In *Intelligent Systems*, E. A. Yfantis (ed), vol. 1, pp. 717-625, Kluwer Academic Publishers, 1995.

Efficient Construction of Networks for Learned Representations with General to Specific Relationships

Cory Barker and Tony Martinez
Brigham Young University, Provo, Utah 84602
cory@axon.cs.byu.edu, martinez@cs.byu.edu

Abstract

Machine learning systems often represent concepts or rules as sets of attribute-value pairs. Many learning algorithms generalize or specialize these concept representations by removing or adding pairs. Thus concepts are created that have general to specific relationships. This paper presents algorithms to connect concepts into a network based on their general to specific relationships. Since any concept can access related concepts quickly, the resulting structure allows increased efficiency in learning and reasoning. The time complexity of one set of learning models improves from O(n log n) to O(log n) (where n is the number of nodes) when using the general to specific structure.

1. Introduction

Many machine learning algorithms learn by either generalizing or specializing existing representations [Michalski 1983, Mitchell 1982, Hayes-Roth 1978, Vere 1975, Barker 1993]. The learning process typically involves a search for the best new representation. The search is potentially time consuming and can be benefited by using parallel architectures for implementation. This paper presents a set of algorithms to create a structure that connects rules or concepts based on their general to specific relationships. The resulting natural structure has potential to increase performance of learning algorithms.

Learning systems often represent concepts, rules, or features as sets of attribute-value pairs or input-value pairs. Such rules cover subsets of the input space. Each input-value pair (i, x) contained in the representation of a rule restricts the rule to match only those input points where input $i = x$. Thus, rules with fewer input-value pairs are more general while rules containing more input-value pairs are more specific.

If a rule A has input-value pair (i, x) and rule B has input-value pair (i, y) such that $x \neq y$, then A and B are said to be *discriminated*. The sets of input points covered by A and B are disjoint. If two rules are not discriminated they are said to *overlap*.

Rules can be generalized by removing input-value pairs and specialized by adding input-value pairs. The specialization S obtained by adding one or more input-value pairs to a rule R matches a subset of the points in the input space that are matched by R. S is said to be *more-specific-than* R [Mitchell 1982].

The more-specific-than relation can be computed by comparing the input-value pairs that define two rules. Given two rules A and B with input-value sets p and q, respectively, A is more-specific-than B if-and-only-if $p \supset q$.

The more-specific-than relation defines a *partial order* on the set of rules. A partial order gives rise to a *semi-lattice*. The algorithms described in this work create this lattice structure. The structure is a digraph whose vertices or nodes are rules and which has an edge or connection from rule r to rule s if and only if s is more-specific-than r. The resulting network structure can be implemented using either software or hardware.

Using a structure that exploits the general-specific relationship between rules allows increased efficiency in learning and reasoning. One family of learning models [Barker 1994] incorporates a new training example in time O(n log n) (where n is the number of nodes) without using the structure. The complexity improves to O(log n) when the structure is used. The contribution of this paper is to present efficient algorithms for construction of the network.

2. General-Specific (GS) architecture

A network using the GS topology is a set of nodes N and a set of edges E. Let A, B, and C be nodes from N. Let r_a, r_b, and r_c be the rules for nodes A, B, and C, respectively. The set of nodes N always contains a node R that is the root node. The root node contains the rule that is fully general; it matches all points in the input space. Suppose the network is defined such that there is an edge from A to B if-and-only-if r_a is a generalization of r_b.

An example of such a network is shown in « GS». The example has three inputs {*Wind Speed, Wind Direction, and Visibility*} (abbreviated S, D, and V, respectively). *Wind Speed* has three possible values {*Low, Medium, High*} (abbreviated L, M, and H, respectively). *Wind Direction* has two values {*Head, Tail*} (abbreviated H and T, respectively). *Visibility* has two values {*Clear, Cloudy*} (abbreviated Clr and Cld, respectively). The root node is connected to all other nodes since the root is a generalization of every possible rule. Node 1 is a generalization of both Nodes 2 and 3. Note that the connections between the root and Nodes 2 and 3 are essentially redundant. These connections can be obtained by computing the transitive closure of other connections. The number of connections required in the network can be reduced by removing these redundant connections.

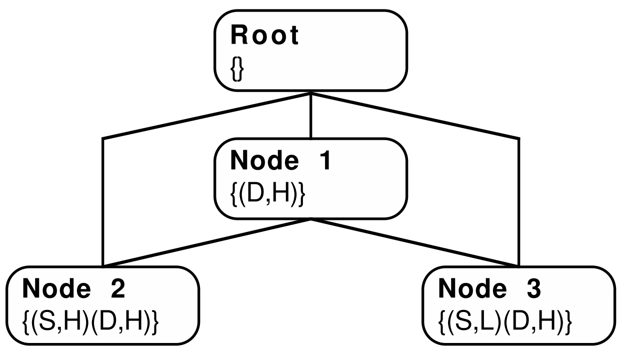

Figure 1. General/Specific Architecture

The definition for edges is modified to remove edges that are redundant. An edge connects nodes A and B if-and-only-if r_a is a generalization of r_b and there is not some other node C, such that r_c is a generalization of r_b and r_c is a specialization of r_a. In other words, two nodes are connected if the rule in one node is either the most specific generalization or the most general specialization of the rule in the other node.

An example of a network without redundant connections is shown in « GS2». Note that a node can still have multiple generalizations. Node 3 is connected to Nodes 1 and 2 since both nodes are generalizations of Node 3 but Nodes 1 and 2 are not generalizations or specializations of each other. The resulting network is thus a directed acyclic graph (DAG) rather than a tree.

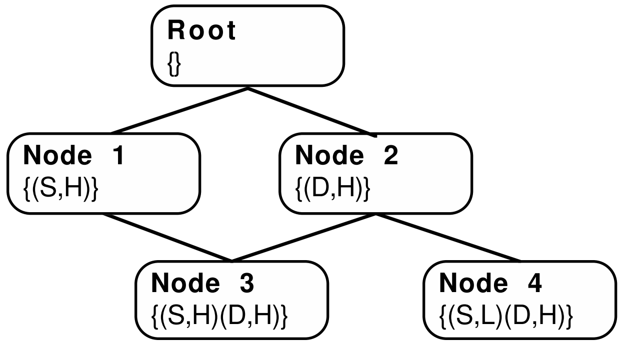

Figure 2. General/Specific Architecture

3. Adding Nodes

When adding a new node to a GS network the new node must be connected to existing nodes that are the most specific generalizations of the new node. The new node must also be connected to existing nodes that are the most general specializations of the new node. It is desirable to reduce the number of existing nodes that must be searched in order to connect a new node. By using information that is already known about a new node the new node can be connected in a more efficient manner.

New nodes that have no known relationship with other nodes in the network are linked to their generalizations by starting at the root node. The procedure is described in Section 3.1. Then, using one of the generalizations *N* that was linked to the new node, specializations of the new node can be found efficiently by starting at node *N* in the network. This process is described in Section 3.2. When a node is known to be a generalization of an existing network node *N*, generalizations of the new node can be found efficiently by starting at node *N* in the network. This process is not described in this work but is essentially the opposite of that described in Section 3.2. Once a node is linked into the network, other links previously existing in the network may have become redundant. Section 3.3 shows how these links are removed.

3.1. Finding Generalization Links

The routine *LinkRoot* in « LinkRoot» finds the most specific generalizations of a new node without knowing any relationship between the new node and any existing nodes. The routine is called starting at the root node. The root node is known to be a generalization of all possible other nodes. However, there may be some specialization *S* of the root that is also a generalization of the new node. In such a case the new node should not be linked to the root because the root is not the most specific generalization.

The routine checks all specializations of the root node. If no specialization is found to be a generalization of the new node then the root is known to be the most specific generalization and the root is linked to the new node.

If any specialization *S* of the root node is found to be a generalization of the new node then LinkRoot is called recursively starting at *S*. A flag is set to indicate that the root node should not be linked to the new node. LinkRoot must be called recursively because node *S* may have a specialization that is a generalization of the new node. In such a case the new node should not be linked to node *S* because *S* is not the most specific generalization.

The specializations of node *S* are compared to the new node. If no specializations are found to be generalizations of the new node then node *S* is linked to the new node. Otherwise the process is repeated with the specialization by again calling LinkRoot recursively and node *S* is not linked.

Note that the search is limited to nodes that are generalizations of the new node. Thus the number of nodes visited is relatively small.

LinkRoot(*G*(search node**), *N*(**new node**)).**

```
    1.  Link = TRUE.
    2.  For all nodes S that are specializations connected to node G.
            3.  If S is a generalization of N then
                    4.  LinkRoot(S, N).
                    5.  Link = FALSE.
            6.  End.
    7.  End.
    8.  If Link and a link between G and N is not a redundant link then
            9.  Link G and N.
    10.         End.
End.
```

Figure 3. Link Generalizations

« LinkRootExample» shows an example of adding a new node (Node 8) using the LinkRoot procedure. LinkRoot begins at the root node. Specializations connected to the root node, Nodes 1 and 2, are compared to the new node. Node 1 is a generalization of the new node, so (1) a flag is set indicating that the root node should not be linked to the new node and (2) LinkRoot is called recursively starting at Node 1.

Specializations of Node 1, Nodes 5 and 6, are compared with the new node. Neither node is a generalization of the new node, so neither node is processed further. The *Link* flag remains set to true since no specializations of Node 1 were found to be generalizations of the new node. Therefore, Node 1 is linked to the new node as a generalization. The recursion terminates and returns to the LinkRoot procedure at the root node.

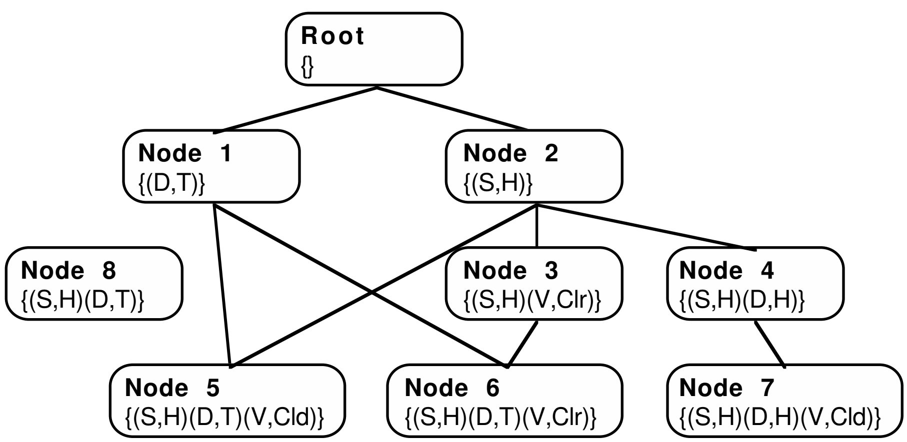

Figure 4. Finding Generalizations using LinkRoot

Node 2, the remaining specialization of the root node, is a generalization of the new node, so LinkRoot is called recursively starting at Node 2. Nodes 3, 4, and 5 are compared to the new node. None of the nodes are generalizations of the new node, so they are not processed further. The *Link* flag remains set to true since no specializations of Node 2 were found to be generalizations of the new node. Therefore, Node 2 is linked to the new node as a generalization. The recursion terminates and returns to the LinkRoot procedure at the root node. No specializations of the root node remain to be processed, so the linking procedure has completed. The resulting network is shown in « LinkRootExample2».

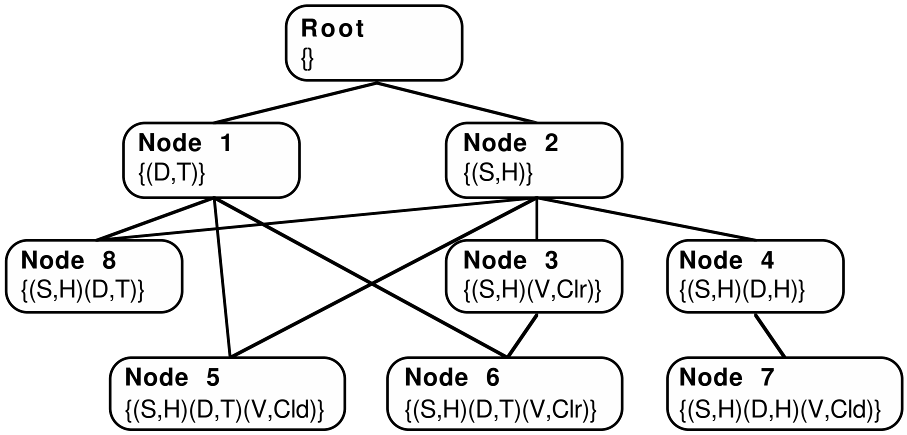

Figure 5. Network After Linking Generalizations

3.2. Specialization Links

This section explains how a new node is linked to its specializations. When linking to specializations, we assume that a generalization of the node is always available, since the node can be linked to its generalizations using the method described in the previous section.

The *specialization DAG* of a node *N* is the set of nodes and connections starting at *N* and going in the direction of specialization.

The routine *LinkSpec* shown in *« LinkSpec»* finds the most general specializations of a new node *N* given an existing node *G* in the network that is a generalization of *N*. By using node *G* that is already connected in the network the search is narrowed to only those nodes in the specialization DAG starting at node *G*. Since *G* is a generalization of *N*, *N* is a specialization of *G*, and all specializations of *N* are guaranteed to be contained in the specialization DAG of *G*.

The routine searches the specialization DAG of *G* starting at *G* and proceeding in a general to specific direction. Let *T* be a node in the specialization DAG of *G* that is also found to be a specialization of *N*. Since the search is general to specific, *T* is guaranteed to be a most general specialization. Therefore, Node *T* is linked to node *N*.

Node *T* is not guaranteed to be a most general specialization if the network allows redundant links. If redundant links are allowed, *G* can be linked to two specializations, S_1 and S_2, where S_1 is a specialization of S_2. The algorithm would link both nodes to *N* but the link to S_1 would be redundant.

All nodes in the specialization DAG beyond *T* are specializations of *T* so any links between those nodes and node *N* are redundant links. Therefore, the search of the DAG does not continue beyond node *T*.

Let *F* be a specialization of *G* that is not a specialization of *N*. If node *F* overlaps node *N* it is possible that some specialization of *F* is a specialization of *N*. Therefore, the search of the DAG is continued to the next level by calling the routine recursively starting with node *F*. If node *F* does not overlap (is discriminated with) node *N* then there is no specialization of *F* that can ever be a specialization of *N*, so the search of the DAG does not continue.

LinkSpec(*G*, *N*).

1. For all nodes *S* that are specializations connected to node *G*.
    2. If *S* is a specialization of *N* then
        3. If a link between *S* and *N* is not a redundant link then
            4. Link *S* and *N*.
        5. End.
    6. else If *S* overlaps *N* then
        7. LinkSpec(*S*, *N*).
    8. End.
9. End.

End.

Figure 6. Link a Node to its Specializations

As an example of LinkSpec, Node 8 in « LinkSpecExample» from the last section will be linked to its specializations. Since Node 8 is already linked to its generalizations, one of the generalizations can be selected and used as the starting point for LinkSpec. Suppose Node 2 is selected.

The routine starts at Node 2. Nodes 3, 4, and 5 are compared to Node 8 since they are specializations connected to Node 2. Node 5 is found to be a specialization of Node 8, so Node 5 is linked to Node 8 as a specialization. Node 3 is not a specialization of Node 8. However, Node 3 does overlap with Node 8. Therefore, LinkSpec is called recursively starting at Node 3.

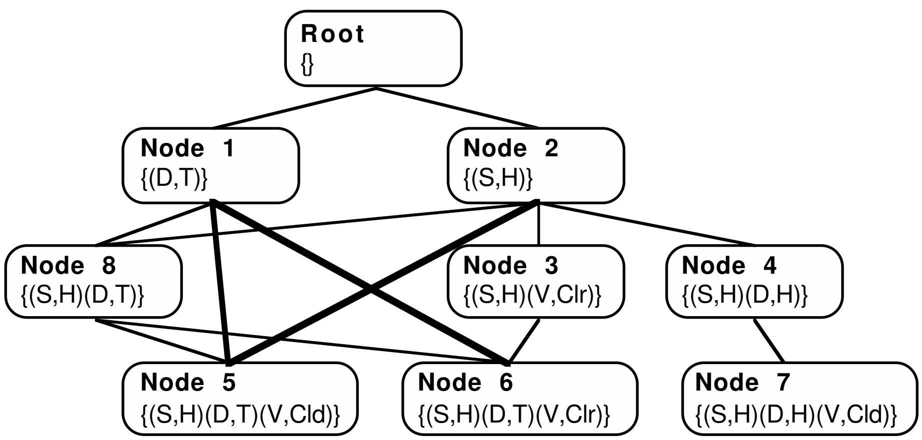

Figure 7. Completed Specialization Links

Specializations of Node 3 are compared with Node 8. Node 6 is found to be a specialization of Node 8, so Node 6 is linked to Node 8. No other specializations of Node 3 exist, so the recursion terminates and processing continues back with specializations of Node 2.

The remaining specialization of Node 2, Node 4, is discriminated with Node 8 by the *Wind Direction* input. Therefore LinkSpec is not called recursively on Node 4 and the linking process is complete. The resulting network is shown in « LinkSpecExample2».

3.3. Removing Redundant Links

« RedunLinks» shows Node 8 after it has been linked to both specializations and generalizations as described in the previous sections. Note that linking Node 8 into the network has created three redundant links (Node 1 to Node 5, Node 1 to Node 6, and Node 2 to Node 5). The routine *RemoveRedundant* must be run on Node 8 to remove the extra links.

RemoveRedundant considers all pairs of nodes (*G, S*), where *G* is a generalization directly connected to *N, S* is a specialization directly connected to *N,* and *N* is a new node that has just been linked into the network. If a link exists between *G* and *S,* the link is redundant and is removed. This simple routine keeps the network free from redundant links because it is run whenever a new node is added to the network. Thus, when the routine is run, the only redundant links in the network are those that were created by adding the new node. The routine is not intended to remove arbitrary redundant links.

RemoveRedundant(*N*).
1. For all nodes *G* that are generalizations connected to node *N*.
    2. For all nodes *S* that are specializations connected to node *N*.
        3. If *G* and *S* are linked then
            4. Unlink *G* and *S*.
        5. End.
    6. End.
7. End.

End.

Figure 8. Remove Redundant Links After Linking a New Node

The example shown in « RedunLinks» has four general/specific pairs for Node 8, (Node 1, Node 5), (Node 1, Node 6), (Node 2, Node 5), and (Node 2, Node 6). The first three pairs of nodes are linked so the links are removed giving the network shown in « RemoveRedunExample».

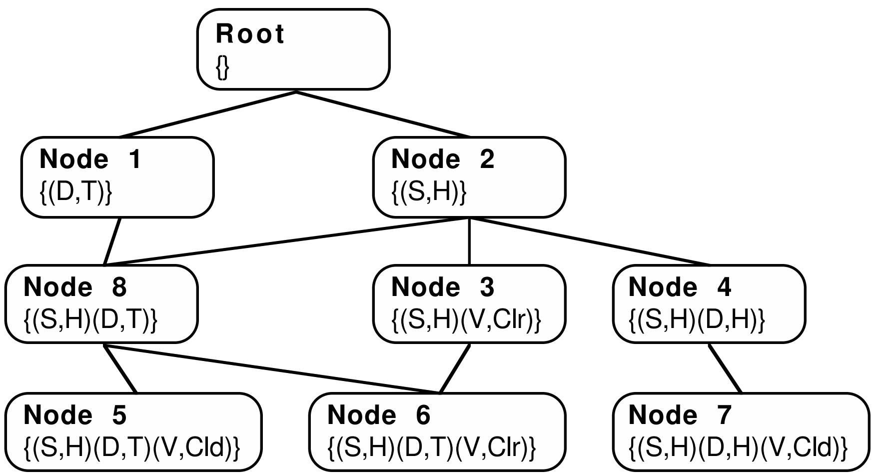

Figure 9. Network After Removing Redundant Links

4. Deleting Nodes

When a node *N* is deleted, all pairs of nodes (*G, S*), where *G* is a generalization directly connected to *N* and *S* is a specialization directly connected to *N* must be considered for linking. The process is essentially the reverse of removing redundant links when a node is added. If a link between *G* and *S* is not redundant (treating the node *N* as though it has already been removed from the network) then *G* and *S* are linked. If the node *N* is not treated as though is has already been removed from the network, then *N* will cause all links between all pairs (*G, S*) to appear redundant.

After the nodes around the node being deleted are linked together, then the node being deleted is unlinked from the network.

RemoveLinks(*N*).

1. For all nodes *G* that are generalizations connected to node *N*.
    2. For all nodes *S* that are specializations connected to node *N*.
        3. If a link between *G* and *S* is not a redundant link (assuming *N* is deleted) then
            4. Link *G* and *S*.
        5. End.
    6. End.
7. End.
8. For all nodes *G* that are generalizations connected to node *N*.
    9. Unlink *G* and *N*.
10. End.
11. For all nodes *S* that are specializations connected to node *N*.
    12. Unlink *S* and *N*.
13. End.

End.

Figure 10. Removing Links when Deleting a Node

As an example of removing a node from a network, Node 2 will be removed from the network in Figure 9. Three new links must be considered for creation (Root, Node 3), (Root, Node 4), and (Root, Node 8). Only the last link (Root, Node 8) is redundant so the other two are created. Then Node 2 is removed from the network giving the result shown in Figure 11.

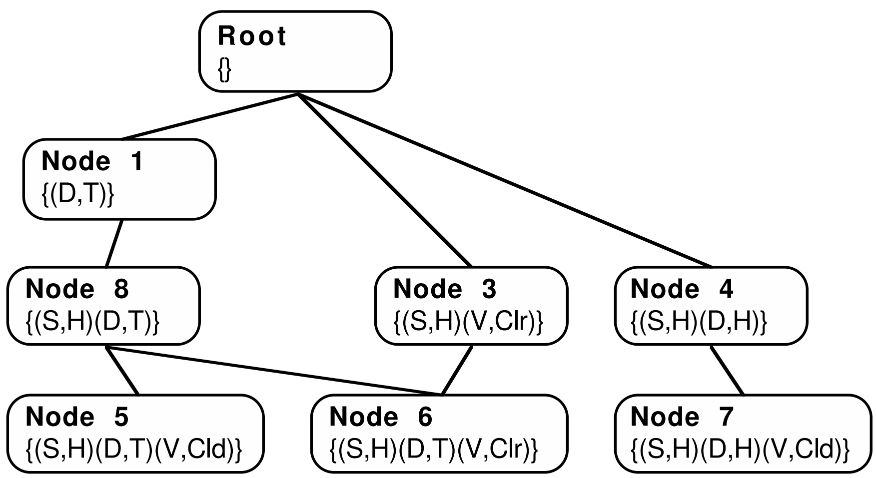

Figure 11. Network After Deleting Node 2

5. Conclusion

Learning systems often use representations for rules, features, or concepts that give rise to a more-specific-than relation. Using the more-specific-than relation, rules can be connected into a lattice structure such that rules with general-to-specific relationships are connected. The resulting network structure can be used to efficiently link new nodes into the structure as well as to delete existing nodes from the network. The connections between related nodes can be used to increase performance of learning and reasoning.

6. References

Barker, J. C. and Martinez, T. R. (1993a). GS: A Network that Learns Important Features. *Proceedings of The World Congress on Neural Networks*, Portland, Oregon, July 11-15, 1993.

Barker, J. C. and Martinez, T. R. (1993b). Generalization by Controlled Intersection of Examples. *Proceedings of The Sixth Australian Joint Conference on Artificial Intelligence*, Melbourne, November, 1993.

Barker, J. C. (1994). *Eclectic Machine Learning*. Doctoral Dissertation, Computer Science Department, Brigham Young University, Provo, Utah.

Hayes-Roth, F. and McDermott, J. (1978). An Interference Matching Technique for Inducing Abstractions. *Communications of the ACM, 21, 5*, 401-410.

Michalski, R. S. (1983). A Theory and Methodology of Inductive Learning. *Artificial Intelligence, 20*, 111-161.

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

Vere, S. A. (1975). Induction of Concepts in the Predicate Calculus. *Proceedings of the Fourth International Joint Conference on Artificial Intelligence*, IJCAI, Tbilisi, USSR, 281-287.