Proceedings of the 10th WSEAS International Conference on COMPUTERS, Vouliagmeni, Athens, Greece, July 13-15, 2006 (pp278-284)

Multi-Strategy Learning made easy

FRANCISCO REINALDO[1,2], MARCUS SIQUEIRA[1,2], RUI CAMACHO[1],
LUIS PAULO REIS[1],
[1] LIACC, Faculty of Engineering, University of Porto, Portugal
[2] GIC, Department of Computer Science, Unileste-MG, Brazil

Abstract: - This paper presents the AFRANCI tool for the development of Multi-Strategy learning systems. Designing a Multi-Strategy system using AFRANCI is a two step process. The user interactively designs the structure of the system and then chooses the learning strategies for each module. After providing the datasets all modules as automatically trained. The system is aware and takes into consideration the inter-dependency of the modules. The tool has built-in learning algorithms but can use external programs implementing the learning algorithms.
The tool has the following facilities. It allows any user to design in an interactive and easy fashion the structure of the target system. The structure of the target system is a collection of interconnected modules. The user may then choose the different learning algorithms to construct each module. The tool has several built-in Machine Learning algorithms has interfaces that enables it to use external learning tools like WEKA and CN2. AFRANCI uses the interdependency of the modules to determine the sequence of training. For each module the system uses a wrapper to tune automatically the parameters of the learning algorithm. In the final step of the design sequence AFRANCI generates a compact and legible ready-to-use ANSI C open-source code for the final system.

To illustrate the concept we have empirically evaluated the tool in the context of the RoboCup Rescue domain. We have developed a small system that uses both neural networks, decision trees and rule induction in the same system. The experiment have shown that a very significant speed up is attained in the development of systems when using this tool.

Key-Words: - Artificial Intelligence Tool, Machine Learning, Multi-Strategy Learning, Agents.

1 Introduction

Over the past few years, there has been a significant on the interest in working with heterogeneous learning algorithms in order to achieve fast and complex behaviours in autonomous agents. Autonomous Agents (AA) can be seen as a collection of components or heterogeneous learning modules with well-defined interfaces and fine tuning behaviours. It has been recognised that different behaviours may require different learning strategies.

A popular AA architecture considers several learning algorithms/modules arranged in horizontal or vertical levels to compose behaviours with different levels of abstraction. This layered learning is specially adequate for domains that are too complex for a direct mapping from the input to the output representation to work [9]. This approach brings some new challenges on the arrangement of these modules to achieve inner reasoning, prediction and abstraction methods instead of classical planning research. Additionally, the use of robust tools are necessary to support this kind of approach.

However, when using a conventional programming language, like C or C++ to encode such modules, an expert programmer is often required. Each new user starts a project from scratch, and occasionally it results in a bad program code structure revealing problems whenever the code needs to be extended or updated. To overcome such difficulties some tools propose uniform and easily modelling facilities for the learning modules. Such tools include Matlab© [6] and SNNS (Stuttgart Neural Network Simulator)© [14] [13]. These tools are limited as far as the user interface aspects are concerned since these are often neglected. In addition, the tools do not support the design of several modular, hierarchic and complex structures of heterogeneous learning modules in the same environment as AFRANCi does.

The AFRANCI tool [7] offers solutions for assembling and linking together graphically on the screen tel modules required to create large scale behaviour-based systems. In the development of AA applications there is the need to aggregate different kinds of behaviours working together to increase the complexity of behaviours. The user's choice is not restricted to low levels but is free to choose the suitable abstraction level. Another advantage of AFRANCI is the possibility of using external learning tools/algorithms such as the WEKA© [11] library to compose heterogeneous learning modules. The tool makes easily adjustable and extensible connection with WEKA learning modules without the user's perception. These aspects will be presented in Section 2.

According to the proposed objectives, this work has enabled the development of a tool that can: a) support graphic designs of large and extensible behaviour-based architectures composed of heterogeneous clusters with cognitive processes; b) make available a wide set of varied functionalities of control modules; c) offer facilities and resources to interconnect several structures of external control processes so that they could communicate in a multi-environment; d) arrange learning modules in horizontal or vertical levels to insert new or substitute the previous agent characteristics by an adopted technique from software engineering [7] [12]; e) offer parallelized network training; f) automatically generate a ready-to-use ANSI C open-source code from the screen.

The contribution of this paper goes beyond a object-oriented implementation of ideas [10]. It addresses abilities, specialisations and policies to accomplish specific tasks in an Autonomous Agent setting by using a multi-strategy learning in order to achieve complex behaviours. We offer a scalable modelling and parallel environments with the purpose of speeding up behaviour agent simulations. Furthermore, the tool is a valuable contribution to users without knowledge about deep programming language and learning modules because it offers high level of abstraction and automatic creation of source-code, as observed in 5.

The rest of the paper is organised as follows. Section 2 presents the tool to design behaviour structures or interconnected architecture in order to achieve the best agent performance. Experiments, results and discussion are showed in Section 5. Section 6 presents an overview of AFRANCI architectures and how it handles the generated system's modules. Finally, some conclusions are presented in Section 7.

2 AFRANCI Tool

The AFRANCI Tool [7] is a tool built over some classes of open-source ANSI C Pyramid-Net Framework [8] platform, which extensions for using external libraries of learning algorithms like the WEKA© library. Pyramid-Net was restricted to a subset of Neural Network algorithms whereas with the ability to use WEKA© library the user has access to a useful repository of machine learning algorithms for data pre-processing, classification, regression, clustering, association rules and visualisation.

The AFRANCI tool is composed of three main parts, which are the Graphic User Interface (GUI), the Machine Learning Modules (MLM) and the Automatic Open-Source Code Generator (ACG). First, GUI is a set of

main classes for handling and modelling learning modules, using graphic elements that will interact with the user. MLM implements the construction of the modules using the chosen learning algorithms. Finally, ACG receives a description of pictorial representation of the system structure and produces the short automatic open-source code.

The tool allows users to design and implement behaviour-based architectures through the interconnection of elementary heterogeneous control modules in the form of a circuit diagram. New structures can be constructed by linking standard structures or modules together. This form is very familiar to engineers in digital systems design and in model analysis systems tools (e.g. SIMULINK©). It makes some items available, such as desktop constructor, sensor, actuator, line links, skins of learning modules, menus, dialogue box. Each graphical element can be dragged and dropped to a re-sizable screen and its features (colour, label, format and size) be adjusted . Heterogeneous control modules are standard boxes with input and output of data. For instance, if the user is working with ANN, he/she has access to a number of hidden neurons, neuron transfer functions, training parameters, etc.

With the purpose of speeding up the implementation of learning modules, the tool offers three other important features: the multi parallel environment, the automatic network design of modules and the automatic development of open-source code. First, each new environment includes a powerful parallel resource to training the proposed designed architecture. The user can link and put together several sub-projects and run them, at the same time because it is free of manual scheduler. In the design process, once adjustments have been made, the module is unlighted and the training and simulation can be run again quickly. The second feature is the on screen automatic/wizard design of networks of learning modules by importing of CSV (Comma Separated Values) files. When using WEKA library, the user links standard learning modules together; each learning module can be simulated at any level of abstraction for fine tuning of the assembled system. The user needs little knowledge about it. As a last step the tool produces a clean and ready-to-use ANSI C open-source for information fusion, planning and coordination with a few mouse clicks. By using a high-performance interpretation algorithm, this functional but compact ANSI C executable core is created from the drawn project. This ANSI C code can be edited on screen in order to be modified and compiled easily in different operating systems.

3 Designing a Structure

In this section we briefly analyse the simple steps to design, train and obtain a finished system composed of different learning modules.

First, it is necessary to design the agent's structure. The user should plan how many modules, how they are interconnected and what algorithms to use in each module. In the next step the user draws a project in the tool desktop (screen). The project must have input, control modules, outputs and links. After designing the project, every input receives a database file. The database file stores data to feed the learning modules in the training phase.

The interconnections are established between inputs, control modules and outputs to make a complete wired network. The user can connect two environments: from the output of modules in the first environment to the input of other assembled of modules in a second environment. This tool allows the user to interconnect everything without loss of performance because it uses a parallel environment. The user can tune each module by changing the default module parameters. Careful is needed in this step because a badly planned set up may lead to the emergence of wrong reasoning processes. This step is the only one that requires a little more knowledge about arranging of modules.

The training process is simple and consists of training and fine tuning of the behaviour-based architecture to achieve the agent's goal.

The automatic training process is responsible for almost everything and based on the data flux sequence to trigger modules to training. Independently of the horizontal or vertical architecture level, the user can follow the iteration training process of each module by graphics, data windows and others.

Finally, the user can generate a ready-to-use ANSI C open-source to plug it in the agent. The source code is a codification of the graphic modelling. Inside of this code, user will find input and output connections, weights, an activation functions, in the case of ANN, a compact main executable core, and some input and output matrixes. The code can be compiled using any C compiler because it uses ANSI C. In conclusion, these quick steps help user to substitute large programs, time consuming projects and confusing lines of code.

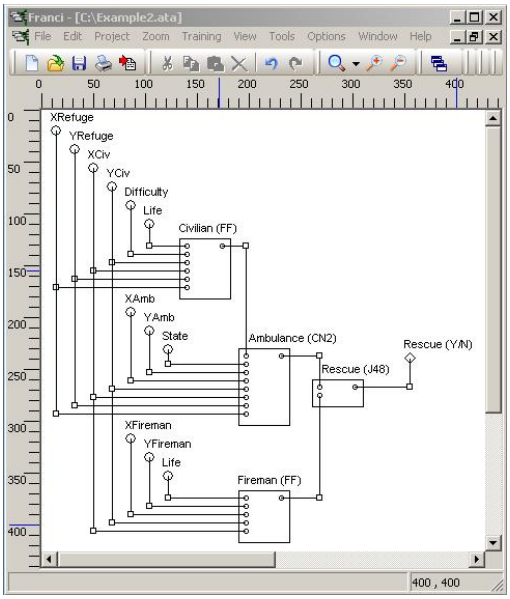

Fig. 1: System with two ANNs, a Rule Induction module and a Decision Tree module designed with AFRANCI.

4 Wrappers

Almost all Machine Learning systems have parameters that must be tuned to achieve a good quality of the constructed model. An experienced practitioner knows that changes in parameter's values may lead to quite different results. To tune a system's parameters requires knowledge of the system. This is most often an impediment to a wider use by non-experts since they have no feeling on what to do.

One approach to overcome such a situation is by the use of a *wrapper* [3]. A *wrapper* produces several models using different combinations of the leaning algorithm and returns the "best" model. In our tool the wrapper optimises the test set error rate estimation. This automatic tuning of parameter completely hides the details of using the learning algorithms from the user. It is therefore a way to make the tool usable by a wider ranger of users.

As future work we intend to extend the use of *wrappers* to do feature (subset)-selection as in [4] and [5]. This facility would require however a tighter relationship between the modules synthesis and their inter-connection. If the wrapper decided that some feature is not relevant for the classifier then that input would have to be removed in the modules inter-connection design stage.

5 Experiments

To illustrate the features of AFRANCI and facilitate the reader's understanding we generated a simple artificial problem and dataset in the RoboCup Rescue setting. The problem we devised is to decide if a ambulance or fireman should rescue or not a civilian to a nearest refuge. The civilian is somewhere in a burning building. The decision is commonly made on localisation and agent's and civilian's life conditions. The independent variables include the position (X, Y) of the ambulance, fireman and of the civilian person, the life measure of the fireman[1], the life measure of the civilian person, the state of the ambulance (busy/free) to receive the civilian, the difficulty of the civilian rescue situation and the position (X, Y) of the nearest refuge (rescue building).

The system devised is composed by two modules that encode the decision of the fireman and the civilian, a module that encode decision of the ambulance and a third module to combine fireman and ambulance deci-

[1]A measure between 0 and 100 of the energy the fireman can use.

```
J48 pruned tree
------------------
ambulance_apt = TRUE: RescueAmbulance (7.0)

ambulance_apt = FALSE: RescueFireman (3.0/1.0)
```

a)

```
IF    occupied = occupied
THEN  class = n  [52 0]
ELSE
IF    civilian = notapt
THEN  class = n  [20 0]
ELSE
IF    Yamb > 8164.50
  AND Xciv < 9204.50
THEN  class = y  [0 7]
ELSE
IF    Yamb > 340.00
  AND 386.00 < Yciv < 7731.50
  AND Xrefuge > 3800.00
THEN  class = n  [9 0]
ELSE
IF    Yamb < 5721.00
  AND Xrefuge > 866.00
THEN  class = y  [0 8]
ELSE
(DEFAULT) class = n  [4 0]
```

b)

Fig. 2: a) Decision Tree generated by WEKA's J48 learner. b) Rule set generated by CN2.

sions. Figure 1 shows the modular and heterogeneous structure of the system constructed interactively by the user. In Figure 2 we can see the contents of two modules as generated by (a) J48 Decision Tree generator and (b) CN2 Rule inducer.

The fireman and civilian decision modules are encoded using AFRANCI built-in ANN (Feedforward) whereas the ambulance module was constructed using CN2 Induction algorithm [1] and last module was constructed using WEKA's J48 Decision Tree algorithm. The user had only to select the number and place in the window of the modules, to connect them and choose input and output variable. All this was done using drag-and-drop operations. He then provided the dataset and the tool trained the modules in the correct sequence and generated a C program that encodes the system.

The action of deciding which agent will be responsible for rescuing the injured civilian is taken in the Decision Tree module. In Figure 1, the fireman agent will perform a rescue action only if the ambulance agent will not be able to so. In the case of both the fireman and ambulance are capable of rescuing the civilian, the module decides in favour of the ambulance agent because a fireman agent has to extinguish fires in burning buildings with the aim of preserving the city. In order to verify if the ambulance is entirely apt to rescue a civilian, Figure 2, the rules induced by CN2 establish that: (rule 1) if the ambulance is occupied then it is useless to attempt the rescue; (rule 2) if the civilian has not enough "energy" then it is also not rescued; (other rules) the civilian will be rescued if it has enough "energy" and the ambulance is between the civilian and the rescue place otherwise it will not be rescued.

6 Related Work

In order to develop a robust tool, we analysed two relevant ones. The MatLab© SIMULINK© tool was mainly concerned with the offer of a detailed design of a control process [2] and to automatically generate a ready-to-use code for it. However, that tool has two main limitations that affect the design of the whole project. First, the user cannot design a complex structure composed of several heterogeneous modules nor a interconnected architecture because the tool does not offer facilities nor a specific multi-environment to work with. These limitations inhibit the user's ability to handle different levels of abstraction, several processes in the same environment. Second, the ready-to-use code was generated from a simple learning algorithm is large to be worked, complex to be understood and unpractical to be used. Other studied tool was the SNNS© (Stuttgart Neural Network Simulator) [14] [13]. This tool is a software simulator exclusively to work with Artificial Neural Networks (ANN) in order to create their applications. Although the tool offers a good graphic environment, its repertory

is limited to only a behaviour level a at time, without offering a parallel environment to train the ANN. Moreover, it is limited concerning expandability or new ways to graphically preview the whole structure that is being worked with.

In Section 2, we presented the tool which offers a user friendly parallel graphic environment that facilitates the development and training of Modular and Hierarchic Systems by including several modules with different learning algorithms. The systems architecture produced with the AFRANCI tool promoted the multi-strategy learning as long as various learning modules are used to compose the structure.

7 Conclusions

In this paper we described a tool for the development of multi-strategy learning systems. Using a friendly graphical interface the user may define the modular structure of the system and choose the learning algorithms to construct each module. He then provides the dataset and lets the tool train, in the correct sequence, each module and produce a complete self containing program encoded in the C language.

The tool provides several learning algorithms and is able to call external learning algorithms to construct the modules. The deployment of the tool confirmed the assumption that it was easy and fast to develop multi-strategy system with AFRANCI. We tested it using ANNs (built-in), Decision Trees (external, J48 – WEKA) and Rule Induction (external - CN2) algorithms. In our implementation the user does not have to tune any of the module's parameters. That is done automatically by a *wrapper*.

As future work we intend to extend the use of *wrappers* to do feature (subset)-selection.

References:

[1] Clark, P. & Niblett, T. The cn2 induction algorithm. *Machine Learning 3*, 4 (1989), 261–283.

[2] Dorf, R. C. & Bishop, R. H. *Modern control systems*. Tenth, 2004.

[3] John, H. G. Cross-validated c4.5: Using error estimation for automatic parameter selection. Technical note stan-cs-tn-94-12, Computer Science Department, Stanford University, California, October 1994.

[4] Kohavi, R. *Wrappers for Performance Enhancement and Oblivious Decision Graphs*. Tese de Doutorado, Stanford University, 1995.

[5] Kohavi, R. & Summerfield, D. Features subset selection using the wrapper method:overfitting and dynamic search space topology. In *First International Conference on Knowledge Discovery and Data Mining (KDD-95)*, 1995.

[6] Mathworks. *Matlab*. Mathworks, Inc, Natick, MA, 1999.

[7] Reinaldo, F.; Camacho, R. & Reis, L. P. Afranci: An architecture for learning agents. Phd report, FEUP, Porto, Portugal, August 2005.

[8] Reinaldo, F. A. F. Projecting a framework and programming a system for development of modular and heterogeneous artificial neural networks. Dept. of computer science, Federal Univ. of Santa Catarina, Florianopolis, Brazil, Feb 2003.

[9] Stone, P. & Veloso, M. M. Layered learning. In *Machine Learning: ECML 2000, 11th European Conference on Machine Learning, Barcelona, Catalonia, Spain, May 31 - June 2, 2000, Proceedings*, 2000, vol. 1810, Springer, Berlin, p. 369–381.

[10] Wirfs-Brock, R. J. & Johnson, R. E. Surveying current research in object-oriented design. *Commun. ACM 33*, 9 (1990), 104–124.

[11] Witten, I. H. & Frank, E. *Data Mining: Practical machine learning tools and techniques*, 2nd. ed. Morgan Kaufmann, San Francisco, 2005.

[12] Wray, R.; Chong, R.; Phillips, J.; Rogers, S. & Walsh, B. *A Survey of Cognitive and Agent Architectures.* University of Michigan, 1994.

[13] Zell, A.; Mache, N.; Huebner, R.; Schmalzl, M.; Sommer, T. & Korb, T. SNNS: Stuttgart neural network simulator. Relatório técnico., Stuttgart, 1992.

[14] Zell, A.; Mache, N.; Sommer, T. & Korb, T. Design of the snns neural network simulator. In *7. Österreichische Artificial-Intelligence-Tagung*, H. Kaindl, Ed. Springer, Berlin, Heidelberg, 1991, p. 93–102.