In *Journal of Parallel and Distributed Computing*, vol. **26**, pp. 125-131, 1995.

Analysis of the Convergence and Generalization of AA1

Christophe Giraud-Carrier and Tony Martinez

Department of Computer Science, Brigham Young University

Abstract

AA1 is an incremental learning algorithm for Adaptive Self-Organizing Concurrent Systems (ASOCS). ASOCS are self-organizing, dynamically growing networks of computing nodes. AA1 learns by discrimination and implements knowledge in a distributed fashion over all the nodes. This paper reviews AA1 from the perspective of convergence and generalization. A formal proof that AA1 converges on any arbitrary Boolean instance set is given. A discussion of generalization and other aspects of AA1, including the problem of handling inconsistency, follows. Results of simulations with real-world data are presented. They show that AA1 gives promising generalization.

1. Introduction

The last decade has seen a renewed interest in connectionist computing. Some of the reasons for this interest can be traced to the discovery of the wide range of neural network applications, the increasing interest in self-organizing systems, and the design of improved learning algorithms. A good overview of the current state of the art is found in [11]. Most of the current models are based on static networks of computing nodes, where learning is effected by changing the weights of the connections between nodes and/or by modifying the nodes' functions. Adaptive Self-Organizing Concurrent Systems (ASOCS) are dynamic networks that learn by adapting both the function performed by the nodes and the overall network topology. Hence, the network grows over time to fit the problem.

ASOCS operate in one of two modes: learning and execution. In execution mode, the network receives inputs and produces outputs as the data flow asynchronously and in parallel through the network. In learning mode, the system is incrementally presented a set of Boolean rules, which is called the *instance set*. As each instance is presented, the network adapts both its topology and its nodes' functions to learn the new instance, while preserving consistency with previously acquired knowledge.

Several learning algorithms have been developed for ASOCS. They include Adaptive Algorithm (AA) 1 [3], AA2 [4] and AA3 [5]. AA1 is the object of this paper. AA1 learns by discrimination and implements knowledge in a distributed fashion. This paper analyzes the convergence and generalization properties of AA1.

Convergence refers to the system's ability to learn the instance set. Convergence has played a significant historical role in the development of neural networks. Rosenblatt's perceptron [9], the first neural network, was quickly proven to be limited to linearly separable functions [7]. However, it was also shown that a (at least) three-layer network could be used to represent any arbitrary function. The challenge was thus to design a learning algorithm for multi-layer networks. Researchers struggled until the first multi-layer learning algorithm, backpropagation [11, 12], was proposed. However, because it uses a gradient-descent technique and is thus subject to local minima, there is no guarantee, in practice, that backpropagation will find a plausible solution to a training set. The first part of this paper reviews AA1 and formally proves that AA1 converges on any arbitrary Boolean instance set. Moreover, unlike backpropagation which requires several passes over the training set, AA1 is a "one-shot" learner with fast, bounded learning time.

During learning, the system is presented with only a subset of the function it is to solve. Generalization refers to the system's ability to correctly guess the rest of the function. To be of any practical use, a system should not be confined to simply remembering (i.e., rote learning), but should also be capable of inducing new knowledge from past experience. Sections 4 and 5 of this paper discuss AA1's generalization scheme and show that it produces promising results. AA1's originality lies in the fact that it generalizes not on the basis of how closely the new unknown input matches one of the learned rules, but rather on how well the network can discriminate the unknown input from the learned features of the opposite class.

Section 2 gives an overview of AA1. Section 3 contains the proof that AA1 converges on any arbitrary Boolean instance set. Section 4 discusses other features of AA1 including generalization, and the handling of inconsistency. Section 5 presents results of simulations of AA1 on real-world data. Section 6 summarizes the paper.

2. AA1 Learning Algorithm

This section gives a short overview of AA1. Only those details essential to the proof of the following section are presented. For more details on AA1, see [2, 3].

An *instance set* is a set of *instances*, where each instance is a conjunction of input values together with the output value they imply. We use V and V' to mean *V is positive* (i.e., true) and *V is negative* (i.e., false), respectively. The following are examples of instances:

$$AB \rightarrow Z \quad \text{(i)} \qquad B'C \rightarrow Z \quad \text{(ii)} \qquad AC' \rightarrow Z' \quad \text{(iii)}$$

The conjunctive part of instances need not contain an input value for each input variable. Such instances are used as short-hand. If I is an instance in which no value is specified for some input variable V, then I represents both instances obtained from I by adding V and V', respectively, to the conjunctive part of I. For example, if A, B and C are the only 3 input variables, then instance (i) represents both instances $ABC \rightarrow Z$, and $ABC' \rightarrow Z$. In other words, instance (i) means that when A and B are both positive, Z must become true, regardless of the values of any other input variables.

The value of the output implied by an instance is its *polarity*. Instances (i) and (ii) are positive and instance (iii) is negative. Two instances with the same polarity are said to be *concordant* with respect to each other, while two instances of opposite polarity are *discordant*. Instances (i) and (ii) are concordant; instances (ii) and (iii) are discordant.

The basic architectural unit in AA1 is a *node*. During execution, each node computes the AND or OR function of its (possibly inverted) inputs. During learning, since all inputs may not have a value in an instance, these functions are extended to accommodate unspecified (denoted ?) inputs (e.g., 0 AND ? implies 0, 0 OR ? implies ?). In addition, each node records, in the form of a *node table* (*NT*), the value it outputs for each instance in the training set. The NT has two columns, P and N, holding the output values for positive and negative instances, respectively, and a column D which is used in the node selection part of the algorithm presented below. Figure 1 shows an example of a node.

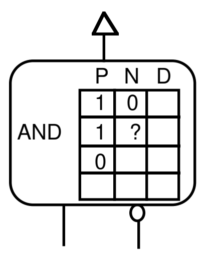

Figure 1 - Example of a Node

AA1 incrementally builds a network with a single top node whose output is the network's output. Figure 2 shows an example of such a network (for simplicity, the D columns are not represented). During execution, the node tables are ignored and the system functions as a logic network. For example, if B, C, D, and G are all positive, then the network of Figure 2 outputs 1. During learning, the node tables are used to control discrimination as explained below.

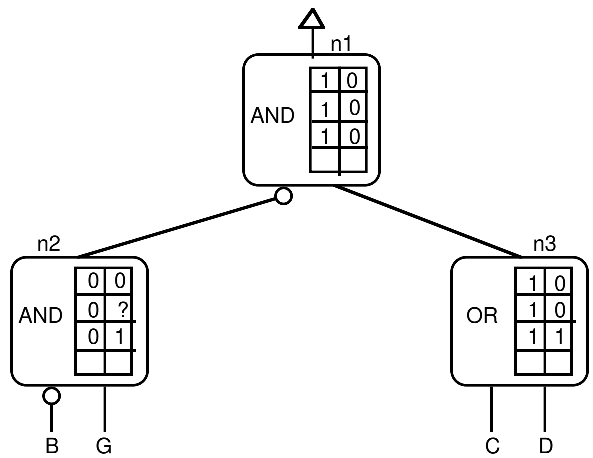

Figure 2 - Example of a Network

If a cell in the NT contains the value 0 or 1, the node *discriminates* the instance represented by that cell from all discordant instances whose cells contain the opposite value. For example, let K be the node n3 of Figure 2. Consider the first cell of the P column whose value is 1, and let I be the corresponding instance. If K outputs 1, there is no way to tell if it is I that is matched or if it is the instance corresponding to the third cell of the N column. However, if K outputs 0, clearly I is not matched. But K outputs 0 for the instances corresponding to the first and second cells of the N column, so K is able to discriminate between I and these two instances. Note that cells containing ?'s cannot participate in discrimination. There are four classes of nodes, based on discrimination:

- *discriminant node*: discriminates at least one positive instance from one negative instance,
- *non-discriminant node*: does not discriminate at least one positive instance from one negative one,
- *complete discriminant node*: discriminates every positive instance from every negative instance,
- *one-sided discriminant node*: asserts one value for either all positive or all negative instances, and the opposite value for at least one discordant instance (see node n2 of Figure 2).

Discrimination is the key factor in AA1. When it is presented an instance set, AA1 learns to discriminate between positive and negative instances. AA1 *fulfills* the instance set (i.e., converges) if the network it builds outputs 0 for every negative instance and 1 for every positive instance. That is, AA1 converges if and only if the final network's top node is complete discriminant and has 1's in its P column.

Learning proceeds as follows. Instances are introduced one at a time in an incremental fashion. For each new incoming instance, AA1 goes through a preprocessing phase that maintains the instance set consistent and seeks to minimize it. An instance set is inconsistent if it contains discordant instances whose conjunctive parts could be true simultaneously. For example, instances $AB \rightarrow Z$ and $BC \rightarrow Z'$ produce an inconsistency since for A, B and C positive, the first one implies that Z is positive while the second one implies that Z is negative. Inconsistency is solved by giving precedence to the newer instances (see [6] for details). Complete minimization is not reasonable, but AA1 attempts partial minimization through pairwise comparison of the training instance and instances in the current instance set. The correctness of this aspect of the algorithm has been proved elsewhere [10]. The correctness of the actual learning algorithm has not. Section 3 fills this gap.

The result of the above preprocessing phase is a *delete-list* and an *add-list* containing the instances to be removed and added, respectively, from the current instance set to keep it consistent and somewhat minimal. To process the delete-list, AA1 simply causes each node to empty the corresponding cells in its NT. Each instance in the add-list is then added to the current instance set and presented to the network. Each node places its output for the new instance in the corresponding cell of its NT. The network's output is then checked. If it is concordant with that of the new instance, no changes are made to the network and the next training instance can be processed. If, on the other hand, it is discordant or ?, then the current top node is no longer complete discriminant and modifications must be made to the network. Since the current top node correctly discriminates the instance set, with the exception of the new instance, AA1 constructs a new node that discriminates the new instance from all discordant instances. This one-sided discriminant node (OSDN) is then combined with the current top node to build a new complete discriminant node. The network finally undergoes a phase of *self-deletion* in which nodes that are no longer needed (such as non-discriminant nodes) are removed from the network. Self-deletion increases parsimony. Details on the various kinds of self-deletions may be found in [2, 3].

The construction of OSDN is effected by a process of *node selection* and *node combination*. Node selection consists of a greedy search through the network for a set of nodes that, when combined, give rise to OSDN. The nodes that are selected are those nodes that discriminate the new instance from the largest number of remaining non-discriminated discordant instances (the D column of the NT keeps track of which discordant instances are already discriminated by the selected nodes). If node selection fails, i.e., if the selected nodes would not combine to create OSDN, then *node creation* takes place, and AA1 guides the construction of new nodes that can discriminate the new instance from all remaining non-discriminated discordant instances. From each remaining non-discriminated instance an input is randomly selected that is sufficient to discriminate it from the new instance. Pairs of these chosen inputs are then used as inputs for the new nodes whose functions must be set so that they output 0 or 1 when both inputs are matched and the complement when either one is not matched. Once all the necessary *growth* nodes have been selected and/or created, they are combined so as to build OSDN. Each combination connects two nodes to a new node that discriminates the union of the discriminations done by the nodes from which it was created.

Due to space, we can only give a high-level example of how AA1 updates the network upon receipt of a new instance. A detailed example is found in [3]. Consider the network of Figure 2. Assume the new instance is positive and causes the top node to output ?, node n3 to output 1 and node n2 to output ?. The result of processing the new instance is in Figure 3. Note that Node Selection took place and node n5 was selected as a growth node, since it discriminates the new instance from two of the three discordant instances. Node Addition was then necessary and resulted in the creation of input A (sufficient to discriminate the new instance from the remaining discordant non-discriminated instance). Node n5 was subsequently combined with input A (degenerated growth node) to produce a OSDN (node n3). The OSDN was finally combined with the old top node (node n2) to produce a new complete discriminant top node (node n1).

When two nodes N_1 and N_2 combine and give rise to a parent node *PN* with function *f*, *PN*'s node table is loaded by applying *f* pairwise to corresponding cells in N_1's and N_2's node tables. It follows that *PN*'s node table contains the values that *PN* would output for each instance in the current instance set (i.e., there is no need to rebroadcast all the instances of the instance set to the modified network).

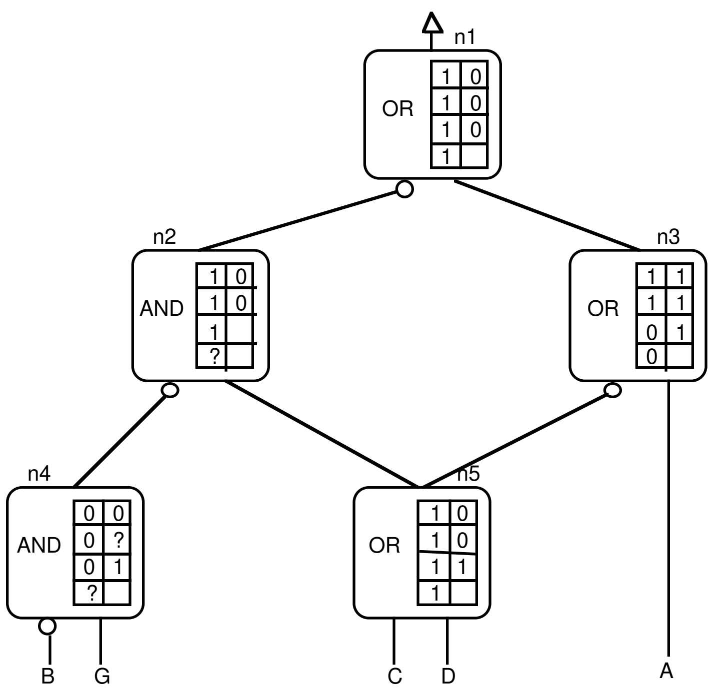

Figure 3 - Modified Network

3. AA1 Is Correct

In this section, we prove that AA1 always fulfills the current instance set. Note first that processing the delete-list is trivially correct since emptying cells does not change the complete discriminant nature of the current top node. We show that processing the add-list is also correct.

Let *NI* be the instance being processed. Let *I* be the set of current discordant instances that *NI* must be discriminated from, and *OSDN* a one-sided discriminant node that discriminates *NI* from *I*. Let *CTN* be the current top node, and *NTN* the node resulting from combining *CTN* and *OSDN*.

Lemma 1 shows that Node Selection and Node Addition give rise to a set of growth nodes, each of which discriminates *NI* from some subset of *I* such that the union of these subsets is *I*. Lemma 2 shows that when any two growth nodes, discriminating *NI* from two subsets of *I*, are combined, the resulting parent node discriminates *NI* from the union of these subsets. Theorem 1, which shows that the result of Node Combination is *OSDN* follows immediately by induction. Theorem 2 then shows that AA1 fulfills the instance set after *NI* has been processed. The result follows immediately from the finiteness of the add-list.

<u>Lemma 1:</u>

Let $G_1,\ldots,G_k$ be the growth nodes resulting from Node Selection and Node Addition. Let $I_1,\ldots,I_k$ be the respective sets of instances that each G_i discriminates from *NI*. Then $I=I_1\cup\ldots\cup I_k$.

<u>Proof:</u>

If Node Selection is sufficient, then the result follows trivially from the algorithm, since Node Selection consists precisely of recruiting nodes that each discriminate *NI* from some subset of *I* until the union of these subsets is *I*. Suppose then that Node Addition is necessary.

Let $G_1,\ldots,G_j$ ($j<k$) be the growth nodes resulting from Node Selection alone, and $I'=I_1\cup\ldots\cup I_j$. We must show that $G_{j+1},\ldots,G_k$ resulting from Node Addition discriminate *NI* from $I''=I - I'$, or similarly that $I''=I_{j+1}\cup\ldots\cup I_k$. For each instance in I'', there exists at least one variable which is the complement of one of the variables in *NI* (otherwise the current instance set would be inconsistent). One such variable is chosen for each instance in I''. Pairs of these selected variables serve as inputs to $G_{j+1},\ldots,G_k$ which are set to the AND or OR function to guarantee that their output is either 1 or 0 when both inputs are matched and the complement when either one is not matched. Hence, each added node will output one value for *NI* (one of the inputs is not matched) and the complement for the instances of I'' from which its input variables were selected (both inputs are matched). Therefore, $G_{j+1},\ldots,G_k$ discriminate *NI* from subsets $I_{j+1},\ldots,I_k$, respectively, of I''. Since a variable is selected from each one of the instances in I'', it follows immediately that $I''=I_{j+1}\cup\ldots\cup I_k$.

<u>Lemma 2:</u>

Let G_1 and G_2 be two growth nodes other than *CTN* and *OSDN*. Let I_1 and I_2 be the sets of current instances that are discriminated from *NI* by G_1 and G_2, respectively. Let *PN* be the parent node created when G_1 and G_2 are combined. Then *PN* discriminates *NI* from $I_1\cup I_2$.

<u>Proof:</u>

G_1 and G_2 output either 0 or 1 for *NI* (? is not possible, otherwise they would not discriminate *NI* and thus could not be growth nodes). Since $PN\neq NTN$, it does not matter whether *PN* outputs 0 or 1 for *NI*. Hence, there are eight possible functions for *PN*, summarized in [2, Table 6.2]. We give the complete proof for the case where both G_1 and G_2 output 1 for *NI*. Then, the function of *PN* is the AND of the outputs of G_1 and G_2. Now, because G_1 discriminates *NI* from the instances in I_1, G_1 outputs 0 for every instance in I_1. Similarly, G_2 outputs 0 for every instance in I_2. Applying the AND function pairwise to G_1 and G_2 will thus cause *PN* to 1) output 1 for *NI* and 2) output 0 for every instance in I_1 and every instance in I_2. It follows that *PN* discriminates *NI* from $I_1\cup I_2$.

<u>Theorem 1:</u>

Let $G_1,\ldots,G_k$ be the growth nodes resulting from Node Selection and Node Addition. Let $I_1,\ldots,I_k$ be the respective sets of instances that each G_i discriminates from NI. Let *PN* be the node obtained after the G_i's have combined. Then *PN* is *OSDN*.

<u>Proof:</u>

Each node that is the result of combining two growth nodes is labeled as a growth node and can therefore participate in combination with other nodes. Any pair of growth nodes may combine. Hence, the combination process happens finitely many times (exactly k-1 combinations) and halts when there is a single growth node left, namely *PN*. By induction on the result of Lemma 2, *PN* discriminates *NI* from every instance in $I_1\cup\ldots\cup I_k$. But, from Lemma 1, we have $I=I_1\cup\ldots\cup I_k$. It follows that *PN* discriminates *NI* from every instance in *I*, and hence, *PN* is *OSDN*.

Theorem 2:

NTN is a complete discriminant node such that AA1 fulfills the new instance set.

Proof:

There are only four ways, depending on the polarity of *NI* and the output of *OSDN* for *NI*, that *OSDN* and *CTN* may combine to produce *NTN*, as summarized in [2, Table 6.3]. We give the complete proof for the case where *OSDN* outputs a 1 for *NI* and *NI* is positive. Then *NTN*'s function is the OR of the outputs of *CTN* and *OSDN*. Since *OSDN* outputs 1 for *NI* and *NI* is positive, the N column of *OSDN* contains only 0's. Since *CTN* fulfills the old instance set, the N column of *CTN* contains only 0's. Hence applying the OR function pairwise to the N columns of *CTN* and *OSDN* will produce the N column of *NTN* that contains only 0's. Similarly, the P column of *CTN* contains 1's in every cell, except the one corresponding to *NI*. But by assumption the cell corresponding to *NI* contains a 1 in the P column of *OSDN*. Hence applying the OR function pairwise to the P columns of *CTN* and *OSDN* will produce the P column of *NTN* that contains only 1's. Consequently, *NTN* is a complete discriminant node, and since *NTN* outputs 1 for positive instances and 0 for negative instances, AA1 fulfills the new instance set.

Theorems 3, 4 and 5 show that AA1 always fulfills the instance set after any one of the three self-deletion procedures is applied. It follows then from these and the above that AA1 is correct.

Remark:

A node's output cannot be inverted in AA1; however, since only the AND and OR functions are used, Boolean logic guarantees that an equivalent result can be obtained by inverting the node's inputs and changing the node's function from OR to AND, or vice versa. The entries in the NT can then simply be inverted (note that ? is its own inverse).

Theorem 3:

Complete discriminant deletion is correct.

Proof:

Let *CN* be a complete discriminant node, other than the top node. The only nodes needed to build up to *CN* are those that are in the directed graph rooted at *CN*. All the other nodes can be removed and the result follows from the above remark, since *CN* can always be made to fulfill the instance set.

Theorem 4:

Non-discriminant deletion is correct.

Proof:

This result follows from the fact that if *NN* is a non-discriminant node, *DN* a discriminant node, and *PN* the parent node resulting from combining *NN* and *DN*, then *PN* always discriminates a subset of *DN* and can thus be replaced by it. We prove this fact for non-inverted inputs. There are three cases.

Case 1: *NN* contains only one value (0 or 1). It is clear then that *PN* is either *NN* or *DN*. So *PN* discriminates a subset of *DN* (i.e., either nothing at all or the same as *DN*).

Case 2: *NN* contains two values ((0 and ?) or (1 and ?)). There are three subcases: a) *PN*'s function is AND and *NN* contains 1s and ?s or *PN*'s function is OR and *NN* contains 0s and ?s, b) *PN*'s function is AND and *NN* contains 0s and ?s, c) *PN*'s function is OR and *NN* contains 1s and ?s. In case a), *PN* is *DN* with some 0s or 1s possibly replaced by ?, and thus *PN* clearly discriminates a subset of *DN*. In case b), *PN* is *NN* with some ?s possibly replaced by 0, hence *PN* contains only 0s and ?s and is thus a non-discriminant node. So *PN* trivially discriminates a subset (namely the empty one) of *DN*. Case c) is similar to case b), but *PN* now contains only 1s and ?s. In all three possible subcases, *PN* discriminates a subset of *DN*.

Case 3: *NN* contains all three values (0, 1 and ?). Assume that *PN*'s function is the AND function (the case OR is complementary). The 0s and 1s must be in the same column of *NN* and the other column must contain only ?, otherwise we would be in Case 2 (or *NN* would be a discriminant node). Without loss of generality, assume that the N column contains only ?. Then the N column of *PN* will have 0s wherever *DN* had 0s and ? everywhere else. Let p be a positive cell containing 1 in *DN* and I_p the set of discriminated discordant cells (i.e., cells containing 0 in the N column). The p cell of *PN* contains 0, 1 or ?, and each cell of I_p contains 0. Thus *PN* discriminates a subset of *DN* for

p (i.e., the same or nothing). If p contains 0 in *DN*, it also contains 0 in *PN*. However each cell of I_p contains ? in *PN* so *PN* no longer discriminates p. Thus, *PN* discriminates a subset (namely the empty set) of *DN* for p. The above applies to any cell p of *DN*, hence *PN* discriminates a subset of *DN*.

Theorem 5:

Locally redundant deletion is correct.

Proof:

Both cases of locally redundant deletion applicable to AA1 are essentially special cases of non-discriminant deletion. Therefore, this theorem follows almost immediately from Theorem 4.

4. Discussion

Inconsistency

One important aspect of real-world training sets is their inherent inconsistency. There are at least two sources of inconsistency in a training set. The first is due to the fact that in real-world situations the training set is obtained from experimentation and thus may contain errors. The second one has to do with the fact that certain applications may not be characterized as functions, but rather as distributions (e.g., the task of classifying objects into overlapping classes).

A very important ability, inexistent in most training set learners, but found in AA1, is the ability to use general rules (short-hand instances) in learning. This allows one to construct an instance set that reflects human learning by first presenting general rules to the system, followed by exceptions. It has long been a criticism of neural networks, that they can not incorporate general rules, and thus are very limited models of human learning. AA1 handles both very specific examples and very general rules. A side-effect of such ability, however, is the introduction of inconsistencies (in the form of exceptions) in the instance set.

In AA1, inconsistency is solved by giving precedence to the newer instances. This simple scheme, though reasonable, has several disadvantages. It requires a smart teacher that knows something about the target function and can thus give the instances in the correct order. Also, it is not necessarily best to try to maintain the instance set consistent (nature is replete with inconsistencies). Current research seeks to expand the learning scheme by removing AA1's rigid precedence given to newer rules and by allowing inconsistencies to subsist (see for example [1]).

Complexity

The hierarchical architecture of AA1 guarantees that execution is O(log$n$) where n is the number of nodes in the network. During learning, note that only the training instance requires full execution of the learning algorithm. Modified instances in the add-list are still fulfilled and will therefore not cause node selection, addition and combination. Learning is bounded by a polynomial function of the number of instances in the instance set.

Generalization

In Section 3, we proved that AA1 converges on any arbitrary Boolean instance set. A more interesting aspect of AA1 has to do with generalization, or its ability to induce new knowledge from past experience. The instance set consists of only a subset of the complete function to be learned, and AA1 must guess the remainder of the function. Even though there is no guarantee that the final network is smallest, AA1's bias towards simplicity results not only in increased parsimony, but also in the ability to generalize. This bias is manifest in two ways. First, for each new instance, the output of the current network is checked and when found concordant no changes are made to the network. Second, after a new instance has been processed, unnecessary nodes are deleted. Empirical studies confirm these findings.

The originality of AA1 lies in the fact that it does not generalize on the basis of proximity of the new input to one of the learned instances, but rather on the basis of how well the network is able to discriminate the new input from learned features of the opposite class.

Extensions

Several extensions can be made to AA1. For example, the greedy search used in Node Selection, as well as the choice of inputs to be used to create new nodes in Node Addition, could be enhanced by heuristics. Another important problem to be addressed is AA1's memory requirement at each node. It is clear that each node need not store a D column in its node table. A global D column would be sufficient to meet the goal of Node Selection. However, it is still unclear how to do away with (or effectively replace) the P and N columns.

5. Simulation Results

AA1 was tested on several real-world applications, drawn from the Irvine machine learning database [8]. No minimization was attempted, and self-deletion was limited to complete discriminant deletion. The results are summarized in the table below. For Congressional Voting Data and Hepatitis, the results are averages (on the test set) over 10 runs of AA1. In each run, the instance set and the test set were regenerated with 9/10 of the data used for training and 1/10 used for testing.

Congressional Voting Data	92.4%
Monk1	98.6%
Monk2	82.6%
Monk3 (with noise)	91.2%
Monk3 (without noise)	99.1%
Hepatitis	70.3%

6. Summary

This paper presents a formal review of Adaptive Algorithm 1 (AA1) of Adaptive Self-Organizing Concurrent Systems (ASOCS). After a brief overview of AA1 and a description of the algorithm, a detailed proof of correctness for AA1 is given. A series of lemmas and theorems that capture the actions of the algorithm are stated and formally proved, thus showing that AA1 is guaranteed to converge on any arbitrary Boolean instance set. The question of inconsistency and AA1's solution to it are discussed. Intuitive reasons for AA1's ability to generalize are briefly outlined. Extensions to AA1 are suggested. Results of simulations of AA1 on real-world data sets are also reported and show that AA1 has promising generalization performance.

References

[1] Giraud-Carrier, C. A Precept-Driven Learning Algorithm. Master's Thesis, Brigham Young University, Department of Computer Science, 1993.

[2] Martinez, T.R. Adaptive Self-Organizing Networks. Ph.D. Thesis, UCLA Tech. Rep. - CSD 860093, 1986.

[3] Martinez, T.R., and Vidal, J.J. Adaptive Parallel Logic Networks. J*ournal of Parallel and Distributed Computing*, 5, 1 (Feb. 1988), 26-58.

[4] Martinez, T.R., and Campbell, D.M. A Self-Adjusting Dynamic Logic Module. *Journal of Parallel and Distributed Computing*, 11, 4 (1991), 303-313.

[5] Martinez, T.R., and Campbell, D.M. A Self-Organizing Binary Decision Tree for Incrementally Defined Rule Based Systems. In *IEEE Transactions on System, Man, and Cybernetics*, 21, 5 (1991), 1231-1238.

[6] Martinez, T.R., Hughes, B., and Campbell, D.M. Priority ASOCS. To appear in *Journal of Artificial Neural Networks*, 1993..

[7] Minsky, M., and Papert, S. *Perceptrons*. MIT Press, Cambridge, MA (1967).

[8] Murphy, P.M., and Aha, D.W. *UCI Repository of machine learning databases*. Irvine, CA: University of California, Department of Information and Computer Science, 1992.

[9] Rosenblatt, E. The Perceptron: A Probabilistic Model for Information Storage and Organization in the Brain. *Psychological Review*, 65, (1958), 386-408.

[10] Rudolph, G. A Location-Independent ASOCS Model. Master's Thesis, Brigham Young University, Department of Computer Science, 1991.

[11] Rumelhart, D. and McClelland, J. *Parallel Distributed Processing*. MIT Press, Cambridge, MA (1986), Vol. 1.

[12] Werbos, P.J. Beyond Regression: New Tools for Prediction and Analysis in the Behavioral Sciences. Ph.D. Thesis, Harvard University, Cambridge, MA, 1974