Efficient Specialization of Relational Concepts

Technical Report PCG-9

Kurt VanLehn

Departments of Psychology and Computer Science
Carnegie-Mellon University
Pittsburgh, PA 15217 U.S.A.

1 July 1987

Running Head: Specialization of Relational Concepts

Acknowledgments

This research was supported by the Information Sciences Division, Office of the Naval Research, under contract number N00014-86-K-0678. Reproduction in whole or in part is permitted for any purpose of the United States Government. Approved for public release; distribution unlimited.

Unclassified
SECURITY CLASSIFICATION OF THIS PAGE

REPORT DOCUMENTATION PAGE

1a. REPORT SECURITY CLASSIFICATION: Unclassified	1b. RESTRICTIVE MARKINGS
2a. SECURITY CLASSIFICATION AUTHORITY	3. DISTRIBUTION / AVAILABILITY OF REPORT: Approved for public release; Distribution unlimited
2b. DECLASSIFICATION / DOWNGRADING SCHEDULE	
4. PERFORMING ORGANIZATION REPORT NUMBER(S): PCG-9	5. MONITORING ORGANIZATION REPORT NUMBER(S): Same as Performing Organization
6a. NAME OF PERFORMING ORGANIZATION: Carnegie-Mellon University; 6b. OFFICE SYMBOL (If applicable)	7a. NAME OF MONITORING ORGANIZATION: Information Sciences Division, Office of Naval Research (Code 1133)
6c. ADDRESS (City, State, and ZIP Code): Departments of Psychology and Computer Science, Pittsburgh, PA 15213	7b. ADDRESS (City, State, and ZIP Code): 800 N. Quincy St., Arlington, VA 22217-5000
8a. NAME OF FUNDING / SPONSORING ORGANIZATION: Same as Monitoring Organization; 8b. OFFICE SYMBOL (If applicable)	9. PROCUREMENT INSTRUMENT IDENTIFICATION NUMBER: N00014-86-K-0678
8c. ADDRESS (City, State, and ZIP Code)	10. SOURCE OF FUNDING NUMBERS p400005ub201/7-4-86

PROGRAM ELEMENT NO	PROJECT NO.	TASK NO.	WORK UNIT ACCESSION NO
N/A	N/A	N/A	N/A

11. TITLE (Include Security Classification): Efficient Specialization of Relational Concepts

12. PERSONAL AUTHOR(S): Kurt VanLehn

13a. TYPE OF REPORT	13b. TIME COVERED	14. DATE OF REPORT (Year, Month, Day)	15. PAGE COUNT
Technical	FROM 86Sep15 TO 91Sep15	87 July 1	7

16. SUPPLEMENTARY NOTATION

17. COSATI CODES			18. SUBJECT TERMS (Continue on reverse if necessary and identify by block number)
FIELD	GROUP	SUB-GROUP	Machine learning, artificial intelligence, version spaces, concept induction

19. ABSTRACT (Continue on reverse if necessary and identify by block number)

See reverse side.

20. DISTRIBUTION / AVAILABILITY OF ABSTRACT: ☐ UNCLASSIFIED/UNLIMITED [✓ SAME AS RPT] ☐ DTIC USERS	21. ABSTRACT SECURITY CLASSIFICATION
22a. NAME OF RESPONSIBLE INDIVIDUAL: Alan Meyrowitz	22b. TELEPHONE (Include Area Code): (202) 696-4302; 22c. OFFICE SYMBOL: 1143

DD FORM 1473, 84 MAR — 83 APR edition may be used until exhausted. All other editions are obsolete.

SECURITY CLASSIFICATION OF THIS PAGE
Unclassified

Abstract

This research note presents an algorithm for a common induction problem, the specialization of overly general concepts. A concept is too general when it matches a negative example. The particular case addressed here assumes that concepts are represented as conjunctions of predicates, that specialization is performed by conjoining predicates to the overly general concept, and that the resulting specializations are to be as general as possible. It is shown that the problem is simple if the concept representation language is propositional, but NP-complete if the language is first-order (i.e., relational). Nonetheless, there exists an algorithm, based on manipulation of bit vectors, that provides good average-case performance.

This research note presents an adequately efficient algorithm for a common type of induction problem, the specialization of overly general concepts. A concept is too general when it matches a negative example. This problem is fairly common in inductive concept formation. For instance, when updating the G set of a version space (Mitchell, 1982), if the current example is negative and some of the elements of the G set match it, then the first step of the update-G process is to specialize those concepts. In algorithms, such as ID3 (Quinlan, 1986), that represent concepts as decision trees, extending a node of the tree is specialization of the concept expressed by the branch of the tree ascending from that node to the root.

This note concerns the special case wherein concepts are represented as a conjunction of atomic predicates. Much work in concept formation has used such conjunctive representations. It is also assumed that specialization is limited to conjoining predicates to the overly general concept. Although there are several other ways to specialize concepts, this is one of the most common ones. For instance, it is the only form of specialization allowed in ID3. Lastly, it is assumed that the induction algorithm should produce concepts that are as general as possible. Again, this is a common assumption in machine learning work.

If the representation is further restricted so that the predicates have no arguments, then the representation is equivalent to a feature set or an attribute-value vector. For this propositional case, there are several well-know algorithms for solving the specialization problem. An easy one to describe is the problem of updating a version space where both concepts and examples are represented as sets of features, and matching is implemented by set inclusion. The algorithm takes three arguments: the overly general concept, C; the negative example, N; and a third concept, P, which is a member of the S set of the version space. The algorithm simply takes each feature in the set difference P-N, and creates a new concept by adding it to C. The algorithm returns this set of concepts, each of which specializes C (i.e., it is a superset of C) and fails to match N (i.e., it is not a subset of N). The basic idea of this algorithm appears in the other propositional algorithms as well, with the modification that the source, P, of positive features may not be a member of the S set but may instead be a positive example (as in Langley's (1987) discrimination algorithm) or even the whole vocabulary of predicates (as in ID3).

The basic idea of the proposition algorithms is to find a set of differences between P and N, and add each to C. Because the new concepts are required to be as general as possible, only "minimal" differences between P and N are added to C. If the representation is propositional, then minimal differences are always exactly one predicate, and never a conjunction of predicates. However, when the algorthm is extended to relational representations (i.e., conjunctions of predicates with variables) a minimal difference between P and N is not necessarily a single relation, but in some cases can be a small conjunction of relations. For instance, consider the following situation, where numbers represent objects and letters are variables:

```
N =  P(1,2) & Q(2,1)
S =  P(y,x) & Q(x,y) & P(x,y)
```

The minimal differences between P and N are subconcepts of P that do not match N. In this case, there are seven such subconcepts, and only three fail to match N:

```
1.  P(y,x)                Matches under the substitution {x/2, y/1}
2.  Q(x,y)                Matches under the substitution {x/2, y/1}
3.  P(x,y)                Matches under the substitution {x/1, y/2}
4.  P(y,x) & Q(x,y)       Matches under the substitution {x/2, y/1}
5.  P(y,x) & P(x,y)                    No match
6.  P(x,y) & Q(x,y)                    No match
7.  P(y,x) & Q(x,y) & P(x,y)           No match.
```

Given some overly general concept C that matches N, any of the last three could be conjoined to C in order to specialize it. However, the last subconcept is not a minimal difference, because it properly includes other subconcepts that do not match N (i.e., subconcepts 5 and 6), so only subconcepts 5 and 6 should be added to C. This produces two new maximally general concepts, assuming that C was maximally general before N was received. The main point of this example is that the smallest concepts that serve to specialize C are conjunctions of two predicates.

Incidentally, conjoining a subconcept to C can be nontrivial. For instance, if C is `P(u,v) & Q(v,u)`, then conjoining `P(x,y) & Q(x,y)` to it should yield both `P(u,v) & Q(v,u) & Q(u,v)` and `P(u,v) & P(v,u) & Q(v,u)`. Fortunately, this problem is similar to the Update-S problem, and nearly the same algorithm can be used for it.

A direct, but inefficient way to generate minimal differences between P and N is to search for them by chaining together predicates that share variables. At one time, the procedure-learning program Sierra (VanLehn, 1987) used this technique for updating its version spaces. It often took 30 hours or more just to handle a few negative examples. With some help from Johan de Kleer, a much faster algorithm was invented. Now Sierra completes the same calculations in a minute or two.

The key idea is to convert the problem of finding minimal differences into a set covering problem. The first step is to enumerate all substitutions of objects in N for variables in P. For example, if there were 5 variables in P and 7 objects in N, then there are either 7^5 possible substitutions when distinct variables can be bound to the same object, or 7*6*5*4*3 substitutions when distinct variables must be bound to distinct objects.[1] Having enumerated the substitutions, each predicate in P is assigned a bit vector. The bit vector has one bit for each substitution. If the predicate is in N under a given substitution, then the bit is zero. If the predicate is not in N, then the bit is one. If any predicate has a bit vector that is all ones, then it is not in N under any substitution, so it is a minimal difference between P and N. Assuming there are no such predicates, the algorithm must to find a small set of predicates such that the union of their bit vectors is all ones.

Attaching bit vectors to predicates converts the minimal difference problem into a well-known problem, the set covering problem: Given a target set and a collection of subsets of it, find a cover for the target set, where a *cover* is a set of subsets such that the union of those subsets equals the target set. There are several different versions of the set covering problems, depending on the kind of cover desired. An *irredundant* cover is a cover that is not properly included in any other cover (i.e., none of its subsets is redundant in that it can be removed from the cover without affecting the cover's equality to the target set). Irredundant covers correspond to minimal differences between P and N, which in turn are the differences that lead to maximally general specializations of C. So for most applications, the covering algorithm desired will be one that finds irredundant covers. Another possibility is an algorithm for calculating minimal covers, which are covers with the fewest number of elements. This would cause the specialization algorithm to generate some, but not necessarily all, of the maximally general concepts.

The most straightforward way to generate irredundant covers is breadth-first search. However, this approach is very space inefficient. Sierra uses an algorithm from Wells (1971, section 6.4.3), which is based on depth-first search. The trick is to prune the search whenever adding a new subset to the cover causes the cover to become redundant. Well's algorithm is presented in table 1. The second clause of the Cond implements the search pruning.

[1]Sierra's concept representation enforces additional constraints that reduces the number of substitutions still further. This reduction is crucial, because Sierra's concepts usually had between 10 and 50 variables. In retrospect, the same reduction could be achieved more elegantly by assigning types to variables and objects, then enumerating only substitutions that paired objects and variables of the same type.

Table 1: An algorithm for finding irredundant covers

```
(Defun FindCover (Cover Covered Duplicates Usable)
(And Usable
 (Let
  ((Candidate (Car Usable))
   (NewCover (Cons Candidate Cover))
   (NewCovered (Union Covered (BitVector Candidate)))
   (NewDuplicates
     (Union Duplicates (Intersect Covered (BitVector Candidate)))
   (NewUsable (Cdr Usable))
  (Cond
   ((For X in NewCover thereis
        (SubsetP (BitVector X) NewDuplicates))
    (FindCover Cover Covered Duplicates NewUsable))
   ((TotalSetP NewCovered)
    (Cons NewCover
      (FindCover Cover Covered Duplicates NewUsable))
   (T
    (Append
      (FindCover NewCover NewCovered NewDuplicates NewUsable)
      (FindCover Cover Covered Duplicates NewUsable)))))))
```

FindCover returns a list of irredundant covers. Cover is a list of predicates paired with their bit vectors. Covered is the bit vector for the current coverage of the cover. Duplicates is the bit vector for the substitutions that are covered twice by members of the cover. Usable is a list of potential candidates for adding to the cover.

Table 2 presents estimates for the time and space complexity of this depth-first algorithm, the breadth-first algorithm, and the algorithm based on chaining through variables. Note that all the algorithms are exponential. This is inevitable, because set covering is an NP-complete problem (Aho, Hopcroft & Ullman, 1974, theorem 10.2.9). However, the exponents tends to be small in typical applications. For Sierra, p and n are usually about 100, and m is usually 1 or 2.

Table 2: Complexity estimates for three minimal difference algorithms

Algorithm	Time	Space
Depth-first	$(p \log p)^m$	$p \log p$
Breadth-first	p^m	p^m
Chaining	$p^m n^v$	p^m

Where: p is the number of predicates in P, n is the number of predicates in N, m is the average number of predicates in a minimal difference, and v is the average number of variables in a minimal difference.

As mentioned earlier, this algorithm was used with success as part of a version space maintenance module. It would be equally useful in machine learning programs, such as AQ11 (Michalski & Larson, 1978), that employ version-space-like techniques as components. It would probably be useful for programs, such as ID3 (Quinlan, 1986) and PRISM (Langley, 1987), that induce decision trees. All these programs were initially developed with propositional concept representations. The algorithm described above extends them for use with relational concept representations.

References

Aho, A.V., Hopcroft, J.E. & Ullman, J.D. (1974). *The Design and Analysis of Computer Algorithms.* Reading, MA: Addison-Wesley.

Langley, P. (1987). A general theory of discrimination learning. In Klahr, D., Langley, P. & Neches, R. (Ed.), *Production System Models of Learning and Development.* Cambridge, MA: MIT Press.

Michalski, R.S. & Larson, J.B. (1978). *Selection of most representative training examples and incremental generation of VL1 hypotheses: the underlying methodology and the description of programs ESEL and AQ11* (Technical Report 867). Computer Science Department, University of Illinois.

Mitchell, T.M. (1982). Generalization as search. *Artificial Intelligence*, *18*, 203-226.

Quinlan, J. R. (1986). The effect of noise on concept learning. In R. S. Michalski, J. G. Carbonell, & T. M. Mitchell (Ed.), *Machine Learning: An Artificial Intelligence Approach. Volume II.* Los Altos, CA: Morgan Kaufman.

VanLehn, K. (1987). Learning one subprocedure per lesson. *Artificial Intelligence*, *31*(1), 1-40.

Wells, M.B. (1971). *Elements of Combinatorial Computing.* New York, NY: Pergamon Press.