

A tool for Multi-Strategy Learning

Francisco Reinaldo^{1,2}, Marcus Siqueira^{1,2}, Rui Camacho¹, and Luís Paulo Reis¹

¹ LIACC, Faculty of Engineering, University of Porto, Portugal

² GIC, Department of Computer Science, UnilesteMG, Brazil
reifeup,rcamacho,lpreis@fe.up.pt, marvinsiq@gmail.com

Abstract. This paper presents the AFRANCI tool for the development of Multi-Strategy learning systems. AFRANCI allows users to build, in an interactive and easy way, complex systems. Systems are built using a two step methodology: design of the structure of the system; and fill in the modules. The structure of the target system is a collection of interconnected modules. The user may then choose among a variety of learning algorithms to construct each module. The tool has several built-in Machine Learning algorithms and interfaces that enable it to use external learning tools like WEKA or CN2. AFRANCI uses the interdependency of the modules to determine the sequence of their training. To improve usability, the tool uses a wrapper that hides from the user the parameter tuning procedure for each algorithm. In a 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 and rules 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.

1 Introduction

Over the past few years, there has been a significant increase in the interest of working with heterogeneous learning algorithms in order to achieve fast and complex behaviours in autonomous agents. An Autonomous Agent (AA) can be seen as a collection of components of heterogeneous learning modules with well-defined interfaces and fine tuned 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 and/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 [1]. 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. Although not in the context of AA

construction, but also using a hierarchical organisation of modules, Alan Shapiro [2] proposes the use of “structured induction” to address the problem of inducing complex concepts.

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© [3] and SNNS (Stuttgart Neural Network Simulator)© [4] and [5]. 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 [6] offers solutions for assembling and linking together graphically on the screen the 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 [7] library or CN2 induction algorithm [8] to compose heterogeneous learning modules. The tool makes easily adjustable and extensible connection with WEKA learning modules and CN2 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 [6] and [9]; e) offer parallelised 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. 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 Section 5.

The rest of the paper is organised as follows. Section 2 presents an overview of AFRANCI architectures and how it handles the generated system’s modules.

Section 3 describes how to design behaviour structures in the interconnected architecture in order to achieve the best agent performance. The use of wrappers is described in Section 4. Experiments, results and discussion are presented in Section 5. Related work is presented in Section 6. Finally, some conclusions are drawn in the last section.

2 AFRANCI Tool

The AFRANCI Tool [6] is a tool built over some classes of open-source PyramidNet Framework [10] platform, which extends the use to external libraries of learning algorithms like the WEKA[©] library and CN2 induction algorithm. PyramidNet was restricted to a subset of Artificial Neural Network (ANN) algorithms whereas with the new ability to use WEKA[©] library and CN2 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 net-

works of learning modules by importing of CSV (Comma Separated Values) files. When using ANN/WEKA/CN2, 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 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 module receives the database file that stores data to feed it 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. Summing up, these quick steps help user to substitute large programs, time consuming projects and confusing lines of code.

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 the parameter's values may lead to quite different results. To tune a system's parameter requires knowledge of the system. This is most often a severe obstacle to the wide spread use of such algorithms.

As proposed by John [11] one possible approach to overcome such a situation is by the use of a *wrapper*. A *wrapper* produces several models using different combinations of the learning 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 range of users.

When the total number of combinations of parameter's values is small³ the tool tries all the combinations and chooses the best one. When the number of combinations is larger the parameters are tuned using a Genetic Algorithm (GA) [12] with mutation and cross-over GA operators. For the ANN we also choose automatically their structure. A GA is used to choose the best number of hidden layers and number of neurons in each layer for each ANN. The type of ANN is also included in the automatic choice.

As future work we intend to extend the use of *wrappers* to do feature (subset)-selection as in [13] and [14]. 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 based on the localisation and agent's and civilian's life conditions. The main independent variables include: the position (X, Y) of the ambulance, fireman, building on fire, fire brigade, the nearest refuge (rescue building), and of the civilian; the life condition measure of the fireman and civilian⁴, the building volatile information is composed of earlier burnt, state and structure; the state of the ambulance (busy/free) to receive the civilian and of the fireman (busy/free) to extinguish the fire or to rescue the civilian; a measure that assesses the difficulty of the civilian rescue situation.

The system devised is composed by several modules that encode the decision of the fireman and the civilian and particularly a main module to combine fireman and ambulance decisions. Figure 1 shows the modular structure of the

³ In the current implementation *small* is less than 20.

⁴ A measure between 0 and 100 of the energy the fireman can use.

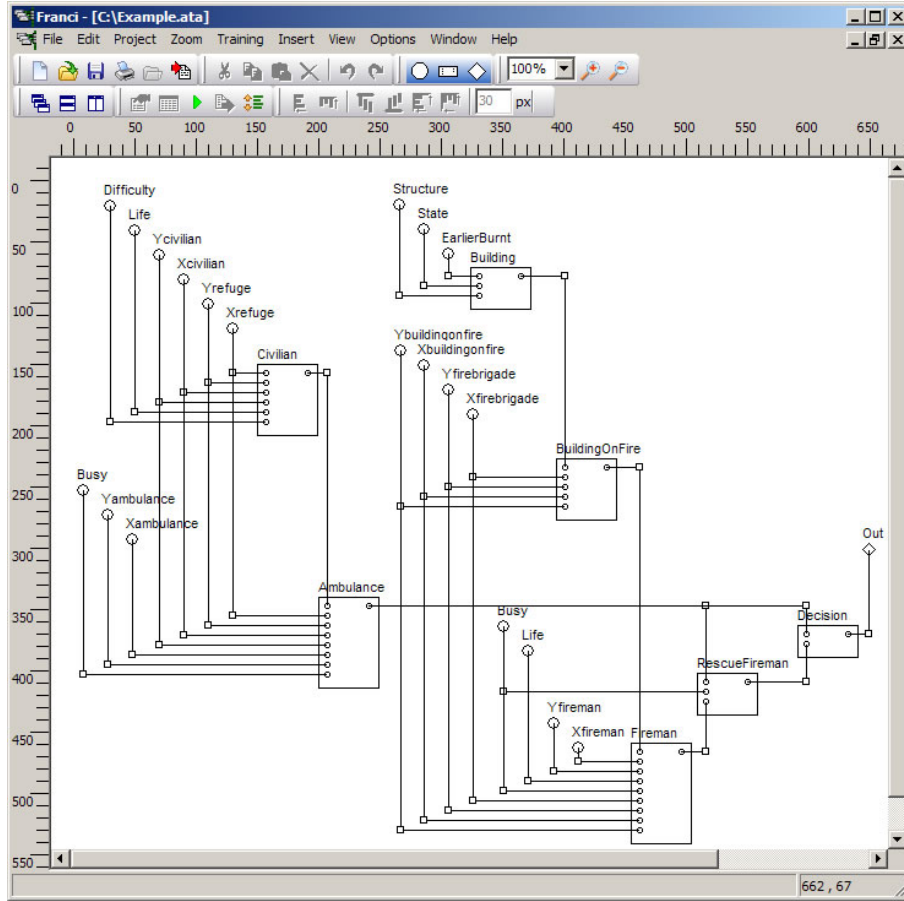


Fig. 1. Architectural Design.

system constructed interactively by the user. Not perceived in the figure is the heterogeneity of the modules. Different modules use different representations and where induced using different learning algorithms. Referring to the module's labels in Figure 1 the following algorithms were used. The Civilian module was constructed using AFRANCI built-in feedforward ANN. The Ambulance, BuildingOnFire, Fireman and RescueFireman modules were constructed using CN2 rule learner. The other modules, Building and Decision, were 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 names. 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.

<pre> J48 pruned tree ----- ambulance_apt = TRUE: RescueAmbulance ambulance_apt = FALSE: RescueFireman </pre>	<pre> 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] </pre>
a)	b)

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

The action of deciding which agent will be responsible for rescuing the injured civilian is taken in the Decision module (see Figure 1). The fireman agent will perform a rescue action only if the ambulance agent will not be able to do 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 check if the ambulance is entirely apt to rescue a civilian (see 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 "vitality" 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 Mat-Lab © and SIMULINK © tools are mainly concerned with the offer of a detailed design of a control process [15] 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 software simulator was the SNNS © (Stuttgart Neural Network Simulator) [4] [5] that it exclusively works with 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 time, without offering a parallel environment to train the ANNs. 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 system 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 produces 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.

As future work we intend to develop *wrappers* (John, 1994b) to automatically tune the modules parameters, optimising the test set error rate estimation.

References

1. Peter Stone and Manuela M. Veloso. Layered learning. In *Machine Learning: ECML 2000, 11th European Conference on Machine Learning, Barcelona, Catalonia, Spain, May 31 - June 2, 2000, Proceedings*, volume 1810, pages 369–381. Springer, Berlin, 2000.
2. A.D. Shapiro. *Structured Induction in Expert Systems*. Adison-Wesley, Wokingham, 1987.
3. Mathworks. *Matlab*. Mathworks, Inc, Natick, MA, 1999.

4. A. Zell, N. Mache, T. Sommer, and T. Korb. Design of the snns neural network simulator. In H. Kaindl, editor, *7. Österreichische Artificial-Intelligence-Tagung*, pages 93–102. Springer, Berlin, Heidelberg, 1991.
5. Andreas Zell, Niels Mache, Ralf Huebner, Michael Schmalzl, Tilman Sommer, and Thomas Korb. SNNS: Stuttgart neural network simulator. Technical report, Stuttgart, 1992.
6. Francisco Reinaldo, Rui Camacho, and Luís Paulo Reis. Afranci: An architecture for learning agents. Phd report, FEUP, Porto, Portugal, August 2005.
7. Ian H. Witten and Eibe Frank. *Data Mining: Practical machine learning tools and techniques*. Morgan Kaufmann, San Francisco, 2nd. edition, 2005.
8. P. Clark and T. Niblett. The CN2 algorithm. *Machine Learning*, (3):261–283, 1989.
9. Robert Wray, Ron Chong, Joseph Phillips, Seth Rogers, and Bill Walsh. *A Survey of Cognitive and Agent Architectures*. University of Michigan, 1994.
10. F. A. F. Reinaldo. 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.
11. H. George John. 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.
12. J.H. Holland. *Adaption in Natural and Artificial Systems*. University of Michigan Press, Ann Arbor, Michigan, 1975.
13. Ron Kohavi. *Wrappers for Performance Enhancement and Oblivious Decision Graphs*. PhD thesis, Stanford University, 1995.
14. Ron Kohavi and Dan Summerfield. 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.
15. Richard C. Dorf and Robert H. Bishop. *Modern control systems*. Tenth, 2004.