1995

To Learn or Not to Learn ....

Anupam Joshi

Report Number:
95-021

TO LEARN OR NOT TO LEARN ....

Anupam Joshi

Department of Computer Science
Purdue University
West Lafayette, IN 47907

CSD-TR-95-021
March 1995

To Learn or Not to Learn*.....

Anupam Joshi
Department of Computer Sciences
Purdue University
West Lafayette, IN 47907–1398 USA
Phone: 317-494-7821, Fax: 317-494-0739
email: joshi@cs.purdue.edu

Keywords: multiagent systems, learning, neurofuzzy systems, epistemic utility, parallel computing models

1 Introduction

Systems whose behaviour emerges from the interaction of multiple autonomous agents are becoming increasingly important in computer science. There is a need to study the modalities and mechanisms used by individual agents in interacting with each other, and reacting to their environment. Since the environment is affected by the activities of multiple autonomous agents, it seems evident that an interaction strategy that adapts to the changing circumstances would be better than a static, non adaptive one. It also seems intuitive that the adaptive behaviour would occur as a product of *learning*. The intuitive, however, is not always correct, and it is our thesis that such is the case here. Specifically, we will argue that multiple strategies are needed for efficient coordination. Some of these react by learning, others react following predetermined formats. We thus propose a combination of nativist and empiricist approaches to the problem.

Rather than argue the thesis *in vacuo*, we will marshal our arguments in the context of a concrete and complex computational system. We limit our analysis here to cooperative agents. Our interest in multi agent systems arises out of work we are doing in the domain of scientific and ubiquitous computing.

In the scientific research environment, computational simulation of experimental processes is now an essential component of scientific experimentation. An important goal in the area of Computational Science & Engineering (CS&E) is to develop a unified environment where the experimental and computational models can interact with and complement each other in the problem–solving process. Such an environment has been described as a Problem Solving Environment (PSE)[2]. It supports the construction of mathematical models for the experimental process, the definition of the geometry and associated conditions, the specification of the solution strategy, and finally, the generation and validation of results. It also speaks to physical experimentation – allowing the specification of experimental input, process, data acquisition and data analysis in such a way that the information could be used to control the physical equipment in the laboratory. Another important characteristic of this environment is its ability to access information. Information of all kinds, including that which was traditionally published in archival journals or conference proceedings, is increasingly available on line. Besides being concentrated in traditional repositories such as libraries, such information is also increasingly distributed, residing in workstations and computers belonging to individual researchers or research groups, and linked together to form an infosphere. The World Wide Web (WWW, Web) is an example of such a scenario. A PSE is thus a system that provides all the computational facilities necessary to solve a target class of problems. These features include advanced solution methods, automatic or semiautomatic selection of solution methods, and ways to easily incorporate novel solution methods.

*This work was supported in part by NSF awards ASC 9404859 and CCR 9202536, AFOSR award F49620-92-J-0069 and ARPA ARO award DAAH04-94-G-0010

Moreover, PSEs use the language of the target class of problems, so users can run them without specialised knowledge of the underlying computer hardware or software.

Scientific computing systems will also become increasingly *ubiquitous*[4]. While one component of ubiquity involves computers that are mobile and connected over wireless links, another equally important aspect is computers that can be used by everyone. In other words, ubiquity aims to bring computers everywhere, and for everyone. This requires that the systems be easy to use, and that interaction with such systems follow an indirect management, rather than a direct manipulation, approach. In order to develop systems that are truly easy to use, PSEs need to provide the user with a high level abstraction of the complexity of the underlying computational facilities[16]. The user can not, and should not, be expected to be well versed in selecting appropriate numerical, symbolic and parallel systems, along with their associated parameters, that are needed to solve a problem. The goal of these PSEs is to assist the user to carry out the numerical solution of mathematical models and analyze their solutions. Depending on the mathematical characteristics of the models, there are "thousands" of numerical methods to apply, since very often there are several choices of parameters or methods at each of the several phases of the solution. On the other hand, the numerical solution chosen must satisfy several objectives, primarily involving error and hardware resource requirements.

1.1 PYTHIA

In the PYTHIA[3] project, our aim is to develop a system that will accept a description of a problem from the user, and then automatically select the appropriate numerical solver and computing platform, along with values for the various associated parameters. While the theoretical framework underlying PYTHIA is being developed in the generic context of scientific computing, our specific implementation deals with Partial Differential Equation (PDE) based systems. Currently, it automatically selects only the solution method.

PYTHIA was originally conceived as a stand alone system, a single agent in this context. Its input would be a description of a PDE in a special format that we have developed[1]. In response, PYTHIA produced a recommendation regarding a solution method, and its confidence in ranking this method as the best. This recommendation was based on the information contained in its knowledge base regarding similar problems, and what methods had worked best for them. Clearly, in order to recommend a good method for a given problem, PYTHIA must have seen a similar problem before.

Recently, we have begun to move towards making PYTHIA a collaborative multiagent system. This is, as we shall illustrate, a more natural implementation. PDEs can be widely varying. Most application scientists tend to solve only a limited kind, and hence any PYTHIA agent they are running is likely to be able to answer questions effectively only about a limited range of problems. If there were mechanisms that allowed PYTHIA agents of various application scientists to collaborate, then each agent could share knowledge and potentially answer a broader range of questions – call upon the collective wisdom of all agents, as it were.

2 When to learn, ...

The question that PYTHIA is trying to answer is the following. Given a problem, and user imposed constraints on it (like error bounds, max time allowable), which method should be used to solve the problem, and on which hardware platform. Each individual agent is therefore learning a mapping from *(problem, constraints)* to *(method, platform, parameters)*. The parameters term here refers to things like how long the solution is expected to take, what error can be expected. Clearly, some parameters of the solutions will reflect the effort by the PYTHIA agent to conform to the user's constraints. Other parameters will reflect confidence measures of the agent in proposing the method and hardware. Our recent work has concentrated on applying various learning techniques to this single agent problem. Specifically, we have used Bayesian belief nets[15], neural networks[5] and fuzzy systems[11]. We are now working on applying learning to the multiagent scenario. Specifically, what happens if the agent discovers that it does not have "enough" confidence in the prediction it is making ? We propose that the agent initially use some broker/agent name

[1]The format is a characteristic vector, whose elements denote various PDE properties.

server to find out which other PYTHIA agents are available to answer queries, and send them all the query. Presumably, it will get a set of answers from the its peer agents, and will need to decide which one is "correct". As remarked upon earlier, it is likely that some given PYTHIA agent will know a great deal about a certain type of problem. Thus from the answers received by an agent, it should be able to "learn" a mapping from they type of the problem to the (peer) agent which is most likely to have a correct answer. In future thus, it could direct queries more effectively, rather than using a flooding like technique to seek answers.

We are using a neuro-fuzzy method of learning to this end. The reason to use a fuzzy system is that in the PYTHIA scenario, classification of problems is not crisp. For instance, the solution of a given PDE could have a singularity, and also show some oscillatory behaviour on the boundaries. Thus the given PDE would have membership (to different extent) in the classes representing "solution–singular" and "solution–oscillatory". A conventional, binary membership function would not model this situation accurately. We feel that such fuzziness will be inherent in the learning task whenever agents model complex, real world problems.

The basic idea of the method we use was proposed by Simpson[13, 14], and is a variation of the leader cluster algorithm, enhanced with the notion of fuzziness. Similar methods have been proposed by Newton[10] and Grossberg *et. al.*[1]. Simpson describes a supervised learning neural network classifier that uses fuzzy sets to describe pattern classes. Each fuzzy set is the fuzzy union of several n-dimensional hyperboxes. Such hyperboxes define a region in n-dimensional pattern space that have patterns with full-class membership. A hyperbox is completely defined by it's min-point and max-point and also has associated with it a fuzzy membership function (with respect to these min-max points). This membership function helps to view the hyperbox as a fuzzy set and such "hyperbox fuzzy sets" can be aggregated to form a single fuzzy set class. This provides degree–of–membership information that can be used in decision making. Thus each pattern class in the given space is represented by an aggregate of several fuzzy sets and the resulting structure fits neatly into a neural network assembly. Learning in the fuzzy min-max network proceeds by placing & adjusting the hyperboxes in pattern space. Recall in the network consists of calculating the fuzzy union of the membership function values produced from each of the fuzzy set hyperboxes. The fuzzy min-max network provides good accuracy, facilitates *single pass learning*, has few parameters to tune & most importantly, provides on-line adaptation.

However, the method as proposed by Simpson does not allow for classes that are mutually non exclusive. It would thus fail to account for a situation where more than one agent might be expected to provide a correct answer. We have enhanced the method to allow it operate under this situation. Initial results from this approach [11] have been very promising. We are also studying other improvements to this method using techniques from computational geometry. The method as it stands tries to form classes by using isothetic hyperboxes. Clearly, this approach is extremely naive, since it would cover regions of space that did not belong to a class. We are trying to study improvements that can be obtained by allowing the boxes to have arbitrary orientation, as well as by using hyperspheres/hyperellipsoids as our space covering primitives.

3 ..... what to do while learning, ...

Once an agent has learned a mapping from problem types to (other) agents which are likely to know the answer, it can direct queries to other agents appropriately. However, learning in this instance would require some known exemplars. Since the PYTHIA agent is assumed to be a *tabula rasa* at start, it does not have any such exemplars. The straightforward approach would be to provide the system with a list of agents to query for each problem type. While this would allow the system to direct its queries, it would still not provide labelled exemplars of the type "agent a provides the correct/best answer for problem type p." Another option would be to involve the user in the process. PYTHIA would present all the answers obtained from peer agents, and the user would select the best. Given this kind of a scenario, Lashkari *et. al.*[6] have developed a trust function which each agent uses to measure its belief that a peer agent has a correct answer. This is self defeating in our case since the aim of the system is to allow a non expert (in HPC) to use it.

We propose an alternate approach formulated in terms of *epistemic utility*. We summarize the ideas here following Lehrer's[7] presentation of internal coherence and personal justification in humans. The basic idea here is that each agent has an *acceptance system*, which it uses to accept certain hypothesis as true. This system is based on two principles, obtaining truth and avoiding error. It informs an agent that it is more reasonable to accept some things than others. It enables the agent to judge which sources of information to trust and which not. Adapting Lehrer's definitions of acceptance and coherence to the agent scenario, we define

Definition 1 *Agent $\mathcal{A}$ is justified in accepting proposition $\mathcal{P}$ at time t if and only if $\mathcal{P}$ coheres with the acceptance system of $\mathcal{A}$ at t.*

Definition 2 *$\mathcal{P}$ coheres with the acceptance system of $\mathcal{A}$ if and only if it is more reasonable for $\mathcal{A}$ to accept $\mathcal{P}$ than any other competing claim $\mathcal{Q}$.*

In effect, we are saying that of all the competing hypotheses, an agent should accept the one which is more reasonable. Since we introduce time as a factor in the definitions, we leave open the possibility that an agents acceptance system, and hence its notion of what is reasonable, will evolve over time. Note that reasonableness is not the same as he probability of being true. Consider, for instance, the following statements

It looks like there is snow on the ground.

There is snow on the ground.

Now, it could appear that there is snow on the ground for a whole bunch of reasons (white confetti, TP-ing by a fraternity party, hallucination ....) other than there actually being snow on the ground. Accepting the first statement therefore is less likely to make us err. On the other hand, it doesn't really tell us quite as much as the second statement, so we don't gain in the area of obtaining truth. To obtain a quantitative measure of *reasonable*–ness, we need to combine two factors, one which denotes the probability of a proposition q being true, and the other which denotes its utility. Specifically, let $U_t(q)$ denote the positive utility of accepting q if it is true, $U_f(q)$ denote the negative utility of accepting q if it is false. Further, let $p(q)$ be the probability that q is true. Then, the reasonableness of accepting q can be defined as [7]

$$r(q) = p(q)U_t(q) + p(not(q))U_f(q).$$

In the case of PYTHIA, each agent produces a number denoting confidence in its recommendation being correct, so $p(q)$ is trivially defined, and $p(not(q))$ is simply $1 - p(q)$. The utility is a more tricky measure. We have earlier posited that the more problems of a type that an agent has seen, the more likely it is to recommend an appropriate method etc. for a new problem of the same type. Moreover, the reason for the epistemic module is to provide exemplars for learning. Thus the utility of accepting an agent's recommendation (and using it as an exemplar for learning) should reflect the number of problems of the present type that it has seen. In this instance, we chose to make the positive and negative utilities the same in value, but opposite in sign. This is done since the value of utility is measuring the amount of knowledge that an agent appears to have.Thus $U_t(q) = -U_f(q) = f(N_e)$, where f is some squashing function mapping the domain of $(0, \infty)$ to a range of $(0, 1]$, and N_e is the number of exemplars of a given type(that of the problem being considered) that the agent has seen. We chose $f(x) = \frac{2}{1+e^{-x}} - 1$.

4 ..... and when not to learn

An important component of scientific computation is the optimal use of the available, heterogeneous High Performance Computing (HPC) hardware. We view each hardware platform as an agent, with bounded (computational) capability. Part of this capability is inherent in the hardware. The other part is a function

of the amount of load on it. We have outlined in the previous section a mechanism using which PYTHIA can decide which method would best solve a problem, and propose a hardware platform as well. However, this problem is actually more complicated, since the hardware platforms in question are mostly parallel, or networked workstation clusters. Thus their configurations (for instance, the number of processors devoted to a problem) can be changed. So the mapping to be learned is from *(problem, method, hardware, config)* to *time*. Even if one were to restrict the notion of configuration to merely the number of processors, it would still be a computationally intense task to learn the time taken by a given hardware with a given number of processors to solve a given problem by a given method. The naive approach here, in our opinion, would be to throw a learning mechanism at this problem. We believe that direct learning is not required in this case, and the system can be adaptable without it.

The adaptability can be achieved by a combination of learning and modeling. We have illustrated in a previous section how PYTHIA will recommend a hardware platform, and give an estimate of the time required to solve the problem. This time estimate would be with respect to some standard configuration of the hardware in question. The appropriate configuration of the system can be obtained by providing the PYTHIA agent the capability to model the configuration to speedup characteristics of various hardware agents. Since a heterogeneous system would not have more than a few tens of different computing platforms, this would not be a burden. Analytic models predicting speedups in parallel platforms are notoriously difficult to obtain. However, seminal work in this direction has been done by colleagues in our group [8]. The system proposed by them parametrizes scalability of computation using three quantities that are in turn functions of the number of processors. These are:

1. $E(P)$, the number of communication events
2. $f(p)$, the fraction of time spent in sequential and duplicated work
3. $I(P)$, the instruction execution rate

Using such an approach, the PYTHIA agent can model the behaviour of hardware agents to predict what number of processors would be optimal for a given problem on a given platform.

The speedup scheme has been expounded in the context of a single program executing on a given platform at a time. However, it provides the requisite framework for extension to the case where each hardware agent is actually a multiprocessing system. In such cases, the (computational) capability of the hardware agents will dynamically vary. Each agent will be aware of the load on it at any given time, and can query other agents for their loads as well. Our multiagent system will be responsive to this dynamic behaviour, and will adapt by moving computations around. This will be achieved by systems similar to those proposed in [9] by Rego *et. al.* . Such systems use threads of control to efficiently migrate tasks across processing systems, and have been shown to be extremely effective in simulations [12].

5 Conclusion

That Multiagent systems have to be adaptable and involve learning is, in our opinion, evident. In this paper, we have presented a multiagent system operating in a complex, scientific computing environment. We have argued that learning is not the panacea that will make the difficulties of coordination in multiagent systems disappear. Specifically, we have shown scenarios from our research where learning is used, and where the system adapts based on *a priori* known models of other agents. We have also shown how *epistemic utility* theory can be used to facilitate learning in an unsupervised manner. Of course, where supervised learning is available, it can be (and is) trivially incorporated into the system.

References

[1] G. Carpenter, S. Grossberg, and S. Rosen, *Fuzzy ART: Fast stable learning and categorization of analog patterns by an adaptive resonance system*, Neural Networks 4·(1991), 759–771.

[2] E. Gallopoulos, E. Houstis, and J.R. Rice, *Computer as Thinker/Doer: Problem-Solving Environments for Computational Science*, IEEE Computational Science and Enginerring **1** (1994), no. 2, 11–23.

[3] E. Houstis, S. Weerawarana, A. Joshi, and J.R. Rice, *The PYTHIA projet*, Proceedings First Intl. Conf. on Neural, Parallel and Scientific Computing, 1995, (to appear).

[4] A. Joshi, T. Drashansky, E. Houstis, and S. Weerawarana, *SciencePad: An Intelligent Electronic Notepad for Ubiquitous Scientific Computing*, International Conference on Inteligent Information Management Systems, June 1995, (to appear).

[5] A. Joshi, S. Weerawarana, and E. Houstis, *The use of Neural Networks to support intelligent scientific computing*, Proceedings IEEE Intl. Conf. Neural Networks, IEEE, IEEE Press, July 1995.

[6] Y. Lashkari, M. Metral, and P. Maes, *Collaborative Interface Agents*, Proceedings AAAI '94, AAAI, 1994.

[7] K. Lehrer, *Theory of Knowledge*, Westview Press, Boulder, CO, USA, 1990.

[8] D. Marinescu and J.R. Rice, *On the scalability of Asynchronous Parallel Computations*, J. Parallel and Distributed Computing **22** (1994).

[9] E. Mascarenhas and V. Rego, *Ariadne: Architecture of a Portable Threads system supporting Mobile Processes*, Tech. Report CSD-TR-95-017, Department of Computer Sciences, Purdue University, 1995.

[10] S. Newton and S. Mitra, *Self organizing leader clustering in a neural network using a fuzzy learning rule*, SPIE Proc. 1565: adaptive signal processing, SPIE, 1991.

[11] N. Ramakrishnan, A. Joshi, S. Weerawarana, and E. Houstis, *Neuro-fuzzy systems for intelligent scientific computing*, (submitted to ANNIE '95), 1995.

[12] J. Sang, E. Mascarenhas, and Rego V., *Process mobility in distributed memory simulation systems*, Proceedings Winter Simulation Conference, 1993, pp. 722–730.

[13] P.K. Simpson, *Fuzzy min-max neural networks-part I: Classification*, IEEE Trans. Neural Networks **3** (1992), 776–786.

[14] ———, *Fuzzy min-max neural networks-part II: Clustering*, IEEE Trans. Fuzzy Systems **1** (1993), no. 1, 32–45.

[15] S. Weerawarana, *Problem solving environments for partial differential equation based systems*, Ph.D. thesis, Department of Computer Sciences, Purdue University, 1994.

[16] S. Weerawarana, A. Joshi, E. Houstis, and A. Catlin, *Using NCSA Mosaic to build notebook interfaces for CS&E applications*, Tech. Report CSD-TR-95-006, Department of Computer Sciences, Purdue University, 1995, (submitted to IFIP WG2.7 EHCI '95).