Joint Concept Formation

Huan Liu *and* Wilson X. Wen*

Summary

Many concept formation systems construct disjoint-concept trees. However, *a priori* imposed tree structures may restrict the application of these systems in some domains. A joint concept formation scheme is thus proposed, which learns from observation, and constructs acyclic directed concept graphs (trees are a special case). We show that the joint concept formation system can avoid or alleviate some problems the disjoint concept formation system would face, such as the unique winner and oscillation problems. We also demonstrate that a joint concept formation system is able to generate a concept tree if such a regularity is found among the data. The experimental results are consistent with the expectations that the joint system is a generalized version of the disjoint system and improves the learning performance. Joint concept formation extends the classic works, such as COBWEB, ARACHNE.

1 Introduction

There exist many views of concepts. In their review of human concepts and concepts formation, Medin and Smith [1984] summarized three views: the classical, probabilistic and exemplar views. The *classical* view holds that all instances of a concept share common properties that are necessary and sufficient conditions for defining the concept. The *probabilistic* view argues that concepts are represented in terms of properties that are only characteristic or probable of class members. Membership in a category can thus be graded rather than all-or-none. The *exemplar* view claims that categories may be represented by their individual exemplars, and that assignment of a new instance to a

*The order of the authors is alphabetical. Authors can be reached at AI Systems Section, Telecom Research Laboratories, 770 Blackburn Rd, Clayton, Victoria 3168, Australia. {h.liu, w.wen}@trl.oz.au

category is determined by whether the instance is sufficiently similar to one or more of the category's known exemplar. One of the functions a concept serves is simple categorization [Rey, 1984]: the means by which people decide whether or not something belongs to a simple class. The defining properties in the classic view of concepts are most effective in performing simple categorization. However, to obtain the defining properties directly has been proved to be very difficult [Medin and Smith, 1984].

In machine learning, researchers focus on the probabilistic and exemplar concepts. Concept formation stems from conceptual clustering [Michalski and Stepp, 1983], and is different from statistical clustering. The latter has been used primarily to provide different views of the same data or to explore data by using arbitrary similarity metrics and rules for group membership. Furthermore, statistical approaches tend to focus on the entire data set in their attempt to reduce the similarity measures into a more compact representation. In contrast, conceptual clustering is to help one better understand the world (represented by the data), and to make predictions about its future behavior [Hanson and Bauer, 1986; Quinlan, 1993]). The task of concept formation is to take a large number of unlabeled training instances; to find clusterings that group those instances in categories; to find an intensional definition for each category that summarizes its instances; and to find a hierarchical organization for those categories[Gennari *et al.*, 1989].

2 Related Work

Much of the recent research on this topic builds on Fisher's [1987] COBWEB. Fisher's system assumes that each instance is described as a conjunction of attribute-value pairs, and employs a probabilistic representation for concepts. All attributes of an instance are used in clustering and classification. COBWEB organizes its conceptual knowledge into a *tree* hierarchy, with nodes partially ordered according to their generality; thus the root node summarizes all instances that have been observed, terminal nodes correspond to single instances, and intermediate nodes, i.e. concepts, summarize clusters of observations. McKusick and Langley's ARACHNE [1991] goes further and seeks to construct well-formed concept hierarchies while maintaining high predictive accuracy by introducing some local constraints (see Section 3.3). Their comparative studies showed that ARACHNE creates better tree structures than COBWEB does after permuting the order of instances. A problem with these systems is that many of them impose a tree structure *a priori* [Cheeseman, 1990]. In the system proposed here, this constraint is relaxed. It is up to the data and their regularity that either a tree or an acyclic directed graph should be resulted.

UNIMEM [Lebowitz, 1987], a concept formation system, organizes knowledge into a concept hierarchy through which it sorts new instances, modifying the hierarchy in the process. Each concept has an associated set of instances stored with it; each instance may be stored with multiple nodes, and categories need not be disjoint. A node could be indexed by more than one attribute value from its parent nodes. If an instance matches the description on a node closely enough, then it sends the instance down those links that contain features in the instance, and it continues the process with the relevant children. Both the number of features necessary for this match and closeness of each value (for numeric attributes) are system parameters. For a detailed review of COBWEB and UNIMEM, please refer to [Gennari *et al.*, 1989].

3 A Joint Concept Formation System

We propose SGNN[1] here as a joint concept formation system which bears many similarities to COBWEB. It represents knowledge into a hierarchy of concepts, classifies new instances by sorting them down this hierarchy. However, SGNN can construct a concept hierarchy as an acyclic directed graph (ADG) if such a need arises from data. SGNN differs from UNIMEM in that an instance is only stored at one place, i.e., as one leaf node in the hierarchy, in addition to the differences between COBWEB and UNIMEM. COBWEB constructs probabilistic concepts, SGNN exemplar ones. Below we discuss, in turn, the SGNN algorithm, its updating rule, and the idea of achieving well structured hierarchies which was acclaimed in [McKusick and Langley, 1991], in an ADG.

3.1 The SGNN Algorithm

This section describes the SGNN generation algorithm. First we give some definitions:

[1] The name was originally coined for the work about Self-Generated Neural Networks.

An *instance* e_i is a vector of attributes: $e_i =< a_{i1}, ..., a_{in} >$.

A *node* n_j is a 3-tuple $< W_j, P_j, C_j >$, where

- W_j is the vector of the node: $W_j =< a_{j1}, ..., a_{jn} >$,
- P_j is the parent node set of n_j, and
- C_j is the child node set of n_j.

An ADG is a network of $< \{n_j\}, \{lnk_k\} >$, which is generated automatically from a set of training instances by the SGNN algorithm given below, where $\{n_j\}$ is the node set and $\{lnk_k\}$ is the link set of the ADG. A node is a *parent* node of another node if there is a directed link from the former to the latter. The latter is a *child* node of the former. A node without any child node is a *terminal* node. A *link* set of a node is a set that contains all the directed links that connect the node to its child nodes. An ADG can be divided into *layers*. The input is considered as layer 0. A node n_i is at later k if all of its parents are at layers l ($0 \leq l < k$) and at least one of its parents is at layer k - 1. A subnetwork of an ADG is said to be rooted by a node n_i if all nodes in this subnetwork can be reached from n_i through a directed path.

Initially, the network is empty. There is no node or link. When the first instance is presented, a node at layer 1 is generated for it and a link is established between the input (layer 0) and the node (layer 1).

Suppose now there is an intermediate ADG. When the next instance, e_i, is presented, the algorithm will search the network layer by layer to look for winners of each layer from layer 1. A node n_j is a *winner of layer 1*, l_1, if $|d(n_j, e_i) - minD| \leq \eta$, where $minD$ is the shortest distance between e_i and all nodes at layer 1. All such nodes in $\{n_j\}$ compose a *winner set*, W_1. A node n_j is a *winner of layer k*, l_k ($k > 1$), if $n_j \in W_{k-1} \cup C_k$ and $|d(n_j, e_i) - minD| \leq \eta$, where W_{k-1} is the winner set of l_{k-1}, all nodes in W_{k-1} are actually in l_{k-1} , C_k is the set of children of the nodes in W_{k-1}, and $minD$ is the shortest distance between e_i and all the nodes considered at layer l_k. If the winners of l_k are all at layer l_{k-1}, the search stops, otherwise, it continues at layer l_{k+1} until a terminal layer is reached. After the search stops, a *final winner set*, FW, is formed from those found winners that have no child node which is also a winner.

Above $d(n_j, e_i)$ is the distance between node n_j and instance e_i, and $\eta \geq 0$. A winner set contains all nodes whose distance to e_i exceeds the *minimal* distance, $minD$, by at most η. Any distance measure could be used, which measure should be used is subject to the application at hand. A Euclidean distance measure is used as an example here:

$$d(n_j, e_i) = \sqrt{\frac{\sum_{k=1}^{n}(a_{jk} - a_{ik})^2}{n}},$$

where n is the total number of attributes. The algorithm can be summarized as follows:

SGNN Algorithm (ADG Generation)

Input:

1. A set of training instances $E = \{e_i\}$, $i = 0, 1, ..., N$.
2. A threshold $\xi \geq 0$.
3. A distance measure for each attribute in nodes.

Output: An ADG generated from E.

Method:

```
copy(root,e0);
for(i=1,i<=N,i++)
    minDistance = test(ei);
    for(all fw∈FW)
        if(minDistance>ξ)
            if(terminal(fw))
                copy(n1,fw);
                connect(n1,fw);
            copy(n2, ei);
            connect(n2,fw);
        update(fw,ei);
```

where the routines are defined as follows:

1. `copy(n,e)`: generate a node n and copy the attribute values in the node e to the attributes in n.

2. `test(e)`: find all final winners, FW, in the current ADG for instance e and return the largest distance between the final winners and e.

3. `terminal(n)`: return 1 if node n is a terminal node in the current ADG, otherwise return 0. A node in an ADG is called a terminal node if it has no child node.

4. `connect`(n_0, n_1): connect node n_0 to n_1 making n_0 as a child node of n_1.

5. `update(n,e)`: update the attribute vector of node n by the attribute vector of e according to the updating rule (1) given below.

Choosing values for parameters η and ξ depends on applications at hand. Both could be set to zero if wanted. η is a discrimination threshold that determines how accurately a winner should be defined. The smaller η is, the fewer winners there could be and thus the fewer joint concepts. ξ is a splitting threshold that controls whether a new node should be generated. The larger ξ is, the fewer new nodes would be generated. Too small a value for either η or ξ is an over requirement for learning. However, an ideal value is very difficult to determine without having some experience on the data. The values of η and ξ have been chosen among 0.0, 0.1 and 0.01 for the tests in Section 5.

3.2 The Updating Rule

According to the above SGNN algorithm, each terminal node in an ADG corresponds to one or more instances and its attribute values are the averages of the corresponding attribute values of the instance or instances. The attribute values of each non-terminal node n_i are the averages of the corresponding attribute values of all the instances at the bottom of the subnetwork rooted (covered) by n_i, or

$$w_{ij} = \frac{1}{N_i} \sum_{k=1}^{N_i} a_{j,k}$$

where w_{ij} is the j-th attribute value of node n_i, N_i is the number of instances covered by n_i, and $a_{j,k}$ is the value of the j-th attribute of k-th instance covered by n_i. Thus we have the following successive updating rule:

$$w_{ij,k+1} = w_{ij,k} + \frac{1}{k+1} \cdot (a_{j,k+1} - w_{ij,k}) \qquad (1)$$

where $w_{ij,k}$ is the j-th attribute value of n_i after having seen the first k instances which are covered by n_i.

3.3 Achieving Well Structured Hierarchies

In order to reorganize the network into a better structure and make it more robust and accurate for clustering/classification, McKusick and Langley suggested some local constraints in [1991]. These are heuristics aiming at constructing a sensible concept structure. They are extended here for an ADG construction.

Definition 1. A node is *horizontally well placed* in an ADG if its distance to any of its parents is not greater than its distance to any of its siblings.

Definition 2. A node is *vertically well placed* in an ADG if its distance to any of its parents is not greater than its distance to its grandparent.

Definition 3. An ADG is *well organized* if all of its nodes are well placed both horizontally and vertically.

Optimizations may be performed to reorganize the network.

Horizontal Optimization: If a node n is not horizontally well placed in the network, e.g. the distance $d(n, p)$ to some of its parents p is greater than the distance $d(n, s)$ to one of its siblings, s. If s is its only sibling, delete s and move s's children up; otherwise, n need to be moved down under s. If n and s have common children, the connection from s to these children should be deleted. A new node need to be generated as another child of s with the same attributes as those of s whose connection from s has been deleted and the attributes of s need to be re-updated as follows:

$$w_{sj} = \frac{N_s \cdot w_{sj} + N_n \cdot w_{nj}}{N_s + N_n}$$

where N_s is the number of instances covered by s, etc. The number of instances covered by s should be, of course, increased to $N'_s = N_s + N_n$.

Vertical Optimization: If a node n is not vertically well placed in the network, i.e. $d(n, p)$ is greater than the distance $d(n, g)$ to some of its grandparents g, n need to be moved up as a child of g. If n has another parent p' which is also a child of g, only the connections from p' to n need to be deleted, no node should be removed. If p' has other parents in addition to g, the re-updating should be back propagated accordingly along those parents. The attribute value of p need to be re-updated according to

$$w_{pj} = \frac{N_p \cdot w_{pj} - N_n \cdot w_{nj}}{N_p - N_n}.$$

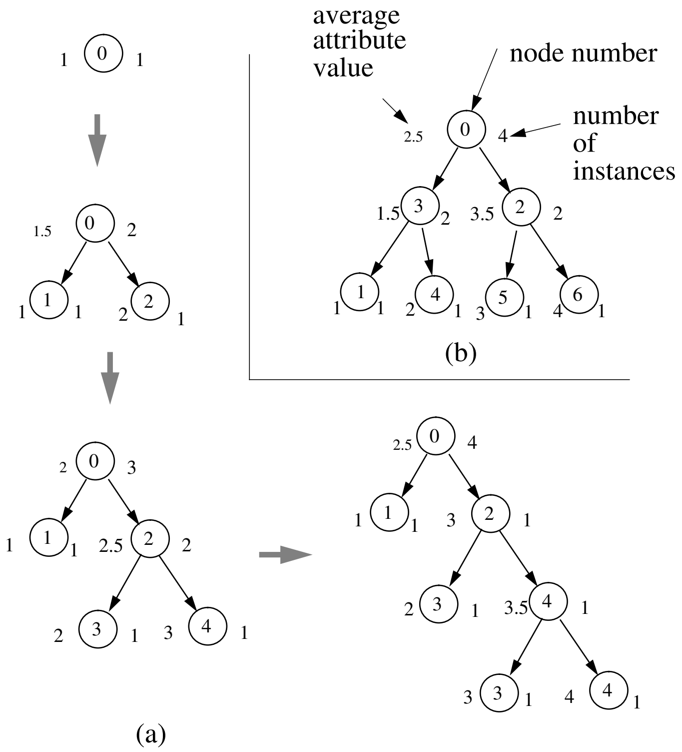

Figure 1: The use of horizontal and vertical optimization

The number of instances covered by p is also decreased to $N'_p = N_p - N_n$.

The use of vertical and horizontal (V/H) optimization can be demonstrated by the following simple example. Suppose there are four instances, 1, 2, 3, and 4. Each instance has a single slot representing both its class and attribute. The problem is to form a structure so that it can correctly classify those numbers if they are encountered again. In Figure 1, we ran through the generation algorithm to form the required structure. Two structures (a) and (b) are generated. Structure (a) is obtained without applying V/H optimization. It happens that structure (a) cannot classify 2 correctly, but structure (b) can after optimization is applied. However, as pointed out in [McKusick and Langley, 1991], by and large, no evidence suggests that a well structured tree can improve the predictive accuracy of a poorly structured one.

3.4 Classification and Prediction

In the proposed structure, detailed (statistical) information is stored at the leaf layers only. In general, each node only keeps a summary of its child nodes' attributes as shown in the updating rule (1). Information (statistics) at each intermediate node can be reconstructed by traversing the subtree rooted at the node[2]. During classification, the statistical information is not needed due to the successive nature of the updating rule. Classification is to assign an instance to a certain class. If there exists a single winner, the instance would be classified under the winner. The procedure continues until there is no class to which the instance can be classified from the root down in an ADG. Its parent node is then chosen to represent the instance (*prediction*). If the parent node has class information, the instance is then classified to that class. In a case of multiple winners, the instance would be sorted down through these winners until each path is terminated. When the classification finishes, the winners who don't have any child winner are chosen to represent the instance. Hence, an instance could be classified into several classes. Winners can be ranked according to their distances to the instance. The shorter the distance, the better a winner is. When missing attribute values and/or class name occur in an instance, winners' counterparts are used to predict these missing values.

If a probabilistic representation is adopted, however, the statistical information has to be stored at every node in order to calculate category utility. Both clustering and classification would be much slower by a COBWEB like system than SGNN since the successive nature of the updating rule tremendously simplifies computation.

4 Why Joint Concept Formation?

Most of the work on concept formation so far has been done for concept tree structure. Although a tree structure is simple and efficient, an ADG structure has some advantages:

1. There are domains in which data cannot be exclusively clustered. Since a tree structure has to choose only one winner among more than one winner even if they may be equivalently good (but an ADG need not do so), this may cause an unexpected classification (see Example 1).

2. An ADG is more robust than the corresponding tree structure. A failure of one non-terminal node may not induce the failure of the whole subtree rooted by it.

[2]Statistics would be needed in construct a logical representation for a concept (node).

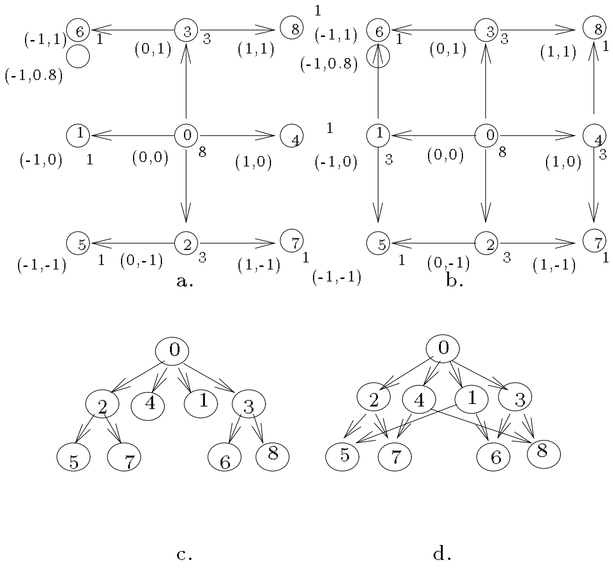

Figure 2: A Comparison between a tree and an ADG

3. Joint concept formation could be viewed as an attempt toward "multiple levels of aggregation" as [Gennari *et al.*, 1989] pointed out in their further work directions.

Example 1. Suppose we would like to learn the directions on a map (Figure 2). We use X coordinate values 0, 1, -1 to represent *central, east* and *west*, and Y coordinate values 0, 1, -1 to represent *central, north*, and *south.* Given a training set

$$\begin{array}{llll} (1,0), & (-1,0), & (0,-1), & (0,1), \\ (-1,-1), & (-1,1), & (1,-1), & (1,1) \end{array}$$

we may generate a well organized tree in Figure 2a which is equivalent to Figure 2c topologically.

4.1 The Unique Winner Problem of a Tree Structure

A tree of Figure 2c learned from the training set can classify the training instances correctly. However, when the classification of the testing data $(-1, 0.8)$ is attempted, the tree classifies it as *west* $(-1, 0)$, instead of *north-west* $(-1, 1)$, because the winner in the first layer of the tree is $(-1, 0)$, node 1, which is a terminal node, an unexpected classification, *west*, is resulted. A correct classification, *north-west*, is missed. Furthermore, the concept *north-west* or $(-1, 1)$ has never been learned properly because only one parent is allowed in the tree, either node 1, *west*, or node 3, *north*, but not both. This is caused by having to choose one winner only for a tree structure. Assuming that there is a link between nodes 1 and 6 instead of the link between nodes 3 and 6, then an instance (-0.8, 1) would not be classified properly.

By using an ADG of Figure 2d which is topologically equivalent to that in Figure 2b, since there are two links from nodes 1 and 3 to node 6, the previous wrong classifications of the testing instances disappear.

4.2 The Oscillation Problem of Optimization

Horizontal and vertical optimizations among nodes with a single parent may cause a problem of oscillation. Consider a case in which there are instances, (-1, 0), (0, -1), (0, 1) and (1, 0), nodes 1, 2, 3 and 4 respectively. Nodes 1, 2, 3, and 4 are well organized under node 0 (the root) in Figure 3a. As node 5, (-3, 3), is incorporated, a structure of Figure 3b is constructed, which is well organized as well. Now comes node 6, (3, -3). Since its distances to nodes 2 and 4 are equivalent, we can take either one to construct a tree structure. Suppose node 4 is chosen to proceed. Node 7 is introduced as in Figure 3c. Then node 4 is not vertically well placed since its distance to its parent, node 7, is larger than the one to its grandparent, node 0. Hence, node 4 is moved up by one layer. We obtain Figure 3d thereof. Then, node 6 is not horizontally well placed since its distance to its sibling, node 4, is shorter than the one to its parent, node 0. Horizontal optimization changes Figure 3d back to Figure 3c. We thus observe oscillation.

A learning process cannot end if oscillation occurs. The oscillation problem is mainly caused by the vertical and horizontal (V/H) optimization among the nodes with a single parent. The use of ADG's appears to be more robust than the use of trees since an ADG allows multiple paths to a node from its parent nodes. The oscillation observed in Figure 3 disappears in an ADG because nodes 2 and 4 are both parents of node 6. The oscillation problem can thus be alleviated in an ADG. However, since there could still be nodes that have only one parent in an ADG, oscillation could occur while V/H optimization is attempted. In other words, for a densely connected ADG, when V/H optimization is performed, there might be a smaller probability to observe oscillation. However, for

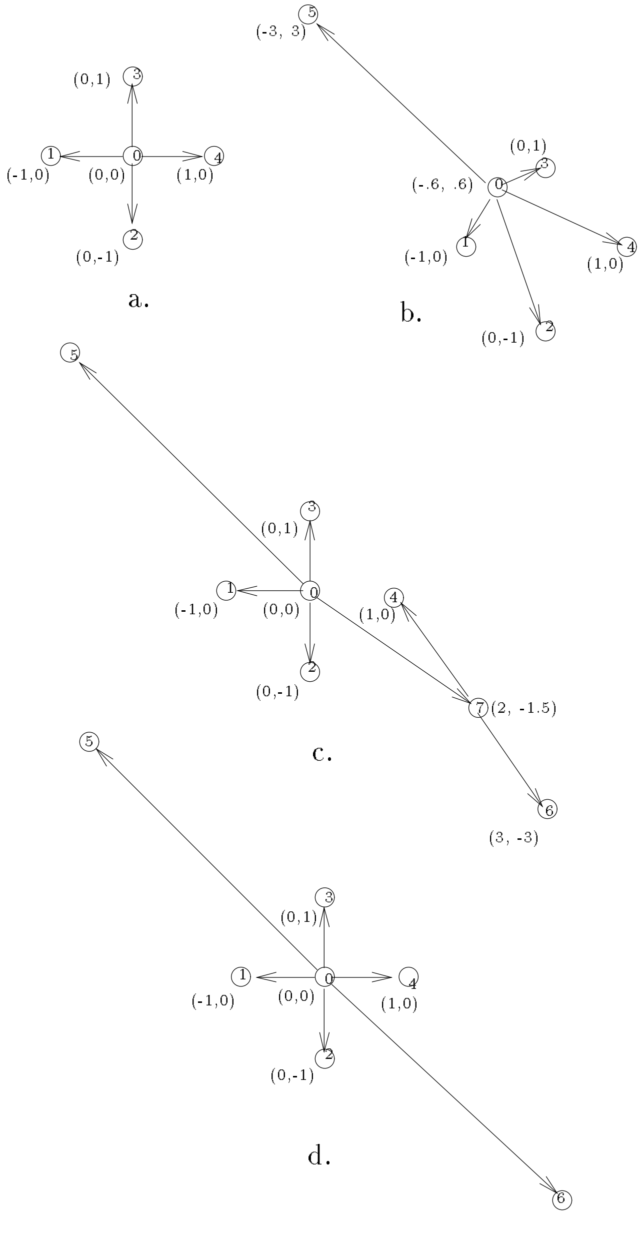

Figure 3: An oscillation observed.

a densely connected ADG, the role of V/H optimization is not obvious. In our experiment, V/H optimization didn't improve the performance (predictive accuracy) much, yet required a substantial increase in computation. Occasionally, oscillation occurred and the learning process had to be killed manually - in practice, this type of termination of a program could be very costly for some applications. It is not a good idea to perform V/H optimization if one cannot afford the risk of oscillation. In the reported performance of the three testing data sets, V/H optimization was not performed.

4.3 From an ADG to a Tree

Employing an ADG, SGNN can not only alleviate the oscillation problem, but also improve the predictive accuracy. Recall that a tree is a special case of an ADG. SGNN forms either an ADG or a tree based on the nature of the data presented if η and ξ are properly chosen. If the data is exclusively classifiable, a tree is created since at every layer, there is only one winner. If the data is non-exclusively classifiable, however, an ADG is constructed since there could be more than two winners at each layer. Moreover, a tree can be always generated by SGNN if the constraint is enforced that only one winner should be chosen. The flexibility of building a tree or ADG is a significant extension to tree-structure concept formation systems since it can be used in more applications.

As mentioned above, the idea of extending a disjoint concept formation system to a joint one is quite intuitive and simple. However, its implementation is not that straightforward, clustering and classification complexities increase due to its multiple paths, and data structure is more complex because of the need for reorganization of the concept hierarchy. Furthermore, a tree version of SGNN runs faster than an AGD one.

5 Comparisons and Experiment Results

SGNN has been developed in $C/C^{++}/Xt$ on SUN Sparc station with an X window interface under UNIX. Some initial experiments have been conducted to evaluate the system and the method. The results are given below. Testing data are public domain data sets from University of California, Irvine. The class information in these data sets is not used in the training phase, but in the testing phase in order to obtain a predictive accuracy.

x_1:	head_shape	$\in$	round, square, octagon	x_2:	body_shape	$\in$	round, square, octagon
x_3:	is_smiling	$\in$	yes, no	x_4:	holding	$\in$	sword, balloon, flag
x_5:	jacket_color	$\in$	red, yellow, green, blue	x_6	has_tie	$\in$	yes, no

Table 1: The monk's problems: attributes and values.

1. US Voting:
   - There are 17 attributes $\times$ 435 instances in 2 classes.
   - 300 training instances are randomly chosen from the whole instance set,
   - The rest of the 135 instances form the testing set.
   - Test result for clustering/classification:
     - training data: 100% correctness,
     - testing data: 95% correctness (This is the same as the best result reported previously [Schlimmer, 1987]).
2. Mushroom classification:
   - There are 23 attributes $\times$ 8124 instances in 2 classes.
   - 1000 training instances are randomly chosen from the whole set,
   - The rest of the 7124 instances are the testing set.
   - Test result for clustering/classification:
     - training data: 100% correctness,
     - testing data: 99.6% correctness for tree structure, 100% correctness for ADG's (the best result reported for previous unsupervised learning is 95% correctness for about only 3000 instances [Schlimmer, 1987; Iba *et al.*, 1988]).
3. The monk's problems [Thrun *et al.*, 1991]: The problems rely on an artificial robot domain, in which robots are described by six different attributes in Table 1. Three learning tasks are denoted as M1, M2, and M3.

   M1: (head_shape = body_shape) or (jacket_color = red).

From 432 possible examples, 124 were randomly selected for the training set. There were no misclassifications.

M2: exactly two of the six attributes have their first value.

From 432 possible examples, 169 were randomly selected. No noise.

M3: (jacket_color is green and holding a sword) or (jacket_color is not blue and body_shape is not octagon).

From 432 examples, 122 were selected randomly, and among them there were 5% misclassifications, i.e. noise in the training set.

The comparison results on predictive accuracy is shown in Table 2. CLASSWEB is a combination of the algorithms COBWEB [Fisher, 1987] and CLASSIT [Gennari *et al.*, 1989]. ECOBWEB is a variation of COBWEB. Please refer to [Thrun *et al.*, 1991] for the details of CLASSWEB and ECOBWEB (pp 62 and 97 respectively).

Learning method tested	Test results				Tester(s)
	M_1	M_2	M_3	**Average**	
SGNN	82.6%	78.2%	84.5%	**81.8%**	Wen et. al.
CLASSWEB 0.10	71.8%	64.8%	80.8%	**72.5%**	Kreuziger et. al.
CLASSWEB 0.15	65.7%	61.6%	85.4%	**70.9%**	Kreuziger et. al.
ECOBWEB leaf prediction	71.8%	67.4%	69.1%	**72.5%**	Reich & Fisher
ECOBWEB information utility	82.7%	71.3%	68.0%	**74.0**%	Reich & Fisher

Table 2: Accuracy comparisons for unsupervised learning methods.

6 Conclusion

In this paper, we present a joint concept formation system, SGNN, that extends the previous work of concept formation [Fisher, 1987; McKusick and Langley, 1991]. SGNN is able to generate either disjoint concept trees or acyclic directed concept graphs, according to the characteristics inherited in domain data. Furthermore, with certain controls applied to the number of winners at each concept layer, SGNN can also be used to only construct disjoint concept trees regardless of the regularity inherited in data. It is demonstrated by example that SGNN is robust in terms of the oscillation problem, and overcomes the single winner problem, performing well when a tree structure encounters difficulties in clustering and classification. It should also be noted that SGNN runs much faster than the systems compared above due to its successive updating rule.

Using V/H optimization sometimes helps build a more balanced concept structure. However, we also observe the oscillation problem. An ADG structure can only alleviate the problem, but cannot eliminate it. Because of that, V/H optimization should not be used freely. For a densely connected ADG, V/H optimization might not be helpful at all, yet consumes a lot computation time with a risk of oscillating.

As mentioned earlier, a choice of a distance measure depends on an application at hand. We have experimented measures like hamming distance, city blocks, inner product in addition to Euclidean distance. Performances of the three testing sets vary marginally with different measures. Further study would help us understand why SGNN is robust towards these measures. It is also our interests to understand the effects of different representations (a probabilistic one[3] vs. a distance one in particular) on nominal, ordinal, or numerical data. Our preliminary experiments elsewhere [Liu *et al.*, 1993] show that the former produces a more regular concept structure but with a less predictive accuracy; the latter yields a less regular structure but with a better predictive accuracy with some nominal data. More study is needed to have a full account of the effects of representations [Weiss and Kulikowski, 1991; Quinlan, 1993].

Joint concept formation systems can be applied to more domains (both exclusively and non-exclusively classifiable data) than disjoint ones. Currently, SGNN has been applied to several domains. Among them are: handwritten characters recognition, fault diagnosis, data/image compression, and information indexing/retrieval. The preliminary results are fairly encouraging. Further research includes: (1) searching for comprehensible (e.g., defining) concepts, or a method which can transform probabilistic or exemplar concepts to defining concepts. This work is important due to the role of concept formation in knowledge acquisition and reuse. Concepts that are hard to comprehend would be difficult to use as well; and (2) constructing concept structures using Minimum Description Length/Minimum Message Length [Wallace and Boulton, 1968;

[3]ECOBWEB and CLASSWEB learn concepts based on category utility.

Rissanen, 1983] in addition to using the evidence from the psychological background, so that we might be able to find the link between information theory and cognitive psychology, better explain the learned concepts.

Acknowledgements

The permission of Managing Director, Research and Information Technology, Telecom Australia Research Laboratories to publish this paper is acknowledged. The authors also wish to thank helpful comments from anonymous reviewers and Dr. Brian R Gaines.

References

[Cheeseman, 1990] P. Cheeseman. On finding the most probable model. In J. Shrager and P. Langley, editors, *Computational models of scientific discovery and theory formation*. Morgan Kaufmann, San Mateo, California, 1990.

[Fisher, 1987] D.H. Fisher. Knowledge acquisition via incremental conceptual clustering. *Machine Learning*, 2:139–172, 1987.

[Gennari *et al.*, 1989] J.H. Gennari, P. Langley, and D. Fisher. Models of incremental concept formation. *Artificial Intelligence*, 40:11–61, 1989.

[Hanson and Bauer, 1986] Stephen Jose Hanson and Malcolm Bauer. Machine learning, clustering and polymorphy. In L.N. Kanal and J.F. Lemmer, editors, *Uncertainty in Artificial Intelligence*, pages 415–28. Elsevier Science Publishers B.V., North-Holland, 1986.

[Iba *et al.*, 1988] W. Iba, J.Wogulis, and P.Langley. Trading off simplicity and coverage in incremental concept learning. In *Proc. 5th International Conference on Machine Learning*, pages 73–79, 1988.

[Lebowitz, 1987] M Lebowitz. Experiments with incremental concept formation. *Machine Learning*, 1:103–138, 1987.

[Liu *et al.*, 1993] H. Liu, W.X. Wen, and C. Rowles. Knowledge acquisition through concept formation. Technical report, Telecom Australia Research Laboratories, 1993.

[McKusick and Langley, 1991] K. McKusick and P. Langley. Constrains on tree structure in concept formation. In *Proc. IJCAI'91*, volume 2, pages 810–816, Sydney, Aug. 1991. Morgan Kaufmann.

[Medin and Smith, 1984] D.L. Medin and E.E. Smith. Concepts and concept formation. *Ann. Rev. Psychol.*, 35:113–38, 1984.

[Michalski and Stepp, 1983] R.S. Michalski and R.E. Stepp. Automated construction of classifications: Conceptual clustering versus numerical taxonomy. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 5(4), 1983.

[Quinlan, 1993] J. Ross Quinlan. *C4.5: Programs for Machine Learning.* Morgan Kaufmann Publishers, San Mateo, California, 1993.

[Rey, 1984] G. Rey. Concepts and stereotypes: a critical discussion of categories and concepts. *Cognition*, 1984.

[Rissanen, 1983] J. Rissanen. A universal prior for integers and estimation by minimum description length. *The Annals of Statistics*, 11(2):416–431, 1983.

[Schlimmer, 1987] J.S. Schlimmer. Concept acquisition through representational adjustment. Technical report 87-19, Department of Information and Computer Science, University of California, Irvine, 1987.

[Thrun *et al.*, 1991] S.B. Thrun, J. Bala, E. Bloedorn, I. Bratko, B. Cestnik, J. Cheng, K. De Jong, S. Dzeroski, S.E. Fahlman, D. Fisher, R. Hamann, K. Kaufman, S. Keller, I. Kononenko, J. Kreuziger, R.S. Michalski, T. Mitchell, P. Pachowicz, Reich Y., H. Vafaie, W. Van de Welde, W. Wenzel, J. Wnek, and J. Zhang. The monk's problems: A performance comparison of different learning algorithms. Technical Report CMU-CS-91-197, Carnegie Mellon University, December 1991.

[Wallace and Boulton, 1968] C.S. Wallace and D.M. Boulton. An information measure for classification. *Computer Journal*, 11(2), 1968.

[Weiss and Kulikowski, 1991] Sholom M. Weiss and Casimir A. Kulikowski. *Computer Systems That Learn.* Morgan Kaufmann Publishers, San Mateo, California, 1991.