Learning Relations by Pathfinding

Bradley L. Richards
Dept. of Computer Sciences
University of Texas at Austin
Austin, Texas 78712
bradley@cs.utexas.edu

Raymond J. Mooney
Dept. of Computer Sciences
University of Texas, Austin
Austin, Texas 78712
mooney@cs.utexas.edu

Abstract

First-order learning systems (e.g., FOIL, FOCL, FORTE) generally rely on hill-climbing heuristics in order to avoid the combinatorial explosion inherent in learning first-order concepts. However, hill-climbing leaves these systems vulnerable to local maxima and local plateaus. We present a method, called relational pathfinding, which has proven highly effective in escaping local maxima and crossing local plateaus. We present our algorithm and provide learning results in two domains: family relationships and qualitative model building.

1 Introduction

Many recent learning systems for first-order Horn clauses, such as FOIL, FOCL, and Forte ([Quinlan, 1990], [Pazzani, Brunk, and Silverstein, 1991], and [Richards and Mooney, 1991]) employ hill-climbing to learn clauses one literal at a time. One of the problems faced by such hill-climbing systems is what we call the *local plateau* problem (see Figure 1). This arises when, in order to improve the classification accuracy of a rule, we must add two or more literals simultaneously. There may be many single relations which do not decrease the category accuracy, but which of these will lead to an eventual improvement?

This paper presents a new method for dealing with this problem: *relational pathfinding*. This approach is based on the assumption that, in relational domains, there usually exists a fixed-length path of relations linking the set of terms satisfying the goal concept. For example, in a family domain, there is always a fixed-length path between a grandparent and a grandchild; this path consists of two parent relations. We present a method for finding these paths, and we demonstrate this method in three domains: family relationships, qualitative model building, and logic programming.

The remainder of this paper is organized as follows: Section 2 gives a brief background on first-order learning systems. Section 3 describes our algorithm for relational pathfinding, illustrating it in the domain of family relationships. Section 4 presents results for the more complex problem of deriving qualitative models of dynamic systems from observed behaviors. Section 5 gives results in the domain of logic programming. Section 6 compares relational pathfinding to related work in the field and gives our recommendations for future work. Section 7 summarizes and concludes the paper.

2 First-Order Learning

There are several first-order learning systems capable of pure inductive learning. Among these are FOIL [Quinlan, 1990], FOCL [Pazzani, Brunk, and Silverstein, 1991], GOLEM [Muggleton and Feng, 1990], and Forte [Richards and Mooney, 1991]. FOIL and GOLEM are designed as pure inductive learners. FOCL is an enhancement of FOIL which, if given an initial theory, uses it to provide hints to the learning process. Forte is designed primarily as a theory-revision system but, by revising an empty theory, can perform pure inductive learning. All of these systems learn first-order categorization rules, and all but GOLEM learn "top-down" by specialization, adding a single literal at a time to their rules. The top-down systems are vulnerable to local plateaus and local maxima. Relational pathfinding is designed to aid this type of system. GOLEM works "bottom-up" by generalization, and would not be helped by this technique.

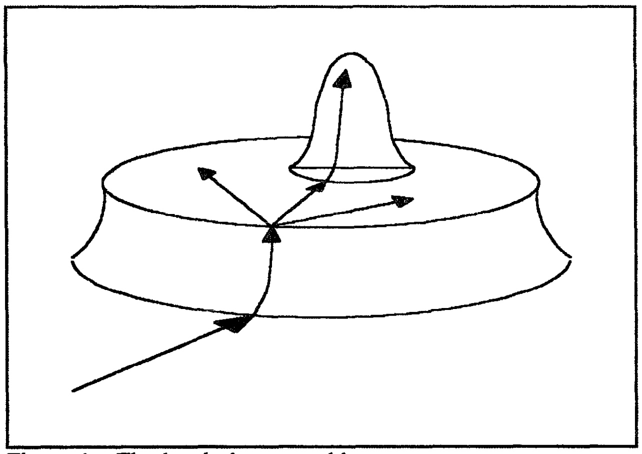
Figure 1. The local plateau problem.

Top-down systems work by learning Horn clauses one at a time until all positive examples are covered. Each clause is generated by adding one literal at a time using a simple hill-climbing technique. At each step, instantiations of each predicate in the data are tested for their ability to discriminate the remaining positive and negative examples, and the best discriminator is added to the clause. When all negative examples have been excluded, the clause is complete. Additional clauses are formed to cover any remaining positive examples. Of course, this hill-climbing approach is vulnerable to local plateaus and local maxima. Relational pathfinding is an attempt to avoid these locality problems.

3 Relational Pathfinding

The idea of pathfinding in a relational domain is to view the domain as a (possibly infinite) graph of constants linked by the relations which hold between the constants. For example, a portion of the data in Hinton's family domain [Hinton, 1986] is shown in Figure 2. This domain is particularly easy to visualize since all relations are binary.

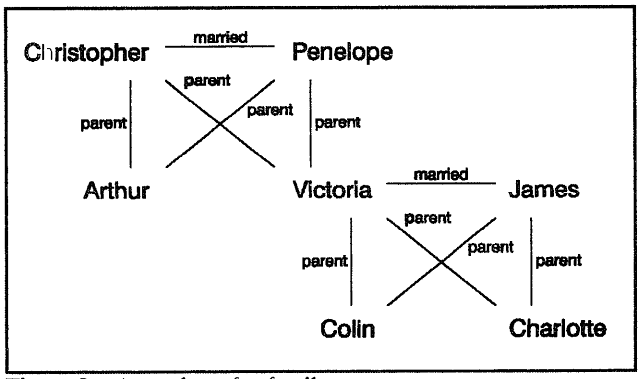

Figure 2. A portion of a family tree.

We can see the local plateau problem by trying to define the grandparent relation using only the instances

positive: **grandparent(Christopher, Colin)**
negative: **grandparent(Christopher, Arthur)**

There is no single antecedent which will discriminate between these instances. Both Colin and Arthur have parents, neither has children, and neither is married[1]. Also, determinate literals do not help in this example; The only determinate literal available is **married(Christopher, Penelope)**, since all parents have two children and all children have two parents. In order to create a correct theory, we must simultaneously add both of the required parent relations, i.e.,

$$\mathbf{grandparent}(x, y) \leftarrow \mathbf{parent}(x, z) \land \mathbf{parent}(z, y).$$

Relational pathfinding is based on the assumption that, in most relational domains, important concepts will be represented by a small number of fixed paths among the constants defining a positive instance. For example, the grandparent relation is defined by a single fixed path consisting of two parent relations.

```
instantiate rule with a positive instance
find isolated sub-graphs
for each sub-graph
  constants become initial end-values
end for
repeat
  for each sub-graph
    expand paths by one relation in all possible ways
    remove paths with previously seen end-values
  end for
until intersection or resource-bound exceeded
if we have intersections
  for each intersection
    add path-relations to original rule
    if the new rule contains new singletons
      add relations using the singletons
      if we eliminated all singletons
        keep the expanded rule
      else
        discard the rule
      end if
    end if
    replace constants with variables
  end for
  select most accurate rule
end if
if negatives are still provable
  use normal specialization to finish the rule
end if
```

Algorithm 1. Overview of the relational pathfinding algorithm.

Relational pathfinding can be tried anytime a clause needs to be specialized and does not have relational paths joining all of its variables. If, after pathfinding, the rule is still too general, we do further specialization using a standard FOIL-like technique. This arises, for example, when a rule requires non-relational antecedents.

Relational pathfinding finds paths by successive expansion around the nodes associated with the constants in a positive example, in a manner reminiscent of Quillian's spreading activation [Quillian, 1968]. We arbitrarily choose a positive instance and use it to instantiate the initial rule. The constants in the instantiated rule are nodes in the domain graph, possibly connected by antecedents in the rule. We then identify isolated subgraphs among these constants; if the initial rule contains no antecedents, then each constant forms a singular subgraph.

We view a sub-graph as a nexus from which we explore the surrounding portion of the domain graph. Each exploration which leads to a new node in the domain graph is a path, and the value of the node it has reached is the path's end-value. Initially, constant in a sub-graph is the end-value of a path of length zero.

Taking each subgraph in turn, we find all new constants which can be reached by extending any path with any defined relation. These constants form the new set of path end-values for the subgraph. We check this set against the sets of end-values for all other subgraphs, looking for an intersection. If we do not find an intersection, we expand the next node. This process continues until we either find an intersection or exceed a preset resource bound.

When we find an intersection, we add the relations in the intersecting paths to the original instantiated rule. If the new relations have introduced new constants that appear only once, we complete the rule by adding relations which hold between these singletons and other constants in the rule. If we are unable to use all such singletons, the rule is rejected. Finally, we replace all constants with unique variables to produce the final, specialized theory clause. If we simultaneously discover several intersections, we develop clauses for all of them and choose the one which provides the best accuracy on the training set.

While the pathfinding algorithm potentially amounts to exhaustive exponential search, it is generally successful for two reasons. First, by searching from all nodes simultaneously, we greatly reduce the total number of paths explored before we reach an intersection. Second, most meaningful relations are defined by short paths, which inherently limits the depth of search. However, a practical implementation of this method includes a resource bound.

As an example, suppose we want to learn the relationship *uncle*[2], given an initially empty rule and the positive instance **uncle(Arthur, Charlotte)**. The process is illustrated in Figure 3. We begin by exploring paths from the node labelled **Arthur**, which leads us to the new nodes **Christopher** and **Penelope**. We then expand from the node labelled **Charlotte**, leading to the nodes **Victoria** and **James**. At this point we still do not have an intersection, so we lengthen all paths originating from node **Arthur**. We eliminate any end-values which we have already used (and which, therefore, do not give us an intersection). This leaves us with only one path end-value: **Victoria**. Since **Victoria** is also an end-value of one of the paths originating from **Charlotte**, we recognize an intersection.

There are two paths leading from **Arthur** to **Victoria**, but in this case they are identical (merely leading through different grandparents). If we had found several paths, we would select the one providing the best overall accuracy. The final path is

$$\mathbf{uncle}(x, y) \leftarrow \mathbf{parent}(z, x), \mathbf{parent}(z, w), \mathbf{parent}(w, y)$$

The literal **male(x)**, which is required to complete this rule, is not a relation and is therefore added by ordinary specialization.

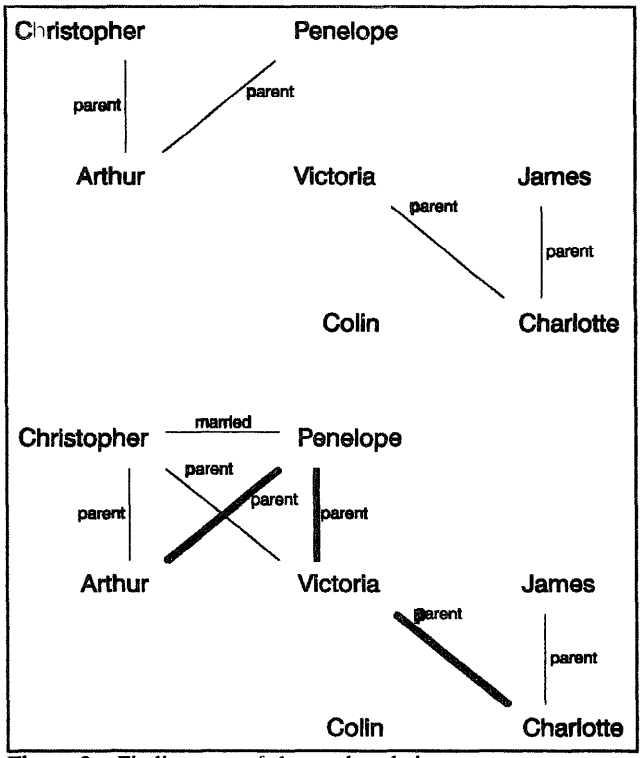

Figure 3. Finding one of the uncle relations.

To test the hypothesis that relational pathfinding improves the accuracy of an empirical learning system, we ran Forte [Richards and Mooney, 1991] on the family data used in [Hinton, 1986] and [Quinlan, 1990], both with and without relational pathfinding. Training sets were randomly selected from a set containing all 112 positive instances and 272 "near-miss" negative instances, with the remainder serving as the test set. The data includes

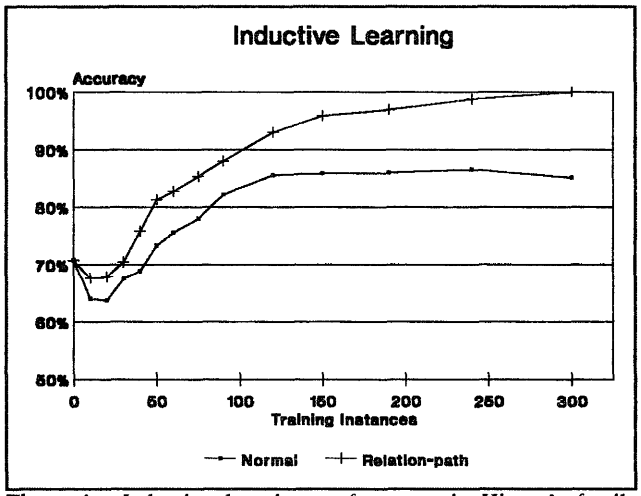

Figure 4. Inductive learning performance in Hinton's family domain, averaged over 20 trials per data point.

twelve family-relation concepts, so a training set of 60 instances includes an average of 5 instances for each concept. The results (see Figure 4) show a significant gain in learning performance for any size of training set. Above 120 instances, learning without relational pathfinding levels out while relational pathfinding leads to a complete and correct theory. By comparison, FOIL [Quinlan, 1990] achieved a maximum accuracy of 97.5% on this data, using the equivalent of 2400 training instances.

4 Qualitative Modelling

The family domain is ideal for demonstrating our approach since the relations are binary and form a fairly simple graph. However, relational pathfinding works equally well in more complex domains, as long as concepts can be viewed as fixed-length paths joining nodes in a graph. An example of this is the domain of qualitative modelling. Qualitative model building is a complex domain for two reasons. First, not all relations are binary. Second, values associated with new variables are not simple atoms, but full behavioral descriptions which may be only partially instantiated.

One limitation of systems which derive qualitative models from behaviors is that there has been no satisfactory method of identifying missing variables. For example, if we seek to model the cascaded tank system shown in Figure 6, a complete model (see Figure 5) must include variables for inflow, outflow, amount, and net-flow for each tank. However, an observer is likely to measure only the externally visible variables, and to thereby omit the net-flow variables.

We work with qualitative models as defined by QSIM [Kuipers, 1986]. A QSIM model is a set of variables and a conjunction of constraints on those variables. Typical constraints (relations) include *derivative* (d/dt), *add* (+), *M*+, and *M*-. The constraints describe qualitative relationships among the variables over time. For example, the M+ constraint states that two variables are related by a strictly monotonically increasing function.

Relational pathfinding provides a way to introduce missing variables into a model. Qualitative constraints impose restrictions on the values of their arguments; when used by relational pathfinding to generate end-values of paths, they partially instantiate the values to enforce their restrictions. Thus, the values are partial behavioral descriptions of hypothetical system variables. Two paths intersect when the restrictions on their end-values are consistent (i.e., when they can be unified). When the new rule is generalized, the intersection value becomes a new variable in the model.

Forte, using relational pathfinding, is able to create correct models when one or more system variables have been omitted from the input behavioral descriptions. Consider a system of two cascaded tanks, A and B, where the inflow to tank A is constant, and the outflow from tank A provides the inflow to tank B. For this system, given

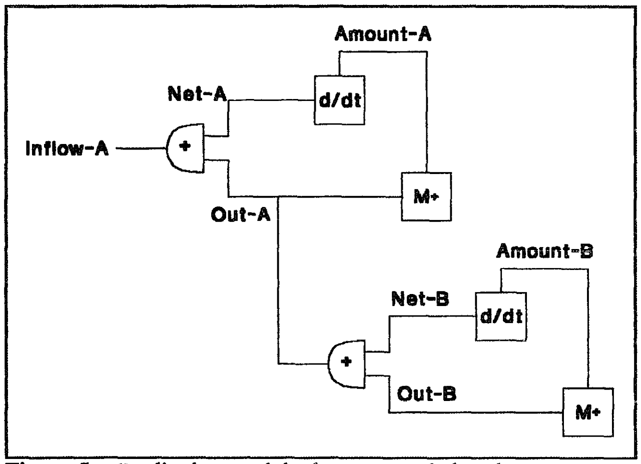

Figure 5. Qualitative model of two cascaded tanks.

Inflow A:	⊖+	⊖+	⊖+	⊖+	⊖+
Amount A:	↑0	↑+	⊖+	⊖+	⊖+
Outflow A:	↑0	↑+	⊖+	⊖+	⊖+
Amount B:	⊖0	↑+	↑+	↑+	⊖+
Outflow B:	⊖0	↑+	↑+	↑+	⊖+
Time:	→ → → → → → → → →				

Table 1. Single behavior of two cascaded tanks reaching equilibrium.

the single behavior shown in Table 1[3] (which omits any mention of the net-flow variables), Forte automatically produces the model

```
model(Amount_A, Amount_B, Inflow_A, Out_A, Out_B)
    m_plus(Amount_B, Out_B),
    m_plus(Amount_A, Out_A),
    m_minus(Out_A, Net_A),
    m_minus(Amount_A, Net_A),
    derivative(Amount_B, Net_B),
    derivative(Amount_A, Net_A),
    add(Net_B, Out_B, Out_A),
    add(Net_A, Out_A, Inflow_A),
    constant(Inflow_A).
```

This model is the same as the correct model shown in Figure 5, with the addition of two redundant M-constraints.

5 Logic Programming

Any Horn clause theory can be viewed as a logic program. However, in the domains of family relationships and qualitative models, theory are not recursive. If we wish to synthesize predicates for relations like append, reverse, or sort, we must be able to produce recursive clauses.

The only new issue we must consider is how to evaluate recursive calls. This is a problem because we are in the process of modifying the very predicate we wish to evaluate. Our answer is to use the positive instances in the training set as an extensional definition of the predicate. The means that, when synthesizing logic programs, we expect the training set to contain a complete set of positive examples (e.g., our training set for list reversal contains all instances for lists of length three or less, using up to three distinct atoms). Using the training set to evaluate recursive calls allows relational pathfinding to develop recursive clauses.

Consider the predicate **merge_sort**. Suppose we already have definitions for split and merge, and we have already learned the base case. If relational pathfinding uses the instance

merge_sort([4,3,2,1],[1,2,3,4])

it will develop the ground path shown in Figure 6. This path contains two singleton constants: [4,2] and [2,4]. However, we are able to eliminate both singletons by adding the relation **merge_sort([4,2], [2,4])**. Replacing the constants with variables produces the final, correct rule

```
merge_sort(A, B) :-
    split(A, C, D),
    merge_sort(C, E),
    merge_sort(D, F),
    merge(E, F, B).
```

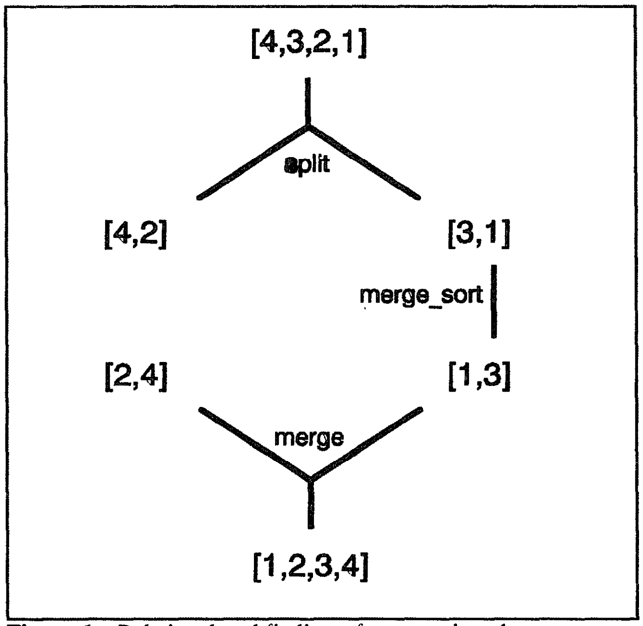

Figure 6. Relational pathfinding of a recursive clause.

6 Related Work

Perhaps the earliest work on overcoming locality problems in first-order learning was [Vere, 1977], which introduced the idea of "association chains" composed of determinate binary relations. This idea appeared ahead of its time, since there were no first-order learning systems which could take advantage of it, but it foreshadows both determinate literals (see below) and relational pathfinding.

Adding ***determinate literals*** to a rule is an idea used by Muggleton and Feng in GOLEM, and later added to FOIL [Quinlan, 1991]. Determinate literals are literals which, given the bindings derivable from a positive instance and prior literals, have only one possible ground instantiation. After adding all possible determinate literals (up to a predefined depth limit), learning proceeds normally. If, by adding the determinate literals, we added all but one of the relations necessary to cross any local plateaus or escape any local maxima, we will be able to learn a correct rule. When learning is complete, excess determinate literals are discarded.

In theory, any chain of determinate literals can be found by relational pathfinding. However, in domains where relational paths are long, using determinate literals may be more efficient. In other domains, where we cannot find a chain of determinate literals to cross a local plateau or escape a local maximum, relational pathfinding will have the advantage.

Another method for dealing with locality problems is that of ***relational clichés***, presented in [Silverstein and Pazzani, 1991]. In this approach, the learning system has a predefined set of templates describing combinations of relations which often appear together. For example, the predicate **part-of(*x*, *y*)** generally appears along with a definition for part *y*. The learning system can add entire templates rather than single relations, thus avoiding some of the local maxima or plateaus which it might otherwise encounter. The method of relational pathfinding is more general than relational clichés since it does not depend on predefined templates; however, if the predefined templates are adequate for the learning domain, using relational clichés will be more efficient.

A domain-specific system for building qualitative models is MISQ, presented in [Richards, Kraan, and Kuipers, 1992]. To date, the only general purpose learning system applied to this problem is GOLEM [Bratko, Muggleton, and Varšek, 1991]. GOLEM's performance in this domain is limited since it requires negative examples and its definitions of qualitative constraints are incomplete (e.g., it ignores corresponding values). Both MISQ and Forte can build qualitative models using only positive information. Relational pathfinding allows new model variables to be introduced in a natural way.

7 Conclusion

In this paper we presented a new method, ***relational pathfinding***, which helps first-order learning systems escape local maxima and cross local plateaus. It is similar to the approaches of determinate literals and relational clichés in that all of these approaches add multiple relations to a rule. Although each of these methods addresses the same problem of escaping local maxima, they are useful in different circumstances. Relational pathfinding is more general than either of these methods, but potentially less efficient.

We presented results in three domains in which relational pathfinding has proven useful. When learning family relationships, it provides a substantial performance advantage, requiring many fewer examples to learn accurate definitions. In qualitative modelling, it allows a system to learn an accurate model even if behavioral information on some variables is missing. And in logic programming, it provides an effective way to learn recursive clauses. In all of these domains, relational pathfinding allows the system to overcome the problem of local maxima and local plateaus while still limiting combinatorially explosive search.

Acknowledgements

This research was supported by the Air Force Institute of Technology faculty preparation program, by the NASA Ames Research Center under grant NCC 2-629, and by the National Science Foundation under grant IRI-9102926. Also, our thanks to Michael Pazzani for his helpful comments on an earlier draft of this paper.

References

I. Bratko, S. Muggleton, and A. Varšek, "Learning Qualitative Models of Dynamic Systems," *Proceedings of the Eighth International Workshop on Machine Learning*, pp. 385-388, 1991.

G. E. Hinton, "Learning Distributed Representations of Concepts," *Proceedings of the Eighth Annual Conference of the Cognitive Science Society*, 1986.

B. Kuipers, "Qualitative Simulation," *Artificial Intelligence*, 29:289-338, 1986.

S. Muggleton and C. Feng, "Efficient induction of logic programs," *Proceedings of the First Conference on Algorithmic Learning Theory*, 1990.

M. J. Pazzani, C. A. Brunk, and G. Silverstein, "A knowledge-intensive Approach to Relational Concept Learning," *Proceedings of the Eighth International Workshop on Machine Learning*, pp. 432-436, 1991.

M. R. Quillian, "Semantic Memory," *Semantic Information Processing*, MIT Press, pp. 227-270, 1968.

J. R. Quinlan, "Learning Logical Definitions from Relations," *Machine Learning*, 5:239-266, 1990.

J. R. Quinlan, "Determinate Literals in Inductive Logic Programming," *Proceedings of the Eighth International Workshop on Machine Learning*, pp. 442-446, 1991.

B. L. Richards, I. Kraan, and B. J. Kuipers, "Automatic Abduction of Qualitative Models," *Proceedings of the Tenth National Conference on Artificial Intelligence* (AAAI 1992), 1992.

B. L. Richards and R. J. Mooney, "First-Order Theory Revision," *Proceedings of the Eighth International Workshop on Machine Learning*, pp. 447-451, 1991.

G. Silverstein and M. J. Pazzani, "Relational clichés: Constraining constructive induction during relational learning," *Proceedings of the Eighth International Workshop on Machine Learning*, pp. 203-207, 1991.

S. A. Vere, "Induction of Relational Productions in the Presence of Background Information," *Proceedings of the Fifth International Joint Conference on Artificial Intelligence* (IJCAI 1977), pp. 349-355, 1977.

Notes

[1]For this simple example we disregard the possibility of using negation, i.e., grandparent(x, y) $\leftarrow \neg$parent(x, y).

[2]The uncle relationship is completely defined by two paths, one of length 3 and one of length 4. We illustrate here the process of finding the shorter of these two paths.

[3]For simplicity, we show only the direction-of-change and the sign of each variable at each time point. The complete behavior includes qualitative magnitudes and dimensional information. Direction-of-change can be increasing ($\uparrow$), decreasing ($\downarrow$), or steady ($\ominus$). Sign is plus (+), minus (-), or zero (0).