University of Bristol

DEPARTMENT OF COMPUTER SCIENCE

Background Knowledge in the Tertius First Order Knowledge Discovery Tool

T. S. Dahl

Also issued as ACRC-99:CS-006

March 1999 | CSTR-99-006

Abstract

This paper describes the three formalism available for specifying background knowledge in the Tertius first order logic discovery tool. It also describes the way background knowledge effects the results of Tertius.

The way Tertius handles integrity constraints is compared to an approach suggested in work done on the CLAUDIEN [1] learning system.

Finally a number of examples of use of Tertius for analysing the 'audiology' data-set [1] are also provided.

[1] The 'audiology' data-set is taken from the University of California at Irvine's Machine Learning Repository; http://www.ics.uci.edu/AI/ML/MLDBRepository.html

Contents

Chapter 1

Introduction

The Tertius first order logic discovery tool finds *confirmed* logic clauses of the form $\forall(A \leftarrow B)$ in data sets. A and B are arbitrary formulae possibly containing free variables and $\forall(.)$ denotes the universal closure over the free variables.

The confirmation measure of a clause is given as a combination of its *novelty* and its *satisfaction*. These two values are calculated on the basis of the expected and the observed frequencies of *counter-instances* of the clause.

The frequency of *counter-instances* is defined according to Table 1.1.

Tertius creates all possible ground instances of a clause. The literals of these instances are then tested against the data. For every ground instance of the clause, the value of the cell in the table that correctly describes the presence/absence of the different literal instantiations in the data-set is increased.

The marginal corresponding to a particular dimension is found by adding up all the cells in that dimension, i.e. the cells subscripted with the dimension we are looking at.

The formula Tertius uses to find the expected frequency of *counter-instances* is given in Equation 1.1

$$\mu_{\overline{A}B} = \frac{n_{\overline{A}} n_B}{N} \tag{1.1}$$

Tertius finds the observed number of counter-instances by finding how many of the ground instances of the formula has a satisfied body and a non-satisfied head.

	B	$\overline{B}$	
A	n_{AB}/μ_{AB}	$n_{A\overline{B}}/\mu_{A\overline{B}}$	n_A
$\overline{A}$	$n_{\overline{A}B}/\mu_{\overline{A}B}$	$n_{\overline{A}\overline{B}}/\mu_{\overline{A}\overline{B}}$	$n_{\overline{A}}$
	n_B	$n_{\overline{B}}$	N

Table 1.1: The contingency table used to evaluate the clause $\forall(A \leftarrow B)$.

The definition of *novelty*, $\Delta_{\overline{A}B}$, in terms of expected and observed frequencies of *counter-instances* is given in Equation 1.2

$$\Delta_{\overline{A}B} = \frac{\mu_{\overline{A}B} - n_{\overline{A}B}}{N} \tag{1.2}$$

The corresponding definition of *satisfaction*, $\sigma_{\overline{A}B}$, when $\mu_{\overline{A}B}$ is greater than zero is given in Equation 1.3. When $\mu_{\overline{A}B}$ is zero, $\sigma_{\overline{A}B}$ is one.

$$\sigma_{\overline{A}B} = \frac{\mu_{\overline{A}B} - n_{\overline{A}B}}{\mu_{\overline{A}B}} \tag{1.3}$$

The complete formula defining *confirmation* in terms of *novelty* and *satisfaction* is given elsewhere. The specialised idea of an expected value for a clause with a given background knowledge is expanded on in Section 3.1.

The way in which the background knowledge influences the expected value is given in Section 3.2.

Chapter 2

Representing Background Knowledge in Tertius

It is possible for a user to influence the results of the Tertius first order logic discovery tool by stating explicitly the knowledge he/she has of the problem domain. Tertius checks if the given clause or clauses constructed from rearranging the literals of the given clause, are implied by the background knowledge. Tertius then uses this knowledge in the evaluation of the *satisfaction* of the given clause. The procedure is described in detail in Sections 3.1 and 3.2.

The background knowledge can be presented in one out of three different formalisms; horn clauses, full predicate calculus or horn clauses with integrity constraints. The three different formalisms have three different corresponding proof procedures to check for clause implication. The formalisms and the corresponding proof procedures used to find implied clauses are presented in Section 2.1, 2.2 and 2.3 respectively.

2.1 Normal Clauses and Negation by Failure

The first formalism in which background knowledge can be presented to Tertius is as normal clauses. When background knowledge is represented in this format, Tertius uses SLDNF-Resolution to check whether a clause is implied or not.

To check if a general clause $(A_1, ..., A_n \leftarrow B_1, ...B_m)$ is implied by the given background knowledge, Tertius goes through the following three steps.

1. All variables in the clause are skolemised.
2. All the literals in the body, $B_1, ..., B_m$, are asserted into the background knowledge.
3. Query the literals in the head one by one, $A_1, ..., A_n$.

A success in proving any one of the literals in the clause head, $A_1, ..., A_n$, shows that the clause is implied by the background knowledge.

Tertius uses calls Sicstus Prolog to do the resolutions.

2.2 Full Predicate Calculus and Linear Resolution

Another formalism in which background knowledge can be presented to Tertius is a normal form which allows explicit negation and disjunctions in the head of clauses with empty bodies.

In this formalism, the logical inference is done using linear inference, a resolution method that can prove a larger set of clauses than the SLDNF-Resolution used in Prolog.

These two extensions to the Prolog framework allows the presentation of domain specific knowledge such as mutual exclusiveness of attribute values in the example below;

- $not\_blue(X); not\_red(X))$

The linear resolution is done by a Prolog Technology Theorem Prover implemented by Mark Stickel [3].

2.3 Normal Clauses with Integrity Constraints

The last formalism in which background knowledge can be presented is normal clauses with integrity constraints.

The integrity constraints are of the general form $ic(A_1, ..., A_n \leftarrow B_1, ..., B_m)$. The constraints are meta-level integrity constraint that describe what must be true in the produced rules at all times, taking the normal rules in the background knowledge into account.

Formalised, this responds to finding a set of rules r within the scope of a background theory T and a set of integrity constraints IC, where $\neg(T \rightarrow r)$ and $\neg((IC \cup r) \models \square)$.

The way this is implemented in Tertius, is by adding a constraint checking step after the implication checking step for background knowledge with normal clauses.

First, the skolemised literals in the body are asserted into the background theory. Second the head literal is queried.

If the clause is implied by the normal clauses in the background knowledge, Tertius will discard it for not being novel. If the clause is not implied by the normal clauses of the background knowledge, the satisfaction of the integrity constraints are checked before the clause is returned as novel and valid.

The integrity constraints are checked by first checking if the antecedal conjunction is satisfied by the normal clauses in the background knowledge. This checking is done by SLDNF resolution, i.e. the conjuncts are posed as Prolog queries in turn.

If the antecedents are not satisfied, the integrity constraint is taken to be satisfied. If the antecedents are satisfied, the constraint is only satisfied if the conclusion is satisfied as well. The conclusion is checked in the same way as the antecedent.

If a clause makes the conclusion of an integrity constraint false while the antecedent is true, the clause is discarded for contradicting the integrity constraints. This is done by setting the value of the corresponding cell in the contingency table to 0.

Chapter 3

How Tertius handles Background Knowledge

3.1 *Satisfaction* in Multi-dimensional Contingency Tables

In order to let background knowledge influence Tertius, the basic definition of *satisfaction* given in Section 1 is changed.

Instead of considering a clause as having a conjunction B, as an antecedent, we consider it to have an antecedent consisting of a conjunction of literals $B_1, ..., B_m$.

The contingency table we use to consider this clause is correspondingly expanded with $m + 1$ dimensions so that it has one dimension for the head of the clause and one for each of the literals in the body.

For the general three literal clause $(A \leftarrow B_1, B_2)$ we get the contingency table given in Table 3.1.

The value of the cell subscripted $\overline{A}B_1B_2$ in Table 3.1 is now taken to be the expected value for the clause.

The definitions of *confirmation*, *novelty* and *satisfaction* are adjusted to reflect the multi-dimensionality of the tables.

	B_1	$\overline{B_1}$	B_1/n_{B_1}	$\overline{B_1}/n_{\overline{B_1}}$	
A	$n_{AB_1B_2}$	$n_{A\overline{B_1}B_2}$	$n_{AB_1\overline{B_2}}$	$n_{A\overline{B_1B_2}}$	n_A
$\overline{A}$	$n_{\overline{A}B_1B_2}$	$n_{\overline{A}\overline{B_1}B_2}$	$n_{\overline{A}B_1\overline{B_2}}$	$n_{\overline{AB_1B_2}}$	$n_{\overline{A}}$
	B_2/n_{B_2}		$\overline{B_2}/n_{\overline{B_2}}$		N

Table 3.1: The three-dimensional table for the clause, $(A \leftarrow B_1, B_2)$

3.2 Redistribution of Instances

For a given set of literals there is a corresponding multi-dimensional contingency table. Every cell in the table corresponds to a clause with the negated clauses in the head and the positive literals in the body.

The first step Tertius takes in calculating the expected value for a clause, is to construct all clauses possible from the literals in the given clause and their negations. The number of ground instances that satisfy the constructed clauses are then entered into a multi-dimensional contingency table as observed values.

The next step is to check whether any of the constructed clauses are implied by the background knowledge. The value of any cell that corresponds to a clause implied by the background knowledge is set to zero. This step means that Tertius hold the background knowledge to be true even in the light of contradictory examples. If the value of a cell that corresponds to an implied clause was not zero, then the instances in that cell are treated as noise or bad data. When the data is perfect, the cell has a zero value already.

Thirdly, the marginals for both the positive and the negated literals in a dimension in calculated by adding up the values of the cells concerning that version of the literal. This is done from the after the zeroes have been entered for implied clauses.

Lastly, a new multi-dimensional table corresponding to the same set of literals is created. This table has the value zero in the cells corresponding to implied clauses and the value one in the other cells. This table is used as a base on which to redistribute the previously calculated marginals according to the following algorithm:

```
t = the new table of zeroes and ones
do until min_change < threshold
 min_change = 0
 for every dimension d of t
  for both the positive and the negated literal l of d
   s = the sum of all cells concerning l
   m = the previously calculated marginal for l
   d = m/s
   c = 1 - d
   if |c| > min_change then min_change = |c|
   for all the cell values v of cells concerning l
    v = v*d
```

The value of the cell that corresponds to the initial clause in the table of redistributed values is taken to be the expected value for that clause.

This redistribution allows background knowledge of clauses related to the initial clause to influence the expected value of the initial clause.

An Example Consider the clause $A \leftarrow B_1, B_2$ and the background knowledge $B_2 \leftarrow B_1$. Say the initial values as found from the data-set is as given in Table 3.2.

The background knowledge would make us expect zero values in the cells corresponding to the clauses $\overline{B_2} \leftarrow A, B_1$ and $\overline{B_2} \leftarrow \overline{A}B_1$. The resulting table is given as Table 3.3.

	B_1	$\overline{B_1}$	$B_1/7$	$\overline{B_1}/14$	
A	2	4	1	3	10
$\overline{A}$	4	3	0	4	11
	$B_2/13$		$\overline{B_2}/8$		21

Table 3.2: The initial table constructed from the data-set

	B_1	$\overline{B_1}$	$B_1/6$	$\overline{B_1}/14$	
A	2	4	0	3	9
$\overline{A}$	4	3	0	4	11
	$B_2/13$		$\overline{B_2}/7$		20

Table 3.3: The initial table with zeroes for implied clauses.

Tertius now records the marginals from Table 3.3.

The base table that is the basis of the redistribution is given as Table 3.4. In that table, the marginals from Table 3.3 are kept. The cell values of Table 3.4 are the same as those in Table 3.3 with all non-zero values replaced by the value one.

Tables 3.5 and 3.6 show the Table 3.4 after one and two cycles of the redistribution algorithm respectively. Table 3.7 shows the redistributed values.

	B_1	$\overline{B_1}$	$B_1/6$	$\overline{B_1}/14$	
A	1	1	0	1	9
$\overline{A}$	1	1	0	1	11
	$B_2/13$		$\overline{B_2}/7$		20

Table 3.4: The base table for redistributing the instances.

	B_1	$\overline{B_1}$	$B_1/6$	$\overline{B_1}/14$	
A	2.7	3.03	0	3.27	9
$\overline{A}$	3.3	3.7	0	4	11
	$B_2/13$		$\overline{B_2}/7$		20

Table 3.5: The table after the first cycle of the redistribution algorithm.

	B_1	$\overline{B_1}$	$B_1/6$	$\overline{B_1}/14$	
A	2.7	3.12	0	3.18	9
$\overline{A}$	3.3	3.8	0	3.9	11
	$B_2/13$		$\overline{B_2}/7$		20

Table 3.6: The table after the second cycle of the redistribution algorithm.

	B_1	$\overline{B_1}$	$B_1/6$	$\overline{B_1}/14$	
A	2.7	3.14	0	3.16	9
$\overline{A}$	3.3	3.84	0	3.86	11
	$B_2/13$		$\overline{B_2}/7$		20

Table 3.7: The redistributed instances.

Chapter 4

Integrity Constraints in Tertius' Background Knowledge

This section describes how Tertius is an improvement on other learning systems because it can use integrity constraints in its background knowledge. In particular it compares using Tertius with using CLAUDIEN in a integrity constraint setting that was originally presented to show how CLAUDIEN can handle integrity constraints [1].

Combining explanatory and descriptive Inductive Logic Programming can be done in a framework of declarative background knowledge with integrity constraints [1]. This framework is more expressive than normal horn clauses with negation as failure [2] and allows the learning of inherent non-monotonic theories.

The suggested algorithms can be implemented as scripts using any learning system that can do both explanatory and descriptive learning. Tertius has gone one step further. By taking integrity constraints into account while doing the learning it reduces the need for evaluation of learnt rules on a script level.

Tertius also provides a measure for satisfaction. This is a natural basis on which to judge which rules are suitable as integrity constraints.

4.1 Problem Definition

This Problem definition is taken from work done with the CLAUDIEN system [1]. It is quoted here to introduce the framework in which the integrated learning takes place.

4.1.1 The Hypothesis Language

The hypothesis language used in the CLAUDIEN work is identical to the language used by Tertius in the integrity constraint mode.

"A hypothesis is a triple $< T, C, IC >$, where C is a set of predicate symbols (the concepts to be learned), T is a Definite Horn theory (of rules defining concepts in C) and IC is a set of first order formulae (integrity constraints on the theory T)."

4.1.2 Entailment in the Hypothesis Language

"Let $H =< T, C, IC >$ be a hypothesis. A Generalised model, M, of H is any maximal subset of the minimal Herbrand model, N,"

4.1.3 The Learning Problem

The learning problem with integrity constraints in the background knowledge is specified as follows:

"Given:

1. background knowledge, B
2. a set, C, of concept predicates to be learned
3. training data, E, on the predicates of C, containing a set of positive examples E^+ and a set of negative examples E^-, such that $B \cup E$ is consistent.

Find: a hypothesis $H =< B \cup T, C, IC >$, such that:

1. $B \cup E^+ \models IC$
2. H covers e^+ for every e^+ in E^+
3. H does not cover e^-, for every e^- in E^-"

4.2 The 'Superpenguin' Example

The CLAUDIEN system has been used to demonstrate the advantages of using integrity constraints in background knowledge. By showing how one of their examples can be solved in Tertius, we demonstrate that this advantage is also available in that system.

"Example: Consider the background theory B
$bird(x) \leftarrow penguin(x)$
$penguin(x) \leftarrow superpenguin(x)$
$bird(a), bird(b), penguin(c), penguin(d),$
$superpenguin(e), superpenguin(f)$
with the set of concepts to be learned $C = flies(-)$. Consider also the training data $E = E^+ \cup E^-$ consisting simply of a set of positive and negative examples as follows: $E^+ = flies(a), flies(b), flies(e), flies(f)$ and $E^- = flies(c), flies(d)$."

4.2.1 Running Tertius on the 'Superpenguin' Example

We ran Tertius on the small example given in Section 4.2. In order to get more realistic distributions, we added 14 flying non-specified birds and 80 non-birds (20 of them flying), bringing the total number of examples to 100.

```
Tertius% tertius 3/1 birds2
/* 0.530012 0.000000 */ flies(A); penguin(A) :- bird(A).
/* 0.455868 0.020000 */ flies(A) :- bird(A).
/* 0.420437 0.200000 */ bird(A) :- flies(A).
/* 0.217857 0.000000 */ bird(A) :- penguin(A).
/* 0.162791 0.000000 */ superpenguin(A) :- flies(A), penguin(A).
/* 0.129987 0.160000 */ penguin(A) :- bird(A).
/* 0.125309 0.000000 */ flies(A) :- superpenguin(A).
/* 0.120914 0.020000 */ superpenguin(A) :- penguin(A).
/* 0.067323 0.160000 */ superpenguin(A) :- flies(A), bird(A).
/* 0.064851 0.180000 */ superpenguin(A) :- bird(A).
```

The (overly general) rule 'birds fly' and the integrity constraint 'only superpenguins are flying penguins', referred to previously, are found as the second and fifth most confirmed rule, respectively. Interestingly, the most confirmed rule can be translated to the negation-as-failure implementation of default rules with exceptions: flies(A):-bird(A),not penguin(A).

4.3 Suggested Learning Algorithms for Integrity Constraints

The learning algorithms proposed in the CLAUDIEN work, are top level scripts independent of what learning system is being used in so far as it only demands that it can do both explanatory and descriptive learning. I outline below how the three given algorithms can be implemented using Tertius.

4.3.1 Algorithm One

Algorithm one first learns the explanatory rules of C and then modifies these rules by adding descriptive rules as integrity constraints in IC. The following definition is given:

Algorithm 1
Find a set of rules R that cover all positive examples;
Decide-bias-of-ics(R,E;Bias);
Generate-ics(Bias,E;IC);
Choose-ics(IC,R,E;C); Return(R,C);

Tertius can be used in explanatory mode to construct a set of rules to cover all positive examples.

Tertius also offers a wide scope of rule bias including the number of literals used and whether to have one or more literals in the heads of clauses.

By using the descriptive learning mode of Tertius, a set of descriptive rules is learnt from which integrity constraints can be created. Tertius provides information

on whether rules are satisfied or not. Satisfied rules are an obvious first choice of rules to include as integrity constraints.

Apart from the indication of degree of satisfaction, this algorithm is the one that benefits the least from Tertius' ability to let integrity constraints guide the learning process.

4.3.2 Algorithm Two

Algorithm two uses the addition of an integrity constraint as a particular form of rule specification. The following algorithm is given:

Algorithm 2
$IC := \emptyset$; $Rules := \emptyset$;
$repeat$
Find-rule(R);
While (R, IC) covers E^-
Extended-Specialisation$(R; R', NIC)$;
$R := R'$; $IC := IC \cup NIC$;
$Rules := Rules \cup R$;
$until$ all positive examples are covered;
Return $Rules, IC)$.

The Extended-Specialisation procedure can choose to add an integrity constraint rather than specialise the rules in R. This relates the integrity constraints more strongly to the rules.

This algorithm can also be implemented by a script using Tertius rather than CLAUDIEN. In addition, the chosen integrity constraints can be added to the Tertius background knowledge when they are used for specialisation. Such an addition influences the further learning automatically rather than through a top level script.

It is not possible to add integrity constraints automatically during learning in Tertius, but using Tertius moves much of the evaluation of the learnt rules down from the script level to the learning system.

4.3.3 Algorithm Three

Algorithm three first constructs a set of descriptive integrity constraints IC and then creates rules which satisfy these constraints until all examples are covered.

Algorithm 3
Generate a set of ICs C;
(use descriptive criteria e.g. degree of applicability)
$repeat$
Generate rule R;
While ($covers(R, E^+) >$ thresh.) and not $satisf(R, C)$
Specialise-rule(R);

$R' = R' \cup R$;
$until$ all positive examples are covered;
$C' :=$ set of ICs violated by the generated rules;
Return (R', C');

Again Tertius can use the satisfaction information on descriptive rules to choose the initial set of integrity constraints.

The main advantage of Tertius over the CLAUDIEN approach in this example is that the construction of rules can be automatically guided by the chosen integrity constraints. This is a much more integrated solution than the top-level script solution outlined in the algorithm above. Tertius automatically excludes rules which do not satisfy the integrity constraints as described in Section 3.

Chapter 5

Experiments with Tertius and the Audiology Data-set

This section describes the results of analysing the *Audiology* data-set from the Machine Learning Repository at the University of California, Irvine. The analysis is done using the Tertius first order logic discovery tool.

The purpose of these experiments is to evaluate how well the general first order logic discovery tool Tertius can handle the specific task of classification. To put the results in context, the results are compared with the results of analysing the *audiology* data-set with pure classification tools.

5.1 Aims

The aim of this exercise is to learn how Tertius can best be used as a classification tool and also to evaluate how efficiently it performs such a function. To do this, the *audiology* data-set is analysed. The experience from performing the experiments are evaluated as well as the result of the experiments.

5.2 Method

The *audiology* data-set consists of 200 patients divided into 24 differently sized classes. The distribution over the classes is shown in Figure 5.1.

For each of the classes we use Tertius to find a classification rule. This is done by forcing the structure of the discovered clauses to have the form of a Horn clause with a grounded class predicate in the head. The grounded class predicate is specified as an initial clause.

All the classification experiments were timed and the promising rules were evaluated on the training set to find how well they performed.

The classifications are done in order of size of the classes with the largest class first.

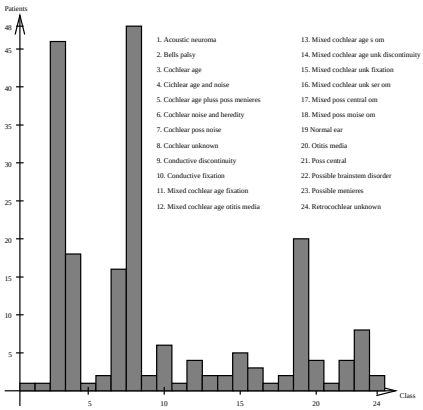

Figure 5.1: The *audiology* class distribution

5.3 The Cochlear Unknown Class

5.3.1 Initial Experiments

Experiment 1 The initial experiment used sampling rate of 0.5 and did not evaluate the result against the full data-set.

```
Tertius% /bin/time tertius -sc -b horn -S 0.5 3/1 audiology_standardized
Best and worst current values:
0.116769 0.333333 - 0.033099 0.020619
/* 0.116769 0.333333 */ class(A,cochlear_unknown) :- o_ar_u(A,normal), tymp(A,a).
/* 0.089522 0.273684 */ class(A,cochlear_unknown) :- o_ar_c(A,normal), o_ar_u(A,normal).
/* 0.077806 0.360000 */ class(A,cochlear_unknown) :- o_ar_u(A,normal).
/* 0.071475 0.520000 */ class(A,cochlear_unknown) :- tymp(A,a).
/* 0.071012 0.172414 */ class(A,cochlear_unknown) :- air(A,mild), ar_u(A,normal).
/* 0.063278 0.009259 */ class(A,cochlear_unknown) :- ar_c(A,elevated), o_ar_c(A,elevated).
/* 0.040366 0.453608 */ class(A,cochlear_unknown) :- ar_c(A,normal), tymp(A,a).
/* 0.039929 0.000000 */ class(A,cochlear_unknown) :- ar_u(A,normal), mod_sn_gt_3k(A).
/* 0.037187 0.010417 */ class(A,cochlear_unknown) :- ar_c(A,elevated), speech(A,normal).
/* 0.033099 0.020619 */ class(A,cochlear_unknown) :- notch_4k(A), o_ar_c(A,normal).

real       48.0
user       46.9
sys         0.0
```

A noted result was the ninth rule, as this was a satisfied rule.

Experiment 2 To get a better feel for the growth in complexity, the number of predicates was increased to four in this experiment.

```
Tertius% /bin/time tertius -sc -b horn -S 0.5 4/1 audiology_standardized
Best and worst current values:
0.121071 0.060976 - 0.047067 0.347368
/* 0.121071 0.060976 */ class(A,cochlear_unknown) :- o_ar_u(A,normal), speech(A,normal), static_normal(A).
/* 0.077806 0.360000 */ class(A,cochlear_unknown) :- o_ar_u(A,normal).
/* 0.071475 0.520000 */ class(A,cochlear_unknown) :- tymp(A,a).
```

```
/* 0.058498 0.323810 */ class(A,cochlear_unknown) :- o_ar_u(A,normal), static_normal(A), tymp(A,a).
/* 0.056160 0.343137 */ class(A,cochlear_unknown) :- ar_u(A,normal), static_normal(A), tymp(A,a).
/* 0.054221 0.306931 */ class(A,cochlear_unknown) :- ar_c(A,normal), ar_u(A,normal), tymp(A,a).
/* 0.052273 0.164948 */ class(A,cochlear_unknown) :- speech(A,normal), static_normal(A), tymp(A,a).
/* 0.049101 0.278351 */ class(A,cochlear_unknown) :- o_ar_c(A,normal), o_ar_u(A,normal).
/* 0.048868 0.164948 */ class(A,cochlear_unknown) :- o_ar_c(A,normal), speech(A,normal).
/* 0.047067 0.347368 */ class(A,cochlear_unknown) :- o_ar_u(A,normal), static_normal(A).

real     4:10.6
user     4:03.5
sys         0.0
```

The first rule stands out by not only being the most confirmed, but also being the least contradicted.

Experiment 3 To test if the noted rule in Experiment 1 was really satisfied the experiment was run again with the results evaluated against the whole data-set.

```
Tertius% /bin/time tertius -sc -b horn -SR 0.5 3/1 audiology_standardized
Best and worst current values:
0.116769 0.333333 - 0.033099 0.020619
/* 0.046005 0.365000 */ class(A,cochlear_unknown) :- o_ar_u(A,normal), tymp(A,a).
/* 0.022531 0.325000 */ class(A,cochlear_unknown) :- o_ar_c(A,normal), o_ar_u(A,normal).
/* 0.036871 0.390000 */ class(A,cochlear_unknown) :- o_ar_u(A,normal).
/* 0.054619 0.605000 */ class(A,cochlear_unknown) :- tymp(A,a).
/* 0.012729 0.215000 */ class(A,cochlear_unknown) :- air(A,mild), ar_u(A,normal).
/* 0.017964 0.020000 */ class(A,cochlear_unknown) :- ar_c(A,elevated), o_ar_c(A,elevated).
/* 0.002868 0.425000 */ class(A,cochlear_unknown) :- ar_c(A,normal), tymp(A,a).
/* 0.014790 0.010000 */ class(A,cochlear_unknown) :- ar_u(A,normal), mod_sn_gt_3k(A).
/* 0.010669 0.025000 */ class(A,cochlear_unknown) :- ar_c(A,elevated), speech(A,normal).
/* 0.005932 0.050000 */ class(A,cochlear_unknown) :- notch_4k(A), o_ar_c(A,normal).

real       45.9
user       43.0
sys         0.0
```

The ninth rule is not satisfied, but it is very weakly contradicted. Tested as a classification rule, it gives the following results:

```
| ?- classify((class(A,cochlear_unknown):-ar_u(A,normal),mod_sn_gt_3k(A))).

Correctly classified: 5
Positive errors: 2
Negative errors: 41
Examples missing values: 2
```

The rule classifies five out of the 48 examples correctly, but also classifies two examples from other classes as *cochlear unknown*.

This number of errors is considered too large compared to the number of correctly classified examples. This rule will therefore not be developed further.

Experiment 4 To verify the quality of the first clause from Experiment 2, the same experiment is run again with the results evaluated against the whole data-set.

```
Tertius% /bin/time tertius -sc -b horn -SR 0.5 4/1 audiology_standardized
Best and worst current values:
0.121071 0.060976 - 0.047067 0.347368
/* 0.030484 0.125000 */ class(A,cochlear_unknown) :- o_ar_u(A,normal), speech(A,normal), static_normal(A).
/* 0.036871 0.390000 */ class(A,cochlear_unknown) :- o_ar_u(A,normal).
/* 0.054619 0.605000 */ class(A,cochlear_unknown) :- tymp(A,a).
/* 0.054110 0.325000 */ class(A,cochlear_unknown) :- o_ar_u(A,normal), static_normal(A), tymp(A,a).
/* 0.038887 0.365000 */ class(A,cochlear_unknown) :- ar_u(A,normal), static_normal(A), tymp(A,a).
/* 0.008523 0.335000 */ class(A,cochlear_unknown) :- ar_c(A,normal), ar_u(A,normal), tymp(A,a).
/* 0.039082 0.190000 */ class(A,cochlear_unknown) :- speech(A,normal), static_normal(A), tymp(A,a).
/* 0.022531 0.325000 */ class(A,cochlear_unknown) :- o_ar_c(A,normal), o_ar_u(A,normal).
/* 0.013659 0.165000 */ class(A,cochlear_unknown) :- o_ar_c(A,normal), speech(A,normal).
/* 0.044354 0.350000 */ class(A,cochlear_unknown) :- o_ar_u(A,normal), static_normal(A).

real     5:22.6
user     3:44.2
sys         0.0
```

The first clause is still the least contradicted one, but the confirmation has taken a drastic fall. Testing how the clause does in classifiction gives the values below:

```
| ?- classify((class(A,cochlear_unknown) :- o_ar_u(A,normal), speech(A,normal), static_normal(A))).

Correctly classified: 19
Positive errors: 25
Negative errors: 27
Examples missing values: 2
```

These values show that the clause indeed covers a large portion of the class. Unfortunately it also includes a high number of examples from other classes.

Experiment 5 To try a different way of guiding the search towards better clauses, the sampling is dropped and only clauses with a contradiction lower that 0.1 are considered. This is done by asking for only satisfied clauses, but allowing a noise level of 0.1.

```
Tertius% /bin/time tertius -sc -b horn -sat -n 0.1 4/1 audiology_standardized
Best and worst current values:
0.009045 0.100000 - 0.000865 0.085000
/* 0.009045 0.100000 */ class(A,cochlear_unknown) :- air(A,mild), speech(A,normal).
/* 0.007218 0.095000 */ class(A,cochlear_unknown) :- air(A,normal), o_ar_u(A,normal), speech(A,normal).
/* 0.003919 0.100000 */ class(A,cochlear_unknown) :- ar_c(A,normal), bone_abnormal(A), o_ar_c(A,normal).
/* 0.012913 0.085000 */ class(A,cochlear_unknown) :- ar_c(A,elevated).
/* 0.008297 0.090000 */ class(A,cochlear_unknown) :- ar_u(A,normal), bone_abnormal(A).
/* 0.002690 0.095000 */ class(A,cochlear_unknown) :- bone_abnormal(A), o_ar_u(A,normal).
/* 0.000000 0.100000 */ class(A,cochlear_unknown) :- air(A,mild), bone(A,mild), bone_abnormal(A).
/* 0.000000 0.100000 */ class(A,cochlear_unknown) :- air(A,normal), bone(A,normal).
/* 0.008798 0.075000 */ class(A,cochlear_unknown) :- notch_4k(A).
/* 0.010115 0.070000 */ class(A,cochlear_unknown) :- o_ar_c(A,elevated).
/* 0.000865 0.085000 */ class(A,cochlear_unknown) :- ar_u(A,normal), speech(A,good).

real     2:57.0
user     2:46.7
sys         0.1
```

Experiment 6 In another attempt to find better clauses, sampling was dropped and a lower limit of 0.2 was set for clauses to be considered.

```
Tertius% /bin/time tertius -sc -b horn -c 0.2 4/1 audiology_standardized
Best and worst current values:
0.000000 0.000000   0.000000 0.000000

real     2:21.7
user     2:19.0
sys         0.0
```

No clauses were found with such a high confirmation.

Experiment 7 A third attempt to find better clauses was to increase the number of clauses that was considered for further refinement to twenty, from the default of ten.

```
Tertius% /bin/time tertius -sc -b horn -k 20 4/1 audiology_standardized
Best and worst current values:
0.070265 0.550000 - 0.023236 0.020000
/* 0.070265 0.550000 */ class(A,cochlear_unknown) :- static_normal(A), tymp(A,a).
/* 0.054619 0.605000 */ class(A,cochlear_unknown) :- tymp(A,a).
/* 0.054110 0.325000 */ class(A,cochlear_unknown) :- o_ar_u(A,normal), static_normal(A), tymp(A,a).
/* 0.046005 0.365000 */ class(A,cochlear_unknown) :- o_ar_u(A,normal), tymp(A,a).
/* 0.044354 0.350000 */ class(A,cochlear_unknown) :- o_ar_u(A,normal), static_normal(A).
/* 0.039082 0.190000 */ class(A,cochlear_unknown) :- speech(A,normal), static_normal(A), tymp(A,a).
/* 0.038887 0.365000 */ class(A,cochlear_unknown) :- ar_u(A,normal), static_normal(A), tymp(A,a).
/* 0.036871 0.390000 */ class(A,cochlear_unknown) :- o_ar_u(A,normal).
/* 0.035404 0.375000 */ class(A,cochlear_unknown) :- ar_u(A,normal), static_normal(A).
/* 0.032210 0.225000 */ class(A,cochlear_unknown) :- speech(A,normal), tymp(A,a).
/* 0.030484 0.125000 */ class(A,cochlear_unknown) :- o_ar_u(A,normal), speech(A,normal), static_normal(A).
/* 0.030028 0.140000 */ class(A,cochlear_unknown) :- ar_u(A,normal), speech(A,normal), static_normal(A).
/* 0.029141 0.285000 */ class(A,cochlear_unknown) :- o_ar_c(A,normal), o_ar_u(A,normal), static_normal(A).
/* 0.028525 0.415000 */ class(A,cochlear_unknown) :- ar_u(A,normal), tymp(A,a).
/* 0.027149 0.015000 */ class(A,cochlear_unknown) :- air(A,mild), o_ar_c(A,elevated), tymp(A,a).
/* 0.025776 0.010000 */ class(A,cochlear_unknown) :- ar_c(A,elevated), o_ar_c(A,elevated), tymp(A,a).
/* 0.025416 0.425000 */ class(A,cochlear_unknown) :- ar_u(A,normal).
```

```
/* 0.024858 0.265000 */ class(A,cochlear_unknown) :- ar_u(A,normal), o_ar_u(A,normal), static_normal(A).
/* 0.024791 0.155000 */ class(A,cochlear_unknown) :- o_ar_u(A,normal), speech(A,normal), tymp(A,a).
/* 0.023236 0.020000 */ class(A,cochlear_unknown) :- air(A,mild), ar_c(A,elevated), tymp(A,a).

real    12:29.6
user    11:14.1
sys         0.1
```

Experiment 8 To try to get some results with a lower limit on the confirmation, the limit of 0.2 from example six was lowered to 0.1.

```
Tertius% /bin/time tertius -sc -b horn -c 0.1 4/1 audiology_standardized
Best and worst current values:
0.000000 0.000000   0.000000 0.000000

real     4:39.7
user     4:13.2
sys         0.0
```

No clauses were found with this limit.

Experiment 9 To try to get some results with a limit on the confirmation, the limit of 0.08 from example nine was lowered to 0.05.

```
Tertius% /bin/time tertius -sc -b horn -c 0.05 4/1 audiology_standardized
Best and worst current values:
0.070265 0.550000 - 0.054110 0.325000
/* 0.070265 0.550000 */ class(A,cochlear_unknown) :- static_normal(A), tymp(A,a).
/* 0.054619 0.605000 */ class(A,cochlear_unknown) :- tymp(A,a).
/* 0.054110 0.325000 */ class(A,cochlear_unknown) :- o_ar_u(A,normal), static_normal(A), tymp(A,a).

real     6:32.0
user     6:27.6
sys         0.0
```

Experiment 10 An increase in the number of literals to five was tried with a sampling rate of 0.5 and with the results evaluated against the complete data-set.

```
Tertius% /bin/time tertius -sc -b horn -SR 0.5 5/1 audiology_standardized
Best and worst current values:
0.077806 0.360000 - 0.055567 0.295918
/* 0.036871 0.390000 */ class(A,cochlear_unknown) :- o_ar_u(A,normal).
/* 0.032476 0.120000 */ class(A,cochlear_unknown) :- o_ar_u(A,normal), speech(A,normal), static_normal(A),
tymp(A,a).
/* 0.030028 0.140000 */ class(A,cochlear_unknown) :- ar_u(A,normal), speech(A,normal), static_normal(A),
tymp(A,a).
/* 0.054619 0.605000 */ class(A,cochlear_unknown) :- tymp(A,a).
/* 0.016954 0.050000 */ class(A,cochlear_unknown) :- air(A,mild), ar_u(A,normal), speech(A,normal).
/* 0.027967 0.105000 */ class(A,cochlear_unknown) :- o_ar_c(A,normal), o_ar_u(A,normal), speech(A,normal),
static_normal(A).
/* 0.016892 0.245000 */ class(A,cochlear_unknown) :- ar_u(A,normal), o_ar_c(A,normal), o_ar_u(A,normal),
static_normal(A).
/* 0.031631 0.000000 */ class(A,cochlear_unknown) :- air(A,mild), ar_c(A,elevated), o_ar_c(A,elevated),
tymp(A,a).
/* 0.038887 0.365000 */ class(A,cochlear_unknown) :- ar_u(A,normal), static_normal(A), tymp(A,a).
/* 0.012668 0.280000 */ class(A,cochlear_unknown) :- ar_u(A,normal), o_ar_c(A,normal), o_ar_u(A,normal).

real     9:09.7
user     7:47.6
sys         0.3
```

Rule eight is a satisfied rule and rule five is very weakly contradicted. The performance of rule eight is:

```
| ?- classify((class(A,cochlear_unknown):-air(A,mild),ar_c(A,elevated),o_ar_c(A,elevated),tymp(A,a))).

Correctly classified: 6
Positive errors: 0
Negative errors: 42
Examples missing values: 0
```

This is the best rule so far and can be used as a start of a sequential covering algorithm.

The performance of the four four-predicate sub-clauses are:

```
| ?- classify((class(A,cochlear_unknown):-air(A,mild),ar_c(A,elevated),o_ar_c(A,elevated))).

Correctly classified: 6
Positive errors: 2
Negative errors: 42
Examples missing values: 0

| ?- classify((class(A,cochlear_unknown):-air(A,mild),ar_c(A,elevated),tymp(A,a))).

Correctly classified: 8
Positive errors: 4
Negative errors: 40
Examples missing values: 0

| ?- classify((class(A,cochlear_unknown):-air(A,mild),o_ar_c(A,elevated),tymp(A,a))).

Correctly classified: 8
Positive errors: 3
Negative errors: 40
Examples missing values: 0

| ?- classify((class(A,cochlear_unknown):-ar_c(A,elevated),o_ar_c(A,elevated),tymp(A,a))).

Correctly classified: 7
Positive errors: 2
Negative errors: 39
Examples missing values: 2
```

This shows that all possible generalisations of the eight rule result in positive classification errors.

Experiment 11 Five literals was tried with the default settings.

```
Tertius% /bin/time tertius -sc -b horn 5/1 audiology_standardized
Best and worst current values:
0.070265 0.550000 - 0.032476 0.120000
/* 0.070265 0.550000 */ class(A,cochlear_unknown) :- static_normal(A), tymp(A,a).
/* 0.054619 0.605000 */ class(A,cochlear_unknown) :- tymp(A,a).
/* 0.054110 0.325000 */ class(A,cochlear_unknown) :- o_ar_u(A,normal), static_normal(A), tymp(A,a).
/* 0.046005 0.365000 */ class(A,cochlear_unknown) :- o_ar_u(A,normal), tymp(A,a).
/* 0.044354 0.350000 */ class(A,cochlear_unknown) :- o_ar_u(A,normal), static_normal(A).
/* 0.039082 0.190000 */ class(A,cochlear_unknown) :- speech(A,normal), static_normal(A), tymp(A,a).
/* 0.038887 0.365000 */ class(A,cochlear_unknown) :- ar_u(A,normal), static_normal(A), tymp(A,a).
/* 0.036871 0.390000 */ class(A,cochlear_unknown) :- o_ar_u(A,normal).
/* 0.035404 0.375000 */ class(A,cochlear_unknown) :- ar_u(A,normal), static_normal(A).
/* 0.032476 0.120000 */ class(A,cochlear_unknown) :- o_ar_u(A,normal), speech(A,normal), static_normal(A),
tymp(A,a).

real    18:53.5
user    15:57.3
sys         0.1
```

All the rules have too high a contradiction.

Experiment 12 An increase in the number of literals to six was tried with a sampling rate of 0.3 and with the results evaluated against the complete data-set.

```
Tertius% /bin/time tertius -sc -b horn -SR 0.3 6/1 audiology_standardized
Best and worst current values:
0.121854 0.017544 - 0.081842 0.032258
/* 0.010776 0.080000 */ class(A,cochlear_unknown) :- ar_u(A,normal), bone_abnormal(A), static_normal(A),
tymp(A,a).
/* 0.020673 0.280000 */ class(A,cochlear_unknown) :- ar_c(A,normal), o_ar_u(A,normal), static_normal(A),
tymp(A,a).
/* 0.014653 0.035000 */ class(A,cochlear_unknown) :- ar_c(A,elevated), o_ar_u(A,normal), tymp(A,a).
/* 0.054619 0.605000 */ class(A,cochlear_unknown) :- tymp(A,a).
/* 0.032476 0.120000 */ class(A,cochlear_unknown) :- o_ar_u(A,normal), speech(A,normal), static_normal(A),
tymp(A,a).
/* 0.015224 0.130000 */ class(A,cochlear_unknown) :- ar_c(A,normal), ar_u(A,normal), o_ar_u(A,normal),
speech(A,normal), tymp(A,a).
/* 0.009601 0.125000 */ class(A,cochlear_unknown) :- air(A,normal), o_ar_c(A,normal), o_ar_u(A,normal),
tymp(A,a).
/* 0.038887 0.365000 */ class(A,cochlear_unknown) :- ar_u(A,normal), static_normal(A), tymp(A,a).
```

```
/* 0.022531 0.325000 */ class(A,cochlear_unknown) :- o_ar_c(A,normal), o_ar_u(A,normal).
/* 0.004814 0.055000 */ class(A,cochlear_unknown) :- ar_u(A,normal), bone_abnormal(A), o_ar_c(A,normal),
o_ar_u(A,normal).

real     5:08.5
user     4:40.5
sys         0.1
```

All high coverage rules also have high contradiction.

Experiment 13 Five predicates was retried with a 0.1 upper limit on the contradiction.

```
Tertius% /bin/time tertius -sc -b horn -sat -n 0.1 5/1 audiology_standardized
Best and worst current values:
0.024931 0.100000 - 0.009121 0.000000
/* 0.024931 0.100000 */ class(A,cochlear_unknown) :- ar_u(A,normal), o_ar_u(A,normal), speech(A,normal),
static_normal(A).
/* 0.016225 0.015000 */ class(A,cochlear_unknown) :- mod_sn_gt_3k(A).
/* 0.014288 0.000000 */ class(A,cochlear_unknown) :- bone(A,normal), bone_abnormal(A).
/* 0.014288 0.000000 */ class(A,cochlear_unknown) :- mod_sn_gt_4k(A).
/* 0.012913 0.085000 */ class(A,cochlear_unknown) :- ar_c(A,elevated).
/* 0.012333 0.100000 */ class(A,cochlear_unknown) :- air(A,normal), o_ar_c(A,normal), o_ar_u(A,normal),
static_normal(A).
/* 0.011571 0.065000 */ class(A,cochlear_unknown) :- ar_u(A,normal), bone(A,mild).
/* 0.010669 0.025000 */ class(A,cochlear_unknown) :- bone_abnormal(A), speech(A,normal).
/* 0.010230 0.095000 */ class(A,cochlear_unknown) :- air(A,normal), ar_c(A,normal), o_ar_u(A,normal),
static_normal(A).
/* 0.010115 0.070000 */ class(A,cochlear_unknown) :- o_ar_c(A,elevated).
/* 0.009121 0.000000 */ class(A,cochlear_unknown) :- bone(A,mild), mod_sn_gt_500(A).
/* 0.009121 0.000000 */ class(A,cochlear_unknown) :- air(A,mild), ar_u(A,normal), o_ar_c(A,absent).
/* 0.009121 0.000000 */ class(A,cochlear_unknown) :- air(A,mild), m_sn_gt_3k(A).
/* 0.009121 0.000000 */ class(A,cochlear_unknown) :- ar_u(A,normal), m_sn_gt_3k(A).

real     4:13.5
user     3:21.6
sys         0.1
```

Rule two and three are satisfied rules. Their performance are:

```
| ?- classify((class(A,cochlear_unknown) :- bone(A,normal), bone_abnormal(A))).

Correctly classified: 3
Positive errors: 0
Negative errors: 16
Examples missing values: 29

| ?- classify((class(A,cochlear_unknown) :- mod_sn_gt_4k(A))).

Correctly classified: 3
Positive errors: 0
Negative errors: 45
Examples missing values: 0
```

Experiment 14 Six predicates was retried with a 0.1 upper limit on the contradiction and a sampling rate of 0.5.

```
Tertius% /bin/time tertius -sc -b horn -SR 0.5 -sat -n 0.1 6/1 audiology_standardized
Best and worst current values:
0.064555 0.010870 - 0.034352 0.052083
/* 0.017964 0.020000 */ class(A,cochlear_unknown) :- air(A,normal), notch_4k(A).
/* 0.017964 0.020000 */ class(A,cochlear_unknown) :- ar_c(A,elevated), o_ar_c(A,elevated).
/* 0.022064 0.095000 */ class(A,cochlear_unknown) :- ar_c(A,normal), ar_u(A,normal), o_ar_u(A,normal),
speech(A,normal), static_normal(A).
/* 0.019234 0.130000 */ class(A,cochlear_unknown) :- o_ar_c(A,normal), speech(A,normal), static_normal(A),
tymp(A,a).
/* 0.021319 0.110000 */ class(A,cochlear_unknown) :- ar_c(A,normal), o_ar_c(A,normal), speech(A,normal),
static_normal(A).
/* 0.009969 0.010000 */ class(A,cochlear_unknown) :- air(A,normal), ar_c(A,normal), ar_u(A,normal),
mod_sn_gt_3k(A).
/* 0.017748 0.135000 */ class(A,cochlear_unknown) :- ar_c(A,normal), o_ar_c(A,normal), o_ar_u(A,normal),
speech(A,normal), tymp(A,a).
/* 0.016954 0.050000 */ class(A,cochlear_unknown) :- air(A,mild), ar_u(A,normal), speech(A,normal).
/* 0.024931 0.100000 */ class(A,cochlear_unknown) :- ar_u(A,normal), o_ar_u(A,normal), speech(A,normal),
static_normal(A), tymp(A,a).
/* 0.010115 0.070000 */ class(A,cochlear_unknown) :- air(A,normal), o_ar_u(A,normal), speech(A,normal),
static_normal(A).

real     4:06.1
user     2:34.6
sys         0.1
```

No suitable rule was found. All rules have low satisfaction.

5.3.2 Conclusions from Initial Experiments

It seems to be quite difficult to find rules that cover any number of examples without too many positive errors. This could be due to the nature of the *cochlear unknown* class, which might be collection class for all unclassifyable cases. This assumption is based on the name of the class and the difficulty of obtaining any satisfying results. The class is not known to be a collection class from the description of the data or any knowledge of the domain.

5.3.3 Covering the Cochlear Unknown Class

The rules found in Section 5.3.1 cover only small parts of the *cochlear unknown class* It is desireable to find a set of rules that together would correctly classify all members of that class without making any positive mistakes.

It is also desireable to have as few rules as possible to ensure that we generalise where possible. Two different approaches to removing are tried below. In both approaches , rule eight from Experiment 10, which covered six examples and the rules from Experiment 13, which both covered three examples, will be the basis for further covering.

A new program which returns the identity of the correctly classified examples was run with the three rules to analyse the relation between the examples they identify:

```
| ?- classify((class(A,cochlear_unknown) :- air(A,mild), ar_c(A,elevated),o_ar_c(A,elevated),tymp(A,a))).

Correctly classified: 6
Correctly classified ids: [e43,e44,e69,e70,e181,e182]
Positive errors: 0
Negative errors: 0
Examples missing values: 42

| ?- classify((class(A,cochlear_unknown) :- bone(A,normal), bone_abnormal(A))).

Correctly classified: 3
Correctly classified ids: [e25,e26,e124]
Positive errors: 0
Negative errors: 0
Examples missing values: 45

| ?- classify((class(A,cochlear_unknown) :- mod_sn_gt_4k(A))).

Correctly classified: 3
Correctly classified ids: [e55,e105,e173]
Positive errors: 0
Negative errors: 0
Examples missing values: 45
```

All these rules have disjunct sets of covered examples. This means that they can be used together without the ordering that is necessary in a sequential covering approach.

5.3.4 Covering by Removing Examples

By removing the examples classified by a rule and learning new rules for the remainding examples, it is possible to create an ordering of rules that together will classify a whole class.

The twelve examples covered by the following rules were removed from the set of facts and a new set of experiments were initiated.

Experiment 15 An initial experiment with three predicates and a sampling rate of 0.5.

```
Tertius% /bin/time tertius -sc -b horn -SR 0.5 3/1 audiology_standardized
Best and worst current values:
0.088627 0.360360 - 0.036982 0.000000
/* 0.050705 0.388298 */ class(A,cochlear_unknown) :- o_ar_u(A,normal), tymp(A,a).
/* 0.025684 0.452128 */ class(A,cochlear_unknown) :- ar_u(A,normal).
/* 0.053674 0.585106 */ class(A,cochlear_unknown) :- static_normal(A), tymp(A,a).
/* 0.045743 0.345745 */ class(A,cochlear_unknown) :- o_ar_c(A,normal), o_ar_u(A,normal).
/* 0.017021 0.356383 */ class(A,cochlear_unknown) :- ar_c(A,normal), ar_u(A,normal).
/* 0.046448 0.643617 */ class(A,cochlear_unknown) :- tymp(A,a).
/* 0.043047 0.414894 */ class(A,cochlear_unknown) :- o_ar_u(A,normal).
/* 0.023391 0.021277 */ class(A,cochlear_unknown) :- air(A,normal), notch_4k(A).
/* 0.012167 0.409574 */ class(A,cochlear_unknown) :- ar_c(A,normal), static_normal(A).
/* 0.012651 0.010638 */ class(A,cochlear_unknown) :- history_roaring(A), speech(A,normal).

real       40.5
user       39.9
sys         0.0
```

No clauses of particular interest.

Experiment 16 The number of predicates was increased to five and the sampling rate reduced to 0.3.

```
Tertius% /bin/time tertius -sc -b horn -SR 0.3 5/1 audiology_standardized
Best and worst current values:
0.150414 0.206349 - 0.079681 0.270270
/* 0.032693 0.260638 */ class(A,cochlear_unknown) :- ar_u(A,normal), o_ar_c(A,normal), o_ar_u(A,normal), \\
static_normal(A).
/* 0.001992 0.085106 */ class(A,cochlear_unknown) :- ar_c(A,normal), speech(A,good).
/* 0.020090 0.175532 */ class(A,cochlear_unknown) :- o_ar_c(A,normal), speech(A,normal), tymp(A,a).
/* 0.046515 0.372340 */ class(A,cochlear_unknown) :- o_ar_u(A,normal), static_normal(A).
/* 0.032189 0.340426 */ class(A,cochlear_unknown) :- ar_c(A,normal), o_ar_u(A,normal), tymp(A,a).
/* 0.034858 0.303191 */ class(A,cochlear_unknown) :- ar_c(A,normal), o_ar_u(A,normal), static_normal(A).
/* 0.012651 0.010638 */ class(A,cochlear_unknown) :- ar_c(A,normal), history_dizziness(A),
history_roaring(A), \\
o_ar_c(A,normal).
/* 0.017605 0.058511 */ class(A,cochlear_unknown) :- air(A,mild), speech(A,normal), static_normal(A),
tymp(A,a).
/* 0.015872 0.063830 */ class(A,cochlear_unknown) :- ar_u(A,normal), bone(A,mild), static_normal(A).
/* 0.049961 0.303191 */ class(A,cochlear_unknown) :- o_ar_c(A,normal), o_ar_u(A,normal), static_normal(A).

real     3:10.3
user     3:09.5
sys         0.0
```

No clauses of particular interest.

Experiment 17 A full search of five predicates was tried.

```
Tertius% /bin/time tertius -sc -b horn 5/1 audiology_standardized
Best and worst current values:
0.054381 0.345745 - 0.036485 0.271277
/* 0.054381 0.345745 */ class(A,cochlear_unknown) :- o_ar_u(A,normal), static_normal(A), tymp(A,a).
/* 0.053674 0.585106 */ class(A,cochlear_unknown) :- static_normal(A), tymp(A,a).
/* 0.050705 0.388298 */ class(A,cochlear_unknown) :- o_ar_u(A,normal), tymp(A,a).
/* 0.049961 0.303191 */ class(A,cochlear_unknown) :- o_ar_c(A,normal), o_ar_u(A,normal), static_normal(A).
/* 0.046515 0.372340 */ class(A,cochlear_unknown) :- o_ar_u(A,normal), static_normal(A).
/* 0.046448 0.643617 */ class(A,cochlear_unknown) :- tymp(A,a).
/* 0.045743 0.345745 */ class(A,cochlear_unknown) :- o_ar_c(A,normal), o_ar_u(A,normal).
/* 0.043047 0.414894 */ class(A,cochlear_unknown) :- o_ar_u(A,normal).
/* 0.036985 0.244681 */ class(A,cochlear_unknown) :- ar_c(A,normal), ar_u(A,normal), o_ar_u(A,normal),
static_normal(A).
/* 0.036485 0.271277 */ class(A,cochlear_unknown) :- ar_u(A,normal), o_ar_u(A,normal), static_normal(A),
tymp(A,a).

real    51:38.6
user    14:22.6
sys         0.1
```

All rules have a high contradiction.

Experiment 18 In the sequential covering approach we are only concerned with clauses that have a very low contradiction value. The next experiment therefore demands that the clauses are satisfied.

```
Tertius% /bin/time tertius -sc -b horn -sat 3/1 audiology_standardized
/* 0.016420 0.000000 */ class(A,cochlear_unknown) :- mod_sn_gt_3k(A), notch_4k(A).
/* 0.010450 0.000000 */ class(A,cochlear_unknown) :- mod_sn_gt_2k(A), notch_4k(A).
/* 0.010450 0.000000 */ class(A,cochlear_unknown) :- mod_sn_gt_3k(A), speech(A,good).
/* 0.010450 0.000000 */ class(A,cochlear_unknown) :- history_roaring(A), m_sn_gt_3k(A).
/* 0.010450 0.000000 */ class(A,cochlear_unknown) :- history_nausea(A), m_sn_gt_3k(A).
/* 0.010450 0.000000 */ class(A,cochlear_unknown) :- o_ar_c(A,elevated), speech(A,good).
.
.
.

real     3:04.9
user     3:03.3
sys         0.3
```

The experiment produced more results than can be reproduced here. This was due to a lot of clauses with the values '/* 0.010450 0.000000 */' and '/* 0.000000 0.000000 */'.

The first rule has the following performance on the reduced data-set:

```
| ?- classify((class(A,cochlear_unknown) :- mod_sn_gt_3k(A), notch_4k(A))).

Correctly classified: 3
Correctly classified ids: [e33,e110,e179]
Positive errors: 0
Negative errors: 0
Examples missing values: 33
```

Note that there are only a total of 36 examples left in the *cochlear unknown* class.

The second rule has the following performance on the reduced data-set:

```
| ?- classify((class(A,cochlear_unknown) :- mod_sn_gt_2k(A), notch_4k(A))).

Correctly classified: 2
Correctly classified ids: [e63,e171]
Positive errors: 0
Negative errors: 0
Examples missing values: 34
```

All the rules with values '/* 0.010450 0.000000 */' cover two examples with no positive errors. As it happens the two rules shown here have disjunct covering sets, but this cannot be expected for all the rules.

These rules give some generalisation on the set of examples, but the generalisation is minimal.

5.3.5 Covering using Background Knowledge

By adding rules to the Tertius background knowledge it is possible to reduce the confirmation of clauses implied by the background knowledge. This can be used to steer the search in Tertius away from already known rules in an attempt to find other complementary rules to cover the complete class.

The three known rules were added as background knowledge by putting them in a background knowledge file and using the switch -bg in the experiments. The -bg switch uses regular Prolog resolution to find if a clause is implied by the background knowledge.

The background knowledge file is given below:

```
Tertius% more audiology_standardized.bg
class(A,cochlear_unknown) :- air(A,mild), ar_c(A,elevated),o_ar_c(A,elevated),tymp(A,a).
class(A,cochlear_unknown) :- bone(A,normal), bone_abnormal(A).
class(A,cochlear_unknown) :- mod_sn_gt_4k(A).
```

A new set of experiments was initiated on the complete data-set.

Experiment 19 A small experiment with only three predicates as the prolog interaction slows down Tertius significantly.

```
Tertius% /bin/time tertius -sc -b horn -bg -SR 0.3 3/1 audiology_standardized
Initiating Prolog
Best and worst current values:
0.420897 0.000000 - 0.236022 0.000000
/* 0.380715 0.000000 */ class(A,cochlear_unknown) v - air(A,mild) v - air(A,normal)
/* 0.310722 0.000000 */ class(A,cochlear_unknown) v - ar_c(A,normal) v - ar_c(A,absent)
/* 0.197602 0.021277 */ class(A,cochlear_unknown) v - ar_c(A,absent) v - o_ar_u(A,normal)
/* 0.204288 0.015957 */ class(A,cochlear_unknown) v - ar_c(A,normal) v - o_ar_u(A,absent)
/* 0.068637 0.000000 */ class(A,cochlear_unknown) v - tymp(A,a) v - tymp(A,as)
/* 0.109072 0.069149 */ class(A,cochlear_unknown) v - ar_u(A,absent) v - tymp(A,a)
/* 0.191425 0.021277 */ class(A,cochlear_unknown) v - ar_c(A,normal) v - o_ar_c(A,absent)
/* 0.238340 0.010638 */ class(A,cochlear_unknown) v - ar_u(A,normal) v - o_ar_c(A,absent)
/* 0.255027 0.000000 */ class(A,cochlear_unknown) v - o_ar_u(A,normal) v - o_ar_u(A,absent)
/* 0.310117 0.000000 */ class(A,cochlear_unknown) v - o_ar_c(A,normal) v - o_ar_c(A,absent)

real     1:08.9
user       16.9
sys         0.2
```

All the satisfied rules reflect trivial mutually exclusive relations between values of attributes. These kind of rules are created when background knowledge is used since the expected value is then calculated on a different basis.

Experiment 20 The trivial rules describing mutual exclusion between values of attributes was added to the background knowledge.

The resulting background file is given below:

```
Tertius% more audiology_standardized.bg
class(A,cochlear_unknown) :- air(A,mild), ar_c(A,elevated),o_ar_c(A,elevated),tymp(A,a).
class(A,cochlear_unknown) :- bone(A,normal), bone_abnormal(A).
class(A,cochlear_unknown) :- mod_sn_gt_4k(A).

class(A,cochlear_unknown) :- air(A,mild),air(A,normal).
class(A,cochlear_unknown) :- ar_c(A,normal),ar_c(A,absent).
class(A,cochlear_unknown) :- tymp(A,a),tymp(A,as)
class(A,cochlear_unknown) :- o_ar_u(A,normal),o_ar_u(A,absent).
class(A,cochlear_unknown) :- o_ar_c(A,normal),o_ar_c(A,absent).


Tertius% /bin/time tertius -sc -b horn -bg -SR 0.3 3/1 audiology_standardized
Initiating Prolog
Best and worst current values:
0.041898 0.000000 - 0.024248 0.000000
/* 0.043472 0.000000 */ class(A,cochlear_unknown) v - ar_u(A,normal) v - ar_u(A,absent)
/* 0.000000 0.075000 */ class(A,cochlear_unknown) v - ar_c(A,normal) v - bone(A,mild)
/* 0.020050 0.005000 */ class(A,cochlear_unknown) v - air_bone_gap(A) v - tymp(A,a)
/* 0.022650 0.000000 */ class(A,cochlear_unknown) v - air_bone_gap(A) v - o_ar_c(A,normal)
/* 0.027698 0.000000 */ class(A,cochlear_unknown) v - ar_c(A,normal) v - ar_c(A,elevated)
/* 0.023900 0.000000 */ class(A,cochlear_unknown) v - speech(A,normal) v - speech(A,good)
/* 0.023136 0.000000 */ class(A,cochlear_unknown) v - speech(A,normal) v - speech(A,very_good)
/* 0.035961 0.000000 */ class(A,cochlear_unknown) v - ar_u(A,normal) v - ar_u(A,elevated)
/* 0.023789 0.000000 */ class(A,cochlear_unknown) v - o_ar_c(A,normal) v - o_ar_c(A,elevated)
/* 0.035789 0.000000 */ class(A,cochlear_unknown) v - o_ar_u(A,normal) v - o_ar_u(A,elevated)

real     1:31.4
user       33.5
sys         0.3
```

All of the satisfied rules are more trivial background knowledge, but rule four tells us something new. Its performance on the complete data-set is:

```
| ?- classify((class(A,cochlear_unknown):-air_bone_gap(A),o_ar_c(A,normal))).

Correctly classified: 0
Correctly classified ids: []
Positive errors: 0
Negative errors: 48
Examples missing values: 0
```

This shows that rule four is just representing an unexpected regularity in the data set and is not in fact a usable classification rule. By adding it to the background knowledge, we tell Tertius not to consider this rule.

Experiment 21 We add the rule from Experiment 20 to the background knowledge together with the complete set of rules to describe mutual exclusion between attribute values. The resulting background file is too long to be presented here.

```
Tertius% /bin/time tertius -sc -b horn -bg -SR 0.3 3/1 audiology_standardized
Initiating Prolog
Best and worst current values:
0.036575 0.000000 - 0.018174 0.000000
/* 0.000000 0.075000 */ class(A,cochlear_unknown) v - ar_c(A,normal) v - bone(A,mild)
/* 0.020050 0.005000 */ class(A,cochlear_unknown) v - air_bone_gap(A) v - tymp(A,a)
/* 0.022022 0.000000 */ class(A,cochlear_unknown) v - air_bone_gap(A) v - ar_u(A,normal)
/* 0.017965 0.005000 */ class(A,cochlear_unknown) v - air(A,normal) v - bone_abnormal(A)
/* 0.021197 0.005000 */ class(A,cochlear_unknown) v - air(A,normal) v - bone(A,mild)
/* 0.003886 0.020000 */ class(A,cochlear_unknown) v - ar_c(A,normal) v - ar_u(A,absent)
/* 0.000000 0.015000 */ class(A,cochlear_unknown) v - bone(A,mild) v - notch_4k(A)
/* 0.015304 0.000000 */ class(A,cochlear_unknown) v - class(A,normal_ear) v - air(A,mild)
/* 0.000000 0.075000 */ class(A,cochlear_unknown) v - bone(A,mild) v - o_ar_u(A,normal)
/* 0.000000 0.040000 */ class(A,cochlear_unknown) v - ar_c(A,absent) v - bone_abnormal(A)

real       40.4
user       33.3
sys         0.3
```

Rules three and eight are satisfied. Their respective performances are:

```
| ?- classify((class(A,cochlear_unknown):-air_bone_gap(A),ar_u(A,normal))).

Correctly classified: 0
Correctly classified ids: []
Positive errors: 0
Negative errors: 48
Examples missing values: 0

| ?- classify((class(A,cochlear_unknown):-class(A,normal_ear),air(A,mild))).

Correctly classified: 0
Correctly classified ids: []
Positive errors: 0
Negative errors: 48
Examples missing values: 0
```

Both of these rules turn out to be regularities in the data-set rather than classification rules.

Experiment 22 Rules four and eight are added to the background knowledge in an attempt to find new classification rules.

```
Tertius% /bin/time tertius -sc -b horn -bg -SR 0.3 3/1 audiology_standardized
Initiating Prolog
Best and worst current values:
0.036575 0.000000 - 0.017239 0.000000
/* 0.000000 0.075000 */ class(A,cochlear_unknown) v - ar_c(A,normal) v - bone(A,mild)
/* 0.020050 0.005000 */ class(A,cochlear_unknown) v - air_bone_gap(A) v - tymp(A,a)
/* 0.017965 0.005000 */ class(A,cochlear_unknown) v - air(A,normal) v - bone_abnormal(A)
/* 0.021197 0.005000 */ class(A,cochlear_unknown) v - air(A,normal) v - bone(A,mild)
/* 0.003886 0.020000 */ class(A,cochlear_unknown) v - ar_c(A,normal) v - ar_u(A,absent)
/* 0.000000 0.015000 */ class(A,cochlear_unknown) v - bone(A,mild) v - notch_4k(A)
/* 0.000000 0.075000 */ class(A,cochlear_unknown) v - bone(A,mild) v - o_ar_u(A,normal)
/* 0.000000 0.040000 */ class(A,cochlear_unknown) v - ar_c(A,absent) v - bone_abnormal(A)
/* 0.012447 0.000000 */ class(A,cochlear_unknown) v - air(A,normal) v - air_bone_gap(A)
/* 0.000000 0.010000 */ class(A,cochlear_unknown) v - mod_sn_gt_3k(A) v - o_ar_c(A,normal)

real       44.7
user       37.1
sys         0.3
```

Rule nine is satisfied. Its performance is:

```
| ?- classify((class(A,cochlear_unknown):-air(A,normal),air_bone_gap(A))).

Correctly classified: 0
Correctly classified ids: []
Positive errors: 0
Negative errors: 48
Examples missing values: 0
```

Another regularity.

5.4 The Cochlear Age Class

Experiment 23 The initial experiment for this class was with five literals and a lower limit on satisfaction of 0.1.

```
Tertius% /bin/time tertius -sc -b horn -sat -n 0.1 5/1 audiology_standardized
Best and worst current values:
0.649853 0.055000 - 0.021565 0.085000
/* 0.649853 0.055000 */ class(A,cochlear_age) :- age_gt_60(A).
/* 0.188654 0.095000 */ class(A,cochlear_age) :- air(A,mild), o_ar_c(A,normal), static_normal(A).
/* 0.160912 0.100000 */ class(A,cochlear_age) :- air(A,mild), ar_c(A,normal), static_normal(A).
/* 0.156391 0.095000 */ class(A,cochlear_age) :- air(A,mild), ar_c(A,normal), o_ar_c(A,normal).
/* 0.109359 0.095000 */ class(A,cochlear_age) :- air(A,mild), ar_u(A,normal), o_ar_c(A,normal).
/* 0.100938 0.095000 */ class(A,cochlear_age) :- air(A,mild), ar_c(A,normal), ar_u(A,normal).
/* 0.088275 0.100000 */ class(A,cochlear_age) :- air(A,mild), ar_c(A,normal), o_ar_u(A,normal).
/* 0.085068 0.095000 */ class(A,cochlear_age) :- air(A,mild), o_ar_c(A,normal), o_ar_u(A,normal).
/* 0.033403 0.080000 */ class(A,cochlear_age) :- air(A,mild), bone_abnormal(A).
/* 0.033403 0.080000 */ class(A,cochlear_age) :- speech(A,good).
/* 0.021565 0.085000 */ class(A,cochlear_age) :- bone_abnormal(A), o_ar_c(A,normal).

real     3:25.9
user     3:13.0
sys         0.0
```

The initial rule has very high satisfaction. It's performance is given below.

```
| ?- classify((class(A,cochlear_age) :- age_gt_60(A))).

Correctly classified: 46
Positive errors: 11
Negative errors: 0
Examples missing values: 0
```

This rule covers 100% of the examples in this class but also includes eleven examples from other classes.

5.4.1 Conclusions from Experiments on the Cochlear Age Class

We have discovered that all members of the *cochlear age* class have an age greater than 60. The two remaining classes that also mention age are *cochlear age and noise* and *cochlear age pluss poss menieres*. The names suggest that these classes are specialisations of the *cochlear age* class, but the respective sizes of the classes are eighteen and one. The total of the two classes, nineteen, exceeds the number of positive mistakes made by the rule discussed in Experiment 23.

5.5 The Cochlear Age and Noise Class

The initial experiment for this class was also five literals and a lower limit on satisfaction of 0.1.

Experiment 24

```
Tertius% /bin/time tertius -sc -b horn -sat -n 0.1 5/1 audiology_standardized
Best and worst current values:
0.132462 0.025000 - 0.022210 0.010000
/* 0.132462 0.025000 */ class(A,cochlear_age_and_noise) :- age_gt_60(A), history_noise(A).
/* 0.086225 0.060000 */ class(A,cochlear_age_and_noise) :- air(A,mild), history_noise(A).
/* 0.038558 0.095000 */ class(A,cochlear_age_and_noise) :- ar_c(A,normal), ar_u(A,normal), history_noise(A).
/* 0.031735 0.095000 */ class(A,cochlear_age_and_noise) :- ar_u(A,normal), history_noise(A),
o_ar_u(A,normal).
/* 0.031735 0.095000 */ class(A,cochlear_age_and_noise) :- history_noise(A), o_ar_c(A,normal),
o_ar_u(A,normal).
/* 0.030639 0.100000 */ class(A,cochlear_age_and_noise) :- ar_c(A,normal), history_noise(A),
o_ar_u(A,normal).
/* 0.030055 0.075000 */ class(A,cochlear_age_and_noise) :- notch_4k(A).
/* 0.027644 0.085000 */ class(A,cochlear_age_and_noise) :- age_gt_60(A), air(A,mild), ar_u(A,normal),
o_ar_u(A,normal).
/* 0.024549 0.100000 */ class(A,cochlear_age_and_noise) :- ar_u(A,normal), history_noise(A),
o_ar_c(A,normal).
/* 0.024549 0.100000 */ class(A,cochlear_age_and_noise) :- age_gt_60(A), air(A,mild), o_ar_c(A,normal),
o_ar_u(A,normal).
/* 0.024319 0.000000 */ class(A,cochlear_age_and_noise) :- age_gt_60(A), air(A,mild), speech(A,very_poor).
/* 0.024319 0.000000 */ class(A,cochlear_age_and_noise) :- history_noise(A), m_m_sn_gt_1k(A).
/* 0.024319 0.000000 */ class(A,cochlear_age_and_noise) :- air(A,mild), o_ar_c(A,normal),
speech(A,very_poor).
/* 0.024319 0.000000 */ class(A,cochlear_age_and_noise) :- age_gt_60(A), m_m_sn_gt_1k(A).
/* 0.022210 0.010000 */ class(A,cochlear_age_and_noise) :- age_gt_60(A), air(A,mild), ar_c(A,absent).
/* 0.022210 0.010000 */ class(A,cochlear_age_and_noise) :- age_gt_60(A), air(A,mild), o_ar_c(A,absent).

real     8:03.0
user     6:01.9
sys         0.2
```

5.6 The Cochlear Age Pluss Poss Menieres Class

Experiment 25 An initial experiment with five predicates and an upper limit on contradiction of 0.1 was tried.

```
Tertius% /bin/time tertius -sc -b horn -sat -n 0.1 5/1 audiology_standardized
/* 0.000903 0.035000 */ class(A,cochlear_age_plus_poss_menieres) :- air(A,severe).
/* 0.000471 0.085000 */ class(A,cochlear_age_plus_poss_menieres) :- speech(A,very_poor).
/* 0.000434 0.095000 */ class(A,cochlear_age_plus_poss_menieres) :- history_dizziness(A).
/* 0.000000 0.000000 */ class(A,cochlear_age_plus_poss_menieres) :- ar_c(A,normal), o_ar_u(A,elevated),
tymp(A,c).
/* 0.000000 0.000000 */ class(A,cochlear_age_plus_poss_menieres) :- ar_c(A,normal), bone(A,mild),
history_fluctuating(A).
/* 0.000000 0.000000 */ class(A,cochlear_age_plus_poss_menieres) :- ar_c(A,normal), bone(A,mild),
mod_sn_gt_2k(A).
/* 0.000000 0.000000 */ class(A,cochlear_age_plus_poss_menieres) :- air(A,mild), ar_u(A,absent),
mod_sn_gt_2k(A).
/* 0.000000 0.000000 */ class(A,cochlear_age_plus_poss_menieres) :- air(A,mild), ar_u(A,absent), m_m_sn(A).
/* 0.000000 0.000000 */ class(A,cochlear_age_plus_poss_menieres) :- air(A,mild), speech(A,very_good),
tymp(A,b).
.
.
.

real    15:34.6
user     6:56.3
sys         0.2
```

The answer were too numerous to present in complete form. This was due to a large number of rules getting the following values:

```
/* 0.000000 0.000000 */
/* 0.000000 0.005000 */
/* 0.000000 0.010000 */
/* 0.000000 0.015000 */
/* 0.000000 0.020000 */
/* 0.000000 0.025000 */
/* 0.000000 0.030000 */
```

The first rule of the rules with the values /* 0.000000 0.000000 */ might be a rule with a non-zero confirmation which has been approximated to zero. To test this the performance is evaluated.

```
| ?- classify((class(A,cochlear_age_plus_poss_menieres) :- ar_c(A,normal),o_ar_u(A,elevated),tymp(A,c))).

Correctly classified: 0
Positive errors: 0
Negative errors: 1
Examples missing values: 0
```

It is obvious that the confirmation for the rules in the group with the values /* 0.000000 0.000000 */ is really zero.

Bibliography

[1] Y. Dimopoulos, S. Džeroski, and A. Kakas. Integrating Explanatory and Descriptive Learning in ILP. In *Proceedings of the 15th International Joint Conference on Artificial Intelligence, IJCAI'97*. Morgan Kaufman, 1997.

[2] K. Eshghi and R. A. Kowalski. Abduction compared with with negation by failure. In *Proceedings of the 6th International Conference on Logic Programming*, pages 234–255, 1989.

[3] M. E. Stickel. A Prolog Technology Theorem Prover: A New Exposition and Implementation in Prolog. Technical Report 464, Artificial Intelligence Centre, SRI International, June 1989.