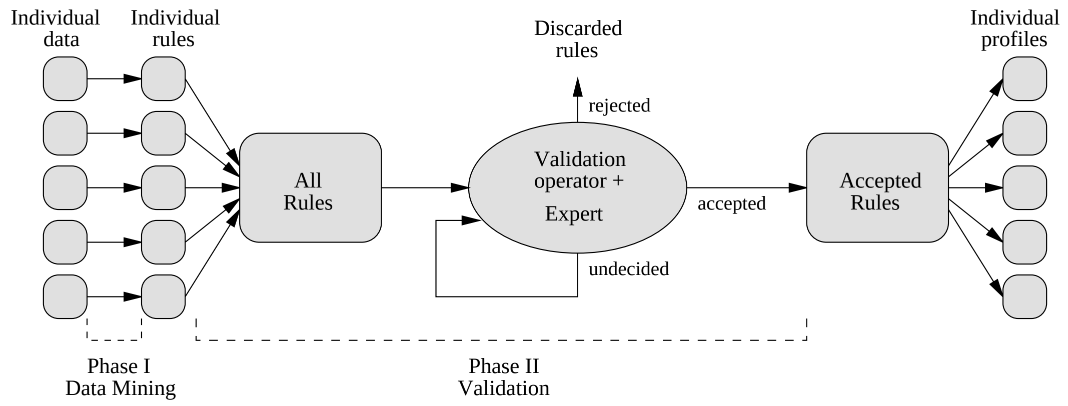


Figure 5.4: The profile building process.

validate *large numbers* of rules (instead of individual rules) at a time with relatively little input from the expert. This is done by applying different *rule validation operators* that were introduced earlier in the dissertation. Then rule validation becomes an *iterative* process and is described in Figure 5.4. In particular, the profile building activity is divided into two phases. In Phase I, the data mining phase, rules describing behaviors of individual users are generated from the users' transactional data as was described in Section 5.2.2.

Phase II constitutes the rule validation process. Rule validation, unlike rule discovery (Phase I), is not performed separately for each user, but rather for *all* users at once. The reason we propose performing rule validation collectively (rather than individually) for all users is that there are usually many similar or even identical rules across different users. For example, the rule *"when shopping on the NetGrocer.com Web site on weekends, user ALW392 usually spends more than \$100 on groceries"* can be common to many users. In addition, although rules *"when user*

ALW392 comes to our Web site from site Y, she usually returns back to site Y immediately," and *"when user KTL158 comes to our Web site from site Z, she usually returns back to site Z immediately,"* are not identical, they are quite "similar" and can be examined by the expert together. The collective rule validation allows one to deal with such common rules once, thus significantly reducing validation effort. Therefore, in the beginning of Phase II, rules from all the users are collected into one set. Each rule is tagged with the ID of the user to which it belongs, so that each accepted rule could be put into the profile of that user at the end of the validation phase.

After rules from all users are collected into one set, the rule validation process is performed as a second part of Phase II. This process is described in Figure 5.5. All rules discovered during Phase I (denoted by R_{all} in Figure 5.5) are considered unvalidated. The human expert selects various validation operators and applies them successively to the set of unvalidated rules. The application of each validation operator results in validation of some of the rules. In particular, some rules get accepted and some rejected (sets O_{acc} and O_{rej} in Figure 5.5). Then the next validation operator would be applied to the set of the remaining unvalidated rules (set R_{unv}). This validation process stops when the *TerminateValidationProcess* condition is met. This condition is set by the human expert and is discussed later in this section. After the validation process is stopped, the set of all the discovered rules (R_{all}) is split into three disjoint sets: accepted rules (R_{acc}), rejected rules (R_{rej}), and possibly some remaining unvalidated rules (R_{unv}). At the end of Phase II all the accepted rules are put into the behavioral profiles of their respective users. This is possible, because all the rules have been tagged with the user ID in

```
Input:   Set of all discovered rules $R_{all}$.
Output:  Mutually disjoint sets of rules $R_{acc}$, $R_{rej}$, $R_{unv}$,
         such that $R_{all} = R_{acc} \cup R_{rej} \cup R_{unv}$.

(1)  $R_{unv} := R_{all}$,  $R_{acc} := \emptyset$,  $R_{rej} := \emptyset$.
(2)  while (not TerminateValidationProcess()) begin
(3)     Expert specifies a validation operator (e.g., $o$).
(4)     $o$ is applied to $R_{unv}$.  Result:  disjoint sets $O_{acc}$ and $O_{rej}$.
(5)     $R_{unv} := R_{unv} - O_{acc} - O_{rej}$,  $R_{acc} := R_{acc} \cup O_{acc}$,  $R_{rej} := R_{rej} \cup O_{rej}$.
(6)  end
```

Figure 5.5: An algorithm for the rule validation process.

the beginning of Phase II as described above.

As was already stated above and shown in Figure 5.5, various validation operators are successively applied to the set of the unvalidated rules until the stopping criterion *TerminateValidationProcess* is reached. The stopping criterion can be specified by the expert and may include such conditions as:

- Only few rules remain unvalidated;
- Only few rules are being validated at a time by one or several validation operators;
- The total elapsed validation time exceeds the predetermined validation time.

In this section we described the overall validation process. We present the detailed description of various specific validation operators in the next section.

5.4 Validation Tools

Validation operators provide a way for the domain expert to examine multiple rules at a time. This examination process can be performed in the following two ways. First, the expert may already know some types of rules that he or she wants to examine and accept or reject based on the prior experience. For this purpose the domain expert can use a template-based rule validation operator that is based on the SVL language introduced earlier as well as some interestingness-based filtering operators. Second, the expert may not know all the relevant types of rules in advance, and it is important to provide methods that *group* discovered rules into classes that he or she can subsequently examine and validate. For this purpose the domain expert can use our similarity-based rule grouping method that was described in detail in Chapter 3. In addition, we describe other tools that can be used in the validation process, including visualization, statistical analysis, and browsing tools.

5.4.1 Template-based rule filtering

Template-based rule filtering tool allows the domain expert to specify in general terms the types of rules that he or she either wants to accept (*accepting* template) or reject (*rejecting* template), i.e., it allows the expert to specify validation operators. After a template is specified, unvalidated rules are "matched" against it. Rules that match an accepting template are accepted and put into user profiles, and rules that match a rejecting template are rejected. Rules that do not match a template remain unvalidated.

This template-based rule filtering tool allows one to define various constraints that the expert can impose on:

- *The syntactic structure of the body (antecedent) and the head (consequent) of the rule.* This part of the tool is based on SVL. As we have shown, the computational complexity of predicates specified with SVL (i.e., cardinality predicates) is linear in the total size of the rules to be filtered.

- *Basic statistical parameters of the rule.* During the rule validation process, restrictions on basic statistical parameters (e.g., support and confidence for association rules) can be imposed using the following template:

  $$\textbf{STATS}\ \{\ par_1\ op_1\ val_1,\quad par_2\ op_2\ val_2,\quad \ldots\ \}$$

  where par_i is the name of a statistical parameter (e.g., *conf* for confidence, *supp* for support); op_i is a comparison operator, such as $>$, $\geq$, $<$, $\leq$, $=$, $\neq$; and val_i is a value of a statistical parameter. This template matches the rules, parameters of which satisfy all the specified restrictions $par_i\ op_i\ val_i$. Examples of such restrictions are $conf < 80\%$ and $supp \geq 35\%$.

  The computational complexity of this filter is linear in the number of rules since each rule requires a constant time to check if it satisfies the constraint.

- *The factual information about a user for whom the rule was discovered.* As mentioned in Section 5.2.1, we assume that the factual information of each user can be stored as a record in a relational table. Our template specification language allows one to formulate a restriction on the factual information of users for the purpose of obtaining only the rules that belong to this "restricted" set of users. Formally, such a template is specified as follows:

FACTS { `restriction` }

This type of filter works in two steps. First, the following SQL statement that returns a set of "qualifying" users (i.e., users that satisfy the restriction) is generated and executed:[1]

SELECT *UserId* **FROM** *FactualData* **WHERE** `restriction`

And, second, the rule set is filtered to include *only* the rules of the users returned by the SQL query described above.

In our template-based filtering tool, each of the above templates can be used individually or several templates can be combined into one using boolean operations AND, OR, and NOT.

The proposed language is related to the data mining query languages, such as the ones described in [46, 70, 41, 54, 77, 57, 43], among them M-SQL [43] and the template language of [46] being the closest to our proposal. For this application, we enhanced our set validation language SVL so that it can validate not only the association rules that contain items, but also the ones containing attribute-value pairs. We have also included features arising from idiosyncrasies of personalization

[1]Therefore, the syntax of the `restriction` element in the FACTS filter allows any expression that is acceptable in the WHERE clause of an SQL SELECT statement.

applications, such as the existence of *individual* data mining rules and of factual information about the users. Furthermore, since the subset and superset predicates were heavily used in this application, we made the syntax of subset and superset macros more intuitive. Some examples of filtering operators are provided below.

1. Accept all the rules that refer to Grand Union stores:

   $\textbf{ACCEPT}: \quad \textbf{RULE} \supset \{ \textit{Store} = \textit{GrandUnion} \}$

2. Reject all the rules that have attribute *Product* in the body (possibly among other attributes) and the head of the rule has either *DayOfWeek* or $\textit{Quantity} = \textit{Big}$ in it:

   $\textbf{REJECT}: \quad \textbf{BODY} \supseteq \{ \textit{Product} \}$
   $\textbf{AND} \ \ \textbf{HEAD} \subset \{ \textit{DayOfWeek}, \ \textit{Quantity} = \textit{Big} \}$

3. Accept all the rules that involve any combination of attributes *DayOfWeek* (only when value is *Mon* or *Wed*), *TimeOfDay*, and *Product*, in the body of the rule, that also have confidence greater than 65%:

   $\textbf{ACCEPT}: \quad \textbf{BODY} \subseteq \{ \textit{DayOfWeek} = \{ \textit{Mon}, \ \textit{Wed} \}, \ \textit{TimeOfDay},$
   $\textit{Product} \} \ \ \textbf{AND} \ \ \textbf{STATS} \ \{ \textbf{conf} > 65\% \}$

4. Reject all the rules that have the attribute *Product* present in their bodies and, possibly, *DayOfWeek* or *TimeOfDay* (but no other attributes besides these):

   $\textbf{REJECT}: \quad \textbf{BODY} \supseteq \{ \textit{Product} \}$
   $\textbf{AND} \ \ \textbf{BODY} \subseteq \{ \textit{DayOfWeek}, \ \textit{TimeOfDay}, \ \textit{Product} \}$

5. Reject all the rules that refer to the purchase of a luxury car for the low-income

users:

$$\begin{aligned} \textbf{REJECT}: \quad & \textbf{RULE} \supseteq \{ \textit{Product} = \textit{LuxuryCar} \} \\ & \textbf{AND FACTS} \{ \textit{YearlyIncome} = \textit{Low} \} \end{aligned}$$

6. The filtering tool can take advantage of an attribute hierarchy that was described in Section 3.2 and was used in the similarity-based grouping tool. That is, aggregated attributes and aggregated values can also be used in a template. For example, if we would like to accept all the rules that involve any type of discount in the body and specify any spring month in the head (based on the attribute hierarchy from Figure 3.3), we would use the following template:

$$\begin{aligned} \textbf{ACCEPT}: \quad & \textbf{BODY} \supseteq \{ \textit{DiscountType} \} \\ & \textbf{AND HEAD} = \{ \textit{Month} = \textit{spring} \} \end{aligned}$$

As we have shown above, the template-based filtering tool is computationally inexpensive. Therefore, as with the similarity-based rule grouping tool, this tool also scales well for very large numbers of rules.

5.4.2 Interestingness-based rule filtering

As described above, our proposed template-based rule filtering tool allows the domain expert to accept or to reject the discovered rules based on their structure, statistical parameters, and factual characteristics of the users. In addition to this, we propose using a filtering tool that selects only the most "interesting" rules according to some interestingness criteria.

There has been much research done in recent years quantifying "interestingness" of a rule, and several metrics have been proposed and used as a result of this work.

Among "objective" metrics, besides confidence and support [8], there are gain [36], variance and chi-squared value [59], gini [58], strength [27], conviction [20], sc- and pc-optimality [13], etc. "Subjective" metrics include unexpectedness [73, 53, 79, 60] and actionability [66, 73, 1].

Any of these metrics can be used as a part of the interestingness-based filtering tool, and the validation system can support different interestingness criteria. Moreover, the domain expert can specify interestingness-based filters using a syntax similar to the syntax of the template-based filters. For example, the filter

$$\textbf{ACCEPT}: \quad \textbf{INTERESTINGNESS} \; \{\, \textbf{gain} > 0.5, \; \textbf{unexpected} \,\}$$

specifies that all the high-gain and unexpected rules should be accepted. Moreover, the uniform syntax for both template-based and interestingness-based filter specifications allows to combine filters of both types into one. For example, the following template accepts all actionable rules that mention the purchase of a luxury car in the body of the rule:

$$\begin{array}{ll} \textbf{ACCEPT}: & \textbf{BODY} \supseteq \{\, \mathit{Product} = \mathit{LuxuryCar} \,\} \\ & \textbf{AND INTERESTINGNESS} \; \{\, \textbf{actionable} \,\} \end{array}$$

We would like to point out that such interestingness-based filters can be added to the profile validation system as external modules, thus making the system more versatile. The efficiency of such interestingness-based filters depends on their inherent complexity (i.e., some interestingness measures are inherently more complex to calculate than others) and their particular implementation.

Redundant rule elimination. One class of non-interesting rules are *redundant* rules. For example, consider the association rule "$\mathit{Product} = \mathit{AppleJuice} \Rightarrow$

Store = Grand Union (supp=2%, conf=100%)" that was discovered for customer ALW392. This rule appears to capture a specific aspect of the customer behavior: customer ALW392 buys apple juice *only* at Grand Union, and we may add it to his behavioral profile. However, assume, that is was also determined from the data that this customer does *all* of his shopping at Grand Union. Then the above mentioned rule constitutes a special case of this finding.

The redundant rule elimination filter finds all the redundant rules and removes them from the user profiles. In other words, this tool eliminates the rules that, by themselves, do not carry any new information about the behavior of a user. One particular case of redundancy occurs when the consequent Y of a high-confidence rule $X \Rightarrow Y$ has a high support. For instance, following the previous example, the rule "*Product = AppleJuice* $\Rightarrow$ *Store = GrandUnion (supp=2%, conf=100%)"* would be removed from the profile of user ALW392 and only the fact *"Store = GrandUnion (supp=100%)"* (i.e., this customer shops only at Grand Union) will be kept.

The computational complexity of such redundant rule elimination filter is linear in the number of rules to be filtered, because for each rule we only have to check whether its consequent has a very high support measure. This check can be done in constant time using a lookup table that holds a most frequent value of each attribute (along with its actual frequency). There is no extra work needed to create such table, since it can be obtained as a by-product of a rule discovery algorithm (e.g., Apriori) from the set of frequent 1-itemsets.

We implemented the redundant rule elimination tool described above as a part of the validation system. However, we would like to point out that this is just one of many possible redundant rule elimination approaches. Other approaches, e.g., based on ideas presented in [7, 14, 13, 55], can also be used in the rule validation process.

5.4.3 Other Validation Tools

Although rule grouping and filtering proved to be the most useful and frequently used validation tool as is demonstrated in Section 5.8, they can be complemented with various other validation tools. We briefly describe some of these tools below.

- **Visualization Tools**. Allow the expert to view the set of unvalidated rules or various parts of this set in different visual representations (histograms, pie charts, etc.) and can give the expert insights into what rules are acceptable and can be included in profiles.

- **Statistical Analysis Tools**. Statistical analysis tools can compute various statistical characteristics (value frequencies, attribute correlation, etc.) of unvalidated rules. This allows the expert to have many different "views" of these rules, therefore helping him or her during the rule validation process.

- **Browsing Tools**. As mentioned above, visualization and statistical analysis tools allow the expert to have "aggregated" views of the unvalidated rules through various visual representations and statistical characteristics. Browsing tools, on the other hand, can help the expert to inspect individual rules directly.

  Browsing tools are especially useful when combined with the similarity-based grouping method described in Section 3.2. Instead of browsing through individual rules and manually validating (accepting or rejecting) them on the one-by-one basis, the expert can apply the grouping algorithm and then browse the resulting groups (aggregated rules) and manually validate the selected groups.

Browsing tools can have some additional capabilities, such as being able to sort the content to be browsed in various ways. For example, it might be helpful for the expert to be able to sort rules by the user ID or by some interestingness measure, sort groups by their size, etc.

5.5 Incremental Profiling

In most e-commerce applications user transactional histories usually change over time since users continue their browsing and purchasing activities. Therefore, user behavioral profiles usually change over time, and there is a need to keep these profiles current by removing behavioral rules that are no longer valid and adding new rules that characterize user's emerging behaviors.

A straightforward approach to maintaining user profiles would be to rebuild them periodically "from scratch." However, this is, clearly, a very computationally intensive and time consuming process, especially since profiles often do not change significantly with new data.

An alternative approach would be to develop efficient *incremental* profile construction techniques that would adjust user profiles based on the new data without rebuilding them from scratch. One way to accomplish this would be to keep track of the sequence of all the validation operators $< o_1, \ldots, o_k >$ that were performed during the initial profile validation process. Then, when new incremental data ΔD is added to the initial dataset D, the previously used data mining algorithm can be applied to the dataset $D \cup \Delta D$ to discover all the new rules R_{new}. After that, each of the previously used validation operators o_i can be applied to the set of rules R_{new} in the *same* sequence as they were applied during the initial validation process. We would like to point out that this technique provides for *automatic* incremental validation of user profiles without any additional participation of the domain expert (until he or she decides to revisit the sequence of validation decisions).

Moreover, this incremental validation method can be improved further by using one of the existing incremental rule discovery techniques [25, 33, 82] instead of using the "from-scratch" rule discovery method considered before. Data monitoring triggers, such as the ones proposed in [85, 1], can also be used for this purpose.

5.6 Case Study

We implemented the tools described in the previous section in the 11Pro system.[2] The 11Pro system takes as inputs the factual and transactional data stored in a database and generates a set of validated rules capturing personal behaviors of individual users following the approach illustrated in Figure 5.4. The 11Pro system can use any relational DBMS to store user data and various data mining tools for discovering rules describing personal behaviors of users. In addition, 11Pro can incorporate various other tools that can be useful in the rule validation process, such as visualization and statistical analysis tools as mentioned in Section 5.4.

[2]11Pro stands for One to One Profiling System.

The current implementation of 11Pro uses association rules to represent behaviors of individual users. Also, the current implementation of 11Pro supports similarity-based grouping, template-based filtering, redundant rule elimination, and browsing tools.

We tested 11Pro on a "real-life" marketing application that analyzes the purchasing behavior of customers. The application included data on 1903 households that purchased different types of beverages over a period of one year. The data set contained 21 fields characterizing purchasing transactions, including the information about the time of purchase, product purchased, amount spent, coupons used, and related advertisements seen. The whole data set contained 353,421 records (on average 186 records per household). The data mining module of 11Pro executed a rule discovery algorithm on the individual household data for *each* of the 1903 households and generated 1,022,812 association rules in total, on average about 537 rules per household. Minimal values for the rule support and confidence were set at 20% and 50%, respectively.

It is interesting to observe that the majority of discovered rules pertain to a very small number of households. For example, nearly 40% of all the discovered rules (407,716 in total) pertain only to five or fewer households (out of the total of 1903), and half of those rules (196,384 in total) apply only to a single household. This demonstrates that many discovered rules capture truly idiosyncratic behavior of individual households. Since the traditional segmentation-based approaches to building customer profiles, such as the ones described in Section 5.1, do not capture idiosyncratic behavior, this means that these methods would not be able to identify many of the rules discovered in this application. On the other extreme,

Validation	Number of rules:		
tool	accepted	rejected	unvalidated
1. Redund. elimination	0	186,727	836,085
2. Filtering	0	290,427	545,658
3. Filtering	0	268,157	277,501
4. Filtering	6,711	0	270,790
5. Filtering	0	233,013	37,777
6. Grouping (1,046 gr.)	16,047	1,944	19,786
7. Grouping (6,425 gr.)	4,120	863	14,803
Final:	26,878	981,131	14,803

Figure 5.6: Example of a validation process for a marketing application: promotion sensitivity analysis.

several discovered rules were applicable to a significant portion of the households. For example, nine rules were pertained to more than 800 households. In particular, the rule *"DayOfWeek=Monday ⇒ Shopper=Female"* was applicable to 859 households.

Three case studies of user profile validation were performed for this application. In the first case study, we performed promotion sensitivity analysis, i.e., analysis of customer responses to various types of promotions, including advertisements, coupons, and various types of discounts. As a part of this application, we wanted to construct customer profiles that reflect different types of individual customer behaviors related to promotional activities. Since we are very familiar with this application, we assumed the role of the domain experts. In the second case study, we performed seasonality analysis, i.e., we constructed customer profiles that contain individual rules describing seasonality-related behaviors of customers, such as the types of products that a customer buys under specific temporal circumstances (e.g., only in winter, only on weekends) and the temporal circumstances under which a customer purchases specific products. In the third case study, we asked a marketing expert to perform the seasonality analysis from her point of view. To illustrate the validation process, we describe the first case study in detail below. We also report the results from the other two case studies in this section.

5.6.1 Analysis of Sensitivity to Promotions

As mentioned above, we performed the role of experts in the promotion sensitivity analysis and validated the 1,022,812 discovered rules ourselves using the sequence of validation operators presented in Figure 5.6. As shown in Figure 5.6, we first applied the redundant rule elimination tool that examined the heads of all the rules and removed those rules whose heads by themselves are "implied" by the data in the sense explained in Section 5.4.2. It turned out that this operator rejected about 18% from the set of all the discovered rules, namely 186,727. Then we performed the filtering operation (operation 2 in Figure 5.6) that rejects all the rules with household demographics-related information in their heads. As a result of this filtering operation, the number of unvalidated rules was reduced from 836,085 to 545,658. After that, we performed several additional filtering operations (3, 4 and 5 in Figure 5.6). One of them (3) rejected rules where either body or head contains only the market research company-specific attributes without any other information. Another filtering operation (4) accepted rules that state direct relationship between kinds of products purchased and various promotions, i.e., rules that have product information (possibly among other attributes) in the body and promotion-related information (discount, sale, coupon used, or advertisement seen) in the head. Another filtering operation (5) rejected all the rules that do not have any promotion-related information in the body as well as in the head of the rule. By performing all these filtering operations, we reduced the number of unvalidated rules to 37,777. Then we applied two grouping operations, using the attribute hierarchy, a fragment of which is presented in Figure 5.7. First, we applied the grouping tool using the cut presented in Figure 5.7(a) to get fewer, but more aggregated (therefore, less descriptive) groups (operator 6 in Figure 5.6). This operation grouped the remaining 37,777 unvalidated rules into 1,046 groups, where the biggest group contained 2,364 rules and the smallest group had just 1 rule in it. We inspected the 50 biggest groups and were able to validate 38 of them (31 accepted and 7 rejected), which brought the unvalidated rule count down to 19,786. We were unable to decide on whether to accept or reject the remaining 12 groups (out of 50) and left them as "undecided" for further analysis. Finally, we applied another grouping operation (operation 7) to the remaining unvalidated rules using the cut presented in Figure 5.7(b). We obtained 6,425 groups. The biggest group had 237 rules but about 80% of groups contained 5 rules or less. Again, we inspected 50 biggest groups and validated 47 of them (34 accepted and 13 rejected). As the result, we validated 4,983 more rules.

We stopped the validation process at this point because there were no large groups that we could validate as a whole and it started taking us more and more time to validate smaller and less "understandable" groups. The whole valida-

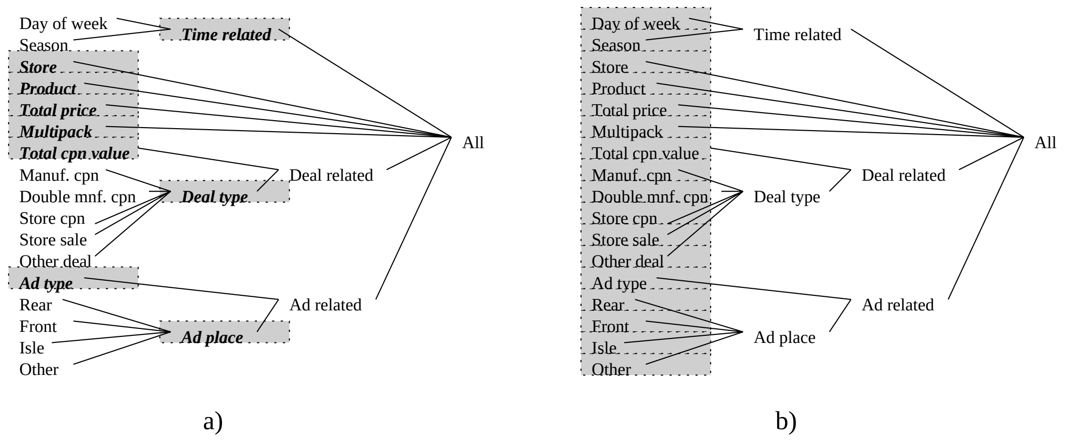


Figure 5.7: Fragment of an attribute hierarchy used in a marketing application.

tion process, including expert and computing time, took about 1.5 hours,[3] during which we validated 98.5% of the initially discovered rules (only 14,803 rules out of 1,022,812 remained unvalidated). The total number of accepted and rejected rules constituted 2.6% and 95.9% respectively of the initially discovered rules. The total number of rules accepted and put into profiles was 26,878 (on average, about 14 rules per household profile).

We performed the validation process described above on all the 1,022,812 rules generated by the rule discovery algorithm. Alternatively, we could have specified constraints, for example, using the methods proposed by [77] or [14], on the types of rules that we are interested in prior to the data mining stage. As a result of this, fewer data mining rules would have been generated, and there would have been

[3]This time includes several minutes of computing time and the remainder constitutes the time for the expert to browse through the rules, think, and decide on the validation operations to be performed. This time does not include rule discovery and the construction of attribute hierarchies.

no need to apply some of the elimination filters described in this case study. For example, we could have specified the constraints corresponding to the validation operations (1) and (2) in Figure 5.6 before applying a rule discovery algorithm. As a result, we would have generated only 545,658 rules, all of them satisfying these two conditions, and there would have been no need to apply validation operations (1) and (2) in the post-analysis stage. Although very useful, the constraint specification approach cannot replace rule validation in the post-analysis stage of the knowledge discovery process. We will elaborate on this further in Section 5.7.

5.6.2 Seasonality Analysis

In addition to the analysis of customer responses to promotions described in detail above, we used the same set of discovered rules to perform another related market research task – seasonality analysis. In particular, in the second case study, we constructed customer profiles that contain individual rules describing seasonality-related behaviors of customers, such as the types of products that a customer buys under specific temporal circumstances. It took us about one hour to perform this task. As the result, we validated 97.2% of the 1,022,812 discovered rules, where 40,650 rules were accepted and 953,506 rules were rejected. The summary of the validation process is presented in Figure 5.8.

More specifically, we started the validation process by eliminating redundant rules and followed it with the application of several filtering operators, most of which were rule elimination filters. Elimination of redundant rules and repeated application of rule filters helped us validate 93.4% of the rules.

After we managed to validate a large number of rules with relatively few filtering

Validation	Number of rules:		
tool	accepted	rejected	unvalidated
1. Redund. elimination	0	186,727	836,085
2. Filtering	0	285,528	550,557
3. Filtering	0	424,214	126,343
4. Filtering	0	48,682	77,661
5. Filtering	10,052	0	67,609
6. Grouping (652 gr.)	23,417	6,822	37,370
7. Grouping (4,765 gr.)	7,181	1,533	28,656
Final:	40,650	953,506	28,656

Figure 5.8: Example of a validation process for a marketing application: seasonality analysis.

operations, we next decided to switch to a more "subtle" validation process and apply the grouping tool to the remaining unvalidated rules. The grouping algorithm generated 652 groups. After that, we examined several of the largest groups and validated the rules belonging to them. As a result, we managed to validate all but 37,370 rules. We then again applied the grouping tool to these remaining rules, this time using a different "cut," and validated another set of 8,714 rules. At this point, we encountered the law of diminishing returns (each subsequent application of a validation operator managed to validate smaller and smaller number of rules), and we stopped the validation process. It took us one hour to perform the whole validation process (including the running time of the software and the expert validation time). As a result, we validated 97.2% of all the rules, among which 4.0% were accepted and 93.2% rejected.

As a result of the validation process, we reduced the average size of the customer profile from 537 unvalidated rules to 21 accepted rules. An example of an accepted rule for one of the households is: *"Product=FrozenYogurt* & *Season=Winter* $\Rightarrow$ *CouponUsed=YES"*, i.e., during the winter, this household buys frozen yogurt mostly using coupons. This rule was accepted because it reflects an interesting seasonal coupon-usage pattern. An example of a rejected rule for one of the households is: *"Product=Beer* $\Rightarrow$ *Shopper=Male"*, i.e., the predominant buyer of beer is male in this household. This rule was rejected because of its obviousness.

5.6.3 Seasonality Analysis: Marketing Analyst

For the third case study, we asked a marketing analyst to perform seasonality analysis with 11Pro. She started the analysis with applying redundant rule elimination and several template-based filtering rejection operations to the rules (e.g., reject all the rules that are not referring to the *Season* or the *DayOfWeek* attributes). After that, she grouped the remaining unvalidated rules, examined several resulting groups, and then stopped the validation process. At that point, she felt that there is nothing more to reject and decided to accept all the remaining unvalidated rules.[4] As a result, she accepted 42,496 rules (4.2% of all the discovered rules) and spent about 40 minutes on the whole validation process.

The results of all the three case studies are summarized in Figure 5.9.

We received the following feedback from the marketing expert at the end of the validation process. First, she liked the flexibility of 11Pro and the ability to apply a variety of validation tools in the analysis. In particular, she liked our grouping and filtering tools, but felt that we should provide better ways for presenting results, including certain visualization capabilities. Second, we observed that her validation "style" was to keep rejecting groups of irrelevant rules and accept all the remaining rules when there was nothing left to reject further. Such style can be explained by the fact that the expert was only marginally familiar with 11Pro and did not utilize fully its capabilities to reject and accept groups of rules in an interleaving manner. Third, we discussed the issue of the "quality" of the validated rules. The marketing expert felt that the rule evaluation process is inherently subjective because different marketing experts have different opinions, experiences, understanding the specifics of the application, etc. Therefore, she believed that different marketing experts would arrive at different evaluation results using the validation process described

[4]Although she accepted all the remaining rules, we personally felt that if she continued the validation process she could have found some more "bad" rules.

Number of rules	Case Study I		Case Study II		Case Study III	
Rejected	981,131	(95.9%)	953,506	(93.2%)	980,316	(95.8%)
Accepted	26,878	(2.6%)	40,650	(4.0%)	42,496	(4.2%)
Unvalidated	14,803	(1.5%)	28,656	(2.8%)	0	(0.0%)

Figure 5.9: Summary of case studies.

in this paper because of the various biases that they have.

We would like to point out that the accepted rules, although valid and relevant to the expert, may not be *effective* in the sense that they may not guarantee actionable results, such as decisions, recommendations, and other user-related actions. Therefore, to address the effectiveness issue, we are currently working on connecting 11Pro to a recommendation system to produce recommendations to the customer. We believe that the resulting system will provide more effective recommendations to customers than a stand-alone recommendation system. Our proposed framework on how to improve traditional recommender systems is outlined in [4].

5.7 Discussion

The experiments performed on a medium-size problem (1903 households, 21 fields, and 1,022,812 discovered rules) reported in the previous section produced encouraging results: based on the first case study, we managed to validate 98.5% of 1,022,812 rules in only 1.5 hours of inspection time. The results of this and other case studies produce several important observations and raise several questions.

"Quality" of generated rules. One of the central questions is how "good" the profiles are that were generated by the domain expert. In other words, would it be possible for the domain expert to discard "good" and retain "bad" rules in the user profiles during the validation process. As was pointed out earlier, the terms "good" and "bad" can take different meanings, such as statistical validity, acceptability by an expert, and effectiveness. Generating statistically valid rules is the prerogative of data mining algorithms and objective interestingness metrics that can be applied to the discovered rules in the post-analysis stage. We considered the problem of validating the rules by an expert in our research. As was pointed out in the previous section, there is no single objectively "correct" set of validated rules that the expert should be able to discover because different experts have different evaluation biases. One possible approach lies in assigning a certain metric to the rules and then measuring the quality of validated rules according to this metric. For example, in the context of recommender systems, one can measure the quality of discovered and validated rules in terms of the quality of recommendations that these rules generate.[5] However, this approach deals with the rule effectiveness issues. As pointed out in Chapter 1, the problem of generating effective rules has not been addressed in this paper and is left as a topic of future research.

Scalability. Our experimental results indicate that 11Pro can handle medium-size problems well. An interesting question is how well our approach would scale up to large problems having millions of users and dozens of attributes. If the

[5]In fact, we have started looking into this issue in [84] and are planning to conduct this research by using recommender systems and judging the quality of profiles via the quality of resulting recommendations.

number of attributes increases, then the rule mining methods, such as Apriori, will generate exponentially larger number of rules and would constitute a bottleneck of the profile generating process (rather than the rule validation phase). If the number of attributes is fixed and the number of users grows, then an application of validation operators should scale up linearly with the total number of users. This is the case, because, our proposed validation operators run in time linear in the total size of the rules, and we observed that the number of discovered rules grows linearly with the number of users.[6]

Constraint-based rule generation vs. post-analysis. In our experiments we applied a rule discovery algorithm to generate *all* the association rules for pre-specified confidence and support levels and then applied several filtering operators to remove "uninteresting" rules from this set (e.g., as shown in Figure 5.6). Alternatively, we could have applied a constraint-based version of association rule discovery methods, such as the ones presented in [77, 14]. As a result, we could have obtained the number of rules smaller than 1,022,812 produced by the unconstrained rule discovery algorithm.

Although the constraint-based approach reported in [77, 14] provides a partial solution to the validation problem by reducing the total number of rules generated during the initial data mining stage, it does not provide the complete solution for the following reason. It is very hard to figure out *all* the relevant constraints *before* the data mining algorithms are launched. The human expert, most likely, will be able to come up with many important filters only after inspecting data mining results using browsing, grouping, or visualization tools. Alternatively, an expert

[6]Although we have not conducted rigorous experiments to prove this point.

can make a mistake and specify a filter that happens to be too strict (i.e., rejects too many rules). If such constraint was specified before mining, the whole rule discovery algorithm would have to be reexecuted with the correct constraint, which is more computationally expensive than to reexecute a correct filtering operator in the post-analysis phase. The benefits of iterative analysis of data mining results are also pointed out by several researchers, including [32, 72, 67, 50, 69].

Therefore, neither post-analysis nor the pre-specification of constraints works best as a stand-alone method, and the two approaches should be combined into one integral method. The main question pertaining to this combination is what kinds of constraints should be pre-specified by the user for the rule generation phase and what functionality should be left for the post-analysis phase. This topic was addressed by several researchers within the rule discovery context [67, 38]. We are currently working on extending this line of work to the personalization problem.

Examination of groups of rules. One of the main features of our approach is the ability for the domain expert to examine groups of rules and to decide whether to accept or reject a group as a whole. One of the concerns for such method is that the domain expert can make mistakes by accepting "bad" and rejecting "good" rules. This issue is addressed in 11Pro by providing the capability for the domain expert to evaluate a group of rules *recursively* in case the expert is unable to decide whether or not to accept or reject this group as a whole. In other words, the expert can apply validation operators just to this particular group of rules and examine its subgroups. By examining smaller subgroups, the expert can then make more reliable decisions.

Future research. This paper opens several directions for future work. One of such directions includes studies of measures of effectiveness of discovered rules and development of efficient algorithms for discovering such rules. Moreover, the marketing expert pointed to us that some additional validation tools could be added to our system, and we plan to work on this issue. Finally, we plan to study tradeoffs between constraint-based generation and post-analysis of rules in the context of personalization applications.

5.8 11Pro System

We implemented the methods presented in this chapter as a part of the 11Pro system.The 11Pro system takes as inputs the factual and transactional data stored in a database or flat files and generates a set of validated rules capturing personal behaviors of individual customers as illustrated in Figures 5.3 and 5.4. The 11Pro system can use any relational DBMS to store customer data and various data mining tools for discovering rules describing personal behaviors of customers. In addition, 11Pro can also incorporate various other tools that can be useful in the rule validation process, such as visualization and statistical analysis tools.

The architecture of the 11Pro system follows the client-server model and is presented in Figure 5.10. The server component of the 11Pro system consists of the following modules:

1. *Coordination module.* It is responsible for the coordination of the profile construction process, presented in Figures 5.3 and 5.4, including the process of generating behavioral rules of the customers from the data stored in the database and the subsequent validation process.

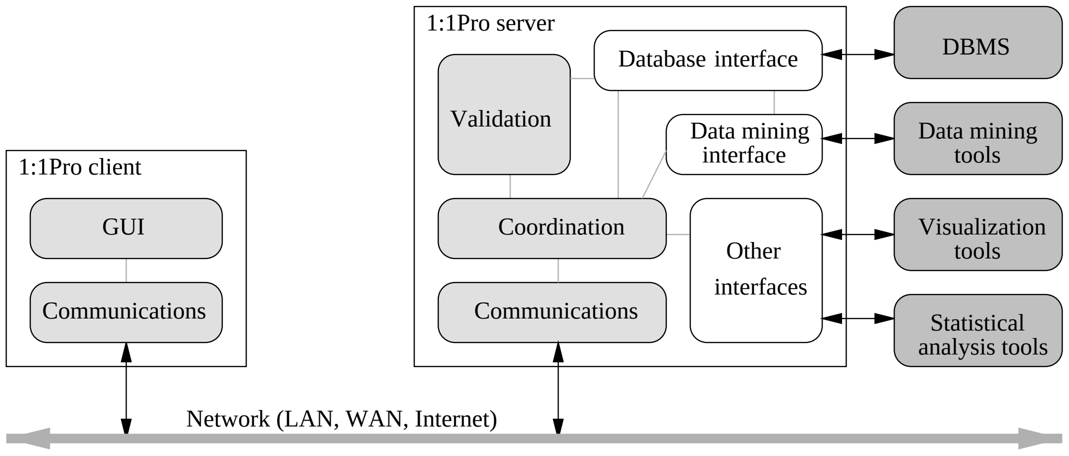


Figure 5.10: Architecture of the 11Pro system.

2. *Validation module.* It is responsible for validating the rules discovered by data mining tools according to the methods presented in Section 5.4. The current implementation of the 11Pro system supports similarity-based grouping, template-based filtering, redundant rule elimination, and browsing tools.

3. *Communications module.* It is responsible for all the communications with the 11Pro client.

4. *Interfaces to the external modules.* As stated above, 11Pro can use various external modules, such as a DBMS, a data mining and visualization tools. Each of these modules requires a separate interface, and the 11Pro server provides such interfaces as shown in Figure 5.10.

The client component of 11Pro contains the graphical user interface (GUI) and the communications modules. The GUI module allows the expert to specify various validation operations described in Section 5.4 and view the results of the iterative validation process. In particular, the communication module takes the validation request specified by the expert and sends it to the server. The server receives validation operators and passes them through the coordination module to the validation component of 11Pro for subsequent processing. Since some of the validation tools, such as the statistical analysis tools, generate outputs, these outputs are sent from the validation module of the server to the GUI module of the client via the communication modules of the server and the client.

One of the design objectives of 11Pro is to make it an open system so that it would be easy to incorporate a broad range of various data sources, data mining, visualization, and statistical analysis tools into it. Therefore, the database interface is designed to support a broad range of various types of data sources, including relational databases (such as Oracle and SQL Server), flat files, Web logs, etc.

We used association rule discovery methods to build customer profiles in the current implementation of 11Pro. However, our methods are not restricted to a particular structure of the rules nor to any particular algorithm that discovers these rules. Other commercial and experimental data mining tools can be easily plugged into 11Pro, that generate behavioral profiles based on methods other than association rules, such as decision tree methods [19]. The main difficulty in this process is the need to convert the rule representation formats used by these tools into our internal rule formats. This is accomplished by developing rule converters for each of the external data mining tools interfaced with 11Pro.

The server component of the 11Pro system is running on Linux/Unix platforms and is implemented in C and Perl. The client component of the 11Pro system was implemented in Java and can run either on the same or on a different machine as the server component. It can also run from a Web browser or as a stand-alone application.

Chapter 6

Handling Very Large Numbers of Association Rules in the Analysis of Microarray Data

6.1 Motivations and Related Work

One of the major goals of molecular biology is to study how different genes regulate each other, and a major research effort is targeted towards understanding and discovery of gene regulation patterns. Over the past 10 years, the microarray method [65] became very popular for these discovery purposes. This method allows biologists to monitor behavior of large number of genes on a single microarray chip (e.g., up to 40,000 genes [17]) in order to profile gene expressions under various environmental conditions, therapeutic agents, types of tissues, and other factors affecting gene expressions. Typically, biologists would generate a sample of tests using the microarray method, where a single test would contain information on the genes that are being profiled, and the sample size would range from a few dozens to a few hundred or even thousands of tests. Then biologists need to analyze these samples to discover gene regulation patterns. For example, one may discover that whenever gene X is upregulated and gene Y is downregulated, then gene Z is usually also upregulated.

Many bioinformatics and data mining researchers have been working on applying data mining methods to the analysis of microarray data. In particular, clustering methods, first proposed for this problem in [29], group genes into clusters that exhibit "similar" types of behavior in the experiments. Also, Tamayo et al [80] used Self-Organizing Maps to organize human genes into biologically relevant clusters with respect to the problem of hematopoietic differentiation. The clustering methods allow biologists to design experiments helping them to understand further the relationships among the genes. However, they do not provide deep insights into specific relationships among the genes and understand underlying biological processes in the cell.

In order to identify previously unknown functions of the genes, Brown et al [21] proposed to use Support Vector Machines (SVMs) that build classifiers predicting gene functions. One of the distinctive features of SVM is that they learn classifiers using both positive and negative examples. In cases when it is difficult to collect negative examples, Pavlidis et al [62] proposed a new probabilistic method using the techniques borrowed from biological sequence analysis. In addition, Kurra et al [48] described a classification method to discriminate two types of leukemia using heuristic feature selection and a certain variant of perceptron-based classification method that separates these two classes of leukemia. Also, Friedman et al [35] used Bayesian networks to analyze gene expression data and to discover and describe interactions between genes. Furthermore, Barash and Friedman [11] used context-specific Bayesian clustering method to help in understanding the connections between transcription factors and functional classes of genes. Moreover, Bicciato et al [16] presented a neural network-based method to find relationships among different genes involved in major metabolic pathways and to relate specific genes to different classes of leukemia.

Since gene expression data is often represented in the binary or ternary form (i.e., gene expressions are up- or down-regulated [52], or unchanged, as will be discussed in the next section) and since biologists often look for interactions among various genes, it is natural to use association rules [8] to represent interactions among multiple genes. Besides providing the relationship between gene expression profiles, association rules also provide the direction of the relationship unlike other techniques, such as clustering and classification. Some initial work on using association rules to capture these interactions has been reported in [15] and [47]. While Berrar et al [15] used the standard association rule discovery algorithm (Apriori [9]), Kotala et al [47] introduced a new method for discovering associations in the microarray data that is based on Peano count trees (P-Trees).

However, association rule methods face certain challenges when applied to microarray data that have not been addressed in [15] and [47]. Since microarray data has a very large number of variables (i.e., genes are often measured in thousands), association rule discovery methods tend to generate very large numbers of rules (often measured in hundreds of millions or even billions). Although some of these rules contain very important biological relationships among different genes, other rules often represent spurious, irrelevant and already known relationships. Therefore, the challenge is to be able to explore and analyze very large numbers of biological relationships (rules) in an effective manner and to separate "good" relationships from the "bad." One way to deal with the combinatorial explosion problem is to specify tight constraints on the types of association rules that are of interest to the biologists using constraint-based mining methods such as the ones proposed in [14, 77], including the selection of only particular gene types to be used as inputs for the association rule generation algorithms. For example, Berrar et al [15] followed this strategy and restricted the set of genes used for the generation of association rules from 1500 to only 20 and the number of drugs from 1500 to only 20. However, restricting the problem to only 20 genes and 20 drugs created a potentially very limiting situation of leaving out important variables and, therefore, missing important relationships among the genes because some of the genes involved in these relationships might be omitted.

Our research addresses the challenge of analyzing very large numbers of discovered rules in the *post-processing* stage of the data mining process by providing the biologist with a set of tools necessary for the analysis of very large numbers of association rules generated by standard association rule generation algorithms (such as Apriori [9]) on the microarray data. These tools leverage the knowledge about genomics that biologists have and let biologists play a central role in the discovery process through close interactions between the tools and the biologist. By utilizing these tools in an interactive and exploratory manner, the biologist can identify and focus on really important rules and separate them from the irrelevant, trivial or already known relationships among the genes.

In the rest of this chapter is we formulate the gene regulation pattern discovery and post-analysis problem. We also present rule exploration methods to address this problem. Finally, we present a case study that tested our methods on some microarray data.

6.2 Problem Formulation

One way to understand how different genes regulate each other is to measure *gene expression* levels [52] produced by a cell using *microarray* technologies [65]. Typically, biologists conduct a number of experiments measuring gene expression levels of a cell or a group of cells under various conditions affecting these expression levels. Biologists are interested in how different gene expressions change depending on the type of a tissue, age of the organism, therapeutic agents, and environmental conditions. From the computational point of view, the expression level is represented as a real number. Therefore, the result of a single experiment is an array of N real numbers. Moreover, the set of genes used in microarrays is determined by a biologist and remains the same for all the experiments. Therefore M microarray experiments result in an $M \times N$ matrix $\{x_{ij}\}_{i=1,M,\ j=1,N}$, where x_{ij} represents the expression level of gene j in experiment i. Depending on a particular biological application, the number of experiments M can vary from a few dozens to several hundreds or even thousands, and the number of genes N can vary from several hundred to tens of thousands (e.g., [17] reports putting 40,000 genes on a single microarray chip). Moreover, the biologists are more interested in how gene expressions vary in these experiments relative to normal expression levels in an organism, rather than in their absolute values. Therefore, biologists typically normalize expression levels x_{ij} according to certain normalization criteria and the results are discretized according to whether they exceed certain predetermined thresholds or not. This process results in a discretized $M \times N$ matrix $\{r_{ij}\}_{i=1,M,\ j=1,N}$ where the values r_{ij} are of three different levels: unchanged (denoted as #), upregulated [52] (denoted as $\uparrow$), and downregulated [52] (denoted as $\downarrow$). Certainly, one may choose more than 3 levels of gene expression level discretization. However, in many applications biologists assume these three levels of discretization, and we will follow this assumption in the paper.

Given normalized and discretized gene expressions $\{r_{ij}\}_{i=1,M,\ j=1,N}$, we can discover gene regulation relationships in the form of association rules. In particular, an experiment i can be represented as a transaction [8] consisting of N items (genes) $\{r_{ij}\}_{j=1,N}$, each item r_{ij} taking one of the three values (#,$\uparrow$,$\downarrow$). Given this data, we want to find all the association rules with specified levels of confidence and support and let the biologists select only these rules that are interesting to them from a biological point of view. As mentioned earlier, one of the problems with this approach lies in large numbers of associations generated by rule discovery methods. Depending on the number of genes N in the gene expression matrix $M \times N$ and on the particular values of confidence and support, the number of discovered association rules can be measured in billions.

As mentioned earlier, one can impose constraints on the types of generated association rules, as was done in [14, 77]. However, we either (1) still may end up with a very large number of discovered association rules, or (2) impose overly restrictive constraints on the data and many potentially important biological relationships may be missed. In Section 3 we present an approach that addresses this problem by providing a biologist with a set of tools for exploring very large numbers of discovered association rules in an iterative and systematic manner.

6.3 Analyzing Discovered Gene Regulation Patterns

In this section we adapt our rule post-processing methods allowing biologists to analyze very large numbers of regulation relationships among genes and select those that are of interest to them.

Post-processing of discovered gene regulation relationships is an iterative and interactive process in which the biologist selects specific rule exploration tools, such as rule filtering, rule grouping and other rule analysis tools and applies them to the set of discovered gene regulation relationships. As a result, the biologist can examine *groups* of relationships at a time, decide which groups are interesting and worth further exploration, which groups of rules are irrelevant, trivial, or already known, and discard these relationships from further consideration. By selecting these rule exploration tools in an interactive manner, the biologist can quickly focus on the rules that are of interest to him or her. We will further elaborate on this when we present our case study. In the rest of this section, we will describe specific rule exploration tools, such as filtering, grouping and others.

6.3.1 Rule Filtering

This tool allows the biologist to impose various constraints on the syntactic structure of the discovered biological rules using templates in order to reduce the number of rules to be explored. In other words, biologists have the ability to "focus" their analysis and exploration efforts only on the subset of rules that is of a specific interest to them. For this purpose, we have adapted our set validation language SVL to the domain of microarray data analysis. We will briefly present the "adapted" SVL language and show how it can be used by the biologists to specify rule templates.

Rule templates filter biological rules from the total set of discovered rules by specifying various restrictions on the combinations of genes and their expression levels that can appear in the body and the head of the rule. These templates can be specified using the following notation:

$$\textit{RulePart}\ \mathbf{HAS}\ \textit{Quantifier}\ \mathbf{OF}\ C_1, C_2, \ldots, C_N\ [\mathbf{ONLY}] \qquad (6.1)$$

Here *RulePart* can be BODY, HEAD, or RULE, and it specifies the part of the rule (antecedent, consequent, or the whole rule, respectively) on which the constraint is being placed. $C_1, C_2, \ldots, C_N$ is a *comparison* set, i.e., it represents a list of genes (possibly with their expression levels) against which the discovered rules will be compared. Each element C_i of this list can be one of the following:

- A gene, e.g., $G17$;
- A gene with a particular expression level, e.g., $G17\uparrow$;
- A gene with a list of allowable or unallowable expression levels, e.g., $G17 = \{\uparrow, \#\}$ or $G17 \neq \{\downarrow\}$;

- A group (or category) of genes[1], represented by the name of the gene category, e.g., $[DNA\_Repair]$. We use square brackets with category names to differentiate between the category names and gene names;
- A group of genes with an expression level, e.g., $[DNA\_Repair]\uparrow$;
- A group of genes with a list of allowable or unallowable expression levels, e.g., $[DNA\_Repair] = \{\uparrow, \#\}$ or $[DNA\_Repair] \neq \{\downarrow\}$.

Quantifier is a keyword or an expression that specifies how many genes specified by the $C_1, C_2, \ldots, C_N$ list have to be contained in *RulePart* in order to be accepted by this template. We use parentheses to enclose the quantifier expression. *Quantifier* can be one of the following:

- Either (ALL), (NOTZERO), or (NONE), specifying the number of genes (all of them, at least one of them, none of them, respectively) from $C_1, C_2, \ldots, C_N$ the *RulePart* must have;
- A numeric value; e.g., (2), which specifies that a rule must have *exactly* 2 genes from the comparison set;
- A range of numeric values; e.g., (1-3), which specifies that a rule must have 1, 2, or 3 genes from the comparison set;
- A list of numeric values and/or ranges; e.g., (1, 3, 5-7), which specifies that a rule must have either 1, 3, 5, 6, or 7 genes from the comparison set in order to

[1]Genes can often be categorized or grouped according to various (standardized) gene categories, e.g., by their function.

be accepted by this template; keywords ALL and NONE (but not NOTZERO) can also be present in this list, e.g., (NONE, 2, 3) or (1, 3-ALL).

Finally, an optional keyword ONLY can be used to indicate that $RulePart$ can have only the genes that are present in the $C_1, C_2, \ldots, C_N$ list.

Some examples on how to use templates to filter biological rules are provided below.

- All rules that contain at least one of the following genes: $G1$, $G5$, and $G7$:

  $$\text{RULE } \textbf{HAS} \text{ NOTZERO } \textbf{OF } G1, G5, G7$$

  For example, rule $G1 \uparrow \Rightarrow G3\#$ matches this template, and rule $G2 \uparrow \Rightarrow G3\#$ does not.

- All rules that contain some of the genes $G1$, $G5$, $G7$, but no other genes:

  $$\text{RULE } \textbf{HAS} \text{ NOTZERO } \textbf{OF } G1, G5, G7 \textbf{ ONLY}$$

  For example, rule $G1 \uparrow \Rightarrow G5 \uparrow$ matches this template, and rule $G1 \uparrow \Rightarrow G3\#$ does not.

- All rules that contain exactly one of the following genes: $G1$, $G5$, $G7$. Moreover, only rules with upregulated $G1$, downregulated $G5$, and upregulated or downregulated (but not unchanged) $G7$ are acceptable:

  $$\text{RULE } \textbf{HAS} \text{ 1 } \textbf{OF } G1 \uparrow, G5 \downarrow, G7 = \{\uparrow, \downarrow\}$$

  For example, rule $G1 \uparrow \Rightarrow G3\#$ matches this template, and rule $G1 \uparrow \Rightarrow G7 \uparrow$ does not.

- All rules that contain either both $G1$ and $G2$ in the body (antecedent) or none of them:

  $$\text{BODY } \textbf{HAS} \text{ NONE}, \text{ALL } \textbf{OF } G1, G2$$

  For example, rule $G5 \uparrow \Rightarrow G3\#$ matches this template, and rule $G1 \uparrow \Rightarrow G7 \uparrow$ does not.

- All rules that contain a DNA repair gene (i.e., a gene that belongs to the DNA repair group of genes) in the head (consequent):

  $$\text{HEAD } \textbf{HAS} \text{ NOTZERO } \textbf{OF } [\mathit{DNA\_Repair}]$$

- All rules that contain exactly 3 genes in the body. In addition, all of them must be upregulated:

  $$\text{BODY } \textbf{HAS} \text{ 3 } \textbf{OF } [\mathit{ALL\_GENES}] \uparrow$$

As mentioned before, in our template-based rule filtering tool each of the above templates can be used individually or several templates can be combined into one using Boolean operations *AND*, *OR*, and *NOT*. We now present some examples of composite templates that use these Boolean operations.

- All rules that have up to 3 DNA repair genes and no other genes in the body, as well as gene $G7$ in the head of the rule:

  $$\begin{aligned}\text{BODY } \textbf{HAS } 1-3 \textbf{ OF } [\mathit{DNA\_Repair}] \textbf{ ONLY AND}\\ \text{HEAD } \textbf{HAS} \text{ NOTZERO } \textbf{OF } G7\end{aligned}$$

- All rules that have exactly 2 genes in the body and at least one of them belongs either to DNA repair or to Transcription category:

$$\text{BODY } \mathbf{HAS} \text{ 2 } \mathbf{OF} \ [\mathit{ALL\_GENES}] \ \mathbf{AND}$$
$$\text{BODY } \mathbf{HAS} \text{ NOTZERO } \mathbf{OF} \ [\mathit{DNA\_Repair}], [\mathit{Transcription}]$$

- All rules that have no DNA repair genes, but 2 or more upregulated Transcription genes present:

$$\text{RULE } \mathbf{HAS} \text{ NONE } \mathbf{OF} \ [\mathit{DNA\_Repair}] \ \mathbf{AND}$$
$$\text{RULE } \mathbf{HAS} \ 2 - \text{ALL } \mathbf{OF} \ [\mathit{Transcription}] \uparrow$$

As with SVL, we use cardinality predicates as a basis for the biological rule templates. Therefore, we implemented them in a very efficient manner using lookup table-based data structures. Using our implementation, the filtering algorithm runs in the time linear in the total size of the rules to be filtered.

To make it more intuitive for biologists, we also provide a number of *predefined* templates (i.e., macros) that directly support specific questions that biologists may ask. For example, a biologist may want to explore whether one group of genes influences (implies) another group of genes and vice versa. We provide a macro template

$$\mathbf{POSSIBLE\_INFLUENCE}(\mathit{GeneSet1}, \mathit{GeneSet2})$$

that returns all the rules that contain some genes from GeneSet1 in the body (and possibly some other genes) and a gene from GeneSet2 in the head or vice versa, some genes from GeneSet2 in the body (and possibly some other genes) and a gene from GeneSet1 in the head. Using our template specification language, this macro template was defined as:

POSSIBLE_INFLUENCE(*GeneSet1*, *GeneSet2*) :=

BODY **HAS** NOTZERO **OF** *GeneSet1* **AND**

HEAD **HAS** NOTZERO **OF** *GeneSet2* **OR**

BODY **HAS** NOTZERO **OF** *GeneSet2* **AND**

HEAD **HAS** NOTZERO **OF** *GeneSet1*

Consider the following template:

POSSIBLE_INFLUENCE($\{G1, G2\}, \{G3, G4\}$)

Then, the rule $G1 \uparrow \wedge G7 \uparrow \Rightarrow G3 \uparrow$ matches this template, and the rule $G1 \uparrow \Rightarrow G2 \uparrow$ does not.

Another example of a macro template is the CONTRADICT operator, that allows the biologist to gather all the rules that "contradict" a certain hypothesis in the sense defined in [20, 23, 29]. This macro template was defined as follows:

CONTRADICT(*GeneExprSet*, G, *ExpLevel*) :=

BODY **HAS** ALL **OF** *GeneExprSet* **AND**

HEAD **HAS** ALL **OF** $G \neq \{$ *ExpLevel* $\}$

That is, if a biologist believes that a certain set of gene expressions *GeneExprSet* (possibly with some other genes) induces a particular gene G to have a certain expression level *ExpLevel*, the operator CONTRADICT can be used to check if there are any rules that contradict this belief, i.e., if there are any rules where *GeneExprSet* (possibly with some other genes) induces gene G to have the expression level other than *ExprLevel*. Assume, for example, that we have a template:

$$\textbf{CONTRADICT}(\{G1\uparrow, G2\uparrow\}, G4, \downarrow)$$

That is, we are assuming (e.g., based on previous biological knowledge) that whenever $G1$ and $G2$ are both upregulated, $G4$ should be downregulated. The above template would match any rule that includes $G1\uparrow$ and $G2\uparrow$ in its body and $G4\uparrow$ or $G4\#$ in its head, e.g., the rule $G1\uparrow \wedge G2\uparrow \wedge G23\downarrow \Rightarrow G4\uparrow$.

In summary, we took our proposed validation language SVL that is based on cardinality predicates and adapted it for the purpose of exploring the results of association rule mining algorithms on the biological microarray data. Note, that our validation approach is very flexible – we did not have any problems incorporating the specific knowledge from the domain of genomics, such as gene hierarchies, specific values of gene expression levels in filters, and the genomics-specific macros, including the CONTRADICT macro template.

6.3.2 Rule Grouping

There can be many "similar" rules among the discovered rules, and it would be useful for biologists to be able to explore and analyze all these similar rules together rather than individually for several reasons. First, instead of potentially dealing with millions of rules, a biologist can be provided with a much smaller number of rule classes (groups), which is easier to handle from the scalability point of view. Second, the ability to group the discovered rules provides a biologist with a high-level overview of the rules and with the capability of the exploratory top-down rule analysis of the rules. Therefore, it is important to provide methods that could group discovered rules into classes, according to some similarity criteria, that biologists could subsequently explore and analyze. For this purpose, we use our proposed similarity-based rule grouping method, described in Chapter 3.

In particular, we introduce a gene hierarchy that is specified by the domain expert in a form of a tree (some of these hierarchies are standardized already by biologists and he/she only needs to select the right one). Moreover, genes can be grouped in several different ways, and the biologist needs to select the most appropriate grouping of genes for the problem at hand. For example, one can organize genes based on their functions, e.g., several genes of the yeast (*S.cerevisiae*) are categorized by the biologists as having the *DNA repair* function. Therefore, the leaves of the attribute hierarchy represent all the genes from the microarray dataset to which rule discovery methods were applied, i.e., all genes that can potentially be present in the discovered rules. The non-leaf nodes in the tree are obtained by combining several lower-level nodes (genes) into one parent node (functional category of the genes). For instance, Figure 6.1 presents an example of such a hierarchy, where genes G1, G2, and G3 are combined into functional category F1 and genes G4 and G5 are combined into functional category F2. Then, functional categories F1 and F2 are further combined into a category ALL that represents all genes and corresponds to the root node of the tree.

Given a gene hierarchy, the discovered rules are grouped by specifying the *gene aggregation level* in this hierarchy. A gene aggregation level is defined by the subset of all the nodes (leaf and non-leaf) of the gene hierarchy, such that for every path

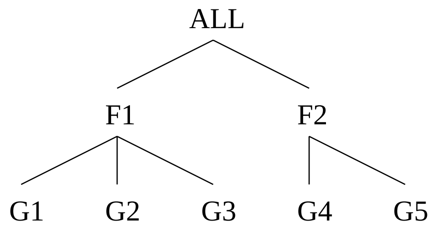


Figure 6.1: An example of a gene hierarchy.

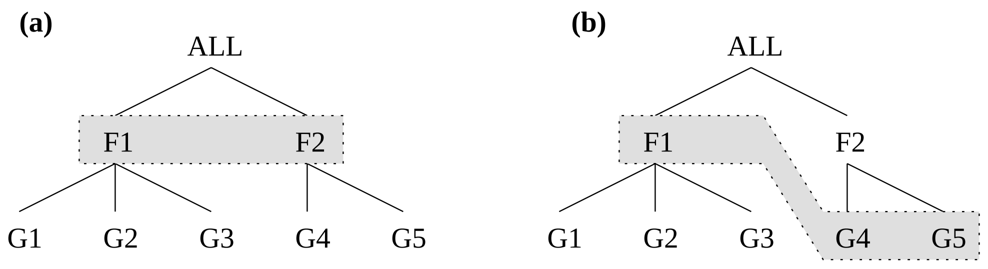


Figure 6.2: Examples of different gene aggregation levels.

from a leaf node to the root, exactly one node on such path belongs to this subset. Figure 6.2 presents several examples of gene aggregation levels. As mentioned earlier, the gene aggregation level is usually provided by a biologist.

After the rule aggregation level is specified, the discovered rules are *aggregated* by performing the following *syntactic* transformation: individual genes and their values (expression levels) in the rules are replaced with the corresponding node in the specified aggregation level. Consider, for example, the rule $G1 \uparrow \wedge G2 \downarrow \Rightarrow G5 \downarrow$ that indicates that whenever gene G1 is upregulated and gene G2 is downregulated, gene G5 is usually downregulated. By applying the aggregation level from Figure 6.2(a), we will get the aggregated rule $F1 \Rightarrow F2$, since both G1↑ and G2↓ will be replaced with F1 in the body of the rule and G5↓ will be replaced with F2 in the head of the rule.

When a set of rules is aggregated, different rules could be mapped into the same aggregated rule. For example, consider the rules $G1 \uparrow \wedge G2 \downarrow \Rightarrow G5 \downarrow$ and $G1 \uparrow \Rightarrow G4 \uparrow$. Using the aggregation level from Figure 6.2(a), both of these rules are mapped into the same aggregated rule $F1 \Rightarrow F2$. Therefore, we can group a set of biological rules as follows. Two rules R1 and R2 belong to the same group according to the specified aggregation level if and only if they are mapped into the same aggregated rule R. Therefore, two different aggregated rules represent two disjoint groups of rules.

Note, that all attributes (i.e., genes) in the microarray dataset have *exactly* the same set of possible values (i.e., expression levels ↑, ↓, and #). Therefore, we can use the expression level information in the grouping process, therefore providing even more flexibility to biologists.

Consider, for example, rule $G1 \uparrow \Rightarrow G4 \uparrow$. As demonstrated above, if we use the gene aggregation level from Figure 6.2(a) and do not use expression level information in the aggregation process, this rule will be mapped into the aggregated rule $F1 \Rightarrow F2$. However, if we choose to use the expression level information in the aggregation process, the rule $G1 \uparrow \Rightarrow G4 \uparrow$ will be mapped into the aggregated rule $F1 \uparrow \Rightarrow F2 \uparrow$. Which of these grouping schemes to use (i.e., to use the expression values or not) in the rule grouping process is up to the biologist. Note that, given the same gene aggregation level, using the gene expression levels in the grouping process will always produce more groups than not using them. Some examples of

Initial rule set S	Groups obtained by grouping S using gene aggregation level from Figure 6.2(a)	
	without exp. levels	with exp. levels
1) $G1\downarrow\Rightarrow G4\downarrow$	$F1\Rightarrow F1$ (rules 3,5)	$F1\uparrow\Rightarrow F1\downarrow$ (rules 3,5)
2) $G1\downarrow\Rightarrow G5\downarrow$	$F1\Rightarrow F2$ (rules 1,2,4)	$F1\downarrow\Rightarrow F2\downarrow$ (rules 1,2)
3) $G3\uparrow\Rightarrow G1\downarrow$		$F1\uparrow\wedge F1\downarrow\Rightarrow F2\downarrow$ (4)
4) $G1\uparrow\wedge G3\downarrow\Rightarrow G5\downarrow$		
5) $G1\uparrow\wedge G2\uparrow\Rightarrow G3\downarrow$		

Table 6.1: Grouping of biological rules using different grouping schemes.

rule grouping using these two different grouping schemes are presented in Table 6.1.

The proposed rule grouping method has the following distinguishing features that make it useful in the biological data mining applications requiring post-analysis of large numbers of discovered rules. First, unlike the traditional clustering methods [28], where the user has only a limited control over the structure and sizes of resulting clusters, a biologist has an explicit control over the granularity of the resulting rule groups in our approach. That is, a biologist can specify different gene aggregation levels in the gene hierarchy, which allows grouping the discovered rules according to the granularity important to that biologist (i.e., depending on how he or she wants to explore the rules). This property is very useful in the applications dealing with very large numbers of discovered rules (e.g., millions), because traditional clustering methods may still generate an unmanageable number of clusters. Moreover, the proposed method allows the biologist to incorporate the domain knowledge into the grouping process by specifying a gene hierarchy.

Second, the rule groups (denoted by different aggregated rules) obtained by the proposed rule grouping method are *equivalence classes*, since any two biological rules R1 and R2 belong to the same rule class if and only if they are mapped into the same aggregated rule, given a specific gene aggregation level. This means that we can determine what rule class the particular biological rule R belongs to based solely on the structure of rule R. This property makes the proposed rule grouping method consistent and predictable, since the biologist knows to what class a rule belongs *regardless* of what other discovered rules are. This is in contrast to some of the traditional distance-based clustering methods, where any two rules may or may not be in the same cluster depending on the other discovered rules.

Third, one of the limitations of some of the other rule grouping approaches (e.g., [31]), as applicable to the analysis of microarray data, lies in that it is not clear how to describe concisely the resulting rule cluster to the end-user for the purpose of evaluation, since rules belonging to the same cluster may have substantially different structures. In contrast, in our proposed grouping approach that is based on gene hierarchies, every rule cluster (group) is uniquely represented by its aggregated rule (common to all rules in that cluster), that is concise and descriptive.

Finally, the proposed rule grouping method works with large numbers of attributes, both numerical and categorical. Also, it scales up well. In fact, using lookup tables for gene hierarchies and hash table-based structures for storing aggregated rules, the grouping algorithm is linear in the total size of the rules to be grouped. This is especially important in bioinformatics applications dealing with very large numbers of rules.

6.3.3 Other Rule Processing Tools

Besides rule filtering and grouping tools, we have also developed *rule browsing* and *data inspection* tools. The rule browsing tool allows the biologist to view either individual rules or groups of rules based on the rule structure and select particular rules or groups of rules for a more in-depth analysis. Also, it turns out that biologists often want to know which experiments (transactions) a particular rule "covers," i.e., which experiments support (contribute) to this particular rule. To realize this requirement, we have implemented the data inspection tool that returns all the experiments on which the rule holds. Finally, all the rule exploration tools described in this section are combined into one integrated system that allows biologists to select them out of a "toolbox" and apply to the rulebase of genomic relationships in an iterative and interactive manner.

6.4 Case Study

We tested our methods on an Affymetrix GeneChip microarray data described in [44]. This data contained information on how cells of yeast (*S. cerevisiae*) respond to various environmental and chemical damaging factors, such as methylmethane sulfonate (MMS), 4-nitroquinoline n-oxide (4NQO), tert-butyl hydroperoxide (t-BuOOH), γ-radiation, and other factors. Yeast cells were treated by these various factors, and gene expression levels were recorded before the treatment and at 10, 30, and 60 minutes after the treatment. Altogether, 28 measurements were recorded on 28 microarray chips, each chip containing expression levels of the whole yeast genome containing approximately 6200 genes.

The biologists wanted to know how yeast cells respond to various damaging factors. To answer this broad question, they formulated several more specific questions that we explored with them using the methods described in Section 3. To demonstrate how our method works on this particular genomic problem, we focus on the following two biological questions that were of interest to the biologists:

1. When genes involved in the DNA repair are upregulated, what other gene categories are also up- or downregulated?

2. What genes induced after 10 minutes affect the transcription of genes induced after 30 minutes?

Since each of these questions deals with specific groups of genes, we applied different constraints to the association rule generation algorithm (using the method similar to the one described in [28]) for each of these two questions and therefore, generated *different* association rule sets for these two questions. This allowed us to generate fewer association rules for each question than we would have generated for the union of all possible questions. For example, based on the advise from biologists, one of the constraints for the first question was to generate only the rules that contain up to 4 genes (i.e., up to 3 genes in the body of the rule, 1 in the head). In the rest of this section, we will describe how we used our proposed biological rule filtering and grouping methods to handle these two questions.

The first question was relatively simple. The association rule discovery algorithm generated about 70,000,000 rules based on our initial constraints. To find all the rules that are relevant to this question we first applied the following template

$$\text{BODY } \textbf{HAS} \text{ ANY } \textbf{OF } [\mathit{DNA\_Repair}] \uparrow \ \textbf{ONLY AND}$$
$$\text{HEAD } \textbf{HAS} \text{ ANY } \textbf{OF } [\mathit{ALL\_GENES}] = \{\uparrow, \downarrow\}$$

which finds all rules of the form $X_1 \wedge \ldots \wedge X_k \Rightarrow Y$, where each gene expression X_i ($i = 1, \ldots, k$) represents an upregulated gene belonging to the DNA repair category, and Y is a gene expression involving a gene (any gene) that is either upregulated or downregulated, but not unchanged. This template matched 1,673 rules.

Then we identified similar rules among all the rules that matched the above template. Two rules were said to be similar if they could be mapped into the same aggregated rule by using the primary functional category of each gene as its aggregation level. Based on this similarity specification, we grouped these 1,673 rules into 78 groups. For example, 328 of the 1,673 identified rules belonged to the group $[\mathit{DNA\_Repair}] \uparrow \Rightarrow [\mathit{Protein\_Synthesis}] \downarrow$, which contains all the rules that have only upregulated DNA repair genes in the body and some downregulated protein synthesis gene in the head. These resulting 78 groups were of direct interest to biologists, and they were able to explore them further.

The second question was more involved. To answer this question, we grouped genes into three groups: group GrpA contained all the genes that were induced (i.e., up- or down-regulated) after 10 minutes, group GrpB contained all the genes induced only after 30 minutes (but not after 10), and group GrpC contained all the genes induced only after 60 minutes (but not after 10 or 30). Then the second question is reduced to the following more specific three questions:

- How genes in GrpA affect genes in GrpB;

- How genes in GrpA affect genes in GrpC;
- How genes in GrpB affect genes in GrpC.

To answer these questions, we started with the generation of the association rules pertinent to them. In particular, we used the constraints that removed genes with unknown function (category), the genes that were always upregulated (at all times), and a few other heuristics. Moreover, we used confidence and support levels of 70% and 5 records respectively. As a result, we managed to reduce the total number of genes from 6200 to 1809 as elements of single-item itemsets. Among these genes, 118 belonged to GrpA, 175 to group GrpB and 245 to GrpC. Using these genes, we generated 485,999 rules of the form $X \Rightarrow Y$, i.e., one gene expression in the body implies one gene expression in the head. Since in this case the biologists were interested in knowing how *pairs* of genes affect each other, we limited our considerations only to this type of rules for this question.

Because the other two questions were processed in a similar fashion, we will focus only on the first question in this paper, i.e., how genes in GrpA affect genes in GrpB. A fragment of the exploratory analysis performed for this question is presented in Figure 6.3. In particular, we started our explorations with the following template.

Template 1:

$$\begin{aligned} &\text{BODY } \textbf{HAS } 1 \textbf{ OF } [\mathit{GrpA}] = \{\uparrow, \downarrow\} \textbf{ AND} \\ &\quad \text{HEAD } \textbf{HAS } 1 \textbf{ OF } [\mathit{GrpB}] = \{\uparrow, \downarrow\} \end{aligned}$$

This template matched 1,725 rules. We next decided to refine the previous template in the following way.

Template 2:

$$\text{BODY } \textbf{HAS } 1 \textbf{ OF } [\mathit{GrpA}] \uparrow \ \textbf{AND}$$
$$\text{HEAD } \textbf{HAS } 1 \textbf{ OF } [\mathit{GrpB}] \downarrow$$

which produced no rules at all (this discovery was confirmed by the biologists based on their prior knowledge about behavior of these two groups of genes). Then we decided to apply the following template.

Template 3:

$$\text{BODY } \textbf{HAS } 1 \textbf{ OF } [\mathit{GrpA}] \uparrow \ \textbf{AND}$$
$$\text{HEAD } \textbf{HAS } 1 \textbf{ OF } [\mathit{GrpB}] \uparrow$$

and it produced 1,654 rules. Finally, we decided to apply the grouping tool that grouped these 1,654 rules into 140 groups, based on the functional categories of genes. These groups were examined by biologists, and some interesting groups were identified. Sample groups that were discovered (among the above- mentioned 140 groups):

- $[\mathit{Stress\_Response}] \uparrow \Rightarrow [\mathit{Amino\_Acid\_Metabolism}] \downarrow$

  This group contained 15 rules.

- $[\mathit{Transcription}] \uparrow \Rightarrow [\mathit{Protein\_Degradation}] \uparrow$

  This group contained 3 rules.

As a result of doing this type of analysis using our system, we managed to provide answers to the questions about gene regulation relationships that were of interest to the biologists.

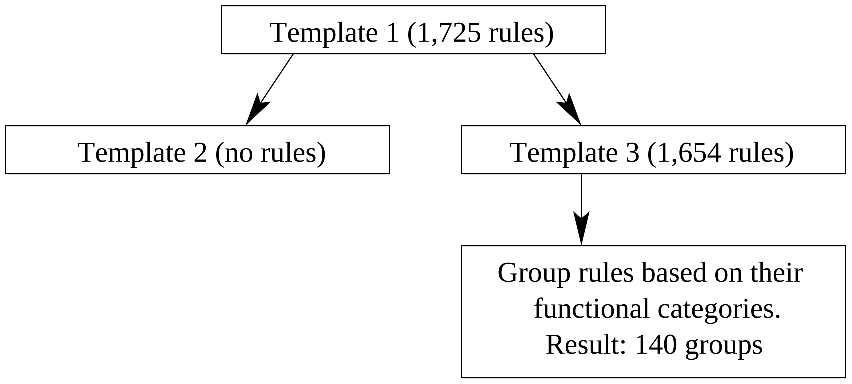


Figure 6.3: A sample exploratory analysis of biological rules.

6.5 Conclusions

One of the main challenges related to applying association rule discovery methods to microarray data is the scalability issue. Since the number of genes placed on microarray chips is often measured in thousands, association rule discovery methods tend to generate very large numbers of association rules describing how different genes regulate each other.

We addressed this problem by presenting several rule validation tools, such as rule filtering, grouping, browsing and data inspection tools that help the biologists explore and analyze very large numbers of discovered rules in an interactive and systematic manner. We tested our methods on some microarray data together with the biologists that produced the data. As a result, we managed to answer the questions about gene regulation relationships that were of interest to them using our methods.

Our approach empowers biologists by providing them with a set of tools that allow them to systematically analyze biological data on their own without any explicit help from the data miners. We believe that the development of such end-user tools is one of the major trends in data mining and believe that our work contributes in a significant way to this effort. Although we applied our system to the analysis of microarray data, our methods are more general and are also applicable to the analysis of other types of biological data, including proteomic and pharmacogenomic data.

Bibliography

[1] G. Adomavicius and A. Tuzhilin. Discovery of actionable patterns in databases: The action hierarchy approach. In *Proceedings of the Third International Conference on Knowledge Discovery and Data Mining*, August 1997.

[2] G. Adomavicius and A. Tuzhilin. User profiling in personalization applications through rule discovery and validation. In *Proceedings of the Fifth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, August 1999.

[3] G. Adomavicius and A. Tuzhilin. Expert-driven validation of rule-based user models in personalization applications. *Journal of Data Mining and Knowledge Discovery*, 5(1/2):33–58, 2001.

[4] G. Adomavicius and A. Tuzhilin. Multidimensional recommender systems: A data warehousing approach. In L. Fiege, G. Mühl, and U. Wilhelm, editors, *Proceedings of the Second International Workshop on Electronic Commerce (WELCOM 2001), Lecture Notes in Computer Science 2232*, pages 180–192, 2001.

[5] G. Adomavicius and A. Tuzhilin. Using data mining methods to build customer profiles. *IEEE Computer*, 43(2):74–82, 2001.

[6] C. C. Aggarwal, Z. Sun, and P. S. Yu. Online generation of profile association rules. In *Proc. of the Fourth Int'l Conference on Knowledge Discovery and Data Mining*, August 1998.

[7] C. C. Aggarwal and P. S. Yu. Online generation of association rules. In *Proceedings of the Fourteenth International Conference on Data Engineering*, 1998.

[8] R. Agrawal, T. Imielinsky, and A. Swami. Mining association rules between sets of items in large databases. In *Proceedings of the ACM SIGMOD Conference*, pages 207–216, 1993.

[9] 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*, chapter 12. AAAI Press, 1996.

[10] C. Allen, D. Kania, and B. Yaeckel. *Internet World Guide to One-to-One Web Marketing.* John Wiley & Sons, 1998.

[11] Y. Barash and N. Friedman. Context-specific bayesian clustering for gene expression data. In *Proceedings of the Fifth Annual International Conference on Computational Molecular Biology (RECOMB-2001)*, 2001.

[12] P. Baudisch, editor. *CHI'99 Workshop: Interacting with Recommender Systems*, May 1999. http://www.darmstadt.gmd.de/rec99/.

[13] R. J. Bayardo and R. Agrawal. Mining the most interesting rules. In *Proceedings of the Fifth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, August 1999.

[14] R. J. Bayardo, R. Agrawal, and D. Gunopulos. Constraint-based rule mining in large, dense databases. In *Proceedings of the 15th International Conference on Data Engineering*, March 1999.

[15] D. Berrar, W. Dubitzky, M. Granzow, , and R. Eils. Analysis of gene expression and drug activity data by knowledge-based association mining. In *Proceedings of Critical Assessment of Microarray Data Analysis Techniques (CAMDA'01)*, pages 25–28, 2001.

[16] S. Bicciato, M. Pandin, G. Didone, and C. Di Bello. Analysis of an associative memory neural network for pattern identification in gene expression data. In *Proceedings of BIOKDD'01*, 2001.

[17] D. D. Bowtell. Options available–from start to finish–for obtaining expression data by microarray. *Nature Genetics*, 21(1 Suppl):25–32, 1999.

[18] R. J. Brachman and T. Anand. The process of knowledge discovery in databases: A human-centered approach. In *Advances in Knowledge Discovery and Data Mining*, chapter 2. AAAI Press, 1996.

[19] L. Breiman, J. H. Friedman, R.A. Olshen, and C.J. Stone. *Classification and Regression Trees.* Wadsworth Publishers, 1984.

[20] S. Brin, R. Motwani, J.D. Ullman, and S. Tsur. Dynamic itemset counting and

implication rules for market basket data. In *Proceedings of the ACM SIGMOD Conference*, 1997.

[21] M. P. Brown, W. N. Grundy, D. Lin, N. Cristiani, C. W. Sugnet, T. S. Furey, M. Ares, and Haussler D. Knowledge-based analysis of microarray gene expression data by using support vector machines. *Proceedings of National Academy of Sciences*, 97(1), January 2000.

[22] C. Brunk, J. Kelly, and R. Kohavi. Mineset: An integrated system for data mining. In *Proceedings of the Third International Conference on Knowledge Discovery and Data Mining*, August 1997.

[23] *Communications of the ACM*, 40(3):56–89, 1997. Special issue on Recommender Systems.

[24] P. K. Chan. A non-invasive learning approach to building web user profiles. In *Workshop on Web Usage Analysis and User Profiling (WEBKDD'99)*, August 1999.

[25] D. Cheung, J. Han, V. Ng, and C. Y. Wong. Maintenance of discovered association rules in large databases: An incremental updating technique. In *Proceedings of 1996 International Conference on Data Engineering*. IEEE Computer Society, 1996.

[26] S. Clearwater and F. Provost. RL4: A tool for knowledge-based induction. In *Proceedings of the Second International IEEE Conference on Tools for Artificial Intelligence*, 1990.

[27] V. Dhar and A. Tuzhilin. Abstract-driven pattern discovery in databases. *IEEE Transactions on Knowledge and Data Engineering*, 5(6), 1993.

[28] R. Duda, P. Hart, and Stork D. *Pattern classification.* John Wiley & Sons, Inc., 2001.

[29] M. B. Eisen, P. T. Spellman, P. O. Brown, and D. Botstein. Cluster analysis and display of genome-wide expression patterns. *Proceedings of National Academy of Sciences*, 95(25), 1998.

[30] T. Fawcett and F. Provost. Combining data mining and machine learning for efficient user profiling. In *Proceedings of the Second International Conference on Knowledge Discovery and Data Mining*, August 1996.

[31] T. Fawcett and F. Provost. Adaptive fraud detection. *Journal of Data Mining and Knowledge Discovery*, 1(3):291–316, 1997.

[32] U. M. Fayyad, G. Piatetsky-Shapiro, and P. Smyth. From data mining to knowledge discovery: An overview. In *Advances in Knowledge Discovery and Data Mining*, chapter 1. AAAI Press, 1996.

[33] R. Feldman, Y. Aumann, A. Amir, and H. Mannila. Efficient algorithms for discovering frequent sets in incremental databases. In *Proceedings of the Workshop on Research Issues in Data Mining and Knowledge Discovery (DMKD'97)*, May 1997.

[34] W.J. Frawley, G. Piatetsky-Shapiro, and C.J. Matheus. Knowledge discovery in databases: an overview. In G. Piatetsky-Shapiro and W.J. Frawley, editors, *Knowledge Discovery in Databases.* AAAI / MIT Press, 1991.

[35] N. Friedman, M. Linial, I. Nachman, and D. Pe'er. Using bayesian networks to analyze expression data. In *Proceedings of the Fourth Annual International Conference on Computational Molecular Biology (RECOMB-2000)*, 2000.

[36] T. Fukuda, Y. Morimoto, S. Morishita, and T. Tokuyama. Data mining using two-dimensional optimized association rules: Scheme, algorithms, and visualization. In *Proceedings of the 1996 ACM SIGMOD International Conference on the Management Of Data*, pages 13–23, 1996.

[37] M. R. Garey and D. S. Johnson. *Computers and Intractability: A Guide to the Theory of NP-Completeness.* W. H. Freeman and Company, 1979.

[38] B. Goethals and J. Van den Bussche. A priori versus a posteriori filtering of association rules. In *Proceedings of the 1999 ACM SIGMOD Workshop on Research Issues in Data Mining and Knowledge Discovery*, 1999.

[39] J. Hagel. Keynote address at the personalization summit. San Francisco. November 16, 1999.

[40] J. Hagel and M. Singer. *Net Worth: Shaping markets when customers make the rules.* Harvard Business School Press, 1999.

[41] J. Han, Y. Fu, W. Wang, K. Koperski, and O. Zaiane. DMQL: A data mining query language for relational databases. In *Proceedings of the SIGMOD Workshop on Research Issues on Data Mining and Knowledge Discovery*, Montreal, June 1996.

[42] R. J. Hilderman and H. J. Hamilton. Knowledge discovery and interestingness measures: A survey. Technical report, University of Regina, 1999.

[43] T. Imielinski and A. Virmani. MSQL: A query language for database mining. *Journal of Data Mining and Knowledge Discovery*, 3(4), 1999.

[44] S. Jelinsky, P. Estep, G. Church, and L. Samson. Regulatory networks revealed by transcriptional profiling of damaged saccharomyces cerevisiae cells: Rpn4 links base excision repair with proteasomes. *Molecular and Cellular Biology*, 20(21), November 2000.

[45] H. Kautz, editor. *Recommender Systems. Papers from 1998 Workshop. Technical Report WS-98-08*. AAAI Press, 1998.

[46] M. Klemettinen, H. Mannila, P. Ronkainen, H. Toivonen, and A. I. Verkamo. Finding interesting rules from large sets of discovered association rules. In *Proceedings of the Third International Conference on Information and Knowledge Management*, December 1994.

[47] P. Kotala, Perera A., J. Kai Zhou, S. Mudivarthy, W. Perrizo, and E. Deckard. Gene expression profiling of DNA microarray data using peano count trees (p-trees). In *Online Proceedings of the First Virtual Conference on Genomics and Bioinformatics*, October 2001. URL: http://midas-10.cs.ndsu.nodak.edu/bio/.

[48] G. Kurra, W. Niu, and R. Bhatnagar. Mining microarray expression data for classifier gene-cores. In *Proceedings of BIOKDD'01*, 2001.

[49] E. L. Lawler. Sequencing jobs to minimize total weighted completion time subject to precedence constraints. *Annals of Discrete Mathematics*, 2:75–90, 1978.

[50] Y. Lee, B. G. Buchanan, and J. M. Aronis. Knowledge-based learning in exploratory science: learning rules to predict rodent carcinogenicity. *Machine Learning*, 30:217–240, 1998.

[51] B. Lent, A. N. Swami, and J. Widom. Clustering association rules. In *Proceedings of the Thirteenth International Conference on Data Engineering, April 7-11, 1997 Birmingham U.K*, pages 220–231. IEEE Computer Society, 1997.

[52] B. Lewin. *Genes VI.* Oxford University Press, 1997.

[53] B. Liu and W. Hsu. Post-analysis of learned rules. In *Proceedings of the AAAI Conference*, pages 828–834, 1996.

[54] B. Liu, W. Hsu, and S. Chen. Using general impressions to analyze discovered classification rules. In *Proceedings of the Third International Conference on Knowledge Discovery and Data Mining*, August 1997.

[55] B. Liu, W. Hsu, and Y. Ma. Pruning and summarizing the discovered associations. In *Proceedings of the Fifth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, August 1999.

[56] B. Liu, M. Hu, and W. Hsu. Multi-level organization and summarization of the discovered rules. In *Proceedings of the Sixth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, August 2000.

[57] R. Meo, G. Psaila, and S. Ceri. An extension to SQL for mining association rules. *Journal of Data Mining and Knowledge Discovery*, 2(2):195–224, 1998.

[58] Y. Morimoto, T. Fukuda, H. Matsuzawa, T. Tokuyama, and K. Yoda. Algo-

rithms for mining association rules for binary segmentations of huge categorical databases. In *Proceedings of the 24th VLDB Conference*, pages 380–391, 1998.

[59] S. Morishita. On classification and regression. In *Proceedings of the First International Conference on Discovery Science*, 1998.

[60] B. Padmanabhan and A. Tuzhilin. A belief-driven method for discovering unexpected patterns. In *Proceedings of the Fourth International Conference on Knowledge Discovery and Data Mining*, August 1998.

[61] B. Padmanabhan and A. Tuzhilin. Unexpectedness as a measure of interestingness in knowledge discovery. *Decision Support Systems*, 27(3):303–318, 1999.

[62] P. Pavlidis, C. Tang, and W. Noble. Classification of genes using probabilistic models of microarray expression profiles. In *Proceedings of BIOKDD'01*, 2001.

[63] D. Peppers and M. Rogers. *The One-to-One Future.* Doubleday, 1993.

[64] Personalization summit. San Francisco. November 14-16, 1999.

[65] P. A. Pevsner, Y. Lysov, K. R. Khrapko, A. Belyavski, Floreny'ev, and A. Mirzabekov. Improved chips for sequencing by hybridization. *Journal of Biomolecular Structure and Dynamics*, 9(2):399–410, 1991.

[66] G. Piatetsky-Shapiro and C. J. Matheus. The interestingness of deviations. In *Proceedings of the AAAI-94 Workshop on Knowledge Discovery in Databases*, 1994.

[67] F. Provost and D. Jensen. Evaluating knowledge discovery and data mining. In *Tutorial for the Fourth International Conference on Knowledge Discovery and Data Mining*, August 1998.

[68] J.R. Quinlan. *C4.5: Programs for Machine Learning.* Morgan Kaufmann, 1993.

[69] S. Sahar. Interestingness via what is not interesting. In *Proceedings of the Fifth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, August 1999.

[70] W.-M. Shen, K.-L. Ong, B. Mitbander, and C. Zaniolo. Metaqueries for data mining. In *Advances in Knowledge Discovery and Data Mining*, chapter 15. AAAI Press, 1996.

[71] A. Silberschatz and A. Tuzhilin. On subjective measures of interestingness in knowledge discovery. In *Proceedings of the First International Conference on Knowledge Discovery and Data Mining*, Montreal, Canada, August 1995.

[72] A. Silberschatz and A. Tuzhilin. User-assisted knowledge discovery: How much should the user be involved. In *Proceedings of the SIGMOD Workshop on Research Issues on Data Mining and Knowledge Discovery*, Montreal, June 1996.

[73] A. Silberschatz and A. Tuzhilin. What makes patterns interesting in knowledge discovery systems. *IEEE Transactions on Knowledge and Data Engineering*, 8(6), December 1996.

[74] I. Soboroff, C. Nicholas, and M. J. Pazzani, editors. *ACM SIGIR'99 Workshop on Recommender Systems: Algorithms and Evaluation*, August 1999. http://www.cs.umbc.edu/∼ian/sigir99-rec/.

[75] R. Srikant. *Fast Algorithms for Mining Association Rules and Sequential Patterns.* PhD thesis, University of Wisconsin, Madison, 1996.

[76] R. Srikant and R. Agrawal. Mining generalized association rules. In *Proceedings of the 21st International Conference on Very Large Databases*, September 1995.

[77] R. Srikant, Q. Vu, and R. Agrawal. Mining association rules with item constraints. In *Proceedings of the Third International Conference on Knowledge Discovery and Data Mining*, August 1997.

[78] C. Stedman. Data mining for fool's gold. *Computerworld*, 31(48), 1997.

[79] E. Suzuki. Autonomous discovery of reliable exception rules. In *Proceedings of the Third International Conference on Knowledge Discovery and Data Mining*, August 1997.

[80] P. Tamayo, D. Slonim, J. Mesirov, Q. Zhu, S. Kitareewan, E. Dmitrovsky, E. Lander, and T. Golub. Interpreting patterns of gene expression with self-organizing maps: Methods and applications to hematopoietic differentiation. *Proceedings of National Academy of Sciences*, 96, March 1999.

[81] P.-N. Tan, V. Kumar, and J. Srivastava. Selecting the right interestingness measure for association patterns. In *Proceedings of the Eight ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, July 2002.

[82] S. Thomas, S. Bodagala, K. Alsabti, and S. Ranka. An efficient algorithm for the incremental updation of association rules in large databases. In *Proceedings of the Third International Conference on Knowledge Discovery and Data Mining*, August 1997.

[83] H. Toivonen, M. Klemettinen, P. Ronkainen, K. Hatonen, and H. Mannila. Pruning and grouping discovered association rules. In *ECML-95 Workshop on Statistics, Machine Learning, and Knowledge Discovery in Databases*, April 1995.

[84] A. Tuzhilin and G. Adomavicius. Integrating user behavior and collaborative methods in recommender systems. In *CHI'99 Workshop. Interacting with Recommender Systems*, May 1999.

[85] A. Tuzhilin and A. Silberschatz. A belief-driven discovery framework based on data monitoring and triggering. Technical Report IS-96-26, Stern School of Business, New York University, December 1996.

[86] K. Wang, S. H. W. Tay, and B. Liu. Interestingness-based interval merger for numeric association rules. In *Proceedings of the Fourth International Conference on Knowledge Discovery and Data Mining*, August 1998.