South African
Computer
Journal
Number 12
November 1994

Suid-Afrikaanse
Rekenaar-
tydskrif
Nommer 12
November 1994

Computer Science and Information Systems

Rekenaarwetenskap en Inligtingstelsels

The South African Computer Journal
An official publication of the Computer Society of South Africa and the South African Institute of Computer Scientists

Die Suid-Afrikaanse Rekenaartydskrif
'n Amptelike publikasie van die Rekenaarvereniging van Suid-Afrika en die Suid-Afrikaanse Instituut vir Rekenaarwetenskaplikes

Editor
Professor Derrick G Kourie
Department of Computer Science
University of Pretoria
Hatfield 0083
Email: dkourie@dos-lan.cs.up.ac.za

Subeditor: Information Systems
Prof Lucas Introna
Department of Informatics
University of Pretoria
Hatfield 0083
Email: lintrona@econ.up.ac.za

Production Editor
Dr Riël Smit
Mosaic Software (Pty) Ltd
P.O.Box 23906
Claremont 7735
Email: gds@mosaic.co.za

Editorial Board

Professor Gerhard Barth
Director: German AI Research Institute

Professor Judy Bishop
University of Pretoria

Professor Donald D Cowan
University of Waterloo

Professor Jürg Gutknecht
ETH, Zürich

Professor Pieter Kritzinger
University of Cape Town

Professor Fred H Lochovsky
University of Science and Technology Kowloon

Professor Stephen R Schach
Vanderbilt University

Professor Basie von Solms
Rand Afrikaanse Universiteit

Subscriptions

	Annual	Single copy
Southern Africa:	R50,00	R25,00
Elsewhere:	$30,00	$15,00

to be sent to:
Computer Society of South Africa
Box 1714 Halfway House 1685

Phone: +27 (11) 315-1319 Fax: +27 (11) 315-2276

CALL FOR CONTRIBUTIONS

South African Computer Journal (SACJ): Special Issue

Information Technology and Development

South Africa, as a developing country, needs to find ways to harness it's enormous potential in a rapid and sustainable way. It is argued by some that the developments in information technology are increasingly creating opportunities for socio-economic development to be enabled by the diffusion of IT. If this is true, then these opportunities should surely be explored. Consequently, the editorial board of SACJ has decided to devote a special issue to this theme.

Some of the questions that the special issue might address are:

- To what degree can information technology enable (or accelerate) socio-economic development?
- How can the diffusion of information technology be achieved in order to maximize its contribution towards socio-economic development?
- What are the conditions required for such technology enabled development?
- Are there high leverage areas where quick returns can be achieved?
- Should the state play an active role and in what way?
- What are the experiences of other developing countries in information technology enabled socio-economic development?
- What are the moral and ethical issues involved in information technology enabled socio-economics development?

We invite *all* researchers in the field of information technology and other disciplines to contribute to this special issue. All contributions will be reviewed by three independent reviewers. Contributions should be sent in four copies to:

Prof Lucas D. Introna
South African Computer Journal (SACJ)
Special Issue: IT and Development
Dept. of Informatics
University of Pretoria
Pretoria, 0002
SOUTH AFRICA

Contributions should be received by June 5, 1995. Contributions sent by facsimile or e-mail will not be accepted. For any information regarding the special issue contact Prof Lucas Introna at +2712 4203376 (office), +2712 434501 (fax) or Internet: lintrona@econ.up.ac.za.

Important Dates

Contribution deadline:	June 5, 1995
Notification to author(s):	September 1, 1995
Final versions due:	November 1, 1995

LEXICA

an integrated environment for machine translation

LEXICA's capabilities include :

- **Translator's Workbench**
- **Powerful word-based Post Editor**
- **Automatic translator**
- **Integrated on-line dictionaries**
- **Support for most Windows-based wordprocessors**

LEXICA is a language independent platform

New language pairs can be developed on request

LEXICA provides unique support for languages spoken on the African continent, including :

- **Swahili**
- **Tswana**
- **Zulu**
- **Afrikaans**
- **Portuguese**

as well as :

- **German and French**

Eenheid vir Programmatuur-Ingenieurswese
Unit for Software Engineering

in association with
Department of Computer Science
University of Pretoria, Pretoria 0002 South Africa

Telephone: (012) 420-2504
Fax: 43-6454
E-mail: lexica@cs.up.ac.za

Specialization by Exclusion

Hendrik Theron Ian Cloete
Department of Computer Science, University of Stellenbosch, Stellenbosch 7600

Abstract

This paper presents a theoretical analysis of the search schemes employed by AQ, CN2 and the recently introduced BEXA algorithm. These covering algorithms induce disjunctive concept descriptions, and employ a general-to-specific search when constructing the conjunctions in these expressions. BEXA specializes a conjunction by excluding values from it, while CN2 and AQ specialize a conjunction by appending atoms to it. It is shown that the latter two algorithms' search process can also be viewed as one of excluding values. This makes it possible to show that the three algorithms' search schemes differ only with respect to the number of values excluded at each specialization step, and the number of different specializations that are constructed. We show that the search for accurate and simple conjunctions can be restricted to find elements of the set C_M of most general and consistent conjunctions. C_M is characterized precisely. BEXA exploits this characterization to induce simple and accurate concept descriptions efficiently.
Keywords: *Learning from examples, irredundant set covers*
Computing Review Categories: *I.2.5, I.2.6*

1 Introduction

The problem we consider is that of inducing a disjunctive VL_1 expression that distinguishes between the positive instances P and negative instances N of a given concept. Such an expression is said to be *consistent* and *complete*, i.e. it covers (matches) no instance in N, and it covers all the instances in P. VL_1 is Michalski's multiple-valued extension to propositional logic [4]. Table 1 contains a sample training set and examples of consistent and complete VL_1 expressions.

The two best known algorithms for inducing VL_1 expressions are Michalski's AQ algorithm [5, 2] and CN2 [2, 1]. There are many algorithms in the AQ family, but all of these algorithms share the same basic AQ algorithm that we will consider here [7]. BEXA is a new covering algorithm that was introduced recently by Theron and Cloete [10][1]. All three these algorithms follow the same basic **covering** [8] approach to construct concept descriptions: Consistent conjunctions are generated until all the positive instances are covered. AQ and CN2 construct a conjunction by starting with the constant **true**, and then appending atoms to the current conjunction until it covers only positive instances. In contrast, BEXA starts with the most general conjunction, denoted by **mgc**, and specializes a conjunction by excluding values until it becomes consistent. In both cases the key problem is to restrict the search process in such a way that accurate and simple conjunctions can be found in a reasonable time. Ideally, the search restrictions should avoid the generation of useless specializations without discarding potentially good specializations.

[1] BEXA was called GCA in [10]

Two of the covering algorithms, namely AQ and CN2, have been compared previously by Clark and Niblett [2] and Gams *et al.* [3] However, these comparisons did not characterize the search path of each algorithm through the lattice of VL_1 conjunctions. It is therefore not clear how the algorithms differ regarding the restrictions that they impose on the search process, nor what the relative merits of the different restrictions are.

Mitchell [6] showed that the expressions in many representation languages form a lattice under the partial ordering "is more specific than or equal to". He described the paths traced through this lattice by learning algorithms that follow a depth-first, breath-first, or candidate elimination approach to find a concept description. Mitchell also compared these search paths to point out the relative merits and weaknesses of each approach. However, his general lattice framework is not suitable for a detailed analysis of search schemes employed by AQ, CN2 and BEXA. Firstly, Mitchell's framework is only practical for representation languages where concept descriptions are conjunctive expressions, while the three algorithms considered in this paper induce disjunctive VL_1 expressions. Secondly, AQ, CN2 and BEXA follow a depth-first search. Mitchell's lattice framework is too general to illustrate the important differences in their respective implementations of this particular approach. The reason is that he exploits only the partial ordering relation among expressions in a representation langauge, and not the particular properties of a specific language which provide more specific guidance to the rule induction process.

This paper presents a theoretical analysis of the search schemes employed by BEXA, AQ and CN2. It is shown

Table 1. A sample learning problem and examples of VL_1 concept descriptions

Concept to learn
It will stop raining tomorrow

Attributes

Name	Type	Domain
outlook	nominal	{sunny,overcast,rain}
autumn	nominal	{yes,no}
temp	linear	{15..35}

The training set

#	outlook	autumn	temp	class
1	sunny	yes	17	–
2	overcast	no	18	–
3	rain	yes	16	–
4	sunny	yes	22	–
5	sunny	no	29	–
6	overcast	yes	30	–
7	overcast	no	35	–
8	rain	yes	23	–
9	rain	no	27	–
10	sunny	yes	28	+
11	overcast	no	23	+
12	sunny	no	27	+
13	rain	no	23	+

Consistent and complete concept descriptions

1. [outlook $\in$ {overcast,sunny}][22 < temp $\leq$ 28] $\vee$ [autumn = no][temp = 23] $\Rightarrow$ yes
2. [outlook = sunny][22 < temp $\leq$ 28] $\vee$ [autumn = no][temp = 23] $\Rightarrow$ yes

that the appending-atoms approach of AQ and CN2 can also be viewed as one of excluding values. This perspective is then used to show that the search restrictions employed by AQ, CN2 and BEXA differ only with respect to the number of values that are excluded at each specialization step, and the number of different specializations that are constructed for a conjunction at each step. The different restrictions are described precisely, and the merits of each approach are discussed.

The second contribution of this paper is a characterization of the set C_M of most general and consistent conjunctions. We stated above that search restrictions should discard useless specializations of a conjunction while retaining the good specializations. It will be shown that conjunctions in C_M are likely to be both accurate and simple. The search for good conjunctions can thus be restricted to find conjunctions in C_M. We will show that conjunctions in C_M correspond to irredundant set covers of N, the set of negative instances of a concept. This characterization of C_M provides a simple and powerful way to restrict the search for good conjunctions, and forms the basis for BEXA's search restrictions.

Section 2 describes the subset of VL_1 considered in this paper. Section 3 explains the problem of finding simple and accurate concept descriptions and describes the vital role that C_M plays in this search. Section 4 describes the BEXA algorithm. The search paths of BEXA, CN2 and AQ through the lattice of VL_1 conjunctions are described and compared in Section 5.

2 Basic Concepts and Definitions

The concept learning problem is defined as follows. *Nominal* attributes have unordered domains, e.g. outlook may have the domain {sunny,overcast,rain}. *Linear* (integer or real) attributes have totally ordered domains, e.g. 0 $\leq$ temp $\leq$ 120. Let A_1,...,A_n denote a set of attributes with domains D_1,...,D_n. The *instances space* I defined by these attributes is the cross-product $D_1 \times ... \times D_n$. An *instance* is the feature vector <**x**,c> where **x** $\in I$ and $c = +$ to denote a positive instance or $c = -$ to denote a negative instance.

We follow Haussler (1988) and use standard logic terminology to describe the subset of VL_1 [4] employed by BEXA, AQ and CN2. *Atoms* (*selectors* in Michalski's terminology) relate attributes to values. *Elementary* nominal atoms take the form $[A_i = a_i]$ with $a_i \in D_i$, e.g. [outlook = rain]. Elementary linear atoms take the form $[a_i \# A_i \# b_i]$ with # $\in \{<, \leq\}$, for example [10 $\leq$ temp < 20]. Open-ended intervals can be denoted with the universal bounds, e.g. [$-\infty$ < temp < 20] and [10 $\leq$ temp < ∞]. Atoms such as [10 $\leq$ temp $\leq$ 10] may be written as [temp = 10]. *Compound* nominal atoms take the form $[A_i = a_i \vee ... \vee a_l]$. We will denote such atoms as $[A_i \in S_i]$, where $S_i = \{a_i, ..., a_l\}$, e.g. [outlook $\in$ {sunny,rain}]. Compound linear atoms consist of any disjunction of elementary linear atoms, e.g. [(temp = 10) $\vee$ (20 $\leq$ temp < 30)]. A compound linear atom will always be written in its simplest form. For example, the atom

[(temp=10) $\vee$ (10<temp$\leq$25) $\vee$ (23<temp<40)]

will be written as [10 $\leq$ temp < 40]. *Expressions* are defined as follows. (1) Atoms are expressions. (2) A single atom, or a conjunction of atoms, is a *conjunctive* expression, e.g.

[outlook $\in$ {sunny,rain}][temp $\leq$ 20].

An implicit $\wedge$ (and) is assumed among the atoms in a conjunction. Michalski calls a conjunction a *complex*. (3) A *disjunctive* expression is the disjunction of a set of conjunctions. Haussler (1988) defines expressions that contain only elementary atoms as being *pure*, while expressions that contain at least one compound atom are *internally disjunctive*. We also make a finer distinction: A *linearly pure* expression contains only elementary atoms for linear attributes. For example, Rule 1 in Table 1 is internally disjunctive while Rule 2 is pure. Both rules are linearly pure because they contain only elementary atoms for temp. A disjunctive concept description can also be written as a set of production rules and vice versa: Each conjunction in a

disjunctive expression corresponds to one production rule.

Instances and expressions are related as follows. Let $B \subseteq I$ and let E denote an expression. The subset of instances in B covered by E is called the *extension* of E in B and is denoted by $X_B(E)$. The set $X_P(E)$ is called the positive extension of E, and $X_N(E)$ is called its negative extension. An expression E is thus *consistent* if $X_N(E) = \emptyset$ and *complete* if $X_P(E) = P$. A *set cover* of a given set A is a collection of its subsets whose union equals A. The cover is *irredundant* if no set can be deleted without uncovering some elements of A.

3 Searching for Good Concept Descriptions

Let H, the *hypothesis space*, denote the set of all possible VL_1 expressions that can be constructed from a given set of attributes and their domains. Let V denote the subset of all consistent and complete expressions in H. Mitchell [6] calls V the *version space*. The goal of a concept learning algorithm is to find a "good" expression in V, which usually means a description that is simple and that has a high accuracy on unseen instances (instances in $I - T$). The most common measure for description complexity is the total number of atoms that occur in all the conjunctions of a disjunctive concept description. This is also the measure that we will use. The accuracy of a disjunctive expression is determined by the accuracy of its conjunctions. The training set provided to a learning program is usually only a small subset of the instance space. It is therefore not possible to determine the precise accuracy of a conjunction on all the instances in I since the classifications of unseen instances are not known. Consequently, covering algorithms employ an evaluation function that estimates the accuracy of a conjunction on instances in $I - T$ based on the instances that it covers in T. These functions prefer conjunctions that cover many positive instances (i.e. that have much supporting evidence) to those that cover only a few positive instances [9]. The main problem is thus to find a concept description where each conjunction covers many positive instances and contains only a few atoms.

The eventual goal of this section is to show that accurate and simple concept descriptions can be found by constructing conjunctions in the set C_M of most general and consistent conjunctions. This goal is achie ved by giving a series of characterizations that eventually leads to a characterization of C_M.

The first problem is to distinguish between expressions in V and those in $H - V$, i.e. between expressions that are consistent and complete and those that are not. Let $h = h_1 \vee ... \vee h_n$ denote an expression in H. Then $h \in V$ if and only if

1. each conjunction h_i in h is consistent, i.e. $\bigcup X_N(h_i) = \emptyset$, and
2. the positive extensions of the conjunctions in h form a set cover of P, i.e. $\bigcup X_P(h_i) = P$.

For example, none of the conjunctions in Rule 1 in Table 1 cover any negative instances, and the positive extensions of these conjunctions form the set cover {{10,11,12}, {11,13}} of $P = \{10, 11, 12, 13\}$.

This characterization of V describes how consistent conjunctions can be combined into a complete expression, but it does not explain how the basic building blocks of such an expression, namely the consistent conjunctions, can be found. Let C denote the subset of conjunctions in H, and let C_C denote the subset of consistent conjunctions in C. The basic problem is thus to distinguish between conjunctions in C_C and those in C. Let $c \in C$ and let $R = \{r_1, ..., r_k\}$ denote the set of attribute values removed from the most general conjunction to obtain c. Then $c \in C_C$ if

3. the negative extensions of the values in R form a set cover of N, i.e. $\bigcup X_N(r_i) = N$.

For example, the most general conjunction for the learning problem in Table 1 is

M = [outlook ∈ {sunny,overcast,rain}] ∧ [autumn ∈ {yes,no}][15 ≤ temp ≤ 35].

The consistent conjunction

c_c =[outlook∈{rain,sunny}][autumn=no] ∧ [(15≤temp<27)∨(27<temp<29)∨(29<temp≤35)]

is obtained by excluding [outlook = overcast], [autumn = yes], [temp = 27], and [temp = 29] from M. The negative extensions of these values correspond to the set cover {{2,6,7}, {1,3,4,6,8}, {9}, {5}} of $N = \{1, 2, 3, 4, 5, 6, 7, 8, 9\}$. Each excluded value also uncovers a subset of positive instances. The positive extensions of the excluded values are {11}, {10}, {12} and $\emptyset$ respectively. The positive extension of the final conjunction is thus {13}.

The final problem is to find a way of constructing those consistent conjunctions that cover many positive instances and that contain only a few atoms. We now show that conjunctions in the set C_M of most general and consistent conjunctions have these two properties. The first step is to define C_M formally and to describe its basic properties. Let c and d denote two conjunctions. Then c is "more specific than or equal to" d, denoted by $c \leq d$, if and only if $X_I(c) \subseteq X_I(d)$ [6]. Two conjunctions are considered equal when they cover the same instances, i.e. $c = d$ if and only if $X_I(c) = X_I(d)$. We also define c to be strictly more specific than d, denoted by $c < d$, when $c \leq d$ and $c \neq d$. C_M is the subset of least specific (i.e. most general) conjunctions in C_C. Formally

$C_M = \{m \in C_C \mid$ there is no element $c \in C_C$ such that $m < c\}$

The definition of C_M and Property 3 leads to the following fundamental characterization of conjunctions in C_M: If $m \in C_C$ and $R = \{r_1, ..., r_k\}$ is the set of attribute values removed from the most general conjunction to obtain c, then $c \in C_M$ if

4. the negative extensions of the values in R form an **irredundant** set cover of N.

The irredundancy requirement ensures that generalizing m by adding even a single value to one of its atoms will cause it to become inconsistent. For example, the conjunction c_c given above is an element of C_M because the negative extensions of its excluded values form an irredundant set cover of N. Replacing any of these values will cause c_c to cover at least one negative instance.

Table 2. A small artificial learning problem

Attributes	Training set #	A	B	Class	#	A	B	Class
A ∈ {a,b,c}	1	a	x	+	4	b	y	+
B ∈ {x,y}	2	a	y	−	5	c	x	−
	3	b	x	+	6	c	y	+

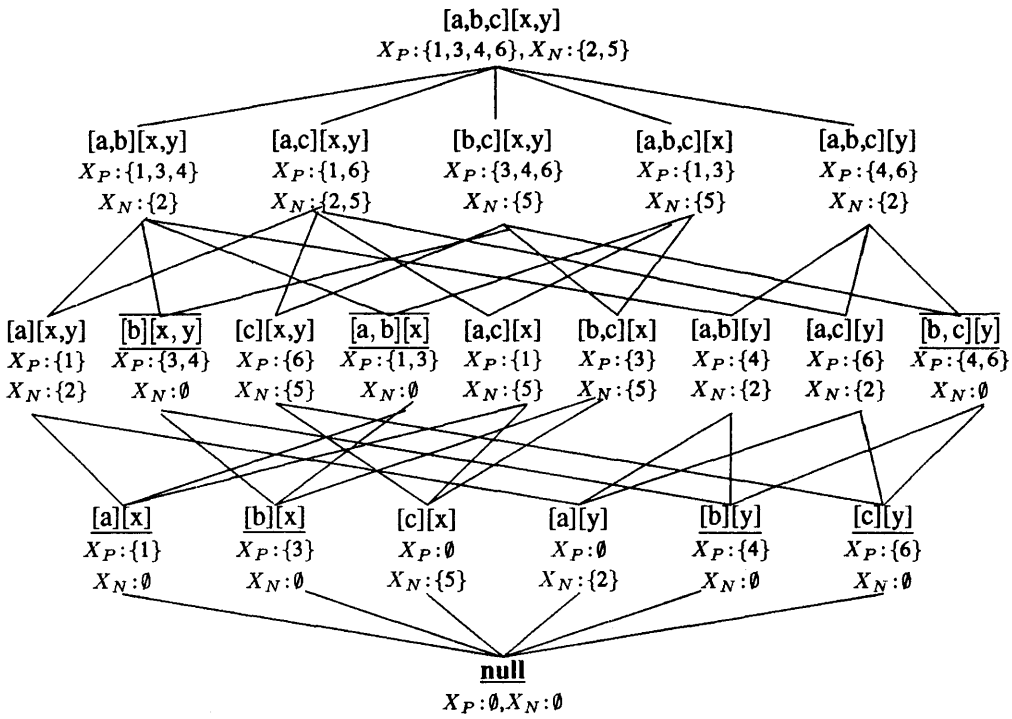

Figure 1. C_C and C_M in the lattice of conjunctions

A conjunction can be simplified if it can be generalized to such an extent that one or more of its atoms take all their values and can therefore be deleted. However, a conjunction in C_M cannot be generalized in any way without making it inconsistent. Conjunctions in C_M therefore contain a least number of atoms. Furthermore, the number of positive instances covered by a conjunction can only be increased by generalizing it. This implies that conjunctions in C_M also cover a maximum number of positive instances, because they cannot be generalized without also covering at least one negative instance. Conjunctions in C_M therefore exhibit both the properties of good conjunctions. The search for good conjunctions can therefore be restricted to find conjunctions in C_M. Property 4 provides a simple method for employing this restriction. It is the basis for BEXA's two most important search restrictions described in the next section.

The relationship between C_M and C_C can be depicted visually as follows. The set C of conjunctions form a lattice under the $\leq$ partial ordering defined above. The lattice is bounded from above by the most general conjunction and from below by the **null** conjunction (the conjunction with X_I(**null**) = $\emptyset$). Table 2 contains a small artificial learning problem, and Figure 1 contains the lattice of VL_1 conjunctions that exist for this problem. Elements of C_C are underlined, while elements of C_M also have a line above them. The consistent conjunction

c : [A = b][B = x]

can be generalized to

m : [A = b][B ∈ {x,y}],

an element of C_M, by replacing the value [B = x]. Conjunction m can be simplified to [A = b] because attribute B takes all its values. Conjunction m also covers two positive instances compared to the one covered by c. This conjunction is in C_M, and cannot be generalized further without covering at least one negative instance.

4 BEXA

Table 3 contains a simplified version of the BEXA algorithm that is sufficient for our discussion. A detailed description can be found in [9].

Procedure Uncover-N implements the search for a good conjunction. It explicitly maintains a conjunction's positive and negative extensions, as well as the sets usable and excluded. The excluded set contains all the values removed from the most general conjunction to obtain the current conjunction. The usable set contains all the values that will yield useful specializations when excluded from

Table 3. BEXA

```
PROCEDURE Cover-P(T: instance_set)
    P := the positive instances in T and N := T − P;
    usable := the set of attribute values a_i in T (determine and store X_P(a_i) and X_N(a_i) for each a_i);
    expression := empty;
    REPEAT
        conjuntion := Uncover-N(P,N,usable);
        expression := expression ∨ conjunction;
        P := P − X_P(conjunction)
    UNTIL P = ∅;
    RETURN expression
END Cover-P;

PROCEDURE Uncover-N (P,N : instance_set; usable : values_set);
    c := the most general conjunction with X_P(c) = P, X_N(c) = N, c.usable := usable, c.excluded := ∅;
    WHILE X_N(conjunction) ≠ ∅ DO
        {Remove from c.usable all values that will lead to unnecessary specializations.}
        FOR each a_i ∈ c.usable DO
            IF (X_P(c) − X_P(a_i) = ∅) OR                                          (1)
                (X_N(c) ∩ X_N(a_i) = ∅) OR                                         (2)
                ({X_N(b_i) | b_i ∈ c.excluded ∪ {a_i}} is a redundant partial cover of N)   (3)
            THEN c.usable := c.usable − {a_i}
        ENDFOR;
        {Create one specialization for each usable value}
        specializations := ∅;
        FOR each a_i ∈ c.usable DO
            c' := c specialized by excluding a_i from it;
            X_P(c') := X_P(c) − X_P(a_i) and X_N(c') := X_N(c) − X_N(a_i);
            c'.usable := c.usable − {a_i} and c'.excluded := c.excluded ∪ {a_i};
            specializations := specializations ∪ {c'}
        ENDFOR;
        c := the best conjunction in specializations according to the evaluation function;
    ENDWHILE;
    RETURN c
END Uncover-N;
```

the current conjunction.

BEXA employs three search constraints. Firstly, values leading to specializations covering no positive instances are deleted from usable because they will not cover any new positive instances (Step 1). Secondly, all values that will not uncover any new negative instances are also excluded from usable (Step 2). Thirdly, BEXA enforces the irredundancy[2] property to ensure that only those values that will lead to conjunctions in C_M are excluded (Step 3). The second restriction is a special case of the irredundancy restriction. The second restriction is important because it is more efficient than the number of subset tests required to implement the irredundancy restriction. It is therefore executed first so that the more time consuming irredundancy restriction is executed only when the first two restrictions do not apply. A detailed analyses of these restrictions can be found in [9]. After each specialization step, the best conjunction is selected for the next specialization step using a user-specified evaluation function. An example of such a function is CN2's Laplace accuracy estimate [1].

BEXA excludes only single values from the current conjunction. If some attributes have linear domains, this may lead to complex conjunctions such as c_c that contains a plethora of intervals. Such conjunctions also tend to have a low classification accuracy on unseen instances because they exclude only those negative instances taking one of the values explicitly excluded from the conjunction. Both problems can be avoided by generating linearly pure expressions. In this case, intervals of the form $[A_i \leq a_i]$ and $[A_i > a_i]$ are created for linear values that occur in the training set. These intervals are then excluded from a conjunction instead of single linear values. For example, the first conjunction in Table 1 is obtained by excluding the nominal value [outlook = rain] and the intervals [temp $\leq$ 22] and [temp > 28] from M. Linearly pure conjunctions corresponding to irredundant set covers of N may not necessarily be most general ones. The reason is that the intervals in some linear attributes can sometimes be enlarged without covering any negative instances. However, such conjunctions are still simplest ones because no atoms can be deleted without admitting some negative instances.

[2]Wells (1971) describes how irredundant set covers can be generated efficiently.

The conjunctions constructed by BEXA are underlined. The best one selected at each level is marked by a •.

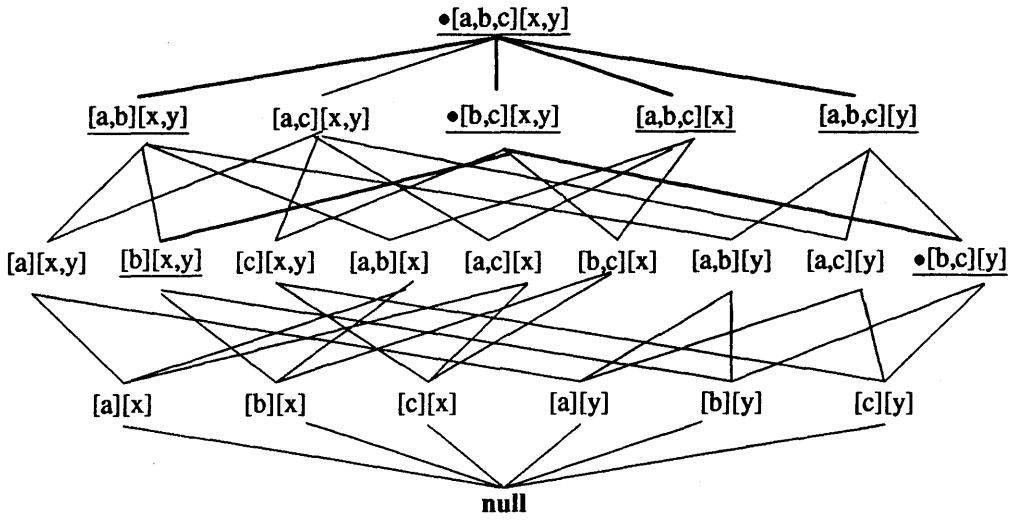

Figure 2. BEXA's path through the lattice of conjunctions

5 The search restrictions of BEXA, AQ, and CN2

A covering algorithm's *specialization model* determines the number and type of specializations that are constructed for a conjunction at each specialization step. This section describes the specialization models of AQ, BEXA, and CN2 and compares the search restrictions that they employ.

BEXA's specialization model

Figure 2 depicts the path followed by BEXA when constructing the first consistent conjunction for the training set given in Table 2. BEXA starts with the most general conjunction [A ∈ {b,c}][B ∈ {x,y}]. It specializes this conjunction by excluding single values from it. It thus considers all the useful specializations of the current conjunction at the next lower level in the lattice. Note that the specialization obtained by excluding [A = b] is not evaluated because this value does not uncover any new negative instances. It is removed from usable, and will therefore not be removed from any subsequent specializations. The specializations obtained by excluding [A = a] or [A = c] are the best because they cover three positive and one negative instance, while the other three useful specializations cover only two positive instances and at least one negative instance. Assume that BEXA selected [A ∈ {b,c}] [B ∈ {x,y}] as the new best conjunction. All its immediate and useful specializations are considered next. Note that [A = b] is not excluded because it had been removed from the usable set. Furthermore, [B = y] is deleted from usable since excluding it does not uncover a new negative instance. The remaining two specializations obtained by excluding either [A = c] or [B = x] both lead to consistent conjunctions covering only two positive instances. Either [A ∈ {b,c}][B = y] or [A = b][B ∈ {x,y}] may thus be returned as the best conjunction. Assume that BEXA selects the first expression as the best and return it.

In summary, each specialization step of BEXA considers all the useful specializations of a conjunction at the next lower level in the lattice. BEXA's three search restrictions discards from usable those values that lead to conjunctions covering no positive instances or that do not correspond to an element in C_M. This ensures that a most general and consistent, and thus a simplest, conjunction is generated. The restrictions delete a value from the usable set at the earliest possible moment. This increases efficiency by ensuring that no unnecessary specialization steps are performed.

AQ's specialization model

AQ generates a consistent conjunction as follows. Initially, the current conjunction c is the constant **true** that matches all the positive and negative instances. Any, as yet uncovered, positive instance s is selected as the *seed*. Then any negative instance n_e covered by c is selected. Specializations of c that do not cover n_e are then constructed as follows. All the attributes that take different values in s and n_e are determined. For each such attribute, the most general atom covering s and not n_e is generated. New specializations are then constructed by appending these atoms to the current conjunction. For example, consider the learning problem given in Table 2. Let instance 3 be the seed, and instance 2 be the first negative instance. These instances differ for both attributes A and B. The most general atom for A that covers 3 and not 2 is [A ∈ {b,c}]. Similarly, the most general atom for B that covers 3 and not 2 is [B = x]. Appending these atoms to **true** leaves them unchanged. The best conjunction is [A ∈ {b,c}] since it covers three positive instances compared to the two covered by [B = x], while both cover the negative instance 5. [A ∈ {b,c}] must

Conjunctions implicitly constructed by AQ are underlined. The one selected at each level is marked by •.

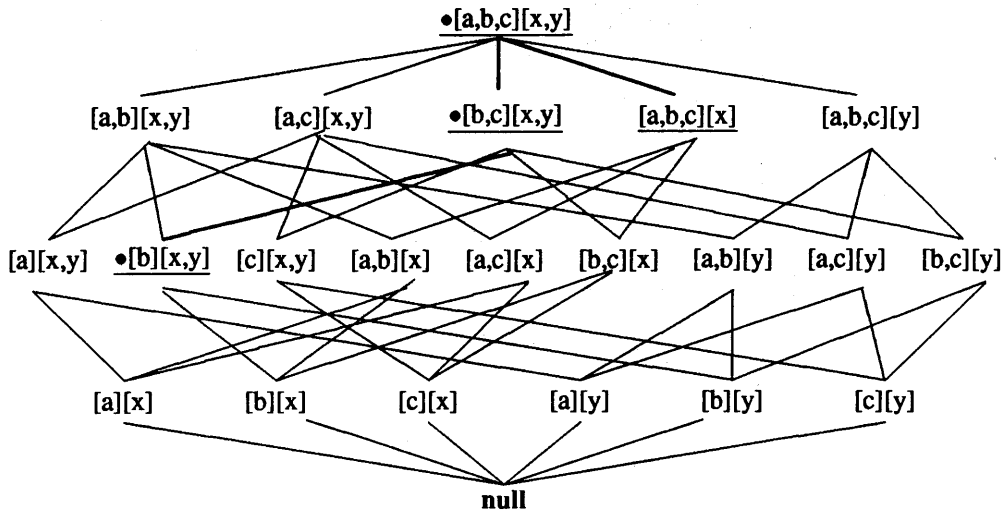

Figure 3. AQ's path through the lattice of conjunctions

thus be further specialized to uncover 5. Instances 3 and 5 differ for attribute A, leading to the atom [A ∈ {a,b}]. Appending it to the current conjunction yields the consistent conjunction [A = b].

AQ's specialization method is usually characterized as a mixed strategy [3]: The construction of the most general atoms that cover the seed but not the selected negative instance is the bottom-up part of the search, while specializing a conjunction by appending these atoms to it is the top-down part of the search. However, AQ's method can also be viewed as a restricted top-down search. AQ specializes a conjunction by appending an atom to it that contains all but one of its possible values. This amounts to implicitly removing one more value from the conjunction when written such that each attribute appears in it. For example, appending [A ∈ {b,c}] to **true** is equivalent to excluding [A = a] from the most general conjunction to yield [A ∈ {b,c}][B ∈ {x,y}]. Figure 3 depicts the path that AQ implicitly follows through the lattice of conjunctions when constructing the consistent conjunction. It differs from that followed by BEXA in that for each attribute, at most one specialization is considered for the current conjunction. AQ thus rigidly restricts the number of specializations to consider at each step to at most a (the number of attributes), while BEXA may consider all v (the number of attribute values) possible specializations at each step.

AQ does not explicitly check that the negative extensions of the values that it implicitly excludes from a conjunction form an irredundant set cover of N. Its conjunctions are therefore not guaranteed to be elements of C_M, and may thus not be simplest ones. Furthermore, a badly selected seed and negative instances may lead to very complex conjunctions that require many specialization steps to obtain. BEXA, on the other hand, performs a minimum number of specialization steps to construct a conjunction because no value is excluded unnecessarily.

CN's specialization model

CN2 generates conjunctions as follows. Initally, the positive and negative extension of each attribute value are determined. Starting with **true**, one new specialization is constructed by appending each of these values to the current conjunction. The best conjunction is selected for the next specialization step, and the process is repeated until the conjunction becomes consistent. For example, consider the problem in Table 2. The first specialization step creates the conjunctions [A = a], [A = b], [A = c], [B = x], and [B = y]. The best conjunction is [A = b] since it covers the two positive instances {3,4} and no negative instances. These positive instances are discarded, and the process is repeated. For the second conjunction, any one of [A = a], [A = b], [A = c], and [B = y] can be selected as the first best conjunction since they all cover one of the remaining positive instances and one negative instance. Assume that BEXA selects [B = y]. One new specialization is now created for each relevant value. This yields the conjunctions [A = a][B = y], [A = b][B = y], and [A = c][B = y]. The last one is selected because it covers a positive instance. The whole process is then repeated once more to find the conjunction [A = a][B = x] to cover the remaining positive instance. CN2 can generate only pure expressions, compared to the internally disjunctive expressions generated by AQ and BEXA. Pure conjunctions are more specific than internally disjunctive ones, and may thus cover fewer positive instances. Consequently, CN2 may have to generate more conjunctions to cover all the positive instances. This is illustrated by our example where CN2 requires at least three conjunctions to cover all the positive instances compared to the two that may be generated by AQ and BEXA.

Conjunctions implicitly constructed by CN2 are underlined. The one selected at each level is marked by •.

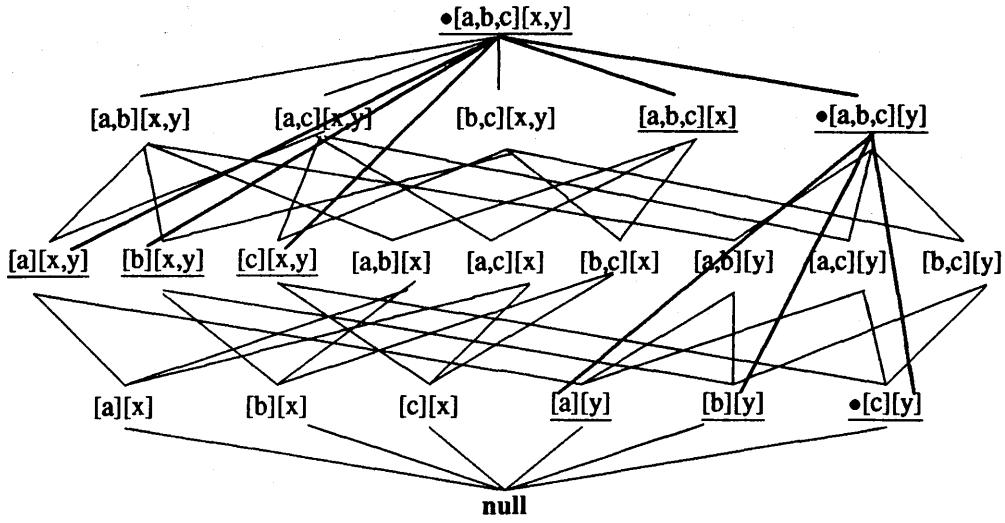

Figure 4. CN2's path through the lattice of conjunctions (when constructing the second conjunction)

CN2 also implicitly generates expressions by excluding values from them. Consider the second conjunction [A = c][B = y]. It was obtained by appending the atoms [A = c] and [B = y] to **true**. This is equivalent to excluding the atoms [A ∈ {a,b}] and [B = x] from the most general conjunction. CN2 thus excludes all but one value from an atom at each specialization step, compared to the single value excluded by AQ and BEXA. CN2 therefore makes larger "jumps" through the lattice of conjunctions at each specialization step than AQ and BEXA. CN2's path through the lattice of conjunctions when constructing the second conjunction is depicted in Figure 4. Similar to BEXA, CN2 does not rigidly restrict the number of specializations to consider at each specialization step. However, CN2 does restrict the type of descriptions that can be constructed to pure ones. Consequently, CN2 will require at most a (the number of attributes) steps to find a consistent conjunction, because all but one value is excluded for an attribute at each step. In contrast, AQ and BEXA may require up to $v - a$ steps, where v is the number of attribute values.

Similar to AQ, CN2 does not explicitly check that the negative extensions of the "values" that it implicitly excludes from a conjunction form an irredundant set cover of N. It may thus be possible to simplify CN2's pure conjunctions by deleting one or more of their atoms.

6 Summary

This paper showed that AQ's and CN2's approach of appending atoms can also be viewed as one of excluding values. This perspective made it possible to describe and analyze the search restrictions imposed by BEXA, AQ and CN2. BEXA and AQ can generate internally disjunctive conjunctions by excluding single values from a conjunction. CN2 generates only pure conjunctions because it always excludes all but one value from a conjunction. BEXA employs three dynamic search constraints that avoid the generation of useless specializations and guides the algorithm to find conjunctions in C_M. It was shown that conjunctions in C_M are simple and have a high value for the evaluation function. We also showed that conjunctions in C_M corresponds to irredundant set covers of N. This property is the basis for two of BEXA's search restrictions. AQ imposes one rigid constraint on the search process: it considers at most a (the number of attributes) different specializations at each specialization step. CN2 does not explicitly impose any restriction on the number of specializations that are constructed. However, it restricts the number of specialization steps to a because it generates pure concept descriptions.

One of the implicit assumptions of this paper was that the instances are noise-free. This is not a severe restriction because pruning schemes simply terminate the specialization process before consistency is achieved. In this case BEXA can guarantee that the conjunctions that are generated cannot be simplified by deleting atoms, nor can the number of positive instances that they cover be increased, without also covering more negative instances. Theron and Cloete [11, 9] compared BEXA experimentally with CN2 and the published results of AQ15 [7] (since none of the AQ algorithms are generally available). Michalski *et al.* [7] give accuracy (percentage correct) and complexity (the total number of atoms in a set of concept descriptions) results for AQ15 for three test databases. BEXA was applied to these three databases [9] following the experimental method described in [7]. AQ15's rules had an accuracy of 82%, 41% and 68% for the three databases, while BEXA's corresponding rule sets were 83%, 44% and

77% accurate. AQ15's rules contained 10, 257 and 7 atoms respectively, compared to BEXA's rules that contained 32, 31 and 2 atoms respectively. BEXA thus generated more accurate rules than AQ15 for all three databases, while its rules were also the simplest for two of the three databases.

Theron and Cloete [11] also compared CN2 and BEXA on ten test databases. Since the CN2 software was available, the significance of the results could be determined using the paired, two-sided t-test. BEXA generated significantly more accurate rules (at the 95% level) than CN2 for one database, while producing rules of similar accuracy but significantly lower complexity for six of the other test databases. CN2 produced rules of similar accuracy but significantly lower complexity for two of the remaining databases, while results for the tenth database were inconclusive (neither algorithm outperforming the other).

References

1. P Clark and R Boswell. 'Rule induction with CN2: Some recent improvements'. In Y Kodratoff, ed., *Machine Learning – European Working Session on Learning EWSL–91*, pp. 151–163, Berlin, (1991). Springer-Verlag.
2. P Clark and T Niblett. 'The CN2 induction algorithm'. *Machine Learning*, **3**:261–283, (1989).
3. M Gams and N Lavrac. 'Review of five empirical learning systems within a proposed schemata'. In I Bratko and N Lavrac, eds., *Progress in Machine Learning*, pp. 46–66. Sigma Press, (1987).
4. R S Michalski. 'Variable-valued logic and its applications to pattern recognition and machine learning'. In D C Rine, ed., *Computer Science and Multiple-valued logic: Theory and applications*, pp. 506–534. North Holland, (1975).
5. R S Michalski, W A Hoff, and R E Stepp. 'INDUCE 2: A program for learning structural descriptions from examples'. Technical report, University of Illinois at Urbana-Champaign, (1983).
6. T M Mitchell. 'Generalization a search'. *Artificial Intelligence*, **18**:203–226, (1982).
7. I Mozetic, J Hong, R S Michalski, and N Lavrac. 'The multi-purpose incremental learning system AQ15 and its testing application to three medical domains'. *Proceedings of the American association of artificial intelligence*, pp. 1041–1045, (1986).
8. J R Quinlan. 'Learning logical definitions from relations'. *Machine Learning*, **5**:239–266, (1990).
9. H Theron. *Specialization by exclusion: An approach to concept learning*. PhD thesis, University of Stellenbosch, South Africa, 1994.
10. H Theron and I Cloete. 'Beam search in attribute-based concept induction'. *South African Computer Journal*, **8**:32–36, (1992).
11. H Theron and I Cloete. 'BEXA: A covering algorithm based on a general covering framework'. Technical report, Department of Computer Science, University of Stellenbosch, (1993).

Acknowledgements: The test databases were provided by Patrick Murphy who manages the UCI Repository of Machine Learning Databases and Domain Theories (ml-repository@ics.uci.edu). CN2 was provided by Peter Clark and Robin Boswell of the Turing Institute in Glasgow.

Received: 3/94, Accepted: 5/94, Final copy: 6/94.

Notes for Contributors

The prime purpose of the journal is to publish original research papers in the fields of Computer Science and Information Systems, as well as shorter technical research papers. However, non-refereed review and exploratory articles of interest to the journal's readers will be considered for publication under sections marked as Communications or Viewpoints. While English is the preferred language of the journal, papers in Afrikaans will also be accepted. Typed manuscripts for review should be submitted in triplicate to the editor.

Form of Manuscript

Manuscripts for *review* should be prepared according to the following guidelines.

- Use wide margins and $1\frac{1}{2}$ or double spacing.
- The first page should include:
  - title (as brief as possible);
  - author's initials and surname;
  - author's affiliation and address;
  - an abstract of less than 200 words;
  - an appropriate keyword list;
  - a list of relevant Computing Review Categories.
- Tables and figures should be numbered and titled. Figures should be submitted as original line drawings/printouts, and not photocopies.
- References should be listed at the end of the text in alphabetic order of the (first) author's surname, and should be cited in the text in square brackets [1–3]. References should take the form shown at the end of these notes.

Manuscripts accepted for publication should comply with the above guidelines (except for the spacing requirements), and may be provided in one of the following formats (listed in order of preference):

1. As (a) LaTeX file(s), either on a diskette, or via e-mail/ftp – a LaTeX style file is available from the production editor;
2. As an ASCII file accompanied by a hard-copy showing formatting intentions:
   - Tables and figures should be on separate sheets of paper, clearly numbered on the back and ready for cutting and pasting. Figure titles should appear in the text where the figures are to be placed.
   - Mathematical and other symbols may be either handwritten or typed. Greek letters and unusual symbols should be identified in the margin, if they are not clear in the text.

   Further instructions on how to reduce page charges can be obtained from the production editor.
3. In camera-ready format – a detailed page specification is available from the production editor;
4. In a typed form, suitable for scanning.

Charges

Charges per final page will be levied on papers accepted for publication. They will be scaled to reflect scanning, typesetting, reproduction and other costs. Currently, the minimum rate is R30-00 per final page for LaTeX or camera-ready contributions that require no further attention. The maximum is R120-00 per page for contributions in typed format (charges include VAT).

These charges may be waived upon request of the author and at the discretion of the editor.

Proofs

Proofs of accepted papers in categories 2 and 4 above may be sent to the author to ensure that typesetting is correct, and not for addition of new material or major amendments to the text. Corrected proofs should be returned to the production editor within three days.

Camera-ready submissions will only be accepted if they are in strict accordance with the detailed guidelines. It is the responsibility of the authors to ensure that their submissions are error-free.

Letters and Communications

Letters to the editor are welcomed. They should be signed, and should be limited to less than about 500 words.

Announcements and communications of interest to the readership will be considered for publication in a separate section of the journal. Communications may also reflect minor research contributions. However, such communications will not be refereed and will not be deemed as fully-fledged publications for state subsidy purposes.

Book reviews

Contributions in this regard will be welcomed. Views and opinions expressed in such reviews should, however, be regarded as those of the reviewer alone.

Advertisement

Placement of advertisements at R1000-00 per full page per issue and R500-00 per half page per issue will be considered. These charges exclude specialized production costs which will be borne by the advertiser. Enquiries should be directed to the editor.

References

1. E Ashcroft and Z Manna. 'The translation of 'goto' programs to 'while' programs'. In *Proceedings of IFIP Congress 71*, pp. 250–255, Amsterdam, (1972).
2. C Bohm and G Jacopini. 'Flow diagrams, turing machines and languages with only two formation rules'. *Communications of the ACM*, **9**:366–371, (1966).
3. S Ginsburg. *Mathematical theory of context free languages*. McGraw Hill, New York, 1966.

South African
Computer
Journal

Number 12, November 1994
ISSN 1015-7999

Suid-Afrikaanse
Rekenaar-
tydskrif

Nommer 12, November 1994
ISSN 1015-7999

Contents

UP Drukkers
UP rinters