Handling a large number of machine learning experiments in a MAS based system

Juan A. Botía*and
Antonio G. Skarmeta
{juanbot, skarmeta}@dif.um.es
Dep. Ingeniería de la Información y las Comunicaciones
Universidad de Murcia,
E–30071 Murcia, Spain

Mercedes Garijo and
Juan R. Velasco
{mga, juanra}@gsi.dit.upm.es
Dep. de Ingeniería de Sistemas Telemáticos,
Universidad Politécnica de Madrid,
E–28040 Madrid, Spain

ABSTRACT

Meta-learning is concerned with the study of the computational learning process in general and is oriented to achieve the best possible theory from data. This type of problems are very characteristic, from a computational point of view. Basically, they require an intensive use of storage and computational resources. In this paper, a software architecture intended to guide the engineering of a multi-agent system to cope with meta-learning is presented.

1. INTRODUCTION

Meta-learning has become, in the last five years, a proper machine learning research field. Basically, meta-learning consists in the study of the computational learning process in general, with a clear objetive: to obtain the most suitable model for a concrete learning task. This paper introduces a software architecture that copes with the typical problems presented in a system that manages this issue. METALA (META-Learning Architecture) is a set of recomendations that guide the building of a MAS (Multi-Agent System) based system to work with machine learning in general and meta-learning in particular. One topic is focused in deep in the present work: the massive inductive learning problem. It is related with the distribution of partial machine learning experiments over the learning servers that compound the distributed system built over the propossed architecture. In section 2, typical meta-learning approaches are introduced along with the problems that a system of this kind generates, from the point of view of scalability. Section 3 is dedicated to the presentation of the basic features of the propossed architecture. Then, in section 4 the problem of handling meta-learning experiments, from the particular view of clasic information systems is analysed. Finally, section 5 points out conclusions and suggests future research directions.

*Work partially supported by the European Comission through the project FEDER 1FD97-0255-C03-01

2. META-LEARNING

Meta-learning deals with the study of the the machine learning process in general. It has a clear objetive that is obtainning the best possible model, for a concrete data set and a possible pool of learning algorithms. This paper is not dedicated to meta-learning. It is dedicated to the software problems caused by supporting it. However, to understand the nature of such problems a brief introduction to the issue is needed.

2.1 A brief taxonomy

Basically, in the study of the problem, the authors have identified the following types of meta-learning:

- Composite: it uses different algorithms, for the same learning task, with similar, but not the same, data sets. It basically merges all classification of local models, to produce a final output. Different schemes for merging these reponses are voting, arbitring and combination. The paradigmical example of this approach is [16]. A complete study of this technique can be found in [5].

- Constructive: this tries to build, on the run, a new learning system, each time a learning process is needed. It is a knowledge based system, in the sense that it uses ontologies for the description of a generic learning process and chunks of primitive processing units. The CAMEL system [17] is a good example of this approach.

- Boosting & Baggin: both of this techniques [7, 4], use the same algorithm, for the same learning task and the same data set. Basically, both of them run T different experiments, sequentially and in parallel, respectively. Once all T different models are obtained, responses are merged to get an overall model.

- Ranking based: this meta-learning technique uses different algorithms, with different data sets, for the same learning task. It rests in the idea of creating a list of classifiers (i.e. a raking), ordered

by a criteria based on performance for the different data sets available. This ranking will be used thereafter to decide which algorithm, or algorithms depending on available resources, can be used to produce a hopefully reasonable good model. In [3] a comparison of three ranking methods can be found.

- Inductive: this can be seen as pure meta-learning, as it is learning over past learning experiences, inductively. In inductive meta learning there are four subtypes depending on whether a single learning algorithm or many of them are used and whether a single data set of many of them are implied. These are SASD (Single Algorithm and Single Data), SAMD (Single Algorithm and Multiple Data), MASD (Multiple Algorithms and Single Data) and MAMD (Multiple Algorithms and Multiple Data). An example of this type of learning is [1].

- Hybridization: as is commonly accepted in the machine learning research community, no automatic learning algorithm performs best for all possible learning tasks. This result is known as the NFL (No Free Lunch) theorem [18] in its theoretical basis. Previous experimental results had already possed the idea [11] in the ground. All learning paradigms have their advantages and disadvantages, depending on the particular problem (i.e. data + learning task) the algorithm has to deal with. However, both advantages and disadvantages of well possed learning techniques are known enough. Hence, why not to use different learning techniques, combined in a way in which each technique would attack the part of the problem that best suits to its own features? This is the main idea of hybridization. There are may examples of successful hybridization like [8, 10, 13]

There are two different blocks of issues that emerge from the analysis of all meta-learning techniques. The first one is related with the algorithmic interaction of the different learning paradigms. Both in constructive meta-learning and hybridization there is a clear need of easy and flexible interaction between different learning processes. This problem has been called, in the context of this work, the **learning processes interaction problem**. However, this is not the topic of this article. The scalability issue is related with the second problem. It gots to do with the necessity of handling a large number of learning experiments and its corresponding results. This is called the **massive inductive learning problem**.

We will use an example to better explain this problem. Suposse that giving a one hidden layer and sigmoidal feedforward neural network with the backpropagation algorithm [14], and a concrete data set, we want to find the best possible model, given a traing and test validation approach [12]. This can be a possible SASD task. For the *canonical* backpropagation algorithm, there are two *degrees of freedom*:

- Hidden nodes: the number of hidden nodes in the network plays a key role in obtainning an accurate and simple model. In order to obtain the best possible model, suposse that hidden nodes range is in $[1, 2, \ldots, n]$, in which n is the number of examples.

- Random initizalization of weights: the backpropagation search is gradient based, so the possibility of finding local minima is very high. Hence, the place where the search starts is very important. If weights are randomly initialized, possibilities of finding a good local minima grow. Lets suposse that the evaluation error follows a normal distribution. Then, more than 30 different initializations would be enough to ensure a good confidence of a complete distribution of initial weights in the search space.

Supossing that in the set of all possible experiments obtained with the combination of values for hidden nodes and weight initialization, we could find the best possible model, $n \times 31$ experiments should be carried out. For a simple data set of 2000 tuples, 62000 experiments should be done. Many problems arise, from the perspective given by this simple example. These are related with

- **Computational power**: it is clear that such a computational task shows an intensive consumption of primary and secundary memory and CPU. Moreover, if more than one server is available, this fact should be taken into account and all servers should be used. Load balancing (see section 4) plays a central role here when many experiments must be distributed into available servers.

- **Coordination**: coordination of client and servers is necessary. A meta-learning system must be capable of accomplishing such a learning task but maintaining a monitorization of what is happenning during learning is also important because this kind of tasks can take hours, or even days to be calculated.

- **Analysis of results**: once all experiments have been executed, it shall be necessary to store them in a persistent storage device. This system should allow flexible storage of complex data structures and flexible recovery of that data for analysis. Analysis is necessary in order to find the best model.

In this section, initial problems are pointed out. However, the present paper is dedicated to show how some of the computational power related problems are initially solved. The other two kinds of problems are not studied here. They were roughly studied in a former work [2].

3. THE ARCHITECTURE

This is a brief section dedicated to the presentation of the architecture. The architecture appears depicted, by means of a structural diagram, in figure 2. METALA has an antecedent, as software architecture. The antecedent is GEMINIS (GEneric MINIng System). This former architecture was also presented in this workshop last year (see [2]) and it is focused on the KDD

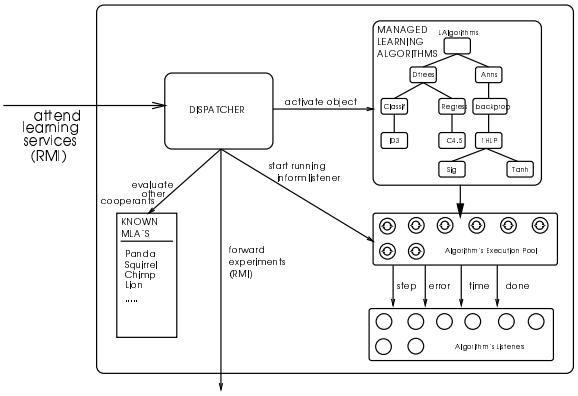

Figure 1: MLA architecture.

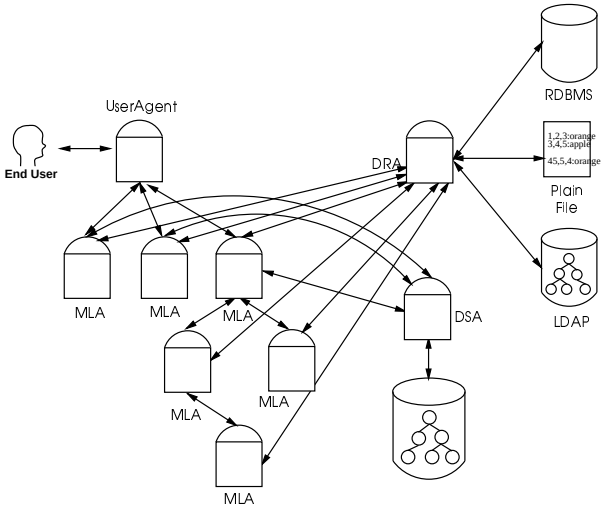

Figure 2: METALA agents architecture.

(Knowledge Discovery in Databases) process [6]. However, meta-learning has become a very important problem so the authors decided to create a new architecture to study the problem more in deep. Basically, the architecture is based on four different types of agents.

The first one is the ser agent, that acts as the interface with the system and it is in charge of the learning task execution planification.

The second type of agent is the MLA. MLAs (Machine Learning Agent) offer machine learning algorithms, as learning services. One MLA can make available many learning algorithms. The propossed architecture for a MLA appears in figure 1. Its main part is the `dispatcher`. This entity is in charge of launching new experiments coming into the agent to the `algorithm's execution pool`. Once an algorithm is launched into the pool, it becomes a new thread of execution in there. Each one of that thread has its corresponding listener, which represents the client monitoring its evolution. The dispatcher can also make use of other MLAs (e.g. the algorithm is an hybridization, and it needs algorithms that are not found in the local algorithm's database) and hence can forward subexperiments to it. In that case, the forwarding MLA becomes the listener of the forwarded experiment. The mechanism used to find other MLAs, by the user agent and the MLAs is the DSA.

The DSA (Directory Service Agent) is the third type of agent in METALA. It is in charge of white and yellow page services, and results repository services. Basically, it encapsulates and LDAP (Lightweight Directory Access Protocol) [9] database.

The fourth type of agent is the DRA (Data Repository Agent) that encapsulates access to learning data, that can come from a plain file, a relational database or even from an LDAP repository.

The user can see all available algorithms, and data. He can design meta-learning experiments by means of the user agent. This task consists of choosing the learning algorithm, its configuration parameters along with possible variation ranges for their corresponding values and the data from which the theories will be created. In a second stage, this agent is in charge of planning the meta-learning execution. While the experiment is being executed, the user agent shows the evolution of each sub-experiment produced. Finally, when all experiments are performed, results can be analysed. Next section is dedicated to the planification of meta-learning experiments execution.

4. PLANNING META-LEARNING EXPERIMENTS

METALA is designed to built an application over a set of heterogeneous servers. Moreover, the number of available servers in the system can be variable at a time. This directly influences the number of MLAs available as well, as only a single MLA can run in a concrete. Once a MLA is launched into the system, it subscribes itself at the directory through the DSA. Subscription consists in telling the DSA what algorithms the MLA is capable of serve and what are their respective configuration parameters. The number of available MLAs is dynamic. It can grow without limit, or it can decrease to even zero agents.

The user agent is in charge, once the meta-learning experiment has been designed, of planning its execution and following their evolution. Clearly, this planning has to take into account all available MLAs and their corresponding workloads (i.e. experiments they are already performing). This planning must be seen as a load distribution problem [15].

In sections 4.1, 4.2 and 4.3 the three different criteria tested for planning learning experiment are introduced, along with an empirical study of the influence of the number of servers running MLAs and the complexity of the learning task. Experiments where not simulated. They were performed using a JAVA-RMI based prototype developed in the Intelligent Systems Group of the Murcia University. The execution environment was heterogeneus because servers had different hardware and software configurations. Table 1 shows the seven servers that take part on experiments. They are offered only with illustrative purposses.

Experiments load was obtained, using a SASD sort of inductive meta-learning. It was inspired in the example appearing in section 2.1. The algorithm used was

Order	Architecture	O.S.	RAM
1	SPARC II (Ultra 5)	Solaris 7	128MB
2	Intel Pentium III	Linux Red Hat 6.2	128MB
3	SPARC II (2Processors)	Solaris 7	256MB
4	SPARC II (Ultra 10)	Solaris 7	256MB
5	SPARC II (2Processors)	Solaris 7	256MB
6	SPARC I (Ultra1)	Solaris 7	64MB
7	Intel Pentium III	Linux Mandrake 7.2	128MB

Table 1: Hosts running the different MLAs for time experiments

backpropagation over a one hidden-layer sigmoidal neural network. All MLAs involved had integrated that concrete algorithm. The data set had over 200 tuples of continuous data with four attributes. The learning task was simply a regression of the fourth attribute using the first three attributes in the dataset as input. Varying the load of the learning task consisted only in augmenting the number of hidden nodes to test.

4.1 A round-robin criteria for load balancing

The first criteria, the easiest to implement, was a round-robin based criteria. For that, the user agent owns a list of all MLA agents that are alive in the system, and a pointer to the next agent that will receive an experiment. MLAs alive on the system are obtained from the DSA. This agent has this information because are MLAs must sign up in its DSA in order to receive learning requests.

The available MLAs list is circular. We can see the proof of the unsuitability of this load balancing technique having a look at figure 3. In that figure, there are three graphs that correspond to experiments using this criteria. Those are the ones labeled with `7 Experiments`, `9 Experiments` and `11 Experiments` corresponding to the execution of seven, nine and eleven subexperiments respectively. The number of servers moves along the x axis. An ideal curve should monotonically decrease but that is not the case here. That is so because machines running MLAs (recall that each different MLA is running in a different machine) do not have the same computational power but the same workload at a time. In this experiment, when a second MLA gets into work, there is a remarkable increment in elapsed time. With this policy, this new MLA would take three of the seven experiments (i. e. experiments 2nd, 4th and 6th). As we can see, the second MLA takes more time executing three of the experiments than the first one executing the whole bunch of them. The potentially strong differences of computational power between machines make this criteria useless. However, this criteria should be useful with the system running in a set of servers of the same kind as the planning strategy is the easiest to implement.

4.2 A statically weighted workload based strategy for load balancing

It would be interesting, as it has been stated in the former section, for a workload balancing strategy to take into account computational power differences derived from different servers. A simple technique taking into account these differences would be one that statically weights the workload of each MLA (understood as the

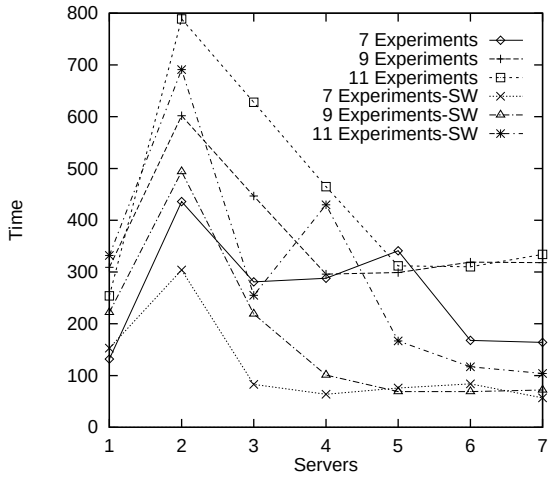

Figure 3: Different meassures of time elapsed. Each curve represents a number of experiments (i.e. a different global load, with also a different load balancing strategy.

number of learning experiments it is carring out at a time). Let a MLA be labeled as h. Then its weight shall be w_h and the workload of h, W would be obtained with

$$W_h = w_h \times e_h,$$

being e_h the number of experiments currently performed by h. Clearly, the more computational power a host shows, the more smaller w_h it should have. A possible set of w_h values for the hosts appearing in table 1 would be

h_1	h_2	h_3	h_4	h_5	h_6	h_7
0.4	0.9	0.1	0.2	0.1	0.7	0.8

This computational power model can be visualiced with the figure 4. It can be easily interpreted. For example, suposse that there are only two MLAs alive in the system, in a concrete moment. Let these MLAs be the ones with $w_6 = 0.7$ and $w_3 = 0.1$. The two of them are idle. Suposse that a new experiment comes. Notice that, in the x axis, at the point with $x = 1$, the workload for the MLA labeled as 3 is smaller. Hence, the experiment would be assigned to it. Now, the workload for the agent labeled as 3 is the one meassured at the point where $x = 2$. However, the following six experiments would be assigned to it, because in the whole x range $[1, 7]$, the workload of the MLA labeled as 6 is never higher than the one of the MLA labeled as 3.

Using this weights, and choosing each moment the MLA with smaller W_h, three disctinct graphs are obtained, using againg 7, 9 and 11 subexperiments. They can be found also in figure 3, labeled with the `SW` suffix. Clearly, this strategy overcomes the one based on round-robin. However, it also shows some limitations. It is not correct to suposse that the w_h must be fixed. Obviously, all hosts are serving other computational tasks for other users and applications, as well as the related to MLAs. Hence, weights could vary dinamically. However, this scheme should be usefull in a configuration in

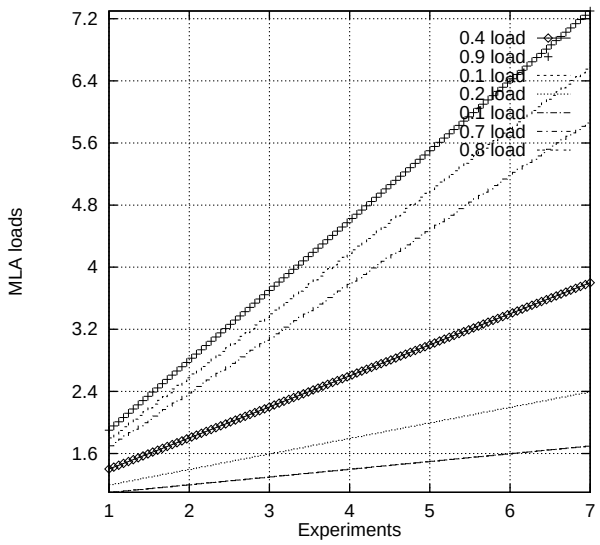

Figure 4: Evolution of all loads, depending on the experiments each MLA is working on for the statically weighted strategy for load balancing.

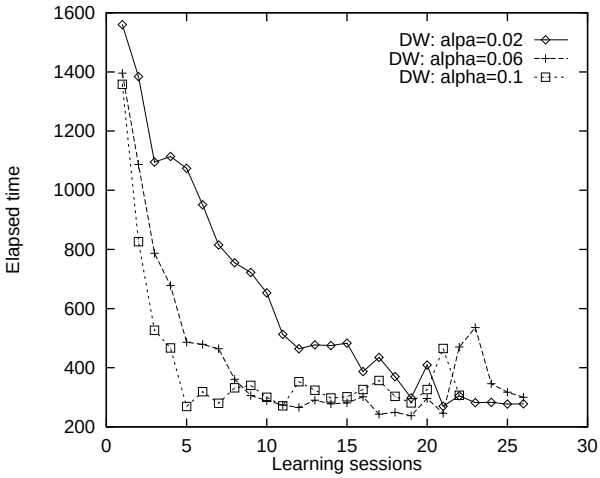

Figure 5: Evolution of elapsed time for the same meta-learning experiment. This experiments involved the seven servers available, and in each session along x axis, there was an adjust of w_h for each.

which servers have different configurations but they are dedicated to this concrete application.

4.3 A dinamically weighted workload based strategy for load balancing

In the last approach envisaged in this work, all learning agents will launch in the system with the same initial w_h. Once a request of experiments is sent to the system, those are distributed, in a round-robin fashion (i. e. all w_h are the same). When all experiments are done, times for each one of the experiments are requested from the DSA and the w_h are adjusted accordingly to the differences between them. Formally, let S be defined as the set $S = \{e1, e_2, \ldots, e_n\}$. S represents a complete learning session. The labels e_i are the experiments that must be executed in that session. Once S is defined by the user and launched into the user agent, and after all experiments are done, t_i, $1 \leq i \leq n$ are obtained, where t_i is the elapsed time since the experiment i has been sended to the pertinent MLA to the time in which the MLA has finished with it. Let μ_S be the arithmetic mean of all experiment times. It would be defined as

$$\mu = \frac{t_1 + t_2 + \cdots + t_n}{n}$$

Then, each weight w_h would be adjusted, for each experiment e_j by means of the expression

$$w'_h = w_h + w_h * \left(-\alpha + \frac{2\alpha(t_j - t_{min})}{t_{max} - t_{min}} \right) \qquad (1)$$

where e_j is an experiment executed by the server h, t_{min} and t_{max} are the minimun and maximun elapsed time for all experiments in S, and α is a parameter that conditions the convergency speed for the adjustment.

Evolution in time for this particular load balancing strategy appears in figure 5.

Notice that when α augments, the corresponding time graph decreases faster. However, when a stabitily situation is reached, a higher α implies a higher unstability in elapsed time, i. e. the graph shows bigger variations from one learning session to another. This can be appreciated also in the figure 6. This three different groups of graphs show the evolution of loads for each agent. Each distintc graph represents the load of an agent, as the number of experiments grows. Each group of graphs correspond to a different α. The upper group correspond to an $\alpha = 0.02$. The second group shows evolution of load with $\alpha = 0.06$ and the third with $\alpha = 0.1$. As experiments grow in complexity, two different groups of loads are being generated corresponding to the three big SPARC servers. The others four correspond to the small SPARCS and all Intel machines. Notice that for the upper group, graphs are very smooth because α is small. In the third group, variations in load are higher.

This kind of strategy is suitable for heterogeneus execution environments, with varying loads in servers because it is adaptative, by means of experiments execution time.

5. CONCLUSIONS

The METALA architecture, a set of recommendations for building meta-learning based systems has been introduced. This architecture is continuously evolving and there is not an stable version yet. Different problems regarding meta-learning has been signaled and focus has been put on load balancing distribution. Three studied strategies, for different execution enviroments have been offered in the three sections above. The third strategy needs more experimentation because workloads from other users and applications were the real ones while experiments were being done. Controlled and exhaustive simulation has to be done in that direction. Moreover, the concept of cost must be taken into account. In a real distributed system, it is possible that

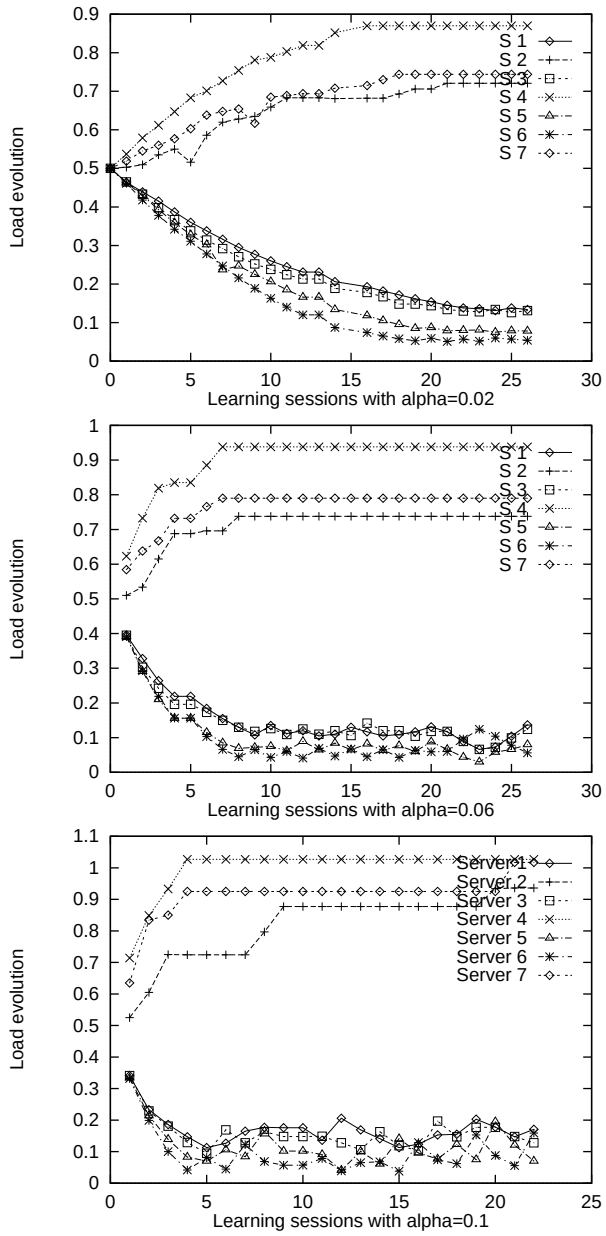

Figure 6: Evolution of all w_h for all h in $1 \leq h \leq 7$, for $\alpha \in \{0.02, 0.06, 0.1\}$. Notice that when α is higher, convergence to stable levels is also higher. Unstability is also higher.

some of the servers that take part in the system would be more critic than others. For example, if one of the servers is also a file server for the whole company or group in which it is located, some cost sould be added to the execution of each experiment in that concrete server because its CPU cicles are more valuable than the rest. Another difficult issue, that has not been taken into account yet, is the estimation of the computational complexity for a concrete subexperiment. There will be meta-learning experiments that generate subexperiments with the same computational complexity, and meta-learning experiments that not. This is now under consideration but no approach has been envisaged yet. However, this three strategies offer different mechanism to plan experiments on a concrete application: meta-learning.

6. REFERENCES

[1] Juan A. Botía, A. F. Skarmeta, M. Valdés, and A. Padilla. Data mining applied to irrigation water management. In *Proceedings of the IWANN-2001*, Granada, Spain, June 2001. (to appear).

[2] Juan A. Botia, A. F. G. Skarmeta, Juan R. Velasco, and Mercedes Garijo. A proposal for Meta-learning through Multi-agent Systems. In Tom Wagner and Omer F. Rana, editors, *Lecture Notes in Artificial Inteligence 1887*. Springer, 2000.

[3] Pavel B. Brazdil and C. Soares. A comparison of ranking methods for classification algorithm selection. In *ECML-99*.

[4] Leo Breiman. Bagging predictors. Technical report, Department of Statistics. University of California., Berkeley, California 94720., September 1994.

[5] Philip Kin-Wah chan. *An Extensible Meta-Learning Approach for Scalable and Accurate Inductive Learning*. PhD thesis, School of Art and Sciences. Columbia University, 1996.

[6] Usama Fayyad, Gregory Piatetsky-Shapiro, and Padhraic Smyth. From data Mining to Knowledge Discovery: An Overview. In Usama Fayyad, Gregory Piatetsky-Shapiro, Padhraic Smyth, , and Ramasamy Uthurusamy, editors, *Advances in Knowledge Discovery and Datamining*. AAAI Press, 1996.

[7] Yoav Freund and Robert E. Schapire. A short introduction to boosting. *Journal of Japanese Society for Artificial Intelligence*, 14(5):771–780, September 1999.

[8] A. Gómez-Skarmeta and F. Jiménez. Fuzzy modeling with hybrid systems. *Fuzzy Sets and Systems*, 104(2):199–208, 1999.

[9] W. YeongT. Howes S. Kille. Lightweight directory access protocol. request for comments: 1777. Technical report, Performance Systems International, University of Michigan and ISODE Consortium, March 1995.

[10] DaeEun Kim and Jaeho Lee. Handling Continuous-Valued Attributes in Decision Tree with Neural Network Modeling. In R. López de

Mántaras and E. Plaza, editors, *European Machine Learning Conference*, LNAI 1810, pages 211–219, Heiderberg Berlin., 2000. Springer-Verlag.

[11] R. D. King, C. Feng, and A. Sutherland. STATLOG: Comparison of Classification Algorithms on Large Real-World Problems. *Applied Artificial Intelligence*, 9:289–333, 1995.

[12] Ron Kohavi. A study of cross-validation and bootstrap for accuracy estimation and model selection. In *IJCAI*, pages 1137–1145, 1995.

[13] M. Malek and A. Labbi. Integration of Case-Based Reasoning and Neural Networks Approaches for Classification. Technical report, TIMC-IMAG, LIFIA-IMAG., Grenoble, France., May 1995.

[14] J. L. McClelland and D. E. Rumelhart. *Explorations in Parallel Distributed Processing.* MIT Press, 1988.

[15] Luis Paulo Peixoto. Load distribution: a survey. Technical Report UM/DI/TR/96/03, Departamento de Informática. Escola de Engenharia. Universidade do Minho, 1996.

[16] Salvatore Stolfo, Andreas L. Prodromidis, Shelley Tselepis, Wenke Lee, and Dave W. Fan. JAM:Java Agents for Meta-Learning over Distributed Databases. In David Heckerman, Heikki Mannila, Daryl Pregibon, and Ramasamy Uthurusamy, editors, *The Third International Conference on Knowledge Discovery & Data Mining.* AAAI Press, August 1997.

[17] A. Suyama, N. Negishi, and T. Yamaguchi. Camlet: A platform for automatic composition of inductive applications using ontologies. In *Recent Advances in Meta-learning and Future Work. ICML-99 Workshop Summary.*

[18] David H. Wolpert and William G. Macready. No Free Lunch Theorems for Search. Technical report, Santa Fe Institute, 1995.