A Distributed Problem-Solving Approach to Inductive Learning

Michael J. Shaw*
Riyaz Sikora**

CMU-RI-TR-90-26 2

*Robotics Institute, Carnegie Mellon University; on leave from The Beckman Institute, University of Illinois at Urbana-Champaign
** The Beckman Institute and Department of Business Administration, University of Illinois at Urbana-Champaign

November, 1990

Contents

List of Figures

List of Tables

Abstract

In this paper we propose a distributed approach to the inductive learning problem and present an implementation of the Distributed Learning System (DLS). Our method involves breaking up the data set into different sub-samples, using an inductive learning program (in our case PLS1) for each sample, and finally synthesizing the results given by each program into a final concept by using a genetic algorithm. We show that such an approach gives significantly better results than using the whole data set on an inductive learning program. We then show how DLS can be generalized to incorporate any learning algorithm and present some of the implications of this approach to DAI (Distributed AI) systems in general and learning methodologies in particular. Complexity analysis further shows that the time complexity of DLS can be made linear with respect to the size of the problem (data set) irrespective of the time complexity of the learning algorithm it uses.

1.Introduction

Distributed Problem-solving (DPS) is becoming an increasingly important means for computing in many different domains (Decker[1987]). This phenomenon can be attributed to many factors such as increasing complexity of problems which transcend functional boundaries, increasing trend towards specialization of skills in narrow functional areas, and the recent advances in processor fabrication and communication technologies, which provide incentives for employing multiple processors. In this paper we present a DPS method, called Distributed Learning System (DLS), for performing the task of inductive learning.

The problem of rule learning or induction from examples is a very widely studied problem in the area of machine learning. Algorithms like Version-spaces (Mitchell[1977]), AQ(Michalski[1983]), ID3(Quinlan[1986]), and PLS1(Rendell[1986]) are a few of the successful algorithms for learning from examples. All these algorithms operate on the complete data set to find the concept (or rules) explaining them.

Applying the DPS approach as a new learning strategy, we consider the distributed (or multi-agent) approach to learning from examples in which the data set is divided into different sub-sets and given to different agents (inductive learning programs). The learning results from the agents are then synthesized into the final solution. We will show that the DPS approach produces better results than using the whole data set on a single agent . The DPS approach uses the architecture of the double-layered learning system (Sikora & Shaw[1990]) in which the problem (data set) is decomposed into several sub-problems (sub-samples) which are then given to PLS1 programs; the solutions given by all the PLS1 programs are then synthesized using a Genetic Algorithm (GA). Using the DPS as a metaphor for the learning process, each PLS1 program can be thought of as an 'agent' and the GA can be thought of as playing the role of synthesizing the 'local solutions' generated by the agents into a 'global solution'.

There are numerous issues involved in the design of any distributed system, among which the following are found relevant for the particular type of learning problems considered in this paper.

- *Problem decomposition and task allocation among the agents* : in what way should the problem (data set) be decomposed for best results?
- *Number of agents* : does the performance depend on the number of agents used?
- *Diversity of individual agents* : is it helpful to have diverse concepts generated by the agents or does homogeneity help?

We tackle each of the above issues and present the computational results which show that (i) the multi-agent approach gives significantly better results than using the whole data set on a single PLS1 program, (ii) the best performance is obtained when the amount of overlap among the different samples (i.e., task overlap among agents) is minimum, (iii) the performance to a certain extent does depend on the number of agents (or sub-samples) used but (ii) above seems to hold in general, (iv) the best performance is obtained when the concepts generated by different agents are more diverse, and (v) the improvement in performance is mainly because of the multi-agent approach and not simply because of combining two algorithms (i.e., PLS1 + GA).

The rest of the paper is organized as follows: in § 2 we give a brief introduction to Distributed Problem-Solving in general together with motivation for using distributed approach for inductive learning, in § 3 we give a detailed description of the Distributed Learning System (DLS) used together with the terminology and the criteria to judge the performance of the system, in § 4 we present an example which shows in detail the advantage of distributed (or multi-agent) approach over single-agent approach, in § 5 we look into each of the above mentioned issues in detail and present the empirical results which suggest possible solutions for the issues considered, in § 6 we present the implications of the Distributed Learning System to the DAI systems in general and to learning methodologies, and finally in § 7 we conclude.

2. Distributed Problem-Solving
2.1 Review

Distributed Problem-solving (DPS) concerns with how the solving of a particular problem can be divided among different modules (or 'agents' in a multi-agent systems) that cooperate at the level of dividing and sharing knowledge about the problem and about the developing solution (Lesser & Corkill[1987]; Smith & Davis[1981]).

There are four phases of the DPS approach: (1) problem decomposition, (2) sub-problem distribution, (3) sub-problem solution, and (4) answer synthesis.
Figure 2.1 below shows the phases of distributed problem solving (Smith & Davis [1981]).

Insert fig. 2.1 here

Bond & Gasser (1988) identify several issues related with the design of any Distributed AI system, which can help provide a framework for understanding the basic components of our DPS method.

(1) **Description, Decomposition, Distribution, and Allocation of Tasks**: Problem description refers to the formulation of the problem or the representation used for the problem. Decomposition refers to the question of breaking up the problem into sub-problems which can be solved by the agents. Decomposition choices are usually dependent on how the problem is described. In some circumstances, the issue of *redundancy* among the sub-problems enters into the decomposition problem. Choices about redundancy are related to the tradeoffs between efficiency and reliability; redundancy should be eliminated to improve efficiency, but may be necessary for reliability. Allocation of tasks refers to the problem of deciding which sub-problems to assign to which agents.

(2) **Interaction, Language, and Communication**: The problem of interaction, language, and communication is considered important because it makes it possible for the agents to combine their efforts. However, several questions such as: what kinds of interactions are possible? what is the result of this interaction? What kinds of communication are possible? etc., arise in the design of a DPS system.

(3) **Coherence and Coordination**: Coherence refers to how well the system behaves as a unit, along some dimensions of evaluation. Typical dimensions of evaluation could be solution quality, efficiency, clarity or conciseness of the final solution etc. Coordination is a property of interaction among some set of agents performing some collective activity. Coordination and coherence are partially related - better coordination may lead to greater efficiency coherence. However, good local solutions by the agents do not necessarily add up to good global behavior, because good local solutions may have unfortunate global effects. This is especially important in the context of inductive learning where the local solutions generated by the agents can be local optima. We will see how DLS overcomes this problem of local vs. global optimum.

(4) **Modeling other agents and Organized Activity** : This concerns with the knowledge each agent has about what other agents are doing and what other agents know, so that they can organize their activities. The main question concerning this is what knowledge and how should the knowledge of other agents be represented and organized? Typical types are knowledge of agents capabilities, resources, demands, beliefs, goals, plans etc.

(5) **Interagent disparities: Uncertainty and conflict** : This concerns with the ability of the agents to cope with problems of disparity and uncertainty between their objectified representations and the affairs to which the representations refer. There is also the important question of conflict resolution between the agents. Negotiation is often proposed in DAI research as a conflict-resolution and information-exchange scheme (see for e.g., Davis & Smith [1983]).

(6) **Synthesis of results** : This concerns with combining the solutions generated by the individual agents (often partial and incomplete) to form the complete solution of the problem. As we will see this is the critical step in the DLS because the individual solutions generated by the agents are usually local optimum and have to be combined to get a globally optimum solution.

To illustrate these issues and relate them to more realistic settings, two widely discussed DPS systems can serve as examples; namely, the group decision support systems (GDSS) and the contract net system.

GDSS's primary goal is to see that the meetings convened by the group of managers for decision making are conducted productively. Productivity loss in group activity results mainly from information loss, information distortion and/or making decisions without sufficient alternatives to consider (Kramer & King [1988]).

Electronic Meeting Systems (EMS) typify a practical application of GDSSs. In EMS, agents solve the problem in a series of explore-and-form-consensus cycles. In the 'explore' part of the cycle each agent searches its own knowledge-base for applicable rules and transforms the problem to a new intermediate state, representing an individual partial solution. In the 'form-consensus' part of the cycle, agents compare their individual partial solutions and arrive at a group partial solution representing agreement or consensus. Each agent now discards its individual partial solution and uses the group partial solution as the starting point for the 'explore' part of the next cycle. This method is efficient only when there is a high degree of overlap between individual agent's knowledge-bases. When overlap is not high, consensus formation essentially involves a few agents (those with high overlap) while the other agents remain passive participants. This leads to poor utilization of the group for problem solving since the passive agents are distracted from their individual focus by the consensus formation activity.

Contract Net was a distributed problem-solving system designed by Randall Davis and Reid Smith (Davis & Smith[1983], Smith[1980], Smith & Davis[1981]). The primary goal of this system was opportunistic, adaptive task allocation among a collection of problem solvers, using a framework called "negotiation", based on task announcements, bids, and awarded "contracts". In this, an agent needing help (called the manager) with a

task can divide it into subtasks and negotiate a contract for each subtask with the other agents (called contractors). The manager for a task makes a task announcement giving a description of the task and eligibility requirements of the agents who may bid for the task. The manager - contractor relationship ceases to exist when the task is completed. The contract net protocol thus dynamically decomposes problems, allocates the tasks, organizes agents in heirarchies for the purposes of control in achieving these tasks, and disbands the heirarchies once the tasks are complete.

2.2 Motivation for applying DPS to Inductive Learning

Traditional work in inductive learning from examples, for the most part, has been set in the context of single agent. In other words, a given problem data set is used by a single inductive learning program to induce a hypothesis. The major motivation for distributed approach to learning lies in the potential it offers for making available more problem solving power, by applying a collection of intelligent agents to the solution of a single problem. It may, for example, be more efficient to use 5 inductive learning programs, each working in a cooperative fashion, on one fifth of the problem, than using a single program on the whole problem, provided there is a way to synthesize those solutions and provided the final solution quality does not suffer. This is especially true in real world applications which involve processing huge amount of data like stock market analysis, chemical process control, effect of product attributes on consumer choices etc., to name a few. Recent advances in parallel processing point to the potential for executing such a distributed learning program in parallel.

There has been a major effort in using distributed approach for inductive learning in the form of *connectionist* research, using neural net modelling where the computational structures used are layers of "neurons" (small processing units) interconnected with weights. However, one major difference between the connectionist approach and DPS approach to inductive learning is the grain size of the processing units. DPS approach addresses the problems of designing and analyzing large-grained coordinated intelligent systems whereas Connectionist research is devoted to explaining higher-level reasoning processes by reference to highly parallel collections of processes made up of very simple computing elements.

Another source of inspiration for DPS approach to induction has been the work of

Laughlin on Collective Induction (CI) (Laughlin [1983,85,88]). In CI, the task assigned to the human group is to induce a concept description from positive and negative exemplars of the concept. The experiment begins with a positive exemplar being presented to the group. Each member then comes up with a concept description which is consistent with this exemplar (individual hypothesis). This is then compared with those of the other group members and by a process of discussion, a consensus is reached on what the concept description should be (group hypothesis). This is then tested in the light of new exemplar and the cycle of problem solving is repeated for a fixed (predetermined) number of times. CI is thus an incremental process where successive refinement of individual and group solutions takes place in response to new data about the problem. Thus, CI uses the explore-and-form-consensus approach but does so after each agent has solved the entire problem.

All these developments indicate that it may be advantageous to employ multiple agents in problem-solving situations. In this paper we use the rule-induction task to illustrate: (1)how a DPS approach can be applied to rule induction, (2) how does the approach compare to other methods for the same task, and (3) what are its performance characteristics.

3. The Distributed Learning System
3.1 Design

As mentioned before, in the distributed approach of problem solving the important steps are: (i) problem decomposition, (ii) task allocation to the individual agents, (iii) problem solving by the individual agents, and (iv) synthesis of the solutions given by the individual agents. Before discussing the DLS system, however, we first define the terminology which we will be using in evaluating the performance of the DLS approach.

Problem decomposition is achieved by dividing the original data set into sub-samples by using the jackknife technique (Efron[1982]). In jackknife technique one or more data points are randomly removed from the data set to obtain a sub-sample. We use the jackknife technique of 'leave-out r ' which we define below:

Definition 3.1 : A random sub-sample from the original data set is obtained by the jackknife technique of 'leave-out r ' ($0 \leq r < 1$) if each example in the original data set has a

probability of (1- r) of being included in the sub-sample.

Since one of the issues is deciding what kind of decomposition is best, we define a decomposibility index d , which also measures the amount of overlap between subsamples.

Definition 3.2 : The decomposibility index d , for a particular problem decomposition, is defined as the ratio of average size of each sub-problem to the size of the original problem, i.e.,

d = (average size of sub-problem) / (size of the original problem)

In this case the size of the problem corresponds to the number of the examples.

Note that $0 < d \leq 1$, the value of 1 corresponding to the extreme where the whole problem is allocated to all the agents. Also, for a jackknife technique of 'leave-out r ', the size of a sub-problem is a binomial random variable with parameters (N, 1- r), where N is the size of the original problem. Therefore,

$$d = N * (1- r) / N = (1- r).$$

As mentioned before, the decomposibility index d also defines the amount of task overlap (or redundancy as mentioned in § 2.1) among the agents. Specifically,

Definition 3.3 : For a given decomposibility index d and the number of agents n , the total amount of task overlap, t_o among the agents is given by

$$t_o = (d * n - 1) N,$$

where N is the size of the original problem.

Note that there is no task overlap, i.e., $t_o = 0$, when $d = 1/n$.

In order to compare the concepts generated by the DLS system based on their generality, we define a generality index g as follows:

Definition 3.4 : For any given concept or rule C, the generality index g is defined as the ratio of number of unseen examples it can cover to the number of all possible examples. In other words,

g = fraction of the instance space covered.

Thus, a concept C_1 is more general than C_2, if $g_1 > g_2$.

3.2 A Distributed Problem-Solving Approach to Learning

The Distributed Learning System for rule learning is shown in fig. 3.1 below. We discuss each of the issues mentioned in § 2.1 as they relate to the DLS.

Insert fig. 3.1 here

The above figure illustrates the different steps in the distributed problem solving approach as mentioned before.

Decomposition: The problem decomposition step consists of breaking the original data set into n random sub-samples using the jackknife technique. The question of redundancy also arises here in the form of: what decomposibility index is best? As mentioned in § 2 we would like to minimize redundancy to improve efficiency but at the same time not reduce the solution quality. As we will see in the empirical results, the best performance is obtained when the redundancy (or task overlap) is minimum. There is also the factor of number of agents. In most of the DPS systems this is not considered a factor, however as we show later the performance of the DLS does depend on the number of agents used.

Allocation of tasks: Each generated sub-sample is allocated to an agent (inductive learning program), in this case PLS1. Allocation of tasks is not a issue here since all the agents are similar. However, an alternative view of this approach would be to view 'tasks as agents' instead of 'resources as agents', in which case the allocation of resources to tasks is uniform since each task (subsample) gets a PLS1 program (resource).

Coherence: Coherence of the DLS, i.e. how well DLS behaves as a unit, is measured in terms of solution quality, efficiency, and the conciseness of the final solution.

Interaction and coordination : Since there is no interaction among the agents in DLS once the subproblems are distributed, hence the traditional issues of a DPS system like interaction, language, communication, conflict, modeling other agents etc. do not hold here.

Solution synthesis: The individual results of all the n agents are taken as input by the genetic algorithm which synthesizes all the solutions and gives the final concept Ω. The synthesis step is the most critical mainly because of two reasons. First, the concepts given by the different agents are themselves complete rather than just being separate parts which

can be added together. In other words, synthesis step does not involve just taking the union of the individual agent results but involves combining the important features of each agent's result to form a concept which is more accurate and concise than each of the individual results. This is thus similar to the formation of 'group hypothesis' from 'individual hypothesis' in Collective Induction, as mentioned in § 2.3. Second, the concepts generated by the agents (PLS1 programs) have to be made compatible with the representation used by the GA (see appendix A for a review on genetic algorithm). We, therefore, describe the synthesis step of the DLS system in detail in the next section.

3.3 Concept Synthesis

In order to understand the synthesis step we should first understand the representations used by the PLS1 programs and that used by the GA (which is similar to PLS1).

3.3.1 Representation used by PLS1 and GA

The concept given by PLS1 is represented in the form of regions (Rendell[1986]), where each region is represented by conjunction of intervals for each attribute. In other words, the concept given by PLS1 can be represented by

$$P = P_1 \vee \ldots \vee P_m$$
$$= \{ (d_{11} \le x_1 \le e_{11}) \& \ldots \& (d_{1k} \le x_k \le e_{1k}) \} \vee \ldots \vee \{ (d_{m1} \le x_1 \le e_{m1}) \& \ldots \& (d_{mk} \le x_k \le e_{mk}) \}$$
$$= \{\xi_{11} \& \xi_{12} \& \ldots \& \xi_{1k}\} \vee \{\xi_{21} \& \xi_{22} \& \ldots \& \xi_{2k}\} \vee \ldots \vee \{\xi_{m1} \& \xi_{m2} \& \ldots \& \xi_{mk}\}$$

where each P_i corresponds to a region and is represented as conjunction of intervals ξ_{ij} corresponding to the attribute x_j, $j = 1,2,\ldots k$.

The representation used by GA is similar, except that instead of using the whole concept P as a single member in the population (for reasons explained in §3.3.2), it uses each region as an individual member. Thus, the concepts generated by the PLS1 programs are first broken into individual disjuncts and then given to the GA as initial population.

3.3.2 Synthesis Step

There are two ways of synthesizing the concepts generated by the PLS1 programs using a GA. One way is to let each member in the population represent a complete concept ($P_1 \vee$

P_2 v ... v P_m) generated by each PLS1 program.

Alternatively, the other method is to let each member be just a single disjunct $P_i = (\xi_{1,i}$ &... & $\xi_{k_i,i})$. Since each PLS1, working on a different sample, gives a concept of different length (number of regions), hence using the first method results in a population for the GA wherein each member has a different length. Also, since we want to get a more concise concept, we would like to get more powerful disjuncts (or rules). Hence, to avoid the problem of variable length of each member and to make the disjuncts compete against each other, we use the second method in our GA. In this method, the GA tries to find the best possible disjunct (i.e., covering as many positive examples as possible) by recombining the disjuncts given by the PLS1 programs. After it converges, the best disjunct found is retained and the positive examples which it covers are removed. The process is again repeated to find a new disjunct to cover as many of the remaining positive instances as possible. This process terminates after all the positive instances are covered. The final rule is then the disjunct of all the disjuncts found.

3.3.3 Evaluation function used by GA

The goal of the ideal learning system would be to obtain a concept which covers all the positive examples without covering any negative examples. However, when dealing with real world data there may be noise associated with the data, in which case the above requirement of not covering any negative examples has to be relaxed. Also, in this specific case, the positive and negative examples actually correspond to two different classes which are complement of each other. Therefore, when the learning program gives a concept C for the positive examples, we would also want C^* (complement of C) to be the concept for the negative examples. Thus a concept C in this case actually represents two different mutually exclusive classes and therefore there are two accuracy terms associated with C. Consequently, the evaluation function should be based on maximizing both the accuracy terms.

Let P - total number of positive examples,
N - total number of negative examples,
p - number of positive examples covered by the concept, and
n - number of negative examples covered.

As mentioned, the evaluation functions can be derived based on the optimization

formulation of maximizing the two accuracy terms. The multi-objective function is reduced to a single objective function by taking a convex combination of the two functions.

The two accuracy terms can be defined as:

$$F_1 = \frac{p}{P} \text{ and } F_2 = \frac{N - n}{N}$$

Let the objective function be a convex combination of F_1 and F_2

$$\text{max. } F = \alpha . F_1 + (1 - \alpha). F_2$$

where $0 \leq \alpha \leq 1$.

Let $\alpha = P / (P+N)$, so that the two accuracy terms get weightage which is proportional to their class representativeness.

The objective function then becomes:

$$F = \frac{1}{P + N} . [p + N - n]$$

Multiplying by the constant (P+N) we get the fitness function F' as:

$$F' = p + N - n, \; F' \in [0, P+N]$$

4. Advantages of Multi-agent Approach over Single-agent: an Example

For the empirical analysis, a large real world data set from a chemical plant was used for all the experiments discussed in this and next section. The problem concerned with controlling a chemical process, producing a certain chemical product, with about 30 process variables. In the process of producing the product an undesirable byproduct was produced which was not measured directly. To remove this byproduct an expensive chemical was added in just sufficient quantities to chemically remove the byproduct from the product. The problem was to change the controllable process variables (9 out of the 30 variables) so that the usage of the expensive chemical was minimized. Since there are no theoretical formulae linking the process variables with the amount of product produced, the only way to solve this problem is to induce the relationships based on a set of actual plant readings. The problem was formulated as a single concept learning problem by considering the examples corresponding to large quantity of the expensive chemical used as being positive examples

and the rest as negative examples.

The data set had 572 instances of which 355 were positive and 217 negative examples and it was randomly broken up into a training set of 458 and a testing set of 114 examples. In this section we present an example to show the advantage of using a multi-agent (distributed) problem solving approach to inductive learning vis-a-vis that of using a single-agent approach.

The following parameter values were used for the DLS: n=5 and d=0.2. We present the detailed empirical results for different values of n and d in §5 where we show that best performance is obtained for these parameters values. Figure 4.1 shows the learning result, in the form of concept description, of the single-agent approach, the results of the individual agents in multi-agent approach and the final results in the multi-agent approach obtained by synthesizing the results of individual agents. We explain in detail each part of the computer output below.

Insert Figure 4.1 here

4.1 Single-agent results

In this sub-section we discuss the results obtained by using the single-agent approach to inductive learning where we used the whole training set of 458 instances on the PLS1 program (with s=1)[1]. As can be seen from Figure 4.1, the rule-size of the concept given by the single-agent approach was 17. The concept is therefore (r_1 V r_2 V ...V r_{17}) where each rule r_i contains the interval ranges for the 9 variables (the complete range is 0-63) followed by the number of positive and negative examples covered from the training set and the number of positive and negative examples covered from the testing set. For example, the first rule says that if the 9 variables are within their respective intervals then it correctly explains 176 of the 276 positive examples from the training set, at the same time wrongly covers 12 of the 182 negative examples, and correctly predicts 50 of the 79 positive examples from the test set while wrongly predicting 6 of the 35 negative examples. Thus, the result from single-agent approach is: (prediction) accuracy = 83.3% and rule-size = 17.

---

[1] s is the significance level used by the PLS1 program which corresponds to the approx. noise level in the data.

4.2 Multi-agent results

Figure 4.1 also shows the results given by the 5 individual agents, each working on 20% of the training data set and the final result given by the DLS, which takes the results given by the 5 agents and synthesizes them using GA. Table 4.1 shows the prediction accuracy and rule-size of each individual agent and that of the final result given by the DLS, together with the generalization index g of the best rule from each of them.

Insert Table 4.1 here

As can be seen, the best result given by an individual agent (agent 3) is: accuracy = 79.8% and rule-size = 4, and the final result obtained by synthesizing the 5 results is: accuracy = 86% and rule-size = 3. Thus, the synthesizing step improves upon the individual results given by the agents by a minimum of about 9% in accuracy and 25% in rule-size and is better than the single-agent result by about 3.2% in accuracy and about 82% in rule-size. Also, the g index shows that the solutions generated by the agents are local in scope as their best rule covers about 1% of the instance space on average as compared to about 3% covered by the best rule from DLS. This shows that the synthesis step (using a GA) has the ability of combining the local solutions reach a "group solution" by performing a more globally oriented search. Figure 4.2 summarizes the comparison of performance of the distributed approach vis-a-vis that of a single-agent.

Insert Figure 4.2 here

---

5. Empirical Results
5.1 Experiment details

The same real world data set from the chemical process control problem was used for all the experiments discussed in this section. In each experiment the data set was randomly broken up into a training set of 458 and a testing set of 114 examples. All the results given are the average of 5 runs with a different training and testing set used in each run.

The following parameter values for the GA were used for all the experiments: total number of generations used was 100, Baker's (1987) SUS algorithm was used for selection, uniform crossover operator was used with probability 0.7, and probability of mutation was 0.05. The population size was determined by the results from the PLS1 programs and was usually directly proportional to the value of d for each n.

The Distributed Learning System is implemented in Common Lisp on a TI-Explorer machine.

5.2 Single-agent (PLS1) results

Table 5.1 shows the PLS1 results which are the average of 5 runs. PLS1 program has a parameter called significance level (corresponding to the noise level in the data) which can vary from 0 to 1, and the table contains the results when sig. level s=0 and s=1. The significance level is an optional input for the PLS1 program which takes the default value of 0 if it is not specified. We have also included the results obtained after pruning the PLS1 results, though this is something which is not part of the PLS1 program and is done manually. In pruning the PLS1 results we eliminated all those regions which covered less than 5 positive examples as they were found to be insignificant. However, for the DLS system we used the PLS1 results (for each sub-sample) for s=1 but without pruning and so the improvement brought on by the DLS system is effectively over that of the PLS1 results with s=1 and without pruning.

Insert table 5.1 here

5.3 DLS results

5.3.1 Effect of problem decomposition and number of agents used.

In order to find a suitable answer for the different issues mentioned in §1, we carried out number of experiments to determine the effect of changing the number of agents used and problem decomposition on the coherence of the system as measured by the prediction accuracy, conciseness (as measured by the rule size), and efficiency (as measured by the

CPU time).

Table 5.2 shows the results (all values are average of 5 runs) for different values of n (number of agents) and d (decomposibility index). Figure 5.1 shows the plot of the prediction accuracy results for different values of d corresponding to different values of n, together with that of PLS1. Figure 5.2 shows the plot of rule size for different values of d corresponding to different values of n, together with that of PLS1. Figure 5.3 shows the plot of CPU time (in hrs.) for different values of d corresponding to different values of n, together with that of PLS1.

Insert Table 5.2, Figure 5.1-5.3 here

We can see that for each value of n there is a general trend where the maximum accuracy is mostly achieved when the amount of task overlap is minimum, i.e., around d =1/n. Also, the peak performance when n = 2 and 5 is significantly better than PLS1 in terms of prediction accuracy and rule size, and the CPU time is also comparable. For n=5 and d =0.2 there is an increase in prediction accuracy over that of PLS1 by about 2% and decrease in the rule size by about 82%. However, the prediction accuracy starts to decrease as the value of n is increased and this can be explained by the loss of data representativeness as the sample size decreases. Thus, the performance of DLS depends on the number of agents used with the best performance occurring when 5 agents are used with no task overlap.

The improvement in performance of DLS over that of PLS1 (especially the improvement in rule size) can also be seen by the improvement in the best rule generated by the system. Table 5.3 shows the comparison of the best rule from the concepts generated by DLS and PLS1.

Insert table 5.3 here

The above comparison shows that DLS improves the prediction accuracy of the best rule from 74.2% to 79.7%, an increase of about 7.4% which helps in reducing the average rule size of the concept from 17.4 to 4.2. It can be seen that DLS produces a more general rule (as shown by the g index) and increases the classification accuracy thereby making it more accurate in prediction. This also shows that multi-agent approach using a GA gives a more globally oriented result than the single-agent approach.

5.3.2 Effect of diversity

We also tested the hypothesis that the performance of the system is increased when there is diversity in the concepts generated by individual agents. We carried out experiments for n=5 and d =0.2 wherein we simulated varying the diversity of the concepts generated by the agents. Table 5.4 shows the results (all values are average of 5 runs) of these experiments, where the diversity number refers to the number of different agents used. For example, the diversity number ' 3 (1+1+2)' means that there were 3 different types of concepts used with one concept used twice to get a total of 5 concepts used by the GA. Figure 5.4 shows the plot of accuracy as a function of diversity. As the results show there is a general trend of increasing accuracy as the diversity increases (except for the diversity number 4).

Insert Figure 5.4 and Table 5.4 here

5.4 Discussion

We have presented an implementation of the Distributed Learning System (DLS) and demonstrated that the distributed problem-solving approach to inductive learning gives significantly better results than the traditional approach. We used one specific example of the DLS where the individual agents were PLS1 programs and tested the performance of the system on the real world data set from a chemical plant .

From the algorithmic standpoint, it is plausible to argue that the improvement brought on by DLS over that of PLS1 is basically because it combines PLS1 algorithm with another algorithm (GA), rather than because of the multi-agent (or distributed) approach. In order to verify the hypothesis that the improvement is mainly because of the distributed approach we carried out an experiment wherein we used the whole training data set on the PLS1 program and then applied the GA on the results of PLS1. The result (average of 5 runs) showed that the rule size remained same as 3.2 but the accuracy dropped to 82.1% as compared to 86.9% from DLS (with n=5, d=0.2). Thus, as per our conjecture the improvement is brought on by the effect of applying multiple learning agents.

5.5 The Application Domains

Besides the process control application used in the empirical study here, other applications of this approach can be found in financial problems (Sikora & Shaw[1990]), where three real world data sets for Bankruptcy analysis, Default loan evaluation, and Loan risk classification are used to test this approach with good results. Thus the improvement brought on by the distributed approach has been demonstrated in different real world classification problems and hence it can be considered sufficiently general.

6. Discussion

Though we have presented a specific example of applying distributed approach to the problem of inductive learning, this method can be extended to other domains in general and (distributed) learning in particular. The most important step which can be invariant across the domains would be the use of a genetic algorithm for solution synthesis. Below we present some of its implications to DAI systems in general and Learning methodologies in particular.

6.1 Implications to DAI Systems in general

Two of the important problems in the design of a DAI system are (i) the problem decomposition and (ii) synthesis of the local solutions generated by the individual agents. Though (i) was straightforward in the example presented, since it just involved breaking up the data set, it is not true in general that a given problem can be decomposed in this way. The question of problem decomposibility in general is domain or problem specific and so will have to be assumed given if we are to have a domain-independent system. Thus the most important implication of the DLS system to the DAI approach is a general solution to (ii) i.e., the feasibility of using a GA as a method for synthesizing the local solutions generated.

Since the concept underlying the working of a GA is based on the *building block hypothesis* , wherein the GA tries to locate good building blocks (in terms of schemata, Goldberg [1989]) which can be combined to get a near optimal solution, it lends itself

easily to the synthesis of partial solutions generated by the agents. One of the problems in a DAI approach is the conflict of interest between an individual agent and the system as a whole. In other words, since the agents usually work on sub-problems, they tend to have a local view of the problem and getting an optimum solution based on the local view does not always lead to a globally optimum solution. Hence, what is needed for synthesizing these local solutions is a method which can take these solutions and combine them using a global search. GA has this unique ability to perform a globally oriented search using the building blocks (which in this case are provided by the agents) and so should be the logical choice for synthesizing the solutions. This is also confirmed in the case of distributed learning by the empirical results presented in §5, with the major difference being that the solutions provided by the agents were complete but with local view rather than being partial with local view.

6.2 Implications to Learning Methodologies

6.2.1 Generalizing DLS to include different algorithms

Though in the DLS system presented, a particular similarity-based learning algorithm (PLS1) was used, it is entirely possible to replace PLS1 with any other learning algorithm depending on the requirements. For example, it is possible to use ID3 programs as agents in which case the GA takes in several different trees generated by ID3 programs based on the sub-samples and combines them to give a more concise and accurate tree. Comparisons with DLS and ID3 show that DLS is significantly better than ID3 (Sikora & Shaw[1990]) and hence it is possible to build a more accurate tree generation program combining ID3 and GA.

In fact, it is even possible to use different learning algorithms as different agents in the DLS provided we make sure that the representations used are made compatible with that of the GA. Since the concept of *bias*[2] is an important one in inductive learning and since different algorithms use different biases (either implicitly or explicitly), using different algorithms in DLS as different agents provides a unique approach of using multiple biases. This also has important implications in terms of the solution (hypothesis) quality and efficiency because usually it is not known a priori which bias is suitable for the problem at

[2]Bias is related to the representational language used by a learning algorithm which allows it to constrain the search space of all possible hypothesis.

hand and use of wrong bias can sometimes make the problem unlearnable or very inefficient. Thus combining different algorithms can be an insurance against the above problems.

6.2.2 Time complexity

There is however the problem of time complexity for the distributed approach. In order to fully justify the distributed approach to learning we should not increase the time complexity over and above that of the learning algorithm (say '*A* ') incorporated in DLS. Specifically, it would be desirable to have a linear time complexity for the distributed approach regardless of the time complexity of the learning algorithm used. That this is indeed the case is explained below.

There are basically two parameters on which the time complexity of a learning algorithm depends, one is the number of attributes (or the dimension of the instance space) and the other is the number of training examples. Strictly speaking the complexity should not depend on the number of examples given because the algorithm will only need a certain number of training examples and after it learns the concept it will not use any more examples. However in practice due to the presence of the noise or due to the algorithmic bias it is not possible to find a concept which can explain all the examples correctly and the algorithm needs to use all of the training examples provided, hence the time complexity does depend on the number of training examples used.

Lets consider the effect of these two parameters independently on the time complexity of DLS. First lets assume that the dimension of the instance space is fixed (i.e., the number of attributes is fixed). Suppose that *A* is linear in time complexity with respect to the number of the training examples. Assume that N is the total number of training examples available and n is the number of agents (each using the algorithm *A*) used in DLS. Then the time complexity of *A* is O(N) and that of DLS is O(N+ LP),[3] where L is the length of each member of population in the GA which depends on the number of attributes (and hence is constant) and P is the population size which depends on the number of agents, n (and hence is constant for a given n). Thus the time complexity still remains linear, however in terms of absolute CPU time DLS would be slower than the learning algorithm used by the agents because of the overhead spent in solution synthesis. But there is another important fact about the DLS architecture (and in general about any DAI system) and that is

[3]Since each agent handles a fraction N/n of the data set and since there are total of n agents, hence their total time complexity is N and the time complexity of a GA is LP

the inherent parallel problem solving done by the agents, which in this case is simulated serially. Hence it is possible to reduce the CPU time if we have multiple processors (equal to the number of agents), each doing the job of a single agent (algorithm A). In fact it is even possible that the total time taken by DLS in this case would be less than the time taken by the algorithm A working on the whole data set.

However if the algorithm A is not linear in time complexity then one would expect that DLS would also be not linear. But it turns out that it can still have linear time complexity provided we have the parallel architecture. This is so because with the increase in the number of training examples N, we can also linearly increase the number of agents n (keeping the number of training examples per agent as constant) and hence keep the time taken by the agents same. Then the only increase in time would be due to the complexity O(LP) of the GA, which is again linear in N.[4] Thus, *the time complexity of DLS remains linear with respect to the number of training examples even if the learning algorithm it uses has exponential time complexity* .

The situation is different for the time complexity with respect to the number of attributes, since in this case we cannot increase the number of agents as the number of examples remain same. Hence, *DLS has linear time complexity with respect to the number of attributes only if the learning algorithm used also has linear complexity.* However, the improvement in CPU time can still be realized by the parallel architecture.

7. Conclusions

In this paper we have presented an alternative view of rule learning based on the distributed problem-solving paradigm. We showed the feasibility of such an approach by implementing a distributing learning system, and showed the improvement in performance brought on by the distributed approach.

We hope that this opens up new possibilities in both the fields of machine learning and Distributed AI. For the machine learning area: *(1) it brings the potential of more problem solving power in the form of distributed systems, (2) it is sufficiently general to be applied with any learning algorithm, and more importantly (3) it improves the overall performance of the learning algorithm* . For the Distributed AI community: *(1) it shows the feasibility of using genetic algorithm as a means of synthesizing individual agent solutions, and (2)*

[4]Because P is linear in n, and n in turn is linear in N

hopefully fills the gap in DAI research on collective learning .

References

BAKER,J.E.," Reducing Bias and Efficiency in the Selection Algorithm,"*Proceedings of the Second International Conference on Genetic Algorithms*, MIT, Cambridge, MA, 1987, pp.14-21.

BOND,A. AND GASSER,L., *Readings in Distributed Artificial Intelligence*, Morgan Kaufmann Publishers, Inc., 1988.

DAVIS,R. AND SMITH,R.G., "Negotiation as a Metaphor for Distributed Problem Solving," *Artificial Intelligence*, 20(1): 63-109, 1983.

DECKER,K.S., "Distributed Problem-Solving Techniques: A Survey," *IEEE Transactions on Systems, Man, and Cybernetics*, SMC-17: 729-740, 1987.

EFRON,B., *The Jackknife, the Bootstrap and Other Resampling Plans*., SIAM, Philadelphia, PA, 1982.

GOLDBERG,D.,*Genetic Algorithms in Search, Optimization and Machine Learning*, Addison-Wesley Publishing Co., Inc., Reading, MA, 1989.

KRAMER,K.L. AND KING,J.L., "Computer-Based Systems for Cooperative Work and Group Decision Making," *ACM Computer Survey*, vol. 20, pp. 329-380, 1988.

LAUGHLIN,P.R. AND SHIPPY,T.A., "Collective Induction," *Journal of Personality and Social Psychology* , vol. 45, pp. 94-100, 1983.

LAUGHLIN,P.R. AND FUTORAN,G.C., "Collective Induction: Social Combination and Sequential Transition," *Journal of Personality and Social Psychology* , vol. 48, No. 3, pp. 608-613, 1985.

LAUGHLIN,P.R., "Collective Induction: Group Performance, Social Combination Processes, and Mutual Majority and Minority Influence," *Journal of Personality and Social Psychology* , vol. 54, No. 2, pp. 254-267. 1988.

LESSER,V.R. AND CORKILL,D.D., "Distributed Problem Solving," in Stuart C. Shapiro, editor, *Encyclopedia of Artificial Intelligence*, pp. 245-251. John Wiley and Sons, New York, 1987.

MICHALSKI, R.S., "A Theory and Methodology of Inductive Learning," in Michalski, R., Carbonell, J., and Mitchell, T. (eds.), *Machine Learning* , Tioga Publishing Co., Palo-Alto, CA, 1983.

MITCHELL,T., "Version spaces: A candidate elimination approach to rule learning," in *Proceedings of IJCAI* , pp. 305-310, Cambridge, MA, 1977.

QUINLAN,J.R., "Induction of Decision Trees," *Machine Learning*, 1, pp. 81-106, 1986.

RENDELL,L.A., "A General Framework for Induction and a Study of Selective Induction," *Machine Learning*, 1, pp. 177-226, 1986.

SIKORA,R. AND SHAW,M., "A Double-Layered Learning Approach to Acquiring Rules for Financial Classification," BEBR Faculty Working Paper No. 90-1693, University of Illinois, Urbana, 1990.

SMITH,R.G., "The Contract Net Protocol: High-Level Communication and Control in a Distributed Problem Solver," *IEEE Transactions on Computers*, C-29(12): 1104-1113, December 1980.

SMITH,R.G. AND DAVIS,R., "Frameworks for Cooperation in Distributed Problem Solving," *IEEE Transactions on Systems, Man, and Cybernetics*, SMC-11(1): 61-70, 1981.

Appendix A: Genetic Algorithm

Genetic Algorithms (GAs) are adaptive search algorithms which have the properties of parallel search and ability to locate global maxima without getting trapped in local maxima. They represent a class of general purpose adaptive search techniques which have been used in a wide range of optimization problems. Goldberg(1989), describes GA as search algorithms based on the mechanics of natural selection and natural genetics. They combine survival of the fittest among string structures with a structured yet randomized information exchange to form a search algorithm with some of the flair of human search. In every generation a new set of artificial creatures(strings) is created using bits and pieces of the fittest of the old; an occasional new part is tried for good measure. While randomized, genetic algorithms are no simple random walk, they efficiently exploit historical information to speculate on new search points with expected improved performance.

A GA should be equipped with the following four components for achieving the effect of rule learning:

(1) a chromosomal representation of solution to the problem.
(2) a way to create an initial population of solutions.
(3) an evaluation function that rates the solutions in terms of their "fitness".
(4) genetic operators that alter the composition of solutions during reproduction

In addition, in applying GA, one needs to decide the various values for the parameters that the genetic algorithm uses, such as the population size, the number of generations, and the probability of mutations. As will be seen later in this paper, varying the mutation rate greatly enhances the GA in terms of avoiding a local optimum and facilitating the convergence to a good solution.

Since the GA works with string structures (analogous to *chromosomes* in biological systems), the **hypothesis** (or rules or solutions) should be encoded and represented in a string form. This low level representation with which a GA works is called *genotype* , and the corresponding set of apparent characteristics is called *phenotype*. The individual elements of the genotype are called *genes* , and their possible values are *alleles*.

The GA's work with a population of hypothesis at a time, the number of hypothesis being a parameter of choice. Each hypothesis is evaluated using the training examples and a "fitness" score, usually measuring the accuracy of the hypothesis, is assigned to it. Starting from an initial population of hypothesis, the GA exploits the information contained in the

present population and explores new hypothesis (abduction) by generating a new population of hypothesis from the old population through application of genetic operators. The genetic operators most often used are: (a) reproduction; (b) crossover, and (c) mutation.

The **reproduction** operator just duplicates the members of the population to be used to derive new members. The number of copies that each member (hypothesis) gets is proportional to its fitness score. Thus the fitness of an individual is clearly related to its influence upon its future development of the population. When many offspring of a given individual survive to reproduce, then many members of the resulting population, the "next generation," will carry the alleles of that individual. Genotypes and phenotypes of the next generation will be influenced accordingly.

After reproduction, new individuals are generated by selecting two individuals at a time from the resulting population and applying the operator of **crossover**. Crossover exchanges the genes between the two selected individuals(parents) to form two different individuals. Crossover is the key to the power of the GAs as it helps in combining information from different hypothesis to discover more useful hypothesis. Usually crossover is applied with a constant probability P_c.

The **mutation** operator randomly changes some of the genes in a selected individual and is applied at a much lower rate(P_m) as compared to crossover operator (i.e., $P_m << P_c$) . The basic GA can thus be described by the following procedure:

PROCEDURE GA (population size n, max. number of generations Ng)

```
begin;
select an initial population of n genotypes {g};
no-of-generations = 0;
repeat;
        for each member b of the population;
                compute f( g), the fitness measure for each member; /* evaluation */
        repeat;
                stochastically select a pair of genotypes g1, g2
                        with probability increasing with their fitness f; /* reproduction */
                using the genotype representation of g1 and g2,
                        mutate a random bit with probability pu; /* mutation */
                randomly select a crossover point and
                        perform crossover on g1 and g2 to
                        give new genotypes g'1 and g'2; /* crossover */
        until the new population is filled with n individuals g'i;
        no-of-generations = no-of-generation + 1;
until the number-of-generations has reached Ng or
        one of the genotype is good enough; /* termination */
end;
```

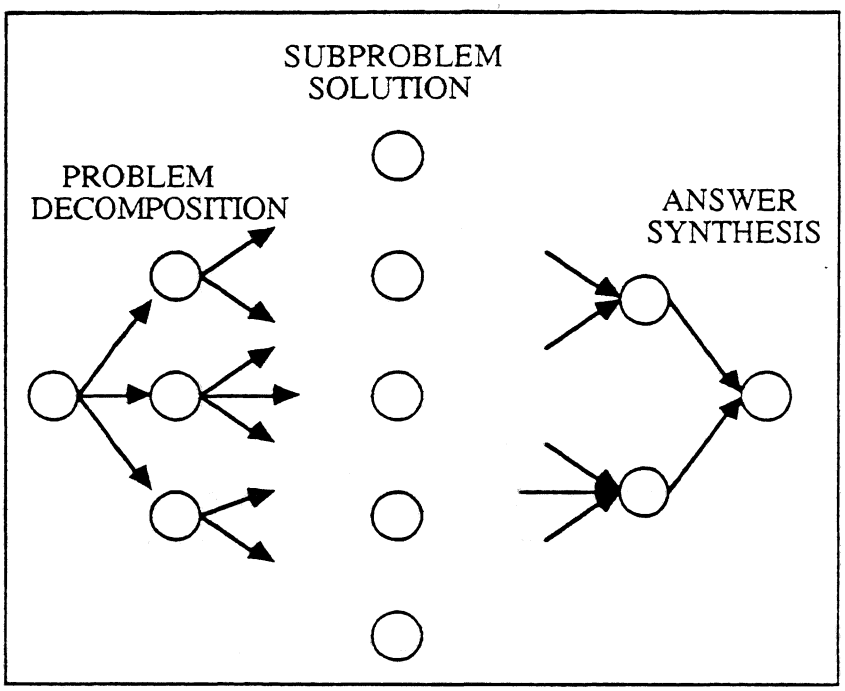

Figure 2.1 Phases of Distributed Problem Solving

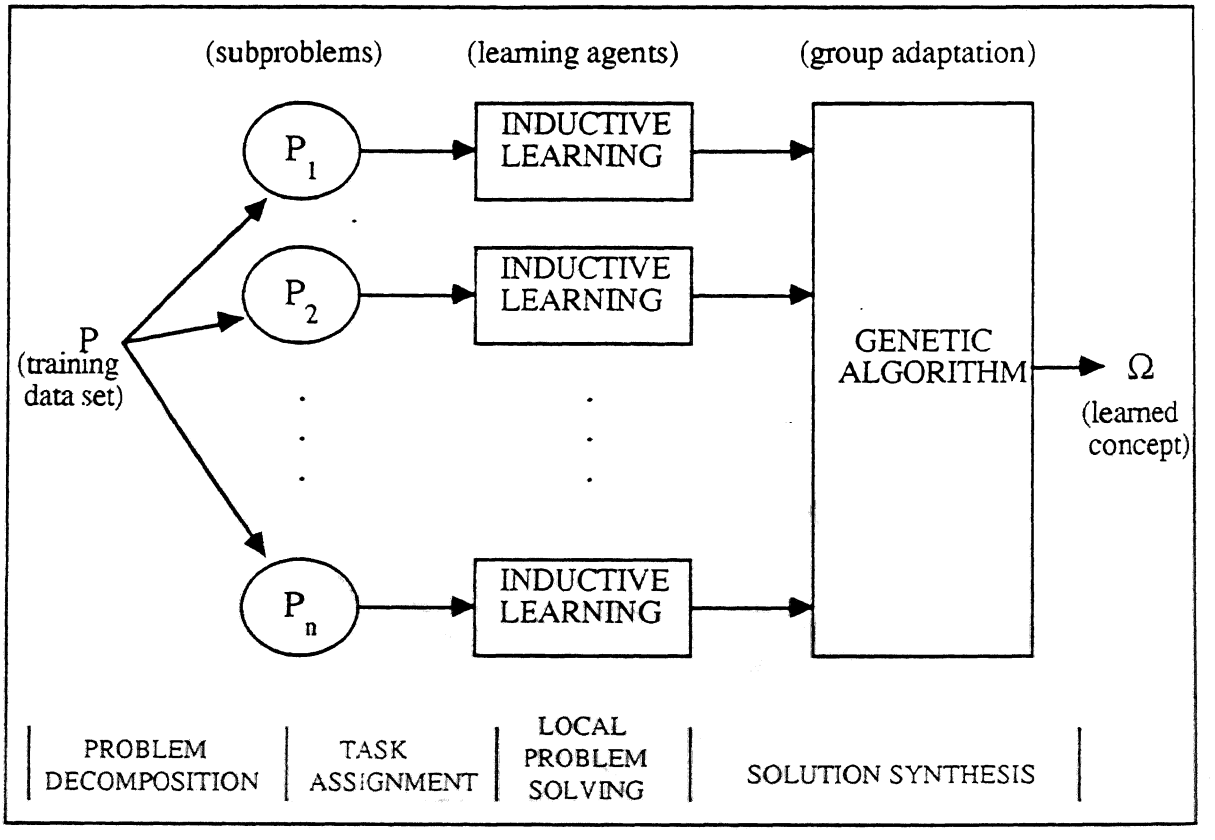

Figure 3.1 The Distributed Learning System (DLS)

```
Nov 12 19:00 1990  multi-agent.example Page 1


> (genetics)
(MULTI-AGENT VS SINGLE-AGENT PROBLEM SOLVING)
(USING SIG. LEVEL OF 1 FOR DLS AND USING LEAVE OUT 80% ONLY FOR PLS PART OF DLS)
(USING 5 DIFFERENT SAMPLE RESULTS FROM PLS1 TO GET THE INITIAL POP.)
(MUTATION RATE OF 5*0.01)
(USING 100 GENERATIONS)
(-------------------------------------------------------------------------------)
(TESTING SAMPLE HAS 79 +VE AND 35 -VE EGS)
(TRAINING SAMPLE HAS 276 +VE AND 182 -VE EGS)
(THE SINGLE AGENT RESULTS ARE)
(RULE SIZE = 17)
(((2 63) (2 63) (8 61) (3 63) (10 21) (5 63) (0 38) (0 63) (28 50)) - (4 0) - (0 0))
(((2 63) (2 63) (8 61) (3 63) (10 11) (5 63) (39 63) (0 63) (28 50)) - (1 1) - (0 0))
(((2 63) (2 63) (8 15) (3 63) (22 63) (22 63) (0 63) (0 63) (28 50)) - (2 2) - (0 1))
(((2 6) (2 63) (16 61) (3 22) (22 63) (10 63) (0 35) (0 55) (28 34)) - (1 0) - (0 0))
(((2 52) (2 63) (16 61) (23 63) (22 63) (10 63) (0 35) (0 30) (28 34)) - (12 0) - (5 0))
(((2 6) (2 63) (16 61) (23 63) (22 63) (10 63) (0 35) (31 55) (28 34)) - (4 0) - (1 0))
(((7 52) (2 63) (50 61) (23 63) (22 63) (10 63) (0 35) (31 55) (28 34)) - (1 0) - (0 0))
(((2 52) (2 63) (16 61) (3 63) (22 63) (10 63) (0 35) (0 55) (35 50)) - (176 12) - (50 6))
(((2 52) (2 63) (16 61) (3 63) (22 50) (10 63) (36 36) (0 55) (28 50)) - (13 4) - (0 0))
(((2 52) (2 63) (16 61) (3 13) (22 63) (5 63) (0 18) (56 60) (28 50)) - (1 1) - (0 0))
(((2 52) (2 63) (16 61) (3 63) (22 50) (5 63) (19 36) (56 60) (28 50)) - (16 3) - (5 1))
(((2 52) (2 63) (16 61) (3 14) (51 63) (5 63) (19 36) (56 60) (28 50)) - (1 0) - (0 0))
(((2 31) (2 63) (16 47) (3 63) (28 63) (22 63) (37 40) (0 45) (28 50)) - (6 1) - (2 0))
(((2 31) (2 63) (16 61) (3 63) (28 63) (5 26) (41 63) (0 45) (28 50)) - (30 4) - (6 3))
(((2 11) (2 63) (16 61) (3 63) (28 63) (27 63) (41 63) (0 45) (28 50)) - (4 0) - (2 0))
(((32 63) (2 63) (16 37) (3 63) (22 50) (5 63) (37 63) (0 60) (28 50)) - (2 0) - (0 0))
(((2 63) (2 63) (38 61) (3 63) (22 63) (5 63) (0 63) (61 63) (28 50)) - (2 1) - (0 0))
(-------------------------------------------------------------------------------)
(RESULTS OF THE 5 AGENTS WHICH ARE USED AS INITIAL POPULATION BY THE GA ARE)
(POPSIZE = 21)
(AGENT 1)
(((2 42) (2 42) (0 56) (10 58) (13 63) (13 57) (0 37) (0 55) (32 38)) - (188 40) - (54 8))
(((2 42) (2 42) (0 56) (10 58) (13 63) (28 57) (0 37) (56 63) (32 38)) - (9 1) - (0 0))
(((2 11) (2 42) (0 56) (10 58) (40 63) (13 57) (38 63) (0 63) (32 38)) - (20 6) - (7 3))
(((12 42) (2 42) (0 56) (10 58) (13 63) (13 57) (60 63) (0 63) (32 38)) - (1 0) - (0 0))
(AGENT 2)
(((2 36) (2 63) (15 58) (4 63) (25 56) (3 57) (0 57) (24 58) (34 40)) - (177 48) - (54 9))
(((37 63) (2 63) (15 58) (4 63) (25 50) (3 57) (0 57) (24 58) (34 40)) - (40 15) - (12 6))
(((2 63) (2 63) (8 12) (7 52) (10 26) (5 63) (0 57) (24 63) (30 39)) - (2 2) - (0 0))
(((2 63) (2 63) (8 12) (7 52) (27 63) (5 63) (0 57) (24 63) (30 39)) - (2 4) - (0 0))
(AGENT 3)
(((2 63) (2 63) (19 61) (7 52) (27 63) (5 63) (0 35) (24 45) (30 39)) - (155 20) - (49 4))
(((2 63) (2 63) (19 61) (7 52) (58 63) (5 63) (36 46) (24 45) (30 39)) - (2 2) - (1 0))
(((2 63) (2 63) (19 61) (7 52) (27 63) (5 63) (47 57) (24 45) (30 37)) - (19 4) - (4 1))
(((2 63) (2 63) (19 61) (7 52) (27 50) (5 63) (0 57) (46 60) (30 39)) - (34 13) - (10 3))
(AGENT 4)
(((2 36) (2 63) (15 58) (4 52) (20 59) (3 63) (0 38) (24 60) (29 50)) - (162 35) - (49 4))
(((37 63) (2 63) (15 58) (4 52) (20 50) (3 63) (0 38) (24 60) (29 50)) - (47 12) - (12 4))
(((2 63) (2 63) (15 58) (4 52) (20 59) (3 63) (0 38) (61 63) (29 36)) - (2 4) - (0 0))
(((2 63) (2 63) (15 58) (4 52) (20 59) (25 63) (39 41) (24 63) (29 50)) - (3 1) - (0 1))
(AGENT 5)
(((2 63) (2 63) (0 58) (4 52) (20 59) (3 63) (53 53) (24 63) (29 50)) - (4 0) - (1 0))
(((2 42) (2 42) (21 56) (7 55) (13 34) (3 63) (0 34) (0 58) (28 50)) - (9 0) - (1 0))
(((2 42) (2 42) (21 56) (7 55) (35 63) (3 63) (0 37) (0 55) (28 50)) - (193 37) - (55 6))
(((2 31) (2 42) (37 56) (7 55) (35 63) (3 63) (38 57) (0 55) (28 50)) - (25 14) - (7 6))
(((2 42) (2 42) (21 34) (7 55) (35 63) (3 63) (0 57) (56 58) (28 50)) - (3 4) - (0 0))
(-------------------------------------------------------------------------------)
(THE FINAL RESULTS AFTER SYNTHESIZING THE ABOVE RESULTS ARE)
(((2 54) (2 43) (21 58) (7 55) (9 63) (11 63) (0 36) (0 55) (28 50)) - (203 30) - (57 4))
(((2 31) (2 63) (18 61) (3 60) (27 59) (1 63) (39 63) (28 47) (24 38)) - (34 7) - (9 2))
(((34 62) (2 42) (0 56) (2 58) (13 63) (4 57) (16 41) (56 62) (32 38)) - (20 7) - (5 2))
Entering Dribble Read-Eval-Print Loop.  Type (DRIBBLE) to exit.
```

Figure 4.1 The Results of Single-agent and Multi-agent approaches to Inductive Learning

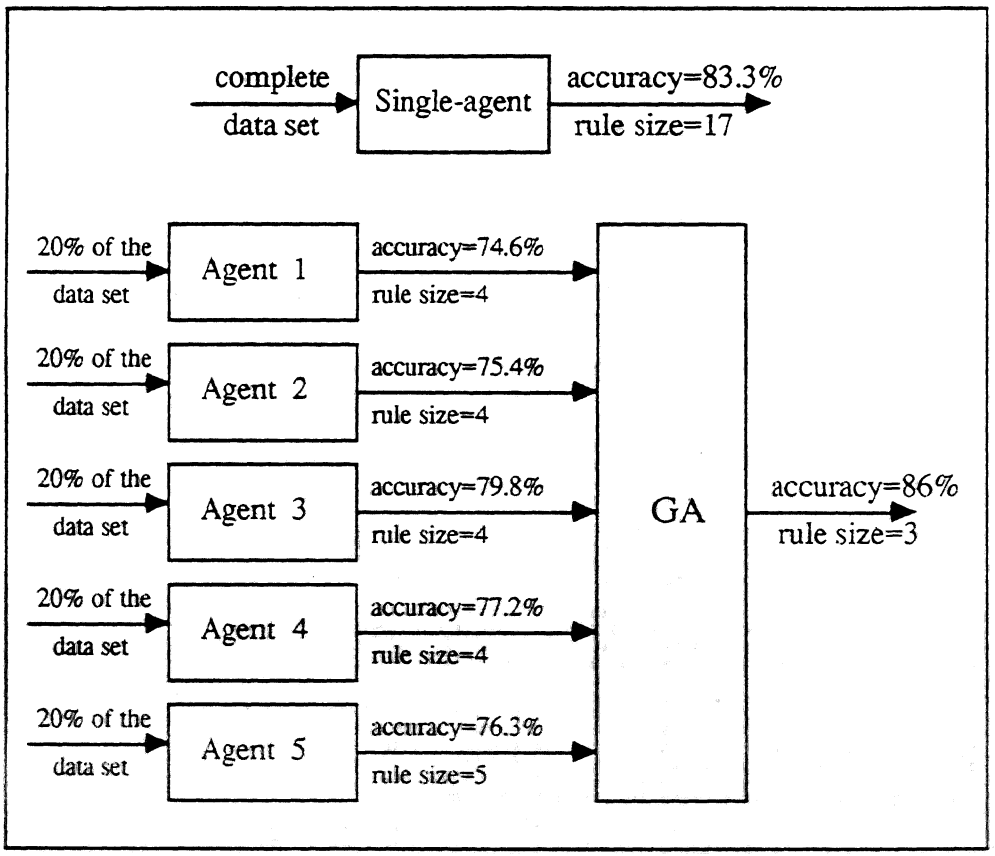

Figure 4.2 Comparison of Multi-agent vis-a-vis Single-agent approach.

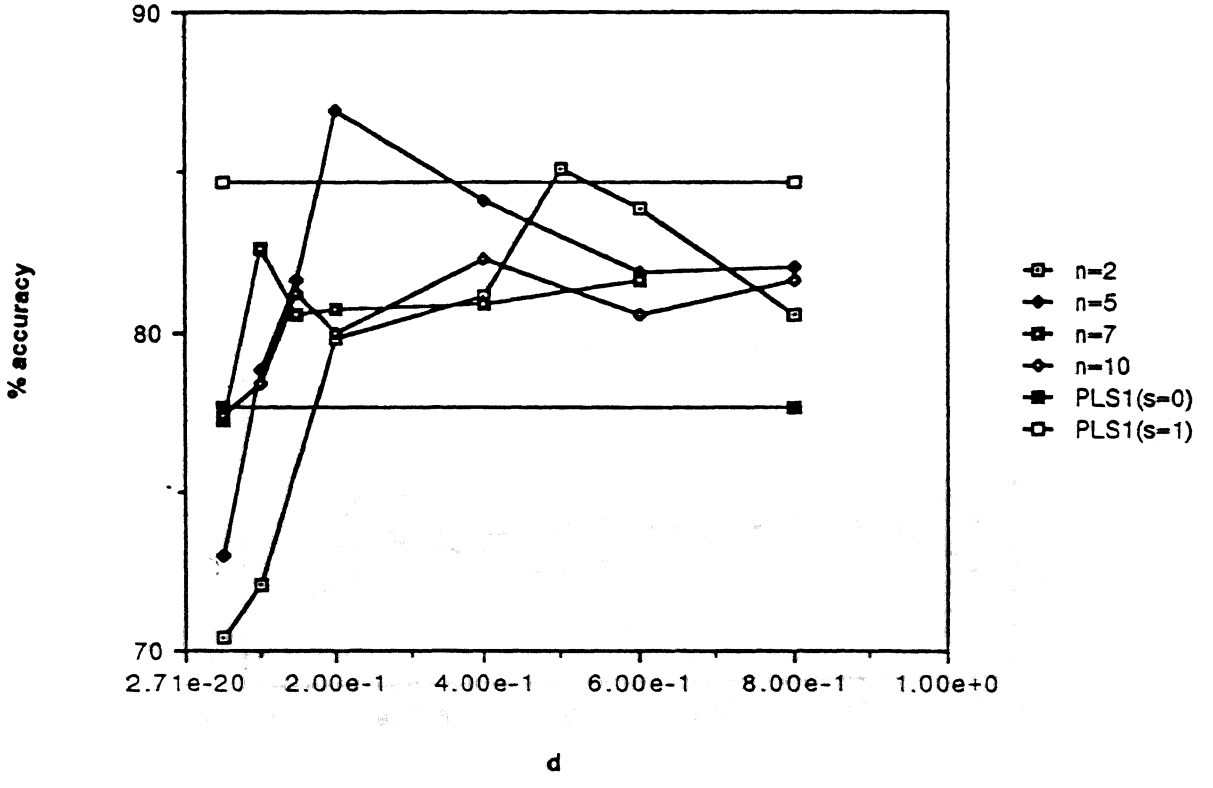

Figure 5.1 Plot of accuracy as a function of decomposibility index for different values of n together with that of PLS1

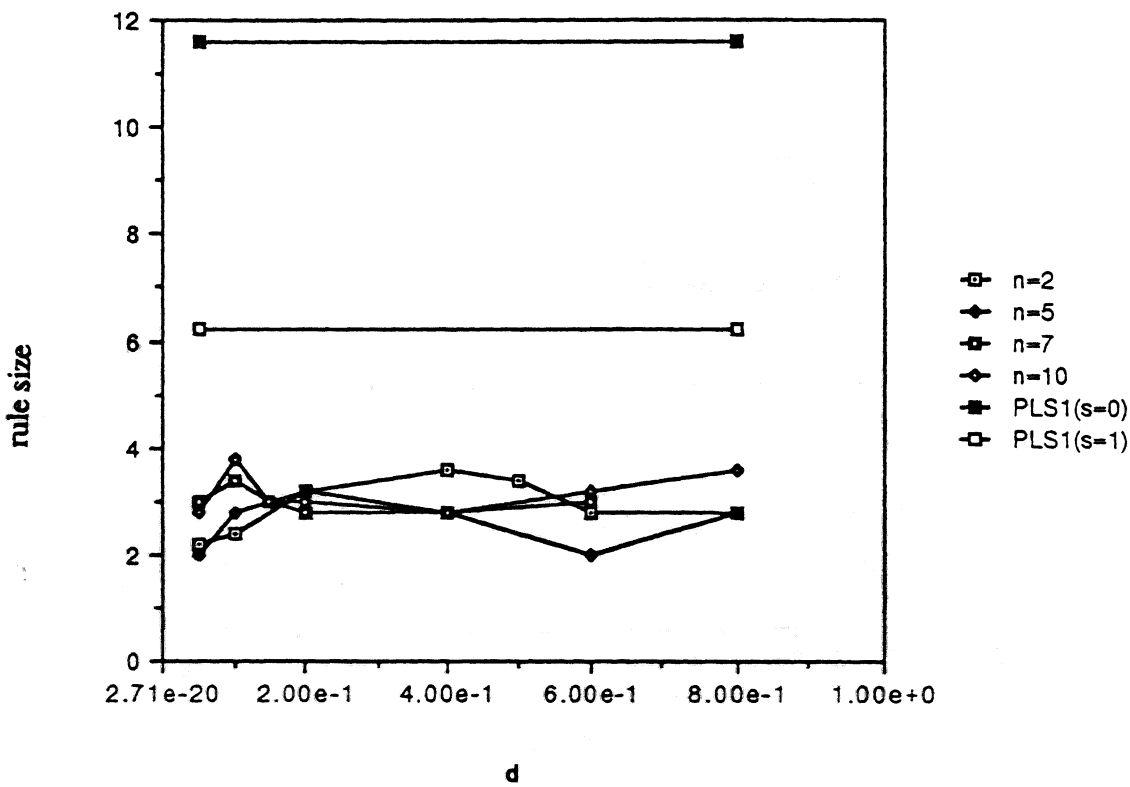

Figure 5.2 Plot of rule size of the concept as a function of decomposibility index for different values of n together with that of PLS1

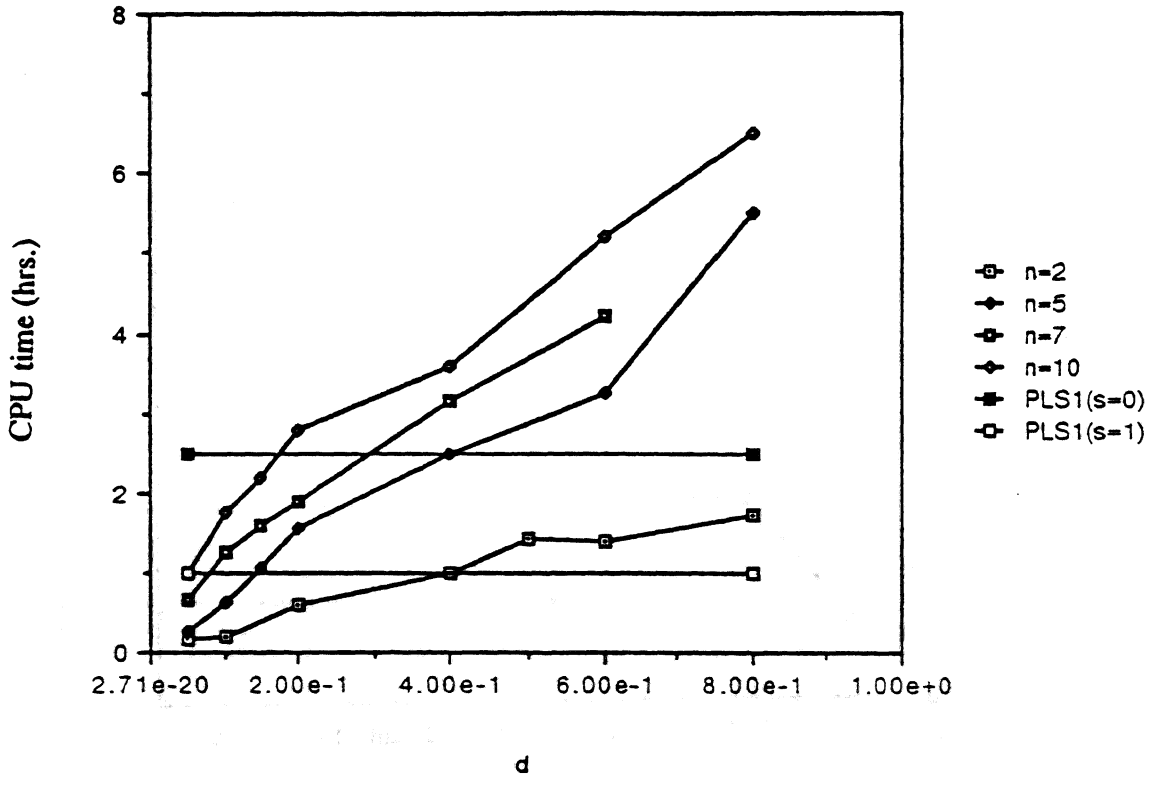

Figure 5.3 Plot of CPU time as a function of decomposibility index for different values of n together with that of PLS1

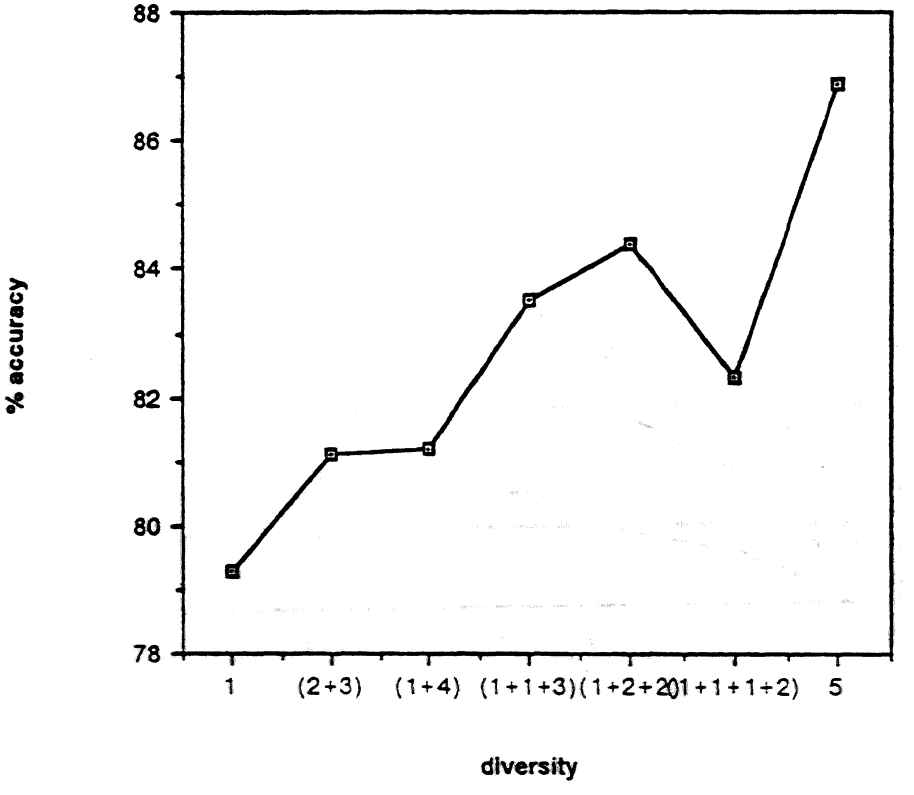

Figure 5.4 Plot of accuracy as a function of diversity number

Agents	Prediction accuracy	Rule size	g
Agent 1	74.6%	4	0.00739
Agent 2	75.4%	4	0.00655
Agent 3	79.8%	4	0.00621
Agent 4	76.3%	4	0.0184
Agent 5	76.3%	5	0.01293
Synthesis	86%	3	0.0296

Table 4.1 Results of the 5 individual agents together with the final synthesized result.

PLS1		Accuracy	Rule-size	CPU time	g
s = 0	w/o pruning	82.3%	49.8	2.5 hrs.	0.00424
	w pruning	77.6%	11.6	-	0.00424
s = 1	w/o pruning	85.4%	17.4	1 hr.	0.0658
	w pruning	84.7%	6.2	-	0.0658

Table 5.1 PLS1 results

n=2						
d	Prediction accuracy	Rule-size	CPU time	population size	g	t_0
0.05	70.4%	2.2	10 min.	5.4	0.0124	-
0.1	72.1%	2.4	12 min.	5.6	0.0345	-
0.2	79.8%	3.2	36 min.	11.4	0.0332	-
0.4	81.1%	3.6	1 hr.	16	0.0998	-
0.5	85.1%	3.4	1.42 hrs.	21.4	0.081	0
0.6	83.9%	2.8	1.4 hrs.	21.8	0.1152	0.2N
0.8	80.5%	2.8	1.72 hrs.	25	0.1467	0.6N

n=5						
d	Prediction accuracy	Rule-size	CPU time	population size	g	t_0
0.05	73%	2	16 min.	9	0.0121	-
0.1	78.8%	2.8	37 min.	14.4	0.0314	-
0.15	81.6%	3	1.07 hrs.	20.8	0.0816	-
0.2	86.9%	3.2	1.57 hrs.	29.2	0.0764	0
0.4	84.1%	2.8	2.5 hrs.	44	0.1052	N
0.6	81.9%	2	3.25 hrs.	49.6	0.1604	2N
0.8	82%	2.8	5.5 hrs.	50	0.1145	3N

n=7						
d	Prediction accuracy	Rule-size	CPU time	population size	g	t_0
0.05	77.2%	3	40 min.	15.4	0.0428	-
0.1	82.6%	3.4	1.25 hrs.	23.2	0.0389	-
0.15	80.5%	3	1.6 hrs.	29.6	0.0414	0.1N
0.2	80.7%	2.8	1.9 hrs.	35.2	0.0508	0.4N
0.4	80.9%	2.8	3.15 hrs.	50	0.0816	1.8N
0.6	81.6%	3	4.2 hrs.	50	0.1346	3.2N

n=10						
d	Prediction accuracy	Rule-size	CPU time	population size	*g*	t_o
0.05	77.4%	2.8	1 hr.	22.6	0.0262	-
0.1	78.4%	3.8	1.75 hrs.	31.6	0.0582	0
0.15	81.2%	3	2.2 hrs.	41.8	0.0928	0.5N
0.2	80%	3	2.8 hrs.	47.6	0.0807	N
0.4	82.3%	2.8	3.6 hrs.	50	0.0797	3N
0.6	80.5%	3.2	5.2 hrs.	50	0.1148	5N
0.8	81.6%	3.6	6.5 hrs.	50	0.125	7N

Table 5.2 DLS results for different values of n and d

Method	Classification accuracy	Prediction accuracy	g
PLS1 (s=0)	56%	53.16%	0.0042
PLS1 (s=1)	76.1%	74.2%	0.0658
DLS (n=5.d=0.2)	78%	79.7%	0.0764

Table 5.3 Comparison of the best rule from the concept generated by PLS1 and DLS

n=5, d=0.2				
Diversity number	Prediction accuracy	Rule-size	CPU time	pop. size
1	79.3%	2.8	1 hr.	26
2 (2+3)	81.1%	2.2	1.1 hrs.	27.6
2 (1+4)	81.2%	2.2	1.3 hrs.	31.4
3 (1+1+3)	83.5%	2.8	1.5 hrs.	30.4
3 (1+2+2)	84.4%	4	1.5 hrs.	26.2
4 (1+1+1+2)	82.3%	2.8	1.5 hrs.	31
5	86.9%	3.2	1.57 hrs.	29.2

Table 5.4 Results of varying the diversity of concepts generated by agents