Chapter 1

ECONOMIC EXPERIMENTS WITH SWARM: A NEURAL NETWORK APPROACH TO THE SELF-DEVELOPMENT OF CONSISTENCY IN AGENTS' BEHAVIOR

Pietro Terna
University of Turin,
Department of Economics and Finance G.Prato,
corso Unione Sovietica 218bis, 10134 Torino, Italy
terna@econ.unito.it

Abstract We underline the usefulness of agent based models in the social science perspective, also focusing on the main computational problems due to the structure of our models: to simplify the task we introduce a generalized Environment-Rules-Agents scheme. Finally, within Swarm, we introduce a neural network tool (Cross Target method), useful in building artificial laboratories, for experiments with learning, self-developed consistency and interaction of agents in artificial worlds, in order to observe the emergence of complexity without a priori behavioral rules.

1. INTRODUCTION

The goal of this work is that of proposing a tool, the **bp–ct** package written in Swarm, as an easy way in building and running artificial laboratories for social scientists.

Section 2., "Agent based models in the social science perspective", introduces some considerations on agent based models and underlines their usefulness in the social science perspective, considering also the wide field of computational economics; Section 3., "The ERA scheme", deals with the main computational problems due to the structure of our models and introduces a generalized Environment–Rules–Agents scheme

(ERA), here primarily used to build the bp–ct package; Section 4., "Neural networks", briefly explains the idea of networks of processing elements as learning computational tools; in Section 5., "The Cross–Target (CT) method", we show the construction of artificially intelligent agents founded upon neural network based algorithms, which can be modified by a trial and error process without a priori behavioral rules; in Section 6., "The Swarm bp–ct package", the new package is explained.

Finally, in Section 7., "An experiment with the CT scheme of actions and effects, to obtain a spontaneous hayekian market", an experiment is built and run, obtaining a no–auctioneer hayekian market; Section 8. plans for "Future improvements".

2. AGENT BASED MODELS IN THE SOCIAL SCIENCE PERSPECTIVE

Choosing the agent based model paradigm we enter a wide unexplored world where methodology and techniques are largely "under construction." We are here working in the bottom–up direction, putting together pieces of software (our agents) which can react to stimuli from the environment or from other agents. Why to do so?

An interesting overview comes from the following Web sites: Syllabus of Readings for Artificial Life and Agent–Based Economics[1]; Web Site for Agent–Based Computational Economics[2]; Agent–Based Economics and Artificial Life: A Brief Intro[3]; Complex Systems at the University of Buenos Aires[4]; Computational Economic Modeling[5]; Individual–Based Models[6]. We can also refer to works in the social simulation perspective (Conte *et al.*, 1997), artificial intelligence (Russel and Norvig, 1995) and in economics (Beltratti *et al.*, 1996); also of interest is The Complexity Research Project of the London School of Economics and Political Science[7].

At present, the best plain introduction to agent based modeling techniques is reported in the introduction to the Sugarscape model (Epstein and Axtell, 1996). As Epstein and Axtell (Chapter 1) note:

> Herbert Simon is fond of arguing that the social sciences are, in fact, the hard sciences. For one, many crucially important social processes are complex. They are not neatly decomposable into separate subprocesses–economic, demographic, cultural, spatial–whose isolated analyses can be aggregated to give an adequate analysis of the social process as a whole. And yet, this is exactly how social science is organized, into more or less insular departments and journals of economics, demography, political science, and so forth (...)
>
> The social sciences are also hard because certain kinds of controlled experimentation are hard. In particular, it is difficult to test hypotheses concerning the relationship of individual behaviors to macroscopic reg-

> ularities, hypotheses of the form: If individuals behave in thus and such a way–that is, follow certain specific rules–then society as a whole will exhibit some particular property. How does the heterogeneous micro–world of individual behaviors generate the global macroscopic regularities of the society?
>
> Another fundamental concern of most social scientists is that the rational actor–a perfectly informed individual with infinite computing capacity who maximizes a fixed (nonevolving) exogenous utility function–bears little relation to a human being. Yet, there has been no natural methodology for relaxing these assumptions about the individual.
>
> Relatedly, it is standard practice in the social sciences to suppress real–world agent heterogeneity in model–building. This is done either explicitly, as in representative agent models in macroeconomics (Kirman, 1992), or implicitly, as when highly aggregate models are used to represent social processes. While such models can offer powerful insights, they filter out all consequences of heterogeneity. Few social scientists would deny that these consequences can be crucially important, but there has been no natural methodology for systematically studying highly heterogeneous populations.
>
> Finally, it is fair to say that, by and large, social science, especially game theory and general equilibrium theory, has been preoccupied with static equilibria, and has essentially ignored time dynamics. Again, while granting the point, many social scientists would claim that there has been no natural methodology for studying nonequilibrium dynamics in social systems.

The response to this long, but exemplary, quotation, is exactly social simulation and agent based artificial experiments. This means being capable of starting from the basis, with humility, to have our agents interacting, to observe what emerges (if something does) from their action. Partially this is the same exercise done when we explore the wide world of computational economics; computational economics is a complement of the theory (Judd, 1996), since verifies and explains theoretical issues numerically solving and iterating equations, with true or guessed parameters. But, as Judd points out, we have here also the possibility of using computational (or agent based, my addition) results as results at all, as they would be theorems:

> The increasing power of computers presents economic science with new opportunities and potential. However, as with any new tool, there has been discussion concerning the proper role of computation in economics. Some roles are obvious and noncontroversial. Most will agree that computation is necessary in econometric analysis and offers some guidance in policy discussions. Theorists will admit that examples are useful in illustrating general results. However, the discussion frequently gets heated when one raises the possibility of using the computer and computer–generated "examples" instead of the classical assumption–theorem–proof form of analysis for studying an economic theory. In

> economic terms, the first roles for computation are complements to and a useful ingredient in standard research activities; however, the activity of computational theory appears to be a substitute for conventional theoretical analysis. In this essay, I will focus on the potential role of computational methods in economic theory and their relation to standard theoretical analysis, asking "Are they complements or substitutes?"

But how to build an agent based model? Basically, we have to construct (Gilbert and Terna, 1999)a computer model in which agents (pieces of software) operate (behave).

First, we must consider the agents as objects: i.e., pieces of software capable of containing data and rules to work on the data. The rules provide the mechanisms necessary to react to messages coming from outside the object.

Second, we have to observe the individual agents' behaviour via the internal variable states of each agent–object and, at the same time, the results arising from their collective behaviour. One important feature of the software is therefore to synchronise the experiment clocks: we have to be sure that the observations made at the aggregate level and the knowledge that we are picking up about the internal states of the agents are consistent in terms of the experimental schedules.

Now let us suppose that a community of agents, acting on the basis of public (i.e. common to all agents) and private (i.e. specific to each agent) information and of simple internal local rules, shows, as a whole, an interesting, complex, or "emergent" behaviour. There are two kinds of emergence: unforeseen and unpredictable emergence. An example of unforeseen emergence occurs when we are looking for an equilibrium state, but some sort of cyclical behaviour appears; the determinants of the cyclical behaviour are hidden in the structure of the model. Unpredictable emergence occurs when, for example, chaos appears in the data produced by an experiment. Chaos is obviously observable in true social science phenomena, but it is not easy to reverse engineer in an agent–based simulation.

3. THE ERA SCHEME

Swarm is a natural candidate to this kind of structures, but we need some degree of standardization, mainly when we go from simple models to complex results. Here a crucial role for the usefulness and the acceptability of the experiments is played by the structure of the underlying models. For this reason, we introduce here a general scheme that can be employed in building agent–based simulations.

The main value of the Environment–Rules–Agents (ERA) scheme shown in Figure 1.1 is that it keeps both the environment, which models

the context by means of rules and general data, and the agents, with their private data, at different conceptual levels. To simplify the code, we suggest that agents should not communicate directly, but always through the environment; as an example, the environment allows each agent to know the list of its neighbours. This is not mandatory, but if we admit direct communication between agents, the code becomes more complex.

With the aim of simplifying the code design, agent behaviour is determined by external objects, named Rule Masters, that can be interpreted as abstract representations of the cognition of the agent. Production systems, classifier systems, neural networks and genetic algorithms are all candidates for the implementation of Rule Masters.

We may also need to employ meta–rules, i.e., rules used to modify rules (for example, the training side of a neural network). The Rule Master objects are therefore linked to Rule Maker objects, whose role is to modify the rules mastering agent behaviour, for example by means of a simulated learning process. Rule Masters obtain the information necessary to apply rules from the agents themselves or from special agents charged with the task of collecting and distributing data. Similarly, Rule Makers interact with Rule Masters, which are also responsible for getting data from agents to pass to Rule Makers.

Agents may store their data in a specialized object, the DataWarehouse, and may interact both with the environment and other agents via another specialized object, the Interface (DataWarehouse and Interface are not represented in Figure 1.1, having a simple one to one link with their agent; they are used in the bp–ct package and in its derivatives).

Although this code structure appears to be complex, there is a benefit when we have to modify a simulation. The rigidity of the structure then becomes a source of invaluable clarity. An example of the use of this structure can be found in the code of the Swarm application bp–ct in which the agents are neural networks; also the specialized code related to the hayekian experiment reported in Section 7., being built upon bp–ct, uses the same structure. The codes of these packages are included in the CD–Rom attached to this book; future releases can be obtained directly from the author (may be also from the anarchy section of the Swarm site).

A second advantage of using the ERA structure is its modularity, which allows model builders to modify only the Rule Master and Rule Maker modules or objects whenever one wants to switch from agents based on neural networks, to alternatives such as production systems, classifier systems or genetic algorithms.

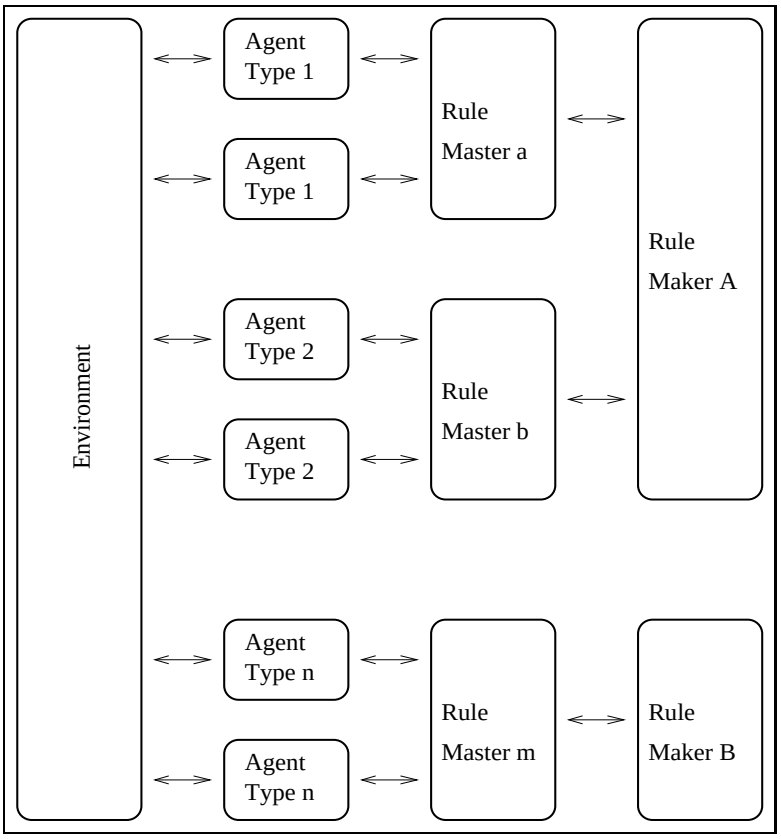

Figure 1.1 The ERA scheme.

In Swarm terms, the Environment coincides with the ModelSwarm object, while the ObserverSwarm object is external to this structure.

The bp–ct code represents a basic package from which we can derive any sort of application, via: (i) the modification of Interface objects if we are keeping unchanged the neural network choice and the CT (Section 5.) framework or we are using any internal system of rules producing inputs and targets for the neural networks; (ii) the substitution of external training and verification sets, if we are using the package as a backpropagation tool in the neural network field; (iii) the modification of RuleMaster and RuleMaker objects, if we are shifting to other paradigms, as described above.

4. NEURAL NETWORKS

The choice of neural networks (Rumelhart and McClelland, 1986) as standard tool for the implementation of bp–ct relies on the capabilities of these vectorial functions, specified in equation (1.1), as structures very close, in econometric terms, to nonlinear multidimensional regression functions.

$$\mathbf{y} = f\left(\mathbf{B}\ f\left(\mathbf{Ax}\right)\right) \tag{1.1}$$

We can imagine a class of simple Artificial Neural Networks (ANN), the multilayer perceptron or feed forward network, as a system of connections linking several processing units of the kind shown in Figure 1.2. The processing element (PE) can be considered as an over simplified sketch of a natural neuron, where the parameters w_{kj} represent the strength of the synapses. The a_j values ("activations") are the inputs of the PE or neuron, which calculates their weighted sum (this operation explains the use of the conventional name "weight" to designate the parameters of a function in the ANN jargon). The same operation, in matrix algebra, is done by the product $\mathbf{Ax}$, where the $\mathbf{x}$ vector contains the a_j activations and the matrix $\mathbf{A}$ the w_{kj} parameters. The PE produces an output applying a non linear transformation to the weighted sum, as that shown in equation (1.2), where k is the parameter that determines the closeness of the function to a crude step function.

$$f(z) = \frac{1}{1 + e^{-kz}} \tag{1.2}$$

A simplified form of ANN, not used here, does not consider the external transformation function and assumes the form $\mathbf{B}\ f(\mathbf{Ax})$.

Each row of $\mathbf{A}$ and $\mathbf{B}$ contains the weights of a PE. In Figure 1.3 we have an example of a network with an input layer of four PEs (from 1 to 4), an internal or hidden layer with three PEs (from 5 to 7) and an output layer with two PEs (from 8 to 9). Note that without the hidden layer the ANN would be a trivial non linear structure of the form $f(\mathbf{Ax})$.

As an example: the matrix $\mathbf{A}$ (2 rows and 3 columns) in Figure 1.3 contains the w_{kj} values of the PEs from 8 to 9, such as w_{97} in the Figure; the matrix $\mathbf{B}$ (3 rows and 4 columns) contains those of the PEs from 5 to 7; the $\mathbf{x}$ vector contains the four values of the input PEs, which are here simply the receptors of the stimuli arriving to the ANN from outside.

It is useful to consider also an input PE and a hidden PE as a constant value (e.g. always 1) to allow the definition of a w parameter operating as

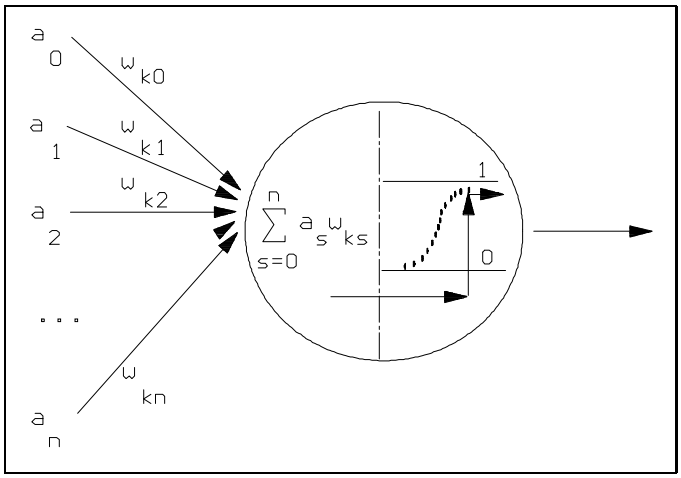

Figure 1.2 A Node or Artificial Neuron

a bias value in the $-kz$ exponent of the logistic function (1.2), where $z = \sum_{s=1}^{m} a_s w_{ks}$, obtaining the correction $-k\,(z+b)$ with $b = w_{k0}$, directly as $-k\widetilde{z}$ with $\widetilde{z} = \sum_{s=0}^{m} a_s w_{ks}$. Formally this structure can be expressed as in equation (1.3).

$$\mathbf{y} = f\left(\mathbf{B}\ \left(1, f\left(\mathbf{A}\left(1, \mathbf{x}'\right)'\right)'\right)'\right) \tag{1.3}$$

To find the **A** and **B** values (i.e. the ANN parameters) we have to use a numerical approximation with a simplified gradient method, the so–called backpropagation method (which comes from "propagating back" the error measures from the output layer to the hidden one), starting with random parameters and adapting the performances of the ANN function on the basis of a set of training examples. Other methods also exist, but the backpropagation (the bp part of the name of our code) is used here for its simplicity and robustness. A detailed explanation of the bp method is contained in the comments of the RuleMaker.m file of the bp–ct code.

The process of parameter determination is also called "learning", because the quality of the result is obviously related to the number of numerical iterations of the trial and errors correction cycle upon the examples (patterns, always in the ANN jargon). Generally a second set of examples, the verification set, not used in the learning phase, it is instead employed to check the performances of the ANN function.

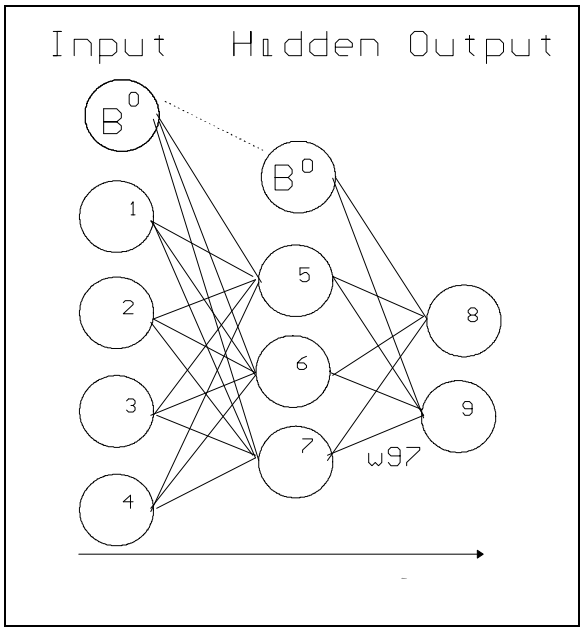

Figure 1.3 A simple Artificial Neural Network

The particular way neural networks work and are trained, makes them able to reproduce the performance of any function as "universal approximator" (White, 1990).

5. THE CROSS–TARGET (CT) METHOD

5.1 SKETCHING THE IDEA

To develop our agent based experiments, we introduce the following general hypothesis (GH): an agent, acting in an economic environment, must develop and adapt her capability of evaluating, in a coherent way, (1) what she has to do in order to obtain a specific result and (2) how to foresee the consequences of her actions. The same is true if the agent is interacting with other agents. Beyond this kind of internal consistency (IC), agents can develop other characteristics, for example the capability of adopting actions (following external proposals, EPs) or evaluations of effects (following external objectives, EOs) suggested from the environment (for example, following rules) or from other agents (for examples, imitating them). Those additional characteristics are useful for a better tuning of the agents in making experiments.

To apply the GH we are employing here artificial neural networks; we observe, anyway, that the GH can be applied using other algorithms and tools, reproducing the experience–learning–consistency–behavior cycle with or without neural networks.

An introductory general remark: in all the cases to which we have applied our GH, the preliminary choice of classifying agents' output in actions and effects has been useful (i) to clarify the role of the agents, (ii) to develop model plausibility and results, (iii) to avoid the necessity of prior statements about economic rational optimizing behavior (Beltratti *et al.*, 1996).

Economic behavior, simple or complex, can appear directly as a by–product of IC, EPs and EOs. To an external observer, our Artificial Adaptive Agents (AAAs) are apparently operating with goals and plans. Obviously, they have no such symbolic entities, which are inventions of the observer. The similarity that we recall here is that the observations and analyses about real world agents' behavior can suffer from the same bias. Moreover, always to an external observer, AAAs can appear to apply the rationality paradigm, with maximizing behavior.

Complexity can be more frequently found out of the agents – in the framework emerging from their interaction, adaptation and learning – than within them; exactly in the same way, also rationality (and Olympic rationality) can be found out of agents, simply as a by–product of environment constraints and agents bounded capabilities. The same for optimization, as a by–product of interaction and constraints, emerging externally to the agents mind.

The main problem is: obviously agents, with their action, have the goal of increasing or decreasing something, but it is not correct to deduce from this statement any formal apparatus encouraging the search for complexity within agents, not even in the *as if* perspective. With our GH, and hereafter with the Cross Target (CT) method, we work at the edge of Alife techniques to develop Artificial Worlds of simple bounded rationality AAAs: from their interaction, complexity, optimizing behavior and Olympic rationality can emerge, but externally to the agents.

Finally, we want to consider learning from the point of view of the bounded rationality research program (Arthur, 1990):

> In designing a learning system to represent human behaviour in a particular context, we would be interested not only in reproducing human rates of learning, but also in reproducing the style in which humans learn, possibly even the ways in which they might depart from perfect rationality. The ideal, then, would not simply be learning curves that reproduce human learning curves to high goodness–of–fit, but more ambitiously, learning behaviour that could pass the Turing test of being

> indistinguishable from human behaviour with its foibles, departures and errors, to an observer who was not informed whether the behaviour was algorithm–generated or human–generated.

In order to implement this ideal target without falling in the trap of creating models that are too complicated to be managed, we consider artificially intelligent agents founded upon algorithms which can be modified by a trial and error process. In one sense our agents are even simpler than those considered in neoclassical models, as their targets and instruments are not as powerful as those assumed in those models. From another point of view, however, our agents are much more complex, due to their continuous effort to learn the main features of the environment with the available instruments.

The name cross–targets (CTs) comes from the technique used to figure out the targets necessary to train the ANNs representing the artificial adaptive agents (AAAs) that populate our experiments.

Following the GH, the main characteristic of these AAAs is that of developing internal consistency between what to do and the related consequences. Always according to the GH, in many (economic) situations, the behavior of agents produces evaluations that can be split in two parts: data quantifying actions (what to do) and forecasts of the outcomes of the actions. So we specify two types of outputs of the ANN and, identically, of the AAA: (i) actions to be performed and (ii) guesses about the effects of those actions.

Both the targets necessary to train the network from the point of view of the actions and those connected with the effects are built in a crossed way, originating the name Cross Targets. The former are built in a consistent way with the outputs of the network concerning the guesses of the effects, in order to develop the capability to decide actions close to the expected results. The latter, similarly, are built in a constant way with the outputs of the network concerning the guesses of the actions, in order to improve the agent's capability of estimating the effects emerging from the actions that the agent herself is deciding.

CTs, as a fulfillment of the GH, can reproduce economic subjects' behavior, often in internally ingenuous ways, but externally with complex results.

The method of CTs, introduced to develop economic subjects' autonomous behavior, can also be interpreted as a general algorithm useful for building behavioral models without using constrained or unconstrained optimization techniques. The kernel of the method, conveniently based upon ANNs (but it could also be conceivable with the aid of other mathematical tools), is learning by guessing and doing:

the subject control capabilities can be developed without defining either goals or maximizing objectives.

5.2 TECHNICAL DETAILS

We choose the neural networks approach to develop CTs, mostly as a consequence of the intrinsic adaptive capabilities of neural functions. Here we will use feed forward multilayer networks as introduced in Section 4.

Figure 1.4 describes an AAA learning and behaving in a CT scheme. The AAA has to produce guesses about its own actions and related effects, on the basis of an information set (the input elements are $I_1, ..., I_k$). Remembering the requirement of IC, targets in learning process are: (i) on one side, the actual effects – measured through accounting rules – of the actions made by the simulated subject; (ii) on the other side, the actions needed to match guessed effects. In the last case we have to use inverse rules, even though some problems arise when the inverse is indeterminate. Technical explanations of CT method (Beltratti *et al.*, 1996)are reported below.

A first remark, about learning and CT: analyzing the changes of the weights during the process we can show that the matrix of weights linking input elements to hidden ones has little or no changes, while the matrix of weights from hidden to output layer changes in a relevant way. Only hidden–output weight changes determine the continuous adaptation of ANN responses to the environment modifications, as the output values of hidden layer elements stay almost constant. This situation is the consequence both of very small changes in targets (generated by CT method) and of a reduced number of learning cycles.

The resulting network is certainly under trained: consequently, the simulated economic agent develops a local ability to make decisions, but only by adaptations of outputs to the last targets, regardless to input values. This is short term learning as opposed to long term learning.

Some definitions: we have (i) short term learning, in the acting phase, when agents continuously modify their weights (mainly from the hidden layer to the output one), to adapt to the targets self–generated via CT; (ii) long term learning, ex post, when we effectively map inputs to targets (the same generated in the acting phase) with a large number of learning cycles, producing ANNs able to definitively apply the rules implicitly developed in the acting and learning phase.

A second remark, about both external objectives (EOs) and external proposals (EPs): if used, these values substitute the cross targets in the acting and adapting phase and are consistently included in the data set

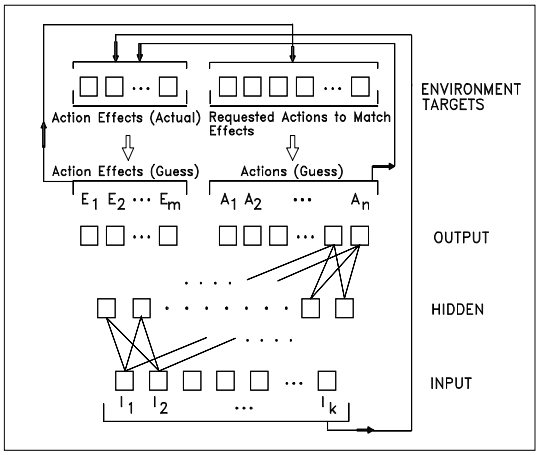

Figure 1.4 The Cross-Target (CT) scheme.

for ex post learning. Despite the target coming from actions, the guess of an effect can be trained to approximate a value suggested by a simple rule, for example increasing wealth. This is an EO in CT terminology. Its indirect effect, via CT, will modify actions, making them more consistent with the (modified) guesses of effects. Vice versa, the guess about an action to be accomplished can be modified via an EP, affecting indirectly also the corresponding guesses of effects. If EO, EP and IC conflict in determining behavior, complexity may emerge also within agents, but in a bounded rationality perspective, always without the optimization and full rationality apparatus.

Now we may introduce some technical explanations about CTs, with the aid of the general scheme of Figure 1.4, observing that (i) the inputs of the model are mainly data coming from the environment or from other agents' behavior, (ii) they can be dependent or independent from the previous actions of the simulated artificial subject, (iii) targets are known only when actions take place.

The CT algorithm is a learning and acting one: action is necessary to produce the information by which we can construct targets to train the ANN that simulates the subject. A training set cannot be constructed here in the usual way because rules linking inputs and outputs of the ANN have 'to be discovered' by the experiments led by AAAs.

Learning and acting take place in four steps each 'day'; a day is the sum of the four steps required to perform a full cycle of estimation of

outputs and of backpropagation of errors, correcting the neural network weights. Initial weights are randomized in a given range.

Looking at Figure 1.4, the four steps can be introduced in the following sequence.

1. Outputs of the ANN: the actions to be accomplished, reported in the right side of Figure 1.4, and the effects of these actions, reported in the left side of the same figure, are guessed following inputs and network weights.

2. Targets for the left side of the network: the targets for the effects supposed to arise from actions, as guessed in the left side of the output layer in Figure 1.4, are figured out by the independently guessed actions. In this way, guesses about effects become more close to the true consequences of actual actions.

3. Targets for the right side of the network: the differences measured in step (2) among targets and ANN outputs on the effect side can be inversely interpreted as starting points for action modifications, to match the guessed effects. So they are used to build the targets for the mechanism that guesses the actions. Being the inverses of the formulas shown below often undefined, corrections are shared randomly among all the targets to be constructed; besides, when several corrections concern a target, only the one with the largest module is chosen. In this way, we would like to imitate the actual behavior of a subject requested of obeying to several independent and inconsistent commands: probably the most imperative, here the largest value, will be followed.

4. Backpropagation: learning takes place, correcting weights in order to obtain guessed effects closer to the consequences of guessed actions, and guessed actions more consistent with guessed effects. Thus, we have two learning processes, both based upon the guesses of the elements of the opposite side of the network.

This double sided process of adaptation, with interaction among agents and long term learning introduced above, ensures the emergence of non trivial self–developed behavior, from the point of view of time paths of the values generated by the outcomes of the agents.

We can now explain in a formal way the acting and learning algorithm of CTs, introducing a generic effect E_1 arising from two actions, named A_1 and A_2. The target for the effect is:

$$\widehat{E}_1 = f(A_1, A_2) \tag{1.4}$$

where $f(\bullet)$ is a definition, linking actions to effects on an accounting basis.

Our aim here is to obtain an output E_1 (the guess made by the network) closer to $\widehat{E}_1$, which is the correct measure of the effect of actions A_1 and A_2. The error related to E_1 is:

$$e = \widehat{E}_1 - E_1$$

or, by convention in ANN development, one half of the square of $\widehat{E}_1 - E_1$. To minimize the error, we backpropagate it through network weights.

Our aim is now finding the actions, as outputs of our network, more consistent with the outputs produced by the effect side. So we have to correct A_1 and A_2 to made them closer to $\widehat{A}_1$ and $\widehat{A}_2$, which are actions consistent with the output E_1. We cannot figure out the targets for A_1 and A_2 separately. From (1.4) we have:

$$A_1 = g_1(\widehat{E}_1, A_2) \tag{1.5}$$

$$A_2 = g_2(\widehat{E}_1, A_1) \tag{1.6}$$

Choosing a random value τ_1 from a random uniform distribution whose support is the closed interval $[0, 1]$ and setting $\tau_2 = 1 - \tau_1$, from (1.5) and (1.6) we obtain:

$$\widehat{A}_1 = g_1(\widehat{E}_1 - e \cdot \tau_1, A_2) \tag{1.7}$$

$$\widehat{A}_2 = g_2(\widehat{E}_1 - e \cdot \tau_2, A_1) \tag{1.8}$$

Functions g_1 and g_2, being obtained from definitions that link actions to effects mainly on an accounting basis, usually have linear specifications; so equations (1.7) and (1.8) generally give solutions that are globally consistent. The errors to be minimized are:

$$a_1 = \widehat{A}_1 - A_1$$

$$a_2 = \widehat{A}_2 - A_2$$

Eqs. (1.7) and (1.8) would be unacceptable as inversions of true dynamic functions, but they are used here as a simplifying tool (mainly for the presence of random separation obtained by t_1 and t_2 values), always to generate time paths for variables, without a priori or external suggestions.

When the actions determine multiple effects, they are included in multiple definitions of effects. So, those actions will be affected by several corrections; as reported in point 3 above, only the largest absolute value is chosen.

Input and target variability, generated both in deterministic and random ways, is required to ensure the economic plausibility of the experiments, but is also necessary to ensure that the outputs and the targets of the ANN change. Lacking such variability, on the basis of initial random weights of the network and following CTs, in most cases all outputs would be frozen at about 0.5, with perfect but merely apparent learning results.

With the proper variability, we repeat for a given number of cycles (days) the four steps introduced describing Figure 1.4. The learning following the fourth step of each day gives a sort of local adaptation to the changes of the environment.

6. THE SWARM BP–CT PACKAGE

We introduce here the bp–ct code as a package useful to develop and run ANNs in the Swarm framework. A complete "How to use bp–ct" explanation is included as a text file in the distribution of the package. Here we only outline the use of the program both with simple examples and with an economic application (in Section 7.).

At present the code runs with Swarm v.1.4.1 and it will be continuously adapted to the new releases of Swarm.

6.1 BP–CT AS BP, WITH EXTERNAL INPUTS AND TARGETS

The first example is related to the use of the program as a simple backpropagation tool. In the window reported in Figure 1.5 we point out: (i) the use of a unique agent–ANN (but the code can train and run in a parallel way any number of ANNs, with only memory and time limitations); (ii) the structure of the ANN (inputNodeNumber, hiddenNodeNumber, outputNodeNumber); (iii) the number of patterns in the Training Set and in the Verification Set; (iv) the Number of Epochs (an Epoch is here a complete learning cycle upon all the patterns, correcting the weights after the evaluation of the ANN outputs for each pattern).

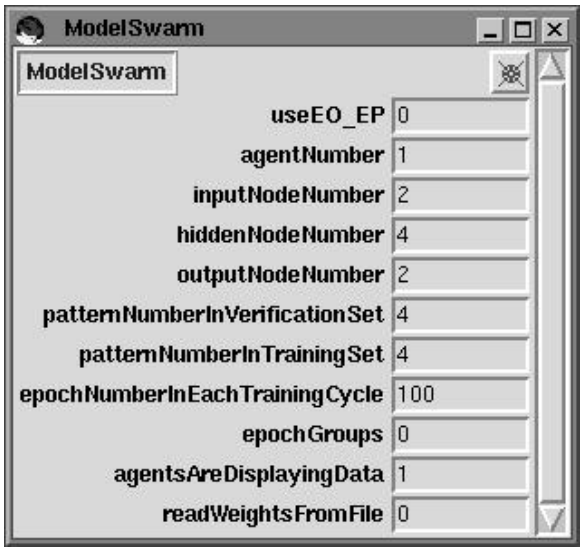

Figure 1.5 The control window of bp–ct in the external inputs and targets case

The patterns of this example represent a classic in the field of ANNs testing. It is based on the XOR logical function with $XOR(F,F) = F$; $XOR(T,F) = T$; $XOR(F,T) = T$; $XOR(T,T) = F$. In this case we can have only four patterns for the Training Set; the same patterns form the Verification Set (numerically, we will use 0 for F and 1 for T). We have here two outputs because our ANN simultaneously maps also the OR function on the input data.

Being positive the number of patterns pointed out for the Training and the Verification Set in the window of Figure 1.5, the program read the pattern data from two files (the name of the files is specified in the code and can be changed). After about 30 learning Epochs the XOR output of our ANN, as reported in Figure 1.6, matches perfectly the desired outputs (targets) of Figure 1.7.

6.2 BP–CT AS CT, WITH INTERNAL GENERATION OF INPUTS AND TARGETS

The second example is related to the use of the program with internal generation of inputs and targets, as a step toward the CT use. In the window reported in Figure 1.8 we point out the same choices of the previous example, except the point (iii) where we have the number of patterns in the Training Set equal to -1 and those in the Verification

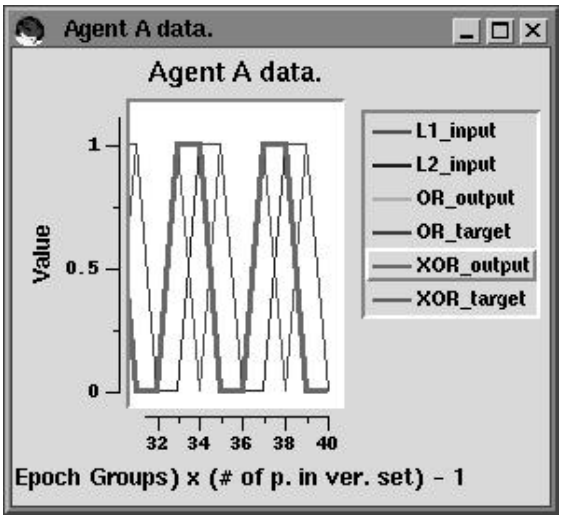

Figure 1.6 XOR outputs

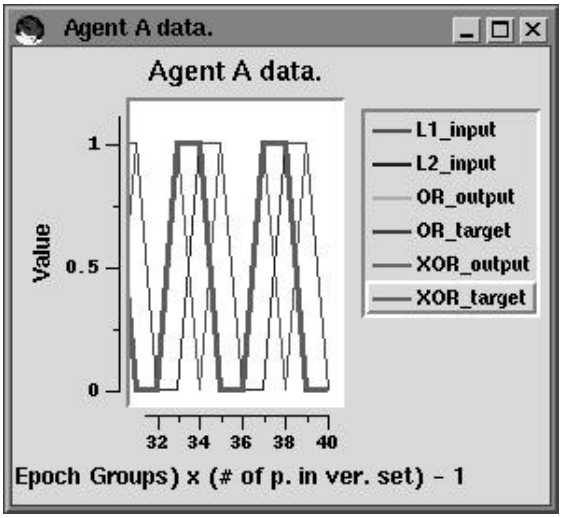

Figure 1.7 XOR targets

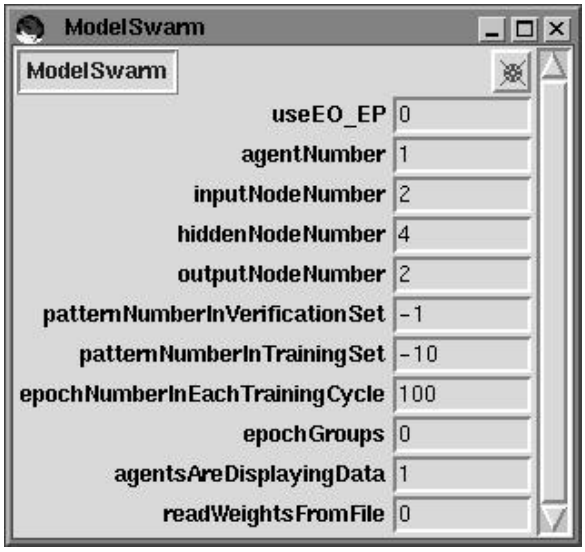

Figure 1.8 The control window of bp–ct in the internal inputs and targets case

Set equal to -10. A negative value has here the meaning of internal data generation: if negative, the patternNumberInTrainingSet variable can be only equal to -1; the negative value of the patternNumberInVerificationSet variable establishes the number of auto–generated patterns to be collected to form each training Epoch.

Also this example is founded on the XOR logical function: after about 30 learning Epochs the XOR output of our ANN, as reported in Figure 1.9, matches perfectly the desired outputs (targets) of Figure 1.10.

7. AN EXPERIMENT WITH THE CT SCHEME OF ACTIONS AND EFFECTS, TO OBTAIN A SPONTANEOUS HAYEKIAN MARKET

The main goal of the bp–ct project is that of using the code in the perspective of interacting agents, with internally generated inputs and targets, following the ERA scheme (Section 3.) and, above all, applying the rule structure described in the CT framework (Section 5.).

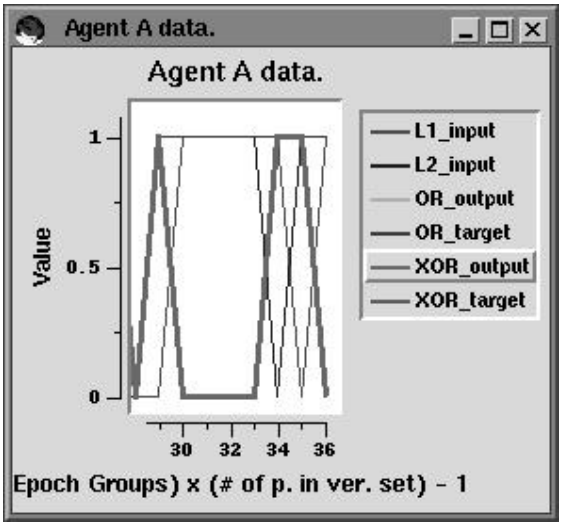

Figure 1.9 XOR outputs

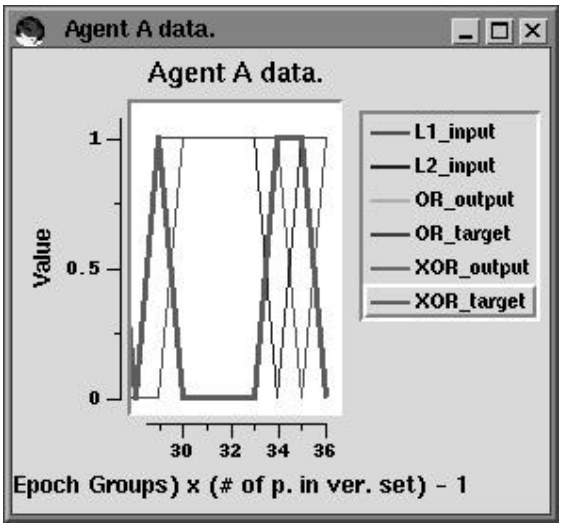

Figure 1.10 XOR targets

7.1 THE STRUCTURE OF THE EXPERIMENT

With CT we build an experimental framework with AAAs to test the consequences of the learning capabilities of the agents.

The experiment has the following structure (a detailed definition of the structure is reported in the code ct–hayek, which is the adaptation of bp–ct to the specific experiment).

1. The consumer agent has in input the variables: $Expenditure_0$, the expense of the previous period; $Requirement_0$, the requirement of the previous period; p_0, the price of the unique good, as proposed by the agent in the previous period; q_0, the agent buying proposal in the previous period.

2. The producer agent has in input the variables: $Revenue_0$, the revenue of the previous period; $ProductionStream_0$, the production stream requirement of the previous period; p_0, the price of the unique good, as proposed by the agent in the previous period; q_0, the agent selling proposal in the previous period.

3. The outputs of the ANN simulating the consumer agent and the related targets are: $actualP$, the guess about the exchange price (the target is simply the exchange price, true or latent, see below); $actualQ$, the guess about the exchanged quantity (the target is simply the exchanged quantity, may be 0); the $actualP$ and $actualQ$ values are effects in CT jargon, but they strictly depend by the action of another agent; being here not strategical to develop the agent capability of forecasting these values, they are not directly included in the cross targets to action; $Expenditure$, the guess about the expense in the current period (the target is obtained multiplying p and q); $Requirement$, the requirement in the current period (the target is $actualQ$); p and q, the price and the quantity proposed by the agent to exchange (the targets come from $Expenditure$ and $Requirement$ and so indirectly from $actualQ$).

4. The outputs of the ANN simulating the producer agent and the related targets are: $actualP$ and $actualQ$ (see above); $Revenue$, the guess about the revenue in the current period (the target is obtained multiplying p and q); p and q, the price and the quantity proposed by the agent to exchange (the targets come from $Revenue$ and $ProductionStream$ and so indirectly from $actualQ$).

5. We have 10 consumers and 10 producers; the production process is undefined; the producers are simply supposed to offer the quantity

that they are producing in each time unit; consumers and producers meet randomly each "day". The exchange is possible if the producer proposed price is less or equal to the price proposed by the consumer; the price used in the exchange is that proposed by the producer and the quantity is the minimum between the consumer proposal and the producer proposal; if the exchange is not possible the reference price (used in the determination of the mean price of the day) is that of the consumer and the quantity is 0.

6. We introduce also external objectives (EOs) which are explained in detail in the distributed ct–hayek code. In the consumer case: if $useEO\_EP$ variable (see Figure 1.11) is set to 0 we have no EOs; if $useEO\_EP$ variable is set to 1, the effect Expenditure is trained with a lowering target; if $useEO\_EP$ variable is set to 2, the effect Requirement is trained with a constant target; if $useEO\_EP$ variable is set to 3 both the previous EOs operate. The CT consequences are modification of the p guess, remembering that the EOs modifies the outputs of the effect side of the underlying ANN, but via CT also the action side; the modification of the guess about p facilitates or prevents the exchange measured by actualQ. In the producer case we act symmetrically, when $useEO\_EP$ is set to 1 or 2 or 3, increasing the Revenue target and keeping constant the ProductionStream one; ProductionStream can be interpreted as the adequate production related to an existing fixed capital structure.

Following the distinction between short and long term learning introduced in Section 5., the experiment is run (i) with short term learning (Figures 1.12, 1.15, 1.18 and 1.21), with agent adapting only to local the local situation, (ii) with a light form of long term learning (Figures 24, 1.16, 1.19 and 1.22), where agent start developing the capability of acting globally and (iii) finally (Figures 1.14, 1.17, 1.20 and 1.23) with a complete form of long term learning.

The starting window of the ct–hayek code, with the no EOs and light long term learning case, is reported in Figure 1.11, where we can see the choice about EOs, the double choice of the agent number, the CT selection with the negative numbers in the two pattern number boxes.

In the Hayek sense of individualism and economic order (Hayek, 1949), we are here dealing with agents behaving on a strictly individualistic basis, without the over–simplification of an artificial auctioneer market. A market with stable prices anyway emerges, with an empirical quasi–equilibrium. Note that we explicitly exclude from our model any form of prior description of this kind of equilibrium and that the agents have

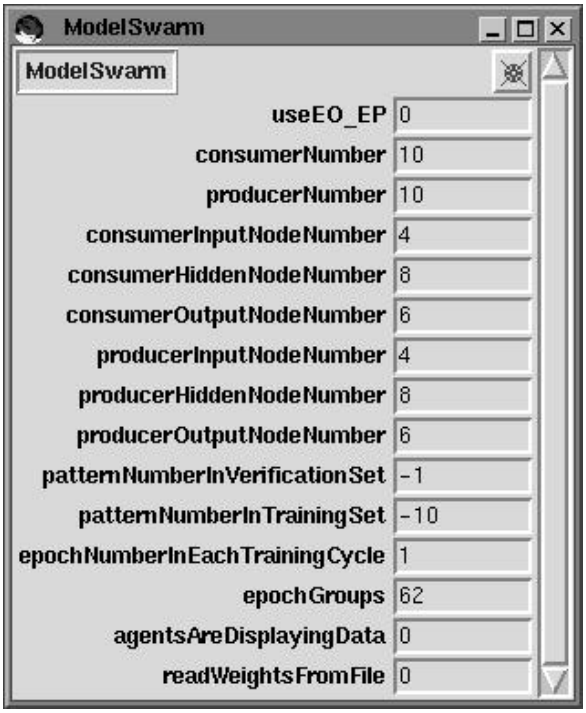

Figure 1.11 The control window of bp–ct as it appears in ct–hayek package

no knowledge about the mean price, which is only used by the observer to externally describe the experiment.

We have to read the lines of the graphics of the Figures from 1.12 to 1.23, as (bottom up): (i) mean quantity of the exchanges (divided by 10 for a scale necessity); (ii) Min price; (iii) mean price; (iv) Max price. Prices are obtained by the p variable described above.

7.2 EXPERIMENT RESULTS

A market spontaneous equilibrium emerges immediately in our first run of the model, with $useEO\_EP$ variable set to 0. We have a complex equilibrium situation: in the case of short term leaning, Figure 1.12, with a low price increase as a consequence of the internally generated (without EOs) values of $Requirement$ and $ProductionStream$; in the case of light relearning, Figure 1.13, the complexity of the system increases, showing a few agents (the mean is always very close to the Max) leaving the general situation; the third case, the one with heavy relearning, reported

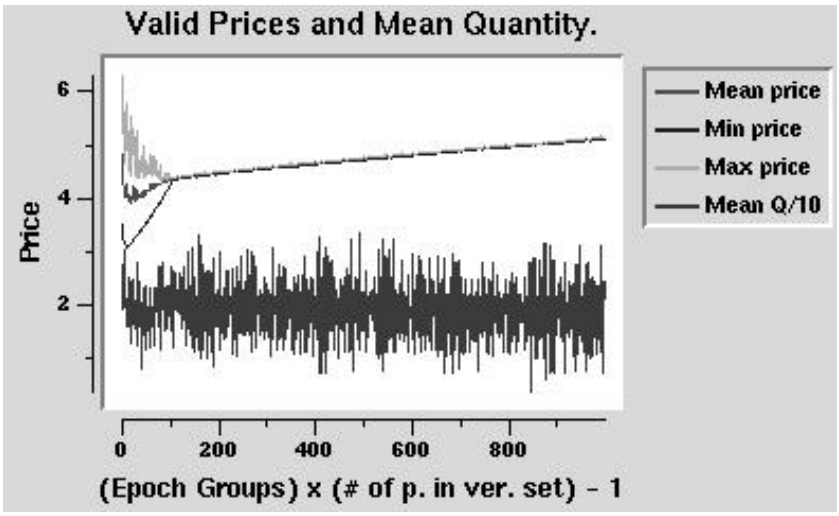

Figure 1.12 Short term learning, without EOs

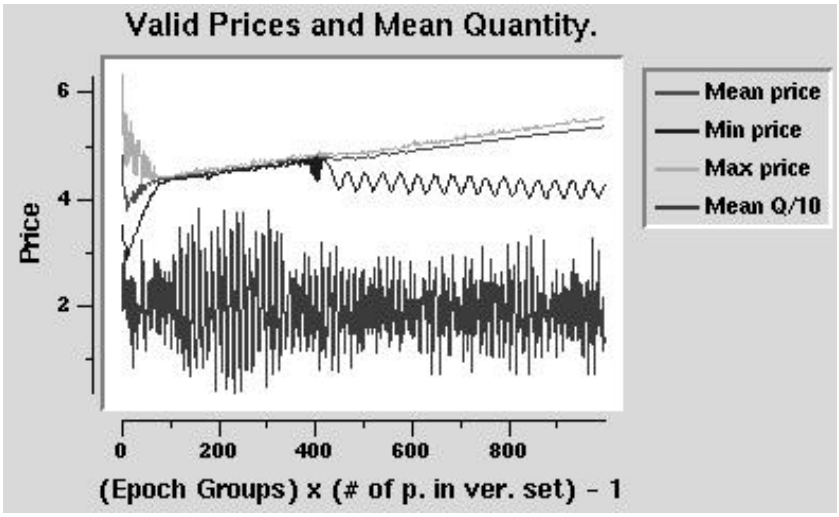

Figure 1.13 Light long term learning, without EOs

in Figure 1.14, we can see a robust equilibrium, with some individuals differentiating their behavior.

In the second run of the model, with $useEO\_EP$ variable set to 1, we have a perfectly calibrated model in the case of short term learning of Figure 1.15. The presence of long term learning introduces, both in Fig-

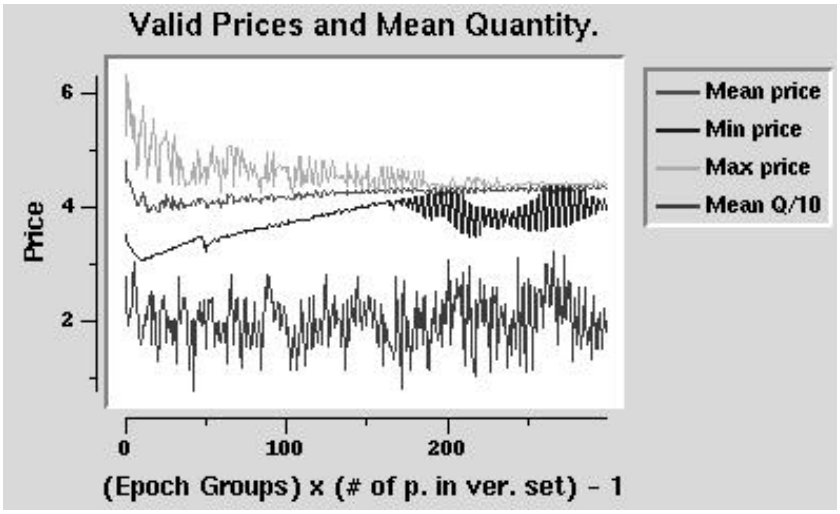

Figure 1.14 Long term learning, without EOs

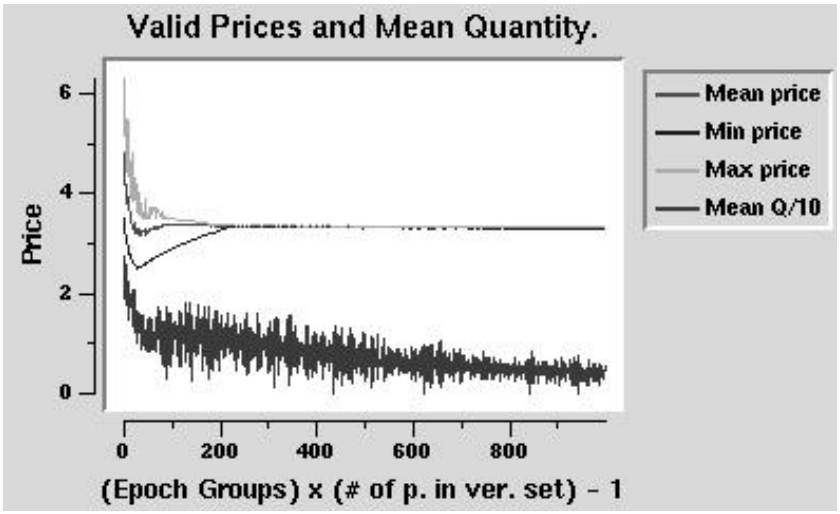

Figure 1.15 Short term learning, with EOs on Expenditure and Revenue

ure 1.16 and in Figure 1.17 a few cases of divergent behavior. Effectively the goal proposed, via EOs, to the agents have a poor economic sense if considered alone, being the suggestion of doing impossible things (to diminish expenditure and to raise revenue, without boundary conditions); the inconsistency of this situation emerges with the relearning process.

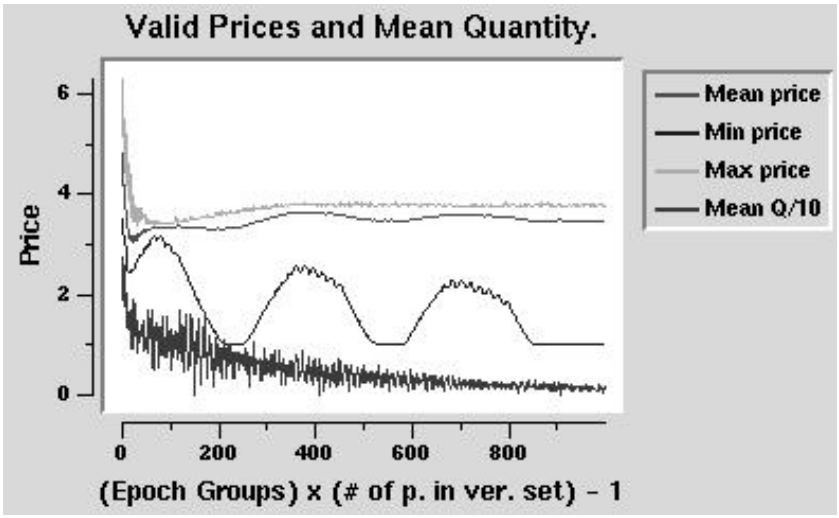

Figure 1.16 Light long term learning, with EOs on Expenditure and Revenue

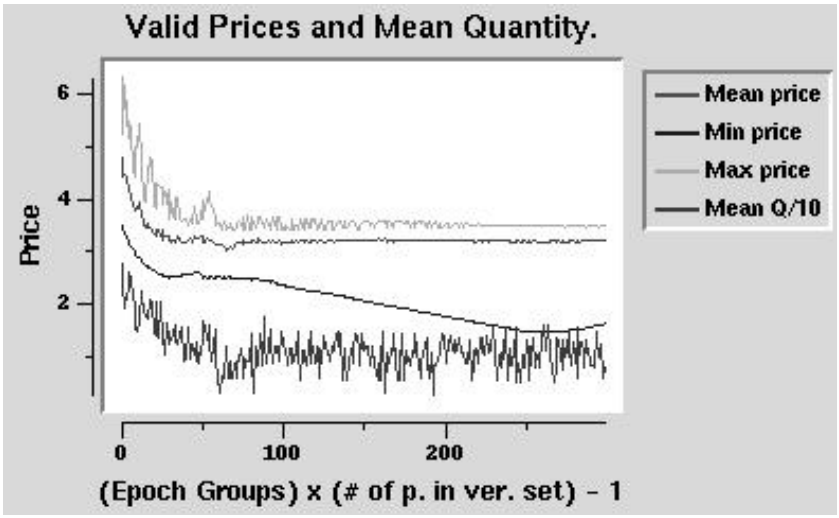

Figure 1.17 Long term learning, with EOs on Expenditure and Revenue

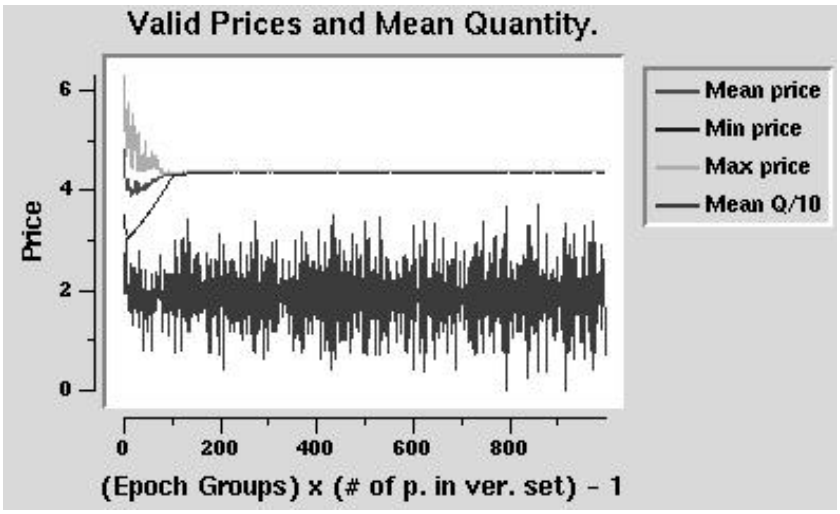

Figure 1.18 Short term learning, with EOs on Requirements and ProductionStream

The third run of the model, with $useEO\_EP$ variable set to 2, shows a stabilized situation in prices (not in quantities). The consequence of keeping requirements and production streams constant is represented by the equilibrium situation of Figure 1.18, 1.19 and 1.20, where prices stay near constant, but the quantities are heavily oscillating.

Finally, with $useEO\_EP$ variable set to 3, i.e. operating simultaneously to diminish expenditures, raise revenues, stabilize requirements and production streams, we have stabilized the market, with low oscillations in quantities, mainly when the agents are able to operate globally, after long term learning. This is true also without long term learning, in Figure 1.21, but becomes more evident with learning in Figure 1.22 and Figure 1.23. In Figure 1.22 we have also an interesting effect of stabilization of the market when learning takes effectively place, after an initial period of price diminution.

8. FUTURE IMPROVEMENTS

We report here some improvements to be included in the future versions of bp–ct.

We will introduce a probe to each agent, to allow the direct inspections of agent variables while the model is running. An important improvement will be that of the automation of the production of the CT formulas mainly on the basis of definitions of actions and effects (at present the formulas containing the CT rules are handwritten in the Interface code).

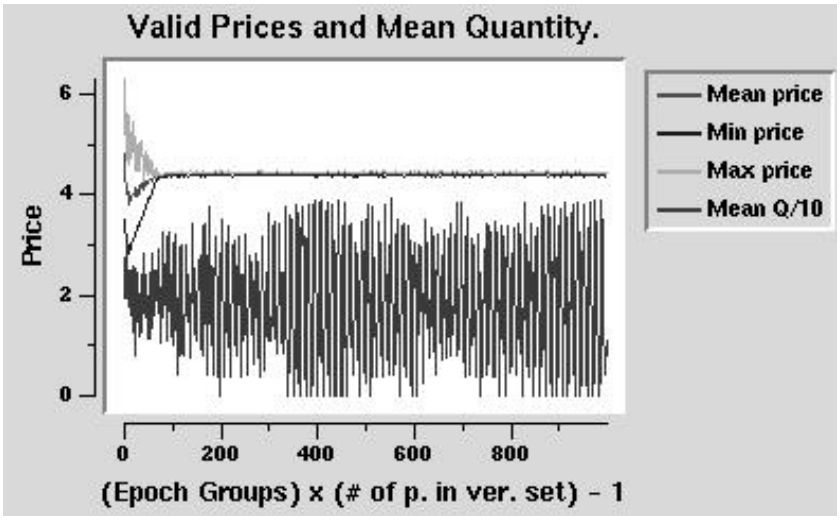

Figure 1.19 Light long term learning, with EOs on Requirements and ProductionStream

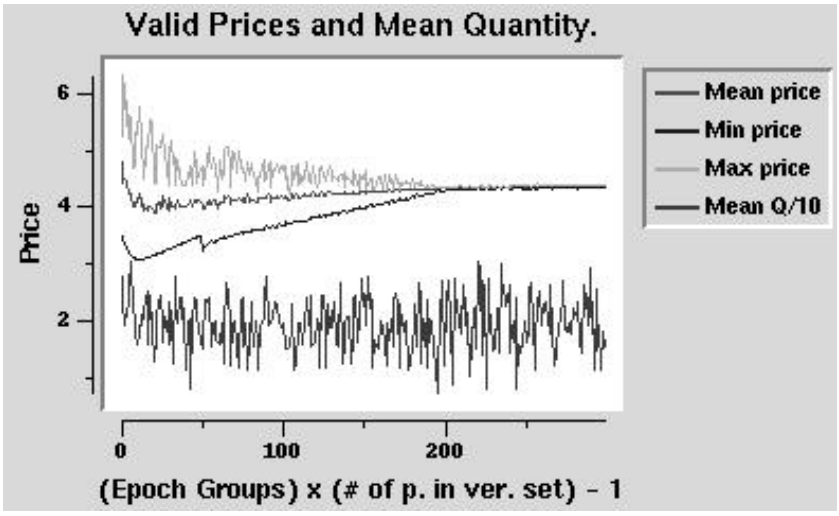

Figure 1.20 Long term learning, with EOs on Requirements and ProductionStream

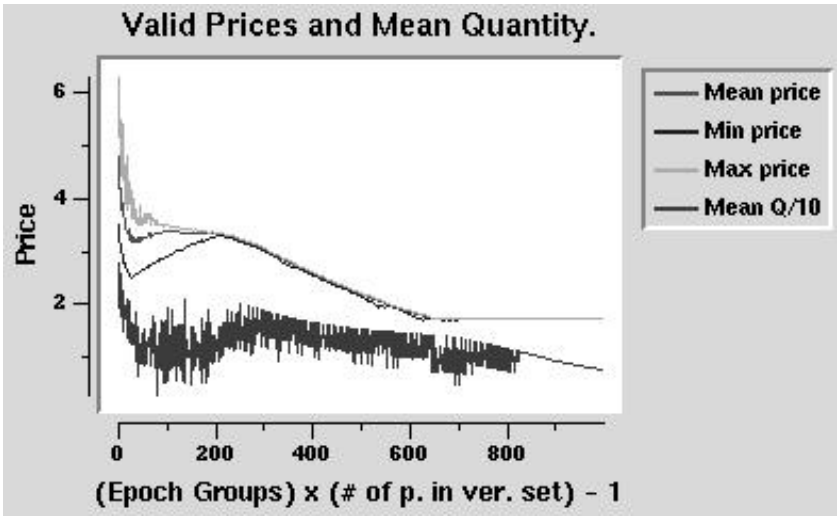

Figure 1.21 Short term learning, with EOs on Expenditure, Revenue, Requirements and ProductionStream

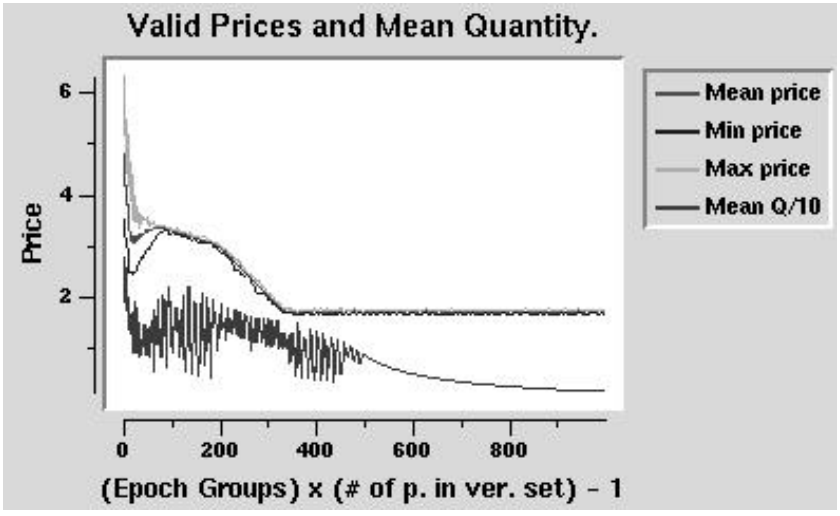

Figure 1.22 Light long term learning, with EOs on Expenditure, Revenue, Requirements and ProductionStream

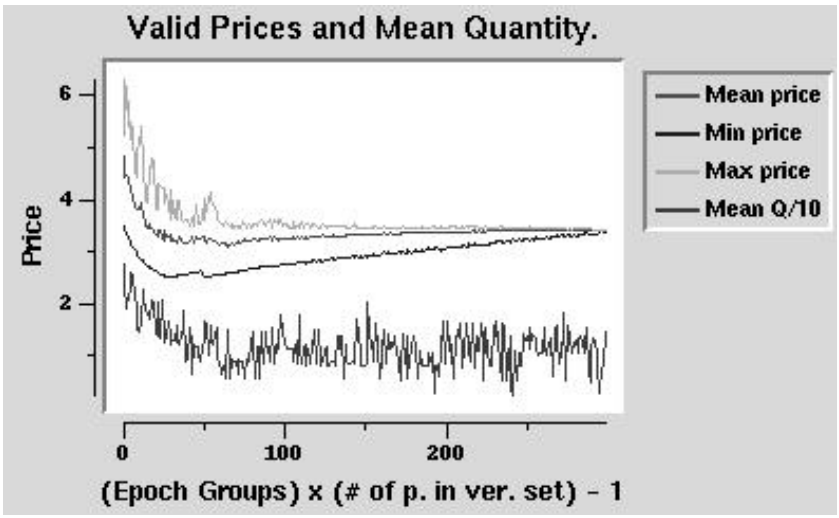

Figure 1.23 Long term learning, with EOs on Expenditure, Revenue, Requirements and ProductionStream

Finally we will operate to allow the automatic or quasi–automatic substitution of neural networks with classifier systems or with more general genetic algorithms.

We think that it would be useful to develop the GH of Section 5. also in other ways, to verify the reproducibility of our results in other contexts. We have to introduce algorithms capable of the same performances in order to reproduce short and long term learning, without the aid of ANN. Our algorithm must be able to modify its outputs in a smooth way, following cross–target suggestions about actions and guesses of action effects, to self–develop the behavioral skills of the acting and adapting phase. The algorithm – may be not the same – has also to develop a strong mapping ability between input and output (target) vectors, to produce the same behavioral results without further learning.

Acknowledgments

This research has been supported by grants from the MURST project "Intermediazione finanziaria, funzionamento dei mercati ed economia reale."

Notes

1. `http://www.econ.iastate.edu/tesfatsi/sylalife.htm`
2. `http://www.econ.iastate.edu/tesfatsi/ace.htm`

3. http://www.econ.iastate.edu/tesfatsi/getalife.htm
4. http://www.cea.uba.ar/aschu/complex.html
5. http://zia.hss.cmu.edu/econ/
6. http://hmt.com/cwr/ibm.html
7. http://www.lse.ac.uk/lse/complex/

References

Arthur, W.B. (1990). "A Learning Algorithm that Mimics Human Learning." *Santa Fe Institute Working Paper 90-026.* .

Beltratti, A., Margarita, S., Terna, P. (1996). *Neural Networks for Economic and Financial Modelling.* London: International Thomson Computer Press.

Conte, R., Hegselmann, R., Terna, P. (eds.) (1997). *Simulating Social Phenomena.* Berlin: Springer.

Epstein, M.E. and Axtell, R. (1996). *Growing Artificial Societies - Social Science from the Bottom Up.* Washington: Brookings Institution Press. Cambridge, MA: MIT Press.

Gilbert, N. and Terna, P. (1999)."'How to build and use agent-based models in social science." *Mind & Society*, 1, forthcoming.

von Hayek, F.A. (1949). *Individualism and Economic Order.* London: Routledge.

Judd, K.L. (1996). "Computational Economics and Economic Theory: Substitutes or Complements?" ¡http://bucky.stanford.edu/¿.

Kirman, A. (1992). "Whom or What Does the Representative Agent Represent?" *Journal of Economic Perspectives*, 6, pp.126-39.

Rumelhart, D.E. and McClelland, J.L. (1986). *Parallel Distributed Processing. Explorations in the Microstructure of Cognition.* Cambridge, MA: The MIT Press.

Russel, S. and Norvig, P. (1986). *Artificial Intelligence. A Modern Approach.* Upper Saddle River, NJ: Prentice-Hall.

White, H. (1990). "Connectionist Nonparametric Regression: Multilayer Feedforward Networks Can Learn Arbitrary Mappings." *Neural Networks*, 5, pp. 535-550.