INDICE

Titolo: SIMULAZIONE DI UN MERCATO DI BORSA CON UN MODELLO AD AGENTI

Capitolo 1: Simulazione e modelli basati su agenti

Introduzione

L'uso della simulazione con il computer per l'analisi e lo studio dei fenomeni sociali rappresenta un modo nuovo di fare ricerca scientifica che si aggiunge alle due metodologie tradizionali dell'argomentazione verbale e della formalizzazione matematica e che ne supera, in parte, le limitazioni ed i vincoli.

Quando le rappresentazioni dei processi sociali vengono espresse in termini puramente verbali, per il ricercatore risulta difficile determinare precisamente le implicazioni delle idee che stanno alla base del modello e verificare le ipotesi concernenti i legami tra le diverse grandezze.

In alcuni campi della ricerca sociale, come l'economia, il modello del fenomeno oggetto di studio viene generalmente espresso in termini di equazioni matematico-statistiche che spesso, però, risultano troppo complicate per essere trattate analiticamente e rendono quindi necessario, al fine di ottenere equazioni risolvibili, il ricorso a semplificazioni (talvolta poco plausibili) che in alcuni casi finiscono per snaturare la teoria sottostante.

Per superare, almeno in parte, tali ostacoli è possibile ricorrere alla simulazione.

Con la simulazione, il modello da analizzare viene rappresentato come un programma al computer che può essere utilizzato per descrivere teorie sia qualitative che quantitative e che è in grado di affrontare e rappresentare relazioni anche non lineari tra le variabili del modello.

Come affermato in Parisi (2001) una simulazione è un altro modo di esprimere una teoria scientifica; i concetti, i meccanismi, i processi postulati dalla teoria non vengono descritti da parole o simboli matematici, ma vengono incorporati in un programma, che riproduce i fenomeni che la teoria intende spiegare.

Il metodo della simulazione differisce radicalmente dai metodi tradizionali della ricerca scientifica perché non mira solamente a descrivere, prevedere o spiegare i fenomeni della realtà, ma cerca di ricrearli; tale metodologia formula le proprie teorie e ipotesi sotto forma di progetti per la costruzione di sistemi artificiali; costruisce tali sistemi e verifica la loro capacità di descrivere la realtà oggetto di studio.

Le simulazioni, in altre parole, aiutano a comprendere la realtà ricreandola all'interno del computer[1].

In ingegneria e nelle scienze naturali, la simulazione è stata ampiamente adottata come un'utile metodologia di studio; nelle maggior parte delle scienze sociali, invece, rappresenta un strumento di analisi relativamente nuovo.

Recentemente, tuttavia, diversi fattori hanno spinto gli scienziati sociali a prestare una maggiore attenzione verso questo nuovo campo di ricerca. In particolare, come affermato in Conte ed altri (1997):

[1] Una simulazione può essere definita una teoria 'attiva', in quanto 'girando' nel computer produce direttamente i fenomeni simulati, le predizioni empiriche (le quali devono corrispondere, per dare validità alla simulazione, ai fatti osservati nella realtà).

- le difficoltà incontrate dall'analisi tradizionale e dai metodi della ricerca empirica nello studio della dinamica sociale e nella verifica delle corrispondenti teorie e modelli;
- l'attenzione alle questioni riguardanti le strutture sociali e l'emergere di modelli comportamentali;
- il crescente interesse per i fenomeni sociali di autorganizzazione;

hanno costituito importanti stimoli per l'applicazione della simulazione anche al vasto campo delle discipline sociali.

1.1 I benefici della simulazione

L'introduzione del metodo simulativo può avere conseguenze rivoluzionarie per le scienze sociali; tale metodologia consente, infatti, di rimuovere alcune delle debolezze strutturali che hanno ostacolato e rallentato lo sviluppo delle scienze dell'uomo - il carattere soggettivo dei fenomeni oggetto di studio, la mancanza di un legame stretto tra teorie e osservazioni empiriche, l'impossibilità di ricorrere ad esperimenti di laboratorio - e che hanno reso lo studio dei fenomeni umani un compito più difficile rispetto allo studio dei fenomeni naturali.

A differenza dei fenomeni naturali molti importanti processi sociali sono complessi e non sono nettamente scomponibili in separati sotto-processi – economici, culturali, demografici, ecc - la cui analisi separata possa, poi, essere aggregata per dare una adeguata spiegazione del processo sociale nel suo complesso[2].

Nelle discipline sociali, inoltre, risulta difficile ricorrere a forme di controllo e di verifica dei risultati ottenuti. In particolare, risulta difficile verificare le ipotesi concernenti le relazioni tra comportamenti individuali e il manifestarsi di regolarità a livello 'collettivo'.

L'uso di tecniche di simulazione al computer si pone come obiettivo principale lo sviluppo di un metodo computazionale che permetta lo studio dei sistemi nel loro complesso; una metodologia di analisi che consenta di studiare i differenti processi sociali insieme, mentre si sviluppano e si influenzano reciprocamente.

Con la simulazione al computer si ha la possibilità di costruire 'società artificiali', costituite da agenti computazionali, utili per eseguire 'esperimenti di laboratorio' e per analizzare le conseguenze, in termini di comportamenti emergenti, di determinate configurazioni iniziali dei parametri dell'esperimento.

Grazie alle tecniche simulative diventa possibile, anche per le scienze sociali, effettuare verifiche sperimentali volte a controllare la validità delle teorie elaborate.

Una simulazione, infatti, può essere considerata come un 'laboratorio virtuale'; un luogo, cioè, dove il ricercatore sociale può osservare i fenomeni simulati in condizioni controllate, può modificare le condizioni dell'esperimento variando le variabili e i valori dei parametri e può verificare quali sono le conseguenze di queste sue manipolazioni.

[2] Il comportamento degli individui e il modo in cui le società si evolvono nel tempo sono il risultato di una moltitudine di cause diverse, che interagiscono in modi non sempre lineari e che non sono in genere isolabili dal contesto in cui si verificano.

L'importanza della simulazione come strumento di analisi è legata, inoltre, alla necessità di incorporare nei modelli, ipotesi di base più realistiche di quelle solitamente adottate dalle tradizionali metodologie utilizzate dalle scienze sociali.

In economia, la maggior parte dei modelli ipotizza che gli agenti siano perfettamente informati, abbiano infinita capacità computazionale e massimizzino una funzione di utilità esogena. Tali assunzioni risultano estremamente utili per generare soluzioni ai problemi teorici, ma 'pretendono' molto dal comportamento degli agenti, molto più di quanto questi possano dare. L'individuo, in questi modelli, appare quasi un 'robot' programmato per arricchirsi.

Nella realtà, invece, gli agenti non sempre assumono decisioni sulla base del criterio di efficienza economica, ma si fanno influenzare nelle proprie scelte da fattori diversi tra i quali: il grado di cultura, l'imitazione, l'istinto.

Nelle discipline sociali, inoltre, è una pratica comune, nella costruzione dei modelli, eliminare l'eterogeneità degli individui del mondo reale.

In economia ciò viene fatto sia esplicitamente, come avviene nei modelli microeconomici con agenti razionali, sia implicitamente, come avviene nei modelli aggregati usati per rappresentare i processi economici. Tali modelli, sebbene offrano una elevata capacità di analisi, non sono, tuttavia, in grado di cogliere le importanti conseguenze dell'eterogeneità degli agenti.

Grazie alla simulazione è possibile esercitare un maggior controllo delle ipotesi sottostanti i modelli; in particolare, è possibile superare l'impostazione basata sugli agenti perfettamente razionali e muoversi verso modelli con agenti a razionalità limitata (con informazione imperfetta, difficoltà di scelta, soggetti all'influenza dell'ambiente e degli altri agenti); gli individui, inoltre, possono essere differenziati nelle preferenze, nella disponibilità di informazioni, nel livello di 'intelligenza' realizzando, in questo modo, una reale eterogeneità di comportamento che rende più realistici i modelli e riduce lo scarto tra modelli teorici e realtà.

Un altro importante vantaggio della simulazione al computer è la possibilità di studiare più approfonditamente determinati aspetti dei fenomeni sociali, che le tradizionali tecniche di analisi delle scienze sociali non sono in grado di affrontare adeguatamente.

Nella maggior parte dei processi sociali l'analisi della dinamica risulta molto più significativa della ricerca dei punti di equilibrio. Le scienze sociali, tuttavia, hanno spesso concentrato la propria attenzione sulla ricerca di equilibri statici, tralasciando l'analisi della dinamica dei fenomeni oggetto di studio e più in generale il carattere temporale dei fenomeni umani[3].

La simulazione, invece, consente di approfondire l'analisi della dinamica dei processi sociali e di affrontare, grazie alle elevate capacità computazionali, aspetti della realtà (come la complessità del sistema economico) difficilmente analizzabili con i tradizionali strumenti di studio delle scienze sociali.

La simulazione con il computer rappresenta, infine, una grande opportunità di integrazione sia con le discipline che utilizzano tecniche di simulazione computazionale - scienza cognitiva, reti neurali, teoria ad

[3] I fenomeni sociali non sono indipendenti dal tempo: quello che i fenomeni sono oggi, dipende da quello che sono stati in passato e dall'evoluzione nel corso del tempo; le simulazioni consentono di tenere conto nei modelli della dimensione 'storica' dei fenomeni.

agenti - sia con le discipline appartenenti al vasto 'arcipelago' delle scienze sociali che talvolta appaiono eccessivamente frammentate e poco inclini alla sintesi[4].
Le metodologie simulative, essendo strumenti utilizzabili in qualunque campo di studio, rendono più facile la collaborazione interdisciplinare e consentono di far cadere alcune delle 'barriere artificiali' che dividono i vari campi di ricerca (e che costituiscono un ostacolo al progresso della conoscenza).

1.2 Utilizzo dei modelli di simulazione

Come affermato in Axtell (2000), una simulazione può avere tre differenti utilizzi; può essere impiegata:

- come semplice strumento per la presentazione dei risultati ottenuti analiticamente;
- come 'complemento' alla formalizzazione matematica;
- come uno strumento di analisi 'alternativo'.

Il primo utilizzo, che è anche il meno comune, si presenta quando le equazioni che descrivono il funzionamento del modello oggetto di studio possono essere completamente risolte o quando il modello è essenzialmente stocastico e si vuole caratterizzare lo spazio delle possibili soluzioni. In questi casi le simulazioni possono essere utilizzate come controllo-verifica delle soluzioni trovate analiticamente e come utile strumento per la descrizione dei risultati ottenuti.
Molto più importante, però, è l'utilizzo della simulazione come strumento di analisi complementare alla modellizzazione matematica. Nella maggior parte dei casi il modello che si vuole analizzare è solo parzialmente risolvibile mediante la manipolazione analitica. Può accadere, ad esempio, che non sia possibile giungere a determinare il punto di equilibrio di un sistema, o che l'equilibrio del sistema esista, ma non sia stabile o ancora che sia difficile determinare l'influenza di particolari parametri esogeni (e delle ipotesi effettuate) sui risultati ottenuti.
In queste circostanze è possibile ricorrere all'utilizzo della simulazione per analizzare il comportamento del modello.
Una caratteristica fondamentale dei modelli simulativi - che li rende particolarmente utili come strumento di analisi – è, infatti, la capacità di studiare sistematicamente la dinamica dei sistemi; una volta creato il modello, questo fornisce non solo un aspetto della soluzione – l'equilibrio o la stabilità – ma l'intero processo che conduce alla soluzione; effettuando esecuzioni ripetute del modello è possibile, poi, ottenere un quadro della dipendenza dei risultati dai parametri e dell'importanza delle ipotesi del modello.
In questi casi, le simulazioni rappresentano un utile aiuto all'intuizione del ricercatore; uno strumento complementare alla formalizzazione matematica che consente, spesso, di 'fare luce' sui meccanismi di funzionamento dei fenomeni analizzati.

[4] La maggior parte dei fenomeni reali sono interconnessi: è impossibile studiare e capire un fenomeno senza prendere in esame i fenomeni con cui è legato; grazie alle simulazioni è possibile tenere presenti nella teoria incorporata nel programma le interazioni tra fenomeni diversi, ma tra loro interconnessi.

Un ultimo utilizzo delle tecniche simulative si presenta, infine, quando il modello oggetto di studio risulta complesso[5] o intrattabile. Può accadere, ad esempio, che non esista una soluzione finita di un sistema di equazioni differenziali o che le equazioni governanti il modello siano non lineari.
Molti dei fenomeni considerati dalle scienze sociali ricadono proprio in quest'ultima definizione; in questi casi, la modellizzazione matematica, poiché implica l'uso di numerose semplificazioni che limitano il realismo e la plausibilità del fenomeno oggetto di studio, non rappresenta lo strumento adeguato di analisi; le simulazioni, grazie alla loro estrema flessibilità, sono spesso la sola tecnica disponibile in grado di studiare la dinamica di tali sistemi.

1.3 Gli obiettivi della simulazione

Gli obiettivi che possono essere perseguiti con la simulazione al computer sono numerosi, tuttavia il valore della simulazione, come metodologia di ricerca scientifica, risiede principalmente nei seguenti usi[6]:

a) previsione: la simulazione è in grado, dati una serie di input, di elaborare le informazioni possedute mediante un ipotizzato meccanismo di calcolo e di determinare le previsioni come conseguenza delle elaborazioni compiute sui dati;

b) verifica: la simulazione può essere utilizzata per dare validità al modello analizzato; a questo proposito si possono distinguere tre diversi tipi di validità che il modello può soddisfare:
   - la validità 'replicativa', quando i dati simulati riproducono i dati già acquisiti dalle osservazioni reali;
   - la validità 'previsionale', quando i dati simulati riproducono i dati reali prima che questi siano disponibili;
   - la validità 'strutturale', quando il modello non solo riproduce il comportamento del sistema osservato, ma riflette fedelmente il modo in cui il sistema reale opera per produrre il comportamento osservato;

c) scoperta: la simulazione è un utile strumento nella scoperta di nuove relazioni tra le variabili del modello; grazie alle tecniche simulative il ricercatore può saggiare la sensibilità del modello nei confronti della violazione di assunti fondamentali, può verificare le implicazioni delle differenti ipotesi e può valutare l'importanza dei parametri e delle variabili nel generare i fenomeni simulati (la simulazione risulta, pertanto, uno strumento utile non solo per derivare le conseguenze della teoria incorporata nel modello, ma anche per lo sviluppo della teoria e per la verifica della sua coerenza interna).

[5] Un sistema complesso è un sistema in cui sono diversamente coinvolti numerosi elementi indipendenti, legati da numerose interazioni, la cui varietà permette al sistema stesso nella sua totalità, di sviluppare un'auto-organizzazione spontanea.

1.4 Induzione, deduzione e simulazione

La simulazione al computer rappresenta un modo nuovo di fare ricerca scientifica, che si distingue dai metodi tradizionali dell'induzione e della deduzione e che in un certo senso ne rappresenta una 'sintesi'.
La simulazione, come la deduzione, muove infatti da alcune ipotesi esplicite, ma a differenza di quest'ultima non fornisce assiomi e teoremi; genera dati che possono essere analizzati induttivamente.
Mentre l'induzione può essere usata per trovare modelli nei dati, e la deduzione per trovare conseguenze di ipotesi, la simulazione al computer, poiché consente di condurre 'esperimenti di pensiero', può essere usata come un valido aiuto all'intuizione del ricercatore.
In economia la forma prevalente di modellizzazione è basata sul paradigma delle scelte razionali che, sebbene si caratterizzi per ipotesi di base poco realistiche, presenta il vantaggio di consentire spesso il ragionamento per deduzione.
Nel modello economico neoclassico, con agenti razionali che operano con disponibilità di informazioni e capacità di ottimizzare la propria funzione di utilità, le 'proprietà emergenti' possono essere formalmente dedotte.
La principale alternativa all'ipotesi delle scelte razionali è una qualche forma di comportamento adattivo. Quando si sceglie questa seconda impostazione con agenti interagenti che utilizzano regole che hanno effetti non lineari, le conseguenze dei processi adattivi sono spesso difficili da dedurre; le 'proprietà emergenti' sono spesso sorprendenti e risulta difficile anticipare le conseguenze di semplici forme di interazione; in questi casi, la simulazione è spesso l'unico strumento di analisi utilizzabile dal ricercatore.

1.5 *Agent Based Models* (ABM)

Come affermato in Epstein e Axtell (1996), grazie alla tecniche di simulazione al computer è possibile costruire 'società artificiali' da utilizzare per effettuare 'esperimenti di laboratorio' volti a ricreare i fenomeni e le strutture sociali oggetto di studio e di analisi.
I modelli basati su agenti, grazie alla loro estrema flessibilità, rappresentano una delle tecniche di simulazione più adatte per lo studio dei fenomeni sociali.
Negli ultimi anni, grazie anche ai progressi ottenuti nel calcolo computazionale, si è assistito ad un sensibile aumento del numero di applicazioni, non solo economiche, sviluppate secondo tale paradigma.
In un *Agent Based Model*, i sistemi sociali sono modellati come insiemi di entità autonome, denominate agenti; ogni agente del sistema viene rappresentato mediante algoritmi e variabili che ne definiscono il comportamento e ne registrano l'evoluzione dello stato nel tempo.
Le caratteristiche comportamentali degli agenti possono cambiare ed adattarsi nel corso della vita dell'individuo, in seguito alle interazioni con gli altri agenti e con l'ambiente[7].

---

[6] Le simulazioni possono essere utilizzate anche per scopi diversi dalla ricerca scientifica come l'apprendimento, la divulgazione e il divertimento.

[7] La visione dei processi sociali ed economici come sistemi di agenti autonomi che interagiscono ed evolvono non è nuova; tentativi di applicare tali idee al comportamento socioeconomico sono stati fatti già in

L'idea fondamentale che guida i modelli ABM è che comportamenti complessi possano essere il frutto delle interazioni fra agenti che operano, invece, sulla base di regole estremamente semplici.

La sfida di questo nuova metodologia di studio, che si pone per certi versi a metà tra i modelli analitici e l'osservazione empirica, è di spiegare l'emergere 'spontaneo' di regolarità nei processi sociali ed economici come conseguenza di un approccio di tipo *bottom up*.

I processi economici e sociali vengono visti, cioè, come conseguenza dell'interazione tra agenti autonomi, operanti in un ambiente 'artificiale' secondo proprie regole di comportamento, piuttosto che come frutto di meccanismi fittizi di coordinamento di tipo *top-down*[8].

Le strutture sociali ed economiche che emergono dalle simulazioni non vengono definite a priori, ma sono il risultato dell'interazione tra gli agenti; tali strutture sociali esercitano, inoltre, importanti effetti di *feedback* sugli agenti, modificandone il comportamento[9].

L'obiettivo principale dei modelli ABM è quello di svelare i meccanismi fondamentali che operano localmente, a livello di singolo agente, e che sono sufficienti a 'generare' strutture sociali e comportamenti collettivi di interesse.

In altre parole, l'obiettivo è la ricerca della specificazione del modello che conduce a generare la macrostruttura desiderata; se il modello è in grado di ricreare il fenomeno oggetto di studio, allora il modello rappresenta una possibile spiegazione del fenomeno stesso[10].

Grazie alle simulazioni basate su agenti diventa possibile collegare il livello dell'individuo con il livello dei fenomeni sociali. Anche se i fenomeni sociali risultano dal comportamento dei singoli individui, ciò non significa necessariamente che siano totalmente riconducibili agli individui stessi. Un fenomeno sociale è, infatti, un fenomeno complesso; è frutto delle interazioni tra individui e non può essere previsto o dedotto conoscendo anche perfettamente gli individui e il loro modo di comportarsi.

Le simulazioni consentono, a differenza dei metodi tradizionali di analisi, di studiare insieme gli individui e la società: il modo in cui interagiscono e si influenzano reciprocamente.

I modelli ABM, come affermato in Tesfatsion (2000)[11], combinano concetti e strumenti di differenti campi di ricerca - scienza cognitiva, intelligenza artificiale, automi cellulari, algoritmi genetici – in un modo che consente di perseguire tre importanti sviluppi:

---

passato, ma si sono scontrati con le difficoltà di calcolo computazionale (quello che i modelli ABM offrono è la possibilità di sfruttare la potenza dei nuovi strumenti computazionali).

[8] Costruire' un modello ABM equivale semplicemente a creare una popolazione di agenti, lasciarla interagire e monitorare quello che succede; tali modelli sono implementati in modo naturale attraverso linguaggi di programmazione orientati agli oggetti, in cui gli agenti e l'ambiente sono oggetti con propri stati interni e regole comportamentali (vedi Cap.3).

[9] In alcuni casi, quando l'obiettivo della simulazione è l'analisi di aspetti specifici dei fenomeni sociali, i modelli ABM possono definire ex-ante le strutture e istituzioni sociali ed economiche e focalizzare la propria attenzione sulle conseguenze dell'interazione tra gli individui.

[10] Può accadere, tuttavia, che più specificazioni del modello siano in grado di generare la situazione desiderata; in questo caso si procede ad un'analisi delle diverse configurazioni al fine di determinare quale sia la più fondata da un punto di vista empirico.

[11] In Tesfatsion (2000), invece del termine (di applicabilità più generale) ABM si utilizza il termine più specifico ACE (Agent-based computational economics).

- la elaborazione di teorie basate sull'interazione di agenti adattivi più o meno complessi;
- la verifica, il raffinamento e l'estensione di tali teorie attraverso la verifica sperimentale, l'analisi statistica dei risultati ottenuti e il confronto con studi analitici ed econometrici;
- la formulazione e la verifica di teorie socioeconomiche integrate compatibili con le teorie e i dati provenienti dai differenti campi della ricerca sociale.

La metodologia basata su agenti presenta, tuttavia, uno svantaggio rispetto alla formalizzazione matematica: il problema della robustezza dei risultati ottenuti.
Nelle teorie economiche classiche, la verifica della validità dei risultati è spesso formalmente risolvibile; negli ABM, invece, il solo modo per valutare la validità dei risultati ottenuti è di effettuare esecuzioni multiple del modello, variando sistematicamente i parametri o le condizioni iniziali (data la natura dinamica di tali modelli sono necessari numerosi 'esperimenti' prima di dimostrare la convergenza dei comportamenti individuali alla rappresentazione statica e formale di un teorema).

La simulazione al computer di 'società artificiali' richiede la definizione di agenti dotati di proprie regole di comportamento e di un ambiente nel quale tali agenti possano operare ed interagire.
Nei paragrafi successivi verranno illustrate alcune tecniche utilizzate, nei modelli ABM, per la costruzione di agenti 'artificiali' e verrà fornita una particolare rappresentazione dell'ambiente.

1.6 Tecniche di costruzione degli agenti

Il termine 'agente' viene utilizzato in numerosi campi della ricerca sociale; a volte, tuttavia, con notevoli diversità di significato. Nei modelli ABM, il termine 'agente' viene utilizzato per indicare un processo sviluppato al computer che possiede le seguenti proprietà[12]:

- autonomia : controlla il proprio stato e le proprie azioni, senza che sia necessario un intervento diretto da parte di entità esterne;
- abilità sociale : interagisce con gli altri processi-agenti mediante un 'linguaggio' comune;
- reattività : è in grado di percepire l'ambiente in cui vive e di rispondere in modo tempestivo ai cambiamenti che si verificano nell'ambiente;
- '*pro-activity*' : non agisce semplicemente in risposta a stimoli provenienti dall'ambiente, ma è in grado di prendere iniziative; è capace, cioè, di esibire un comportamento finalizzato al raggiungimento di un dato obiettivo[13].

[12] Si fa, qui, riferimento alla nozione 'debole' di agente intelligente illustrata in Wooldridge e Jennings (1995).
[13] Gli 'agenti' dei modelli di simulazione sono processi computazionali il cui scopo è di modellare, in modo molto semplificato, le capacità degli individui; tali processi computazionali vengono utilizzati come metafore degli 'individui' e del loro comportamento; ogni attribuzione di intenzionalità ad un 'agente' deve, quindi, sempre essere considerata metaforica.

Vi sono diverse tecniche utilizzabili per la costruzione degli agenti; l'utilizzo di una tecnica, invece di un'altra, dipende spesso dallo scopo che si pone la simulazione. Alcuni caratteristiche, tuttavia, sono comuni; in particolare ogni metodologia deve garantire alcune funzionalità di base che permettano all'agente di ricevere input dall'ambiente, di registrare una storia delle precedenti azioni, di elaborare i dati posseduti al fine di determinare le azioni future e infine di eseguire le azioni e di valutarne gli effetti.

L'obiettivo di realizzare agenti, come 'sistemi intelligenti artificiali', può essere raggiunto utilizzando tre differenti paradigmi di studio: l'intelligenza Artificiale, la Vita Artificiale e il connessionismo[14].

Prima di illustrate, in breve, alcune caratteristiche che contraddistinguono questi tre campi di ricerca è opportuno fare una descrizione sintetica del processo di evoluzione dei sistemi intelligenti. Tale evoluzione si è, infatti, caratterizzata per una dinamica complessa e poco lineare.

L'idea di costruire sistemi artificiali in grado di riprodurre alcuni aspetti dell'intelligenza umana ha potuto cominciare ad essere messa in pratica circa cinquant'anni fa, all'inizio degli anni '50, grazie al rilevante contributo di importanti studiosi, come Turing e von Neumann, e allo sviluppo della tecnologia dei computer.

Dopo la fase iniziale in cui i vari approcci disciplinari e di ricerca erano sostanzialmente intrecciati e convivevano insieme, a metà degli anni '60 con l'affermazione del calcolatore elettronico si assiste all'accantonamento della neurocibernetica e all'emergere dell'Intelligenza Artificiale, quale disciplina destinata a rendere i calcolatori elettronici capaci di prestazioni 'intelligenti'.

L'architettura alla von Neumann dei computer diventa il modello per studiare e riprodurre l'intelligenza e proprio per tale ragione lo studio dell'intelligenza artificiale, concepita come attività simbolica e razionale, si separa dallo studio dell'intelligenza biologica (che opera su principi di base diversi).

I modelli neurali, con il loro accento sull'apprendimento, l'adattamento e l'autorganizzazione incontrano, in questi anni, difficoltà teoriche e vengono accantonati.

All'inizio degli anni '80, però, proprio quando l'Intelligenza Artificiale si andava consolidando e andava acquisendo credibilità scientifica ed economica con i suoi primi successi commerciali, cominciano ad emergere alcuni limiti di tale paradigma computazionale e si assiste ad uno spostamento dell'interesse della comunità scientifica verso i più flessibili sistemi computazionali che fanno riferimento al cervello e ai meccanismi di funzionamento dei processi biologici ed evolutivi.

Allo stato attuale vi sono sostanzialmente due comunità di ricercatori interessati a costruire sistemi intelligenti: quella dell'IA, costituita prevalentemente da informatici e legata alla logica, e una comunità che raggruppa connessionisti e ricercatori della Vita Artificiale, caratterizzata invece, da apporti interdisciplinari di matematici, fisici, informatici, biologi e neuroscienziati[15].

1.6.1 Intelligenza artificiale (IA)

[14] I confini tra queste discipline non sono definiti precisamente; il connessionismo e la Vita Artificiale, ad esempio, presentano numerose caratteristiche comuni tra le quali la natura tipicamente distribuita e parallela e l'analogia tra i principi di funzionamento dei sistemi artificiali elaborati da tali discipline e i principi di funzionamento dei meccanismi biologici.

[15] Anche se persistono alcune incompatibilità di fondo tra i vari approcci, in particolare tra quello connessionista e quello simbolico dell'Intelligenza Artificiale, la tendenza in atto sembra, tuttavia, convergere verso un'integrazione dei diversi paradigmi di ricerca.

Con il termine 'Intelligenza Artificiale' si fa riferimento all'insieme di teorie e di metodologie che consentono la realizzazione di sistemi artificiali - volti a fornire modelli computazionali dei processi cognitivi - in grado di

produrre comportamenti 'intelligenti'[16].

L'IA, fin dall'inizio ha abbracciato come metodologia per la generazione di comportamenti intelligenti la programmazione dei calcolatori seriali; l'architettura di questi sistemi è, infatti, quella definita da von Neumann; il sistema è diviso in due parti fondamentali: una unità centrale di elaborazione (la CPU dei computer) e una memoria passiva di dati; il sistema funziona in quanto l'unità centrale esegue una sequenza di istruzioni sui dati conservati nella memoria.

Secondo tale impostazione un sistema intelligente è fondamentalmente un sistema di elaborazione delle informazioni, che interagisce con un ambiente complesso e che prescinde dalla struttura biologica di un cervello umano; con le tecniche di IA, la generazione di comportamenti intelligenti non ha nessuna relazione con il modo in cui l'intelligenza viene generata nei sistemi naturali; le proprietà dell'intelligenza - secondo tale disciplina - sono proprietà puramente funzionali, tali da poter essere realizzate mediante la semplice programmazione al computer.

L'IA si basa, in altre parole, su una concezione simbolica dell'intelligenza; l'intelligenza viene considerata, cioè, come un'attività sequenziale sistematica e razionale volta al raggiungimento di un determinato scopo[17].

Come affermato in Langton (1992), l'Intelligenza Artificiale, di fatto, si è soprattutto concentrata sulla produzione di 'soluzioni intelligenti', piuttosto che sulla produzione di 'comportamenti intelligenti' (l'IA, ad un esame oggettivo, appare soprattutto come una tecnologia e una branca dell'informatica più che una scienza dell'intelligenza naturale; tale disciplina non è, infatti, interessata a formulare modelli teorici dell'intelligenza effettiva degli essere umani; è, invece, rivolta a realizzare 'sistemi intelligenti' utili sul piano pratico).

Di seguito viene descritto uno sviluppo particolare dell'IA utilizzabile per la costruzione di agenti nell'ambito dei modelli basati su agenti: i sistemi esperti. Grazie a questa tecnica è possibile concepire l'agente economico come un sistema in grado di costruirsi un proprio insieme di regole e di adattarsi dinamicamente ai mutamenti dell'ambiente.

1.6.1.1 Sistemi esperti

[16] Nell'ambito dell'Intelligenza Artificiale è possibile individuare due distinte impostazioni: un'impostazione 'debole', secondo la quale i computer rappresentano un strumento utile per l'analisi della mente, così come sono utili per lo studio, ad esempio, dei processi biologici e una impostazione 'forte' secondo la quale i modelli basati sul calcolatore sono in grado di replicare la 'mente'; nel primo caso il computer, opportunamente programmato, viene utilizzato per simulare i processi cognitivi umani (senza nessuna ipotesi sulla verosimiglianza del processo simulato con quello mentale); nel secondo caso, invece, sulla base della considerazione che il pensiero non è altro che manipolazione di simboli formali, il computer viene utilizzato (essendo in grado di manipolare tali simboli mediante opportuni programmi) come lo strumento ideale (la metafora) per 'replicare' le capacità cognitive dell'uomo.

[17] Tale concezione depura l'attività mentale dalle sue irrazionalità e ambiguità per costruire sistemi efficaci ed efficienti.

Un sistema esperto è ' un programma al computer che esegue un compito normalmente svolto da un essere umano con specifiche competenze'[18].

Tali dispositivi, che rappresentano il successo più grande dell'Intelligenza Artificiale, sono stati utilmente impiegati in numerose applicazioni industriali.

L'obiettivo dei sistemi esperti è quello di rendere esplicite, mediante l'uso di un sistema di regole, le conoscenze e i modi di operare degli esperti umani; per realizzare tale obiettivo, i sistemi esperti devono possedere le seguenti caratteristiche fondamentali:

a) inferenza: il programma deve essere in grado di trarre conclusioni - di assumere decisioni - anche senza disporre di tutta l'informazione possibile sul problema da affrontare;
b) acquisizione interattiva dei dati: il programma deve gestire, in modo efficiente, l'acquisizione diretta di nuove informazioni dall'ambiente;
c) giustificazione delle conclusioni: il sistema esperto deve essere in grado di giustificare il processo mediante il quale viene adottata una decisione; deve, cioè, disporre di un controllo sulla logica interna del sistema;
d) struttura modulare: il sistema esperto è formato generalmente da tre componenti principali:
    - un insieme di conoscenze di base con le informazioni sul problema specifico che il sistema esperto deve gestire; in genere, le informazioni vengono rappresentate da regole di comportamento del tipo 'IF-THEN'; tali regole sono costituite da due parti: una condizione che specifica quando la regola deve essere applicata ed una azione che specifica quali debbano essere le conseguenze dell'attivazione della regola;
    - il 'motore di inferenza': il compito di tale dispositivo è di determinare, in risposta agli stimoli provenienti dall'esterno, quali azioni devono essere attivate e di gestire la memoria di lavoro del sistema[19]; è la componente fondamentale del sistema esperto; in teoria, è indipendente dal tipo di problema specifico che il sistema deve gestire; non varia, cioè, da un sistema esperto ad un altro;
    - un'interfaccia utente: tale componente collega il 'motore di inferenza' all'ambiente esterno usando le tecniche di programmazione standard.

Benché i sistemi esperti riescano a replicare le abilità degli esseri umani in determinati campi di competenza, rimangono, tuttavia, notevolmente limitati per via della specificità del loro campo di applicazione; inoltre, tali dispositivi non hanno una delle caratteristiche fondamentali dell'intelligenza: la capacità di apprendere ed aumentare automaticamente le proprie conoscenze.

Usando un sistema esperto è relativamente facile costruire agenti in grado di rispondere agli stimoli provenienti dall'ambiente; risulta, invece, molto più difficile costruire agenti 'cognitivi' con capacità di 'riflettere' sulle decisioni da prendere. I sistemi esperti, infatti, sono in grado di reagire a situazioni previste,

[18] Gallant S.(1993).

[19] L'applicazione di una regola può modificare lo stato del sistema e realizzare la condizione che determina l'applicazione di un'altra regola; il motore di inferenza agisce, perciò, in modo iterativo sull'insieme di regole.

ma non a situazioni che non essendo prevedibili da un essere umano non sono state incluse nel programma e nell'insieme delle regole che costituiscono il sistema esperto[20].

Tali meccanismi si caratterizzano, quindi, per un'eccessiva rigidità, tipica dell'impostazione simbolica, sequenziale e programmatoria dell'intelligenza.

1.6.2 Connessionismo

Il connessionismo capovolge in maniera radicale la maggior parte delle assunzioni del paradigma cognitivista.

Mentre le tecniche di Intelligenza Artificiale si pongono come obiettivo principale la produzione di comportamenti 'intelligenti' , ignorando il meccanismo di funzionamento sottostante, cioè il cervello, il connessionismo si pone come obiettivo la produzione di sistemi artificiali 'intelligenti' che siano costruiti proprio sulla base dei principi strutturali e di funzionamento del cervello.

A differenza dell'IA, il paradigma connessionista utilizza il computer, non come modello della 'mente', ma come un utile strumento per la simulazione del meccanismo di funzionamento del cervello; meccanismo che sta alla base del comportamento intelligente degli individui.

Come affermato in Parisi (1989), la differenza tra connessionismo e IA è, in sostanza, la differenza che si riscontra tra emulare l'intelligenza e simulare l'intelligenza; nel primo caso i sistemi intelligenti vengono costruiti senza preoccuparsi molto che la loro intelligenza sia internamente simile a quella umana; nel secondo caso, invece, i sistemi cercano di riprodurre alcuni aspetti del meccanismo di funzionamento dell'intelligenza umana.

I modelli connessionisti sono sistemi che fanno emergere le loro proprietà dal comportamento collettivo di un elevato numero di unità elementari che hanno caratteristiche quantitative e che interagiscono in modo parallelo.

Di seguito è illustrata, brevemente, la struttura principale del connessionismo: la rete neurale artificiale.

1.6.2.1 Reti Neurali Artificiali (RNA): un accenno[21]

Le reti neurali artificiali sono algoritmi di calcolo ispirati ai meccanismi di funzionamento delle connessioni nervose del cervello.

In breve, una RNA, consiste di tre o più strati (*layers*) interconnessi di unità elementari di calcolo, denominate neuroni (o nodi). Il primo strato di neuroni (*input*) riceve le informazioni dall'ambiente, le elabora e trasmette i risultati allo strato successivo (*hidden*); questo, a sua volta, manipola i segnali ricevuti e li trasmette allo strato finale (*output*). Lo strato *output*, infine, trasferisce il risultato delle operazioni compiute all'esterno.

[20] Tali situazioni se si presentano possono determinare il mancato funzionamento del sistema di decisione

[21] In questo paragrafo viene fornita solo una definizione generale di RNA; un'analisi approfondita del meccanismo di funzionamento viene effettuata nel prossimo capitolo.

Come affermato in Parisi (1999), le unità elementari della RNA hanno alcune caratteristiche essenziali delle cellule nervose - i neuroni del sistema nervoso reale - mentre le connessioni, attraverso le quali un'unità influenza fisicamente le altre unità con cui è collegata, hanno alcune delle caratteristiche essenziali dei collegamenti sinaptici.

Ogni neurone della RNA riceve input provenienti dai nodi dello strato precedente (o dall'ambiente se fa parte dello strato input), li 'aggiusta' secondo un meccanismo di calcolo basato sui pesi di interconnessione tra i diversi nodi e trasmette il risultato delle proprie elaborazioni – l'applicazione di una particolare funzione ai segnali ricevuti - ai neuroni dello strato successivo.

L'output che la rete genera in risposta all'input ricevuto dall'esterno dipende dai pesi delle connessioni; se tali pesi cambiano nel tempo anche il comportamento della rete si modifica.

I pesi di interconnessione tra i diversi nodi - poiché sono i soli elementi della rete neurale capaci di memorizzare informazione - sono i responsabili del comportamento e della capacità di apprendimento del modello neurale e giocano, quindi, un ruolo fondamentale nella costruzione della rete.

Una rete neurale artificiale, mediante particolari meccanismi di calcolo (come l'algoritmo di *backpropagation* dell'errore, descritto nel prossimo capitolo) può essere adattata in modo che ad ogni insieme di input provenienti dall'ambiente corrisponda un differente insieme di output. Ciò, in pratica, viene realizzato facendo 'apprendere' la RNA sulla base di un insieme di 'esempi noti' (*target*) e aggiustando il valore dei pesi dei singoli neuroni, fino a quando l'output della rete non corrisponde all'output desiderato.

Grazie a questo processo di confronto tra dati prodotti e dati desiderati, la rete neurale diventa 'capace' di apprendere e di applicare a circostanze nuove le conoscenze acquisite (le RNA, data tale proprietà, risultano quindi particolarmente adatte a rappresentare, metaforicamente, le capacità cognitive di agenti a razionalità limitata).

1.6.2.2 Alcune considerazioni

I sistemi connessionistici sono sistemi costruiti in modo tale da apprendere con l'esperienza, senza una conoscenza predefinita; i sistemi realizzati con le tecniche dell'IA, invece, non sono sviluppati per apprendere: il programma di istruzioni in cui risiede l'intelligenza del sistema - sviluppato da un essere umano - non si modifica durante l'applicazione.

Se un sistema esperto sa dare la risposta appropriata ad un certo stimolo – cioè possiede una regola del tipo IF-THEN – questo non significa che sappia automaticamente estrapolare questa conoscenza a stimoli nuovi. Nei sistemi connessionistici, come le reti neurali artificiali, invece, vi è un'intrinseca capacità di estrapolare quello che è stato appresso.

Come affermato in Parisi (1989), con le tecniche di IA, un sistema computazionale è in grado di fornire una certa prestazione perché è stato esplicitamente programmato da un essere umano a svolgere quella prestazione (il programmatore formula la procedura che consente di svolgere una determinata funzione e la realizza in un linguaggio di programmazione).

Le reti connessionistiche, a differenza di quanto avviene con le tecniche di Intelligenza Artificiale, non devono essere programmate a svolgere un determinato compito da un essere umano; apprendono

spontaneamente, tramite il processo di *learning*, come svolgere il compito loro attribuito senza che sia necessario un intervento esterno.

Per il connessionismo, in sostanza, un sistema artificiale intelligente non è un sistema che possiede una certa capacità, ma è un sistema che acquisisce , attraverso un processo di evoluzione o di apprendimento, tale capacità.

Ciò appare molto più vicino a quello che comunemente si intende per intelligenza; un sistema, infatti, è intelligente se scopre autonomamente come comportarsi, se dispone di meccanismi che gli consentano di scoprire nuovi modi di espletare le proprie funzioni e di affrontare i problemi[22].

Il connessionismo è, dunque, uno strumento più efficace per la realizzazione di sistemi intelligenti rispetto alle tecniche di IA. Queste ultime, infatti, si limitano a realizzare una semplice trascrizione delle procedure - note a priori - per risolvere compiti specifici. Con il connessionismo, invece, i sistemi sviluppati apprendono e si autoorganizzano in modi potenzialmente imprevedibili.

L'IA, se da un lato ha compiuto importanti progressi nella riproduzione di alcuni aspetti dell'intelligenza, dall'altro - ignorando del tutto le caratteristiche fisiche e il modo di funzionare del cervello – ha finito per non riuscire a cogliere alcuni delle proprietà più importanti dell'intelligenza come:

- la flessibilità : il sistema deve adattare le risposte alle circostanze nuove;
- la robustezza: il sistema non deve bloccarsi di fronte a difficoltà;
- la sensibilità al contesto: la risposta ad una situazione deve tenere conto del contesto;
- la capacità di apprendimento: il sistema deve essere in grado di modificare spontaneamente le proprie risposte sulla base dell'esperienza maturata.

Il connessionismo permette di superare, entro certi limiti, tali ostacoli.

Connessionismo e IA differiscono, infine, per l'atteggiamento nei riguardi dell'errore.

Gli essere umani fanno continuamente errori dovuti ad imprecisione nei concetti, limiti di memoria o altro. L'IA considera tali errori come qualcosa da evitare, che rende meno efficiente il modo di ragionare e di agire del 'sistema intelligente'. Gli errori, tuttavia - come affermato in Parisi (1989) - possono essere un meccanismo fondamentale di esplorazione di possibilità e di sviluppo; con le tecniche connessionistiche l'intelligenza viene sviluppata proprio sulla base degli errori (il sistema intelligente – la rete neurale – viene addestrata mediante un processo iterativo di prove ed errori).

1.6.3 Vita Artificiale (*Alife*)

La Vita Artificiale - nata all'inizio degli anni 80 per opera del biologo Christopher Langton - ha come obiettivo la generazione di comportamenti di tipo biologico; tale paradigma di ricerca, che utilizza strumenti metodologici e concettuali vicini a quelli del connessionismo e della teoria dei sistemi dinamici, intende sintetizzare, mediante l'utilizzazione del computer, il processo dell'evoluzione e applicare i principi che lo governano alla soluzione di problemi di natura scientifica.

[22] Un sistema che si limiti a eseguire solamente ciò che gli è stato insegnato non può essere definito intelligente.

Come affermato in Langton (1992), mentre l'IA usa il computer come un modello dell'intelligenza, la vita artificiale cerca di sviluppare un paradigma computazionale basato sui processi naturali che caratterizzano gli esseri umani; il computer viene utilizzato per costruire modelli dei meccanismi biologici basilari sottostanti all'evoluzione e alla vita stessa, anziché per simulare processi di pensiero.

Grazie alle tecniche sviluppate da tale campo di ricerca è possibile realizzare sistemi intelligenti in grado di apprendere sulla base dell'evoluzione genetica; le capacità di un sistema di comportarsi in modo 'intelligente' non emergono come conseguenza di un processo di apprendimento, come avviene per le reti neurali, ma emergono attraverso meccanismi di selezione artificiale.

Le metodologie sviluppate dall'*Alife* sono basate sull'idea darwiniana dell'evoluzione come risultato di variabilità e di selezione; invece di avere un ambiente che istruisce un sistema su come modificarsi al fine di esibire comportamenti adatti alle diverse situazioni - come avviene con i metodi di addestramento delle reti neurali - i metodi evoluzionistici operano su popolazioni di sistemi che variano tra di loro; l'ambiente seleziona i sistemi migliori, i più adatti[23].

Di seguito sono illustrati due dei principali metodi evoluzionistici: gli algoritmi genetici e i classifier system.

1.6.3.1 Algoritmi genetici

Gli algoritmi genetici sono 'algoritmi di ricerca basati sui meccanismi di selezione naturale e sulla genetica. Integrano un principio di sopravvivenza della struttura maggiormente adatta all'ambiente con un meccanismo di scambio di informazioni, strutturato ma stocastico, per formare un algoritmo di ricerca che presenta un po' dell'intuito innovativo insito nella ricerca umana.'[24].

Ispirati alla teoria evoluzionistica dei sistemi biologici, gli algoritmi genetici sono meccanismi di calcolo, altamente paralleli, che 'trasformano popolazioni di oggetti matematici individuali, in genere sequenze binarie di lunghezza fissa, in nuove popolazioni utilizzando operatori mutuati dalle operazioni genetiche naturali come la riproduzione sessuata e la riproduzione proporzionale al grado di adattamento'[25].

Il meccanismo di base tipico degli algoritmi genetici prevede le seguenti fasi:

a) generazione della popolazione: sulla base della rappresentazione del problema che è stata definita[26] viene creata (generalmente in modo casuale) una popolazione iniziale composta di un certo numero di individui; ciascun individuo, creato in modo indipendente dagli altri, viene rappresentato come una combinazione di valori binari (stringa o genotipo), in numero sufficiente a contenere tutta la conoscenza che risulta necessaria a descrivere un qualche tipo di processo o di strategia;

---

[23] L'applicazione delle idee evolutive si presta bene alla descrizione del processo economico: l'ambiente-mercato seleziona le imprese con le politiche di comportamento più competitive che rendono le imprese più abili a sopravvivere e crescere.

[24] Goldberg (1989)

[25] Koza (1990)

[26] Gli algoritmi genetici richiedono che i parametri del problema vengano codificati sotto forma di una sequenza di caratteri, storicamente appartenenti al sistema binario; recentemente, tuttavia, negli schemi di codifica si è passati dagli alfabeti binari a quelli di cardinalità superiore; in alcuni casi si sono adottati anche valori reali in ogni singola posizione del genotipo dell'individuo.

b) valutazione degli individui: ad ogni individuo è associato un valore, rappresentativo del grado di adattamento alla funzione da svolgere nell'ambiente (*fitness*); il valore della *fitness*, calcolato in base ai risultati ottenuti applicando la strategia di cui è portatore l'individuo, è di fondamentale importanza, per la successiva selezione degli individui ai fini della riproduzione;

c) riproduzione: in questa fase avviene la selezione degli individui della popolazione e la definizione della tecnica di riproduzione vera e propria; l'obiettivo della selezione è di conferire maggiori probabilità di riproduzione agli individui che hanno il più elevato grado di adattamento; la tecnica generalmente utilizzata è la *roulette wheel selection* che assicura ad ogni individuo una probabilità di riproduzione proporzionale alla propria fitness; in questa fase vengono, anche, definiti gli aspetti quantitativi della riproduzione; in particolare viene stabilito se è l'intera popolazione che viene sostituita da una generazione all'altra o se, invece, sono soltanto alcuni individui che nascono e vanno a sostituire quelli della generazione precedente;

d) applicazione degli operatori 'genetici': mediante le tecniche di 'crossover' e di mutazione vengono fatti nascere nuovi individui, non presenti nella popolazione iniziale (viene esplorato lo spazio delle soluzioni); il crossover, spesso assimilato alla riproduzione sessuata, agisce condizionatamente ad una data probabilità e coinvolge due individui, denominati 'genitori', che fanno confluire il loro patrimonio genetico creando così due nuovi discendenti; il crossover si basa sull'individuazione, in modo casuale, di una posizione qualunque interna alla stringa in cui avverrà la scissione del patrimonio genetico dei due individui e sul successivo scambio delle due sottosequenze successive (o precedenti) a tale posizione[27]; la mutazione, invece, simula gli errori di copia e trasmissione del patrimonio genetico da una generazione ad un'altra e consiste nel cambiamento condizionato ad una data probabilità del valore presente nelle singole posizioni della stringa degli individui;

e) ripetizione del ciclo per un certo numero di generazioni.

Gli algoritmi genetici, formalizzati in Holland (1975), hanno ricevuto negli ultimi anni una notevole attenzione da parte degli studiosi di scienze sociali e di economia.

Il crescente interesse verso tale metodologia può essere attribuito alla estrema generalità del loro approccio e al notevole potenziale applicativo. Gli algoritmi genetici possono essere utilizzati, infatti - grazie all'elevato grado di parallelismo che li caratterizza (e alla capacità di esplorazione intelligente dello spazio delle soluzioni) - per la soluzione di problemi di ottimizzazione numerica e di ricerca combinatoria, ma anche - data la completa assenza di requisiti sulla natura e struttura del problema da risolvere e dell'eventuale funzione sottostante - come un efficace strumento per la costruzione di agenti 'adattivi' nell'ambito dei modelli ABM[28].

---

[27] Esempio:

A = 1 0 1 0 1 | 0 0 0 C = 1 0 1 0 1 | 1 0 1

B = 0 1 1 0 1 | 1 0 1 D = 0 1 1 0 1 | 0 0 0

date le stringhe A e B, si ottengono mediante l'applicazione del crossover C e D.

[28] Nella costruzione di agenti economici gli algoritmi genetici possono essere utilmente utilizzati congiuntamente alle reti neurali: in questo modo diventa possibile abbinare all'apprendimento individuale, realizzato mediante i modelli neurali, una forma di apprendimento collettivo.

1.6.3.2 Classifier System

Un *classifier system* è un sistema di apprendimento automatico che impara regole sintatticamente semplici, denominate *classifier*, per guidare le sue azioni in un ambiente arbitrario[29].

Tale metodologia[30], che viene frequentemente applicata allo studio di sistemi dinamici e all'analisi del comportamento di agenti economici, è composta - come affermato in Margarita (1992) - dai seguenti elementi:

a) un insieme di stringhe di lunghezza fissa che rappresentano le regole di comportamento[31]; tali regole, costruite sulla base di un alfabeto ternario, sono composte da una condizione - rappresentata sotto forma di una sequenza di caratteri scelti tra i simboli {1, 0 , #}- e da una azione - costituita da una sequenza dei caratteri {1, 0}. Al valore 0 o 1 presente in ogni posizione corrisponde un particolare significato; il simbolo # viene, invece, usato come carattere jolly, con il significato di *don't care*; ad ogni classifier è associato, inoltre, un valore rappresentativo della propria forza(*fitness*); tale valore dipende dai risultati ottenuti effettuando l'azione suggerita dalla regola;
b) un sistema di sensori che riceve le informazioni provenienti dall'ambiente e determina quali sono i classifier da attivare; l'attivazione di una particolare regola avviene se ogni carattere della condizione è uguale a quello del messaggio proveniente dall'esterno (oppure al carattere #); in uno stesso momento più regole possono 'unire' la descrizione delle informazioni provenienti dall'ambiente con la propria condizione;
c) un sistema di asta che determina quale dei classifier attivati agisce effettivamente[32]; i classifier che vengono eseguiti possono a loro volta attivare altri classifier oppure produrre un'azione rivolta all'esterno; il classifier che viene selezionato per 'attuare' la propria azione paga un importo, proporzionale al proprio patrimonio, che viene ripartito tra i classifier che lo hanno attivato, contribuendo così ad aumentarne il valore (*bucket brigade algorithm*); in questo modo ciascuna regola può aumentare il proprio patrimonio sia perché ha potuto agire sull'ambiente in modo giudicato positivo, sia perché ha permesso l'attivazione di un'altra regola.
d) un sistema contabile che aggiorna il valore del patrimonio dei singoli classifier sulla base dei premi che vengono percepiti in seguito alle decisioni prese;

[29] Goldberg (1989).

[30] I *classifier system* possono essere considerati un'evoluzione dei sistemi esperti, dai quali si differenziano per alcune caratteristiche fondamentali: il parallelismo nella generazione delle regole, l'apprendimento automatico dell'importanza da attribuire alle singole regole e, soprattutto, la creazione di nuove regole, effettuata autonomamente per mezzo degli algoritmi genetici.

[31] La popolazione iniziale di regole del classifier system può essere generata casualmente o in modo da incorporare regole definite a priori.

[32] Se viene seguito un approccio deterministico, il classifier eseguito sarà quello che dispone del patrimonio più elevato; se, invece, viene seguito un approccio stocastico la probabilità di un classifier di essere eseguito sarà proporzionale al proprio patrimonio.

e) un algoritmo genetico che introduce nel sistema nuovi set di regole in sostituzione di quelle vecchie; l'algoritmo, che viene generalmente attivato quando un messaggio in entrata non corrisponde a nessuno dei classifier presenti nel sistema, consente di operare sull'insieme di regole le seguenti operazioni[33]:
   - diversificazione: se in risposta ad un messaggio proveniente dall'ambiente tutti i classifier rispondono con la medesima azione, viene creato una nuova regola con stessa condizione e diversa azione in sostituzione di un classifier a bassa fitness allo scopo di mantenere un adeguato livello di diversità delle azioni da compiere in una data situazione;
   - creazione: quando nessuna condizione 'unisce' il messaggio ricevuto dall'ambiente, viene creata un nuova regola che ha per condizione il messaggio ricevuto; l'azione, invece, viene scelta casualmente; tale regola va a sostituire un classifier, già esistente, con un basso valore di patrimonio;
   - generalizzazione: attivata con probabilità decrescente nel tempo, determina la nascita di due nuovi classifier, mediante crossover applicato a due regole scelte casualmente tra quelle a più elevata fitness; alle regole generate viene assegnato un patrimonio pari alla media di quello delle regole 'generatrici'. I classifier così creati, prendono il posto nella popolazione di due classifier, scelti casualmente tra quelli a bassa fitness;
   - specializzazione: quando è stata determinata la regola da attivare, con probabilità decrescente nel tempo, si applica - allo scopo di creare una regola più specializzata - una mutazione dei caratteri # della condizione, in modo da trasformarli, con una certa probabilità in 0 o 1.

Le capacità di adattamento del classifier system dipendono dallo sviluppo di individui semplici e diversi tra loro specializzati nell'affrontare situazioni particolari. Rispetto agli algoritmi genetici, tali modelli evolutivi producono strategie - consistenti nell'insieme di regole che vengono fatte evolvere - in modo maggiormente esplicito.

Come affermato in Ferraris (2000), i classifier system possono essere utilizzati per descrivere i processi mentali degli agenti nell'ambito delle simulazioni ABM; questa impostazione si basa sulla considerazione che ogni individuo possa avere nella sua mente con riferimento ad uno stesso problema diverse strategie di soluzione reputate praticabili[34]; i metodi evolutivi intervengono sia nella scelta della strategia migliore che nella produzione di nuove strategie.

Un esempio di applicazione di un classifier system per la descrizione del comportamento di un agente economico è illustrato nel terzo capitolo (il modello "*Artificial Stock Market*").

1.6.4 Una piccola digressione: agenti *mind – no mind*

[33] Tutte le selezioni vengono operate con criteri non deterministici allo scopo di mantenere un certo grado di eterogeneità della popolazione, in modo da garantire una esplorazione sufficientemente accurata dello spazio delle soluzioni.

[34] La plausibilità di tale impostazione emerge chiaramente quando si osservi come gli agenti, nella realtà, non siano in grado di stimare una strategia in termini assoluti (se non per problemi banali) ma solo di effettuare osservazioni comparative.

Il punto di partenza nella costruzione dei modelli simulativi è la definizione degli agenti; come visto sopra, vi sono diverse tecniche utilizzabili per la costruzione di 'individui intelligenti'.

In questo paragrafo, si vuole approfondire un aspetto particolare delle simulazioni con modelli ABM: il livello di intelligenza che gli agenti di una simulazione economica devono possedere e di conseguenza il livello di complessità del modello.

La costruzione di agenti 'sofisticati' con capacità cognitive rappresenta sicuramente un vantaggio per il realismo della simulazione; come affermato in Epstein e Axtell (1996), il sistema socioeconomico è una complicata struttura contenente milioni di unità interagenti come consumatori, imprese, banche; sono queste unità che prendono decisioni circa i consumi, il risparmio e l'investimento; è, quindi, ragionevole aspettarsi che il comportamento di un modello sia più realistico se basato sulla conoscenza di tali unità economiche (come si comportamento, come rispondono a cambiamenti esterni e come interagiscono).

Secondo alcuni ricercatori, tuttavia, i modelli ABM non devono necessariamente avere lo scopo di fornire una accurata e precisa rappresentazione del fenomeno oggetto di studio; lo scopo dei modelli ABM, secondo tale filone di ricerca, è di aiutare la comprensione dei processi fondamentali che sono alla base dell'emergere dei fenomeni macroscopici.

Secondo Axelrod (1997) nella costruzione dei modelli ABM occorre adottare il principio KISS: *'keep it simple, stupid*"; nelle scienze sociali, infatti, modelli di simulazione altamente complicati raramente forniscono una spiegazione chiara del meccanismo di funzionamento del fenomeno analizzato. I ricercatori sociali hanno invece, spesso, scoperto importanti relazioni e principi dall'analisi di semplici modelli di simulazione. A volte più semplice è il modello, più facile risulta scoprire e comprendere i sottili effetti dei meccanismi di comportamento dei processi considerati, al variare dei parametri e delle condizioni iniziali.

Per Axelrod, il punto fondamentale è che mentre il fenomeno oggetto di studio può essere complesso, le ipotesi sottostanti il modello ABM devono essere semplici.

La complessità del modello ABM, non deve risiedere nelle ipotesi del modello, ma deve emergere nei risultati simulati[35].

I ricercatori hanno, infatti, limitate capacità cognitive; quando dalla simulazione emerge un risultato interessante, è bene che lo sviluppatore del modello possa capire l'origine di tale risultato; la semplicità, inoltre, è necessaria al fine di consentire ad altri ricercatori di estendere il modello in nuove direzioni.

In Terna (2001), al fine di classificare i diversi approcci con i quali è possibile realizzare simulazioni basate su agenti, si propone la seguente suddivisione[36]:

a) modelli con agenti 'no-mind' operanti in un ambiente non strutturato;

b) modelli con agenti 'minded' (o cognitivi) operanti in un ambiente non strutturato;

---

[35] Non necessariamente i comportamenti complessi hanno radici complesse; comportamenti complessi possono emergere da insiemi di unità estremamente semplici.

[36] Per agenti 'no mind' si intendono agenti dotati di regole immodificabili e di ridotta percezione dell'ambiente; con agenti cognitivi quelli che possono modificare le regole secondo le quali si comportano (ad es con RNA); per ambiente non strutturato si intende un ambiente senza meccanismi di coordinamento

c) modelli con agenti 'no-mind' operanti in un ambiente strutturato;

d) modelli con agenti 'minded' (o cognitivi) operanti in un ambiente strutturato.

Secondo Axelrod per riprodurre i sistemi e le dinamiche economiche osservate nella realtà non è necessario disporre di agenti particolarmente sofisticati; risultati interessanti possono essere ottenuti con l'impiego di agenti 'no-mind' (impostazioni *a* e *c*).

La ragione principale della validità esplicativa dei modelli 'no-mind' risiede nella considerazione che in realtà quello che interessa è l'analisi delle conseguenze dell'interazione tra gli agenti.

L'economia, partendo dai comportamenti individuali non direttamente oggetto di studio, ricerca le conseguenze che emergono dall'interazione tra individui e i modelli ABM hanno come principale oggetto di studio, proprio, l'analisi delle conseguenze della interazione tra agenti eterogenei.

Il principale 'difetto' dei modelli classici, infatti, non sta tanto nell'ipotesi che gli agenti abbiano capacità di calcolo irrealistiche, ma piuttosto nell'assenza di eterogeneità e interazione tra agenti[37].

In opposizione a tale impostazione, diversi studiosi sostengono che i modelli ABM, per essere validi devono necessariamente essere costruiti con agenti cognitivi, capaci di modificare il proprio comportamento per adattarsi ai mutamenti dell'ambiente circostante e di perseguire obiettivi sviluppati autonomamente; la complessità deve essere, cioè, inclusa negli agenti stessi (impostazioni *b* e *d* citate precedentemente).

Secondo Chattoe (1998) un'implementazione *mindless* di modelli evolutivi conduce, infatti, a spiegazioni implausibili dei processi socioeconomici.

La questione del livello di complessità degli agenti non riceve, dunque, una risposta omogenea dalla comunità dei ricercatori; tuttavia, la scelta sembra propendere per l'uso di agenti ragionevolmente semplici e per la ricerca, nell'interazione e nella presenza di strutture *ex-ante*, dei meccanismi in grado di riprodurre la complessità presente nel mondo reale[38].

1.7 L'ambiente degli agenti: il modello ERA

Una tecnica particolarmente utile per la costruzione dei modelli basati su agenti è lo schema *Enviroment-Rules-Agents* (*ERA*) proposto in Terna (1998a).

---

e, infine, per ambiente strutturato un ambiente in cui operino meccanismi che regolano comportamenti e transazioni (es Borsa).

[37] Come affermato in Leombruni (2000) costruire agenti che risolvono hamiltoniani da zero a infinito in cui, però, tutto il resto del mondo è considerato costante, fatto di agenti tutti uguali e che non interagiscono tra di loro vuol dire costruire agenti abbastanza stupidi; in questo caso, dare ad un agente grandi capacità computazionali vuol dire fargli fare un po' di "algebretta" in più, ma il passo in avanti rispetto ad un agente "stupido" è a misura nulla.

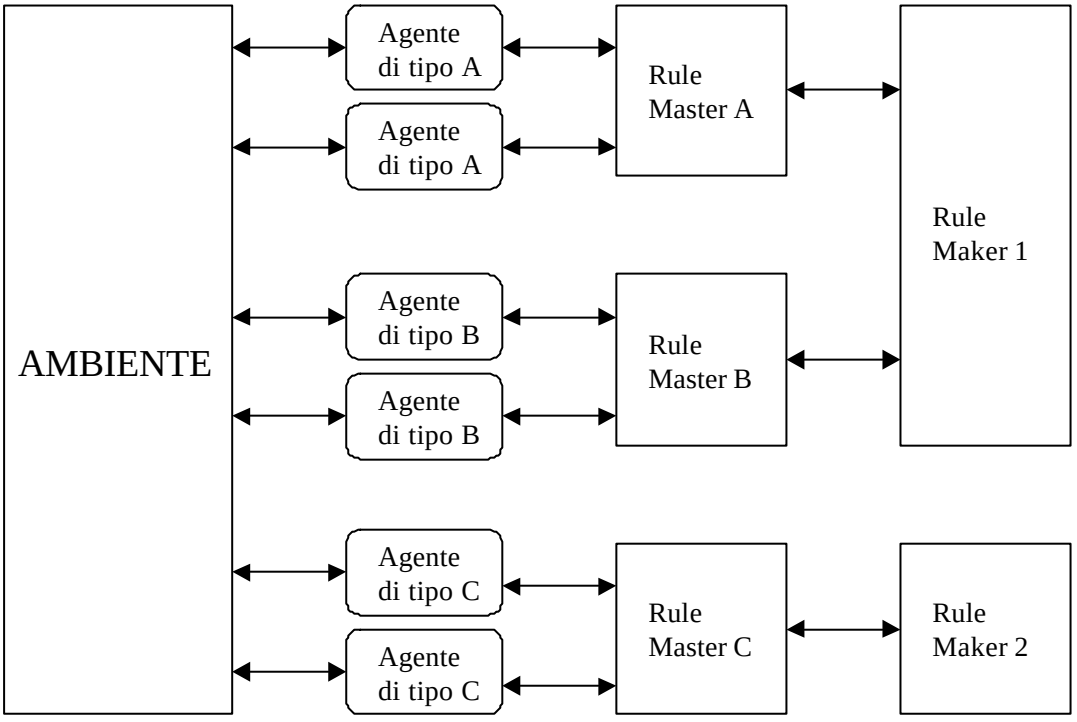
La 'filosofia' di fondo dell'impostazione basata su agenti no-mind è ben sintetizzabile nella domanda: *"Are human agents so far from the complexity of the economic system, as ants are from their anthill?*"

[38] La risposta alla questione sul livello di complessità di un modello dipende anche dal tipo di uso della simulazione; se si vogliono indagare gli effetti della interazione tra agenti si può ricorrere ad agenti semplici; se, invece la simulazione si pone come obiettivo la riproduzione di uno specifico fenomeno, allora occorre prestare particolare attenzione alla fedele descrizione del sistema che si vuole riprodurre; il modello SUM, descritto nel quarto capitolo, segue le impostazioni dei punti *c* e *d*.

Tale modello, il cui scopo è di rendere il più possibile uniforme la struttura dei modelli ABM, fornisce una rappresentazione delle relazioni tra l'ambiente - all'interno del quale gli 'individui' operano ed interagiscono - e gli agenti.

Il principale vantaggio dello schema ERA, riportato nella figura che segue, è che gli agenti e l'ambiente vengono mantenuti su due distinti livelli concettuali.

Come è possibile osservare, con tale impostazione gli agenti non comunicano direttamente tra di loro, ma interagiscono sempre con l'ambiente[39].



Nello schema ERA, il comportamento dei singoli agenti è determinato da un oggetto, il *Rule Master* (gestore di regole), che può essere considerato come la rappresentazione delle capacità cognitive dell'agente. Tali capacità non sono uguali per tutti gli agenti presenti nel modello, di conseguenza vi saranno differenti Rule Master, che rifletteranno le diverse strutture cognitive degli agenti.

In questo modo, si potranno avere agenti 'cognitivi' con Rule Master realizzati con sofisticati meccanismi di calcolo come, ad esempio, le reti neurali e gli algoritmi genetici, e agenti invece con Rule Master meno evoluti e quindi con una struttura 'mentale' più 'limitata'.

Il Rule Master ottiene le informazione necessarie per applicare le regole dall'agente stesso, che dispone di propri dati 'personali', o da oggetti speciali - posizionati ad un livello intermedio fra l'agente ed il gestore di

[39] Ciò risulta particolarmente utile, in fase di realizzazione del modello, quando si deve 'tradurre' il modello ideato in un programma al computer, in quanto permette di semplificare notevolmente la scrittura del codice

regole - che hanno la funzione di raccogliere e distribuire i dati (*Datawarehouse*) e di facilitare la comunicazione con l'ambiente e con gli altri agenti (*Interface*)[40].

Lo schema proposto prevede, inoltre, la possibilità che il comportamento degli agenti, determinato dal Rule Master, possa cambiare nel tempo; nel mondo reale, infatti, gli individui modificano il proprio comportamento sulla base dell'esperienza maturata e delle nuove conoscenze acquisite.

Nel modello ERA, il processo di cambiamento delle regole di comportamento viene realizzato mediante l'utilizzo di un oggetto denominato *Rule Maker* (produttore di regole). Tale dispositivo, ha la funzione di modificare le regole di comportamento determinate dal Rule Master.

Se, ad esempio, il Rule Master di un gruppo di agenti viene realizzato mediante una rete neurale artificiale, il Rule Maker modificherà il comportamento dell'agente sottoponendo, periodicamente, la rete neurale ad un processo di apprendimento.

Sebbene tale impostazione appaia in prima analisi piuttosto complessa, essa offre indubbi vantaggi allo sviluppatore del modello ABM; la rigidità della struttura diventa una fonte di chiarezza in fase di realizzazione del modello.

Come affermato in Terna (1998a) un vantaggio offerto dallo schema ERA è la sua modularità: quando lo sviluppatore del modello vuole modificare la tipologia di agenti deve solamente cambiare le singole componenti, invece di dover ridisegnare l'intera struttura.

La modularità permette, in sostanza, di cambiare il tipo di agente, semplicemente modificando il produttore ed il gestore delle regole.

La struttura dello schema ERA evita, inoltre, la duplicazione delle istruzioni (e la conseguente moltiplicazione delle righe di codice) necessarie allo svolgimento di un'azione e consente ai ricercatori di individuare con facilità la posizione in cui inserire eventuali modifiche volte ad estendere il modello originario.

1.8 Complessità e ABM

Negli ultimi anni un'attenzione particolare è stata rivolta allo studio della teoria della complessità e alla sua applicazione in campo economico[41].

Secondo tale impostazione il sistema socioeconomico è un sistema adattivo complesso[42], composto da un elevato numero di individui che interagendo tra loro localmente, danno luogo a proprietà globali del

---

necessario per realizzare il programma (l'interazione diretta tra agenti determina un aumento del grado di complessità del codice).

[40] L'utilizzo del *datawarehouse* permette a più agenti di condividere lo stesso gestore (e lo stesso produttore di regole) pur mantenendo la totale indipendenza comportamentale gli uni dagli altri; l'*interface*, invece, consente di semplificare il codice che descrive l'agente e le sue azioni, rendendolo al tempo stesso maggiormente indipendente dal tipo di gestore di regole utilizzato; il *Datawarehouse* e *l'Interface* non sono mostrati in figura per non complicare troppo lo schema generale; tali strutture hanno un semplice legame biunivoco con l'agente;.

[41] La Scienza della Complessità è una scienza generale che trova applicazione in numerosi campi scientifici; negli ultimi anni, interessanti applicazioni pratiche sono state sviluppate (principalmente negli Stati Uniti) anche in campo sociale ed economico; i concetti elaborati da tale teoria - che rappresentano una evoluzione delle precedenti idee sviluppate dalla cibernetica, dalla teoria delle catastrofi e dalla teoria del caos - sono

sistema che non sono prevedibili o deducibili anche conoscendo esattamente gli individui e le loro regole di interazione locale.

L'interesse mostrato per il tema della complessità da parte di numerose discipline scientifiche ha reso difficile definirne esattamente il campo di studio[43].

Secondo i ricercatori del Santa Fe Institute - uno dei poli di ricerca all'avanguardia nello studio dei sistemi complessi - la complessità consiste di sei principali caratteristiche:

1) interazioni disperse tra una moltitudine di agenti eterogenei che operano localmente in un determinato ambiente;
2) assenza di controllori globali; le interazioni tra agenti avvengono senza l'intervento di meccanismi centralizzati di controllo e coordinamento;
3) organizzazione gerarchica trasversale: l'organizzazione globale consiste di molti differenti livelli; le unità di ogni livello possono servire come *building blocks* delle unità di livello più alto;
4) adattamento continuo: il comportamento degli agenti si modifica in risposta ai cambiamenti dell'ambiente e in base all'esperienza accumulata;
5) continue 'novità' (*perpetual novelty*): comportamenti e strutture possono stimolare la creazione di nuovi comportamenti e di nuove strutture;
6) dinamica '*out of equilibrium*': dato lo stato di continua evoluzione il sistema non opera vicino a nessun punto di equilibrio ottimale.

Nei sistemi complessi è estremamente difficile individuare rapporti di causa ed effetto tra gli elementi e prevedere come il sistema si evolve nel tempo. Essi, infatti:

- reagiscono in modi imprevedibili alle perturbazioni provenienti dall'esterno: una perturbazione grande può essere riassorbita dal sistema senza apparenti effetti mentre una piccola perturbazione può provocare sensibili variazioni[44];
- sono molto sensibili alle condizioni iniziali: piccole differenze possono condurre a risultati profondamente diversi;

---

stati utilizzati, ad esempio, per spiegare fenomeni di *path dependence*, *lock in* e fenomeni di discontinuità come i *crash* delle bolle speculative o i collassi di interi sistemi economici.

[42] Un sistema adattivo complesso è un sistema composto da numerosi elementi indipendenti (agenti individuali o attori) che hanno la libertà di agire in maniera non totalmente prevedibile e le cui azioni sono interconnesse in modo tale che l'azione di un elemento può influenzare le azioni degli altri elementi; in un sistema di questo tipo, il risultato finale – il comportamento del sistema - emerge dalle interazioni tra gli elementi la cui varietà permette al sistema stesso nella sua totalità, di sviluppare un'auto-organizzazione spontanea (l'autorganizzazione emerge senza che nessuno degli elementi coinvolti nel processo ne sia la guida o sia consapevole del risultato delle sue azioni); tali sistemi sono, inoltre, adattivi, in quanto non si limitano a reagire passivamente agli eventi, ma si modificano e adattano all'ambiente.

[43] Una interessante definizione del termine complesso è stata proposta in Day (1994) : 'A dynamical system is complex if it endogenously does not tend asymptotically to a fixed point, a limit cycle, or an explosion. Such system can exhibit discontinuous behavior and can be described by sets of non linear differential or difference equations, possibly with stochastic elements. But not all such equation systems will generate complexity. The positive exponential function, basis for most growth models, is an example of a noncomplex, non linear system because it just explodes.'

[44] 'Effetto farfalla': secondo tale metafora – che trae spunto dalla teoria del caos - il movimento di una farfalla in una piazza di Pechino può determinare conseguenze apparentemente remote come la nascita di un uragano nel Golfo del Messico.

- non sono indipendenti dal contesto in cui operano;
- sono coinvolti in rapporti di dipendenza reciproca: un elemento del sistema influenza un altro elemento e ne è, a sua volta, influenzato;
- sono adattivi: cioè cambiano in funzione dell'ambiente in cui si trovano e a loro volta modificano tale ambiente;
- il ruolo di ciascun elemento del sistema non è ben identificabile in quanto al sistema non è applicabile il principio di sovrapposizione.

In definitiva, la caratteristica fondamentale di questi sistemi è che le 'proprietà emergenti'[45] sono proprietà delle interazioni tra le diverse parti non proprietà delle parti in se stesse.

Le peculiarità dei sistemi complessi rendono evidente come lo studio dell'economia - come sistema complesso adattivo - non possa essere condotto in modo appropriato con gli strumenti analitici tradizionali, per esempio con sistemi di equazioni; tali strumenti sono, infatti, adatti allo studio dei sistemi lineari[46], ma non si prestano allo studio delle problematiche connesse ai sistemi complessi.

Come affermato in Parisi (2001), la teoria dei sistemi complessi è in un certo senso anch'essa complessa , in quanto riguarda numerosi elementi che vengono fatti interagire tra di loro.

L'analisi di questi sistemi deve, quindi, essere effettuata con nuovi e appropriati strumenti concettuali e metodologici come le simulazioni con modelli basati su agenti; tali simulazioni consentono, infatti - grazie alle elevate capacità di calcolo e di memoria del computer - di tenere in conto l'insieme delle parti e delle loro interazioni (supplendo, così, ai limiti delle capacità cognitive umane) e di sperimentare la non linearità degli effetti aggregati dei comportamenti individuali.

Grazie ai modelli ABM è possibile studiare il comportamento che emerge a livello globale come conseguenza delle numerose interazioni non lineari che hanno luogo a livello locale; tali interazioni sono la base per una varietà estremamente ricca di possibili conseguenze; conseguenze - imprevedibili a priori - che non possono essere analizzate mediante la mera ispezione del modello, ma che necessitano di un metodo di sintesi: un metodo, cioè, che parta dalle componenti del sistema per studiare cosa emerge quando tali componenti vengono messe insieme e fatte interagire.

Come affermato in Waldrop (1995) i ricercatori del Santa Fe Institute sono convinti che l'applicazione delle idee sviluppate dalla complessità consentirà di comprendere le dinamiche spontanee ed autorganizzantisi del mondo in una prospettiva del tutto nuova (con la possibilità di esercitare un impatto immenso sulla conduzione dell'economia, degli affari e della politica); essi, ritengono di disporre degli strumenti matematici e metodologici necessari per costruire un nuovo paradigma di riferimento che costituirà la base per la prima

[45] Come affermato in Terna (1998b) vi sono due differenti tipi di 'emergenza': le emergenze impreviste, che si verificano, ad esempio, quando invece di convergere verso un punto di equilibrio atteso, il sistema mostra un imprevisto comportamento ciclico, e quelle imprevedibili come l'emergere, nel sistema simulato, di dinamiche caotiche.

[46] I sistemi lineari sono sistemi nei quali il comportamento del tutto corrisponde alla somma del comportamento delle sue parti (principio di sovrapposizione); nei sistemi non lineari, invece, il comportamento del tutto è maggiore della somma delle sue singole parti.

alternativa rigorosa al pensiero lineare, che domina la scienza fin dai tempi di Newton, e che ha ormai raggiunto i suoi limiti nell'affrontare i problemi del mondo moderno[47].

BIBLIOGRAFIA

- AXELROD R. (1997), *Advancing the Art of Simulation in the Social Sciences* in CONTE R., HEGSELMANN R. & TERNA P. (eds*)* Simulating Social Phenomena, Berlin: Springer.
- AXTELL R. (2000), *Why agents? On the varied motivations for agent computing in the social sciences*, Center on Social and Economic Dynamics, Working Paper n.17.
- CHATTOE E. (1998), *Just How (Un)realistic are Evolutionary Algorithms as Representation of Social Processes?*, in Journal of Artificial Societies and Social Simulation, vol.1, n.3, http://www.soc.surrey.ac.uk/JASSS/1/3/2.html.
- CONTE R., HEGSELMANN R., TERNA P. (1997), *Social Simulation – A New Disciplinary Synthesis* in CONTE R., HEGSELMANN R., TERNA P. (eds) Simulating Social Phenomena, Berlin: Springer;
- DAY R.H (1994), *Complex Economic Dynamics, Volume1: An Introduction to Dynamical Systems and Market Mechanisms*, Cambridge, MA: MIT Press.
- EPSTEIN J., AXTELL R. (1996), *Growing artificial societies: social science from the bottom up,* Washington DC: Brookings and Cambridge, MA: MIT Press.
- FERRARIS G. (2000), *Algoritmi genetici e classifier system nei modelli di agenti*, relazione convegno IRES.
- GALLANT S. (1993), *Neural Network Learning and Expert Systems,* The MIT Press.
- GOLDBERG D.E (1989), *Genetic algorithms in search, optimization and machine learning*, Reading, Mass, Addison-Wesley.
- HORGAN J. (1997), The End of Science: Facing the Limits of Knowledge in the Twilight of the Scientific Age, Broadway Books.
- HOLLAND J. (1975), *Adaptation in natural and artificial systems*, University of Michigan Press.
- KOZA J.R (1990), *A genetic approach to econometric modeling*, in Bourgine e Walliser (1992).
- LANGTON C. (1992), *Vita Artificiale*, Sistemi Intelligenti, n.2.
- LEOMBRUNI R. (2000)*, Una discussione sulla simulazione in campo sociale: mente e società*, in TERNA P., CONTE R. (2000), Sistemi Intelligenti, n.2, pp.326-337.
- MARGARITA S. (1992), *Verso un "robot oeconomicus": algoritmi genetici ed economia,* Sistemi Intelligenti, n.3, pp. 421-459.
- PARISI D. (2001), *Simulazioni – La realtà rifatta nel computer*, il Mulino, Bologna.
- PARISI D. (1999), *Mente. I nuovi modelli della vita artificiale*, Il Mulino, Bologna.
- PARISI D. (1989), *Intervista sulle reti neurali – cervello e macchine intelligenti*, Il Mulino, Bologna.
- TERNA P. (1998a), *Creare mondi artificiali: una nota su Sugarscape e due commenti*, Sistemi Intelligenti, n.3, pp. 489-496.

[47] Tale visione del ruolo della complessità, tuttavia, non è unanimemente condivisa dalla comunità scientifica; secondo Horgan (1995), infatti, la complessità rappresenta soltanto l'ultimo di una serie di discipline che hanno rappresentato scarsi tentativi di spiegazione dei fenomeni emergenti (la complessità - come i predecessori: cibernetica, teoria delle catastrofi e caos - non riesce, secondo Horgan, a superare lo scoglio della non linearità dei fenomeni).

- TERNA P. (1998b), *Simulation Tools for Social Scientists: Building Agent-Based Models with SWARM, Journal of Artificial Societies and Social Simulation*, vol.1, n.2.
- TERNA P. (2001), Cognitive agents behavings in a simple stock market structure, in corso di pubblicazione.
- TESTFATSION L. (2000), *Introduction to the special issue on agent-based computational economics*, Journal of Economic Dynamics & Control, 25 (2001), pp.281-293.
- WALDROP M. M. (1995), *Complessità, Uomini ed Idee al confine tra Ordine e Caos,* Torino, Instar Libri.
- WOOLDRIDGE M.J, JENNINGS N.R(1995), *Intelligent Agents: Theory and Practice*, Knowledge Engineering Review, vol.10, n.2.

Capitolo 2: Reti Neurali Artificiali

Introduzione

Benché le reti neurali artificiali abbiano una storia relativamente recente, esiste già una vasta letteratura sulla loro costruzione e sul loro funzionamento.

I primi lavori su meccanismi di calcolo computazionale assimilabili concettualmente alle moderne tecniche di apprendimento a rete neurale possono essere fatti risalire a McCulloch W. e Pitts W.(1943), i quali elaborarono i primi modelli formali dell'attività dei neuroni. In seguito contributi rilevanti vennero da Hebb (1949), che formulò il primo algoritmo di apprendimento, Widrow e Hoff.

Nel 1961, Rosemblatt descrisse le proprietà della rete neurale da lui ideata, il *perceptrone*: rete neurale senza nodi nascosti; il suo lavoro, che suscitò notevole interesse tra la comunità dei ricercatori, venne però duramente criticato da Minsky e Papert (1969), due scienziati appartenenti all'area dell'intelligenza artificiale, i quali dimostrarono che le capacità del *perceptrone* erano molto limitate, insufficienti a risolvere un problema semplice, come quello della realizzazione della funzione logica XOR[48]. Le critiche di Minsky e Papert, erroneamente considerate definitive, determinarono una netta caduta dell'interesse per le reti neurali[49].

Negli anni successivi i modelli simbolici divennero il paradigma dominante nel campo dell'intelligenza artificiale e solo un ristretto gruppo di ricercatori, tra i quali Anderson J. e Grossberg S., continuò a sviluppare sistemi basati su tecniche connessionistiche.

Alla metà degli anni '80, però, con il passaggio dal percettore alle reti neurali con unità nascoste e la scoperta - sviluppata in modo indipendente da Rumelhart e McClelland (1986), LeCun e Parker (1986) - del metodo di *backpropagation* dell'errore per l'apprendimento di compiti, anche complessi, da parte delle reti con neuroni nascosti, si assiste al rifiorire dell'attenzione della comunità scientifica verso i modelli neurali.

Negli ultimi anni sono diventate sempre più evidenti le potenzialità dei meccanismi di calcolo parallelo; sono stati ideati e studiati nuovi tipi di reti - che si differenziano nell'architettura, per il tipo di connessioni e di funzione di attivazione - e un numero sempre crescente di ricercatori si è impegnato nell'esplorazione delle

[48] La funzione XOR, dati due input I1 e I2, produce in output i seguenti valori:

I1	I2	Output
0	0	0
0	1	1
1	0	1
1	1	0

[49] In realtà le limitazioni computazionali 'rivelate' dai due studiosi erano riferibili unicamente al particolare meccanismo di apprendimento utilizzato per il *training* dei sistemi a rete neurale.

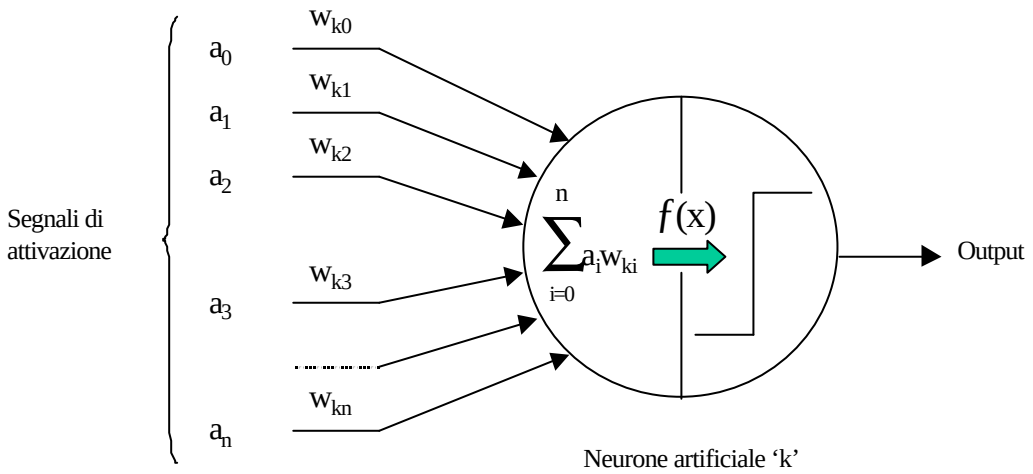
capacità di questi meccanismi di elaborazione, il cui utilizzo è stato esteso dal campo tradizionale della scienza cognitiva al campo delle discipline sociali[50].

2.1 Struttura di una RNA

Una rete neurale artificiale è formata da un certo numero di unità elementari di calcolo; tali unità, denominate 'neuroni' (o nodi), sono collegate tra loro mediante delle connessioni – generalmente unidirezionali - che hanno lo scopo di trasmettere segnali di 'attivazione' o di 'inibizione' da una unità all'altra; ogni elemento della rete ha un certo numero di connessioni in arrivo mediante le quali riceve segnali dalle altre unità e un certo numero di connessioni in uscita, attraverso le quali invia segnali alle altre unità.

Ad ogni connessione tra neuroni è associato un peso; il valore assoluto del peso rappresenta l'intensità della connessione; il segno, invece, corrisponde alla natura del legame: un valore positivo determina un trasmissione di 'attivazione' al neurone che riceve il segnale; un valore negativo, invece, determina la trasmissione di 'inibizione'.

Ogni neurone della rete svolge una funzione molto semplice: effettua una somma "pesata" dei segnali ricevuti in ingresso e utilizza il valore risultante per generare un segnale di uscita sulla base di una determinata funzione, denominata "funzione di attivazione o di trasferimento " (*Transfer Function)*.



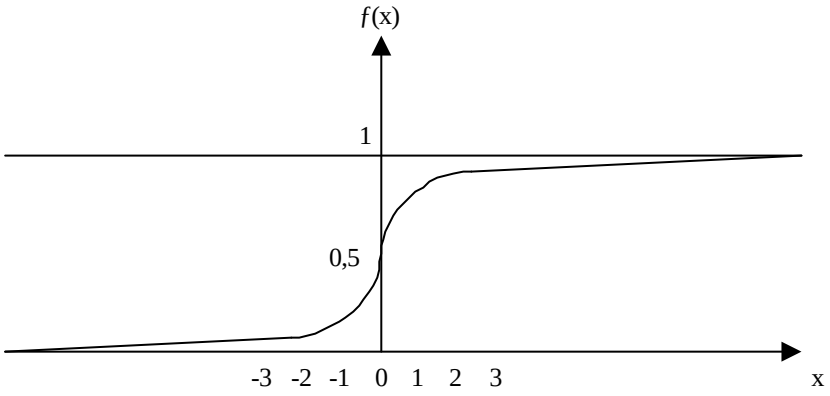
Analogamente ai neuroni naturali, i 'neuroni artificiali' si attivano solo quando i propri input raggiungono una determinata soglia e il livello di attivazione non supera mai un dato livello di saturazione. Di conseguenza la funzione di trasferimento deve presentare due caratteristiche principali: l'esistenza di un valore soglia e la presenza di un limite superiore e di uno inferiore.

[50] Le reti neurali artificiali sono state impiegate in numerosi campi di ricerca: per lo sviluppo di sistemi di controllo di robot o veicoli intelligenti, come aiuto per la diagnosi e la prognosi di patologie, per lo sviluppo di codici compatti e robusti per la compressione di immagini; in economia, modelli neurali sono stati impiegati per l'analisi finanziaria, per la formulazione di modelli economici e la realizzazione di simulazioni.

Numerose sono le funzioni che possono essere utilizzate a tale scopo; in genere, tuttavia, la funzione di attivazione viene approssimata mediante la funzione logistica:

$$f(x) = \frac{1}{1 + e^{-kx}}$$

Tale funzione presenta il vantaggio di assumere valori compresi nel range 0÷1 e di essere differenziabile in ogni punto[51]. Come è possibile notare dal grafico riportato di seguito, per valori di ingresso che si discostino anche di poco dall'ascissa cui corrisponde il flesso, la funzione logistica produce valori di uscita prossimi a 0 o a 1[52].



Funzione logistica

Una volta calcolata la somma pesata dei segnali in ingresso e applicata la funzione di attivazione, l'output risultante determina come il neurone influenza gli altri nodi con cui è collegato da connessioni in partenza.

All'interno della rete i neuroni sono organizzati in diversi strati (*layers*).

Una rete neurale artificiale è costituita normalmente da:

- uno strato input che riceve i segnali dall'esterno; in genere i neuroni appartenenti a questo strato non elaborano i segnali ricevuti, ma li trasmettono ai successivi strati della rete;
- uno o più strati intermedi (hidden) nei quali avvengono le operazioni di elaborazione da parte dei neuroni; il numero di strati nascosti dipende dal tipo di problema e determina la complessità del processo di elaborazione[53];

[51] Ciò, come si vedrà più avanti, consente l'applicazione dell'algoritmo di apprendimento della rete, basato sulla derivate della funzione di attivazione.

[52] il valore k che compare nella funzione, permette di accentuare la pendenza della logistica: più è elevato il valore di k, più la logistica approssima una funzione a soglia.

[53] In genere il numero di nodi hidden in una determinata applicazione è intermedio tra il numero di nodi input e quelli output; la determinazione esatta del numero di unità nascoste è stato trattato nella letteratura, ma

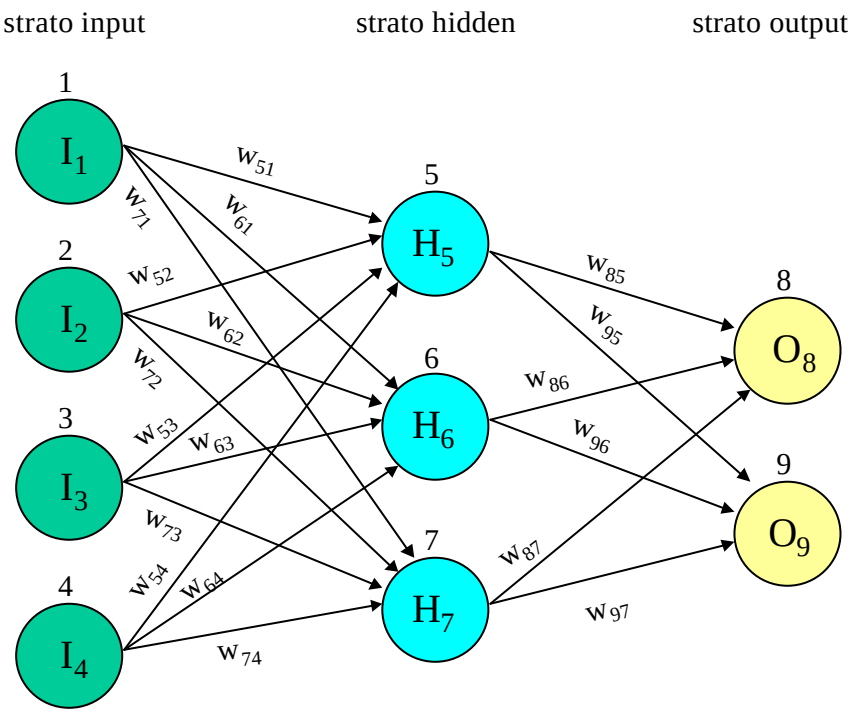
- uno strato output che restituisce uno o più segnali all'esterno[54];

In linea teorica, in una rete neurale ogni unità può essere connessa con qualunque altra unità; nelle reti di Hopfield, ad esempio, ogni neurone è connesso con tutti gli altri attraverso connessioni simmetriche (cioè con due connessioni con identico peso, una in un senso e una nell'altro).

In questo capitolo si fa riferimento alle reti neurali di tipo '*feed-forward*' che sono le più diffuse tra i ricercatori e le più adatte per applicazioni di tipo economico. In questo tipo di rete, i neuroni sono organizzati in strati, caratterizzati dal fatto che i neuroni di uno stesso strato non sono connessi tra loro, ma solo con neuroni appartenenti allo strato successivo: le unità dello strato input sono collegate con quelle dello strato hidden e le unità dello strato hidden sono collegate con quelle dello strato output.

Tali reti presentano, inoltre, una connettività totale tra uno strato e l'altro: tutti i neuroni di uno stato sono connessi con tutti i neuroni dello strato seguente.

Di seguito è riportato un esempio di rete '*feed-forward*':



i neuroni sono raggruppati in tre strati: lo strato input è formato da quattro nodi (1, 2, 3, 4), lo strato intermedio da 3 (5, 6, 7) e lo strato output da due (8, 9)[55].

---

non ha ancora trovato una soluzione standard; nelle applicazioni il problema viene solitamente risolto con un processo di prove e confronti.

[54] I valori di input sono normalmente scalati nell'intervallo 0-1; le grandezze di output, invece, sono in genere limitate tra 0.1 e 0.9; ciò viene fatto per facilitare il processo di apprendimento.

[55] Generalmente, in fase di realizzazione, viene aggiunto un neurone fittizio (*bias*), il cui valore è pari a 1, allo strato input e a quello hidden; la funzione di questo nodo è quella di arricchire i gradi di libertà della funzione di attivazione; come affermato in precedenza, per valori di ingresso che si discostino anche di poco dall'ascissa cui corrisponde il flesso, la funzione logistica produce valori di uscita prossimi a 0 o a 1; i nodi

Come è possibile notare, la struttura della rete è completamente interconnessa: ogni neurone dello strato input è collegato con ogni neurone dello strato hidden e ogni neurone dello strato intermedio è collegato ad ogni neurone dello strato output. Ad ogni connessione è associato un peso, w_{ji}, che rappresenta la forza della connessione che va dal nodo *j* al nodo *i*. Il flusso dei segnali 'scorre' solo in avanti: dallo strato input a quello hidden e da questo allo strato output, senza effetti di retroazione (*feedback*).

Nell'esempio riportato, lo strato input riceve dall'esterno i valori: I_1, I_2, I_3, I_4; tali valori vengono trasmessi mediante le connessioni in partenza ai nodi dello strato hidden. Il segnale complessivo ricevuto dai nodi intermedi è, quindi, pari ad una media pesata - per i valori w_{ji} - dei segnali in partenza dallo strato input.

Per il nodo 5, ad esempio, il segnale complessivo ricevuto in input (net_5) è uguale alla seguente combinazione lineare:

$net_5 = w_{51}\, I_1 + w_{52}\, I_2 + w_{53}\, I_3 + w_{54}\, I_4$;

il neurone 5 applica la funzione di attivazione al segnale ricevuto in ingresso:

$$H_5 = \frac{1}{1 + e^{-net_5}}$$

il valore risultante, H_5, rappresenta lo stato di 'attivazione' del neurone.

Tale valore, insieme con il valore degli altri neuroni dello strato hidden (H_6 e H_7), viene trasmesso ai nodi dello strato output. Il segnale ricevuto dal nodo 8 risulta pertanto pari a:

$net_8 = w_{85}\, H_5 + w_{86}\, H_6 + w_{87}\, H_7$;

il neurone applica, quindi, la funzione di attivazione al segnale in ingresso:

$$O_8 = \frac{1}{1 + e^{-net_8}}$$

il valore ottenuto rappresenta, insieme al valore dell'altro nodo output (O_9), il risultato finale della rete neurale. L'output della rete neurale dipende interamente dai pesi che stanno sulle connessioni tra i diversi neuroni; i pesi, in altre parole, determinano il comportamento del modello neurale (e poiché sono i soli elementi capaci di memorizzare informazione, sono i responsabili della capacità di apprendimento della rete).

Per descrivere, in maniera concisa, l'intero processo di calcolo della rete può essere utilmente utilizzata la notazione matriciale. Indicando, infatti, con:

- A, la matrice dei pesi che legano i nodi input con quelli hidden;
- B, la matrice dei pesi che legano i nodi hidden con quelli output;

$$A = \begin{bmatrix} w_{51} & w_{61} & w_{71} \\ w_{52} & w_{62} & w_{72} \\ w_{53} & w_{63} & w_{73} \end{bmatrix} \qquad B = \begin{bmatrix} w_{85} & w_{95} \\ w_{86} & w_{96} \\ w_{87} & w_{97} \end{bmatrix}$$

bias consenteno di spostare il punto di ascissa in corrispondenza del quale vi è il flesso; in altre parole, consentono di spostare il valore soglia della funzione.

w_{54} w_{64} w_{74}

- I, il vettore degli input;
- O, il vettore degli output della rete;
- f, la funzione di attivazione dei neuroni;

il risultato della rete neurale può essere facilmente espresso come:

$O = f (f (I A) B)$[56]

2.2 Apprendimento (*Learning*)

Lo scopo del processo di apprendimento di una rete neurale è quello di determinare il valore dei pesi delle connessioni tra neuroni, w_{ji}, in modo che la rete produca l'output desiderato dato un certo insieme di input ricevuti dall'esterno. La scelta dei valori w_{ji}, che consente l'unione tra input esterno e output desiderato presenta, tuttavia, diverse difficoltà: il peso di una connessione, infatti, concorre a determinare il valore di attivazione di un neurone appartenente allo strato output insieme ai pesi delle altre connessioni e allo stato di attivazione dei nodi nascosti; gli stati di attivazione dei neuroni nascosti, a loro volta, dipendono dal numero di connessioni provenienti dallo strato input. Dato l'elevato grado di 'intercorrelazione' tra i nodi risulta, dunque, estremamente difficile determinare con precisione, senza un opportuno meccanismo di calcolo, il valore dei pesi da attribuire alle specifiche connessioni.

In genere, le reti neurali artificiali vengono inizializzate casualmente: i pesi delle connessioni tra i diversi neuroni vengono scelti a caso; successivamente attraverso specifici metodi computazionali si procede automaticamente alla variazione dei pesi fino ad assegnare loro i valori che consentono di rispondere in modo desiderato ad un dato insieme di input provenienti dall'esterno. Poiché la prestazione della rete dipende interamente dai pesi delle connessioni, all'inizio la rete fornirà risposte casuali agli stimoli esterni; in seguito, però, esposta ad esperienze ripetute, la rete modificherà progressivamente i suoi pesi in modo da produrre la performance desiderata.

Esistono numerosi metodi di apprendimento per le reti neurali, che è possibile suddividere in due categorie distinte:

- metodi con apprendimento supervisionato;
- metodi con apprendimento non supervisionato.

Nel primo caso, l'apprendimento avviene confrontando l'output generato dalla rete con l'output desiderato (target) fornito dall'esterno; con questa tecnica di apprendimento, la rete modifica i pesi tra i diversi nodi 'facendo tesoro' delle informazioni ricevute dall'esterno.

Nel secondo caso, invece, la rete impara scoprendo regolarità negli input senza che le siano fornite informazioni sull'output desiderato dall'esterno; la rete neurale, in sostanza, configura se stessa in modo tale

che ogni neurone a livello output risponda in modo coerente a segnali di ingresso simili tra loro o in qualche modo correlati.

Di seguito è illustrato il meccanismo standard utilizzato per il *learning* delle reti neurali artificiali: l'algoritmo di *backpropagation* dell'errore basato sull'apprendimento supervisionato.

2.2.1 Algoritmo di *backpropagation* dell'errore (BP)

Con tale metodo, la rete neurale artificiale viene fatta apprendere su un insieme di esempi (*training set*); alla rete viene presentato non solo l'insieme degli input di attivazione, ma anche l'insieme degli output desiderati (target).

All'inizio del processo di calcolo, l'output prodotto dalla rete neurale, data l'assegnazione casuale dei pesi iniziali delle connessioni, si discosta molto dall'output desiderato; con il metodo della *backpropagation*, per ogni unità di output si procede al calcolo dell'errore tra l'output prodotto dalla rete e il target. Tale errore viene, quindi, utilizzato per modificare i pesi delle connessioni in modo da migliorare le prestazioni della rete.

Prima della definizione dell'algoritmo di *backpropagation*, l'ostacolo posto allo sviluppo di reti neurali con unità nascoste era costituito dalla difficoltà di determinare i pesi delle connessioni tra unità hidden e unità input; mentre, infatti, i pesi delle connessioni tra unità appartenenti allo strato output e quelle hidden potevano essere modificati sulla base dell'errore tra output e target, ciò non poteva essere effettuato per i pesi tra unità input e unità hidden in quanto per queste ultime non si disponeva di un target desiderato sulla base del quale calcolare l'errore.

Il metodo della *backpropagation* ha consentito di superare tale ostacolo e di stimare il contributo complessivo di ciascun neurone - compresi quelli appartenenti allo strato hidden - agli errori delle diverse unità di output; sulla base di queste stime l'algoritmo di BP modifica i pesi delle connessioni tra i vari strati della rete.

Più precisamente, l'algoritmo di *backpropagation*, modifica i pesi delle connessioni in modo da minimizzare la seguente funzione[57]:

$$E(W) = \frac{1}{2} \sum_{t=1}^{T} \sum_{j=1}^{J} (T_{tj} - O_{tj})^2$$

dove: J è il numero di nodi output, W è il vettore dei pesi, w_{ji}, T è il numero di input di attivazione della rete in fase di apprendimento, O_{tj} è l'output *j* generato dalla rete in corrispondenza dell'input di attivazione *t* e T_{tj} è il target *j* della rete per l'input di attivazione *t*.

Il processo di apprendimento si articola, nel dettaglio, nelle seguenti fasi:

1) il vettore dei pesi delle connessioni viene inizializzato casualmente; generalmente viene utilizzata a

[56] Una rete neurale può essere definita come una funzione vettoriale di vettori; i nodi di input corrispondono ai valori del vettore I, i valori dello strato hidden corrispondono alle componenti del vettore f (IA), i valori di output della rete corrispondono alle componenti del vettore f (f(IA)B).

[57] Si fa uso della formalizzazione matematica adottata da Beltratti e altri (1996)

tale scopo una distribuzione uniforme con range compreso tra -0.5 e +0.5;

2) un insieme *t* di 'esempi' viene presentato alla rete; i segnali di input forniti dall'esterno vengono propagati nella rete nel seguente modo:

- l'input esterno I_t viene trasmesso allo strato hidden:

$$net_{th} = w_{hO} + w_{h1} I_{t1} + w_{h2} I_{t2} + ... + w_{hK} I_{tK}; \qquad h = 1, 2, ..., H;$$

- la funzione logistica viene applicata ai neuroni:

$$H_{th} = \frac{1}{1 + e^{- net_{th}}}; \qquad h = 1, 2, ..., H;$$

- l'output dello strato hidden viene trasmesso ai neuroni dello strato output, i quali applicano a loro volta la funzione di attivazione alla combinazione dei segnali in ingresso:

$$net_{tj} = w_{j0} + w_{j1} H_{t1} + w_{j2} H_{t2} + ... + w_{jH} H_{tH}; \qquad j = 1, 2, ..., J;$$

$$O_{tj} = \frac{1}{1 + e^{- net_{tj}}}; \qquad j = 1, 2, ..., J;$$

3) vengono calcolati, per ogni pattern *t*, gli errori sugli output (J) mediante il confronto tra il target fornito dall'esterno e il risultato prodotto dalla rete;

$$E_t(W) = \frac{1}{2} \sum_{t=1}^{T} (T_{tj} - O_{tj})^2$$

gli errori E(W) vengono, quindi, utilizzati per modificare i pesi delle connessione;

4) i pesi vengono modificati secondo la regola:

$$\Delta_t w_{ji} = - \alpha \frac{\partial E_t(W)}{\partial w_{ji}} + \beta \Delta_{t-1} w_{ji}$$

il parametro α, denominato *learning rate* (tasso di apprendimento), regola la velocità di apprendimento della rete: quanto deve essere grande, cioè, la variazione del peso di una connessione dato un certo errore; in genere si preferisce fare cambiamenti relativamente piccoli in modo da garantire un apprendimento graduale e senza sbalzi. Il parametro β, denominato *momentum*, permette, invece, di introdurre un certo grado di 'persistenza' nel processo di cambiamento dei pesi, legando la variazione del peso con quella effettuata la volta precedente; β, in

sostanza, indica se e quanto il cambiamento dei pesi al tempo *t* deve essere influenzato dai cambiamenti introdotti sullo stesso peso al tempo *t-1*[58].

Quando si procede al calcolo delle variazioni dei pesi delle connessioni tra nodi output e nodi hidden, il valore della derivata riportata nella formula precedente può essere espressa come:

$$\frac{\partial E_t(W)}{\partial w_{ji}} = \frac{\partial E_t(W)}{\partial O_{tj}} \frac{\partial O_{tj}}{\partial w_{ji}};$$

con i = 0, 1, ..., H e j = 1, 2, ..., J.

Con semplici passaggi matematici si ottiene:

$$\frac{\partial E_t(W)}{\partial O_{tj}} = -(T_{tj} - O_{tj});$$

$$\frac{\partial O_{tj}}{\partial w_{ji}} = \frac{\partial O_{tj}}{\partial net_{tj}} \frac{\partial net_{tj}}{\partial w_{ji}} = O_{tj}(1- O_{tj}) H_{ti}$$ [59];

la variazione dei pesi delle connessioni tra nodi nascosti e nodi output può, quindi, essere descritta dalla seguente formula:

$$\Delta_t w_{ji} = \alpha (T_{tj} - O_{tj}) O_{tj} (1- O_{tj}) H_{ti} + \beta \Delta_{t-1} w_{ji}$$

Quando, invece, si procede al calcolo delle variazioni dei pesi delle connessioni tra nodi input e nodi hidden, l'errore viene calcolato rispetto alla seguente derivata:

$$\frac{\partial E_t(W)}{\partial w_{ji}} = \frac{\partial E_t(W)}{\partial net_{tj}} \frac{\partial net_{tj}}{\partial w_{ji}};$$

con i = 0, 1, ..., K[60] e j = 1, 2, ..., H.

in questo caso, risolvendo si ottiene:

$$\frac{\partial E_t(W)}{\partial net_{tj}} = H_{ti}(1 - H_{ti});$$

[58] Non esistono regole per la determinazione dei valori ottimali del *learning rate* e del *momentun*; il valore di questi parametri deve essere determinato empiricamente; in genere si utilizza un *learning rate* pari a 0,6 e un *momentum* pari a 0,9.

[59] La derivata della funzione logistica f(x) è uguale a f(x) (1 - f(x)).

[60] nel processo di apprendimento viene preso in considerazione anche il nodo bias; di conseguenza anche i valori soglia dei neuroni dello strato hidden e dello strato sono soggetti ad apprendimento.

$$\frac{\partial net_{tj}}{\partial w_{ji}} = I_{tj} \sum_{k=1}^{J} (T_{tk} - O_{tk})\, O_{tk}\, (1- O_{tk})\, H_{tk}\, w_{jk}$$

5) il processo di calcolo degli errori e delle variazioni dei pesi delle connessioni tra i diversi nodi viene effettuato per tutti gli input di attivazione forniti alla rete per l'apprendimento[61];

6) l'intero processo viene quindi ripetuto per un certo numero di epoche[62].

Il meccanismo di *backpropagation* dell'errore permette di modificare i pesi delle connessioni tra i diversi nodi, in modo da diminuire l'errore complessivo e da evitare che la rete cada in 'minimi locali', cioè in configurazioni di pesi che non corrispondono all'errore minimo globale ricercato.

Una volta terminato il processo di apprendimento, la rete viene 'verificata' su un insieme di dati di controllo (*validation set*) e quindi applicata a dati nuovi.

La valutazione della performance della rete, che è basata su indicatori statistici come il coefficiente di determinazione, viene effettuata, in genere, sia sulla base dei dati del *training set* che sui pattern di controllo del *verification set*. Il primo tipo di valutazione riflette la capacità di apprendimento della rete; il secondo, invece, la capacità di generalizzazione.

2.2.2 Alcune considerazioni

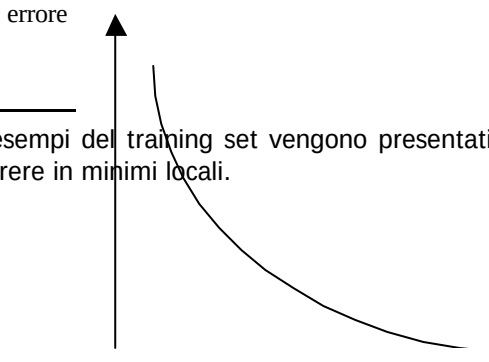
Alcuni aspetti del processo di apprendimento meritano una breve analisi. Innanzitutto occorre notare che, ogni rete ha una propria individualità: se si ripete il processo di apprendimento su un'altra rete non si ottiene lo stesso risultato,

ma solo un risultato simile; due reti che sono state 'allenate' sugli stessi dati, producono, cioè, diversi valori dei pesi delle connessioni tra i neuroni.

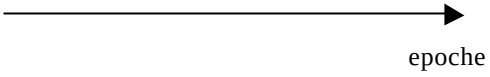
Tale variabilità deriva dal fatto che ogni rete riceve in fase di inizializzazione una specifica assegnazione casuale dei pesi; di conseguenza, dato che il processo di apprendimento varia da rete a rete, non ci si può aspettare un risultato identico; da ciò discende un'altra importante considerazione: le reti sono in grado di dare una stessa prestazione con una varietà di assetti interni dei pesi; ciò che conta, in sostanza, non è il valore di un certo peso, ma l'insieme complessivo dei pesi di tutte le connessioni.

Un altro aspetto che va notato è il decorso dell'apprendimento.

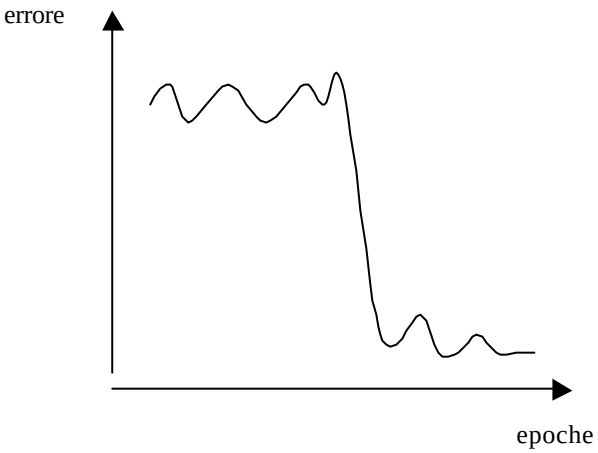
In genere, le reti mostrano un andamento regolare: nelle prime fasi del *learning*, l'errore scende abbastanza velocemente da un'epoca all'altra; in seguito, la velocità di riduzione dell'errore di riduce fin quando si arresta e l'errore si assesta su valore costante.



[61] Normalmente i diversi esempi del training set vengono presentati alla rete non sempre nello stesso ordine in modo da evitare di incorrere in minimi locali.



In altri casi, tuttavia, si può assistere ad una dinamica differente; si può verificare, ad esempio, che la curva di riduzione dell'errore si mantenga su livelli piuttosto elevati e all'improvviso inizi a scendere.



La dinamica dell'errore dipende, infatti, da diversi fattori tra i quali: l'architettura della rete, il *learning rate* e il *momentum*. Se si aumenta il tasso di apprendimento la rete apprende più velocemente; tuttavia, incrementi eccessivi possono determinare andamenti con oscillazioni e improvvise diminuzioni dell'errore; un valore troppo elevato del *learning rate* può, anche, determinare un rallentamento nel processo di apprendimento o, addirittura, un 'arresto'; la rete 'cade' in un minimo locale da cui non riesce più ad 'uscire'.

2.3 Caratteristiche delle RNA

Una volta terminato l'apprendimento la rete, come affermato da Parisi D. (1989), mostra una elevata capacità di generalizzazione; riesce, cioè, ad andare oltre ciò che gli è stato esplicitamente insegnato per fornire risposte 'coerenti' a stimoli nuovi provenienti dall'esterno.
Tale proprietà deriva dalle seguenti caratteristiche delle reti neurali artificiali:

[62] Un'epoca è pari a un ciclo di calcolo sull'intera serie di pattern fornita dall'esterno.

a) Parallelismo: il processo computazionale viene ripartito tra i neuroni dei diversi strati che operano simultaneamente; ciò assicura una elevata rapidità di reazione agli stimoli provenienti dall'esterno (i singoli neuroni sono strutture relativamente lente; tuttavia, quando vengono fatte operare simultaneamente si ottengono importanti risultati in termini di velocità di esecuzione);
b) Rappresentazione subsimbolica della conoscenza: una rete neurale con unità nascoste è in grado di costruirsi una rappresentazione interna della conoscenza. Quando la rete riceve un input dall'esterno, viene attivato uno specifico pattern di attivazione dei nodi nascosti che determina un valore quantitativo per ciascuna unità hidden: il pattern di attivazione sui nodi hidden costituisce la rappresentazione interna della conoscenza. Tale rappresentazione non è simbolica; non fa, cioè, uso di simboli come avviene con il paradigma cognitivista; si tratta, invece, di una rappresentazione quantitativa (il pattern di attivazione delle unità nascoste è un vettore di valori numerici) e distribuita (è l'intero insieme di valori delle unità nascoste, e non un singolo valore, che rappresenta la conoscenza);
c) autorganizzazione: le reti neurali apprendono spontaneamente le proprie rappresentazioni interne; il pattern di attivazione delle unità nascoste dipende dai pesi delle connessioni tra nodi input e nodi hidden che vengono appresi dalla rete mediante l'algoritmo di BP; la rappresentazione interna della conoscenza avviene in modo completamente autonomo: le reti, in sostanza, non devono essere programmate dall'esterno a dare una certa risposta ma sono in grado di imparare da sole ad autorganizzarsi per dare la risposta desiderata (durante il processo di apprendimento la rete effettua una ricerca di 'modelli' cerca, cioè, di scoprire caratteristiche comuni degli input ricevuti);
d) Robustezza all'errore: nei sistemi connessionistici, come le reti neurali, il mancato funzionamento di alcuni neuroni non determina il blocco totale della rete, ma solo un deterioramento più o meno sensibile delle performance; ciò è dovuto al fatto che l'informazione è memorizzata, all'interno della rete, nei pesi delle diverse unità ciascuna delle quali concorre, solo in parte, all'operatività della rete.

Grazie a queste caratteristiche, le reti neurali artificiali hanno una tendenza naturale a estrapolare quello che hanno appreso; ed è proprio tale capacità di rispondere sensatamente a nuovi stimoli provenienti dall'esterno che rappresenta uno dei principali vantaggi dei sistemi a rete neurale rispetto ai sistemi simbolici.

2.4 Interpretazione dei parametri di una rete neurale

La rappresentazione subsimbolica e distribuita della conoscenza, se da un lato rappresenta una delle caratteristiche fondamentali del connessionismo, dall'altro rende più difficile leggere la struttura interna di una RNA al fine di determinarne le regole di comportamento. Per i ricercatori che fanno uso di tecniche a rete neurale uno dei problemi principali è quello di riuscire a capire le rappresentazioni interne e l'autorganizzazione che emerge nelle reti: la natura quantitativa dei modelli, l'intrico delle connessioni e gli effetti di propagazione dei segnali sono, infatti, fattori che rendono difficile una chiara comprensione dei meccanismi e dei principi sui quali si fonda il funzionamento di una rete.

Allo stato attuale della ricerca, non si dispone ancora di sufficienti strumenti di analisi per lo studio dell'organizzazione interna di una rete neurale. Lo sviluppo di questi strumenti è uno dei compiti più importanti per l'evoluzione del connessionismo. Di seguito vengono brevemente illustrate alcune delle tecniche attualmente utilizzabili per l'analisi della struttura interna di una rete neurale, utili per 'fare luce' su alcuni meccanismi base che ne guidano il comportamento.

2.4.1 Nodi nascosti e *cluster analysis*

Quando le reti neurali hanno una struttura relativamente semplice – come nel caso di una rete che realizza la funzione XOR - può risultare utile, per comprendere i meccanismi di funzionamento, fare un'analisi dei valori assunti dai nodi nascosti e dai relativi pesi. In alcuni casi è possibile attribuire uno specifico significato ai nodi e comprendere in modo agevole la struttura interna della rete[63]. Tale possibilità, tuttavia, diventa meno probabile quando la complessità del problema da affrontare implica la costruzione di una RNA con una struttura più articolata. In questi casi le regole che è possibile astrarre esaminando i pesi delle connessioni tra nodi input e quelli hidden e tra questi e i nodi output possono risultare estremamente complesse; si può, allora, ricorrere alla *cluster analysis*.

Mediante *la cluster analysis* è possibile valutare quantitativamente, al variare degli input di ingresso della rete, il grado di 'somiglianza' dei pattern di attivazione dei nodi nascosti al fine di individuare eventuali caratteristiche comuni.

I risultati ottenuti, raggruppando i vettori di attivazione dei nodi hidden in gruppi omogenei, possono aiutare il ricercatore a comprendere i meccanismi che operano all'interno della RNA e che ne determinano la prestazione.

La *cluster analysis* permette, infatti, di rivelare l'organizzazione interna della rete che risulta decisiva nella spiegazione della capacità di generalizzazione.

2.4.2 Tecnica del "lesionamento"

La tecnica del "lesionamento " consiste nel manipolare una rete neurale al fine di verificare le ipotesi sul ruolo di specifiche componenti – neuroni e pesi – nel produrre una determinata prestazione.

Una volta formulate le ipotesi sul ruolo di certe unità nella determinazione del risultato della rete, si lesionano tali unità e si verificano le conseguenze sulla performance.

Una rete può essere lesionata in diversi modi. E' possibile, ad esempio, introdurre segnali di disturbo – rumore – su tutti i pesi della rete o su di uno specifico sottoinsieme; si possono alterare i valori dei pesi di alcune connessioni o ancora è

[63] Anche se il problema è relativamente semplice, non necessariamente si sviluppa una struttura interna della rete facile da comprendere.

possibile 'eliminare' alcuni neuroni, azzerando i pesi su tutte le sue connessioni in uscita. Gli effetti di queste 'lesioni' dipendono da diversi fattori, tra i quali la tipologia della rete e la rappresentazione interna della conoscenza.

Una rete con numerosi neuroni nascosti, in linea teorica, risente meno dell'eliminazione di un neurone rispetto ad una rete con pochi nodi hidden. Tuttavia, ciò può risultare vero solo in parte. Molto, infatti, dipende dal tipo di rappresentazione interna della conoscenza e dal grado in cui questa è distribuita all'interno della rete; a parità di neuroni nascosti vi possono essere reti con una rappresentazione interna concentrata su pochi nodi altamente 'attivati' e reti con rappresentazioni distribuite su un numero più ampio di unità. Nel primo caso, l'eliminazione di un nodo altamente

attivato determina effetti più gravi sul comportamento della rete rispetto all'eliminazione di un nodo di una rete con una rappresentazione della conoscenza distribuita uniformemente.

2.4.3 Analisi delle derivate tra output e input della funzione a rete neurale

Un aiuto alla comprensione della struttura interna della rete può venire dall'analisi delle derivate parziali di ogni elemento di output rispetto ad ogni segnale di ingresso.

Tali derivate possono essere espresse, in modo formale, come segue:

$$\frac{\partial O_j}{\partial I_n} = O_j (1- O_j) \sum_{h=1}^{H} \left[w_{jh} \; H_h (1 - H_h) \; w_{hn} \right]$$

dove O_j rappresenta l'output j, I_n l'input n, H il numero di nodi nascosti, w_{jh} è il peso che collega il nodo nascosto h al nodo di output j, H_h è l'attivazione del nodo hidden h e w_{hn} è il peso che collega il nodo input n con il nodo hidden h.

I valori assunti dalle derivate parziali dell'output in corrispondenza dei diversi valori di input, possono aiutare il ricercatore a scoprire interessanti relazioni causali tra le variabili del modello e a far emergere i meccanismi di comportamento della rete.

Poichè il range dei valori di input è compreso tra 0 e 1, per calcolare il valore delle derivate parziali, in genere, viene utilizzato il valore convenzionale di input pari a 0.5 (input medio); inoltre, per evitare i problemi che sorgono quando ci si imbatte in derivate parziali nulle per singoli valori di input, le derivate parziali vengono calcolate per tutti i pattern di attivazione disponibili; si procede, quindi, al calcolo della media e dello scarto quadratico medio e, infine, allo studio della distribuzione dei valori ottenuti.

2.4.4 Regole empiriche

Una metodologia alternativa per estrarre regole di comportamento da una RNA è quella seguita da Gallant (1993).

L'obiettivo di questa impostazione è quello di individuare regole che:

- si fondino sul minor numero possibile di variabili esplicative;
- siano valide indipendentemente dai valori assunti dalle altre variabili escluse dalle regole;
- non contengano variabili, operatori logici e aritmetici superflui.

L'idea è di costruire le regole in modo addittivo, aggiungendo - sulla base di determinati criteri di valutazione - variabili sino a che la regola divenga valida.

Di seguito sono illustrate, con riferimento ad un esempio tratto da Beltratti ed altri (1996), alcune semplici regole empiriche che possono essere utilmente impiegate a tale proposito.

Si prenda in considerazione un campione artificiale di 100 pattern, con tre valori di input – I_1, I_2, I_3 distribuiti uniformemente nell'intervallo 0÷1 - e tre valori di output costruiti nel modo seguente:

$T_A = 2I_1 + I_2 - 3I_3$;

$T_B = -I_2 + I_3$;

$T_C = I_2 * I_3$;

tale casistica viene fatta apprendere – con l'algoritmo di *backpropagation* - ad una rete neurale con due nodi nascosti.

L'obiettivo dell'esempio è quello di verificare se vi è la possibilità di estrarre le regole che determinano il comportamento della rete - note a priori - mediante l'analisi della sua struttura.

Dopo un apprendimento di 200 epoche, con pattern esaminati in ordine casuale, la rete evidenzia i seguenti risultati[64]:

- errore di BP = 0,006141;
- errore percentuale = 32.68 %;
- i seguenti coefficienti di correlazione lineare tra gli output prodotti dalla rete(O_A,O_B,O_C) e i target:

  tra O_A e T_A, r = 0.915 (R^2 = 0.803);

  tra O_B e T_B, r = 0,960 (R^2 = 0.868);

  tra O_C e T_C, r = 0,979 (R^2 = 0.897);

con solo due nodi hidden la rete mostra un elevato errore percentuale; inoltre, l'esame delle derivate parziali tra output ed input permette di individuare una sola chiara relazione causale; l'indipendenza dell'output Oc dal valore dell'input I_1: la relativa derivata è infatti prossima a zero (-0,026).

∂O/∂I	I_1	I_2	I_3
O_A	0,188	0,301	0,515
O_B	-0,212	-0,476	0,483
O_C	-0,026	0,658	0,594

[64] L'errore percentuale è una valutazione grezza dell'errore ottenuta dividendo la somma dei valori assoluti delle differenze tra output della rete e target con la somma dei target; *r* è il coefficiente di correlazione lineare e R^2 è il coefficiente di determinazione (il rapporto tra la varianza spiegata e la varianza totale).

Se si ripete l'apprendimento con una rete con tre nodi hidden si ottengono i seguenti risultati:

- errore di BP = 0,001061;
- errore percentuale = 11,33 %;
- i seguenti coefficienti di correlazione lineare tra gli output prodotti dalla rete(O_A,O_B,O_C) e i target:
  tra O_A e T_A, r = 0.995 (R^2 = 0.939);
  tra O_B e T_B, r = 0,993 (R^2 = 0.940);
  tra O_C e T_C, r = 0,987 (R^2 = 0.876);

in questo caso si assiste ad una sensibile diminuzione dell'errore percentuale; l'esame delle derivate parziali mostra, inoltre, una maggiore capacità di distinguere tra derivate prossime a zero e derivate non nulle; oltre alla relazione causale tra O_C e I_1 emergono chiaramente anche altre relazioni come, ad esempio, il rapporto di indipendenza tra O_B e I_1, e l'influenza positiva dell'input I_3 (e quella negativa dell'input I_2) su O_B.

∂O/∂I	I_1	I_2	I_3
O_A	0,465	0,214	-0,464
O_B	-0,031	-0,500	0,471
O_C	-0,026	0,682	0,608

Aumentando il numero di nodi nascosti, l'errore diminuisce ulteriormente e le derivate diventano ancora più esplicite facilitando l'individuazione dei rapporti causali tra output e input.

Con 5 nodi hidden si hanno, infatti, i seguenti risultati:

- errore di BP = 0,000216;
- errore percentuale = 4,87 %;
- i seguenti coefficienti di correlazione lineare tra gli output prodotti dalla rete(O_A,O_B,O_C) e i target:
  tra O_A e T_A, r = 0.997 (R^2 = 0.953);
  tra O_B e T_B, r = 0,999 (R^2 = 0.959);
  tra O_C e T_C, r = 0,997 (R^2 = 0.910);

∂O/∂I	I_1	I_2	I_3
O_A	0,460	0,165	-0,402
O_B	-0,001	-0,447	0,432
O_C	-0,009	0,637	0,557

Tuttavia all'aumentare del numero di nodi hidden, si verifica anche un aumento della complessità strutturale della rete; una regola empirica utile da seguire nella costruzione di RNA è quella di incrementare il numero di unità nascoste fino ad ottenere una buona stabilizzazione dei valori delle derivate parziali; nell'esempio riportato la rete con 3 nodi nascosti, rappresenta un buon compromesso tra complessità della rete e chiarezza 'esplicativa' delle derivate.

Al fine di comprendere la struttura interna ed il comportamento della rete con 3 nodi hidden si possono analizzare, inoltre, i valori assunti dalle due matrici dei pesi: la matrice che collega i nodi input con quelli hidden e quella che collega i nodi hidden con quelli output[65].

	H_4	H_5	H_6
I_0	0	0	0,8
I_1	-0,6	1,7	-0,3
I_2	1,3	0	-3,1
I_3	-2,7	-1,5	0,6

	O_A	O_B	O_C
H_0	-1,7	0,4	3,7
H_4	0	-2,8	-5,1
H_5	4,2	-0,6	-2
H_6	-1,5	2	-5,5

Dall'analisi dei valori delle due matrici risulta, tuttavia, estremamente difficile ricostruire i legami tra i segnali di input e l'output della rete: la struttura dei legami appare, infatti, alquanto complessa a causa della rappresentazione subsimbolica e distribuita della conoscenza all'interno della rete e tale da non permette una semplice individuazione delle relazioni causali tra le diverse grandezze.

Una tecnica per superare tale difficoltà consiste nel moltiplicare le due matrici dei pesi (senza i nodi *bias*); il risultato è una matrice – denominata *general weight matrix* (GWM)[66] - con numero di righe pari al numero di nodi input e numero di colonne pari al numero di nodi output. La matrice GWM, il cui significato è puramente qualitativo, viene quindi utilizzata per la creazione, in modo addittivo, delle regole.

Nell'esempio riportato, per la rete con 3 nodi hidden, si ha:

	O_A	O_B	O_C
I_1	7,5	-0,1	1,4
I_2	3,9	-9,6	11,0
I_3	-7,6	9,8	13,7

Al fine di determinare le regole della rete, si procede nel seguente modo: per 'spiegare' l'output O_A si fa operare la rete neurale, precedentemente addestrata, con i soli valori di I_3[67]; poi con I_3 e I_1 e infine con tutte e tre le variabili. L'ordine in cui è scelto l'input dipende dal valore assoluto dei pesi contenuti nella matrice GWM: l'input con peso più elevato viene attivato per prima e a seguire vengono attivati gli altri.

I risultati ottenuti per i diversi nodi output sono:

$O_A(I_3) \rightarrow R^2 = 0{,}434$ $O_B(I_3) \rightarrow R^2 = 0{,}492$ $O_C(I_3) \rightarrow R^2 = 0{,}094$

$O_A(I_3, I_1) \rightarrow R^2 = 0{,}824$ $O_B(I_3, I_2) \rightarrow R^2 = 0{,}939$ $O_C(I_3, I_2) \rightarrow R^2 = 0{,}880$

$O_A(I_3, I_1, I_2) \rightarrow R^2 = 0{,}939$

[65] I pesi compresi tra +0,3 e –0,3 sono stati posti pari a 0.

[66] Beltratti ed altri (1996), pp.100-104.

[67] I_2 e I_1 vengono posti pari al valore costante medio tra il minimo e il massimo presentato alla rete nel training set utilizzato per l'apprendimento.

Dall'analisi dei risultati emergono chiaramente le relazioni causali tra output e input riscontrate in precedenza. La matrice GWM, può quindi essere utilizzata, congiuntamente all'analisi delle derivate parziali, quale utile strumento per la comprensione del comportamento di una rete neurale artificiale.

In questo esempio è stata effettuata solo una 'verifica' delle relazioni già note (T_A,T_B,T_C); in altre applicazioni, con i dati di output del *training set* generati secondo regole non conosciute a priori, l'analisi delle derivate parziali e la matrice GWM possono essere in grado di 'svelare' le relazioni sottostanti il funzionamento della rete.

2.5 Agenti e RNA

Data la capacità di emulare il comportamento umano, le reti neurali artificiali rappresentano uno strumento estremamente utile per la costruzione degli agenti economici nell'ambito dei modelli ABM; mediante l'utilizzazione di tale metodologia computazionale, è, infatti, possibile realizzare agenti artificiali che riproducono alcune delle caratteristiche tipiche del comportamento degli individui reali.

In particolare è possibile realizzare agenti economici artificiali:

- che operano, secondo il principio della razionalità limitata;
- in grado di apprendere e di adattare il proprio comportamento al variare delle condizioni che descrivono l'ambiente;
- capaci di costruire una rappresentazione degli altri agenti e di interagire con essi.

Le reti neurali permettono, in sostanza, di realizzare agenti adattivi capaci di apprendere e di 'inferire', ma anche di commettere errori e di comportarsi in modo incoerente e talvolta irrazionale.

Quando le reti neurali vengono utilizzate per simulare il comportamento degli agenti, all'interno dei modelli basati su agenti, occorre tuttavia tenere conto di alcuni aspetti particolari del processo di apprendimento.

Nei modelli ABM l'apprendimento e le azioni degli agenti avvengono contemporaneamente; gli agenti apprendono sulla base dei dati (input e target) generati dal modello; di conseguenza l'apprendimento è aperto e non avviene mai sugli stessi dati. La generazione dei dati viene ripetuta per un ragionevole numero di 'giorni'; numero che, però, è nettamente inferiore al numero di cicli di apprendimento tipici di una applicazione di *backpropagation*. Ciò può determinare una ridotta significatività dei pesi della rete neurale. Una possibile soluzione a questo problema è quella di ripetere il *learning* sui dati via via generati dal modello. In sostanza, oltre all'apprendimento durante l'azione, ogni *n* giorni la rete viene fatta apprendere *off-line* sulla serie di dati generati dal modello; tale apprendimento ripetuto è concettualmente assimilabile ad un processo si riflessione ex-post da parte dell'agente.

Le RNA consentono di realizzare, oltre ad agenti economici con proprie regole di comportamento apprese, anche agenti che sviluppano le proprie regole di comportamento in modo completamente autonomo.

Nelle reti neurali analizzate fin qui, l'apprendimento avveniva sulla base di un insieme di pattern predefinito; l'output desiderato della rete (target) veniva fornito dall'esterno. Di seguito viene descritta una tecnica di apprendimento della rete – proposta in Terna (1994) - che non fa uso di target definiti dall'esterno; in questo caso, la rete non impara a comportarsi secondo regole precostituite; impara, invece, ad autosviluppare proprie regole di comportamento.

2.6 Il metodo dei cross-target (CT)

Il metodo dei cross-target consente di costruire agenti artificiali adattivi senza usare regole definite a priori. La caratteristica principale degli agenti realizzati con tale metodologia è lo sviluppo di una specifica forma di coerenza interna tra le azioni che l'agente pone in essere, per raggiungere un determinato obiettivo, e la valutazione delle conseguenze di tali azioni.

Gli agenti CT vengono realizzati con reti neurali che presentano nello strato di output una suddivisione in due parti; da una parte vi sono i nodi di uscita relativi alle azioni da compiere e dall'altra i nodi di uscita relativi agli effetti di tali azioni.

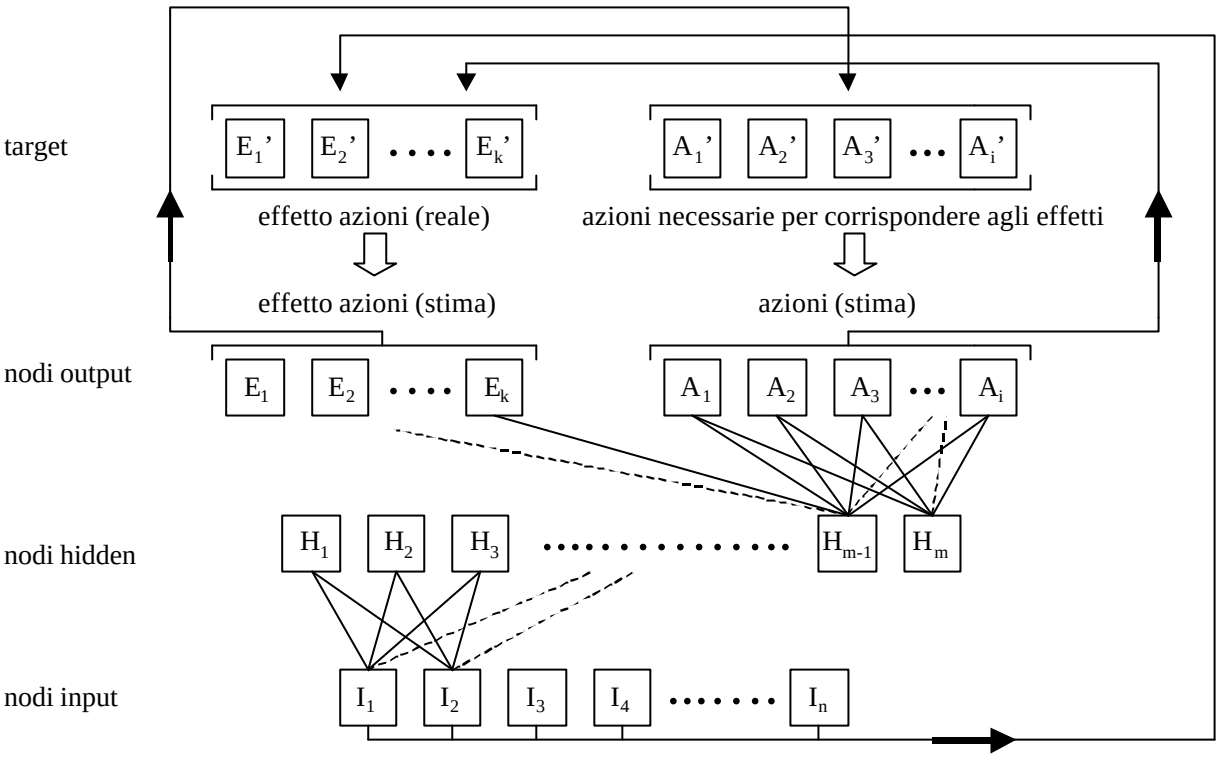
I target necessari per l'apprendimento della rete vengono costruiti in modo incrociato: i target delle azioni vengono costruiti in coerenza con gli output della rete concernenti gli effetti, in modo da sviluppare la capacità dell'agente di decidere azioni che producano i risultati previsti; i target degli effetti, invece, vengono costruiti in coerenza con gli output della rete relativi alle azioni, in modo da sviluppare la capacità dell'agente di stimare gli effetti delle azioni che sta decidendo.

L'obiettivo che si pone la tecnica dei CT è di realizzare agenti economici in grado di sviluppare, con l'apprendimento, la capacità di valutare in modo coerente le azioni da compiere; agenti, cioè, in grado di decidere quale azione eseguire per ottenere un determinato risultato e di comprendere le conseguenze delle azioni compiute.

Tale obiettivo viene realizzato senza la necessità di introdurre, esplicitamente o implicitamente, ipotesi predefinite sul comportamento degli agenti; comportamento che è semplicemente il prodotto dello sviluppo della coerenza tra decisioni circa le azioni da compiere e congetture circa gli effetti di tali azioni.

Come affermato in Terna (1994), ad un osservatore esterno il comportamento degli agenti realizzati con il metodo dei CT può apparire finalizzato al raggiungimento di determinati piani e obiettivi. Nella realtà, tuttavia, l'agente non possiede tali entità che sono semplici 'invenzioni' dell'osservatore.

Nella figura seguente viene illustrato il processo di apprendimento di una rete neurale secondo lo schema dei CT.



la rete sulla base degli input di ingresso - generalmente costituiti da informazioni provenienti dall'ambiente, azioni precedenti dell'agente o azioni degli altri agenti - produce in output le congetture circa le proprie azioni e i loro effetti.

I target necessari per il processo di apprendimento sono rappresentati dall'effetto reale dell'azione compiuta dall'agente e dall'azione necessaria per 'corrispondere' all'effetto congetturato. Tali target sono noti solo al momento dell'azione dell'agente, in quanto le azioni sono necessarie per costruire le informazioni su cui si fonda l'apprendimento.

La tecnica di CT, dunque, produce simultaneamente azione e apprendimento; il training set non è, come per le reti neurali descritte in precedenza, fornito dall'esterno e non può essere costruito a priori perché questo tipo di modelli non si basa su regole definite ex-ante, ma sulla scoperta di regole autosviluppate (il target viene autogenerato; non vi è nessun intervento di entità esterne).

Come è possibile notare dal grafico (direzione delle frecce), gli effetti reali delle azioni - calcolati tenendo conto delle azioni congetturate (e dell'eventuale conseguenza dell'interazione tra agenti) - sono utilizzati per 'allenare' la rete a stimare correttamente gli effetti; le azioni necessarie per corrispondere agli effetti stimati - calcolati tenendo conto degli effetti stimati delle azioni - sono, invece, utilizzate per 'allenare' la rete a stimare le azioni.

Più in dettaglio il processo di apprendimento con la tecnica dei CT, si articola nelle seguenti fasi:

1) generazione degli output: sulla base delle informazioni ricevute in input e dei pesi delle connessioni tra i diversi nodi, la rete neurale effettua una stima delle azioni da compiere (lato destro della figura) e degli effetti di tali azioni (lato sinistro);
2) definizione del target degli effetti: i valori che la rete deve imparare a produrre per quanto riguarda le congetture sugli effetti delle azioni sono costruiti sulla base delle azioni decise; in questo modo le stime degli effetti tendono a diventare coerenti con le valutazioni delle azioni;
3) target delle azioni: la differenza tra target degli effetti e l'output (degli effetti) generato dalla rete viene utilizzata per la modificazione delle azioni al fine di avvicinarle alle congetture degli effetti[68];
4) back-propagation: sulla base degli output generati dalla rete e dei target, si effettua l'apprendimento correggendo i pesi della rete al fine di ottenere stime degli effetti più vicine alle reali conseguenze delle azioni congetturate e stime di azioni più coerenti con le stime sugli effetti.

Tali fasi vengono ripetute per un determinato numero di cicli o giorni in modo da garantire un adattamento continuo della rete alle modificazioni ambientali. Tuttavia, poiché il numero di giorni nei quali la rete viene fatta apprendere è considerevolmente inferiore al numero di cicli di apprendimento tipici di una applicazione di *backpropagation* (dato che l'apprendimento con i CT avviene contemporaneamente all'azione), la rete può risultare sotto-adattata[69] e non in grado di reagire a mutamenti ambientali rilevanti. Per superare tale difficoltà è possibile sottoporre la rete ad un ulteriore processo di apprendimento. La rete, cioè, oltre ad apprendere giorno per giorno viene fatta apprendere ex-post sulla base dei dati storici ottenuti in passato[70]. In questo modo, oltre a modificare i pesi per adattarsi in modo locale ai cambiamenti dell'ambiente, la rete risulta in grado di reagire correttamente anche a cambiamenti di rilievo.

Per una descrizione formale del meccanismo di funzionamento dei cross-target, si veda l'appendice alla fine di questo capitolo.

2.6.1 Obiettivi esterni e proposte esterne

Il metodo dei CT consente di realizzare agenti economici 'cognitivi' in grado di valutare le azioni da compiere e gli effetti di tali azioni.

Lo sviluppo della coerenza interna tra azioni ed effetti determina l'emergere di interessanti meccanismi autosviluppati capaci di determinare azioni anche complesse.

Oltre a consentire lo sviluppo di tale coerenza, la tecnica dei cross-target permette di sviluppare alcune 'evoluzioni' del metodo che possono essere utilizzate dal ricercatore per 'regolare' il comportamento dell'agente.

---

[68] Poiché le formule inverse necessarie per la costruzione dei target delle azioni sono spesso indeterminate, le correzioni sono distribuite in modo casuale fra tutti i target; inoltre, se più correzioni relative agli effetti si riferiscono ad uno stesso target si prende in considerazione solo la correzione più grande in valore assoluto.

[69] Analizzando ogni giorno le matrici dei pesi, si osserva che i cambiamenti maggiori si concentrano nella matrice che collega i nodi hidden con quelli output; la matrice dei pesi tra unità input e unità hidden, invece, si modifica marginalmente

[70] Tale apprendimento può avvenire anche periodicamente sui dati più recenti.

In particolare, è possibile introdurre nello schema CT:

a) obiettivi esterni (EO), per influenzare la determinazione delle stime degli effetti delle azioni;

b) proposte esterne (EP), per influenzare le stime relative alle azioni.

Gli obiettivi esterni vanno a sostituire gli originari target degli effetti costruiti in modo incrociato[71]; le proposte esterne, invece, sostituiscono i target delle azioni.

Per un agente economico un obiettivo esterno può essere, ad esempio, quello di incrementare il proprio patrimonio ad un certo tasso di sviluppo o di aumentare la liquidità del proprio portafoglio di investimento; una proposta esterna può, invece, essere quella di imitare il comportamento degli altri agenti.

L'introduzione degli EO e EP nello schema CT permette di regolare il comportamento della rete e di ottenere interessanti effetti sulla dinamica comportamentale dell'agente.

Nel modello SUM, presentato nel quarto capitolo, si ha un esempio di agente 'cognitivo' realizzato con la tecnica dei CT e con l'uso di obiettivi esterni.

APPENDICE: IL METODO "CROSS-TARGET"[72]

Si consideri una rete neurale che in output debba produrre due azioni, A1 e A2, e un effetto E1.

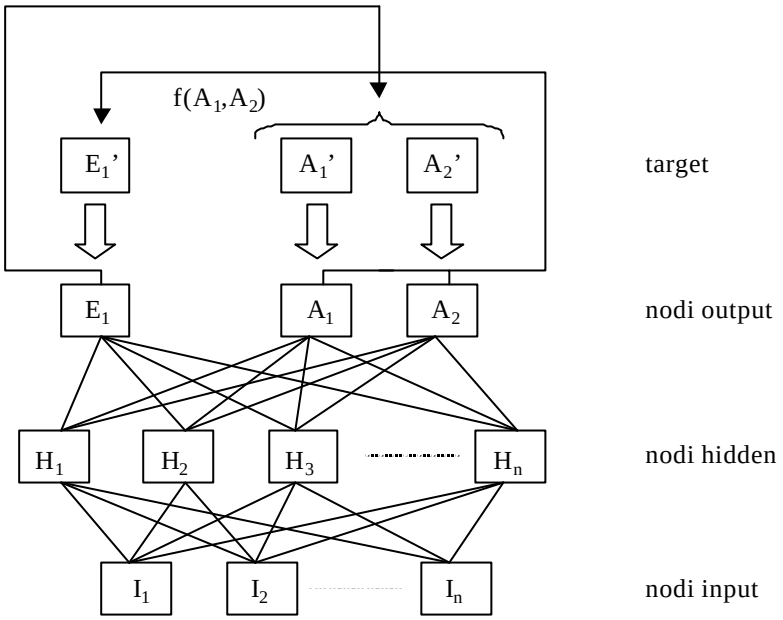
Il target per l'effetto è pari a:

$E_1' = f(A_1, A_2)$ (1)

dove f è una relazione che collega azioni ed effetto in maniera puramente contabile.

[71] I target incrociati vengono, però, ancora calcolati al fine di disporre delle informazioni necessarie per le correzioni dal lato delle azioni.

[72] Terna P.(1994)



La rete ha per obiettivo quello di produrre un output per l'effetto, E_1, che si avvicini al 'vero' effetto delle azioni rappresentato dal target E_1'.

L'errore tra l'effetto reale e quanto stimato dalla rete, pari a:

$e = ½(E_1' - E_1)^2$

viene utilizzato in fase di apprendimento per modificare i pesi delle connessioni tra i diversi nodi in modo da avvicinare le congetture sull'effetto all'effetto reale.

La rete deve produrre, inoltre, due stime di azioni, A_1 e A_2, coerenti con i valori prodotti dal lato dell'effetto (E_1); per ottenere tale risultato si correggono i pesi che determinano le grandezze A_1 e A_2 per renderle prossime ai target A_1' e A_2' (azioni coerenti con l'effetto E_1). Dalla relazione (1) si ottengono le funzioni:

$A_1 = g_1(E_1', A_2)$;

$A_2 = g_2(E_1', A_1)$;

dove g_1 e g_2 sono relazioni 'contabili' che collegano le azioni agli effetti.

Poiché le formule inverse necessarie per la costruzione dei target sono spesso indeterminate, le correzioni degli errori sono distribuite casualmente fra i due target da costruire.

Scegliendo un valore casuale τ_1 dalla distribuzione uniforme 0÷1 e ponendo $\tau_2 = 1-\tau_1$ si ottengono i seguenti target delle azioni:

$A_1' = g_1 (E_1' - e\ \tau_1, A_2)$;

$A_2' = g_2(E_1' - e\ \tau_2, A_1)$;

Gli errori tra le azioni stimate dalla rete e le azioni 'reali', pari a:

$a_1 = ½(A_1' - A_1)^2$;

$a_2 = ½(A_2' - A_2)^2$;

vengono quindi utilizzati - in fase di apprendimento - per avvicinare l'output della rete ai target delle azioni.

BIBLIOGRAFIA

- BELTRATTI A., MARGARITA S., TERNA P. (1996), *Neural Network for Economic and Financial Modelling*, International Thomson Computer Press.
- GALLANT S. (1993), *Neural Network Learning and Expert Systems*, The MIT Press.
- HEBB D., (1949), *The organization of Behavior*, Wiley.
- MCCULLOCH W., PITTS W.(1943), *A Logical Calculus of the Ideas Immanent in Nervous Activity*, Bulletin of Math.Biophysics.

- MINSKY M., PAPERT S. (1969), *Perceptrons: An Introduction to Computational Geometry*, MIT Press.
- PARISI D. (1989), *Intervista sulle reti neurali – cervello e macchine intelligenti*, Il Mulino.
- ROSEMBLATT F. (1961), *Principles of Neurodynamics: Perceptrons and the Theory of Brain Mechanisms*, Spartan Press.
- RUMELHART D., MCCLELLAND J. (1986*), Parallel Distributed Processing: Explorations in the Microstructures of Cognition*, vol.2, MIT Press.
- RUMELHART D., MCCLELLAND J. (1991*), PDP – Microstruttura dei processi cognitivi*, Il Mulino.
- TERNA P. (1994), *Reti neurali artificiali e modelli con agenti adattivi*, XXXV Riunione Scientifica annuale della Società italiana degli economisti.

- LECUN e PARKER (1986) ...

Capitolo 3: SWARM

Introduzione

L'uso della simulazione di mondi artificiali rappresenta un campo di ricerca innovativo, che consente di superare alcuni dei limiti del formalismo matematico standard basato sul metodo deduttivo.
La simulazione di sistemi sociali ed economici permette, in particolare, di realizzare 'esperimenti di pensiero' e fornisce, quindi, un utile aiuto all'intuizione del ricercatore suggerendogli, spesso, interpretazioni originali dei fenomeni del mondo reale.
La simulazione, tuttavia, si è scontrata spesso con problemi di 'leggibilità' del modello sottostante e di 'replicabilità' dei risultati ottenuti, dovuti soprattutto all'utilizzo di differenti linguaggi per la realizzazione dei modelli.
I modelli sono stati sviluppati, infatti, con linguaggi di programmazione come Fortran, C, Pascal, e più recentemente in C++ e Java o con pacchetti software come Gauss, Mathematica e Mathlab. Ciò ha reso difficile un processo di apprendimento cumulativo; spesso risultati significativi sono rimasti isolati e presto dimenticati.
Swarm è uno strumento software, originariamente sviluppato all'interno del Santa Fe Institute sotto la direzione di C.Langton, particolarmente adatto per effettuare simulazione di sistemi sociali e per lo studio di sistemi complessi con agenti eterogenei. L'obiettivo che si propone Swarm è lo sviluppo di una base comune che possa essere utilizzata come riferimento per effettuare simulazioni e che consenta ai ricercatori di focalizzare l'attenzione sul proprio campo di competenza piuttosto che sui problemi tipici di programmazione informatica (come la creazione di grafici e di interfacce utenti).
Gli autori definiscono Swarm come "una struttura multi agente per la simulazione di sistemi adattivi complessi".
In termini pratici, Swarm è un insieme di librerie software utilizzabili per lo sviluppo di modelli di simulazione, nei quali un insieme di agenti eterogenei indipendenti interagiscono tramite eventi discreti.
Le librerie, scritte in Objective-C e Tkl/Tcl, possono essere usate senza problemi di compatibilità con i diversi sistemi operativi (Windows, Linux e Apple) e forniscono gli elementi che la maggior parte dei programmi di simulazione basati su agenti hanno in comune.
Attualmente, Swarm rappresenta uno degli strumenti di simulazione più diffusi e potenti; poiché non impone nessun vincolo inerente al modello o all'insieme di interazioni tra i suoi elementi costitutivi, qualsiasi sistema fisico o sociale può essere potenzialmente simulato. Applicazioni in Swarm sono state sviluppate, infatti, per analizzare fenomeni non solo economici, ma anche biologici, chimici e sociali.
L'utilizzo di Swarm e la conseguente standardizzazione degli strumenti di simulazione, consente inoltre agli sperimentatori di documentare in modo più chiaro ed efficace il proprio lavoro e facilita, in questo modo, il raggiungimento di un maggior grado di collaborazione tra le differenti discipline sociali, impegnate nello studio delle interazioni tra 'sciami' di elementi-individui.

Swarm è realizzato sotto la licenza pubblica GNU; di conseguenza tutte le componenti software, oltre che la documentazione e gli esempi applicativi, sono liberamente disponibili sia in formato eseguibile, sia come codice sorgente e possono essere 'scaricati' dal sito internet www.swarm.org .

3.1 PROGRAMMAZIONE ORIENTATA AGLI OGGETTI

La programmazione orientata agli oggetti (OOP) risulta particolarmente compatibile con le idee guida dei modelli basati su agenti (ABM) e sta recentemente diventando dominante nell'ambito dei linguaggi di programmazione, grazie alla diffusione di strumenti come C++ e Java.

In teoria non c'è nulla che possa essere fatto con un linguaggio OOP che non possa essere ottenuto anche con un linguaggio di programmazione tradizionale. Tuttavia quando ci si trovi a sviluppare un *Agent Based Model*, l'utilizzo del paradigma ad oggetti offre indubbi vantaggi allo sviluppatore del modello.

Rispetto alla programmazione procedurale, la programmazione orientata agli oggetti consente, infatti:

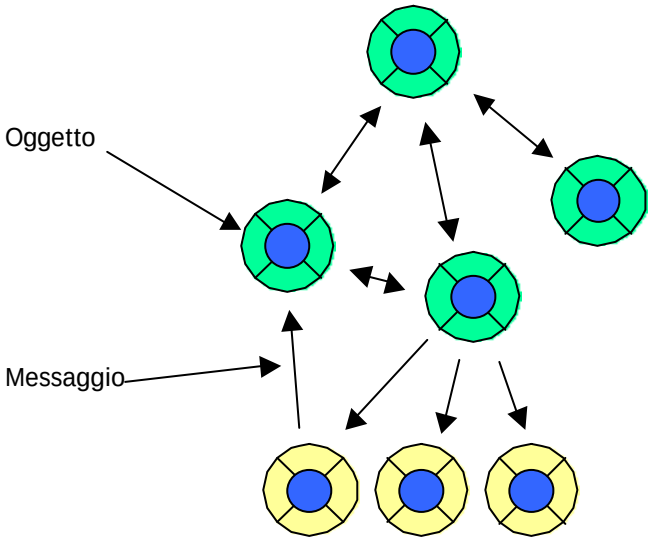
- un maggior grado di astrazione;
- permette di sviluppare in modo più intuitivo i programmi e di modificarne facilmente la struttura
- rende più semplice la comprensione del programma al lettore.

3.1.1 Oggetti, classi e messaggi

La programmazione orientata agli oggetti raggruppa i dati e le operazioni in unità modulari chiamate oggetti, e li combina in strutture reticolari che formano il programma.

Ogni oggetto ha uno stato - dati - ed evidenzia un comportamento - operazioni sui dati - in modo non molto differente da quello che accade con gli ordinari oggetti fisici ed è proprio questa somiglianza con il mondo reale che conferisce agli oggetti un elevato grado di flessibilità'; nei linguaggi OOP, i dati dell'oggetto sono denominati 'variabili istanza' e le funzioni, che ne determinano il comportamento, sono denominate 'metodi'.

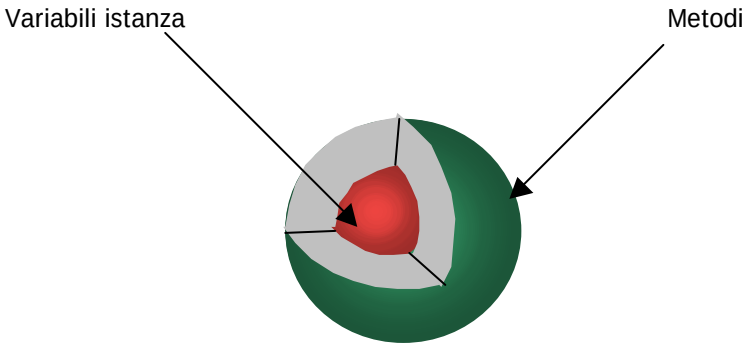
Un programma è semplicemente un insieme di oggetti e di regole che determinano come tali oggetti devono interagire e può essere rappresentato graficamente come una rete di oggetti interconnessi.



Gli oggetti e le interazioni tra oggetti sono gli elementi fondamentali di un programma OOP.
Ogni oggetto ha un ruolo specifico nella struttura generale ed è in grado di comunicare con gli altri oggetti tramite '*messaggi*', che rappresentano richieste di esecuzione di un particolare metodo.
Un oggetto può, in quest'ottica, essere considerato come un 'sottoprogramma autosufficiente' che opera all'interno della struttura più grande definita dal programma (gli oggetti non sono, dunque, semplici contenitori di dati e funzioni, ma rappresentano gli agenti che determinano, per mezzo della propria azione, l'attività del programma).
Gli oggetti di uno stesso tipo formano una *classe*. Tutti i membri di una classe sono in grado di eseguire gli stessi metodi e hanno lo stesso set di variabili istanza; essi, inoltre condividono una definizione comune.
La definizione di una classe non crea nessuna struttura utilizzabile dal programma; crea, invece, un 'prototipo' dell'oggetto, un modello, che viene utilizzato dal programma per la costruzione dei singoli oggetti 'reali', denominati esempi (*istances*) della classe.
Gli esempi di una classe condividono la specificazione delle variabili e dei metodi; non condividono, invece, lo spazio di memoria. Ogni esempio ha uno spazio di memoria allocato per il proprio set di variabili, che contiene valori caratteristici dell'esempio stesso (gli oggetti sono, così, simili ad agenti eterogenei di uno stesso tipo, che possono differenziarsi con il passare del tempo, per via delle differenti esperienze).
La memoria è allocata per le variabili di ogni nuovo oggetto, ma non vi è nessuna necessita di allocare la memoria anche per i metodi. Tutte le istanze di una stessa classe condividono l'accesso allo stesso set di metodi, di conseguenza in memoria vi è solo una copia dei metodi.
Di seguito sono illustrate le proprietà fondamentali che caratterizzano la programmazione orientata agli oggetti.

3.1.2 Incapsulamento, ereditarietà e polimorfismo.

Per garantire un elevato grado di astrazione necessario alla realizzazione dei programmi, i linguaggi orientati agli oggetti prevedono la separazione tra l'interfaccia e l'implementazione. L'interfaccia contiene la descrizione della funzionalità dei metodi di una classe e l'implementazione contiene la descrizione di come tali funzionalità vengono realizzate. L'incapsulamento rende assoluta la barriere tra interfaccia e implementazione, proteggendo quest'ultima da azioni o da accessi non voluti.
Con l'incapsulamento l'accesso ai dati è consentito solamente mediante l'uso dei metodi; in questo modo, l'oggetto, riesce a nascondere al resto del programma le proprie variabili istanza (e l'implementazione dei propri metodi).

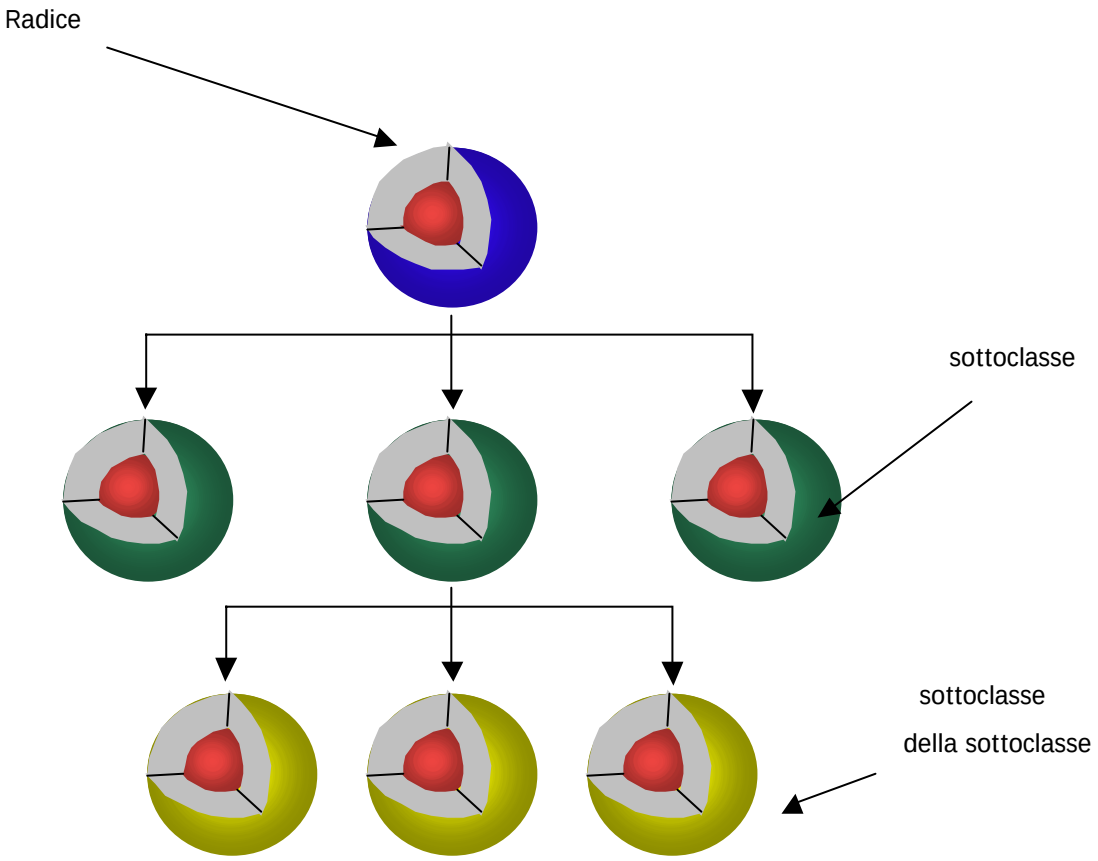


Le variabili istanza di un oggetto vengono, in pratica, nascoste dentro l'oggetto e rese invisibili all'esterno.
Ad una prima analisi tale proprietà potrebbe apparire come un fattore limitativo della libertà del programmatore. Nella realtà, invece, ciò dà al programmatore un maggior spazio di azione e semplifica la costruzione di modelli basati su agenti. Se, infatti, ogni parte dell'implementazione di un oggetto potesse divenire accessibile in ogni punto del programma, ciò 'legherebbe le mani' sia allo sviluppatore dell'oggetto che a coloro che ne vogliano fare uso; nessuno potrebbe fare modifiche senza prima controllarle e verificarle con gli altri.
Grazie all'incapsulamento è, invece, possibile modificare l'implementazione di un oggetto senza dover cambiare i programmi che fanno uso dell'oggetto stesso.
Un'altra caratteristica fondamentale dei linguaggi OOP è il polimorfismo cioè l'abilità dei differenti oggetti di rispondere, ognuno in modo proprio, a messaggi identici.
Tale capacità deriva dal fatto che i nomi definiti all'interno di una classe non sono in conflitto con i nomi definiti all'esterno della classe considerata (ciò risulta valido sia per i nomi dei metodi, che per la struttura dei dati).
I nomi dei metodi e delle variabili istanze non sono, cioè, simboli globali.
Quando un *messaggio* viene spedito ad un oggetto con la richiesta di effettuare un determinato compito, il messaggio riporta il nome del metodo che l'oggetto deve eseguire; poiché oggetti differenti possono avere metodi differenti con lo stesso nome, il significato del messaggio deve essere interpretato in relazione al particolare oggetto che lo riceve; lo stesso messaggio 'spedito' a due differenti oggetti può invocare due differenti metodi.
Grazie al meccanismo dei messaggi diventa, così, possibile 'astrarre' un determinato comportamento dell'oggetto dalla implementazione del metodo.
In definitiva, un oggetto ha accesso solamente ai propri metodi e non può 'confonderli' con quelli definiti per altri tipi di oggetti, anche se aventi lo stesso nome. Il polimorfismo permette, in questo modo, di semplificare l'implementazione dell'interfaccia: invece di inventare un nuovo nome per ogni metodo sviluppato, è possibile riutilizzare lo stesso nome per funzioni che svolgono compiti simili, ma che operano su differenti classi di oggetti.
Il polimorfismo consente, inoltre, di isolare il codice in più metodi (invece di accentrarlo tutto in una singola funzione contenente tutti i casi possibili). Ciò permette di aumentare il grado di 'riutilizzazione' del codice, che risulta, inoltre, più facilmente 'estendibile'.
Quando, infatti, si presenta un caso nuovo, inizialmente non contemplato, il codice esistente non deve essere riscritto; occorre solamente aggiungere una nuova classe con un nuovo metodo, lasciando invariato il codice che è stato già sviluppato.
Un ulteriore caratteristica distintiva del paradigma ad oggetti è l'ereditarietà.
I linguaggi OOP, consentono cioè di fondare una nuova classe su una classe già definita. La classe da cui si eredita è chiamata *'superclasse'*; la nuova classe è, invece, denominata *'sottoclasse'.*
Le due classi sono connesse in modo che la sottoclasse erediti tutti i metodi e le variabili istanza della superclasse.

Generalmente lo scopo di una sottoclasse è quello di specializzare una superclasse o di adattarla a particolari compiti; di conseguenza, la sottoclasse conterrà solamente la specificazione delle parti - metodi e variabili istanza - che differiscono rispetto alla superclasse;

Qualsiasi classe può essere impiegata come superclasse. Una classe, inoltre, può contemporaneamente essere una superclasse per alcune classi e una sottoclasse per altre.

Lo sviluppo dei legami tra le diverse classi può dare origine, pertanto, a complesse gerarchie di ereditarietà.



Come mostrato in figura ogni gerarchia di ereditarietà inizia con una classe 'radice' (*root*), che non ha superclassi. Dalla 'radice' si sviluppano, quindi, le differenti sottoclassi. Ogni classe eredita dalla propria superclasse e attraverso questa, da tutte le classi che si collocano più in alto nella gerarchia. In pratica, ogni classe è la risultante dell'accumulazione di tutte le definizioni di classe presenti nella propria 'catena di ereditarietà'.

La definizione di una sottoclasse risulta particolarmente utile nei seguenti casi:

- quando si vuole ampliare la definizione della classe iniziale per aggiungere nuovi metodi o variabili istanza;

- quando si vuole rimpiazzare un metodo esistente con una nuova versione; ciò viene realizzato implementando un nuovo metodo con lo stesso nome di quello ereditato; la nuova versione 'oscura' (*overrides*) quella ereditata; il metodo ereditato non scompare, ma risulta ancora valido per la classe che lo ha definito e per le altre eventuali classi che lo ereditano senza modificarlo;
- quando si vuole estendere o perfezionare il comportamento ereditato da un metodo. In questo caso il metodo ereditato viene 'conservato' e incorporato nella definizione del nuovo metodo; ciò viene realizzato mandando un messaggio di eseguire il vecchio metodo nel 'corpo' di quello nuovo.

L'ereditarietà consente un maggior grado di 'riutilizzazione' del codice: il codice comune viene condiviso e non vi è la necessità di riscriverlo (le classi che raggruppano i metodi e le variabili istanza comuni a più sottoclassi vengono definite '*classi astratte*').

In definitiva le proprietà descritte sopra consentono al programmatore di concepire un agente del modello come un oggetto incapsulato (*self-contained*), esempio di un iniziale oggetto-astratto, che eredita proprietà generali (che ne definiscono la sua essenza) e che può, in aggiunta, sviluppare funzioni e caratteristiche più specifiche.

Risulta a questo punto evidente il vantaggio di utilizzare un linguaggio OOP per l'implementazione di un modello di tipo ABM; le caratteristiche di un oggetto risultano, infatti, particolarmente adatte per la descrizione di un agente-tipo all'interno di un modello di simulazione.

Vi è infine un ulteriore proprietà dei linguaggi orientati agli oggetti, e dell'Objective-C in particolare, che merita di essere sottolineata: il meccanismo di allocazione della memoria. Nei linguaggi procedurali tradizionali le diverse parti costituenti il programma devono essere unite in un singolo file eseguibile al momento della compilazione o del 'link'. Nuovi moduli o tipi non possono essere introdotti al momento dell'esecuzione del programma. Ciò rappresenta un limite evidente perché costringe a definire lo spazio di memoria sulla base delle informazioni contenute nel codice sviluppato dal programmatore, invece che sulla base delle informazioni ottenute dall'utente durante l'esecuzione del programma.

I linguaggi OOP cercano di superare tale limite in modo da rendere il programma quanto più dinamico e fluido possibile, spostando molto del carico di decisioni da prendere per l'allocazione dello spazio di memoria dal momento della compilazione e del 'link' al momento dell'esecuzione. L'obiettivo è di lasciare decidere all'utente cosa deve fare il programma, piuttosto che limitare le sue azioni artificialmente, in relazione alle necessità del compilatore.

Tale obiettivo viene realizzato mediante i meccanismi di tipizzazione dinamica, di *dynamic binding* e di caricamento dinamico. Il primo meccanismo permette di associare il tipo all'oggetto al momento dell'esecuzione del programma, in *run-time,* in modo da evitare di dover codificare l'oggetto staticamente: un messaggio può, così, 'passare' come argomento un oggetto senza doverne dichiarare, esattamente, il tipo. Con il '*dynamic binding*', invece, il messaggio e l'oggetto ricevente non sono uniti fino al momento in cui il programma viene eseguito e il messaggio spedito; di conseguenza il metodo che verrà attivato in risposta al messaggio potrà essere determinato solo in *run-time* e non al momento della compilazione del codice.

Grazie ai meccanismi di tipizzazione dinamica e di 'dynamic binding' il programma può produrre risultati differenti a seconda della classe dell'oggetto che riceve il messaggio. In questo modo si ha la possibilità che fattori che agiscono al momento dell'esecuzione del programma influenzino la scelta dell'oggetto che riceve il messaggio e di conseguenza il risultato finale.
Il caricamento dinamico, infine, consente alle parti differenti di un programma eseguibile di essere conservate in file differenti. In genere solo la parte principale del programma deve essere caricata in memoria dall'inizio; gli altri moduli possono essere aggiunti nel momento in cui si verifica la richiesta da parte dell'utente. Il programma, grazie al meccanismo di caricamento dinamico, può essere 'lanciato a pezzi' ; ogni parte viene dinamicamente caricata e legata al resto del programma al momento del suo utilizzo. In questo modo le azioni dell'utente possono determinare quali parti del programma caricare in memoria e quali no.

3.1.3 La struttura di un programma OOP

Un programma realizzato con un linguaggio orientato agli oggetti evidenzia due tipi di strutture.
La prima è relativa alla struttura gerarchica delle classi, la seconda emerge dalla rete di connessioni tra gli oggetti che viene realizzata tramite i messaggi. La struttura gerarchica descrive come gli oggetti sono legati per tipo. La rete di connessioni, invece, descrive come lavora il programma.
I programmi OOP sono disegnati in modo da 'distendere' la rete di oggetti con i relativi comportamenti e insiemi di interazioni conformemente con la gerarchia delle classi.
Una parte importante dell'implementazione di un programma OOP riguarda la disposizione della rete degli oggetti, che non deve essere predefinita e statica, ma deve potersi modificare dinamicamente mentre il programma viene eseguito. Poiché i legami tra gli oggetti di una rete possono modificarsi nel tempo e talvolta risultare complessi, è necessario disporre di una struttura che li 'registri'. Il modo più semplice per raggiungere questo scopo è quello di dotare ogni oggetto di variabili istanza che mantengano traccia dei legami con gli altri oggetti, con i quali l'oggetto in questione comunica. Tali variabili, denominate *butlet* (perché registrano l'uscita per i messaggi), definiscono le connessioni principali tra gli oggetti nell'ambito della struttura del programma.
Le connessioni possono evidenziare differenti tipi di relazione tra gli oggetti. Talvolta la connessione lega oggetti che agiscono come partner in una applicazione, ognuno con un proprio ruolo ben definito. Altre volte, invece, un oggetto potrebbe configurarsi come una sotto-componente di un altro oggetto 'chiamante'.
La rete di connessione viene 'messa in moto' per mezzo di stimoli che possono provenire da 'fonti' diverse a seconda del tipo di applicazione implementata e dello scopo che questa si propone.
I programmi OOP vengono spesso attivati da sequenze di eventi, che descrivono una qualche sorta di attività o processo esterno. In applicazioni con interfaccia utente, ad esempio, gli eventi vengono generati con la tastiera o il mouse. Ogni tasto o click del mouse genera eventi che vengono ricevuti ed elaborati dall'applicazione.

Nella realizzazione di un programma orientato agli oggetti, un altro importante compito è la definizione delle classi. In particolare occorre stabilire quando aggiungere funzionalità ad una classe già esistente, definendo una opportuna sottoclasse, e quando invece sviluppare una classe indipendente.

L'ideale è definire oggetti grandi abbastanza da svolgere una funzione sostanziale nel programma, ma allo stesso tempo sufficientemente piccoli da mantenere un ruolo ben definito; nel compiere tale scelta, occorre assicurarsi che la struttura del programma possa essere facilmente compresa.

La scelta tra le due opzioni - aggiungere più funzionalità alla classe o definirne una nuova - spesso dipende dallo scopo che ci si propone. Se l'oggetto può essere usato in più situazioni e contesti, allora, scomporlo in una separata classe incrementa la possibilità di 'riutilizzazione' del codice.

In generale risulta più conveniente evitare di definire classi troppo estese che svolgono compiti diversi, ma che non possono essere adattate in altre situazioni. Quando, infatti, gli oggetti sono ideati come 'componenti' diventa più facile riutilizzare il codice già sviluppato.

Poiché gli oggetti combinano 'stati' e 'comportamenti' come avviene per gli oggetti reali, è possibile immaginarli in termini di azioni compiute (cosa fanno), di modi di agire (come lavorano) e di interagire (come sono connessi).

Un programma OOP, in questa ottica, può essere visto come un modello che descrive il fenomeno oggetto dell'applicazione implementata. Ogni componente del modello – ogni tipo di oggetto – viene descritto in termini di comportamenti e delle sue interazioni con gli altri componenti.

Il processo di 'progettazione' del modello avviene partendo dalla definizione dei comportamenti delle sue singole componenti. Solo dopo che il comportamento di un oggetto è stato definito, viene scelta una opportuna struttura dei dati.

La costruzione di un programma OOP non richiede necessariamente la scrittura di grandi quantità di codice. Il 'riutilizzazione' del codice già sviluppato consente, infatti, di costruire programmi avvalendosi di classi realizzate da altri. I linguaggi di programmazione orientata agli oggetti, inoltre, dispongono di numerose librerie, con centinaia di classi utili per l'implementazione delle diverse applicazioni.

Alcune di queste classi realizzano funzioni base, come l'indirizzamento dei messaggi, e il *data storage*, altre invece sono più specifiche e possono essere utilizzate per sviluppare interfacce utenti e visualizzazioni video.

Generalmente, un gruppo di librerie viene utilizzato al fine di definire una parziale struttura del programma, una 'intelaiatura' utile per costruire una varietà di differenti applicazioni. In sostanza, la struttura definita dalle librerie definisce una parte della rete di oggetti del programma, con la relativa struttura gerarchica, e il codice sviluppato 'in proprio' completa il programma, partendo da questa 'intelaiatura'.

Oltre a facilitare il compito del programmatore nello sviluppo di applicazioni complesse, il paradigma ad oggetti:

- permette di ottenere un maggior grado di collaborazione tra gli sviluppatori delle diverse applicazioni;
- riduce la necessità di coordinare i dettagli di implementazioni realizzate da altri;
- consente di organizzare il programma in moduli, implementabili separatamente e quindi più comprensibili anche in fase di 'debug' ;

- semplifica il disegno del modello e delle sue relazioni grazie all'ereditarietà e aumenta il grado di 'affidabilità' del codice.

3.2 L' OBJECTIVE-C

L'Objective-C è un linguaggio di programmazione orientato agli oggetti, sviluppato come estensione del linguaggio C standard: le 'aggiunte' sono basate principalmente su SmallTalk, uno dei primi linguaggi OOP, e hanno lo scopo di dotare il C delle capacità necessarie per essere utilizzabile, in modo semplice ed efficace, anche secondo il paradigma ad oggetti.

Poiché l'Objective-C, incorpora il linguaggio C, è possibile scegliere quando realizzare una particolare funzionalità con il paradigma ad oggetti, ad esempio definire una classe, e quando invece utilizzare la programmazione di tipo procedurale, ad esempio per definire una funzione.

I punti di forza di questo linguaggio che, lo hanno fatto preferire agli altri linguaggi di programmazione orientati agli oggetti per la realizzazione delle librerie software di Swarm, sono: la facilità d'uso, la sintassi chiara, la capacità di astrazione, ma soprattutto il maggior grado di 'dinamismo'.

L'Objective-C è, infatti, il più dinamico dei linguaggi OOP basati sul C; molte decisioni sull'allocazione dello spazio di memoria che di solito vengono prese al momento della compilazione, vengono invece spostate, in Objective-C, al momento dell'esecuzione del programma, il che conferisce al linguaggio una maggiore flessibilità.

L'Objectice-C può essere usato, inoltre, come estensione del linguaggio C++ il quale, sebbene sia un linguaggio orientato agli oggetti, non possiede i meccanismi di 'dynamic binding' e di tipizzazione dinamica descritti nel paragrafo precedente.

Di seguito sono spiegate alcune delle caratteristiche e delle istruzioni più utilizzate per l'implementazione di un programma; gli esempi riportati sono tratti da: '*Object-Oriented Programming and the Objective-C language, Developer's library*' edito da Apple Computer (1999).

3.2.1 Oggetti, messaggi e classi

In Objective-C gli oggetti sono identificati da un puntatore ad un oggetto, o più precisamente da un puntatore ai dati dell'oggetto, denominato *id*. Tutti gli oggetti, indipendentemente dalle variabili istanza che li caratterizzano, sono di tipo *id* e ciò consente di definirli in modo non restrittivo.

```
id anObject
```

Nell'esempio riportato, il puntatore *id* definisce *anObject* come un oggetto, ma non fornisce nessuna informazione supplementare sul tipo di metodi o di variabili istanza di *anObject*.

Per far eseguire un determinato compito ad un oggetto occorre spedirgli un messaggio con l'indicazione del metodo da 'applicare'. In Objective-C ciò viene realizzato nel modo seguente:

```
[receiver message]
```

Il '*receiver*' è un oggetto e il '*message*' è semplicemente il nome di un metodo dell'oggetto '*receiver*'.
Quando viene spedito un messaggio, il sistema in run-time seleziona il metodo appropriato dall'insieme dei metodi del '*receiver*' e lo esegue. Un messaggio può prevedere l'indicazione di alcuni argomenti:

```
[myRect setArg1:10.0 Arg2: 15.0]
```

In questo caso, l'ipotetico oggetto *myRect* riceve un messaggio con due argomenti che vengono specificati dopo i due punti (generalmente una parola chiave descrive gli argomenti e la sua collocazione precede i due punti).
Un messaggio, inoltre, può essere annidato dentro un altro messaggio

```
[myRect setPrimaryColor:[otherRect primaryColor]];
```

Nell'esempio, il colore di un rettangolo (*myRect*) viene definito con riferimento al colore di un altro rettangolo (*otherRect*).
I metodi hanno un accesso automatico alle variabili istanze dell'oggetto ricevente senza la necessità di doverle dichiarare come argomenti. Se, tuttavia, come nell'ultimo esempio, un metodo richiede informazioni su di una variabile di un altro oggetto, deve mandare un apposito messaggio all'oggetto 'destinatario'.
In Objective-C il polimorfismo viene realizzato mediante il puntatore *id*: se un messaggio denominato *display* viene mandato ad una variabile di tipo *id*, tutti gli oggetti dotati di un metodo *display* sono considerati potenziali 'ricevitori' del messaggio.
L'Objective-C dispone di un numero elevato di classi che il programmatore può utilizzare come classi 'predefinite' da incorporare nel programma o come base per lo sviluppo di specifiche sottoclassi (utilizzabili per specializzare l'applicazione). E' così possibile implementare oggetti estremamente sofisticati scrivendo solamente una piccola parte del codice e riutilizzando il lavoro fatto da altri programmatori.
La classe radice dell'Objective-C è *NSObject*. Tale classe, che definisce la struttura base degli oggetti, conferisce alle altre classi e alle istanze le abilità necessarie per 'comportarsi' come oggetti e per 'cooperare' nell'ambito di un sistema run-time. Per convenzione, nei linguaggi orientati agli oggetti, i nomi delle classi cominciano con un lettera maiuscola e i nomi degli oggetti istanza con una lettera minuscola.
Nella definizione di un oggetto in alternativa al puntatore *id*, può essere utilizzato il nome di una classe. Questo tipo di dichiarazione, nota come 'tipizzazione statica', permette al compilatore di ottenere maggiori informazioni sul tipo di oggetto definito.

```
Rectangle *myRect
```

Gli oggetti in sostanza vengono definiti come puntatori ad una classe e non come puntatori ad un generico oggetto. Ciò oltre a consentire al compilatore di effettuare alcuni tipi di controllo – per esempio avvisare se un oggetto riceve un messaggio al quale non è in grado di rispondere – permette anche di rendere più chiare le 'intenzioni' del programmatore ai lettori del codice.
La funzione principale di un oggetto classe è di creare nuove istanze. In Objective-C ciò viene realizzato utilizzando il metodo *alloc* che alloca dinamicamente la memoria per le variabili istanza del nuovo oggetto e le inizializza al valore zero.

```
id myRect
MyRect = [Rectangle alloc]
```

Nell'esempio riportato la classe *Rectangle* crea una nuova istanza e la assegna alla variabile *myRect*. Come è possibile notare, l'oggetto classe è rappresentato nel codice dal nome della classe (tale equivalenza è però valida solamente all'interno di un messaggio). Nei casi in cui l'inizializzazione di un oggetto risulti più specifica si ricorre generalmente ad un metodo *init*, realizzato secondo le esigenze del programmatore.
Ogni classe ha almeno un metodo, come *alloc*, che permette di produrre nuovi oggetti e un metodo, come *init*, che li inizializza[73] (i metodi di inizializzazione spesso prevedono il passaggio di alcuni argomenti; in questo caso il nome del metodo riflette il compito a loro assegnato).
Il nome della classe oggetto può essere utilizzato, nel codice, in due contesti differenti: per definire staticamente un tipo di oggetto o come '*receiver*' in un messaggio.

```
Rectangle *anObject;
anObject = [ [Rectangle alloc] init];
```

Una dasse e una variabile globale non possono avere lo stesso nome, poiché i nomi della classe sono i soli ad avere una visibilità globale in Objective-C.
La definizione di una classe può includere, oltre ai metodi ideati per gli oggetti istanza, anche metodi appositamente pensati e realizzati per gli oggetti classe. Tali metodi, denominati *metodi di classe*, possono essere ereditati dalle classi più in alto nella gerarchia esattamente come avviene per i metodi degli oggetti istanza.
Poiché gli oggetti classe non hanno accesso alle variabili degli oggetti istanza (e non possono inizializzarli, leggerli o modificarli), in alcune circostanze occorre dotarli di variabili statiche. Tali variabili, che non sono ereditate dalle sottoclassi, conferiscono agli oggetti classe un maggior grado di funzionalità rispetto alla semplice funzione di creazione delle istanze. Alcune classi dichiarano variabili statiche e sono dotate di metodi di classe con lo scopo di gestirle; in questo modo le classi possono essere utilizzate, ad esempio, per

[73] In Swarm la creazione di un oggetto invece del metodo *alloc* utilizza *create* (se occorre creare un oggetto con una specifica inizializzazione delle variabili, invece del metodo *init* si utilizzano i metodi *createBegin* e *createEnd*).

coordinare le istanze, per eliminare istanze da liste di oggetti già creati o gestire altri processi essenziali per le applicazioni. Analogamente agli oggetti istanza, anche gli oggetti classe possono essere inizializzati. Ciò viene realizzato spedendo un messaggio di *inizialite* all'oggetto classe prima che riceva ogni altro messaggio (se una classe fa uso di variabili statiche o globali, il metodo *inizialite* è il posto ideale per definirne i valori).

3.2.2 Definizione di una classe: interfaccia e implementazione

In Objective-C la definizione di una classe prevede:

- una *interfaccia,* che dichiara i metodi e le variabili istanza della classe e la classe da cui si eredita;
- una *implementazion*e, che contiene il codice che 'realizza' effettivamente la classe.

L'interfaccia e l'implementazione sono separati in due differenti file. Sebbene sia possibile dichiarare o implementare più classi in uno stesso file, per facilitare la comprensione del codice ogni classe ha un proprio file di interfaccia e di implementazione. Il file di implementazione ha estensione ".m"; il file di interfaccia, invece, ha estensione ".h".
Lo scopo del file di interfaccia è di dichiarare la classe agli altri 'moduli'. L'interfaccia, infatti, contiene tutte le informazioni, che le altri parti del programma devono conoscere per lavorare con la nuova classe e cioè: come la classe è connessa alla gerarchia delle ereditarietà, quali variabili istanza sono contenute nell'oggetto e quali messaggi possono essere spediti all'oggetto classe e alle sue istanze.
Una volta definito - tramite l'interfaccia - come un oggetto deve interagire con gli altri elementi del programma, l'implementazione della classe può essere liberamente modificata senza dover alterare nessuna parte dell'applicazione.
La dichiarazione dell'interfaccia inizia con l'istruzione del compilatore **@interface** e termina con l'istruzione **@end**.

```
@interface ClassName : ItsSupreclass
{
  instance variable declarations
}
method declarations
@end
```

Dopo l'istruzione **@interface** segue il nome della nuova classe e l'indicazione della superclasse dalla quale si eredita (se il nome della superclasse viene omesso, la nuova classe viene considerata una classe radice).
In seguito si ha la dichiarazione delle variabili istanza, racchiuse tra parentesi graffe, e dei metodi.
I metodi di classe sono preceduti dal segno "**+**", quelli usati dalle istanze della classe sono, invece, preceduti dal segno "-". Il tipo di dati 'restituiti' dal metodo sono dichiarati usando la sintassi C standard (se il tipo non viene esplicitamente dichiarato, viene considerato di tipo *id*).

Ogni 'modulo' che richiama una classe interfaccia – perché, ad esempio, crea una istanza, o spedisce un messaggio o ha bisogno di una variabile - deve includerla nel proprio file di interfaccia mediante l'istruzione **#import**.

Poiché la definizione di una classe viene costruita sulle definizioni delle classi ereditate, il file di interfaccia inizia importando l'interfaccia della superclasse.

```
#import  "ItsSuperclass.h"

@interface ClassName : ItsSuperclass
{
  instance variable declarations
}
method declarations
@end
```

Se l'interfaccia 'menziona' classi non appartenenti alla gerarchia definita, deve importarle esplicitamente o dichiararle mediante la direttiva **@class**.

```
@class Rectangle, Circle;
```

Nell'esempio riportato la direttiva ha il semplice scopo di informare il compilatore che "Rectangle" e "Circle" sono nomi di classe, ma non 'importa' le relative interfacce (generalmente il file di interfaccia usa **@class** per dichiarare le classi e il file di implementazione importa le relative interfacce).

Il file di implementazione inizia, invece, con l'istruzione **@implementation** e termina con **@end**. Ogni file deve importare la propria interfaccia, ma può omettere di indicare il nome della superclasse e la dichiarazione delle variabili istanza.

```
# import "ClassName.h"

@implementation ClassName
method definitions
@end
```

I metodi di una classe sono definiti all'interno delle parentesi graffe.

```
+ alloc

{
 . . .
}

- (BOOL) isfilled

{
 . . .
}

- (void) setFilled (BOOL) flag

{
 . . .
}
```

Per default la definizione di un metodo istanza ha tutte le variabili istanza dell'oggetto. Di conseguenza il metodo può riferirsi alle variabili semplicemente per nome. Quando, tuttavia, le variabili istanza appartengono ad un oggetto che non è il 'ricevente' del messaggio, il tipo dell'oggetto deve essere reso esplicito al compilatore attraverso la tipizzazione statica. In questo caso per riferirsi alle variabili istanza viene usato l'operatore di puntatore alla struttura ("->").

Sebbene siano dichiarate nel file di interfaccia, la variabili istanza sono più legate al momento dell'implementazione. Nell'interfaccia, infatti, vi è l'elencazione dei metodi, ma non la definizione della struttura interna dei dati.

Al fine di rendere quanto più flessibile l'uso delle variabili istanza, il compilatore permette di definirne tre differenti livelli di visibilità, ognuno dei quali è contraddistinto da una specifica direttiva:

- **@protected**: la variabile istanza è accessibile all'interno della classe che la dichiara e delle classi che la ereditano;
- **@private**: la variabile istanza è accessibile solamente all'interno della classe che la dichiara. Ciò può risultare utile, ad esempio, per evitare legami troppo stretti tra una sottoclasse e la implementazione di una determinata classe o quando si voglia evitare che la sottoclasse introduca inavvertitamente 'bugs' nella classe che dichiara la variabile;
- **@public**: la variabile è accessibile ovunque. In questo caso uno dei principi fondamentali della programmazione orientata agli oggetti – l'incapsulamento dei dati all'interno degli oggetti – viene meno; l'uso di questo tipo di variabili dovrebbe essere, quindi, evitato a meno di casi eccezionali.

Per default, tutte le variabili istanza non specificamente definite sono considerate di tipo **@protected**.

L'Objective-C fornisce, infine, due utili istruzioni che possono essere utilizzate all'interno di un metodo per far riferimento agli oggetti: **self** e **super**. Tali istruzioni, contenute nei messaggi spediti ai metodi degli oggetti, indicano al programma, in quale classe ricercare l'implementazione del metodo che si vuole eseguire.

self ricerca l'implementazione del metodo riportato nel messaggio partendo dalla classe dell'oggetto ricevente; *super*, invece, ricerca il metodo partendo dalla superclasse.

Si supponga di disporre di una classe e di una superclasse, dotate di un metodo con un identico nome *negotiate,* e che tale metodo venga richiamato all'interno del metodo denominato *makeLastingPeace*, definito nella sottoclasse. Se

l'implementazione del metodo *makeLastingPeace* è:

```
-  makeLastingPeace
{
     [self negotiate];
     . . .
}
```

il programma ricerca il metodo *negotiate* nella sottoclasse. se invece l'implementazione è:

```
-  makeLastingPeace
{
      [super negotiate];
      . . .
}
```

allora il metodo viene ricercato nella superclasse e l'istruzione super consente, così, di 'oscurare' (*overrides*) il metodo definito nella sottoclasse. In alcuni casi, le classi implementano un metodo che esegue solo una parte del compito e per la parte restante passano il messaggio alla classe super.

```
-  (id)init
{
       [super init];
       . . .
}
```

Il metodo *init*, riportato sopra, opera in questo modo; prima di inizializzare le varibili istanza definite nella propria classe, manda un messaggio di *init* alla classe super per inizializzare le variabili istanza che si erediteranno.

3.3 JAVA

Gli autori di Swarm forniscono, un'interfaccia per sviluppare le applicazioni, oltre che con l'Objective-C, anche con Java. Tale linguaggio, sebbene in prima analisi appaia simile all'Objective-C[74], ne differisce per alcuni importanti aspetti.

Di seguito sono elencate le principali differenze tra i due linguaggi:

- in Java si ha una tipizzazione statica: le variabili e i tipi devono essere sempre specificati; ciò rappresenta uno svantaggio dal punto di vista della flessibilità della programmazione, ma può risultare molto utile in fase di compilazione per il controllo e la prevenzione degli errori. In Objective-C, poiché molte decisioni vengono spostate dalla fase di compilazione al *run-time*, può accadere che l'applicazione vada in errore al momento dell'esecuzione. In Java, invece, ciò viene evitato in quanto il compilatore effettua un controllo preventivo sulla 'coerenza' del programma;
- l'Objective-C è il linguaggio nativo di Swarm; la maggioranza delle risorse sono scritte in questo linguaggio; Java può risultare più lento in quanto il codice viene prima tradotto, in fase di compilazione, in un metacode (bytecode) comprensibile da ogni piattaforma e successivamente viene interpretato dalla specifica piattaforma utilizzata;
- l'Objective-C ha tutte le caratteristiche di basso livello del C, che sono utilizzate principalmente per lo sviluppo dei sistemi operativi; Java, invece, non ha nessuna di queste proprietà. Tuttavia in molte

[74] Entrambi si ispirano alla sintassi del C.

circostanze, per esempio nella costruzione di modelli ABM, ciò non costituisce una limitazione (anzi, l'assenza di tali caratteristiche evita di incorrere in errori non voluti);

- sebbene la comunità degli sviluppatori di applicazioni Swarm abbia una maggiore esperienza di programmazione con l'Objective-C, tale linguaggio ha avuto una scarsa diffusione; Java, invece, è il linguaggio di programmazione OOP più diffuso e ha una naturale predisposizione per le applicazioni su Internet (esempio: *applets*); inoltre dispone di una libreria di funzioni grafiche più ampia di quella di Swarm.
- Java dispone di un meccanismo denominato '*garbage collection*' che ha lo scopo di distruggere automaticamente gli oggetti non più utilizzati e di alleggerire l'esecuzione dell'applicazione; in Objective-C l'eliminazione degli oggetti non più usati, non viene fatta automaticamente, ma devono essere utilizzate opportune istruzioni all'interno del programma; lo sviluppatore dell'applicazione deve effettuare una attenta 'contabilità' degli oggetti al fine di eliminare quelli non più utilizzati, in modo da evitare che la memoria del computer giunga al livello di 'saturazione' (con la conseguenza di un rallentamento della simulazione o addirittura di un '*crash*').

3.3.1 Alcune differenze formali

- La differenza principale riguarda la dichiarazione e l'implementazione delle classi; in Java la definizione di una classe non prevede due distinti file come in Objective-C; la classe viene definita in un unico file con estensione java (è obbligatorio che il nome della classe sia uguale al nome del file).
- in Java il puntatore *id* non esiste e non si può lasciare vuoto il tipo di una variabile o di un metodo (per un oggetto, si usa il tipo *Object*);
- in Objective-C si ha la presenza di un file denominato *main.m* dal quale viene inizializzato Swarm e avviata l'applicazione; in Java, invece, la funzione *main()* è contenuta nel file principale dell'applicazione che per convenzione inizia con la parola *Start*' (la funzione *main()* deve essere dichiarata di tipo *void*);
- in Java, vi sono due modi di dichiarare una *instance* di una classe:
  a) si può utilizzare un metodo simile a quello dell'ObjectiveC usando *createBegin*() e *createEnd*();
  b) si può utilizzare il costruttore della classe, preceduto dalla keyword *new*; il costruttore è semplicemente un metodo della classe che consente di inizializzare le variabili dell'oggetto che si vuole creare (deve avere lo stesso nome della classe). È possibile disporre di più costruttori - con argomenti diversi - per ogni classe. Tale caratteristica, denominata *overloading*, permette di dichiarare più funzioni con lo stesso nome; il compilatore sceglie quale funzione attivare analizzando il numero ed il tipo di argomenti;
- in Java, la dichiarazione dei metodi ha una sintassi diversa rispetto a quella dell'Objective-C; un metodo può essere dichiarato nel seguente modo:

```
public int methodName(int arg1, float arg2, string[12] arg3);
```

oppure

```
public int methodNameUsingArg1$WithArg2(int arg1,float arg2);
```

dove $ corrisponde ai singoli argomenti del metodo.

- in ObjectiveC, la sintassi utilizzata per spedire un messaggio ad un oggetto è:

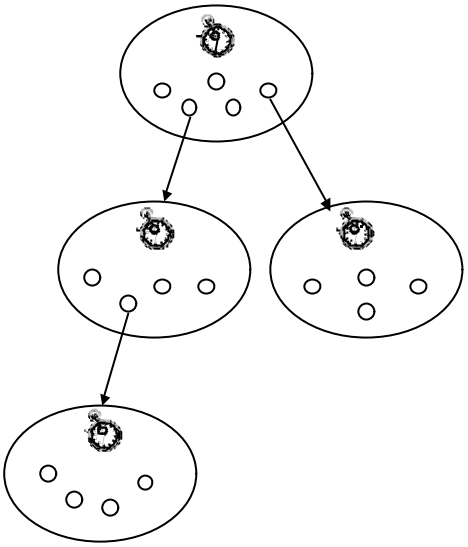
```
[objectName methodName:argomenti];
```

con Java, invece, è:

```
objectName.methodName(argomenti);
```

3.4 SWARM

Una caratteristica fondamentale di Swarm è la capacità di gestire con semplicità il tempo della simulazione.
Nei linguaggi di basso livello, come il C, lo sviluppatore del modello deve gestire sia la struttura dell'agente, generalmente mediante vettori, sia il tempo della simulazione, mediante cicli di iterazione; ciò comporta la scrittura di una elevata quantità di codice.
Con l'Objective-C, invece, il modellatore non si deve preoccupare dei problemi di gestione riguardanti la memoria degli agenti, ma deve preoccuparsi principalmente della gestione del *timing* della simulazione, sempre mediante l'attivazione di opportune iterazioni. Con Swarm, invece, è possibile evitare sia i problemi di gestione della memoria sia quelli di sincronizzazione; con tale strumento, infatti, non solo gli agenti, ma anche gli eventi sono trattati come oggetti[75].
L'unità principale di una simulazione Swarm è l'agente; una simulazione, infatti, non è altro che un insieme di agenti che interagiscono (in una simulazione economica, ad esempio, gli agenti saranno rappresentati da: banche, società, investitori, ecc; in altre parole, le componenti simulate del modello).
Come avviene nei linguaggi OOP, ogni agente è descritto da una classe, che viene utilizzata per produrre-fabbricare il numero di 'individui' che popoleranno il modello di simulazione ideato.
La componente fondamentale che organizza gli agenti è un oggetto denominato '*swarm'*. Tale oggetto contiene gli agenti e la rappresentazione del tempo.
Uno *swarm* è, in sostanza, un insieme di individui, dotati di un proprio elenco - *schedule* - di azioni da eseguire.
In Swarm, le azioni dei singoli agenti hanno luogo in un tempo specifico; il tempo 'scorre' sulla base di un scansione di eventi successivi che viene realizzata mediante particolari strutture dati denominate *schedule.* Tali oggetti raggruppano le azioni degli agenti in uno specifico ordine di esecuzione (ogni azione specifica il tipo di metodo che deve essere applicato sull'oggetto target).
Oltre ad essere contenitore di agenti uno *swarm* può essere esso stesso un agente. In questo caso il suo comportamento emerge dall'azione delle singole entità definite nel proprio *swarm*.
Con Swarm è, dunque, possibile definire modelli gerarchici annidando *swarm* multipli;



Ciò conferisce a tale strumento una notevole potenzialità e flessibilità d'uso: *swarm* multipli possono essere utilizzati, ad esempio, per la costruzione di agenti con componenti cognitive, dotati di propri modelli di rappresentazione del mondo che li circonda; inoltre, poiché gli oggetti *swarm* possono essere creati e distrutti durante l'esecuzione della simulazione, Swarm può essere utilizzato per modellare sistemi nei quali livelli multipli di descrizione emergono dinamicamente.

3.4.1 Struttura di una simulazione con Swarm

Come affermato dagli autori, l'idea base di Swarm è quella di fornire un contesto entro il quale un elevato numero di agenti possano '*live their lives*' interagendo l'uno con l'altro in modo distribuito e concorrente.
Con Swarm, gli esperimenti di simulazione seguono, generalmente, il seguente schema:

a) creazione di un 'mondo artificiale' dotato di spazio, tempo e oggetti; gli oggetti, allocati all'interno della struttura del 'mondo', determinano il proprio stato secondo specifiche regole di comportamento;
b) creazione di un numero di strumenti il cui scopo e di osservare, registrare e analizzare i dati prodotti dal comportamento degli oggetti (nel mondo creato precedentemente);
c) esecuzione della simulazione;
d) interazione con l'esperimento attraverso opportuni oggetti 'sonda', utili per effettuare controlli sperimentali[76].

[3]In questo modo gli eventi possono essere 'schedulati' mediante strumenti 'sensibili' allo scorrere del tempo.
[4]Tramite le sonde è possibile indagare la struttura interna degli oggetti del modello.

Il cuore di una simulazione risiede nella definizione del modello 'mondo'. Nel più semplice dei casi, un modello consiste di un solo *swarm* 'abitato' da un gruppo di agenti con uno *schedule* di attività da eseguire.

Alcuni strumenti di simulazione, in fase di definizione del modello, definiscono l'agente in relazione ad un prefissato ambiente. Una caratteristica distintiva di Swarm è la possibilità di definire l'ambiente senza dover far riferimento a strutture già definite ed immutabili (griglie, reticoli, ecc). L'ambiente entro il quale gli agenti vivono ed interagiscono viene definito come un oggetto con le caratteristiche volute e implementate dall'ideatore del modello.

Una volta definiti gli agenti (e le relazioni con l'ambiente), l'ultimo passo è di metterli assieme in un oggetto *swarm*.

Si tratta, cioè, di 'scrivere' uno *schedule* di attività, che definisca come il tempo debba essere simulato nel sistema (creando un set di azione di uno specifico ordine di esecuzione.).

In genere, al fine di analizzare i dati generati dalla simulazione, vengono predisposti i seguenti strumenti:

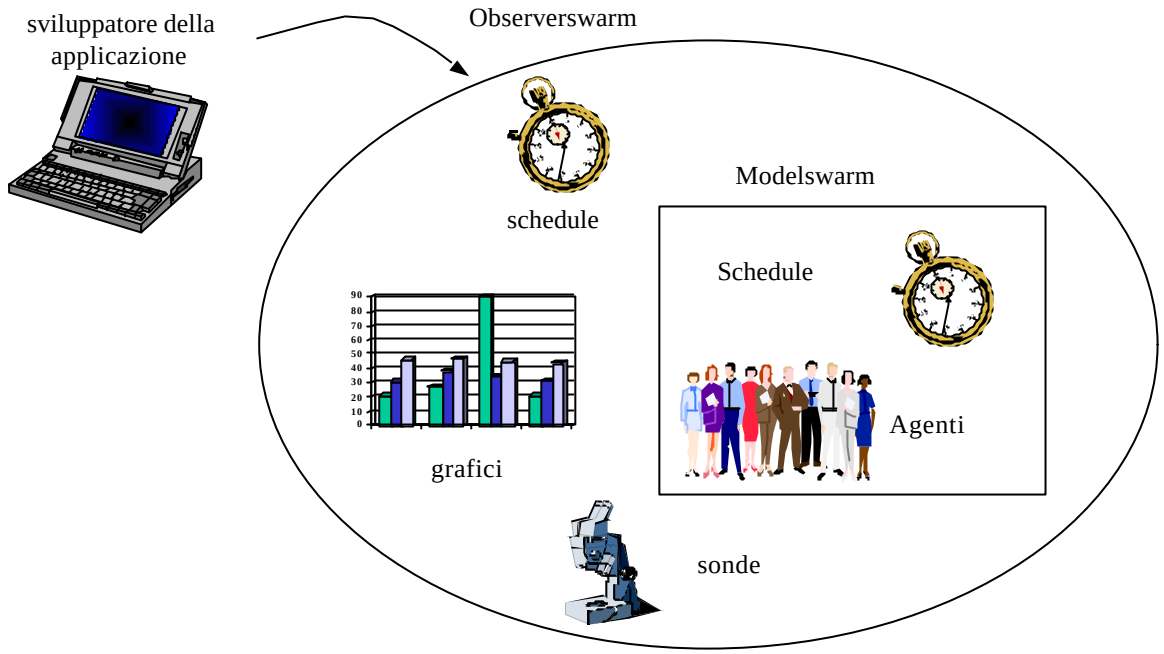
- un osservatore del modello, in grado di seguire la dinamica del modello e di registrarne i dati;
- un insieme di 'sonde' (*probe*) in grado di controllare lo stato degli oggetti durante l'esecuzione della simulazione.

L'osservatore del modello è anch'esso un oggetto *swarm*, cioè un gruppo di agenti con un proprio *schedule* di attività. La struttura della simulazione viene ottenuta combinando l'osservatore con il modello ideato (eseguito come un sotto-swarm dell'osservatore).

In pratica, l'esperimento viene realizzato separando il momento dell'esecuzione del modello - il *modelswarm* - dal momento dell'osservazione dei risultati prodotti dal modello stesso – *l'observerswarm*; da una parte vi è il modello che 'lavora' sulla base di un proprio 'orologio' interno, e dall'altra vi è l'osservatore che visualizza i risultati del modello secondo una propria scansione del tempo.

Con i linguaggi di programmazione tradizionale risulta piuttosto difficile sincronizzare i due livelli; con Swarm, invece, grazie allo presenza di uno strumento flessibile come lo *schedule*, diventa relativamente facile gestire il *timing* della simulazione.

La figura riportata di seguito descrive il procedimento concettuale che guida lo sviluppo di una simulazione con Swarm.



3.4.2 La struttura di una applicazione

Per realizzare una applicazione Swarm occorre definire i seguenti elementi:

- il main: è il file fondamentale del programma; ha lo scopo di inizializzare Swarm e di avviare la simulazione;

```
#import <simtools.h>
#import "ObserverSwarm.h"

int main (int argc, const char **argv)
{
ObserverSwarm *observerSwarm;

initSwarmApp (argc, argv);

observerSwarm =  [ObserverSwarm create: globalZone];
[observerSwarm buildObjects];
[observerSwarm buildActions];
[observerSwarm activateIn: nil];
[observerSwarm go];
return 0;
}
```

in genere, prevede la seguente sequenza standard di operazioni:

a) creazione dell'osservatore del modello;

b) creazione degli oggetti dell'osservatore;

c) creazione dello schedule;

d) attivazione dell'osservatore e dello schedule;

e) avvio della simulazione.

Come è possibile notare dal listato riportato sopra, nel main non vi è nessun riferimento al modello; il modello viene, infatti, creato all'interno dell'osservatore.

- l'observerswarm: è la classe che consente di osservare e controllare l'andamento della simulazione; nell'observerswarm, , il cui schema generale è riportato di seguito,

```
// ObserverSwarm.h

#import "ModelSwarm.h"
#import <simtoolsgui/GUISwarm.h>
#import <analysis.h>

@interface ObserverSwarm: GUISwarm
{
  int displayFrequency;
  . . .
  id displayActions;
  id displaySchedule;
  id <EZGraph> Graph1,Graph2,
  ModelSwarm *modelSwarm
}

+ createBegin: aZone;
- createEnd;
- buildObjects;
```

```
- buildActions;
- activateIn: swarmContext;
- checkToStop;
- . . .;
@end
```

vengono definiti i seguenti oggetti necessari per l'osservazione della simulazione[77]:

a) la 'probe' dell'osservatore, tramite la quale è possibile visualizzare e modificare le variabili e i parametri dell'esperimento relativi all'osservazione dei risultati;

b) il *modelswarm*, cioè il modello da osservare;

c) i grafici e le sonde;

d) il pannello di controllo: tale oggetto consente di seguire la simulazione agendo su 5 bottoni: start (avvia la simulazione), stop (arresta la simulazione), next (effettua un passo), save (salva la disposizione delle finestre con i grafici e le 'probe') e quit (fa terminare l'applicazione).

[5]Ad eccezione della *probe*, definita nel metodo ggetti elencati vengono creati all'interno del metodo *buildObjects*.

Il *timing* dell'osservatore viene definito, in coerenza con il *timing* del modello, all'interno del metodo *buildActions*[78] (in genere, lo *schedule* dell'osservatore riceve come segnali di ingresso le uscite del modello e aggiorna i grafici per la visualizzazione dei risultati).

- il modelswarm: è la classe che contiene la descrizione del modello; è qui che vengono creati gli oggetti necessari per la realizzazione della simulazione, tra i quali gli 'agenti' che popolano il modello;

```
// ModelSwarm.h
#import <objectbase/Swarm.h>
#import <simtools.h>
#import <objectbase.h>
#import <activity.h>
#import <collections.h>

@interface ModelSwarm: Swarm
{
  int dayNumber, . . .;
  float . . .:
```

```
   id <ActionGroup> modelActions1;
   id <Schedule> modelSchedule;
   . . .
}
+ createBegin: aZone;
- createEnd;
- buildObjects;
- buildActions;
- activateIn: swarmContext;
- increaseCurrentDayNumber;
- (int) getCurrentDay;
- . . .

- openProbeTo: (int) ag;
@end
```

analogamente all'osservatore, il modelswarm dispone di una propria *probe* con le variabili e i parametri del modello oggetto di studio (che possono essere modificati dall'utente prima dell'avvio della simulazione) e di uno *schedule* con il quale viene definito il *timing* della simulazione;

- i file che descrivono gli agenti del modello: così come avviene in Objective-C, ogni agente della simulazione viene definito mediante un file di interfaccia (.h) e un file di implementazione(.m);
- il Makefile: serve a definire le condizioni della compilazione del programma; in ObjectiveC, la compilazione usa i file di interfaccia e di implementazione per creare i file oggetto(con estensione .o). Tali file, espressi in formato binario, vengono quindi 'linkati' insieme per generare il file eseguibile dall'utente dell'applicazione (file con estensione .exe)[79]. Il Makefile deve contenere i seguenti componenti: il nome del file eseguibile (alla voce APPLICATION=), l'elenco dei file oggetti (.o) da

[6]Il metodo *buildActions* prima di definire lo *schedule* dell'osservatore manda un messaggio di *buildActions* al modelswarm affinchè il modello crei il proprio *schedule*.

[7]Per compilare il programma occorre digitare dal prompt del terminale di Swarm, dopo essere andati nella directory del programma, il comando 'make'.

creare (alla voce OBJECTS=),l'indicazione della directory in cui si trova il file Makefile.appl, l'elenco dei file che compongono i file oggetto.

3.4.3 Schedule

Poiché lo schema presentato in precedenza prevede due distinti livelli - l'osservatore e il modello - la gestione del *timing* della simulazione viene ottenuta combinando opportunamente due distinti *schedule*: quello *dell'observerswarm* e quello del *modelswarm*.

Tali schedule vengono realizzati all'interno dei metodi *buildActions* e hanno il seguente schema generale[80].

```
- buildActions
{
  int i;

    modelActions1 = [ActionGroup create: self];
  [modelActions1 createActionTo: self message: M(increaseCurrentDayNumber)];
  . . .
     modelActions2 = [ActionGroup create: self];
   [modelActions2 createActionForEach: agentList message: M(act2)];

  modelSchedule = [Schedule createBegin: self];
   [modelSchedule setRepeatInterval:3];
  modelSchedule = [modelSchedule createEnd];

  [modelSchedule at: 0 createAction: modelActions1];
  [modelSchedule at: 1 createAction: modelActions2];
  [modelSchedule at: 2 createActionTo: object message: M(step)];

    return self;
}
```

Swarm, al fine di realizzare la gestione desiderata del tempo, si avvale di due particolari strutture di dati, fornite dalla libreria *activity*: gli *actionGroup* e gli *schedule*.

Un *actionGroup*, è una lista di eventi che associa ad ogni oggetto target il metodo che deve essere eseguito; uno *schedule* invece è, come detto in precedenza, una struttura che combina le azioni degli oggetti in uno specifico ordine di esecuzione.

Mentre lo *schedule* viene utilizzato per definire uno specifico *timing* della simulazione, con azioni eseguite in un certo ordine, gli *actionGroup* vengono utilizzati al fine di raggruppare insiemi di eventi che devono essere eseguiti 'simultanemante'[81].

In definitiva il metodo *buildActions* opera nel seguente modo:

- crea uno, o più, gruppi di azioni (*ActionGroup*): tali gruppi contengono la sequenza degli eventi che devono essere eseguiti simultaneamente dal modello;

[8]Nello *schedule* del modelswarm si hanno i *modelActions* e *modelSchedule*; in quello dell'observerswarm, invece, i *displayActions* e *displaySchedule*; la struttura dello *schedule* è sostanzialmente simile per i due livelli; ciò che cambia sono i messaggi e i target.

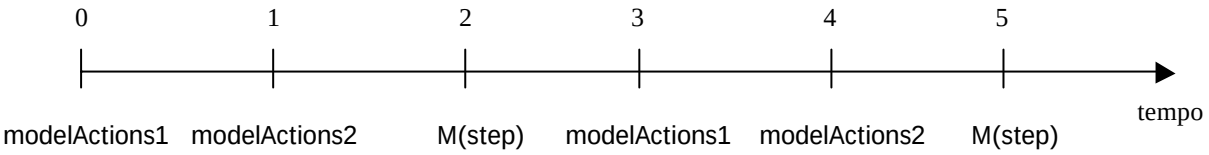
[9]Nella realtà, poiché Swarm non implementa ancora una esecuzione parallela, gli eventi contenuti in un *actionGroup* sono eseguiti sequenzialmente.

- crea lo 'schedulatore' del tempo (*Schedule*), la cui funzione è di stabilire la scansione temporale degli eventi; lo schedule deve definire, inoltre, il *time-step*, cioè l'intervallo di ripetizione degli eventi (*setRepeatInterval*)[82];

Nell'esempio riportato lo *schedule* (*modelSchedule*) definisce un 'orologio' della simulazione con 3 *time-step* e stabilisce che il modello deve eseguire:

- al tempo 0, gli eventi definiti nel primo gruppo di azioni (*modelActions1*);
- al tempo 1, gli eventi definiti nel secondo gruppo di azioni (*modelActions2*);
- al tempo 2, il metodo *step* sull'oggetto *object*.

In altre parole, lo *schedule* definisce il tempo 'relativo' degli eventi (i quali possono essere rieseguiti dopo 3 *time-step*).



3.4.4 Le librerie Swarm

Le librerie di Swarm permettono allo sviluppatore del modello di creare oggetti avvalendosi delle classi realizzate dagli autori; in sostanza, creando semplici 'istanze' delle classi fornite; tale uso si presenta, in genere, quando occorre creare strumenti altamente tecnici come lo *schedule*; in alternativa, lo sviluppatore, per far fronte a particolari necessità, può decidere di creare una propria classe ereditando le funzionalità di una o più librerie di Swarm.

Di seguito sono brevemente illustrate le librerie principali:

- objectbase: contiene le classi fondamentali sulle quali si basa il funzionamento degli agenti; le classi principali di tale libreria sono SwarmObject e Swarm. La prima è la classe radice (*root*) per tutti gli agenti simulati: tutte le classi che definiscono un agente ereditano le funzionalità di SwarmObject (tra le quali l'interfaccia per la gestione della memoria); la classe Swarm, invece, contiene le funzionalità necessarie per la realizzazione del modelswarm e dell'observerswarm;
- gui: tale libreria permette la creazione di immagini, grafici a linee, istogrammi e gestisce le interazioni dell'utente con le interfacce grafiche;
- simtools: contiene le classi necessarie per il controllo dell'esecuzione dell'intera simulazione; tale libreria contiene, inoltre, gli strumenti necessari per caricare o salvare oggetti in file ASCII;
- defobj: è la classe radice (root) nella gerarchia delle classi delle librerie Swarm; definisce la infrastruttura generale del modello, ampliando la struttura di base dell'Objective-C; contiene le

[10]L'intervallo di ripetizione è il numero di *time-step* tra l'invocazione della prima azione e l'inizio di un nuovo processo di 'schedulamento'; tale intervallo equivale, in sostanza, a definire i *tick* dell'orologio della simulazione.

classi che definiscono i metodi base per la creazione e distruzione degli oggetti e le classi che consentono di archiviare le istanze degli oggetti;

- activity: contiene il nucleo centrale dei meccanismi di simulazione; tale classe fornisce gli strumenti necessari per la creazione delle strutture dati che 'schedulano' gli eventi della simulazione;
- collections: permette di creare e gestire oggetti come: liste, vettori, mappe; i metodi di tale libreria permettono, ad esempio, di spedire messaggi a tutti i membri di una lista e di cancellare o aggiungere membri ad un vettore;
- random: fornisce all'utente una serie di generatori di numeri casuali tratti dalla letteratura; avvalendosi di tale libreria è possibile definire set di generatori di numeri casuali indipendenti; tale caratteristica risulta particolarmente utile poiché facilita la ripetibilità degli esperimenti (ed evita che i risultati siano dovuti a generatori con bias o correlazioni);
- probe: consente di definire gli oggetti sonda grazie ai quali è possibile osservare le variabili e i parametri degli oggetti istante per istante;
- tkobjc: tale libreria, basata su Tcl/Tk, fornisce gli oggetti di base necessari per realizzare e gestire una interfaccia grafica;

in aggiunta alle librerie descritte, richieste da tutte le applicazioni Swarm, vi sono diverse librerie 'aggiuntive' che possono essere utilizzate per scopi particolari come, ad esempio, la realizzazione di algoritmi genetici o di reti neurali.

Tali librerie sono, spesso, il risultato dei contributi della comunità degli utenti di Swarm.

Nel paragrafo seguente, al fine di mostrare le potenzialità pratiche di Swarm, viene analizzata una interessante applicazione economica.

3.5 IL MODELLO "ASM"

L' "*Artificial Stock Market*", sviluppato nei primi anni '90 all'Istituto di Santa Fe in New Mexico da Arthur, Holland, LeBaron, Tayler e Palmer, rappresenta uno dei primi modelli di mercato finanziario basato su agenti. Il modello, utile per la comprensione di alcune interessanti caratteristiche dei mercati finanziari reali, combina una ben definita struttura economica del mercato con forme di apprendimento degli agenti realizzate utilizzando un classifier system.

Nel mercato reale, la complessità del problema dell'allocazione del portafoglio di investimento, forza gli operatori ad agire induttivamente, usando semplici regole empiriche nel tentativo di ottimizzare le proprie risorse. Queste regole non sono statiche, ma vengono continuamente valutate e aggiornate secondo la performance realizzata.

Nel mercato artificiale analizzato, gli agenti decidono quali investimenti effettuare sulla base delle proprie previsioni del futuro stato del mercato e dell'esperienza maturata nel corso del tempo; gli agenti non solo aggiornano il modello di previsione lineare, basato sulle aspettative razionali, ma devono anche effettuare una selezione delle informazioni disponibili considerate rilevanti nel processo di formazione delle previsioni.

3.5.1 La struttura del mercato

Gli agenti devono decidere come investire il proprio portafoglio tra due tipologie di attività finanziarie.
La prima è un titolo *"risk free"*, in offerta infinita, che paga un tasso di interesse costante $r_f = 10\%$; la seconda è un titolo rischioso che paga un dividendo stocastico secondo il seguente processo autoregressivo:

$d_t = d + \rho(d_{t-1} - d) + \mu_t$

con $d = 10$, $\rho = 0.99$ e $\mu_t \sim N(0,\sigma^2_\mu)$.

Il numero delle attività rischiose è uguale al numero degli agenti. Il prezzo di una attività rischiosa, p_t, viene determinato endogenamente nel mercato.

3.5.2 Preferenze

Gli agenti, con orizzonte temporale limitato ad un periodo, massimizzano una funzione di utilità, con avversione assoluta al rischio costante (CARA), del tipo:

$E^i_t (- e^{-\gamma W^i_{t+1}})$

soggetta al vincolo:

$W^i_{t+1} = x^i_t(p_{t+1} + d_{t+1}) + (1 + r_f)(W^i_t - p_t x^i_t)$

dove E^i_t indica la migliore previsione dell'agente *i* al tempo *t*, γ è il coefficiente di avversione assoluta al rischio, W^i_{t+1} è il patrimonio dell'agente *i* al tempo *t+1* e x^i_t è la domanda di attività rischiose dell'agente *i* al tempo *t*.

Con la funzione di utilità di tipo CARA, definita sopra, e distribuzione normale per dividendi e prezzi, la domanda per l'attività rischiosa da parte dell'agente *i*, è data da:

$$x^i_t = \frac{E^i_t(p_{t+1} + d_{t+1}) - (1+r_f)p_t}{\gamma\sigma^2_{p+d,i}}$$

dove $s^2_{p+d,i}$ è la previsione dell'agente *i* della varianza di *p+d*.

Tale funzione di domanda è valida se gli shock nei prezzi delle attività rischiose sono distribuiti normalmente; ciò risulta vero nell'equilibrio ad aspettative razionali determinato sotto, ma potrebbe risultare non valido in altre situazioni. Il modello ipotizza che i 'disturbi' sui prezzi non siano troppo lontani dalla distribuzione normale in modo da non invalidare la funzione di domanda; se i disturbi si allontanano dalla normale ciò non invalida l'analisi, ma rompe il legame tra la funzione di domanda definita sopra e la funzione di utilità di tipo CARA.

Poiché gli agenti hanno lo stesso coefficiente di avversione assoluta al rischio risulta facile calcolare l'equilibrio nel caso di aspettative razionali (REE). Ipotizzata, infatti, una relazione lineare tra prezzo dell'attività rischiosa e dividendo del tipo:

$p_t = f\, d_t + e;$

inserendo tale relazione nella funzione di domanda x^i_t e forzando ogni agente a detenere una unità di attività rischiosa ad ogni tempo, si ottiene:

$$f = \frac{\rho}{1 + r_f - \rho} \qquad e = \frac{d(f+1)(1-\rho) - \gamma\sigma^2_{p+d}}{r_f}$$

Data, poi, la relazione diretta tra prezzo e dividendo e la dinamica di quest'ultimo, è possibile esprimere le previsioni ottimali nell'equilibrio ad aspettative razionali nel modo seguente:

$E(p_{t+1} + d_{t+1}) = \rho(p_t + d_t) + (1 - \rho)((1 + f)d + e)$

o in forma più compatta come:

$E(p_{t+1} + d_{t+1}) = a(p_t + d_t) + b$

Questa equazione di previsione rappresenta una delle relazioni principali che gli agenti dovranno stimare è sarà interessante osservare quanto i parametri stimati dagli agenti si avvicineranno a quelli dell'equilibrio ad aspettative razionali. I parametri utilizzati nelle simulazioni eseguite sono riportati nella tabella 1.

Tabella 1: valori dei parametri utilizzati nelle simulazioni

Parametri	valore
γ	0.3
d	10
r_f	0.10
ρ	0.99
σ^2_μ	0.07429
σ^2_{p+d}	4.00
f	9
e	-2
$(1 - \rho)((1 + f)d + e)$	0.98
a range	[0.495,1.485]
b range	[-4.51,6.47]

Il modello descritto fin qui si configura come un modello neoclassico standard: gli agenti sono avversi al rischio con funzione di utilità di tipo CARA; hanno una funzione di domanda standard per l'attività rischiosa, basata sulla previsione del prezzo e del dividendo del periodo successivo; una volta determinate le previsioni calcolano la domanda per il titolo rischioso e la 'sottopongono' ad un banditore (specialist), che determina il prezzo di equilibrio (in modo da uguagliare la domanda aggregata all'offerta fissata); le risorse non investite nell'attività rischiosa sono impiegate in un titolo "*risk free*" (*cash*,, nella simulazione).

L'elemento innovativo del modello ASM, che segna il punto di rottura con i modelli economici classici, risiede nella 'specificazione' degli agenti, i quali utilizzano per la formazione delle previsioni un insieme di differenti regole che vengono fatte evolvere mediante un algoritmo genetico.
A differenza del modello economico standard, nel modello ASM gli agenti dispongono, quindi, di funzioni di domanda eterogenee, frutto dei propri metodi di previsione del prezzo e del dividendo futuro.

3.5.3 Previsioni

Lo scopo dell'agente è di costruire previsioni dei futuri prezzi e dividendi da utilizzare nella funzione di domanda delle attività rischiose. Le previsioni vengono formate usando un classifier system e in particolare usando quelle che sono chiamate regole di 'condizione-previsione'. Ogni agente, in pratica, viene dotato di un insieme di regole di classificazione (classifier rules), che descrivono mediante un sistema binario le condizioni del mercato.
Se la condizione del mercato in un dato periodo 'unisce' la descrizione di una regola, allora la regola è potenzialmente utilizzabile per prevedere il prezzo e il dividendo del periodo successivo. In uno stesso periodo più regole di previsione possono 'unire' lo stato del mercato, dando così all'agente una maggiore possibilità di scelta.
Le regole vengono monitorate secondo l'accuratezza della previsione che hanno generato e solo quelle migliori vengono attivate e utilizzate per creare, in un processo di apprendimento periodico, nuovi set di regole, mediante l'utilizzo di un algoritmo genetico.
Le regole di classificazione richiedono di predefinire il set di stati binari che descrive la condizione del mercato (ciò limita gli agenti a previsioni costruite sulla base di variabili di stato fissate a priori).
Il set di stati definito nel modello comprende sia informazioni 'tecniche' che 'fondamentali' ed è sintetizzato in un vettore di lunghezza 16 bit. Ogni elemento indica se la condizione definita nella tabella 2 è verificata (1) o no (0) nel mercato. Nell'equilibrio ad aspettative razionali, le informazioni contenute in tabella non forniscono nessuna informazione addizionale in quanto il prezzo e il dividendo atteso si basano esclusivamente sul prezzo e dividendo del periodo corrente.

Tabella 2: bit di condizione

Bit	condizione
1	prezzo * interesse/dividendo > 1/4
2	prezzo * interesse/dividendo > 1/2
3	prezzo * interesse/dividendo > 3/4
4	prezzo * interesse/dividendo > 7/8
5	prezzo * interesse/dividendo > 1
6	prezzo * interesse/dividendo > 9/8
7	prezzo * interesse/dividendo > 5/4
8	prezzo * interesse/dividendo > 3/2
9	prezzo * interesse/dividendo > 2
10	prezzo * interesse/dividendo > 4

11	prezzo > media mobile di periodo 5
12	prezzo > media mobile di periodo 20
13	prezzo > media mobile di periodo 100
14	prezzo > media mobile di periodo 500
15	On:1
16	Off:0

Le regole di classificazione sono costituite da 2 parti. La prima è una stringa di bit che 'unisce' il vettore degli stati, la seconda è una previsione connessa allo stato del mercato definito dal vettore. Nella stringa di bit ogni posizione contiene uno dei 3 seguenti simboli: 1, 0, # . L'uno e lo zero 'uniscono' il bit corrispondente nel vettore degli stati, mentre il simbolo # rappresenta una *'wildcard'* e gioca un ruolo determinante, poiché consente agli agenti di decidere quali parti di informazione considerare e quali, invece, ignorare completamente (esempio: la regola 00#11 'unisce' le stringhe: 00111 e 00011).
La seconda parte della regola ha lo scopo di convertire il set di bit in una previsione del prezzo e del dividendo. Per ogni stringa di bit 'unita' vi è un corrispondente vettore di valori reali di lunghezza tre che corrispondono ai parametri di previsione lineare ed ad una stima della varianza.
Il vettore (a_j, b_j, σ^2_j) viene quindi utilizzato nella formazione delle aspettative nel modo seguente:

$$E(p_{t+1} + d_{t+1}) = a_j (p_t + d_t) + b_j$$
$$\sigma^2_{p+d} = \sigma^2_j$$

(I parametri *a* e *b* vengono inizializzati a valori casuali distribuiti uniformemente intorno ai valori di previsione dell'equilibrio REE.)

3.5.4 Trading

Una volta decisa la regola da applicare, l'agente sostituisce i parametri di previsione nella funzione di domanda dell'attività rischiosa.
Per l'agente *i* la domanda del titolo rischioso è:

$$x^i_t(p_t) = \frac{E^i_t(p_{t+1} + d_{t+1}) - (1+r_f)p_t}{\gamma\sigma^2_{p+d,i}}$$

$$x^i_t(p_t) = \frac{a_{i,j}(p_t + d_t) + b_{i,j} - (1 + r_f)p_t}{\gamma\sigma^2_{i,j}}$$

Date le domande di tutti gli agenti, il prezzo del titolo rischioso viene facilmente ricavato uguagliando la domanda totale all'offerta fissata. Una volta definito l'acquisto dei titoli, gli agenti aggiornano l'accuratezza delle regole di previsione utilizzate secondo la seguente media ponderata:

$$v^2_{t,i,j} = \frac{\tau - 1}{\tau} v^2_{t-1,i,j} + \frac{1}{\tau} ((p_t + d_t) - (a_{i,j} (p_{t-1} + d_{t-1}) + b_{i,j})) .$$

Il valore t, posto uguale a 50, gioca un ruolo estremamente importante nel modello, poiché determina la lunghezza dell'orizzonte temporale che l'agente considera rilevante per la propria previsione. Più t è grande, più è lungo l'orizzonte temporale preso in considerazione dall'agente, che in questo modo ipotizza implicitamente che il mercato sia 'stabile'. Se invece t è relativamente piccolo il mercato viene considerato dagli agenti in rapido cambiamento. Nell'ipotesi estrema con t uguale ad uno, l'agente utilizza solo l'ultimo errore di previsione e quindi le regole si fondano essenzialmente sul 'rumore'.

Gli operatori in fase di contrattazione sono soggetti ad alcune limitazioni: possono contrattare un massimo di 10 titoli in ogni periodo, non possono indebitarsi, e non possono superare un limite di vendite allo scoperto fissato in 5 titoli.

3.5.5 Apprendimento ed evoluzione delle regole

Gli agenti cambiano il proprio comportamento modificando il set di regole di previsione a disposizione (pari a 60). Il 10% di regole con performance peggiore viene eliminato e sostituito da nuove regole ottenute mediante l'uso di un algoritmo genetico. Tale algoritmo viene implementato ogni k periodi per ogni agente, in modo asincrono, per non far coincidere l'apprendimento di tutti gli agenti nello stesso periodo. La scelta delle regole da rimpiazzare e di quelle a maggior performance, da utilizzare per generare nuove regole, avviene sulla base di una misura di 'forza' s_j che viene calcolata nel modo seguente:

$\sigma^2_j = v^2_{t,i,j}$

$s_j = - (\sigma^2_j + c B_j)$

dove B_j è il numero di bit non uguali a # in una regola e c è un costo per bit, posto pari a 0.01, che ha lo scopo di penalizzare le regole complesse, assicurando che le informazioni utilizzate siano realmente utili in termini di previsioni (e spingendo allo stesso tempo gli agenti verso l'equilibrio REE).

L'algoritmo genetico opera attraverso il meccanismo del crossover, con probabilità 0.3, e della mutazione, con probabilità 0.01. Con il crossover le nuove regole vengono generate combinando il 'patrimonio genetico' di 2 opportuni 'genitori' scelti nella popolazione delle regole esistenti.

Ciò viene realizzato applicando il meccanismo del crossover uniforme (vedi cap.1) alla prima componente della regola, la stringa di bit, e applicando uno dei tre seguenti metodi per la parte reale (i parametri *a* e *b*):

- con probabilità 0.333, i parametri vengono calcolati come combinazione lineare dei parametri dei 'genitori' pesati per la relativa forza
- con probabilità 0.333, i parametri vengono scelti casualmente tra i 2 genitori
- con probabilità 0.333, i parametri vengono scelti o tutti da un genitore o tutti dall'altro.

La mutazione opera, invece, scegliendo un genitore con la 'tournament selection'. La parte relativa alla stringa di bit viene modificata nel modo seguente: dal simbolo 0 al simbolo # con probabilità 2/3, da 0 a 1

con probabilità 1/3, da 1 a # con probabilità 2/3, da 1 a 0 con probabilità 1/3, da # a 0 o a 1 con probabilità 1/3.

Per i parametri di previsione, la mutazione adotta uno dei 3 seguenti metodi:

- con probabilità pari a 0.05, il parametro viene scelto casualmente nel range definito in tabella 1
- con probabilità pari a 0.2, il parametro viene scelto casualmente in un intervallo più piccolo determinato dal vecchio valore del parametro $\pm$ 0.05 volte il proprio range.
- con probabilità pari a 0.75, il parametro viene lasciato inalterato.

Le nuove regole ereditano laccuratezza di previsione media, σ^2_j, dei due genitori a meno che questi siano stati 'uniti', nel qual caso ricevono l'errore di previsione mediano calcolato su tutte le regole.

Se una regola non viene applicata per 2000 periodi, si procede alla 'generalizzazione': un quinto dei bit viene posto pari a # e la forza della regola viene fissata al valore mediano.

All'inizio le regole sono inizializzate casualmente. I bit della stringa degli stati vengono poste uguali a # con probabilità 0.9 e uguali a 1 o 0 con probabilità 0.1. I valori reali sono posti uguali a valori casuali scelti nel range definito per ogni parametro e la stima della varianza σ^2_j è posta uguale alla stima della varianza nell'equilibrio REE, σ^2_{p+d} = 4. Ogni agente viene, poi, dotato di 1 attività rischiosa e di 10000 unità di moneta.

3.5.6 Swarm

Il modello di mercato finanziario artificiale fin qui descritto è stato realizzato con l'utilizzo di Swarm.

La costruzione della struttura del modello ha richiesto la definizione delle seguenti classi:

- Agent: contiene le variabili e funzioni comuni a tutti gli agenti. Tutte le tipologie di agenti sono sottoclassi di Agent. L'intenzione del progetto è quella di sviluppare nuovi agenti. Brandon Weber , ad esempio, ha sviluppato un agente che apprende sulla base di una rete neurale;

- Dividend: controlla il processo autoregressivo che genera il pagamento del dividendo;

- World: conserva una stringa di bit corrispondente al reale stato del mercato che viene utilizzata dagli agenti al fine di individuare le regole utilizzabili per la previsione. Conserva inoltre le informazioni dei dividendi e prezzi.

- Specialist: gestisce la fase di contrattazione dei titoli. Vi sono 3 differenti tipi di specialist: lo specialist RE, il cui scopo è di controllare se la simulazione raggiunge l'equilibrio REE, lo specialist SP che calcola il prezzo di equilibrio tra domanda e offerta, e lo specialist ETA che calcola il prezzo di equilibrio usando un processo di calcolo basato su di un parametro di stima 'eta'.

- BFAgent: contiene la descrizione dell'agente 'tipo' del modello ('Bit-string-Forecasting' agent) che opera sulla base del classifier system.

- BitVector: crea la struttuta di bit necessaria alla simulazione

- BFCast: è la classe dotata dei metodi che consentono di manipolare i bit (‘trit’) delle informazioni;

- BFParams: gestisce i parametri utilizzati dagli agenti durante la simulazione

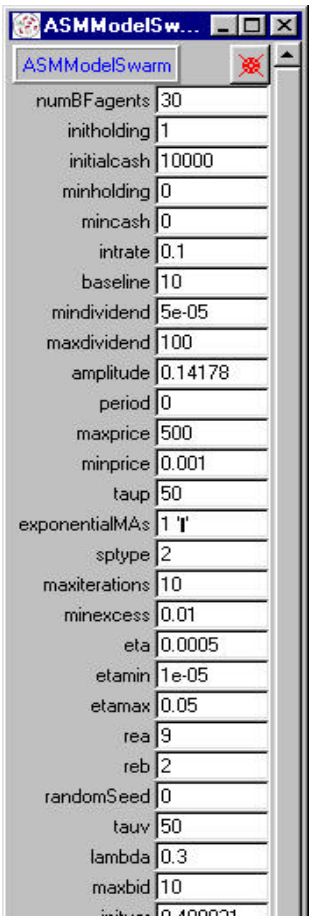
- ASMModelSwarm: definisce il ‘model’ dell’esperimento. In questa simulazione viene utilizzata una struttura Swarm annidata. Poiché gli agenti si basano sulla storia dei dividendi e dei prezzi, se la simulazione viene avviata senza un adeguato periodo di apprendimento, i risultati dei primi periodi risentono delle componenti casuali definite in fase di inizializzazione. Per evitare ciò, viene creato un periodo di ‘warm-up’ per i prezzi e i dividendi.
  ASMModelSwarm agisce, di conseguenza, per mezzo di due gruppi di azione (ActionGroup): *warmupActions* per la fase iniziale della simulazione e *periodActions* per la simulazione a ‘regime’;

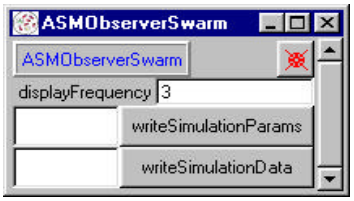
- ASMObserverSwarm: definisce l’*observer* del modello e quindi gestisce la visualizzazione dei grafici sullo schermo;

- Output: controlla la scrittura dei dati e dei parametri su file;

- ASMBatchSwarm: esegue la simulazione a scopo di controllo.

L’esperimento viene gestito agendo sui parametri del modello e dell’osservatore.





Nella finestra dell'ASMModelSwarm è possibile definire le seguenti variabili:

- il numero degli agenti dell'esperimento (*numBFagents*);
- la dotazione iniziale di attività rischiose (*initholding*) e di titoli "*risk free*" (*initcash*), il limite di vendite allo scoperto (*minholding*) e il limite di indebitamento (*mincash*);
- il tasso di interesse sull'impiego esente da rischio (*intrate*);
- i minimi e massimi per dividendi e prezzi (*mindividend*, *maxdividend*, *minprice*, *maxprice*);
- il valore centrale attorno al quale viene calcolato il dividendo (*baseline*) e l'ampiezza delle deviazioni da tale valore (*amplitudine*);
- il periodo di autocorrelazione del processo (*periodo*);
- se utilizzare una media mobile pesata esponenzialmente (*exponentialMAs*=1) o una semplice media mobile di *n* periodi (*exponentialMAs*=0) nel calcolo dell'accuratezza delle regole;
- il parametro τ che definisce l'orizzonte temporale preso in considerazione dall'agente (*tauv*);

- il tipo di specialist (RE = 0, SP = 1, ERA = 2) e i parametri da utilizzare nel calcolo del prezzo di equilibrio (*taup*, *eta*, *etamin*, *etamax*, *maxiterations*, *minexcess*, *rea*, *reb*);
- l'innesco dei numeri casuali (*randomseed* = 0 indica che l'innesco è casuale);
- l'offerta massima di attività rischiose (*maxbid*);
- la varianza iniziale (*nitvar*) e la massima deviazione di una previsione nella stima della varianza (*maxdev*);
- il coefficiente di avversione al rischio (*lambda*).

Dalla finestra dell'ASMObserverSwarm è invece possibile impostare:

- la frequenza di visualizzazione degli oggetti grafici (*displayFrequency*);
- i comandi di input/output: *writeSimulationData* consente di salvare su file i dati di una particolare applicazione e deve essere selezionato prima di avviare la simulazione, *writeSimulationParams* consente di salvare i parametri della simulazione e può essere selezionato in qualsiasi momento durante la simulazione.

3.5.7 Il timing della simulazione

Il modello ASM ha la seguente scansione degli eventi:

tempo 0: la simulazione viene fatta 'lavorare' per un dato periodo di tempo in modo da assicurare un certo grado di apprendimento (*warm-up*);

tempo t: vengono compiuti i seguenti passi:

passo 1: viene determinato il dividendo del periodo *t-1*;

passo 2: il 'World' aggiorna il proprio stato;

passo 3: gli agenti sottopongono la propria domanda, basata sulla previsione, fatta al tempo *t-1*, del prezzo e del dividendo allo specialist, il quale 'aggiusta' il prezzo per 'pulire' il mercato;

passo 4: dopo la determinazione del prezzo e del dividendo reale, gli agenti aggiornano le regole di previsione e la propria ricchezza determinata dai profitti dell'attività rischiosa e da quelli del titolo esente da rischio;

passo 5: gli agenti utilizzano lo stato corrente del mercato e le regole di previsione per prevedere il prezzo e il dividendo del periodo *t+1*;

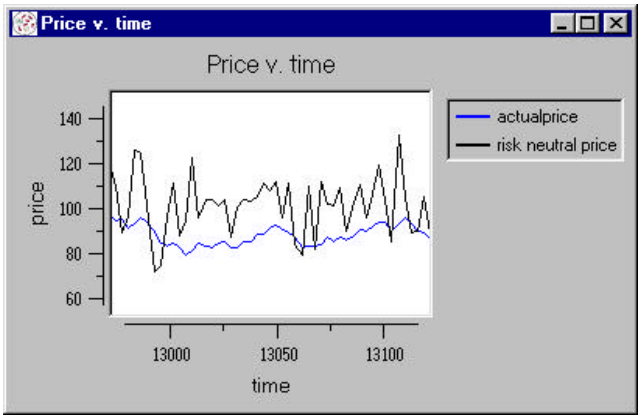
tempo t+1: il processo si ripete

3.5.8 Esperimenti

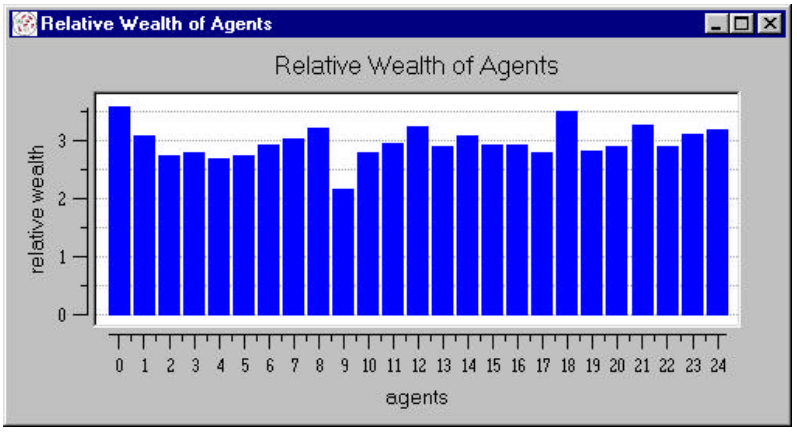
Di seguito sono riportati due esempi di applicazione del modello ASM.

Il primo 'esperimento' è stato condotto con i valori dei parametri definiti nei paragrafi precedenti, ma con due differenti frequenze di apprendimento: k = 100 e k = 1000 ed è stato fatto 'lavorare' per 15000 periodi (per modificare la frequenza di apprendimento e le probabilità di crossover e mutazione occorre modificare il file ASM.scm).

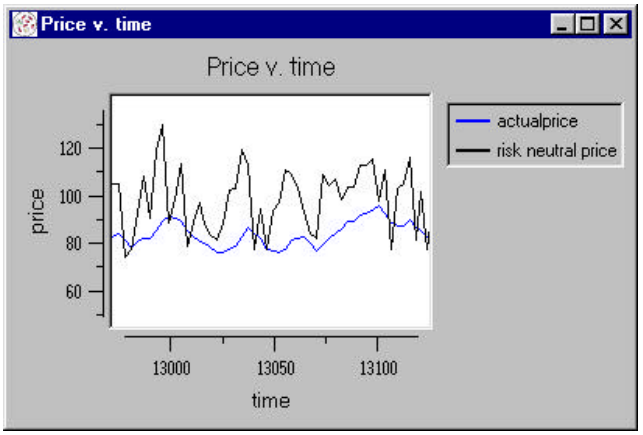
Nel caso di apprendimento 'veloce', la dinamica (relativa ai periodi compresi tra 13000 e 13100) del prezzo effettivo (actualprice) e del prezzo fondamentale (risk neutral price), calcolato come rapporto tra il dividendo e il tasso di interesse è riportata nel grafico che segue:

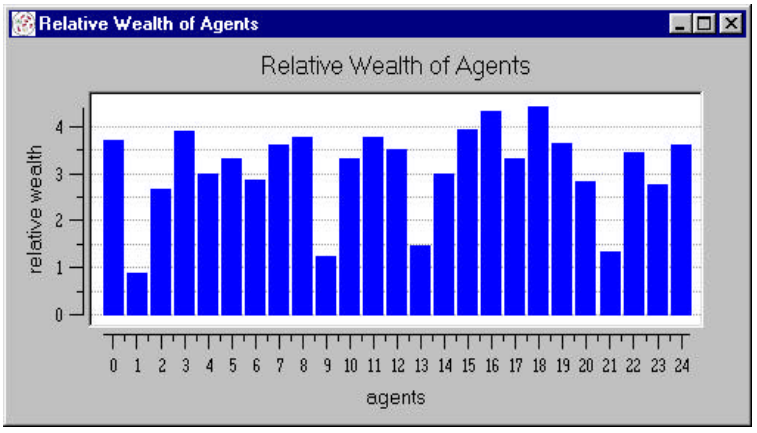


La ricchezza relativa degli agenti, al termine della simulazione, è riportata nel seguente istogramma:



Nel caso di apprendimento 'lento', la dinamica dei prezzi e la ricchezza relativa degli agenti al termine della simulazione sono riportati nei seguenti grafici:



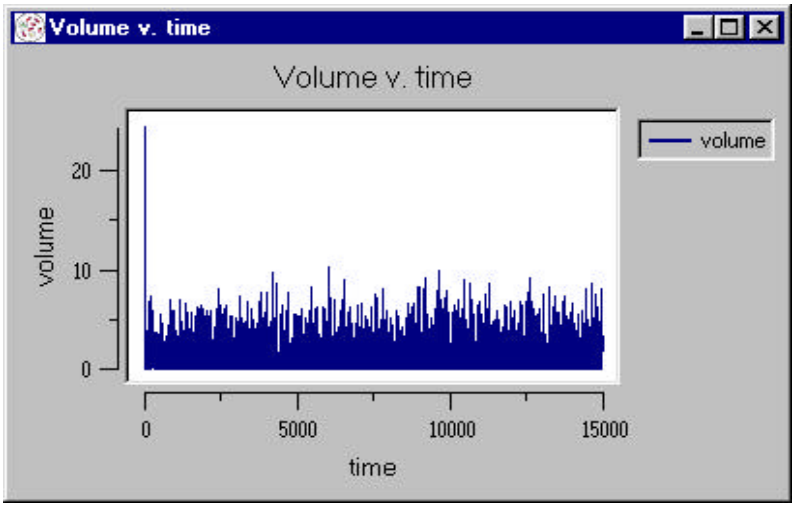


Un risultato interessante che emerge dalla semplice osservazione degli istogrammi è il differente grado di ricchezza relativa che viene raggiunto nei due diversi casi.

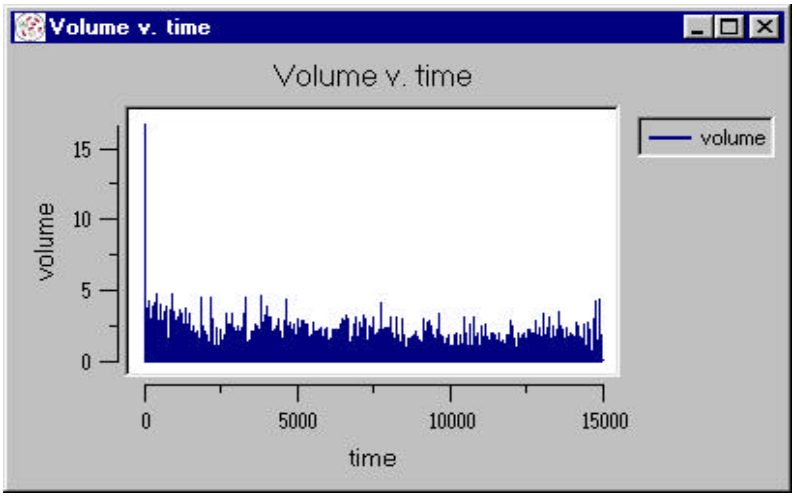
L'apprendimento 'lento' conduce ad un livello di ricchezza relativa superiore a quello che si realizza con un apprendimento più 'veloce'. Con una frequenza di apprendimento pari a 100, solamente il 40% degli agenti raggiunge il livello 3 di ricchezza, e meno del 10% tocca il livello di 3.5. Con una frequenza di apprendimento uguale a 1000, invece, quasi il 70% raggiunge 3, il 40% tocca 3.5 e quasi il 10% supera 4.

Il secondo 'esperimento' è stato condotto, invece, mantenendo costante la frequenza di apprendimento (k=1000), ma facendo variare il parametro t che determina l'orizzonte temporale preso in considerazione dagli agenti nel calcolo dell'accuratezza delle regole di previsione utilizzate. Come è possibile notare dall'osservazione dei grafici riportati di seguito, il mercato finanziario artificiale evidenzia, nei due casi (t: 5 e 250), una notevole differenza nell'ammontare dei volumi trattati:

t = 5



t = 250

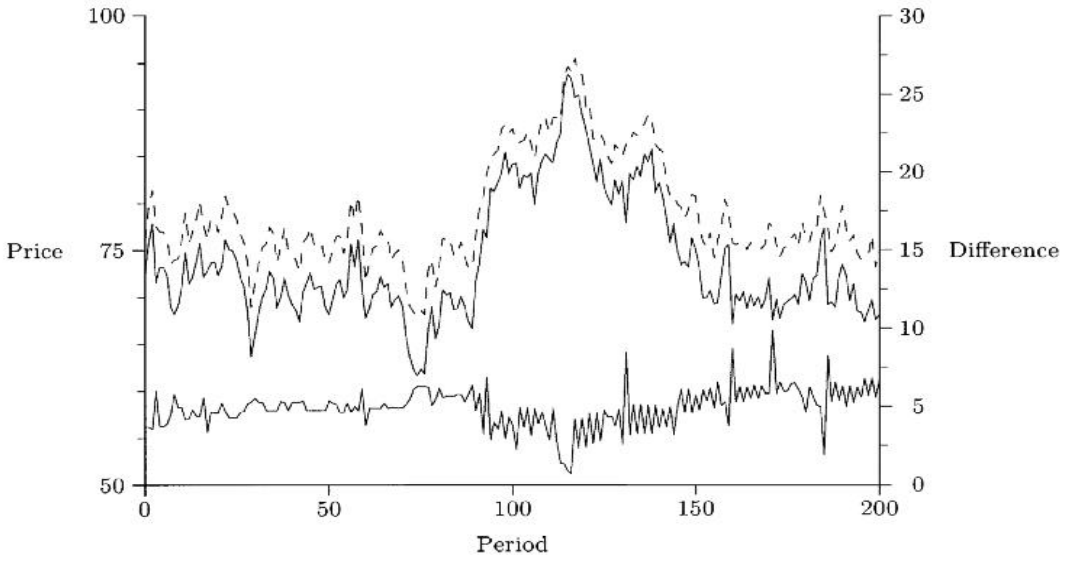


con t pari a 5, i volumi trattati nel mercato arrivano spesso a toccare il livello di 10; con t uguale a 250, invece, i volumi non superano la soglia data dal valore 5.

3.5.9 Analisi del modello

Una attenta analisi dei risultati del modello ASM è stata compiuta da LeBaron (1999), il quale ha effettuato una 'cross-section' di 25 separate simulazioni (ognuna con un differente innesco dei numeri casuali), che è stata ripetuta per due differenti valori della frequenza di apprendimento (k=250 e k=1000).

Ogni simulazione è stata fatta lavorare per 250.000 periodi in fase di 'warm-up', in modo da garantire un sufficiente grado di apprendimento; le serie di tempo, utilizzate per l'analisi, sono quindi state registrate per i successivi 10000 periodi.
Di seguito è riportata la dinamica dei prezzi in una delle 25 simulazioni eseguite con frequenza di apprendimento pari a 250. La linea continua descrive la dinamica del prezzo effettivo, quella tratteggiata il prezzo teorico nell'equilibrio ad



aspettative razionali e la linea in basso nel grafico la differenza tra i due prezzi.
Le serie storiche ottenute da LeBaron evidenziano un insieme di periodi nei quali la differenza tra il prezzo teorico di equilibrio (REE) e il prezzo che si realizza effettivamente non si allontana molto da un valore costante, e periodi, invece, in cui questa differenza fluttua ampiamente. Tale fenomeno risulta una caratteristica comune per molte delle simulazioni eseguite.

Il mercato, in pratica, attraversa fasi nelle quali la valutazione delle attività finanziarie da parte degli agenti avviene in coerenza con l'equilibrio ad aspettative razionali e fasi, invece, nelle quali tale valutazione segue regole differenti.

LeBaron ha eseguito diversi test statistici sulle serie storiche generate dal modello che hanno evidenziato il ruolo estremamente importante giocato dalla frequenza di apprendimento.

Con un apprendimento lento (k = 1000), le serie di tempo dei prezzi risultano simili a quelle ad aspettative razionali e gli agenti imparano ad ignorare i bit di informazione 'inutili'. Una situazione completamente diversa si determina, invece, con una frequenza di apprendimento più veloce (k = 250). In questo caso le serie di tempo si allontanano sensibilmente da quelle dell'equilibrio ad aspettative razionali ed evidenziano caratteristiche simili a quelle che si riscontrano nei mercati finanziari reali.

Con l'apprendimento 'veloce' si assiste ad un raddoppio dell'ammontare delle attività scambiate e gli agenti fanno un maggiore uso dei bit di informazione 'tecnica' e di quelli connessi al rapporto tra il dividendo e il prezzo.

La frazione dei bit relativi alle medie mobili utilizzati per la previsione dei prezzi e dei dividendi si attesta sul 5% nel caso di apprendimento lento e al 20% nel caso di apprendimento più veloce. Occorre tuttavia sottolineare che tali valori, nel caso di apprendimento 'veloce', oscillano sensibilmente e di conseguenza l'uso dell'informazione tecnica non rappresenta una strategia utilizzata regolarmente dagli agenti.

Sia nel caso di apprendimento lento che in quello veloce, il modello evidenzia una elevata persistenza nelle serie dei volumi trattati e una correlazione positiva tra volumi e volatilità. Quest'ultima caratteristica risulta difficilmente giustificabile se analizzata dal punto di vista del singolo agente. All'aumentare della volatilità, infatti, la domanda per l'attività rischiosa, da parte del singolo agente, diminuisce. Il desiderio di riallocare il portafoglio è più grande quando la volatilità è bassa; quando la volatilità è alta gli operatori si 'spaventano' e non vogliono cambiare la propria posizione in risposta a nuove informazioni. Tuttavia ad un livello di analisi macro si può verificare un capovolgimento della relazione valida al livello micro. Occorre infatti analizzare anche la dispersione delle stime di volatilità tra gli agenti. Una elevata dispersione nelle stime della volatilità degli agenti può condurre a valori elevati nei volumi trattati.

Al variare della velocità di apprendimento non si riscontrano grandi differenze per quanto riguarda i parametri di previsione (a_j, b_j, σ^2_j) i cui valori tendono ad avvicinarsi a quelli dell'equilibrio ad aspettative razionali.

Anche questo risultato può apparire anomalo ad una prima analisi, data la diversità riscontrata nell'andamento delle serie di tempo dei prezzi, ma può essere giustificato prendendo nuovamente in considerazione la dispersione dei valori dei parametri di previsione. Questi, infatti, evidenziano una tendenza simile, ma differiscono per quanto riguarda l'ammontare di dispersione. In particolare la dispersione nel caso di apprendimento 'lento' risulta, nel lungo periodo, quasi la metà rispetto a quella che si riscontra con l'apprendimento 'veloce'. Nei primi periodi invece la dispersione è più elevata nel caso di apprendimento 'lento'. Ciò è probabilmente dovuto al fatto che le regole di previsione vengono aggiornate in 'ritardo' rispetto a quanto avviene con l'apprendimento 'veloce'.

3.5.10 Conclusioni

Il risultato più significativo che emerge dall'analisi del modello ASM è il ruolo decisivo giocato dalla frequenza di apprendimento nella spiegazione del comportamento del mercato artificiale.

Il mercato evidenzia due differenti tipi di comportamento, che corrispondono a differenti tassi di applicazione dell'algoritmo genetico all'insieme di regole di previsione degli agenti.

Quando l'algoritmo genetico viene applicato con una frequenza relativamente bassa, le regole di previsione evolvono lentamente, i prezzi sono più stabili, lo sviluppo delle regole è semplice, l'utilizzo di informazioni 'tecniche' è relativamente scarso e i volumi scambiati sono modesti. Il mercato sembra in grado di raggiungere l'equilibrio ad aspettative razionali.

Quando, invece, l'algoritmo genetico viene invocato più frequentemente, le regole di previsione evolvono più velocemente, la varianza delle serie di tempo dei prezzi aumenta, il volume delle attività scambiate cresce, lo sviluppo delle regole è più complesso e gli agenti fanno un maggior uso di informazioni 'tecniche'.

In questo caso, inoltre, la ricchezza accumulata dagli agenti risulta inferiore a quella che si ottiene quando l'evoluzione delle regole avviene più lentamente. Il mercato, in pratica, non riesce a 'stabilizzarsi' e si colloca in una condizione di sub-ottimalità.

Questi risultati, seppure puramente qualitativi, evidenziano chiaramente l'importanza dell'orizzonte temporale degli agenti.

Se gli agenti considerano il mercato come un ambiente in continua evoluzione, saranno portati a rivedere periodicamente le proprie previsioni, che in questo modo includeranno anche elementi 'peculiari' del comportamento individuale, e ciò condurrà ad una generale perdita di profitto.

Se invece gli agenti prendono in considerazione un orizzonte temporale più ampio, gli elementi 'peculiari' del comportamento individuale tenderanno ad annullarsi, e le decisioni saranno così basate solamente sulle informazioni economiche fondamentali; in questo caso il mercato tenderà ad avvicinarsi all'equilibrio ad aspettative razionali.

BIBLIOGRAFIA

- APPLE COMPUTER (1999), *Object-Oriented Programming and the Objective-C language,* Developer's library
- JOHNSON P. (2000), *ASM documentation*, http://artstkmkt.sourceforge.net
- LEBARON B. (2000), *Agent-Based computational finance: suggested readings and early research*, in Journal of Economic Dynamics and Control 24, pp 679-702.
- LEBARON B., ARTHUR W.B., PALMER R. (1999), *Time series properties of an artificial stock market*, in Journal of Economic Dynamics and Control 23, pp 1487-1516.
- LEOMBRUNI R. (1999), *Programmazione ad oggetti & Objective-C*, dispense
- LUNA F., STEFANSSON B. (2000), *Economic Simulation in Swarm: Agent-Based Modelling and Object Oriented Programming*, Kluwer Academic Publishers Boston/Dordrecht/London
- MINAR N., BURKHART R., LANGTON C., ASKENAZI M. (1996), *The Swarm Simulation System: a Toolkit for Building Multi-Agent Simulations*, http://www.santafe.edu/projects/swarm/

Il modello SUM

Introduzione

Il modello SUM, *Surprising (Un)realistic Market*, descritto in questo capitolo è una simulazione basata su agenti che consente di analizzare le micro-fondazioni di un mercato di borsa.
Il modello, sviluppato in Terna (2001), non adotta nessuna ipotesi semplificatrice riguardo il meccanismo di formazione dei prezzi come l'impiego di un banditore per 'pulire' il mercato; riproduce, invece, il funzionamento di un mercato borsistico reale e genera sulla base degli ordini di acquisto e di vendita degli agenti serie di tempo dei prezzi che si modificano continuamente, transizione per transizione.
Gli agenti del modello non sono omogenei, ma differiscono per il tipo di struttura cognitiva; vi sono agenti *no-mind*, dotati di una limitata capacità di adattamento all'ambiente e agenti 'cognitivi' in grado invece di sviluppare autonomamente le proprie regole di comportamento.
Il modello consente di osservare le conseguenze - le proprietà emergenti - delle azioni dei diversi agenti sia sul proprio livello di ricchezza che sulla dinamica del mercato di borsa; l'utente dell'applicazione può variare facilmente le condizioni iniziali (variabili e parametri) del modello e valutare l'effetto delle interazioni tra gli agenti, investigando le problematiche empiriche, come la prevedibilità delle serie di tempo dei prezzi e la volatilità del mercato, che sono difficilmente analizzabili usando i tradizionali strumenti di studio.
La struttura del modello è stata realizzata con Swarm, il quale fornisce gli strumenti necessari - in termini di potenza computazionale e di flessibilità di programmazione - per la gestione della simulazione e per l'analisi dei risultati generati.

4.1 Il mercato azionario come mercato complesso

Uno degli aspetti fondamentali dei mercati finanziari - e di quelli azionari in particolare - è il processo di formazione delle previsioni sull'andamento futuro dei prezzi delle attività finanziarie[83] da parte degli investitori.
Gli operatori professionali – banche, società di intermediazione finanziaria, fondi di investimento - e sempre più anche i piccoli risparmiatori adottano le metodologie dell'analisi *tecnica* e dell'analisi *fondamentale* per prevedere l'andamento futuro dei prezzi e degli indici di mercato.
Le tecniche di analisi fondamentale considerano i dati patrimoniali e finanziari delle singole imprese, attraverso un'analisi approfondita degli indici aziendali (ROE, P/E, *price-to-book*...), degli sviluppi del mercato specifico dell'impresa e delle prospettive di crescita dell'impresa stessa[84].

[83] Le attività finanziarie sono strumenti che permettono di trasferire valore nel tempo; i prezzi delle attività finanziarie riflettono il valore futuro atteso ed incorporano quindi un forte elemento di incertezza, che deriva dal fatto che il valore a cui tali attività saranno vendute non è conoscibile al momento dell'acquisto.

Secondo tale impostazione, i 'valori fondamentali' delle imprese – fatturato, situazione finanziaria, redditività, capacità del management - hanno un forte potere esplicativo; dall'analisi attenta di questi dati è possibile determinare l'evoluzione futura delle imprese, del loro rendimento (utile) e quindi, anche, l'evoluzione del prezzo[85].

L'analisi tecnica, invece, è una metodologia di analisi legata all'osservazione empirica delle serie storiche dei prezzi dei singoli titoli e degli indici di borsa; secondo tale impostazione per cogliere l'evoluzione futura del mercato è necessario imparare a leggere il grafico del prezzo in quanto in esso sono sintetizzate tutte le informazioni rilevanti.

L'analisi tecnica individua gli effetti che si possono attendere con ampia probabilità di accadimento, al presentarsi di ben precise cause, quali la violazione al rialzo o al ribasso di specifici livelli di prezzo; gli analisti tecnici si basano, in sostanza, su una interpretazione 'visiva' delle serie storiche dei prezzi, che vengono ricondotte ad alcune figure geometriche giudicate significative dell'andamento futuro del prezzo di un titolo o dell'indice di borsa[86].

L'adozione di tale metodologia trova una giustificazione nel fatto che spesso, per periodi di tempo non troppo lunghi i prezzi sono soggetti a movimenti irrazionali e soprattutto a giudizi soggettivi che nulla o poco hanno a che vedere con la situazione reale. Il mercato, in pratica, tiene conto solo in parte dei valori 'fondamentali' (le dinamiche dei mercati finanziari più che riflettere le condizioni dei fondamentali economici sottostanti, riflettono i comportamenti degli operatori presenti sul mercato[87]).

Nell'analisi dei mercati finanziari, la teoria economica fa ricorso al principio dell'efficienza; secondo tale principio, legato all'ipotesi di aspettative razionali, i mercati raccolgono tutta l'informazione disponibile riguardo al futuro e la elaborano in modo che le valutazioni del futuro e del rischio siano corrette; in tali condizioni nessun investitore dovrebbe preoccuparsi di reperire ulteriori informazioni; il mercato le rivela attraverso i prezzi dei titoli[88].

Nella realtà, tuttavia, si verificano forti deviazioni dal principio di efficienza: il mercato, spesso, reagisce eccessivamente a nuove informazioni, i prezzi delle azioni mostrano andamenti caratterizzati da improvvise impennate e da crolli inattesi.

L'idea di efficienza del mercato è infatti legata, come affermato in precedenza, all'ipotesi di aspettative razionali; si assume cioè che, in media, le previsioni degli agenti siano corrette, che gli agenti non commettano errori di previsione sistematici. Nella realtà, però, gli agenti non sono completamente

---

[84] L'analisi fondamentale può essere, anche, di tipo macroeconomico; in questo caso, al fine di determinare l'evoluzione futura del mercato, si considerano i dati relativi alla politica economica, monetaria, dei redditi e in generale alle variabili economiche aggregate.

[85] L'analisi fondamentale è rivolta alla previsione; si tratta, cioè, di individuare il 'vero' valore di un'azione e di acquistare o vendere a seconda che il vero valore sia minore o maggiore di quello di mercato

[86] L'analisi tecnica fa uso di diverse metodologie (grafiche, algoritmiche, cicliche, 'mistiche') che hanno come comune denominatore il presupposto che il comportamento degli operatori finanziari sia perlopiù di tipo imitativo.

[87] Il sempre più diffuso utilizzo dell'analisi tecnica da parte degli operatori professionali ne uniforma i comportamenti indirizzando i mercati proprio verso i risultati individuabili con questo tipo di approccio.

[88] Secondo il principio di efficienza, i mercati finanziari analizzano continuamente le informazioni e reagiscono senza ritardo alle informazioni considerate rilevanti.

razionali, non utilizzano in modo efficiente tutta l'informazione disponibile sul futuro, ma operano nel mercato sulla base di regole 'empiriche' di comportamento[89].

I mercati di borsa sono mercati in cui operano una moltitudine di operatori eterogenei ognuno dei quali dispone di proprie aspettative sull'evoluzione futura dei prezzi; il modo in cui ogni investitore valuta la situazione del mercato e la sua possibile evoluzione è largamente soggettiva.

Nelle condizioni di forte incertezza tipiche dei mercati azionari, gli investitori si fanno, spesso, guidare nelle loro decisioni da regole più o meno razionali, da entusiasmi o crisi di panico passeggere, dal 'comportamento del branco', invece che da valutazioni 'fondamentali'.

Un chiaro esempio di questo tipo di comportamento, è la dinamica del prezzo delle azioni ordinarie della Tiscali riscontrata nel corso del 1999-2000; sull'onda dell'entusiasmo per i titoli tecnologici, determinato dall'andamento continuamente crescente dell'indice Nasdaq, tali azioni hanno raggiunto livelli di prezzo molto superiori al loro valore fondamentale, per poi ridimensionarsi dopo il parziale crollo del Nasdaq nella primavera del 2000.

Gli investitori hanno inoltre una naturale tendenza a vedere analogie anche tra situazioni che in realtà presentano forti differenze, e spesso interpretano in modo errato il significato di notizie e avvenimenti. La caduta dei mercati durante la crisi del Golfo del 1990 per il timore di nuova crisi petrolifera è un esempio di questo tipo di errori di valutazione (*representativeness bias*).

L'invasione del Kuwait (e dei suoi giacimenti petroliferi) da parte dell'Iraq e il successivo embargo sul petrolio iracheno determinarono negli investitori la paura di una nuova crisi energetica e la preoccupazione di un brusco rialzo dei prezzi e dell'inflazione.

Larga parte degli operatori, dei media, e degli investitori videro in questa situazione forti analogie con le crisi petrolifere del 1973 e del 1980; in realtà le crisi presentavano forti differenze che erano chiaramente visibili; i fondamentali economici erano, infatti, completamente differenti.

Nel 1990 l'economia mondiale era in una situazione di abbondanza di materie prime; non stava affrontando un periodo di carenza come nelle due precedenti occasioni; i prezzi del petrolio non triplicarono, ma aumentarono del 30%. Inoltre, i paesi appartenenti all'OPEC non operarono congiuntamente un aumento dei prezzi, ma si impegnarono a far fronte all'eventuale ulteriore incremento dei prezzi del petrolio.

Ciononostante le aspettative negative ed il clima di panico che investì numerosi operatori determinarono forti conseguenze sui mercati internazionali.

Come affermato in Keynes (1936) la valutazione convenzionale del prezzo è, in sostanza, 'il risultato della psicologia di massa di un gran numero di individui ignoranti che è soggetta a variazioni violente in seguito a improvvise fluttuazioni dell'opinione (...). Il mercato andrà soggetto a ondate di ottimismo e di pessimismo, irragionevoli e pur tuttavia in un certo senso legittime qualora non esista alcuna base solida per un calcolo ragionevole'.

Nella 'Teoria generale dell'occupazione, dell'interesse e della moneta' Keynes pone l'attenzione su un altro aspetto che determina l'allontanamento dall'ipotesi di mercati finanziari efficienti: il fatto che l'orizzonte temporale dell'investitore professionale è dominato dal breve periodo[90].

[89] Non tutti gli operatori operano in modo razionale, come ipotizzato dai modelli economici tradizionali; solo una parte di essi è informata e riesce a valutare il reale valore dei titoli.

Secondo Keynes 'si potrebbe supporre che la concorrenza fra esperti operatori professionali (...) corregga gli sbandamenti dell'individuo ignorante (...). Si verifica invece che le energie e l'abilità dell'investitore e dello speculatore professionale si esercitano principalmente in altre direzioni. Infatti la maggioranza di queste persone si occupano soprattutto non già di compiere migliori previsioni a lungo termine sul rendimento probabile di un investimento (...), bensì di prevedere variazioni della base convenzionale di valutazione con un breve anticipo rispetto al grosso pubblico. A loro non interessa ciò che un investimento vale realmente (...), bensì il livello cui il mercato lo valuterà sotto l'influenza della psicologia di massa (...). Così l'investitore professionale è costretto a occuparsi di prevedere quel genere di variazioni...nelle notizie e nell'ambiente, che l'esperienza indica maggiormente atte a influenzare la psicologia collettiva del mercato'.

In definitiva, la dinamica del mercato altro non è che la conseguenza di innumerevoli decisioni di investimento prese giorno per giorno da un elevato numero di investitori; dall'interazione tra le strategie dei diversi operatori, dotati di propri modelli di valutazione del mercato e aspettative di evoluzione futura, emerge un mercato complesso, il cui andamento è difficilmente prevedibile a priori.

L'obiettivo che si pone il modello *SUM* è quello di analizzare come la natura degli agenti che operano nel mercato e la loro interazione, possa influire sui meccanismi di formazione del prezzo e sui livelli del prezzo stesso; in questo modo è possibile cogliere l'aspetto complesso del mercato che i modelli tradizionali non sono in grado di rappresentare.

4.2 La struttura del mercato artificiale di borsa: il *book*

Il cuore del modello SUM è rappresentato da una struttura computazionale che riproduce, nelle linee fondamentali, il funzionamento del *book* elettronico del mercato telematico azionario italiano (MTA).

Come affermato nel regolamento dei mercati organizzati e gestiti dalla Borsa Italiana S.p.a (2000), il mercato telematico azionario (MTA) è il comparto di mercato in cui si negoziano azioni, obbligazioni convertibili, warrant, diritti di opzioni e quote di fondi mobiliari ed immobiliari chiusi quotati in borsa.

Le negoziazioni di tali titoli si possono svolgere secondo le seguenti modalità:

a) asta di apertura, articolata nelle fasi di determinazione del prezzo teorico di apertura (pre-apertura); validazione del prezzo teorico di apertura (validazione); conclusione dei contratti (apertura);
b) negoziazione continua.

La volontà negoziale degli operatori viene espressa mediante proposte di negoziazione, che contengono informazioni relative allo strumento finanziario da negoziare, alla quantità, al tipo di operazione, al tipo di

[90] I fondi comuni di investimento e i responsabili delle gestioni patrimoniali devono dare conto a scadenze fisse dei risultati conseguiti e ciò inevitabilmente rende problematica una politica di investimento di lungo termine. Gli investitori istituzionali per ottenere prestazioni soddisfacenti in termini di redditività sono costretti a vendere titoli con buoni fondamentali se per qualsiasi motivo, anche errato, il loro prezzo scende oppure a comprare un titolo 'cattivo' se il prezzo sale, in modo da tener testa all'andamento generale del mercato.

conto ed alle eventuali condizioni sul prezzo; tutte le operazioni di immissione, modifica e cancellazione degli ordini possono essere effettuate sia nella fase di pre-apertura, sia in quella di contrattazione continua.

Le proposte di vendita e di acquisto di azioni - articolo 4.1.3 del regolamento della Borsa Italiana - sono automaticamente ordinate in base al prezzo – decrescente se in acquisto e crescente se in vendita – nonché, a parità di prezzo, in base alla priorità temporale determinata dall'orario di immissione della proposta[91]; le proposte modificate perdono la priorità temporale acquisita, se la modifica implica un aumento della quantità o una variazione del prezzo di riserva.

Il book artificiale del modello SUM, pur mantenendo l'impostazione generale del *book* ufficiale di Borsaitalia S.p.a., è stato semplificato sotto alcuni aspetti, funzionali rispetto allo sviluppo del modello.

Le semplificazioni adottate riguardano principalmente la fase di pre-apertura: il prezzo di apertura non viene ottenuto sulla base della fase di pre-apertura[92], ma si considera come informazione rilevante l'ultimo prezzo della giornata precedente; durante questa fase non si determina nessun prezzo iniziale e non vengono effettuate le fasi di validazione e quella di conclusione dei contratti (apertura).

Gli ordini immessi in pre-apertura dagli agenti del modello[93] non sono conclusi a un prezzo definito, ma vengono girati direttamente alla fase di negoziazione continua.

La funzione principale della fase di preapertura nel mercato di borsa artificiale è, in pratica, quella di far iniziare le contrattazioni con un *book* non vuoto, in modo da evitare anomalie di prezzo ad ogni inizio di giornata.

Nella fase di negoziazione continua del mercato artificiale, le proposte - che possono essere immesse solo con limite di prezzo - sono ordinate in modo decrescente se in acquisto ed in modo crescente se in vendita; a parità di prezzo sono ordinate, come avviene per il *book* ufficiale, secondo la priorità temporale (tutti gli agenti trattano il lotto minimo, anche se ogni agente può immettere più ordini).

Durante la negoziazione continua la conclusione dei contratti avviene, per le quantità disponibili, mediante abbinamento automatico di proposte di segno contrario presenti nel mercato così come indicato di seguito:

- l'immissione di una proposta in acquisto determina l'abbinamento con una proposta di vendita avente prezzo inferiore o uguale a quello della proposta immessa;
- l'immissione di una proposta in vendita determina l'abbinamento con una proposta di acquisto avente prezzo superiore o uguale a quello della proposta immessa;
- per ogni contratto concluso mediante abbinamento automatico, il prezzo è pari a quello della proposta avente priorità temporale superiore.

Il book – che viene 'pulito' all'inizio di ogni giornata di borsa - esegue immediatamente l'ordine ricevuto se vi è una controparte nel suo 'registro' (*log*); in caso contrario registra separatamente le proposte di negoziazione di acquisto e quelle di vendita.

---

[91] Le proposte di acquisto o vendita di azioni, obbligazioni convertibili e warrant possono avere per oggetto quantitativi pari o multipli del lotto minimo di negoziazione (proposte "intere") o quantitativi minori del lotto minimo di negoziazione ("spezzature").

[92] Il prezzo teorico di apertura è il prezzo al quale è negoziabile il maggior quantitativo di strumenti finanziari; tale quantitativo, formato da proposte intere o da spezzature, è sempre pari o multiplo del lotto minimo di negoziazione (articolo 4.1.6, punto 1, del regolamento di Borsa).

4.3 Tipologie di agenti

Nel modello SUM possono operare le seguenti tipologie di agenti:

- Agenti casuali (*randomAgent*): sono agenti "*no-mind*" privi di una visione globale del mercato; conoscono solo l'ultimo prezzo eseguito, decidono se acquistare o vendere casualmente (con probabilità pari a 0.5) e fissano il loro limite di prezzo moltiplicando l'ultimo prezzo eseguito per un coefficiente casuale compreso tra i valori modificabili dall'utente (*minCorrectingCoeff* + *asymmetricRange*) e (*maxCorrectingCoeff* + *asymmetricRange*); rappresentano l'agente base del modello[94]; il loro ruolo è assimilabile a quello degli operatori *noise traders*;

- Agenti che imitano il mercato (*marketImitatingAgent*): sono agenti che decidono di comperare o di vendere non sulla base di una scelta casuale, ma di comperare con probabilità *asymmetricBuySellProb* se il prezzo medio da t-2 a t-1 è salito, oppure di vendere con la stessa probabilità se il prezzo medio da t-2 a t-1 è sceso; come gli agenti casuali fissano il limite di prezzo moltiplicando l'ultimo prezzo eseguito per un coefficiente casuale;

- Agenti che imitano localmente (*LocallyImitatingAgent*): sono agenti che decidono di comperare o vendere a seconda di che cosa hanno fatto in maggioranza gli ultimi *localHistoryLength* agenti[95]; se prevalgono le decisioni di acquisto (vendita), l'agente - con probabilità *asymmetricBuySellProb* – acquista (vende); il limite di prezzo viene fissato come per gli agenti casuali;

- Agenti Stop loss (*StopLossAgent*): sono agenti che possono operare con due modalità possibili:
  - senza memoria del passato (non conta la posizione speculativa dell'agente);
  - con memoria del passato (conta la posizione speculativa '*long*' o '*short*' dell'agente);

  nel primo caso se il prezzo corrente, cioè l'ultimo prezzo eseguito nel giorno *t*, è aumentato (diminuito) rispetto al prezzo medio del periodo *t-stopLossInterval* ad un tasso più grande (o uguale)

---

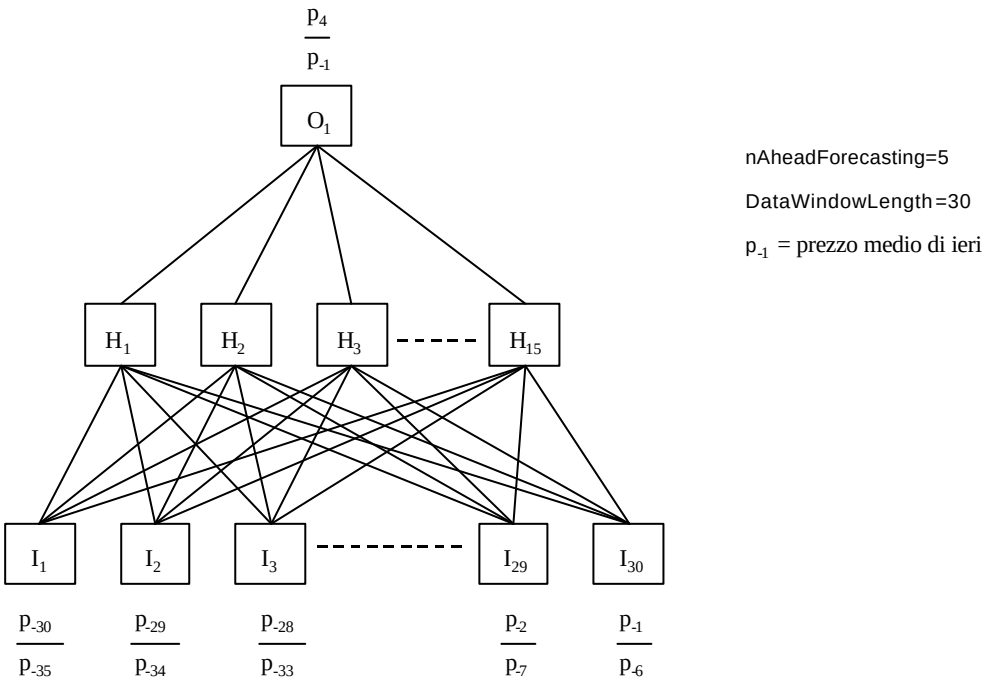
[93] Le proposte sono immesse dagli operatori con una probabilità pari al 5% (modificabile dall'utente dell'applicazione).
[94] Tutti gli altri agenti ereditano le funzionalità base dei *randomAgent*.
[95] La localHistory è continua, da giorno a giorno, non essendo mai azzerata; non include chi non opera (ciò in particolare è necessario nella preapertura).

del parametro *maxLossRate* allora l'agente compra (vende) al prezzo corrente; nel secondo caso - più realistico - se il prezzo corrente, cioè l'ultimo prezzo eseguito nel giorno *t*, è aumentato (diminuito) rispetto al prezzo medio del periodo *t-stopLossInterval* ad un tasso più grande (o uguale) del parametro *maxLossRate*, l'agente compra (vende) se ha una posizione 'short' ('long'); se il prezzo corrente non supera lo stop loss, l'agente si comporta come un agente casuale; il ruolo di questi agenti, e di quelli imitatori descritti precedentemente, può essere assimilato al ruolo svolto nel mercato reale dagli agenti che utilizzano i metodi dell'analisi tecnica per la determinazione della loro posizione speculativa;

- Agente neurale (*ForecastingAgent*): tale agente, che non opera direttamente sul mercato, produce - sulla base di una rete neurale artificiale - previsioni del prezzo del mercato *n* giorni avanti (*nAheadForecasting*) rispetto all'ultima media giornaliera; le previsioni vengono espresse sotto forma di indice.

  La rete neurale artificiale, che rappresenta l'agente:

  - ha *dataWindowLength* nodi input; gli input sono costituiti da indici di variazione dei prezzi medi giornalieri di ogni giorno rispetto al giorno *-nAheadForecasting*;
  - ha (*dataWindowLength/2)* nodi hidden[96];
  - ha un solo output rappresentato dall'indice di previsione del prezzo *nAheadForecasting* giorni avanti;



[96] Le altre caratteristiche di funzionamento della rete neurale vengono illustrate nel prossimo paragrafo.

nell'esempio riportato in figura, la rete neurale - sulla base di una finestra di trenta dati continuamente aggiornati, costituiti da indici di variazione di ogni giorno rispetto allo stesso giorno della settimana precedente – fornisce una previsione del mercato a cinque giorni.

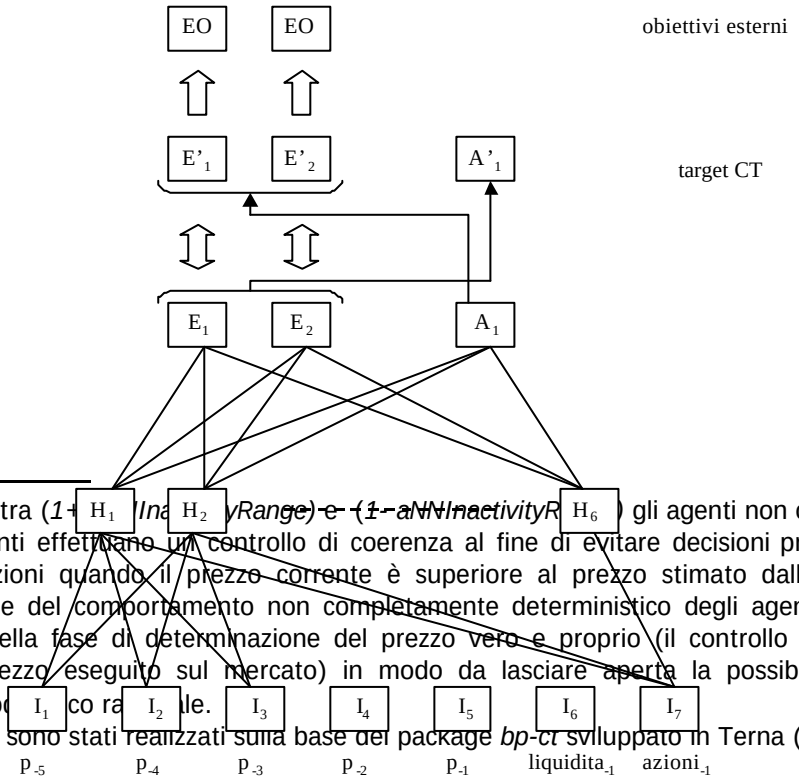
Il ruolo dell'agente neurale può essere assimilato a quello delle istituzioni finanziarie che forniscono previsioni sull'andamento futuro del mercato (in sostanza tale agente utilizza la rete neurale come uno strumento econometrico);

- Agenti che applicano la previsione neurale (*ANNForecastAppAgent*): questi agenti, che operano ogni giorno con probabilità *aNNForecastAppAgentActDailyProb*, comprano o vendono a seconda che il valore dell'indice di previsione fornito dall'agente neurale sia maggiore di (*1+ aNNInactivityRange*) o minore di (*1-aNNInactivityRange)*[97]; se la previsione dell'agente neurale è maggiore di (*1+ aNNInactivityRange)*, gli agenti *ANNForecastAppAgent* acquistano con probabilità pari a *asymmetricBuySellProb*[98];

- Agenti 'cognitivi' di tipo A (*BPCTAgentA*)[99]: sono agenti, caratterizzati da una complessa struttura cognitiva, realizzati secondo la metodologia dei 'cross-target' illustrata nel secondo capitolo.

  La rete neurale che rappresenta tali agenti:

  - ha sette nodi input: le medie dei prezzi dal giorno t-5 al giorno t-1, la liquidità dell'agente al termine del giorno t-1, le azioni dell'agente al termine del giorno t-1;
  - ha sei nodi hidden;
  - ha due output sul lato degli effetti (liquidità e quantità di azioni) e un output dal lato delle azioni (la decisione di acquisto/vendita);



[97] Se l'indice è compreso tra (*1+ aNNInactivityRange)* e (*1- aNNInactivityRange*) gli agenti non operano.

[98] Prima di agire gli agenti effettuano un controllo di coerenza al fine di evitare decisioni prive di senso che portino ad acquistare azioni quando il prezzo corrente è superiore al prezzo stimato dalla rete neurale; tuttavia, in considerazione del comportamento non completamente deterministico degli agenti tale controllo viene effettuato prima della fase di determinazione del prezzo vero e proprio (il controllo viene effettuato sulla base dell'ultimo prezzo eseguito sul mercato) in modo da lasciare aperta la possibilità che ex-post l'agente si comporti in modo contrario rispetto alle.

[99] Gli agenti 'cross-target' sono stati realizzati sulla base del package *bp-ct* sviluppato in Terna (2000).

Gli agenti sviluppano la coerenza interna tra l'azione di acquisto/vendita e le congetture sugli effetti relativi alla liquidità e alla quantità di azioni.

Gli agenti possono, inoltre, perseguire 3 differenti tipi di obiettivi esterni (EO):

- accrescere la liquidità;
- accrescere la quantità di azioni;
- accrescere entrambi[100].

Gli EO dei due effetti sono determinati applicando un delta fisso (modificabile dall'utente) al valore dell'effetto nel giorno precedente; il delta è espresso in unità di azioni; se applicato alla liquidità è moltiplicato per il prezzo medio operativo calcolato come media dei prezzi conclusi dall'agente.

Gli agenti cognitivi di tipo A, e quelli cognitivi di tipo B illustrati di seguito, utilizzano gli obiettivi esterni in modo innovativo rispetto allo schema classico dei CT descritto nel secondo capitolo; gli EO, infatti, non modificano gli output della rete per le variabili cui è collegato l'obiettivo esterno, ma sono utilizzati direttamente 'al volo' nella determinazione dell'azione acquisto/vendita sostituendo l'output relativo alla congettura dell'effetto.

- Agenti 'cognitivi' di tipo B (*BPCTAgentB*): anch'essi realizzati con la tecnica dei 'cross-target', rappresentano un'evoluzione dei più semplici agenti cognitivi di tipo A;

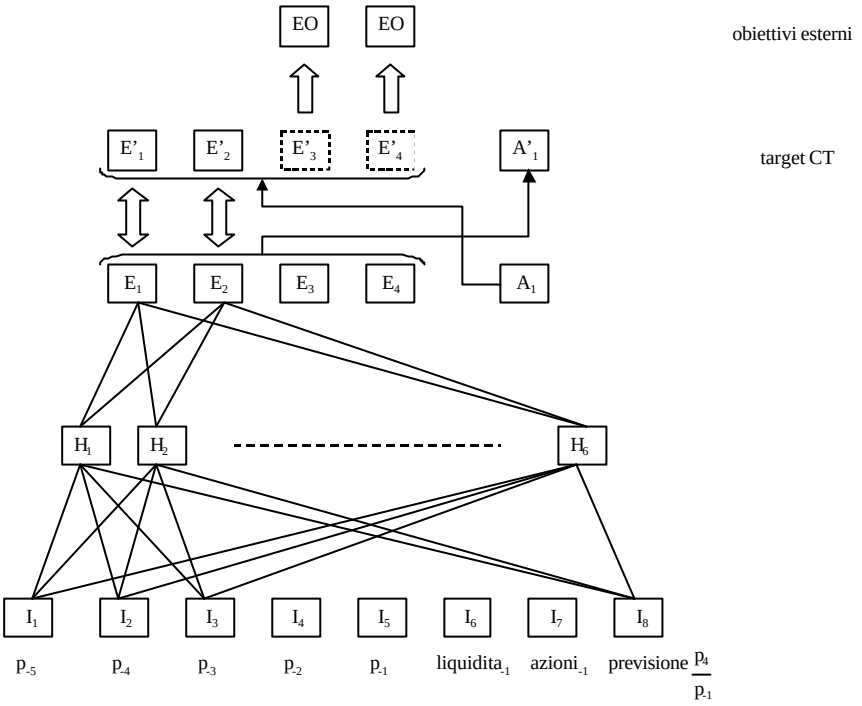
  La rete neurale che rappresenta tali agenti:

  - ha in input otto nodi: i prezzi medi dal giorno t-5 al giorno t-1, la liquidità dell'agente al termine del giorno t-1, le azioni dell'agente al termine del giorno t-1 e la previsione, formulata dall'agente neurale, del prezzo per il giorno t+n-1 espressa come indice (rispetto al valore in t-1);
  - ha sei nodi hidden;
  - ha quattro output sul lato degli effetti[101](liquidità, quantità di azioni, ricchezza dell'agente valutata al prezzo di chiusura e ricchezza dell'agente valutata sulla base del prezzo previsto dall'agente neurale) e un output dal lato delle azioni (la decisione di acquisto/vendita); l'azione dell'agente è rappresentata da un numero reale compreso nel range ± *maxOrderQuantity*; se tale valore è positivo l'agente decide di acquistare azioni, se, invece, è negativo vende (se il valore è pari a zero l'agente non opera sul mercato); la quantità acquistata o venduta è determinata arrotondando il valore dell'azione al valore intero immediatamente superiore.

  L'obiettivo degli agenti è quello di sviluppare la coerenza interna tra l'azione di acquisto/vendita e le congetture sugli effetti. L'agente cognitivo, mediante il metodo dei CT, sviluppa con l'apprendimento, la capacità di valutare in modo coerente le azioni da compiere; in altre parole, l'agente è in grado di

[100] Sebbene quest'ultimo tipo di obiettivo esterno non sia realizzabile contemporaneamente; viene comunque previsto per completezza dello schema architetturale.
[101] Gli effetti vengono valutati alla fine del giorno t.

decidere quale azione eseguire per ottenere un determinato risultato e di comprendere le conseguenze delle azioni compiute.



I target degli effetti - cioè gli effetti reali dell'azione compiuta - vengono costruiti in coerenza con l'output della rete relativo all'azione (l'azione attuata: acquisto o vendita), in modo da sviluppare la capacità dell'agente di stimare gli effetti dell'azione che sta decidendo; il target dell'azione - l'azione necessaria per 'corrispondere' agli effetti congetturati - invece, viene costruito in coerenza con gli output della rete concernenti gli effetti, in modo da sviluppare la capacità dell'agente di decidere una azione che produca i risultati previsti.

I target degli effetti sono determinati sulla base delle seguenti regole contabili:

- target della liquidità (E'1 nel grafico): liquidità del giorno precedente più la variazione di liquidità dovuta alla decisione di acquisto/vendita presa dall'agente nel giorno corrente;
- target della quantità di azioni (E'2): quantità di azioni del giorno precedente più la variazione di azioni dovuta alla decisione di acquisto/vendita presa dall'agente nel giorno corrente;
- target della ricchezza valutata al prezzo di chiusura (E'3): liquidità più la quantità di azioni moltiplicata per l'ultimo prezzo della giornata;
- target della ricchezza valutata al prezzo previsto (E'4): liquidità più la quantità di azioni moltiplicata per il prezzo di chiusura al momento della previsione e per l'indice della previsione.

Per la determinazione del target dell'azione (A'1), invece, si procede al calcolo delle seguenti correzioni:

- correzione per la liquidità: CL = - (E1 - E'1) / PMO[102]; se la liquidità dell'agente è inferiore a quanto previsto, l'agente deve diminuire l'acquisto di azioni o aumentare le vendite;
- correzione per la quantità di azioni: CA = E2 – E'2; se la quantità di azioni realmente posseduta dall'agente è inferiore a quanto egli congetturava, l'agente deve acquistare azioni;
- correzione per la ricchezza valutata al prezzo di chiusura: CRPC = (E3 – E'3) / (PC-PMO)[103]; se la ricchezza realmente posseduta è inferiore a quella congetturata (output della rete o EO), l'agente deve aumentare l'acquisto (o diminuire la vendita) di azioni se il prezzo di chiusura è maggiore del prezzo medio operativo; se, invece, il prezzo di chiusura è inferiore al prezzo medio operativo, l'agente deve incrementare le vendite (o diminuire l'acquisto) di azioni[104];
- correzione per la ricchezza valutata al prezzo previsto: CRPP = (E4 – E'4) / (PP-PMO)[105]; se la ricchezza (valutata al prezzo previsto) realmente posseduta è inferiore a quella congetturata (output della rete o EO), l'agente deve aumentare l'acquisto (o diminuire la vendita) di azioni se il prezzo previsto è maggiore del prezzo medio operativo; se, invece, il prezzo previsto è inferiore al prezzo medio operativo, l'agente deve incrementare le vendite (o diminuire l'acquisto) di azioni;

la correzione più elevata in valore assoluto viene sommata (se positiva, sottratta se negativa) all'output A1 per la determinazione del target dell'azione.

Gli agenti possono perseguire tre differenti tipi di obiettivi esterni (EO):

- accrescere la ricchezza valutata al prezzo di chiusura;
- accrescere la ricchezza valutata al prezzo previsto;
- accrescere entrambe le misure di ricchezza.

Gli EO dei due effetti ricchezza sono determinati sommando al valore dell'effetto del giorno stesso, determinato sulla base dello sviluppo dei cross-target un ammontare fisso pari a delta (espresso in unità di ricchezza, modificabile mediante la 'probe' del modelswarm)[106].

- Agente Astrologo (*ASTROForecastingAgent*): tale agente, che opera come l'*ANNForecastingAgent* all'inizio di ogni giornata di borsa, formula una previsione, espressa sotto forma di indice, sull'andamento del mercato per gli *n* giorni successivi (n=*astroForecastForDays,* modificabile

[102] PMO = Prezzo medio operativo, calcolato come media dei prezzi conclusi dall'agente, PC = prezzo di chiusura (ultimo prezzo concluso nel mercato), PP = prezzo previsto sulla base della previsione neurale.
[103] Se viene applicato l'EO la correzione è pari a (EO – E'3) / (PC-PMO)
[104] Se non viene applicato l'EO si può verificare un 'effetto paradosso' nel caso in cui l'output della rete sia inferiore al target dell'effetto: se, infatti, la congettura sulla ricchezza valutata al prezzo di chiusura è inferiore al valore reale (l'agente congettura di essere meno ricco di quanto in realtà è), allora l'agente aumenta l'acquisto, o diminuisce le vendite, di azioni quando il prezzo di chiusura è inferiore al prezzo medio operativo.
[105] Se viene applicato l'EO la correzione è pari a (EO – E'4) / (PP-PMO)
[106] Si tratta di una impostazione diversa rispetto agli agenti cognitivi di tipo A: gli EO, non si appoggiano al valore di ieri, ma a quello di oggi, incrementandolo.

dall'utente mediante la 'probe' del modelswarm); il valore dell'indice di previsione dell'astrologo, determinato casualmente (varia tra 0.7 e 1.3), individua cinque diverse situazioni:

- se l'indice è compreso tra 1 e 1.2, il mercato, nei prossimi *astroForecastForDays* giorni, crescerà;
- se l'indice è maggiore di 1.2, il mercato registrerà un forte incremento;
- se l'indice è uguale a 1, il mercato sarà stabile;
- se l'indice è compreso tra 0.8 e 1, il mercato registrerà una diminuzione;
- se, infine, l'indice è inferiore a 0.8, il mercato subirà forti perdite.

La previsione rimane costante per gli *astroForecastForDays* giorni; ogni *astroForecastForDays* giorni l'astrologo riformula, in modo casuale, una previsione per i successivi *astroForecastForDays* giorni (oltre alla previsione 'standard' l'astrologo può, su richiesta degli agenti *ASTROForecastAppAgent*, personalizzare la previsione sull'andamento futuro del mercato).

- Agenti che applicano la previsione dell'astrologo(*ASTROForecastAppAgent*): tali agenti, che agiscono ogni giorno con probabilità *astroForecastAppAgentActDailyProb*, operano nel seguente modo:
  - se l'indice di previsione dell'astrologo è compreso tra 1 e 1.2, l'agente compra azioni per un quantitativo compreso tra 1 e maxOrderQuantity;
  - se l'indice è maggiore di 1.2, l'agente compra azioni per il quantitativo massimo consentito;
  - se l'indice è pari a 1, l'agente non opera sul mercato;
  - se l'indice è compreso tra 0.8 e 1, l'agente vende azioni per un quantitativo compreso tra 1 e maxOrderQuantity;
  - se, infine, l'indice è inferiore a 0.8, l'agente vende azioni per un quantitativo pari a maxOrderQuantity.

  Gli agenti *ASTROForecastAppAgent* oltre a seguire fedelmente la previsione formulata dall'astrologo, possono, con probabilità del 5%, richiedere una previsione 'personalizzata' sull'andamento futuro del mercato; in questo modo viene attenuata leggermente l'omogeneità di comportamento degli agenti, introducendo un fattore di variabilità nel loro comportamento.

  Come gli agenti casuali, gli *ASTROForecastAppAgent* fissano il limite di prezzo moltiplicando l'ultimo prezzo eseguito per un coefficiente casuale[107].

Tali agenti, che si differenziano da quelli random in quanto non decidono individualmente se acquistare o vendere sulla base di una scelta casuale, sono stati introdotti nel modello, un po' provocatoriamente - sulla base della considerazione che nei mercati reali i prezzi sono, a volte, soggetti a movimenti irrazionali frutto di giudizi soggettivi che nulla o poco hanno a che fare con la situazione reale - al fine di analizzare le conseguenze, sul mercato di borsa, del comportamento di un gruppo 'omogeneo' di agenti che operano sulla base di valutazioni indipendenti dalla dinamica del mercato.

[107] Quando l'agente decide di comprare (vendere), *buySellSwitch* è pari a 1 (0) e non come per gli agenti imitatori pari a *asymmetricBuySellProb (1-asymmetricBuySellProb)*; in sostanza l'agente segue sempre fedelmente l'astrologo; non c'è probabilità di agire in contrasto.

L'introduzione di questa tipologia di agenti non appare poi così provocatoria se si pensa all'utilizzo nel mercato reale di metodologie come quella dell'analisi 'mistica' (o non convenzionale); tali tecniche – utilizzate da numerosi analisti e trader che le rivestono in seguito con valutazioni di tipo fondamentale – si basano, infatti, su presupposti astrologici (i giorni di luna piena corrisponderebbero a massimi del mercato e quelli di luna nuova a minimi).

- Agenti 'autoregressivi' (*AR2Agent*): sono agenti che operano sul mercato sulla base di un proprio modello dell'evoluzione futura dei prezzi; secondo tali agenti l'andamento del prezzo è descritto da una equazione autoregressiva di secondo ordine del tipo:

  $p_t = a\, p_{t-1} + b\, p_{t-2}$;

L'agente *AR2Agent* sulla base dei prezzi medi del periodo *t-1* e *t-2* stima il prezzo delle azioni al tempo *t*; se il prezzo stimato è maggiore dell'ultimo prezzo eseguito, l'agente acquista azioni; se, invece, il prezzo stimato è inferiore l'agente vende azioni.

Poiché il prezzo di acquisto/vendita viene determinato – come per gli agenti casuali – moltiplicando l'ultimo prezzo eseguito per un coefficiente casuale, l'agente *AR2Agent* effettua un controllo di coerenza al fine di evitare decisioni prive di senso che lo portino ad acquistare (vendere) ad un prezzo superiore (inferiore) a quello previsto.

Inoltre, per evitare che tutti gli agenti *AR2Agent* abbiano uno stesso modello autoregressivo e che il loro comportamento sia così troppo omogeneo i parametri *a* e *b* che guidano le decisioni degli agenti vengono determinati sulla base dei valori minimi e massimi indicati nella 'probe' del modelswarm (modificabili dall'utente dell'applicazione).

4.4 Il modello SUM con SWARM

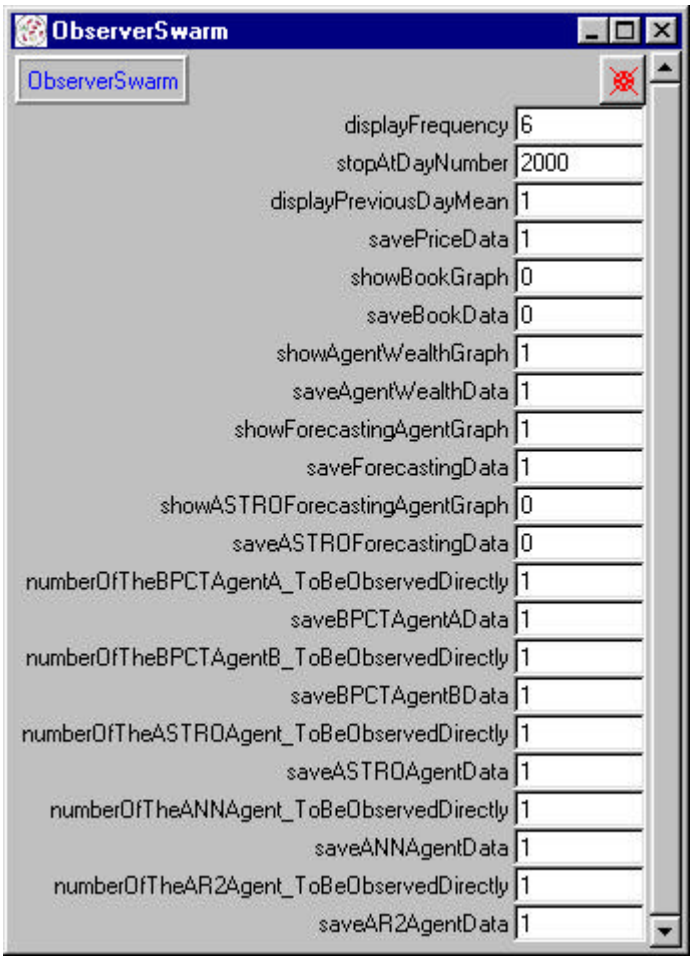
Il modello SUM ha una struttura a due livelli: il livello Observer, che mostra i risultati generati dall'esecuzione della simulazione e il livello Model che contiene il modello vero e proprio: il mercato borsistico artificiale e l'insieme dei differenti agenti[108].
Di seguito sono illustrati i significati dei parametri utilizzati dal modello:

- parametri e variabili utilizzate dall'Observer, modificabili dall'utente per mezzo della 'probe' riportata di seguito:

  - *displayFrequency*: è la frequenza alla quale vengono aggiornati gli oggetti grafici; se il suo valore è pari a 1000, nel grafico verrà riportato un prezzo ogni mille; una buona scelta è di inserire come frequenza di visualizzazione il numero totale degli agenti; in questo modo il grafico visualizzerà l'ultimo prezzo di ogni giorno simulato (ogni agente, infatti, compie una azione al giorno);
  - *stopAtEpochNumber*: rappresenta il numero di giorni simulati al raggiungimento del quale la simulazione termina;

le variabili che seguono hanno il significato di *switch*; se poste pari a 1 attivano la relativa funzione:

- *displayPreviousDayMean*: visualizza il grafico del prezzo medio;
- *savePriceData*: salva i dati dei prezzi in un apposito file;
- *showBookGraph*: visualizza il grafico del book di borsa;
- *saveBookGraph*: salva i dati del book in un apposito file;
- *showAgentWealthGraph*: mostra il grafico della ricchezza degli agenti;
- *saveAgentWealthGraph*: salva i dati della ricchezza degli agenti;
- *showForecastingAgentGraph*: visualizza il grafico con le previsioni dell'agente neurale;
- *saveForecastingData*: salva i dati delle previsioni dell'agente neurale;
- *showAstroForecastingAgentGraph*: visualizza il grafico con le previsioni dell'agente astrologo;
- *saveASTROForecatingData*: salva i dati delle previsioni dell'agente astrologo;

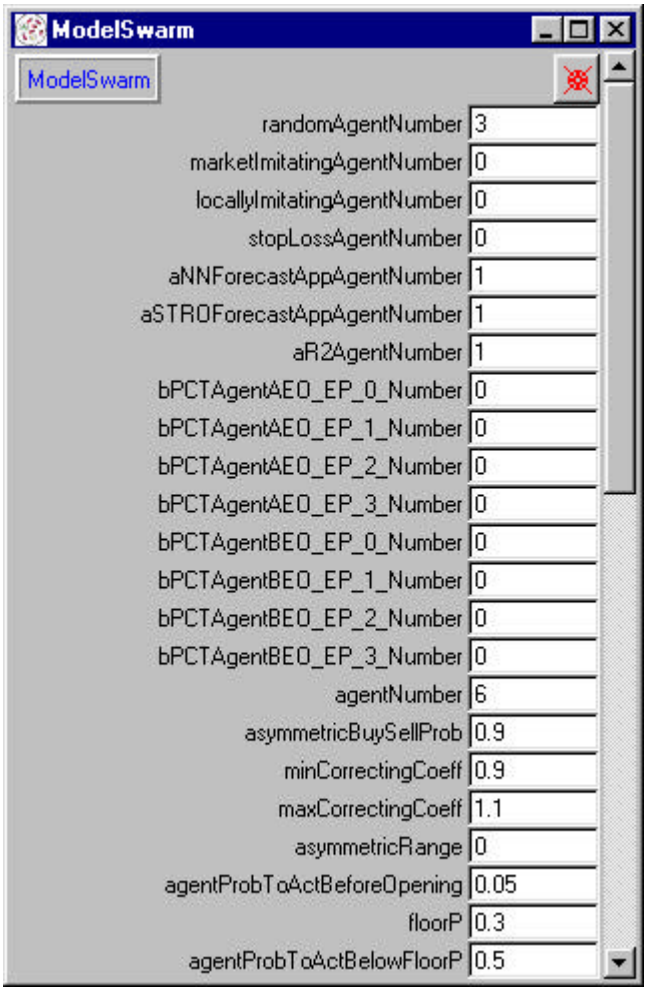


- *numberOfTheBPCTAgentA_ToBeObservedDirectly*: il numero dell'agente BPCTAgentA di cui si vuole visualizzare il grafico con la liquidità, la quantità di azioni e la ricchezza;
- *saveBPCTAgentAData*: salva i dati dell'agente BPCTAgentA in un apposito file;

[108] Il modello SUM, la cui realizzazione ha richiesto la scrittura di oltre 7000 linee di codice, può essere scaricato dal sito internet http://eco83.econ.unito.it/~terna/sum/sum.html.

- *numberOfTheBPCTAgentB_ToBeObservedDirectly*: il numero dell'agente *BPCTAgentB* di cui si vuole visualizzare il grafico con la liquidità, la quantità di azioni e la ricchezza;
- *saveBPCTAgentBData*: salva i dati dell'agente BPCTAgentB in un apposito file;
- *numberOfTheANNAgent_ToBeObservedDirectly*: il numero dell'agente *ANNForecastAppAgent* di cui si vuole visualizzare il grafico con la liquidità, la quantità di azioni e la ricchezza;
- *saveANNAgentData*: salva i dati dell'agente *ANNForecastAppAgent* in un apposito file;
- *numberOfTheASTROAgent_ToBeObservedDirectly*: il numero dell'agente *ASTROForecastAppAgent* di cui si vuole visualizzare il grafico con la liquidità, la quantità di azioni e la ricchezza;
- *saveASTROAgentData*: salva i dati dell'agente *ASTROForecastAppAgent* in un apposito file;
- *numberOfTheAR2Agent_ToBeObservedDirectly*: il numero dell'agente *AR2Agent* di cui si vuole visualizzare il grafico con la liquidità, la quantità di azioni e la ricchezza;
- *saveAR2AgentData*: salva i dati dell'agente *AR2Agent* in un apposito file;

- parametri e variabili utilizzate dal Model, modificabili dall'utente per mezzo della 'probe' riportata di seguito:



tali parametri sono suddivisibili nelle seguenti classi:

- parametri del Model:

- *randomAgentNumber*: numero di agenti che agiscono in modo casuale;
- *marketImitatingAgentNumber*: numero di agenti che scelgono se acquistare o vendere sulla base dell'imitazione del mercato;
- *locallyImitatingAgentNumber*: numero di agenti che scelgono se acquistare o vendere sulla base dell'imitazione locale;
- *stopLossAgentNumber*: numero di agenti che adottano strategie di stop loss;
- *aNNForecastAppAgentNumber*: numero di agenti che seguono la previsione dell'agente neurale (comprano o vendono in modo coerente con le stime dei prezzi futuri);
- *aSTROForecastAppAgentNumber*: numero di agenti che seguono la previsione sull'andamento del mercato formulata dall'agente astrologo;
- *aR2AgentNumber*: numero di agenti 'autoregressivi';
- *bPCTAgentA_EO_EP_0_Number*: numero di agenti BPCTAgentA senza obiettivo esterno;
- *bPCTAgentA_EO_EP_1_Number*: numero di agenti BPCTAgentA con EO di accrescere la liquidità;
- *bPCTAgentA_EO_EP_2_Number*: numero di agenti BPCTAgentA con EO di accrescere la quantità di azioni;
- *bPCTAgentA_EO_EP_3_Number*: numero di agenti BPCTAgentA con EO di accrescere sia la quantità di azioni sia la liquidità;
- *bPCTAgentB_EO_EP_0_Number*: numero di agenti BPCTAgentB senza obiettivo esterno;
- *bPCTAgentB_EO_EP_1_Number*: numero di agenti BPCTAgentB con EO di aumentare la ricchezza valutata al prezzo di chiusura;
- *bPCTAgentB_EO_EP_2_Number*: numero di agenti BPCTAgentB con EO di aumentare la ricchezza valutata al prezzo previsto;
- *bPCTAgentB_EO_EP_3_Number*: numero di agenti BPCTAgentB con EO di aumentare sia la ricchezza valutata al prezzo di chiusura che quella valutata al prezzo previsto.
- *agentNumber*: numero di agenti presenti nel modello[109];

- parametri globali, comuni a tutti gli agenti:

- *asymmetricBuySellProb*: se vengono adottate strategie imitative, tale variabile rappresenta la probabilità di comprare o vendere in accordo con tale strategie[110]; in assenza di comportamenti imitativi, come nel caso di agenti casuali, la probabilità di acquistare o di vendere è pari a 0.5;
- *minCorrectingCoefficient* e *maxCorrectingCoefficient*: il valore minimo e quello massimo del coefficiente usato per fissare il prezzo di una proposta di negoziazione di acquisto o di vendita[111];

[109] Tale valore viene calcolato internamente come somma delle diverse tipologie di agenti.
[110] (1-asymmetricBuySellProb) è invece la probabilità di agire in contrasto con quanto suggerito dalle strategie imitative.

- *asymmetricRange*: correzione (eventualmente) aggiunta a quella descritta sopra, che consente all'agente di adottare un comportamento asimmetrico;
- *agentProbToActBeforeOpening*: la probabilità di agire in fase di preapertura;
- *floorP*: il valore del *floor* per il prezzo[112];
- *agentProbToActBelowFloorPrice*: la probabilità di comprare quando il prezzo scende sotto il floor[113];
- *maxOrderQuantity*: il quantitativo massimo di acquisto o di vendita per ogni ordine collocato dall'agente[114];

➢ parametri utilizzati dagli agenti con strategie imitative:

- *meanPriceHistoryLength*[115]: lunghezza del vettore che contiene i prezzi medi di ogni giornata (utilizzato dagli agenti che imitano il comportamento del mercato);
- *localHistoryLength*: lunghezza del vettore che registra le azioni degli agenti (utilizzato dagli agenti che imitano localmente);

➢ parametri utilizzati dagli agenti stop loss:

- *maxLossRate*: il tasso di perdita oltre il quale l'agente adotta la strategia di stop loss;
- *stopLossInterval*: lunghezza del periodo preso in considerazione nella strategia di stop loss;
- *checkingIfShortOrLong*: se pari a 0, l'agente opera senza memoria del passato; se, invece, è uguale a 1, l'agente, nell'adottare la strategia di stop loss, prende in considerazione la propria posizione speculativa;

➢ parametri utilizzati dall'agente neurale:

- *dataWindowsLength*: il numero di dati utilizzati come input dalla rete neurale artificiale; i dati sono espressi come indici calcolati rapportando il prezzo medio del giorno *t* con il prezzo medio del giorno *t-nAheadForecasting*;
- *nAheadForecasting*: numero di giorni 'in avanti' della previsione neurale; l'output della rete è la stima del rapporto tra prezzo al tempo *nAheadForecasting* e l'ultimo prezzo medio disponibile;

---

[111] Gli agenti, come affermato in precedenza, fissano il limite di prezzo moltiplicando l'ultimo prezzo eseguito per un coefficiente casuale compreso tra *minCorrectingCoefficient* e *maxCorrectingCoefficient*; questa correzione è usata da tutti i tipi di agenti al fine di introdurre un certo grado di casualità nel comportamento degli agenti.

[112] Tutti i tipi di agenti, eccetto i BPCTAgent di tipo A e B, usano questo parametro e quello seguente.

[113] Per l'agente che opera sotto il floor, il limite di prezzo è pari all'ultimo prezzo eseguito (non si moltiplica l'ultimo prezzo per il coefficiente casuale).

[114] Gli agenti scelgono un quantitativo compreso tra 1 e *maxOrderQuantity*; tecnicamente l'immissione nel book dell'ordine per un quantitativo pari a *n* (con 1<*n*<=*maxOrderQuantity*) non viene realizzato in una unica soluzione, ma avviene immettendo *n* ordini unitari.

[115] Tale parametro è utilizzato anche dall'agente neurale e dall'astrologo.

- *forecastingTrainingSetLength*: numero di 'esempi' (formati dagli input e dall'output atteso) utilizzati per allenare la rete neurale; il training set viene continuamente aggiornato[116];
- *epochNumberInEachForecastingTrainingCycle*: numero di epoche di apprendimento in ogni fase di apprendimento della rete neurale artificiale; in ogni epoca la rete effettua *forecastingTrainingSetLength* cicli di apprendimento;
- *learningProcessEveryNDays*: è il numero di giorni che passano tra una fase di apprendimento e l'altra; ogni *learningProcessEveryNDays* il training della rete viene ripetuto;
- *cleanForecastingANNEveryMgtemNDays*[117]: ogni *cleanForecastingANNEveryMgtemNDays* giorni il processo di apprendimento ricomincia da capo; i pesi della rete vengono riinizializzati casualmente; in questo modo è possibile tenere conto degli eventuali cambiamenti nella struttura dei dati di input;

➢ parametri utilizzati dagli agenti che applicano la previsione neurale:

- *aNNInactivityRange*: il range di variazione del prezzo previsto entro il quale l'agente rimane inattivo; se la previsione neurale è compresa tra (1-*aNNInactivityRange*) e (1+ *aNNInactivityRange*) l'agente non agisce sul mercato;
- *aNNForecastAppAgentActDailyProb*: la probabilità di agire sul mercato; tale parametro è introdotto sulla base della considerazione che gli agenti che operano con previsioni non modificano il proprio comportamento continuamente;

➢ parametri utilizzati dagli agenti BPCT[118]:

- *epochNumberInEachBPCTTrainigCycle*: il numero di epoche dell'apprendimento (di tipo 'long term') della rete; il learning è applicato ogni giorno sull'insieme, continuamente aggiornato, dei dati degli ultimi 10 giorni;
- *agentAEO_EPDelta*: parametro utilizzato per misurare l'effetto degli obiettivi esterni per gli agenti cross-target di tipo A;
- *agentBEO_EPDelta*: parametro utilizzato per misurare l'effetto dell'obiettivo esterno per gli agenti BPCT di tipo B.

➢ parametri utilizzati dagli agenti 'autoregressivi':

- *alpha_min, alpha_max*: limite inferiore e superiore del coefficiente *a* dell'equazione autoregressiva: $p_t = a\ p_{t-1} + b\ p_{t-2}$ utilizzata dagli agenti AR2Agent per stimare il prezzo futuro;

[116] Si noti che la *meanPriceHistoryLength* deve essere maggiore o uguale a *forecastingTrainingSetLength* + *nAheadForecasting*; l'apprendimento della rete neurale inizia al giorno (*forecastingTrainingSetLength* + *nAheadForecasting* + *1*).
[117] *MgtemN*: per M più grande (o uguale) e multiplo di N.

- *beta_min, beta_max*: limite inferiore e superiore del coefficiente *b* dell'equazione autoregressiva: $p_t = a\ p_{t-1} + b\ p_{t-2}$;

➢ parametri utilizzati dagli agenti che applicano la previsione dell'astrologo:

- *aSTROInactivityRange*: il range di variazione del prezzo previsto entro il quale l'agente rimane inattivo; se la previsione dell'astrologo è compresa tra (1-*aSTROInactivityRange*) e (1+ *aSTROInactivityRange*) l'agente non opera sul mercato;
- *aSTROForecastAppAgentActDailyProb*: la probabilità di agire sul mercato;

4.5 Le classi di SUM

La realizzazione della simulazione del mercato di borsa ha richiesto la definizione delle seguenti classi:

- BasicSumAgent: è la classe radice degli agenti in SUM; tutti gli agenti, tranne il previsore neurale, ne ereditano le funzionalità[119];

- BasicSumRuleMaster: è la classe radice dei Rule Master; i Rule Master di tutte le diverse tipologie di agenti, eccetto quelli del previsore neurale e degli agenti BPCT, ne ereditano i metodi e le variabili istanza;

- Book: gestisce il funzionamento del book di borsa (assicura il corretto funzionamento della fase di preapertura e di quella di negoziazione continua); in tale classe sono definiti i metodi che consentono l'immissione degli ordini da parte degli agenti, il loro abbinamento automatico, la registrazione delle proposte ineseguite;

- RandomAgent: è la classe che definisce gli agenti casuali; eredita le funzionalità di base da BasicSumAgent;

- RandomRuleMaster: definisce il Rule Master che guida il comportamento degli agenti casuali, degli agenti che imitano il mercato e di quelli che imitano localmente;

- MarketImitatingAgent: eredita da BasicSumAgent; è la classe che definisce gli agenti che operano in borsa seguendo la tendenza del mercato;

- LocallyImitatingAgent: definisce gli agenti che operano sul mercato imitando il comportamento degli altri agenti;

---

[118] I parametri di default utilizzati dagli agenti di tipo BPCT sono contenuti nel file bp.setup.

- StopLossAgent: è la classe che definisce gli agenti di tipo stop loss (tali agenti se non adottano la strategia di limitare le perdite, operano come gli agenti casuali);

- StopLossRuleMaster: tale classe definisce il Rule Master che guida il comportamento degli agenti stop loss;

- ForecastingAgent: è la classe che gestisce il funzionamento della rete neurale incaricata di formulare la previsione sull'andamento del mercato (definisce la struttura generale della rete: l'architettura, i parametri di apprendimento, le strutture dati necessarie per il processo di calcolo degli errori di backpropagation);
- SimpleANNRuleMaster: è il Rule Master che guida il comportamento della rete neurale; contiene i metodi che, permettono di applicare, ai dati ricevuti in input, la funzione a rete neurale al fine di ottenere la previsione;

- SimpleANNRuleMaker: è la classe che consente di modificare il comportamento della rete neurale; contiene i metodi che gestiscono l'algoritmo di back-propagation e la variazione dei pesi di interconnessione tra i diversi nodi della rete;

- TransFunc: tale classe fornisce i metodi necessari per l'applicazione della funzione logistica ai nodi della rete e per il calcolo delle derivate prime utilizzate dall'algoritmo di backpropagation dell'errore;

- VectorTransFunc: fornisce i metodi necessari per il calcolo degli errori in fase di apprendimento;

- ANNForecastAppAgent: eredita da BasicSumAgent; definisce gli agenti che seguono le stime dei prezzi elaborate dall'agente neurale;

- ASTROForecastingAgent: è la classe che gestisce il funzionamento dell'agente astrologo ;

- ASTROForecastAppAgent: eredita da BasicSumAgent; definisce gli agenti che seguono le stime dei prezzi futuri dell'astrologo;

- AR2Agent: è la classe che definisce gli agenti autoregressivi;

- BPCTAgent, BPCTInterface, BPCTDataWareHouse, BPCTRuleMaster, BPCTRuleMaker: tali classi consentono di definire la struttura base degli agenti di tipo 'cross-target';

[119] In BasicSumAgent viene definito il metodo per la gestione della contabilità degli agenti.

- BPCTAgentA, BPCTAgentAInterface, BPCTAgentB, BPCTAgentBInterface, BPCTPriceRuleMaster: ampliano le funzionalità di base definite dalle classi precedenti al fine di realizzare gli agenti cognitivi di tipo A e di tipo B;

- CurrentAgent: è una classe di servizio, utilizzata dal modello per trasferire il messaggio di agire (in fase di negoziazione continua) a tutti gli agenti che popolano la simulazione;

- Matrix, Matrix2, MatrixMult: sono classi che forniscono i metodi necessari per la gestione e la manipolazione dei vettori e delle matrici utilizzate dal modello;

- Quota, QuotaAstro: sono le classi che definiscono le 'probe' del previsore neurale e dell'astrologo; tali 'probe' contengono le informazioni relative alla percentuale di previsioni corrette;

- ModelSwarm: è la classe che definisce il modello della simulazione; è all'interno di questa classe che: vengono creati i diversi agenti, viene creato il book della borsa e viene definito, mediante un'opportuna scansione degli eventi, il timing della simulazione;

- ObserverSwarm: definisce l'*observer* del modello che gestisce la visualizzazione dei diversi oggetti grafici sullo schermo.

Alle classi elencate si aggiungono alcuni file di servizio[120]:

- forecasting.setup: contiene i valori dei parametri utilizzati dalla rete neurale che realizza l'agente neurale - range di inizializzazione dei pesi dei diversi nodi, *learning rate*, *momentum,* indicazione della modalità secondo la quale i pattern del training set devono essere utilizzati in fase di apprendimento (sempre nello stesso ordine o in ordine casuale) - i valori contenuti in questo file modificano quelli di default definiti nella classe modelswarm;

- transf.setup: in questo file viene definito il valore k della funzione logistica $f(x) = 1/(1 + e^{-kx})$, utilizzata dalla rete neurale quale funzione di attivazione;

- bp.setup: contiene i parametri che definiscono il funzionamento delle reti neurali 'cross-target' utilizzate per la realizzazione degli agenti cognitivi (architettura della rete, lunghezza del training set, modalità di apprendimento, range di inizializzazione dei pesi dei nodi, *learning rate*, *momentum*); i valori contenuti in questo file modificano quelli di default definiti nella classe modelswarm;

[120] Ad eccezione di minmaxA.data e minmaxB.data, la presenza di questi file non è indispensabile ai fini dell'applicazione.

- minmaxA.data: contiene i valori necessari per il passaggio dalla metrica esterna, nella quale sono espressi gli input e gli output dell'agente cognitivo A, alla metrica interna della rete neurale;

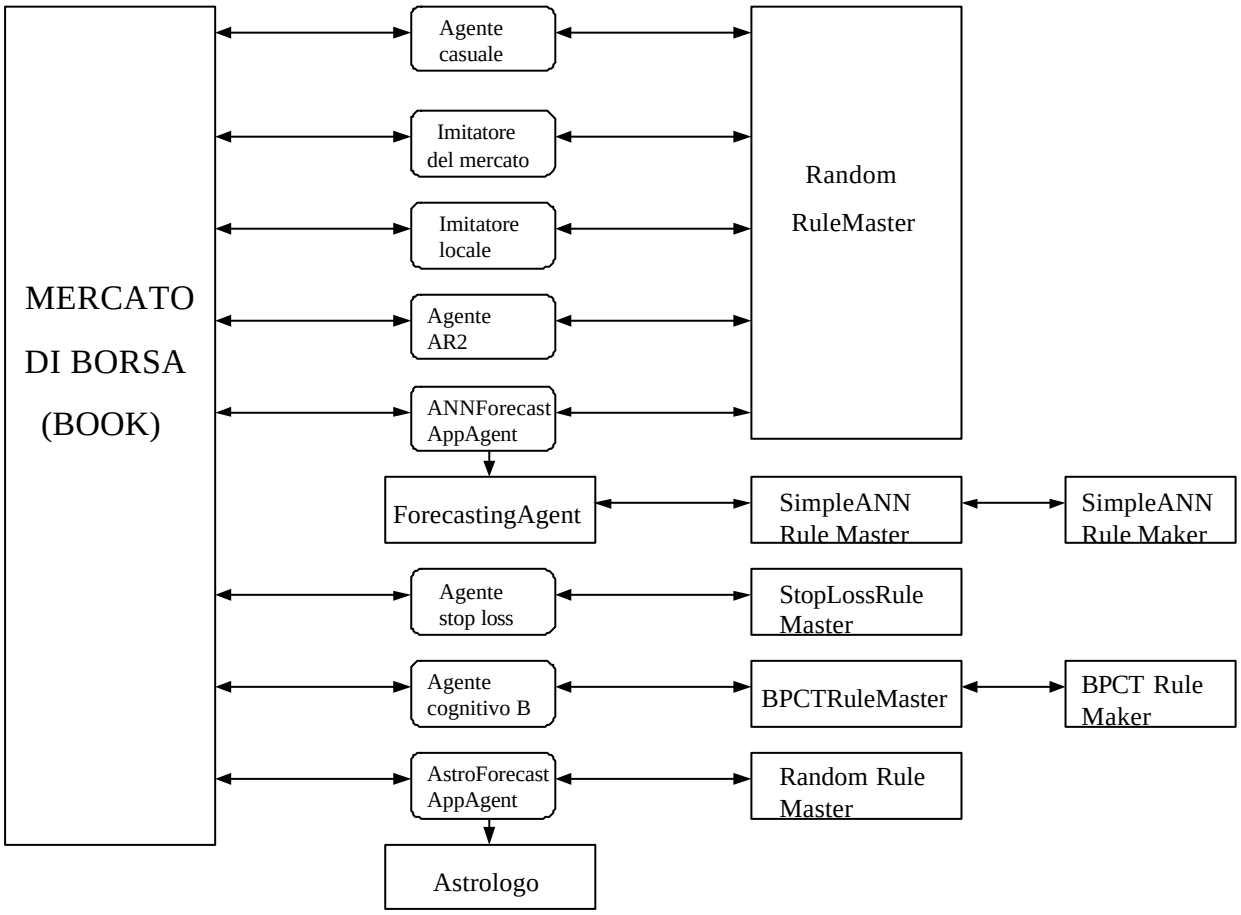
- minmaxB.data: contiene i valori necessari per il passaggio dalla metrica esterna, nella quale sono espressi gli input e gli output dell'agente cognitivo B, alla metrica interna della rete neurale;

- initA.val (e initA.val.off): contiene i valori iniziali, input e output espressi in metrica esterna, dell'agente cognitivo di tipo A;

- initB.val (e initB.val.off): contiene i valori iniziali, input e output espressi in metrica esterna, dell'agente cognitivo di tipo B;

4.6 Lo schema del modello

Il modello è stato realizzato secondo lo schema Enviroment-Rules-Agent (ERA) descritto nel primo capitolo.



Gli agenti del modello non interagiscono direttamente tra loro; comunicano le proprie decisioni al book, il quale ha la funzione di abbinare le proposte di negoziazione dei diversi agenti e di assicurare il corretto funzionamento del mercato artificiale di borsa.
Come è possibile notare dal grafico, gli agenti dotati di una struttura cognitiva 'semplice' sono collegati con solo due oggetti: il book e il Rule Master.
Per gli agenti 'cognitivi', invece, è previsto un 'modulo' (oggetto) supplementare – il Rule Maker – la cui funzione è di realizzare il continuo adattamento dell'agente all'ambiente circostante.
Nella categoria degli agenti con struttura cognitiva limitata rientrano anche gli agenti che applicano la previsioni fornite dalla rete neurale e gli agenti che seguono l'astrologo; tali agenti, a differenza dei precedenti si caratterizzano, però, per la presenza di un collegamento diretto con un 'agente' esterno al mercato.
Gli agenti *ASTROForecastAppAgent*, sono collegati in modo diretto all'agente astrologo dal quale ottengono l'informazione sull'evoluzione futura del mercato (determinata sulla base di valutazioni puramente casuali); gli agenti *ANNForecatAppAgent* sono, invece, collegati al previsore neurale (*ForecastingAgent*), dal quale ottengono le informazioni sull'evoluzione futura dei prezzi determinate sulla base di un processo di apprendimento continuo sui dati 'generati' dal mercato (anche l'agente neurale dispone di un Rule Maker con lo scopo di adattare nel tempo il comportamento della rete).

4.7 Il timing della simulazione

Ogni giornata di borsa si articola nelle seguenti fasi:

- 'clean': il book di borsa viene azzerato; le proposte di negoziazione immesse dagli agenti il giorno precedente vengono cancellate;
- preapertura: gli agenti valutano se immettere o meno ordini di acquisto o di vendita nel book della borsa; tale fase serve esclusivamente per non far iniziare le contrattazioni con un book vuoto; le proposte immesse in preapertura vengono quindi girate alla fase di contrattazione continua;
- negoziazione continua: in questa fase, gli agenti decidono se acquistare o vendere sulla base delle proprie regole di comportamento; le proposte di negoziazione immesse possono essere eseguite immediatamente dal book se vi è una controparte nel 'registro'(*log*); in caso contrario le proposte di negoziazione, con la relativa priorità temporale, vengono registrate e potranno essere concluse più avanti nel corso della giornata;
- contabilità: al termine della giornata di borsa si procede al calcolo del prezzo medio; gli agenti calcolano, quindi, la propria ricchezza composta dalla liquidità e dalle azioni possedute (valutate al prezzo medio della giornata);

Il modello SUM per realizzare la scansione del tempo desiderata opera secondo un 'orologio' interno composto da un numero di *tick* pari al numero degli agenti che popolano la simulazione.
Di seguito è riportato, in dettaglio, il *timing* della simulazione per ogni giorno di borsa:

Giorno *t*

al tempo 0:

- viene aggiornato il giorno corrente della simulazione;
- viene 'pulito' il book della borsa;
- l'agente neurale (e l'astrologo) formula la propria previsione sul prezzo del giorno t+*nAheadForecasting–1*;
- viene aggiornata la percentuale delle previsioni corrette del previsore neurale;
- la lista contenete gli agenti del modello viene 'mischiata' in modo che ogni giorno gli agenti operino non sempre nello stesso ordine;
- fase di preapertura: immissione delle proposte di negoziazione degli agenti che vengono girate alla fase di negoziazione continua;

al tempo 1 e fino al tempo (*NumberAgent* – 1), in modo da consentire ad ogni agente di operare sul mercato:

- fase di negoziazione continua;

al tempo *NumberAgent*:

- viene determinato il prezzo medio della giornata;
- gli agenti procedono al calcolo della loro ricchezza;

4.8 Alcuni aspetti tecnici

La gestione degli agenti del modello SUM viene effettuata utilizzando particolari oggetti, denominati *liste*; tali dispositivi, forniti dalla libreria *collections* di Swarm, consentono di registrare gli indirizzi di memoria dei singoli agenti che popolano il modello.
Grazie all'utilizzo delle liste - che svolgono una funzione di intermediazione fra il *Model* e gli agenti - diventa possibile semplificare l'invio di un messaggio alla popolazione di agenti della simulazione; il messaggio di effettuare una determinata operazione invece di essere mandato ai singoli agenti, viene mandato alla lista, la quale provvederà a 'passarlo' ai vari oggetti.
Nel modello SUM le liste vengono utilizzate per girare agli 'agenti' i messaggi di agire nelle diverse fasi della giornata di borsa (pre-apertura, negoziazione continua e contabilità).
Inoltre, per evitare, di utilizzare sempre la stessa sequenza di attivazione degli agenti, il contenuto delle liste viene rimescolato ogni giorno, grazie ad un oggetto di servizio, denominato *Shuffler*.
Nel modello, inoltre, per garantire che i risultati della simulazione non dipendano da correlazioni 'interne' al modello stesso, sono stati definiti una serie di generatori di numeri casuali indipendenti per le diverse tipologie di agenti; in questo modo è possibile modificare l'innesco dei numeri casuali di una tipologia di agenti senza che questo comporti anche una variazione del comportamento degli altri agenti che popolano la simulazione.

Il modello SUM, oltre alla possibilità di fermare l'esecuzione dell'applicazione e di riprenderla, a discrezione dello sperimentatore, grazie all'interazione realizzata per mezzo del 'Pannello di Controllo', consente all'utente anche di 'sondare' la struttura interna delle diverse tipologie di agenti, indagando lo stato delle variabili e la loro evoluzione nel tempo.
Nel modelswarm è contenuto un metodo, denominato *openProbeTo:* che consente di aprire una 'sonda' ai singoli agenti; immettendo nella 'probe' del modelswarm il numero dell'agente che si vuole 'sondare' il metodo

openProbeTo:, grazie all'utilizzo di una apposita lista è in grado di 'pescare' l'indirizzo di memoria dell'agente e di visualizzarne lo stato delle variabili.
Grazie a Swarm l'utente può, così, indagare la struttura 'di basso livello' del modello SUM: può, ad esempio, analizzare il contenuto delle matrici che registrano l'andamento del prezzo medio, può ottenere gli indirizzi di memoria dei diversi oggetti e può visualizzare nel dettaglio la struttura delle classi che definiscono i singoli agenti.
L'utente dell'applicazione può, inoltre, modificare in modo dinamico il livello di dettaglio dei grafici aumentandone il fattore di zoom.
Infine, per facilitare il compito degli utenti che volessero estendere l'applicazione originaria del modello per introdurre nuove tipologie di agenti, il modello SUM fornisce uno schema generale che individua i segmenti del codice da modificare e da integrare (AgentForge.txt).

APPENDICE: codice

La realizzazione del modello SUM ha richiesto, come affermato precedentemente, la scrittura di oltre 7000 linee di codice; di seguito - per ovvi motivi di spazio - è stata riportata solo una parte del codice del programma: la parte che realizza gli agenti 'autoregressivi', l'agente astrologo e i suoi fedeli (una decina di pagine).

BIBLIOGRAFIA

- BORSA ITALIANA S.P.A (2000), *Regolamento dei mercati organizzati e gestiti dalla Borsa Italiana S.p.A*, http://www.borsaitaliana.it.
- KEYNES J. M. (1986), *Teoria generale dell'occupazione, dell'interesse e della moneta*, Utet, Torino.
- TERNA P. (2000), *Economic Experiments with Swarm: a Neural Network Approach to the Self-Development of Consistency in Agents' Behavior* in Luna F. e Stefansson B. (eds), Economic Simulations in Swarm: Agent-Based Modelling and Object Oriented Programming. Dordrecht and London: Kluwer Academic.
- TERNA P. (2001), *Cognitive agents behavings in a simple stock market structure*, in corso di pubblicazione.

In questo capitolo verranno condotte una serie di applicazioni del modello SUM al fine di analizzare le conseguenze - sulla ricchezza degli individui e sulla dinamica del mercato - della presenza e dell'interazione delle diverse tipologie di agenti.

Il modello consente l'esplorazione di infinite configurazioni del mercato; inizialmente si prenderà in considerazione un mercato di borsa popolato esclusivamente da agenti semplici (caratterizzati da una limitata percezione dell'ambiente); quindi – sulla base dello schema delineato in Terna (2001) – il modello verrà esteso, modificando di volta in volta le variabili e i parametri della simulazione, in modo da includere le diverse tipologie di agenti illustrate nel capitolo precedente e da analizzare l'effetto sul comportamento globale del mercato dell'introduzione di agenti 'cognitivi', realizzati secondo la metodologia dei cross target, con strategie di comportamento autosviluppate; verranno, inoltre, condotte una serie di applicazioni volte ad indagare un particolare aspetto dei mercati di borsa: l'autorealizzazione delle aspettative.

5.1 Primo esperimento: agenti casuali

In questa prima applicazione, il modello è popolato da 300 agenti *random* i quali comprano o vendono azioni sulla base di una semplice scelta casuale[121]; il quantitativo massimo negoziabile è pari a 3 lotti.

In figura 5.1 è riportata la dinamica del prezzo.

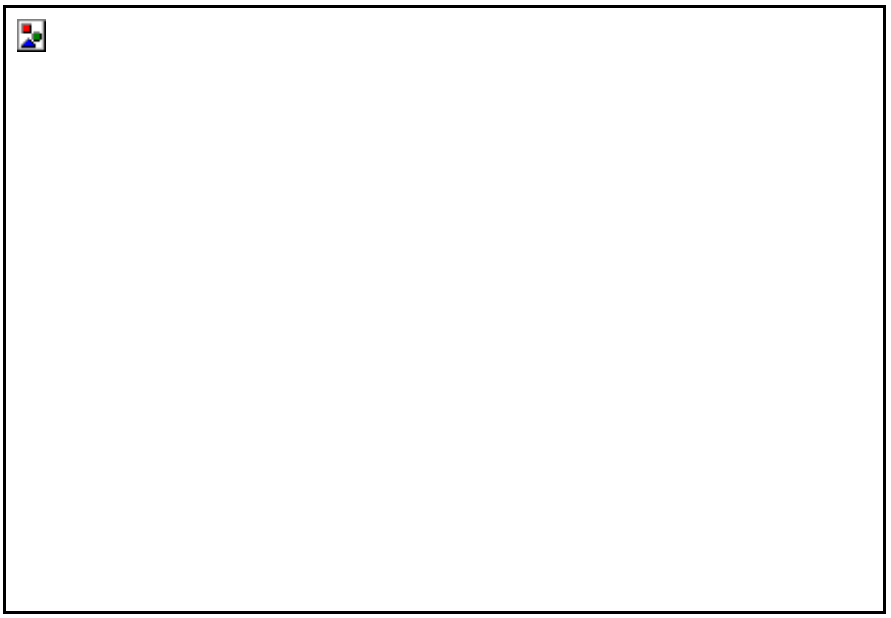


Fig. 5.1- Grafico del prezzo: 300 agenti *random*, con *maxOrderQuantity* pari a 3 .

Come è possibile notare, anche in presenza di semplici agenti *no-mind*, il mercato mostra una dinamica estremamente interessante.

[121] Come è stato illustrato nel quarto capitolo, gli agenti *random* conoscono solamente l'ultimo prezzo eseguito, scelgono casualmente se acquistare o vendere e fissano il proprio limite di prezzo moltiplicando l'ultimo prezzo eseguito per un coefficiente casuale; se il prezzo scende al di sotto del *floorPrice* (pari a 0.3, ma modificabile dall'utente dell'applicazione mediante la probe del modelswarm), gli agenti acquistano azioni all'ultimo prezzo eseguito nel mercato.

Il mercato alterna periodi caratterizzati da un andamento relativamente stabile e regolare con periodi di più intensa variabilità nel livello dei prezzi. Nell'arco di tempo considerato dalla simulazione – 2000 giornate di borsa - si formano due bolle speculative di una certa entità (seguite dai relativi *crash*).

È importante sottolineare che le bolle e i crash non sono effetto di cause esogene al modello, ma trovano la loro spiegazione nella struttura del mercato; le regole di funzionamento del *book* elettronico influenzano, infatti, in modo determinante il modello, poiché intervengono direttamente non soltanto sui possibili comportamenti degli agenti, ma anche sui meccanismi di interazione.

L'importanza delle regole che definiscono la struttura del mercato emerge chiaramente nella simulazione seguente nella quale il numero di agenti random è stato posto pari a 100 e il quantitativo massimo negoziabile è stato aumentato da tre a sei lotti; la dinamica del mercato è riportata in figura 5.2;

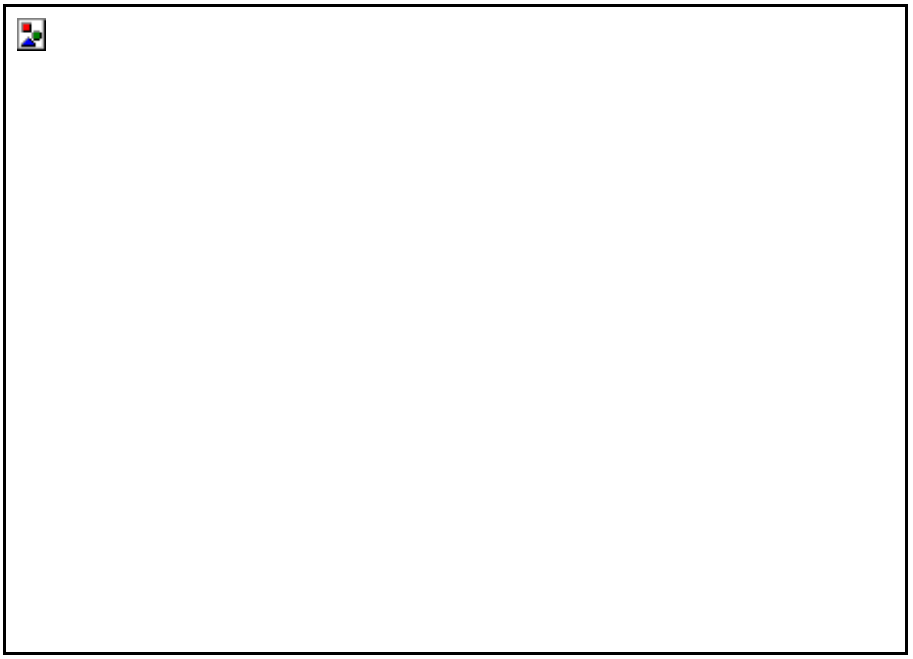


Fig. 5.2- Grafico del prezzo: 100 agenti *random*, con *maxOrderQuantity* pari a 6.

in questo caso si formano bolle di entità rilevante (nella più grande delle quali il prezzo supera il livello di trenta volte il prezzo iniziale).

La causa di tale dinamica può essere individuata nel meccanismo di funzionamento del *book* elettronico della borsa; dall'analisi del *log* (registro) del *book* emerge, infatti, che le bolle speculative sono anticipate da squilibri tra numerosi ordini di acquisto e scarsi ordini di vendita (i crash vengono invece anticipati da squilibri tra proposte di vendita nettamente superiori alle proposte di acquisto).

La presenza di lunghe sequenze di ordini su uno stesso lato del mercato (acquisto o vendita) determina forti movimenti nei prezzi. Se il quantitativo massimo negoziabile dagli agenti è relativamente basso raramente si verificano lunghe code di ordini non eseguiti e di conseguenza il mercato risulta stabile e regolare (prevale il 'rumore bianco'); se, invece, gli agenti possono negoziare quantitativi elevati, allora è più probabile che si verifichino squilibri tra ordini di acquisto e di vendita e che si formino bolle speculative di una certa consistenza.

Se consideriamo il numero di ordini di acquisto (b) e di vendita (s) non conclusi alla fine di ogni giornata di borsa come una misura dell'istante in cui inizia una bolla (o un crash) è possibile calcolare il coefficiente di determinazione R^2 della regressione pi = co + c1b + c2 s, dove *pi* è l'indice del prezzo giornaliero calcolato sulla base dell'ultimo prezzo p_0/p_{-1}. Il valore di R^2 pari a 0.35 conferma l'influenza degli ordini ineseguiti nella spiegazione degli ampi movimenti dei prezzi.

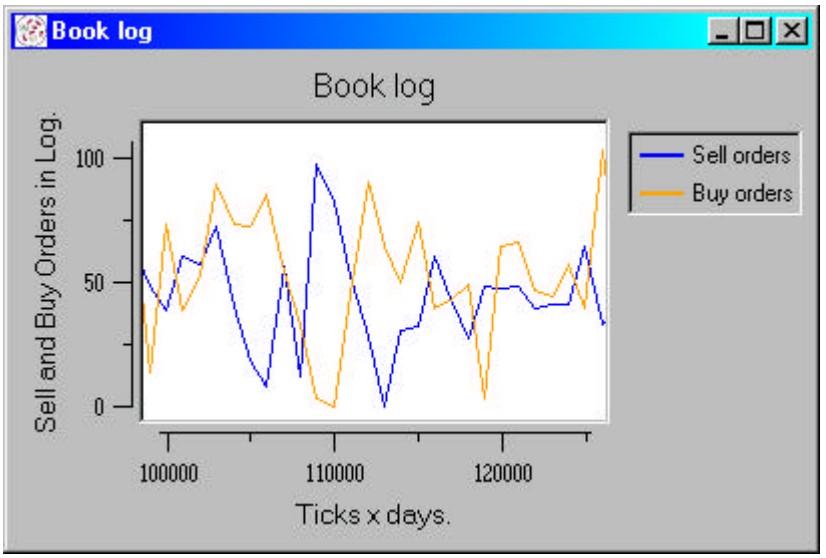


Fig.5.3 - Grafico del *book*[122], nel caso di 100 agenti *random* con *maxOrderQuantity* pari a 6.

Oltre ad analizzare la dinamica dei prezzi, il modello SUM consente di valutare il grado di prevedibilità del mercato artificiale di borsa mediante l'analisi delle stime prodotte dal previsore neurale; tale agente - che non opera direttamente sul mercato - produce delle previsioni sull'andamento futuro dei prezzi[123] e riporta la percentuale di previsioni corrette sul totale delle stime formulate. Nel caso del mercato con 100 agenti casuali e con quantitativo massimo negoziabile pari a 6, la percentuale di previsioni corrette è pari al 52%; il mercato mostra, cioè, un discreto livello di prevedibilità.

Come affermato in precedenza il modello SUM consente l'esplorazione di infinite configurazioni del mercato di borsa; l'utente dell'applicazione può scegliere di indagare le configurazioni che reputa più interessanti; in questo capitolo poiché l'obiettivo è quello di ottenere un mercato relativamente stabile si prende come punto di partenza il modello con 300 agenti casuali (e quantitativo massimo negoziabile pari a tre). Tale configurazione iniziale verrà quindi estesa al fine di indagare l'effetto sul mercato e sulla ricchezza dei singoli agenti dell'introduzione nel modello di individui con strategie di investimento più elaborate.

Nella configurazione base il mercato appare relativamente stabile, ma poco prevedibile: la percentuale di stime corrette del previsore neurale è pari, infatti, solo al 48%.

---

[122] Nel grafico riportato – ottenuto con Swarm - la scala temporale non è espressa in giorni ma in 'tick per giorni' (il numero di tick è pari al numero di agenti del modello).

[123] Nelle applicazioni condotte l'agente neurale effettua previsioni a 10 giorni (la lunghezza dell'arco temporale della previsione può essere modificata dall'utente agendo sulla probe del modelswarm).

Nel grafico seguente è riportata l'effetto che le azioni di vendita e di acquisto degli agenti esercitano sul livello di ricchezza degli agenti stessi[124].

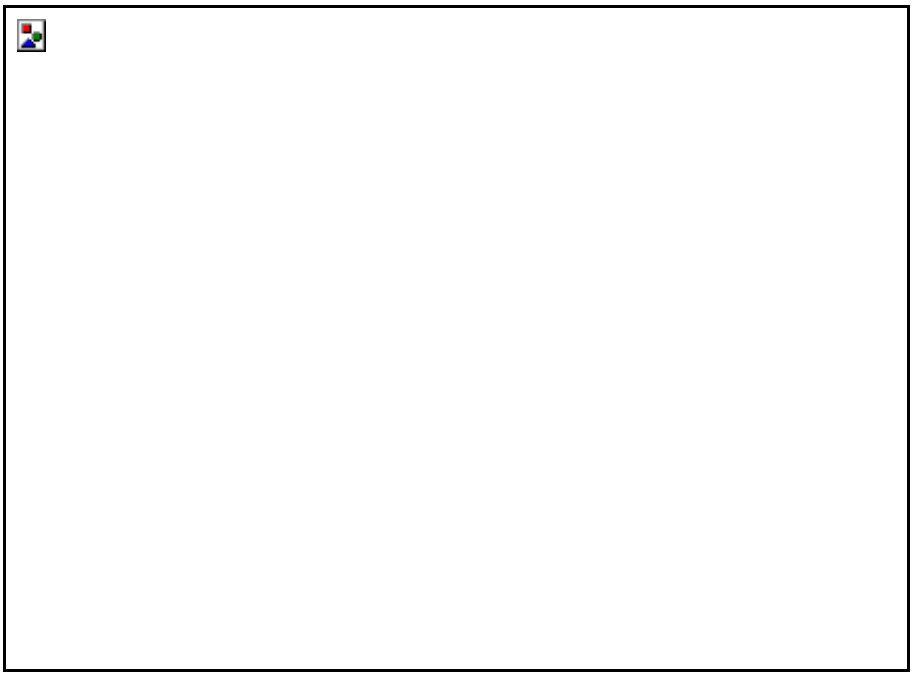


Fig. 5.4 – Grafico del livello di ricchezza in un mercato con 300 agenti casuali e *maxOrderQuantity* = 3.

La ricchezza viene calcolata sommando la liquidità al valore delle azioni, valutato al prezzo medio della giornata di borsa; le dotazioni iniziali di liquidità e di azioni degli agenti sono nulle; gli agenti, tuttavia, possono effettuare operazioni allo scoperto senza limiti.

Come è possibile notare, vi è una notevole differenza di risultati tra gli agenti casuali; alcuni durante le bolle riescono a guadagnare enormemente; altri, invece, si impoveriscono[125].

5.2 Agenti 'tecnici': imitatori del mercato, imitatori locali ed agenti *stop loss*.

L'esperimento iniziale viene esteso introducendo nuove tipologie di agenti che adottano strategie di investimento basate su metodologie assimilabili a quelle dell'analisi tecnica; nell'applicazione seguente il modello viene popolato con 285 agenti casuali e 15 imitatori del mercato, i quali non decidono di acquistare o vendere sulla base di una scelta casuale bilanciata, ma operano imitando il comportamento del mercato[126].

[124] Sulle linee riportate nel grafico vi sono agenti differenti che cambiano con lo 'scorrere' del tempo.

[125] Dato che gli agenti casuali sono i soli che operano all'interno del mercato, il livello medio di ricchezza è ovviamente nullo.

[126] Gli imitatori del mercato se il prezzo medio dal giorno t-2 al giorno t-1 è aumentato (diminuito) comprano (vendono) con probabilità pari a *asymmetricBuySellProb* (0,9, ma modificabile dall'utente); con probabilità pari a (1-*asymmetricBuySellProb*) gli agenti, invece, operano in contrasto con il mercato.

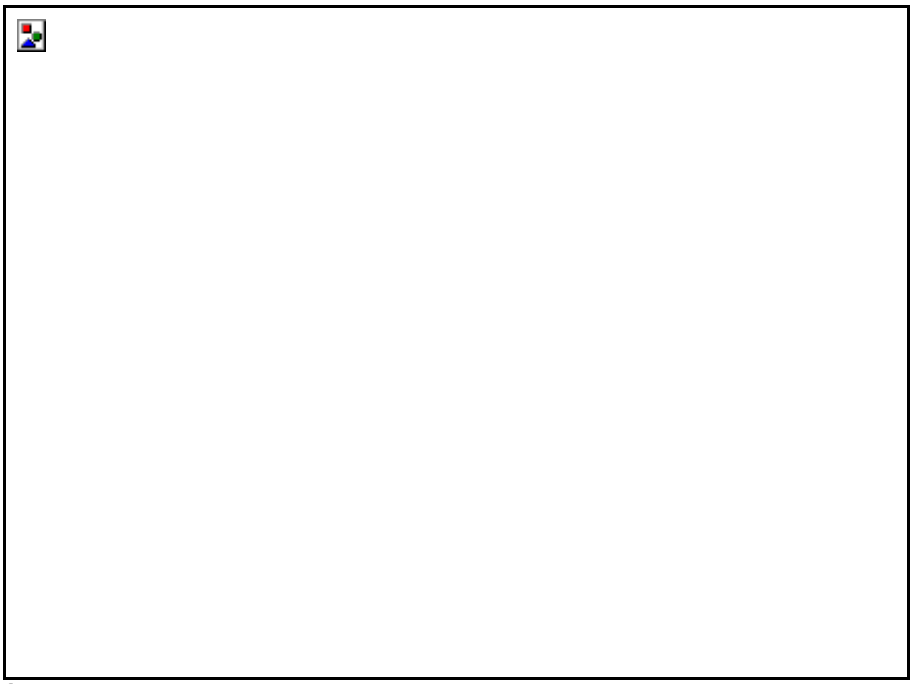


Fig. 5.5 – prezzo corrente con 285 agenti casuali e 15 agenti che imitano il mercato (maxOrderQuantity=3).

La presenza di questo tipo di agenti, anche in numero limitato (nel caso specifico il 5% del totale degli agenti), determina forti incrementi nella volatilità del mercato; nell'applicazione riportata in figura 5.5, è possibile notare la presenza di due enormi bolle speculative.

Anche in questo caso analizzando la dinamica del prezzo si riscontra una robusta correlazione con il numero di proposte di negoziazione ineseguite; se si effettua la regressione tra l'indice di variazione del prezzo e il numero di ordini di acquisto e di vendita non eseguiti al termine di ogni giornata si ottiene un coefficiente di determinazione R^2 pari a 0,3865.

Il legame tra andamento del prezzo e ordini ineseguiti risulta particolarmente rilevante nei periodi caratterizzati dalla formazione di bolle speculative; se, infatti, la regressione viene effettuata - invece che sull'intero arco temporale della simulazione - con riferimento al periodo di tempo caratterizzato dalla più alta variabilità nei livelli di prezzo si ottiene un coefficiente di determinazione pari a 0,54[127].

Osservando il *log* del book è possibile notare come nella fasi di più intensa impennata dei prezzi il rapporto tra gli ordini di acquisto e quelli di vendita non conclusi si collochi su livelli decisamente elevati[128]; nelle fasi di 'sgonfiamento' delle bolle, invece, è il rapporto inverso – quello tra ordini di vendita e di acquisto non conclusi – che si colloca su valori elevati.

Nonostante la forte volatilità del mercato, il previsore neurale riesce, tuttavia, a prevedere meglio l'andamento del mercato (la percentuale di stime corrette è pari al 53%). Per quanto riguarda l'effetto sulla ricchezza – a fronte dei più ampi movimenti del prezzo - si verifica un forte incremento del livello potenziale dei guadagni (e delle perdite) conseguibili dagli agenti.

[127] In definitiva le bolle e i crash che si formano nel mercato sono determinate dalla presenza di lunghe sequenze di ordini ineseguiti, su uno stesso lato del mercato, per un arco di tempo più o meno esteso.

[128] L'incremento di prezzo più elevato si verifica il giorno 275; il prezzo in un solo giorno triplica il proprio valore (2,75); il rapporto tra ordini di acquisto e di vendita non eseguiti è pari a circa 15.

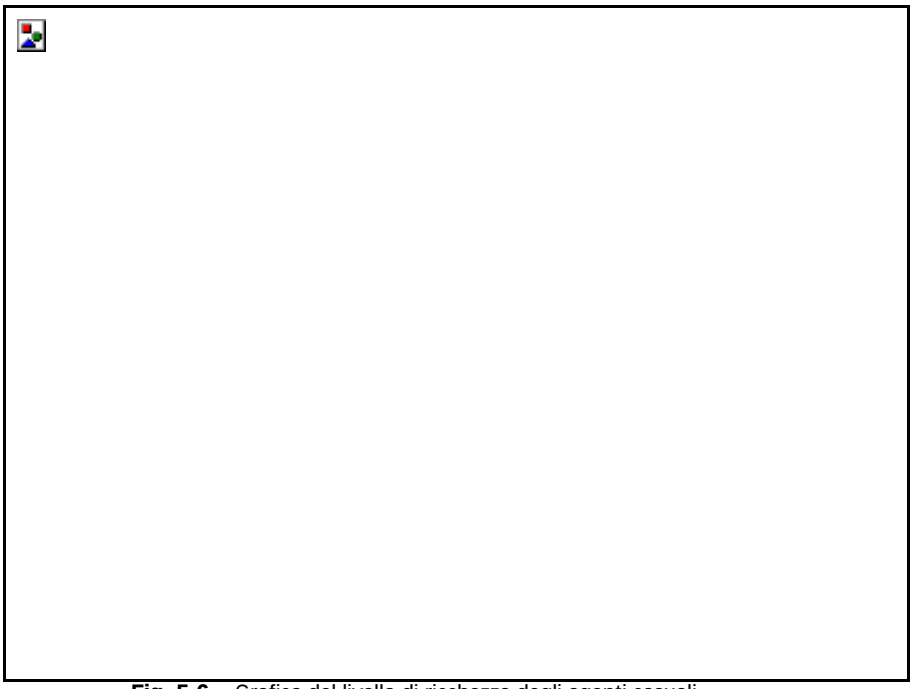


Fig. 5.6 – Grafico del livello di ricchezza degli agenti casuali.

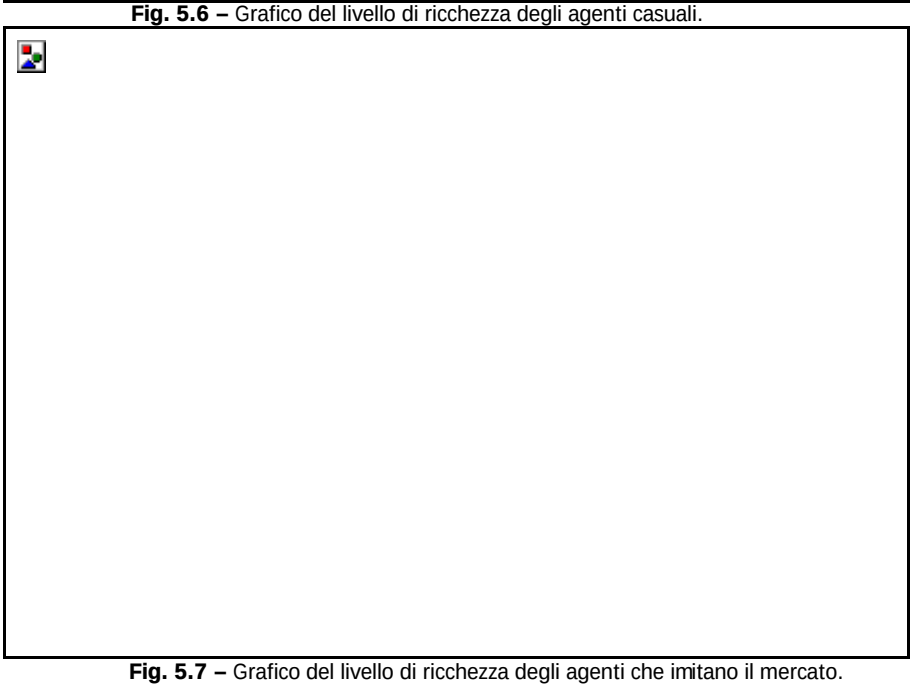


Fig. 5.7 – Grafico del livello di ricchezza degli agenti che imitano il mercato.

L'introduzione degli agenti che imitano il mercato determina rilevanti conseguenze sulle dotazioni dei singoli agenti; come è possibile notare dal grafico 5.7, l'adozione di strategie imitative del mercato consente di limitare il livello di perdite nei periodi di *crash* successivi alle bolle speculative; tali agenti tuttavia non riescono a raggiungere i livelli di guadagno degli agenti casuali.

Risultati differenti si ottengono includendo nel modello agenti che, invece di imitare il mercato, imitano localmente; tali agenti non hanno una visione globale del mercato, ma conoscono soltanto le azioni degli ultimi *n* agenti che hanno operato.

Fig. 5.8 – prezzo corrente con 285 agenti casuali e 15 agenti che imitano localmente (maxOrderQuantity=3).

Il mercato che emerge eseguendo il modello con 285 agenti random e 15 imitatori locali[129] presenta bolle meno intense rispetto al caso precedente, ma risulta anche meno prevedibile: la percentuale di previsioni corrette del previsore neurale è pari, infatti, solo al 47%. Per quanto riguarda l'effetto sulla ricchezza degli agenti, come è possibile notare dai grafici riportati di seguito gli imitatori locali riescono a limitare più efficacemente le perdite durante i periodi di 'sgonfiamento' delle bolle speculative rispetto agli agenti casuali; essi, inoltre, mostrano un livello medio di ricchezza superiore.

A differenza degli imitatori del mercato, gli imitatori locali sembrano maggiormente in grado di cogliere le opportunità di guadagno durante le fasi di crescita del mercato di borsa. L'introduzione nel modello di agenti che imitano localmente determina una dinamica del mercato che alterna periodi relativamente stabili a periodi più agitati caratterizzati da bolle speculative di entità consistente, ma comunque nettamente inferiori a quelle che si registrano nel caso degli imitatori globali.

[129] Gli agenti che imitano localmente comprano (vendono) con probabilità pari a *asymmetricBuySellProb* (0,9) se la maggioranza degli ultimi *n* (20) agenti ha comprato (venduto) azioni.

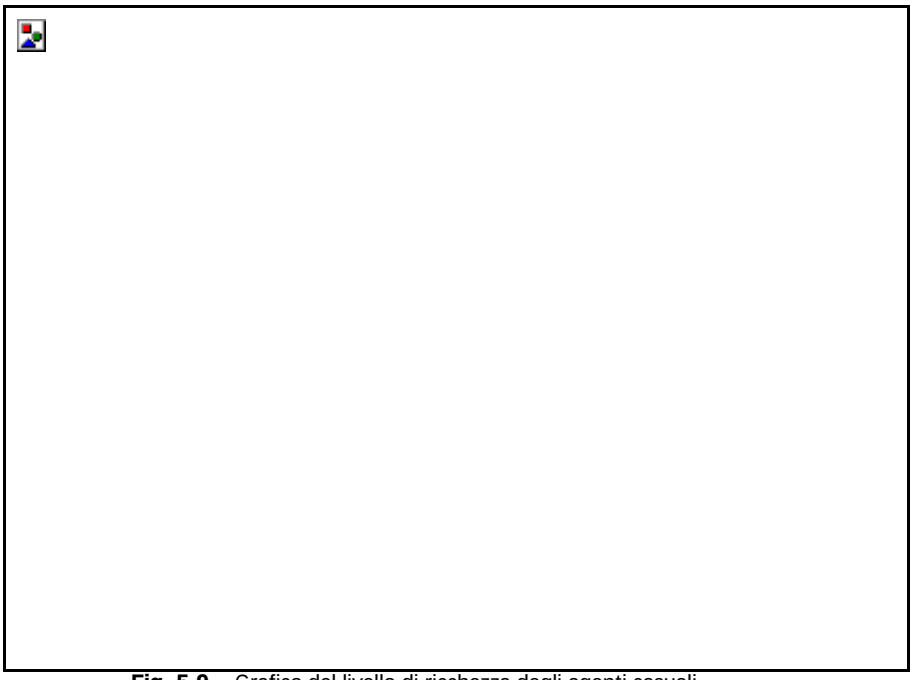


Fig. 5.9 – Grafico del livello di ricchezza degli agenti casuali.

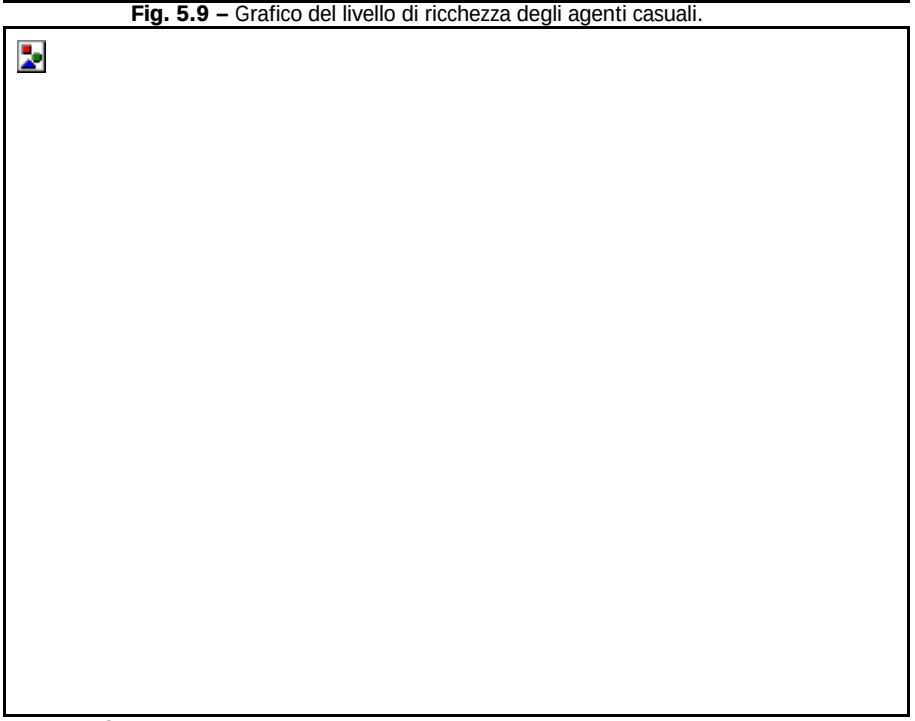


Fig. 5.10 – Grafico del livello di ricchezza degli agenti che imitano localmente.

Una analoga dinamica si registra facendo 'lavorare' il modello con 285 agenti casuali e 15 agenti che operano sulla base di strategie di tipo stop loss[130].

Nella simulazione condotta gli agenti *stop loss* se hanno una posizione lunga (corta) e se il prezzo corrente *t* è diminuito (aumentato), rispetto al prezzo medio del giorno *t-2* ad un tasso uguale o maggiore

[130] Per rendere più realistica la simulazione, gli agenti stop loss operano tenendo in considerazione la propria posizione speculativa (*checkingIfShortOrLong*=1).

del 10% decidono di vendere (comprare) azioni per un quantitativo compreso tra il lotto minimo e la massima quantità negoziabile.

Come mostrato in figura 5.11, l'introduzione degli agenti *stop loss* all'interno del mercato artificiale, determina bolle speculative di una certa entità; nella fase speculativa più intensa il livello del prezzo è, infatti, pari a circa 13 volte il prezzo iniziale.

Il mercato, rispetto alla situazione precedente, risulta più prevedibile: la quota di previsioni corrette del previsore neurale è pari al 54%.

Fig. 5.11 – prezzo corrente con 285 agenti casuali e 15 agenti stop loss (maxOrderQuantity=3).

Per quanto riguarda l'effetto sulla ricchezza, gli agenti casuali mostrano una dinamica simile a quella evidenziata nelle simulazioni precedenti: enormi guadagni per alcuni, forti perdite per altri.

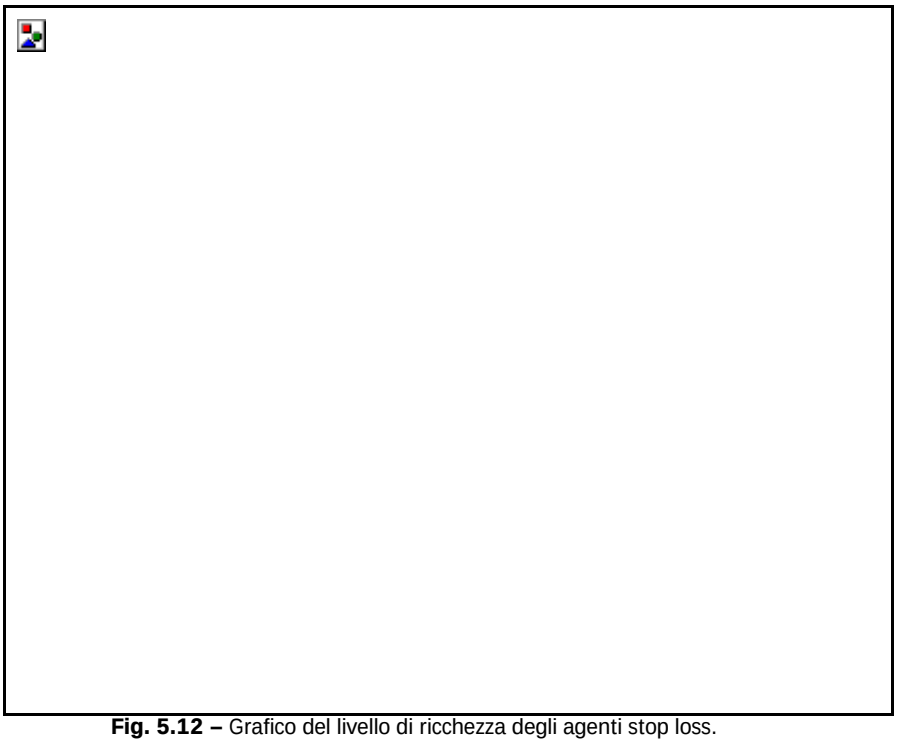


Fig. 5.12 – Grafico del livello di ricchezza degli agenti stop loss.

Il livello di ricchezza degli agenti *stop loss*, invece, mostra un andamento estremamente interessante che rappresenta una conseguenza diretta dell'adozione di una strategia di investimento fortemente difensiva: il livello di ricchezza è sempre prossimo allo zero e non risente minimamente delle bolle e dei *crash* che si verificano nel mercato.

5.3 Agenti che applicano le previsioni neurali (*ANNForecastAppAgent*)

In questo paragrafo viene indagato l'effetto dell'introduzione, nel mercato di borsa, di agenti che decidono di acquistare o vendere azioni sulla base delle stime dell'andamento futuro dei prezzi elaborate dal previsore neurale. Con l'introduzione di questo tipo di agenti, il modello si avvicina molto al campo delle previsioni autorealizzanti; la qualità delle stime prodotte dall'agente neurale[131] dipende, infatti, dal grado di prevedibilità del mercato; se un numero elevato di agenti applica le previsioni dell'agente neurale, la naturale conseguenza è un forte irrobustimento del livello di prevedibilità del mercato stesso.

[131] Nelle simulazioni condotte il previsore neurale produce stime del prezzo futuro (espresse sotto forma di indice) a 10 giorni, usando una rete neurale artificiale con una finestra di 30 dati in input; i dati sono costituiti da indici calcolati come rapporti tra la media giornaliera del prezzo del giorno *t* e la media giornaliera del prezzo del giorno *t-10*. La rete neurale ha un training set di 100 esempi (contenenti input e output attesi); il numero d epoche di apprendimento in ogni ciclo di learning è pari a 100, il processo di apprendimento viene ripetuto ogni 10 giorni e ogni 50 giorni la rete neurale viene 'rinizializzata'.

Fig. 5.13 – Grafico del prezzo con 285 randomAgent, 15 *aNNFoercastAppAgen*t.

Gli agenti *ANNForecastAppAgent*, che agiscono ogni giorno con probabilità del 10%(modificabile dall'utente dell'applicazione), decidono di acquistare o di vendere azioni coerentemente con le stime prodotte dal previsore neurale; l'agente, cioè, acquista azioni se la stima del previsore neurale è maggiore del prezzo corrente; vende nel caso contrario[132]. In figura 5.13 è riportata la dinamica del prezzo che si ottiene facendo 'lavorare' il modello con un numero limitato di agenti *ANNForecastAppAgent* (15). Il prezzo mostra un andamento relativamente regolare con bolle speculative di entità modesta. Il livello di prevedibilità del mercato è pari al 52%.

Per quanto riguarda l'effetto sulla ricchezza degli agenti, dall'analisi delle figure riportate di seguito, è possibile notare la maggiore stabilità del profilo di ricchezza degli agenti che seguono le previsioni neurali rispetto agli agenti casuali.

[132] Più in dettaglio gli agenti decidono di acquistare (vendere) azioni se l'indice del prezzo previsto è maggiore (inferiore) di 1+*aNNInactivityRange* (1-*aNNInactivityRange*); nelle simulazioni eseguite *aNNInactivityRange* è posto pari a 0,02 (viene, inoltre, effettuato un controllo di coerenza in modo da evitare che l'agente acquisti azioni quando il prezzo corrente è già oltre il prezzo previsto).

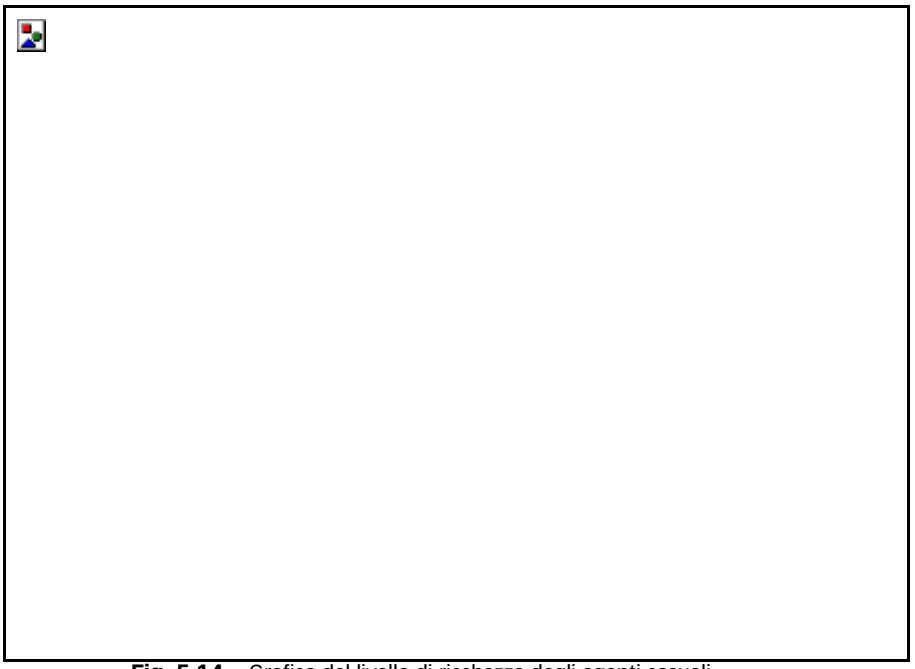


Fig. 5.14 – Grafico del livello di ricchezza degli agenti casuali.

Fig. 5.15 – Grafico del livello di ricchezza degli agenti ANNForecastAppAgent.

La ricchezza media degli agenti *ANNForecastAppAgent* sembra mostrare una dinamica più orientata verso i guadagni; nella dinamica degli agenti casuali, invece, le perdite sembrano prevalere. Se nel modello vengono introdotti un numero limitato di agenti *ANNForecastAppAgent*, emerge un mercato relativamente stabile con bolle di entità contenuta ed una limitata variabilità nel livello dei prezzi.

Una dinamica totalmente differente emerge, invece, se il numero di agenti che applicano le stime del previsore neurale aumenta; se il numero degli agenti che applicano la previsione neurale viene raddoppiato (*aNNFoercastAppAgen*t = 30), l'andamento del prezzo risulta decisamente più 'turbolento';

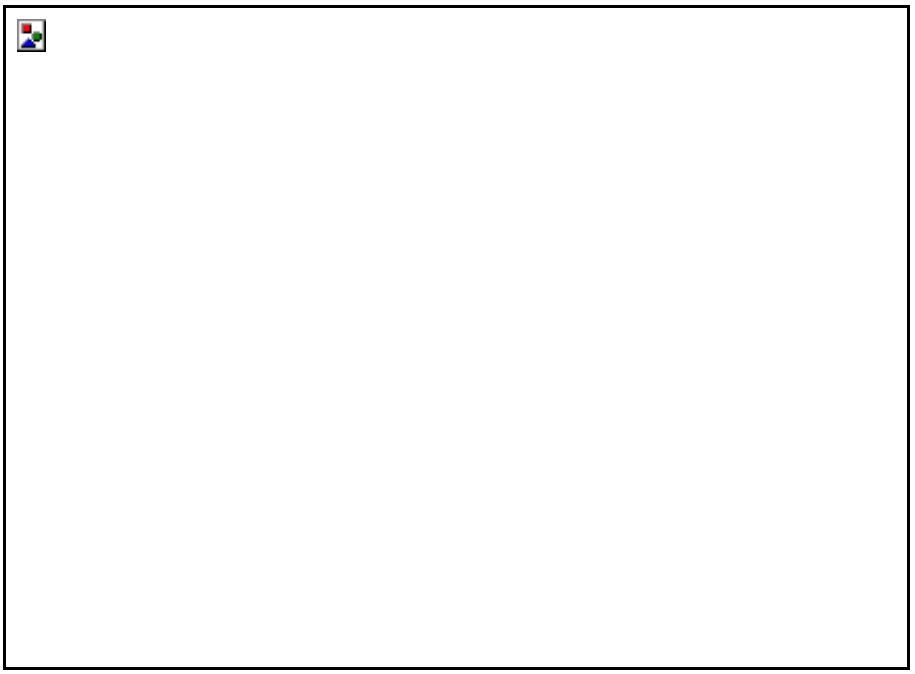


Fig. 5.16 – Grafico del prezzo con 270 randomAgent, 30 *aNNFoercastAppAgen*t (maxOrderQuantity pari a 3).

come è possibile notare dal grafico riportato in figura 5.16, il mercato mostra una enorme bolla speculativa, all'apice della quale il livello del prezzo raggiunge un valore pari a quasi ottocento volte il prezzo iniziale. Nonostante la dinamica 'esplosiva' del prezzo, il livello di prevedibilità cresce sensibilmente (58%), il che appare indicativo di un processo di autorealizzazione delle aspettative.

Uno degli obiettivi principali del modello SUM è l'analisi delle conseguenza dell'introduzione nel mercato artificiale di borsa di agenti con comportamenti omogenei e di agenti 'cognitivi' - costruiti secondo la metodologia dei cross target - dotati di proprie strategie di comportamento autosviluppate.

Per valutare gli effetti che il comportamento di tali agenti determina sul mercato borsistico artificiale occorre definire una struttura di base del mercato che presenti le seguenti caratteristiche: un buon livello di prevedibilità, una dinamica del prezzo non esplosiva e la presenza di bolle di entità contenuta.

Nella figura 5.17 è riportata la dinamica del prezzo in un mercato di borsa 'popolato' da 275 agenti casuali, da 15 agenti *ANNForecastAppAgent* e da 10 agenti che adottano strategie di stop loss. Il mercato soddisfa le caratteristiche elencate in precedenza; esso, infatti, appare sostanzialmente stabile, non evidenzia bolle speculative troppo accentuate e si caratterizza per un buon livello di prevedibilità: la percentuale di previsioni corrette formulate dall'agente neurale si attesta, infatti, al 55%.

Fig. 5.17 – Grafico del prezzo con 275 randomAgent, 15 *aNNFoercastAppAgen*t e 10 stop loss.

5.4 Una provocazione: l'agente astrologo

Prima di analizzare il comportamento degli agenti cognitivi il modello viene esteso introducendo nel mercato artificiale di borsa una nuova tipologia di individui: gli agenti che seguono le previsioni sull'andamento del mercato formulate da un 'astrologo'.

L'introduzione provocatoria di questi agenti si pone come obiettivo quello di indagare le possibili conseguenze sulla dinamica del prezzo del comportamento di un gruppo omogeneo di individui che agiscono sulla base di valutazioni che sono indipendenti dal contesto e dalla situazione 'reale' del mercato.

A differenza degli agenti *ANNForecastAppAgent*, che operano seguendo la previsione formulata da una rete neurale che ha come input gli indici di variazione dei prezzi passati, gli agenti *ASTROForecastAppAgent* seguono una previsione sull'andamento futuro del mercato priva di qualsiasi relazione con la dinamica passata e presente del mercato stesso, ma che si fonda esclusivamente su valutazioni di tipo casuale.

L'introduzione di tali agenti si basa sulla considerazione che nei mercati di borsa reali, operano anche individui con strategie di investimento che sono spesso il frutto di valutazioni soggettive che poco hanno a che fare con valutazioni di tipo economico.

In teoria, l'introduzione nel mercato artificiale di borsa di un numero limitato di agenti *ASTROForecastAppAgent* non dovrebbe determinare ripercussioni rilevanti sulla dinamica del prezzo; se, tuttavia, il numero di 'seguaci' dell'astrologo cresce e acquista consistenza in rapporto al numero totale degli agenti che popolano il mercato, allora è probabile che si verifichino andamenti 'particolari' del prezzo.

Nell'esempio che segue, viene analizzata la dinamica del prezzo e della ricchezza in un mercato popolato da 260 agenti casuali, 10 agenti con strategie di investimento di tipo *stop loss*, 15 agenti che applicano la previsione

dell'agente neurale e 15 agenti che seguono fedelmente la previsione (a 10 giorni) sull'evoluzione futura del mercato formulata dall'agente astrologo[133].

La dinamica del prezzo risulta relativamente stabile, il che conferma la scarsa influenza dell'azione degli agenti *ASTROForecastAppAgent* sul mercato; a parte una fase speculativa di una certa consistenza (prezzo massimo pari a circa 12 volte il prezzo iniziale) e durata (circa 300 giorni), il prezzo rimane per lunghi periodi su livelli che non superano il valore di 2.5 volte il prezzo iniziale. Il mercato mostra un livello di prevedibilità pari al 52%.

Fig. 5.18 – Grafico del prezzo con 260 *randomAgent*, 15 *ANNFoercastAppAgen*t, 10 stop loss e 15 *ASTROForecastAppAgent*.

Gli andamenti dei livelli di ricchezza degli agenti casuali e degli agenti stop loss sono pressoché analoghi a quelli riscontrati nelle precedenti simulazioni: gli agenti casuali possono conseguire importanti profitti, ma anche subire forti perdite, gli agenti stop loss minimizzano, invece, la variabilità del livello di ricchezza riducendo notevolmente sia le possibilità di guadagno che quelle di perdita[134].

[133] Gli agenti *ANNForecastAppAgent* e *ASTROForecastAppAgent* operano ogni giorno con probabilità del 10% (gli agenti che seguono la previsione formulata dall'astrologo possono, inoltre, con probabilità del 5% agire sulla base di una previsione personalizzata).

[134] Gli agenti casuali possono realizzare guadagni (e perdite) massimi pari a 2000; gli agenti stop loss, invece, possono realizzare guadagni e perdite massime che non superano in valore assoluto 150.

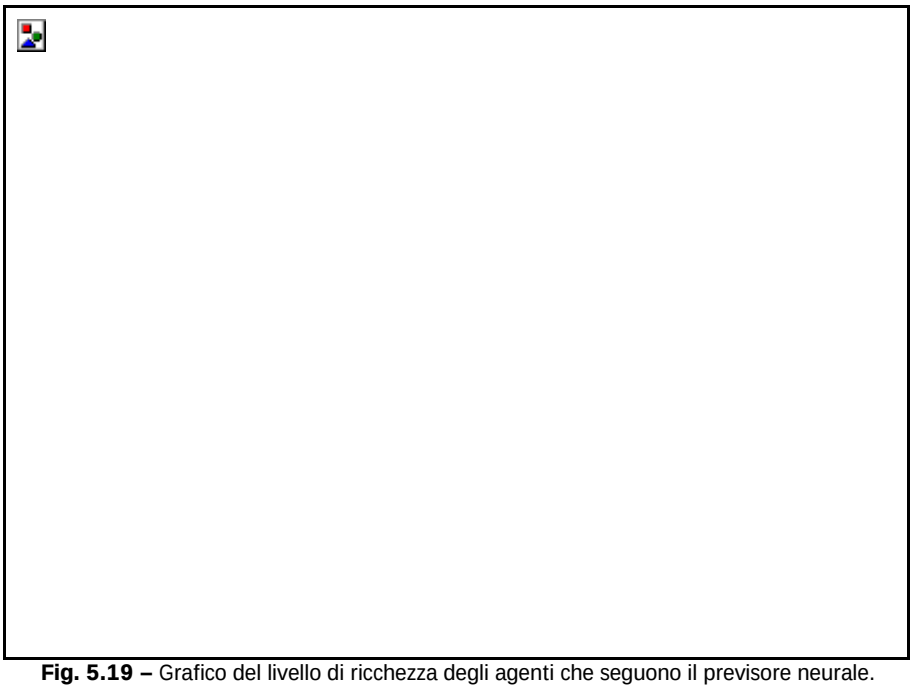


Fig. 5.19 – Grafico del livello di ricchezza degli agenti che seguono il previsore neurale.

Per quanto riguarda gli agenti che seguono la previsione della rete neurale, il grafico riportato in figura 5.19 mostra come tali agenti siano in grado di ottenere buoni livelli di ricchezza e di limitare le perdite potenziali entro limiti accettabili; il livello medio di ricchezza mostra, inoltre, una dinamica quasi sempre positiva.

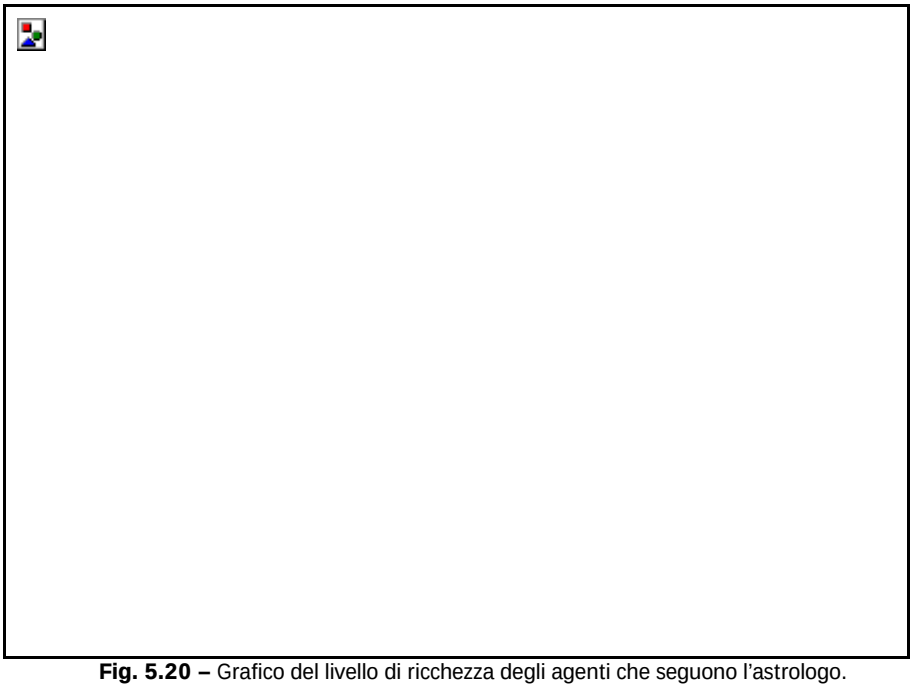


Fig. 5.20 – Grafico del livello di ricchezza degli agenti che seguono l'astrologo.

Sorprendente è l'analisi del grafico relativo al livello di ricchezza degli agenti che seguono l'astrologo. Tali agenti, infatti, sono quelli che, nell'ambito del mercato artificiale di borsa definito in questa simulazione, riescono a conseguire il più efficiente compromesso tra potenzialità di guadagno e rischi di perdita e ad approfittare meglio della fase di forte crescita del mercato che si verifica durante la bolla speculativa.

A fronte di guadagni potenziali che possono raggiungere il livello di 800, le perdite massime non superano il valore di 200; il livello medio di ricchezza si colloca, inoltre, su livelli nettamente superiori a quelli degli altri tipi di agenti.

Il comportamento particolarmente positivo degli agenti *ASTROForecastAppAgent* - soprattutto se confrontato con quello degli agenti casuali e degli agenti che seguono le previsioni neurali - può trovare alcune spiegazioni nelle seguenti considerazioni: gli agenti che seguono l'astrologo, a differenza degli agenti casuali, non operano ogni giorno sul mercato e non modificano continuamente la propria posizione speculativa; hanno una strategia di investimento più stabile e ciò li espone di conseguenza ad una minore variabilità dei risultati; gli agenti neurali - pur ottenendo buone prestazioni - si collocano ad un livello inferiore agli agenti che seguono le previsioni formulate dall'astrologo perché vengono penalizzati dal livello relativamente basso (52%) di prevedibilità del mercato.

Occorre, poi, sottolineare che - data la forte influenza di fattori casuali e la dinamica che si colloca al limite del 'caos' - i risultati delle singole simulazioni hanno una validità 'individuale'; non possono, cioè essere considerate come dimostrazioni di validità di carattere generale.

I risultati si modificano profondamente all'aumentare del numero di agenti che applicano le previsioni dell'astrologo. Nella simulazione seguente, il modello è popolato da 225 agenti random, 10 agenti *stop loss*, 15 agenti che seguono le previsioni neurali e 50 agenti *ASTROForecastAppAgent*.

Fig. 5.21 – Grafico del prezzo con 225 *randomAgent*, 15 *ANNFoercastAppAgen*t, 10 stop loss e 50 *ASTROForecastAppAgent*.

Come è possibile notare dal grafico riportato in figura 5.21, in questo caso l'andamento del prezzo mostra una maggiore variabilità; per lunghi periodi la dinamica risulta, ancora, relativamente stabile; il mercato, tuttavia, evidenzia una fortissima bolla speculativa, all'apice della quale il prezzo raggiunge il livello di circa 80 volte il prezzo iniziale e altre bolle di entità più contenuta che si collocano su livelli di circa 10 volte il prezzo iniziale. Il livello di prevedibilità del mercato è pari al 55,6%.

L'omogeneità di comportamento degli agenti *ASTROForecastAppAgen*t determina una maggiore intensità delle bolle; se il numero di agenti che segue fedelmente le indicazioni dell'astrologo è quantitativamente rilevante, allora le aspettative sull'evoluzione futura del mercato di tali agenti - siano esse giustificabili economicamente o

totalmente infondate - tendono inevitabilmente ad autorealizzarsi in quanto seguite e applicate da una larga parte del mercato.

Una conferma della forte influenza esercitata dagli agenti *ASTROForecastAppAgen*t sul mercato e sulla dinamica del prezzo viene dall'analisi delle correlazioni tra l'andamento del prezzo e le previsioni dell'astrologo; il coefficiente di correlazione lineare di Bravais-Pearson tra le due variabili è pari a 0,21. Se, poi, si prendono in considerazione non tutti i dati giornalieri, ma solo quelli al termine del periodo di previsione (la previsione dell'astrologo rimane costante per tutta la lunghezza del periodo di previsione, 10 giorni) e si calcola la correlazione tra le previsioni dell'astrologo e gli andamenti del prezzo tra l'inizio del periodo di previsione e la fine, si ottiene un coefficiente di correlazione pari a 0,50.

Il coefficiente di determinazione R^2 della regressione pi = co + c1b + c2 s, calcolato sui dati generati in questa simulazione risulta pari a 0.3939; lo squilibrio tra ordini di acquisto e di vendita 'spiega' quindi una buona parte della varianza totale del movimento dei prezzi riscontrato sul mercato. Se si restringe l'orizzonte temporale della regressione, invece che all'intera simulazione, al periodo caratterizzato dalla formazione delle bolle speculative il coefficiente R^2 sale a 0,58[135].

Per quanto riguarda il livello di ricchezza, gli agenti *ASTROForecastAppAgen*t mostrano – proprio in virtù del processo di realizzazione delle aspettative – una dinamica estremamente positiva: essi sono in grado di cogliere, meglio di tutti gli altri agenti, le possibilità di guadagno offerte dalle bolle speculative (che con il loro comportamento hanno contribuito in maniera decisiva a determinare).

Osservando i grafici relativi alla ricchezza è possibile notare come il livello medio di ricchezza degli agenti che seguono l'astrologo si collochi su valori sempre positivi a differenza di quanto invece avviene per gli agenti che seguono le previsioni neurali; questi ultimi accusano forti perdite durante la prima intensa bolla speculativa e perdite seppur più contenute durante le successive bolle.

[135] Al fine di individuare ulteriori fattori determinanti nella spiegazione delle bolle è stato analizzato il ruolo giocato dai volumi scambiati; in alcuni casi, infatti, è possibile osservare, come nell'ambito di un mercato 'sottile' la prevalenza degli ordini ineseguiti su un lato del mercato amplifichi i propri effetti sulla dinamica del prezzo; tuttavia, analizzando i dati generati dalla simulazione nel complesso il legame tra 'spessore' del mercato e andamento del prezzo non trova conferma.

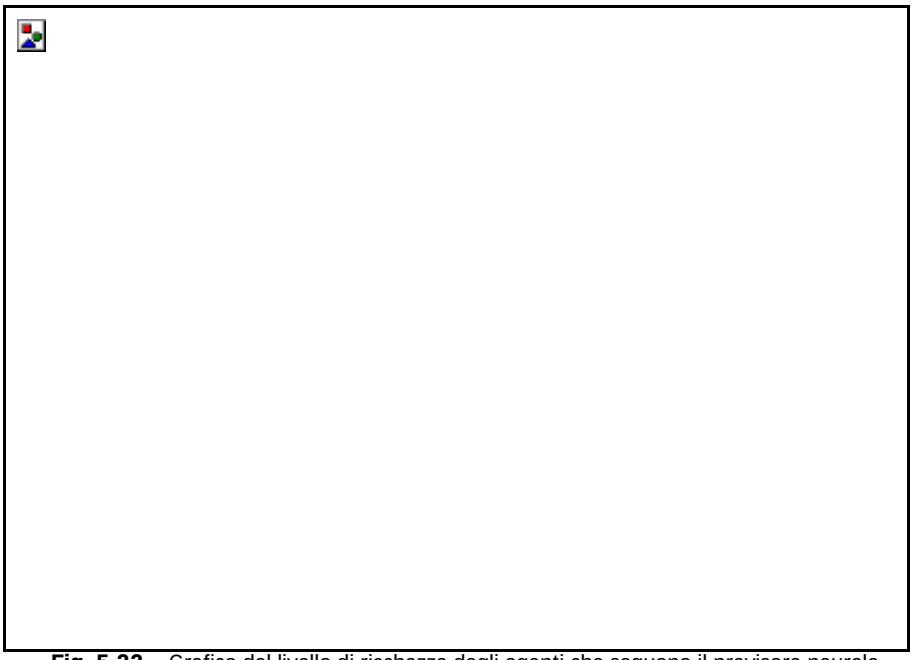


Fig. 5.22 – Grafico del livello di ricchezza degli agenti che seguono il previsore neurale.

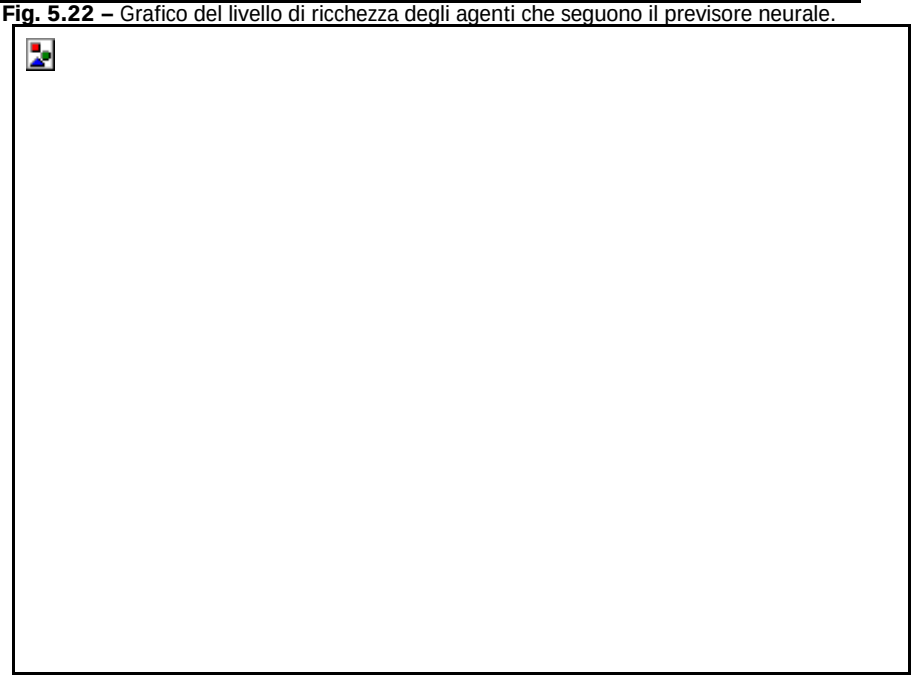


Fig. 5.23 – Grafico del livello di ricchezza degli agenti che seguono l'astrologo.

La rete neurale sembra in grado di produrre previsioni corrette nelle fasi relativamente stabili, ma risulta, invece, incapace di prevedere (e quindi di approfittare) delle fasi di turbolenza del mercato nelle quali le perdite degli agenti neurali si trasformano nei guadagni degli agenti che seguono l'astrologo.

Un aspetto interessante che merita attenzione è la dinamica delle dotazione dei singoli agenti; nelle figure 5.24 e 5.25 sono riportati i grafici dell'andamento nel tempo della liquidità, della quantità di azioni e della ricchezza di un agente che applica le previsioni neurali e di un agente che segue l'astrologo.

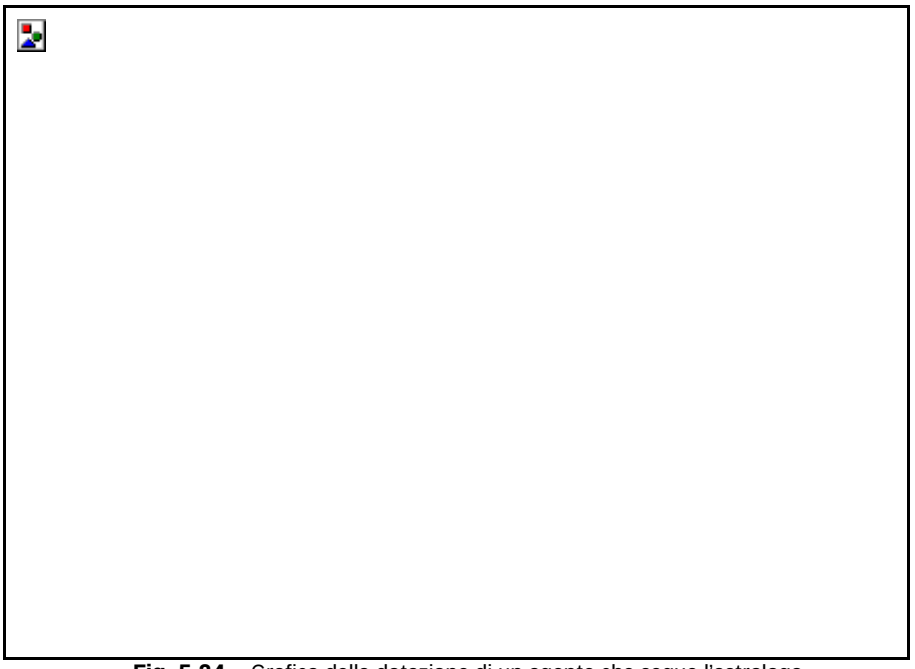


Fig. 5.24 – Grafico della dotazione di un agente che segue l'astrologo.

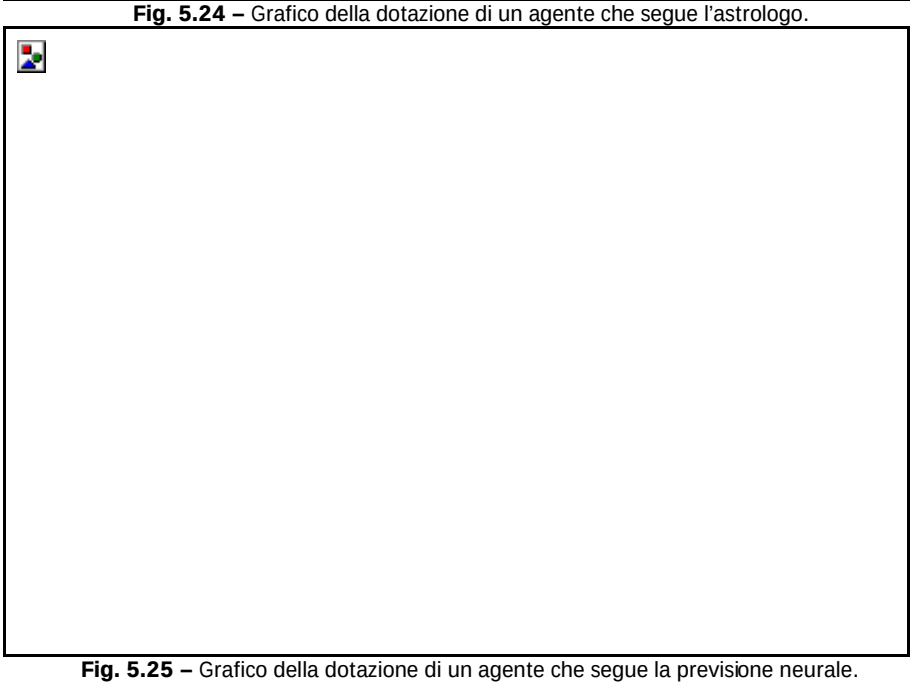


Fig. 5.25 – Grafico della dotazione di un agente che segue la previsione neurale.

È possibile notare il comportamento quasi speculare delle due differenti tipologie di agenti: l'agente neurale, dopo una iniziale fase positiva, durante la bolla speculativa vende azioni incamerando liquidità; l'agente *ASTROForecastAppAgent*, invece, acquista azioni indebitandosi fortemente. Poiché il numero di agenti che segue l'astrologo è rilevante, le aspettative relative ad un aumento del prezzo si realizzano determinando, così, l'arricchimento degli agenti *ASTROForecastAppAGent* che detengono una buona percentuale del loro portafoglio in azioni e un deciso impoverimento degli agenti neurali i quali, invece, hanno una posizione 'corta' sulle azioni.

Solo quando la bolla è in fase di piena 'maturità' la rete neurale riesce a correggere le proprie stime sull'andamento futuro del prezzo; gli agenti neurali rivedono allora la propria posizione e acquistano azioni; immediatamente dopo, però, si verifica un rapido 'sgonfiamento' della bolla speculativa - provocato

principalmente da un'inversione di tendenza delle aspettative future dell'astrologo - che coglie impreparata le rete neurale (la quale non è stata addestrata per una tale situazione); le azioni in portafogli perdono valore e si assiste ad una ulteriore forte diminuzione del livello di ricchezza dell'agente neurale.

I grafici riportati in precedenza si riferiscono al comportamento di due specifici agenti; cambiando gli agenti 'osservati' le dinamiche comportamentali possono differire.

I risultati ottenuti dagli agenti non dipendono, infatti, esclusivamente dalla strategia di investimento adottata, ma anche dal contesto specifico del mercato; si può, ad esempio, verificare che gli agenti non riescano ad attuare le azioni suggerite dalla rete neurale o dall'astrologo perché vengono anticipati dall'azione di altri agenti o perché non vi sono controparti disposte a scambiare le posizioni.

Di seguito sono riportati i grafici relativi ad altri due agenti; come è possibile notare il comportamento in parte differisce; l'agente *ANNForecastAppAgent* preso in considerazione in questo caso riesce ad approfittare della fase rialzista del mercato;

Fig. 5.26 – Grafico della dotazione di un (differente) agente che segue l'astrologo.

esaurita la fase di rialzo, l'agente tuttavia non riesce a riallocare il proprio portafoglio di investimento poiché la rete neurale non è in grado di prevedere la fase di sgonfiamento della bolla. Nelle successive bolle di entità più contenuta l'agente sembra però 'far tesoro' dell'esperienza passata e riesce a guadagnare leggermente.

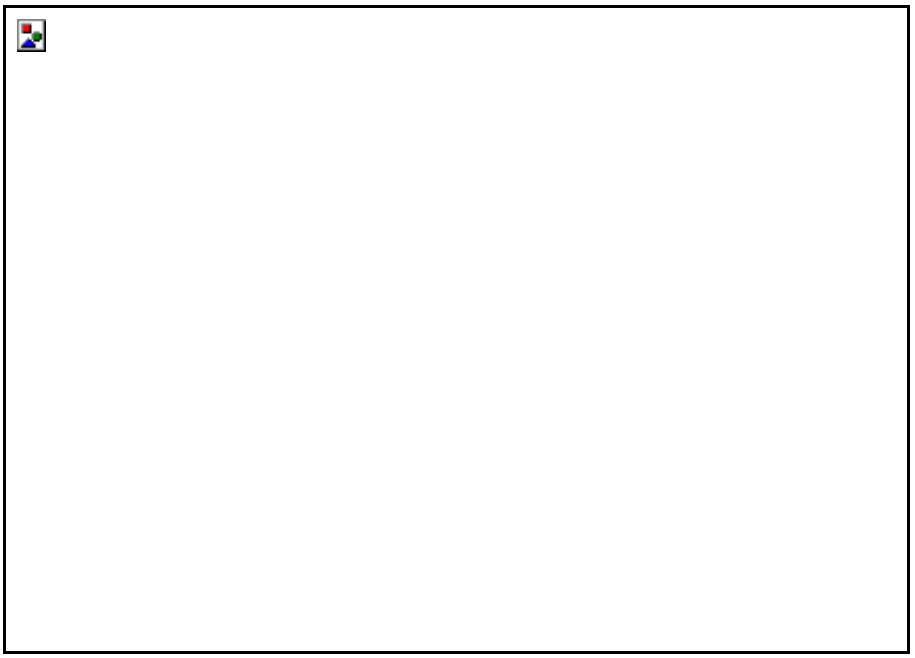


Fig. 5.27 – Grafico della dotazione di un (differente) agente neurale.

Nelle simulazioni condotte in precedenza la probabilità degli agenti *ASTROForecastAppAgen*t di operare sul mercato è del 10%; se si incrementa tale probabilità l'effetto autorealizzante delle aspettative dovrebbe rafforzarsi. Nella simulazione che segue, l'esperimento è stato ripetuto con la stessa struttura del mercato, ma modificando il valore del parametro *astroForecastAppAgentActDailyProb* che regola la probabilità di operare degli agenti che seguono l'astrologo (incrementandola da 0.10 a 0.20).

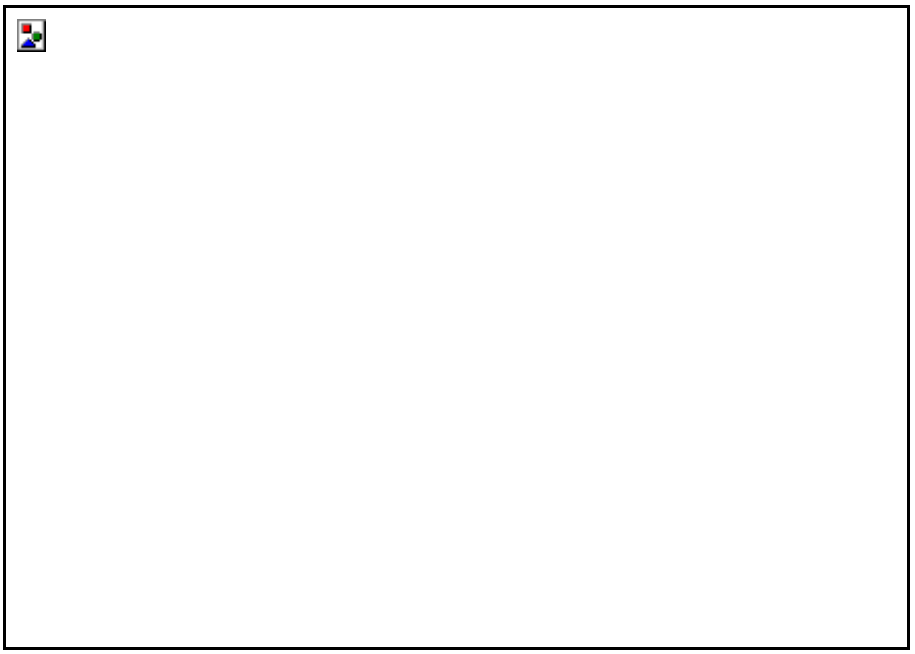


Fig. 5.28 – Grafico del prezzo con *astroForecastAppAgentActDailyProb* = 0.20.

Il risultato sembra confermare quanto affermato in precedenza: che vi sia in atto, cioè, un forte processo di autorealizzazione delle aspettative.

Il mercato che emerge mostra, infatti, una forte instabilità che riflette l'instabilità delle previsioni espresse dell'astrologo; le bolle speculative diventano più intense (il livello più alto raggiunge il valore di circa 150 volte il prezzo iniziale) e più frequenti (le bolle di entità rilevante passano dalle tre della precedente simulazione a più di dieci).

Il grado di prevedibilità del mercato scende considerevolmente, passando dal 55,6% al 49,7%: la rete neurale - data la forte variabilità delle aspettative espresse dall'astrologo - non riesce a stimare l'evoluzione futura del prezzo.

La conferma della forte influenza esercitata dagli agenti *ASTROForecastAppAgen*t sul mercato e sulla dinamica del prezzo viene dall'analisi delle correlazioni tra i prezzi e le previsioni dell'astrologo; in questo caso il coefficiente di correlazione lineare di Bravais-Pearson tra le due variabili è pari a 0.38.

Se, poi, si prendono in considerazione non tutti i dati giornalieri, ma solo quelli al termine del periodo di previsione e si calcola la correlazione tra le previsioni dell'astrologo e gli andamenti del prezzo tra l'inizio del periodo di previsione e la fine, si ottiene un coefficiente di Bravais-Pearson pari a 0,63.

Anche in questa simulazione emerge chiaramente come le bolle speculative e i relativi crash siano determinati da forti squilibri tra gli ordini di acquisto e quelli di vendita.

Il coefficiente di determinazione R^2 della regressione pi = co + c1b + c2 s è pari a 0.49; lo squilibrio tra ordini di acquisto e di vendita 'spiega', quindi, quasi la metà della varianza totale del movimento dei prezzi riscontrato sul mercato.

Se si restringe l'orizzonte temporale della regressione, invece che all'intera simulazione, al periodo caratterizzato dalla formazione delle bolle speculative più intense il coefficiente R^2 sale a 0,63.

5.5 Agenti cognitivi

In questo paragrafo vengono introdotti gli agenti 'cognitivi' - realizzati con la tecnica dei cross target - in grado di sviluppare autonomamente proprie regole di comportamento.

Gli esperimenti condotti con gli agenti *BPCTAgentB*[136] sono stati eseguiti variando il valore del parametro *agentBEO_EPDelta* per rendere più o meno importante l'obiettivo esterno di aumentare la ricchezza valutata sul prezzo previsto.

Nella prima simulazione effettuata, il modello è stato popolato con: 270 agenti casuali, 10 agenti con strategie di stop loss, 15 agenti che seguono la previsione neurale e 5 agenti 'cognitivi' di tipo BPCTAgentB con l'obiettivo esterno di aumentare la ricchezza valutata sulla base del prezzo previsto e un valore del parametro *agentBEO_EPDelta* pari a 1.

Il mercato che emerge – figura 5.29 - risulta relativamente stabile.

[136] Il meccanismo di funzionamento degli agenti cognitivi BPCTAgentB è descritto nel quarto capitolo.

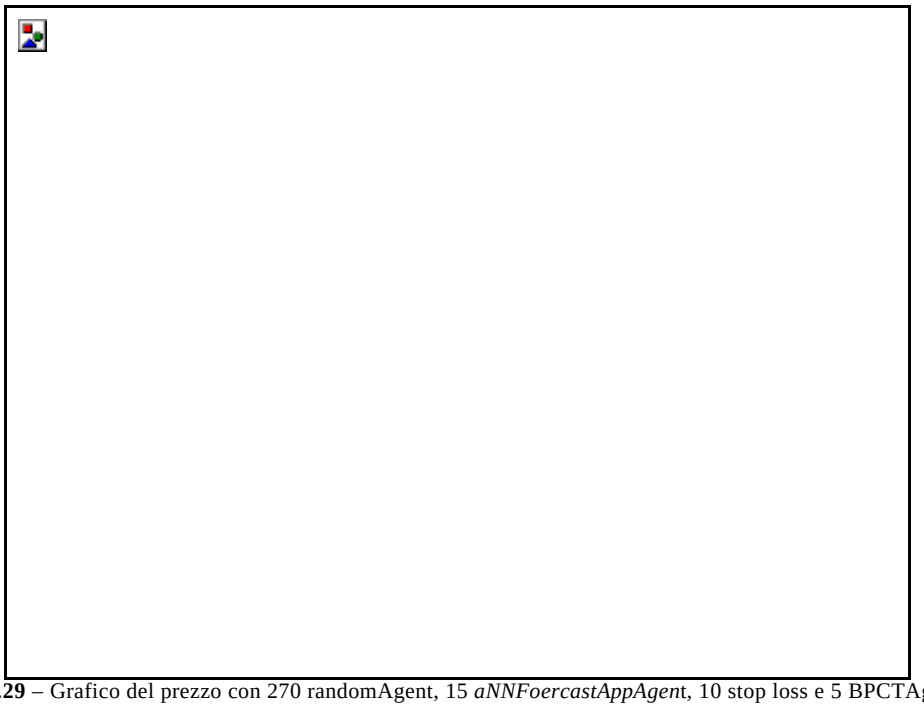


Fig. 5.29 – Grafico del prezzo con 270 randomAgent, 15 *aNNFoercastAppAgen*t, 10 stop loss e 5 BPCTAgentB.

Non si verificano bolle speculative di entità apprezzabile e il livello di prevedibilità del mercato si attesta al 52,5%. Come è possibile notare dal grafico riportato nella figura seguente i livelli di ricchezza degli agenti BPCTAgentB presentano, tuttavia, una dinamica estremamente insoddisfacente; il livello medio di ricchezza, per tutta la durata della simulazione, si colloca su valori negativi. Il valore di *agentBEO_EPDelta* pari a uno appare insufficiente a dare forza all'obiettivo esterno degli agenti di arricchirsi sulla base del prezzo previsto.

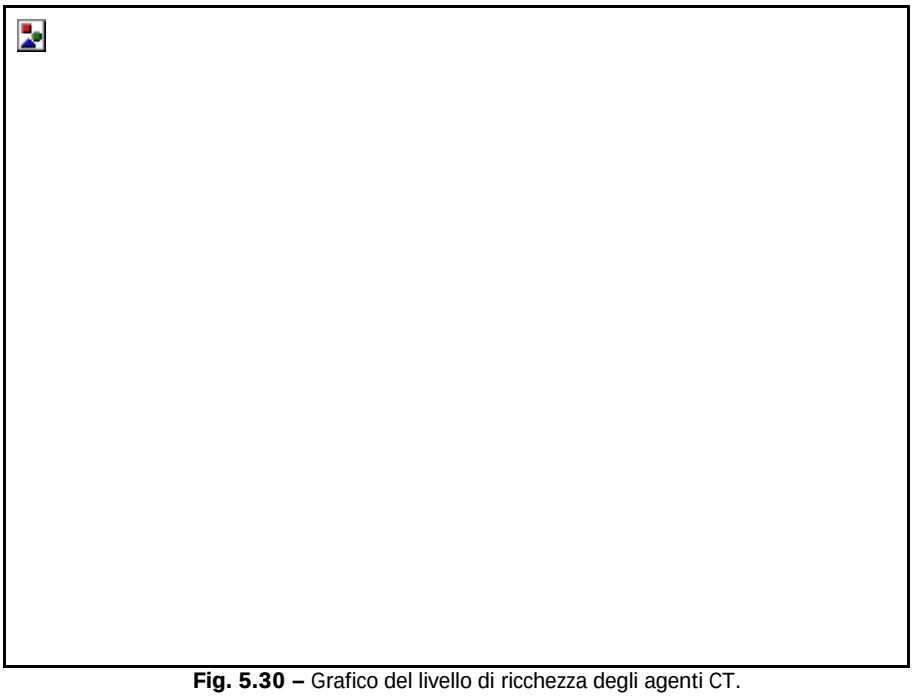


Fig. 5.30 – Grafico del livello di ricchezza degli agenti CT.

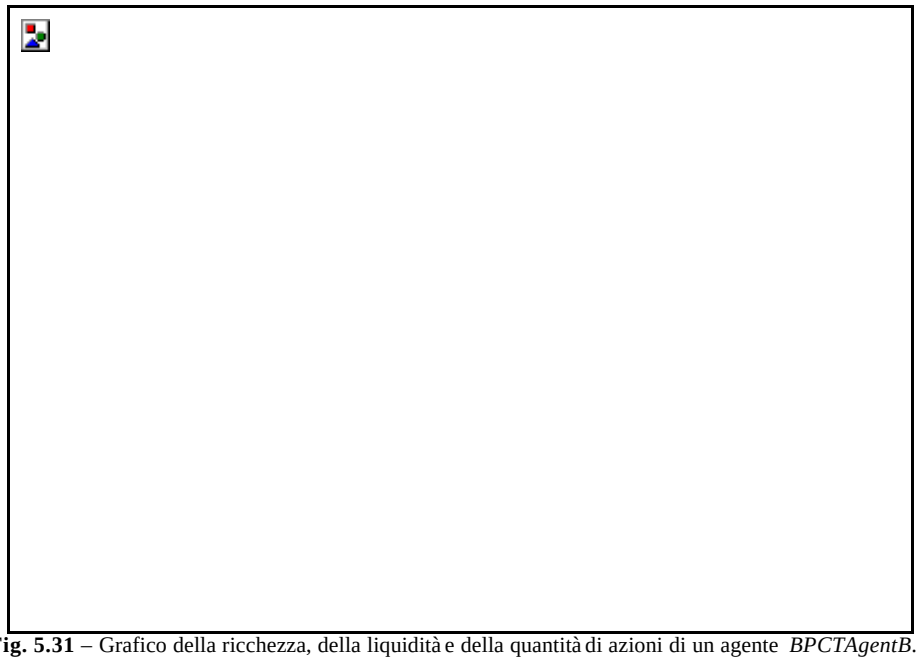


Fig. 5.31 – Grafico della ricchezza, della liquidità e della quantità di azioni di un agente *BPCTAgentB*.

L'esperimento è stato quindi ripetuto con la stessa configurazione del modello ma assegnando al parametro *agentBEO_EPDelta* il valore due in modo da rafforzare l'obiettivo esterno.

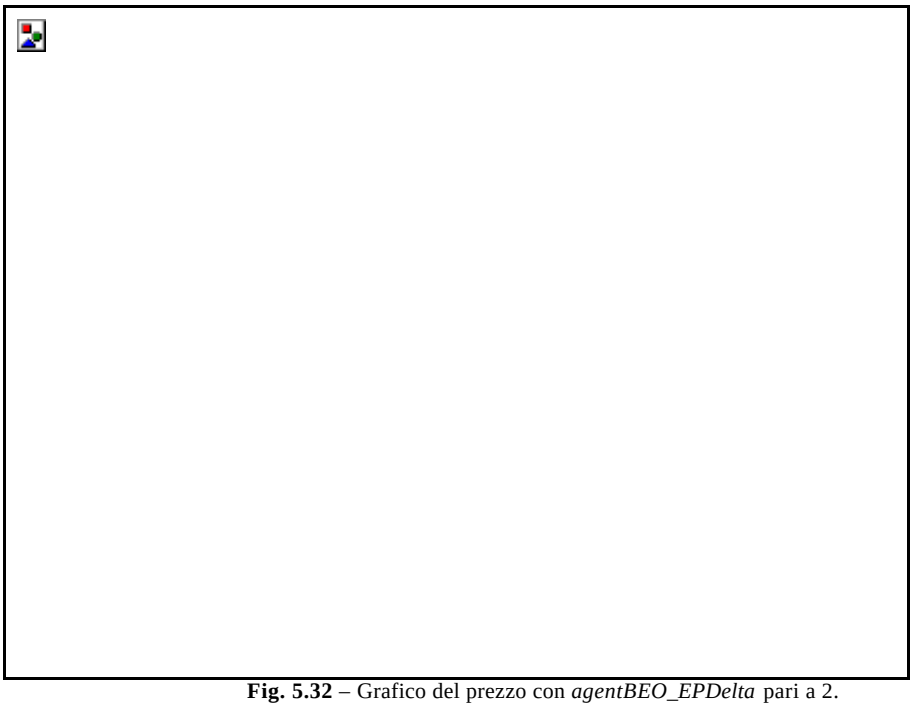


Fig. 5.32 – Grafico del prezzo con *agentBEO_EPDelta* pari a 2.

In questo caso la dinamica del prezzo risulta molto più interessante; il mercato - dopo una fase iniziale con andamento sostanzialmente stabile – mostra due bolle di entità rilevante.

Gli agenti cognitivi determinano un forte aumento del livello di prevedibilità del mercato; la quota di previsioni neurali corrette, nonostante una maggiore variabilità nei livelli del prezzo, cresce infatti

sensibilmente (60%). Gli agenti BPCTAgentB sono ora in grado di approfittare delle fasi di turbolenza del mercato.

Fig. 5.33 – Grafico della ricchezza, della liquidità e della quantità di azioni di un agente *BPCTAgentB*.

Come è possibile notare dalla figura 5.33, l'agente riesce a sfruttare l'andamento del prezzo e in particolare le bolle speculative per incrementare sensibilmente il proprio livello di ricchezza. L'agente acquista azioni nella prima fase di rialzo e non appena la bolla accenna a 'sgonfiarsi' le vende realizzando forti guadagni.

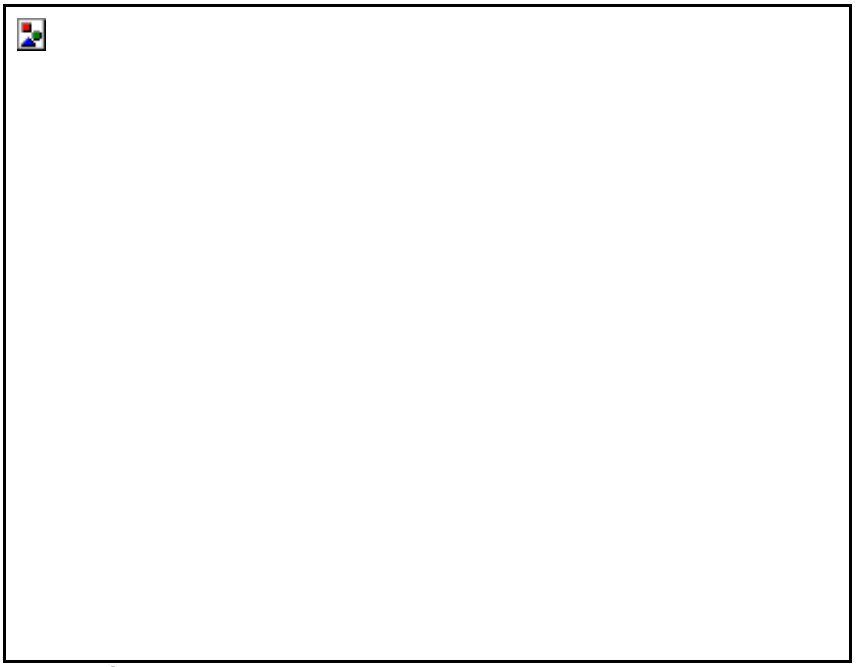


Fig. 5.34 – Grafico del livello di ricchezza degli agenti casuali.

Occorre sottolineare che il comportamento dell'agente viene completamente autosviluppato mediante l'uso della metodologia dei cross target; non viene cioè definito a priori, ma realizzato ex post.

Confrontando le figure 5.34, 5.35, 5.36, 5.37 relative ai livelli di ricchezza delle quattro diverse tipologie di agenti presenti nella simulazione emerge chiaramente come il valore *agentBEO_EPDelta* pari a due dia un significativo vantaggio al gruppo di agenti BPCTAgentB.

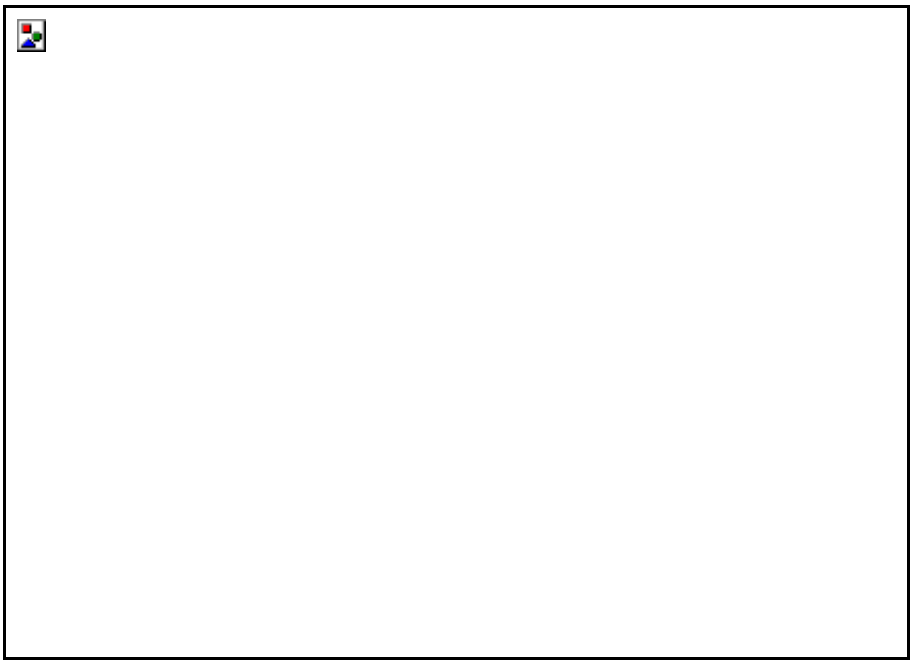


Fig. 5.35 – Grafico del livello di ricchezza degli agenti stop loss.

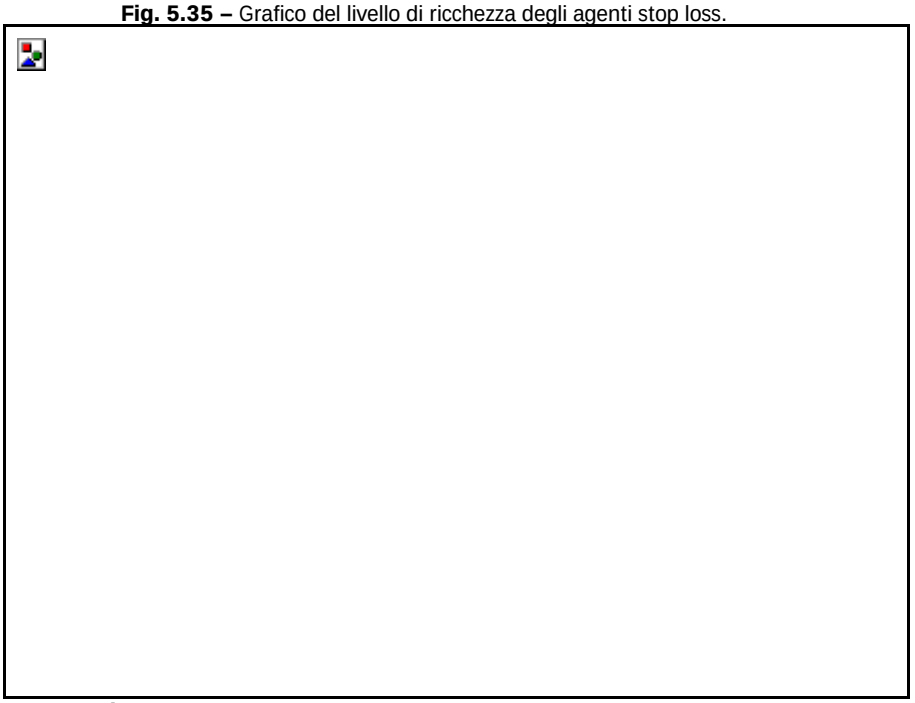


Fig. 5.36 – Grafico del livello di ricchezza degli agenti *ANNForecastAppAgent*.

Gli agenti cross target, assieme agli *ANNForecastAppAgent,* ottengono le migliori prestazioni.

Data l'elevata percentuale di previsioni corrette (60%), gli agenti che seguono la rete neurale riescono ad ottenere buoni livelli di ricchezza; gli agenti BPCTAgentB sono, però, preferibili in quanto – a

fronte di perdite potenziali leggermente superiori - si caratterizzano per livelli medi e massimi di ricchezza più elevati e per una maggiore stabilità delle prestazioni nel tempo.

Al fine di individuare i fattori determinanti del comportamento dell'agente cross target è stata effettuata una analisi statistica dei dati generati dalla simulazione mediante il programma R[137].

Come primo passo è stata calcolata la correlazione tra la decisioni di acquisto/vendita dell'agente cognitivo e l'andamento del prezzo. I risultati ottenuti mostrano la mancanza di un legame significativo tra le due variabili; il coefficiente di correlazione è, infatti, pari solo a 0,05.

Si è quindi analizzato il legame tra decisione di acquisto/vendita dell'agente BPCTAgentB e l'indice di previsione della rete neurale al fine di valutare il grado di influenza della previsione sulla dinamica comportamentale dell'agente. In questo caso i risultati ottenuti correlando la decisione di acquisto/vendita in *t* con la previsione per il giorno *t+10* mostrano una leggera correlazione (0,1266). L'agente cognitivo, nella determinazione della propria posizione speculativa sembra tener conto delle indicazioni sull'evoluzione futura del prezzo provenienti dalla rete neurale. È stato, infine, indagato il legame tra la strategia dell'agente e la sua dotazione; il coefficiente di determinazione R^2 della regressione: acquisto/vendita(t) = azioni(t-1) + liquidità(t-1) + previsioni(t+10), risulta, tuttavia, poco significativo.

In definitiva, risulta piuttosto difficile 'modellizzare' il comportamento dell'agente BPCTAgentB; lo sviluppo dei cross target unito all'obiettivo esterno di incrementare la ricchezza valutata sul prezzo previsto determina, infatti, l'emergere di una complessa dinamica comportamentale non riconducibile a fattori ben individuabili ed isolabili.

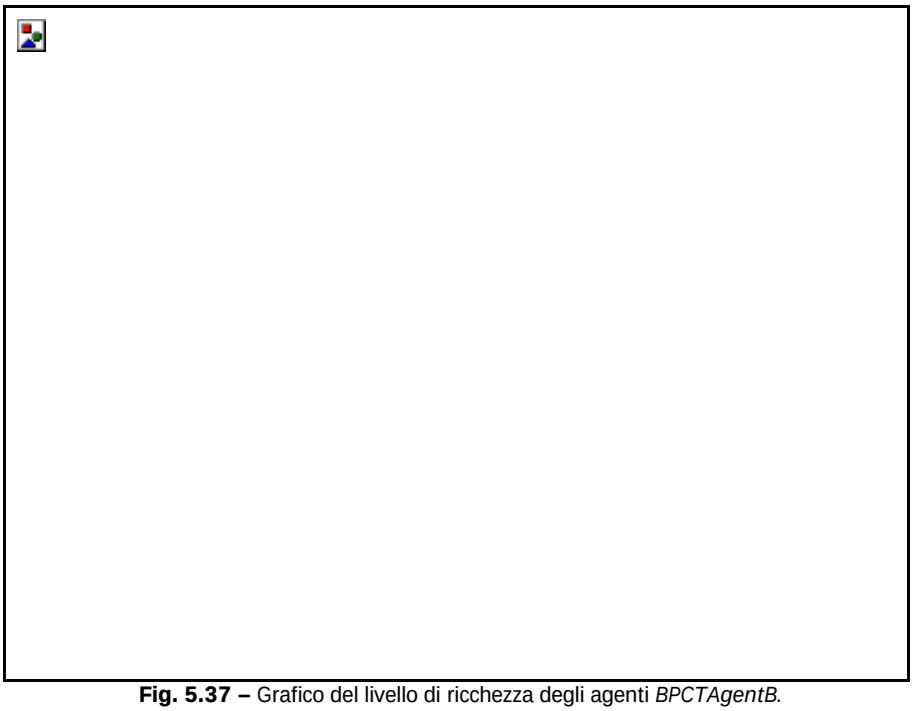


Fig. 5.37 – Grafico del livello di ricchezza degli agenti *BPCTAgentB*.

Al fine di analizzare le conseguenze di un ulteriore incremento del parametro che misura l'intensità dell'obiettivo esterno per l'agente BPCTAgentB, è stata condotta un'ulteriore simulazione con la stessa configurazione del modello, ma con *agentBEO_EPDelta* pari a dieci.

In questo caso, il mercato evidenzia una dinamica simile a quella del primo esperimento.

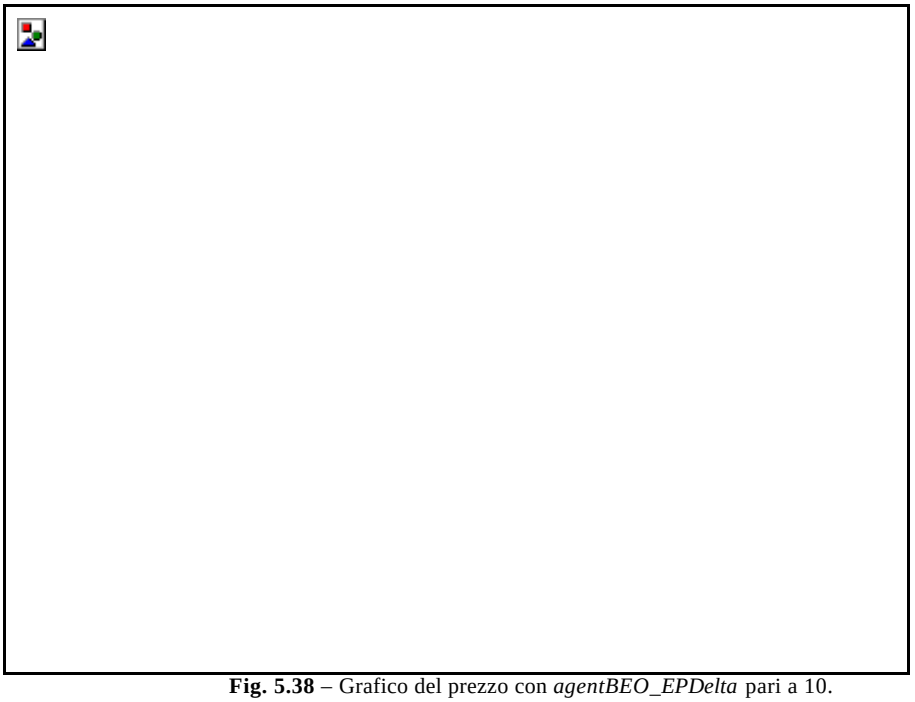


Fig. 5.38 – Grafico del prezzo con *agentBEO_EPDelta* pari a 10.

Non si determinano bolle speculative rilevanti; il prezzo non supera mai il livello di tre volte il prezzo iniziale; la prevedibilità del mercato scende però decisamente - 49,87% - e il comportamento degli agenti BPCTAgentB - come è possibile notare dalle figure riportate di seguito - risulta decisamente controproducente.

Fig. 5.39 – Grafico del livello di ricchezza degli agenti *BPCTAgentB*.

[137] R è un ambiente di programmazione particolarmente utile per l'analisi dei dati e dei grafici; per una breve descrizione delle potenzialità di tale *package* si veda l'appendice al fondo del capitolo.

Fig. 5.40 – Grafico della ricchezza, della liquidità e della quantità di azioni di un agente *BPCTAgentB*.

Gli agenti mostrano, infatti, livelli medi di ricchezza nettamente negativi (addirittura il livello massimo della ricchezza per lunghi periodi si colloca su valori negativi). Il valore *agentBEO_EPDelta* pari a dieci risulta dunque eccessivo e produce rilevanti distorsioni nel meccanismo di comportamento dell'agente cognitivo[138].

5.6 Agenti 'autoregressivi'

In quest'ultimo paragrafo vengono introdotti agenti che operano sul mercato sulla base di un proprio modello di evoluzione futura del prezzo; l'idea è quella di introdurre nel mercato operatori il cui modello autoregressivo:

$p_t = a\ p_{t-1} + b\ p_{t-2}$;

sia basato su stime dei parametri *a* e *b* coerenti con una analisi della serie di tempo del prezzo generata in una simulazione effettuata precedentemente.

L'obiettivo, in sostanza, è quello di indagare le conseguenze dell'introduzione nel mercato di borsa di agenti che abbiano un vantaggio 'informativo' dato dalla 'conoscenza' della struttura sottostante l'evoluzione del prezzo[139].

A tale scopo, si è proceduto - sulla base dei dati generati dall'esperimento con 270 agenti random, 10 agenti stop loss, 15 agenti che seguono la previsione della rete neurale e 5 agenti cognitivi di tipo B con

[138] Al fine di determinare il valore 'ottimale' di *agentBEO_EPDelta*, sono stati condotti diversi esperimenti modificando anche la configurazione del modello; dai risultati ottenuti sembra emergere che il valore ottimale di tale parametro vari a seconda del tipo di comportamento del prezzo. Per andamenti del mercato relativamente stabili con bolle di entità ragionevole *agentBEO_EPDelta* pari a due è una buona soluzione; all'aumentare della dimensione delle bolle sembrano più adatti valori di *agentBEO_EPDelta* leggermente più elevati (5-6).

obiettivo esterno di aumentare la ricchezza valutata sul prezzo previsto (e valore *agentBEO_EPDelta* pari a 2) - al calcolo, mediante il programma R, dei coefficienti *a* e *b* del modello autoregressivo che meglio approssimano la serie di tempo del prezzo riportata in figura 5.32.

I valori ottenuti (a=1.03 e b=-0.05)[140] sono quindi stati utilizzati nella simulazione seguente come valori centrali dei coefficienti *a* e *b* dei 5 agenti 'autoregressivi' introdotti nel modello al posto di cinque agenti casuali.

Di seguito è riportato il grafico del prezzo che si ottiene facendo 'girare' il modello con 265 agenti random, 10 agenti con strategie di stop loss, 15 agenti che applicano le previsioni neurali, 5 agenti cognitivi e 5 agenti *AR2Agent* con coefficienti del modello autoregressivo compresi nei seguenti intervalli: $(0.98 \leq a \leq 1.08)$ e $(-0.1 \leq b \leq 0)$.

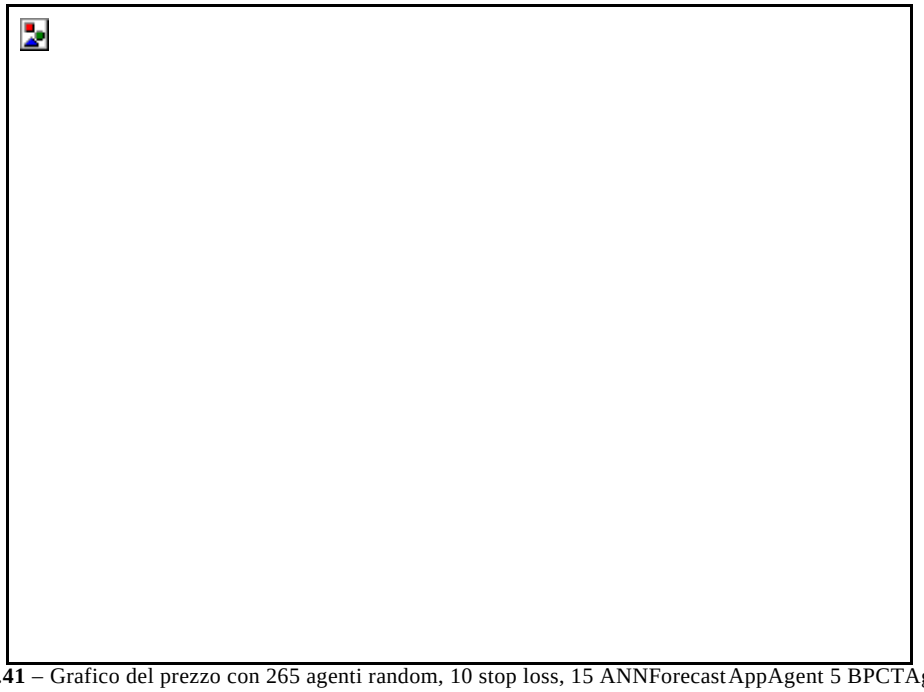


Fig. 5.41 – Grafico del prezzo con 265 agenti random, 10 stop loss, 15 ANNForecastAppAgent 5 BPCTAgentB con EO=2 e *agentBEO_EPDelta* pari a 2 e 5AR2Agent con $(0.98 \leq a \leq 1.08)$ e $(-0.1 \leq b \leq 0)$.

Il mercato che emerge risulta sostanzialmente stabile; non si verificano bolle speculative apprezzabili e il prezzo non supera mai il livello di 2.5 volte il valore iniziale. Il livello di prevedibilità risulta piuttosto elevato; la quota di previsioni corrette della rete neurale è pari, infatti, al 59,59%.

In figura 5.42 è riportata la dinamica della dotazione di un agente AR2Agent nell'arco dei 2000 giorni di simulazione; come è possibile notare la strategia di investimento adottata dall'agente autoregressivo è quella di vendere allo scoperto azioni e di incamerare liquidità; tale strategia determina un trend crescente del livello di ricchezza, ma espone l'agente a forti perdite nei periodi caratterizzati da andamenti crescenti del mercato.

---

[139] In altre parole si ipotizza che tali agenti conoscano il 'vero' modello di evoluzione del prezzo.

[140] Si noti come i coefficienti del modello autoregressivo siano piuttosto vicini a descrivere un processo di tipo 'random walk'.

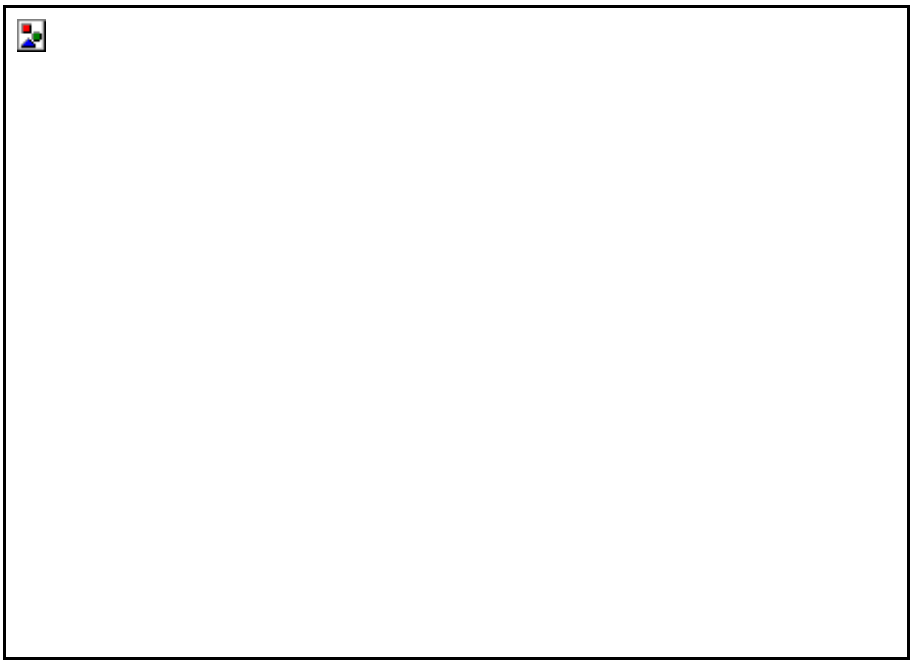


Fig. 5.42 – Grafico della dotazione di un agente AR2Agent.

Tale risultato trova conferma nell'analisi dei livelli di ricchezza delle diverse tipologie di agenti.

Come è possibile notare - figura 5.43 - gli agenti autoregressivi pur conoscendo il modello che approssima meglio l'evoluzione del prezzo nel tempo non sono esenti da perdite rilevanti.

Confrontando i livelli di ricchezza delle diverse tipologie di agenti emerge chiaramente come tali agenti – pur mostrando un trend positivo nei livelli di ricchezza - si caratterizzino per la più alta variabilità dei risultati, superiore anche a quella degli agenti casuali (a fronte infatti di livelli massimi di ricchezza pari a circa 400, le perdite raggiungono valori massimi pari a 1500).

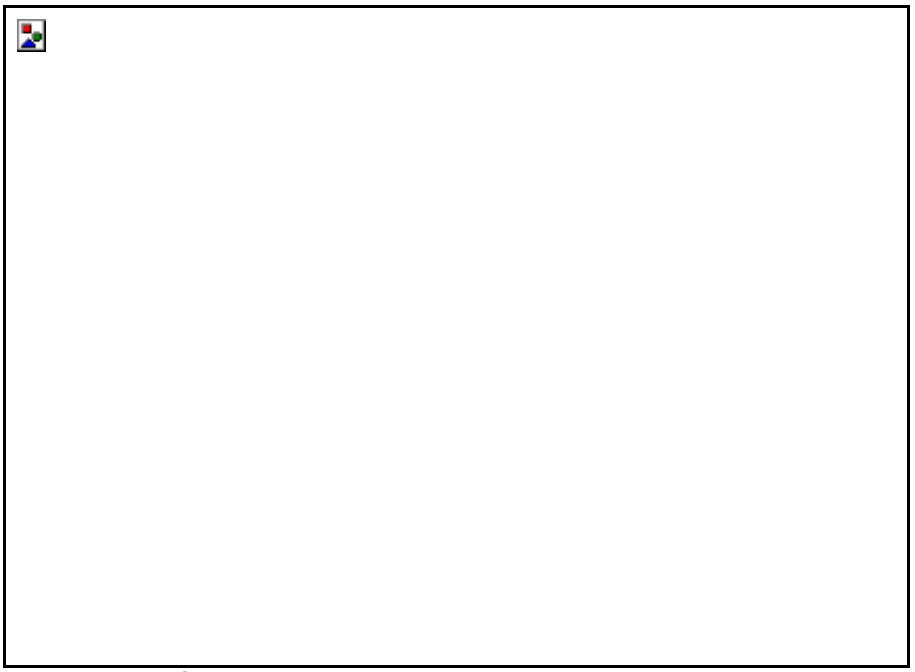


Fig. 5.43 – Grafico del livello di ricchezza degli agenti *AR2Agent*.

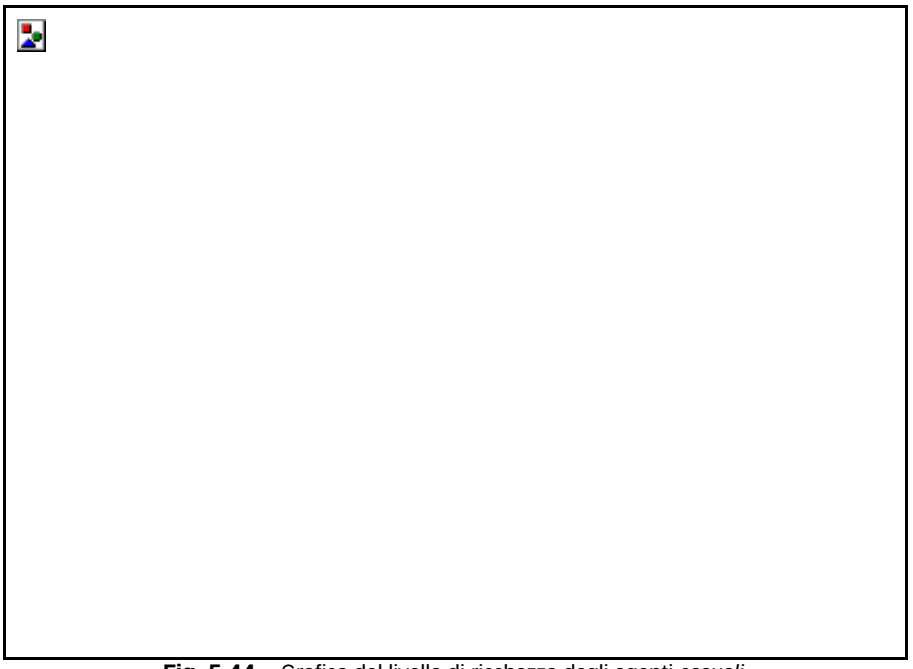


Fig. 5.44 – Grafico del livello di ricchezza degli agenti *casuali*.

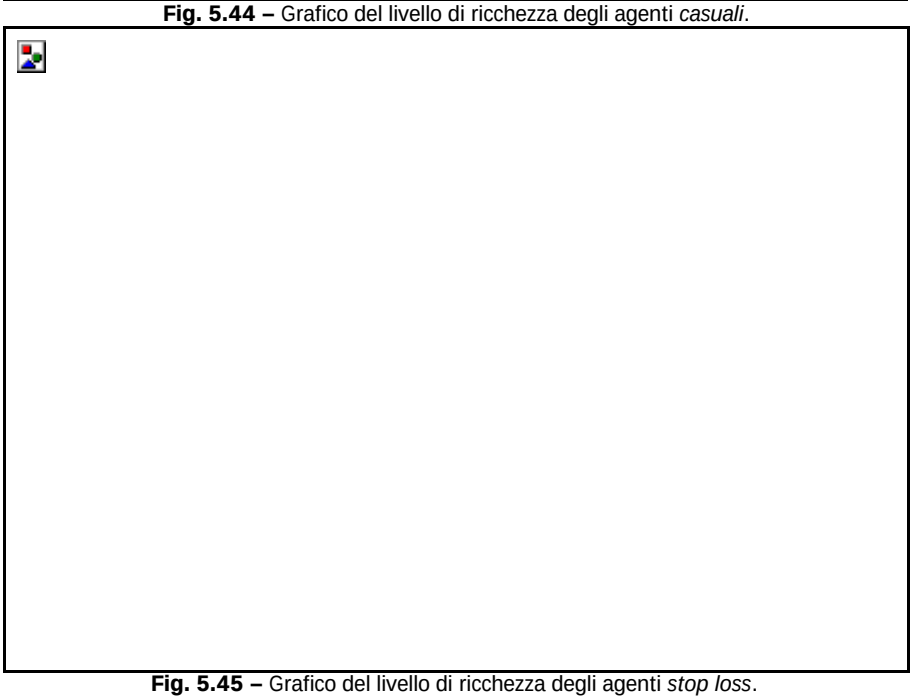


Fig. 5.45 – Grafico del livello di ricchezza degli agenti *stop loss*.

Il comportamento poco soddisfacente degli agenti AR2Agent ha due spiegazioni principali. La prima risiede nella considerazione che la loro introduzione altera la struttura del mercato e di conseguenza l'evoluzione del prezzo nel tempo; le proposte di negoziazione degli agenti AR2Agent determinano inevitabilmente una modificazione del contesto di borsa, che in questo modo non rispecchia più il modello sulla base del quale i coefficienti a e b degli agenti autoregressivi sono stati calcolati. Confrontando la

dinamica del prezzo prima dell'introduzione degli agenti autoregressivi (figura 5.32) con la dinamica riportata in figura 5.41 si nota, infatti, una sensibile differenza tra i due andamenti[141].

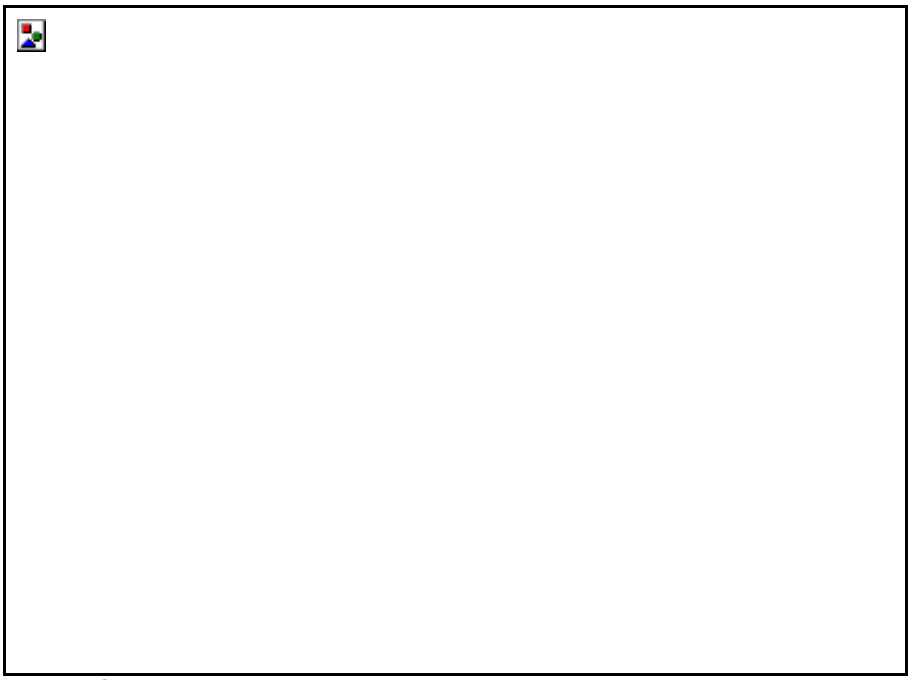


Fig. 5.46 – Grafico del livello di ricchezza degli agenti che applicano le previsioni neurali.

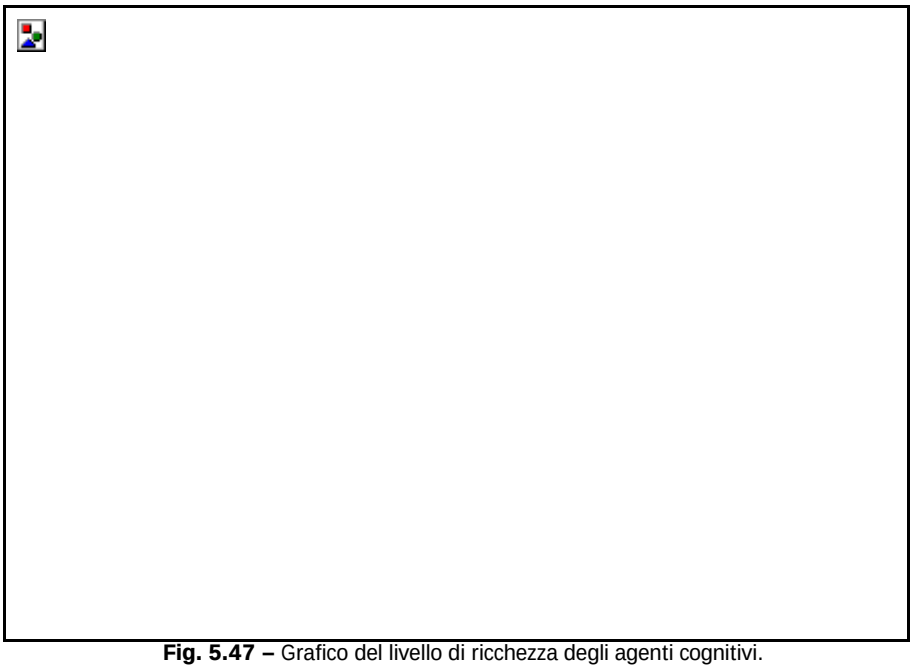


Fig. 5.47 – Grafico del livello di ricchezza degli agenti cognitivi.

La seconda spiegazione – la più importante – del comportamento insoddisfacente degli agenti autoregressivi è la mancanza, da parte di tali agenti, di un vero processo di adattamento all'ambiente; gli

[141] Una ulteriore conferma dell'alterazione della struttura del mercato si ottiene calcolando, mediante R, i nuovi coefficienti a e b del modello autoregressivo di secondo ordine che meglio descrivono la dinamica del mercato (a=1.08 e b =-0.12).

agenti AR2Agent non sono in grado di variare il proprio comportamento in modo da limitare le perdite durante le fasi più turbolente (si caratterizzano per un'eccessiva rigidità nella strategia di investimento).

L'importanza decisiva del processo di adattamento ad un mercato in continua evoluzione è confermata dall'osservazione – figura 5.47 - del comportamento degli agenti cognitivi, i quali anche in questo esperimento dimostrano la capacità di sviluppare autonomamente strategie di investimento intelligenti.

Come è possibile notare dalla figura 5.48, nella quale è riportata la dinamica della dotazione di un agente BPCTAgentB, l'agente cognitivo riesce ad ottenere un livello positivo di ricchezza modificando continuamente la propria strategia al variare delle condizioni del mercato.

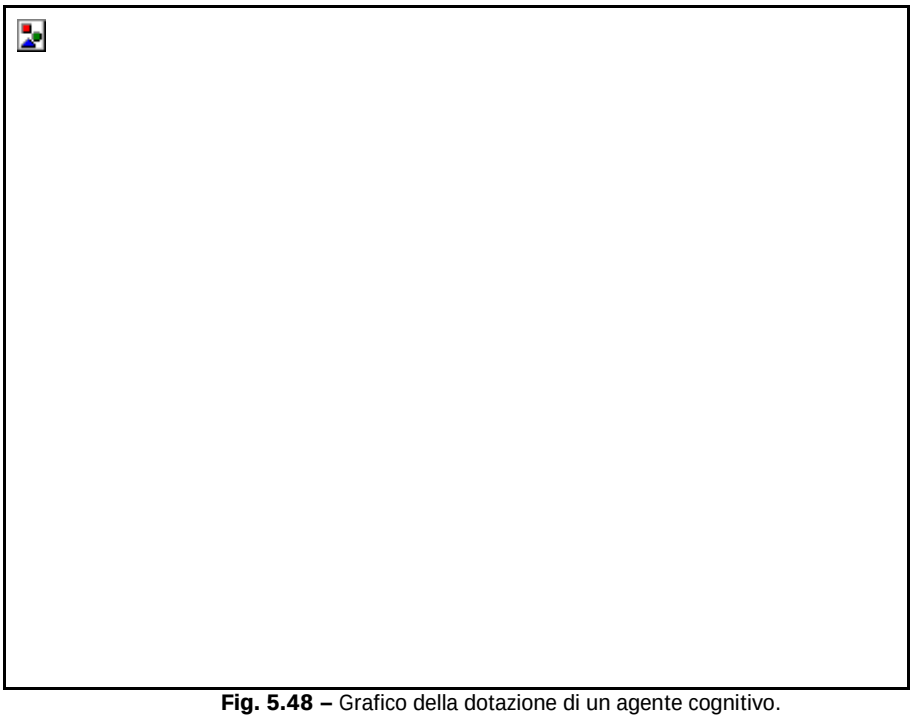


Fig. 5.48 – Grafico della dotazione di un agente cognitivo.

In particolare, l'agente cognitivo sembra adottare una strategia di investimento volta a 'realizzare' i guadagni; durante le 'mini-bolle' l'agente vende azioni e incamera liquidità; successivamente quando il prezzo si 'sgonfia' l'agente investe la liquidità accumulata per acquistare azioni a prezzi relativamente bassi che rivenderà in seguito quando il mercato attraverserà una nuova fase di rialzo. (Durante la prima fase di rialzo l'agente, vede diminuire sensibilmente il proprio livello di ricchezza perché - a fronte delle vendite allo scoperto effettuate - il livello del prezzo per un breve intervallo di tempo continua a crescere; successivamente, però, il mercato mostra una forte inversione di tendenza e il livello del prezzo scende sensibilmente premiando la strategia ribassista dell'agente cognitivo).

Il legame tra livello del prezzo e la quantità di azioni detenute dall'agente trova conferma nell'analisi della correlazione tra le due variabili (coefficiente di Bravais-Pearson pari a – 0.23)[142]; viene, inoltre, confermato il

[142] Se si restringe il campo di osservazione ai periodi di più elevata variabilità del prezzo la correlazione risulta rafforzata; durante la prima mini-bolla il coefficiente di correlazione tra la quantità di azioni detenuta dall'agente e il livello del prezzo è pari a –0,42; nella bolla più intensa invece, il coefficiente di correlazione è uguale a –0,55. Inoltre, in quest'ultimo caso, il coefficiente di determinazione R^2 della regressione:
azioni(t)= a + c0 prezzo(t-1) + c1 previsioni(t+10) risulta pari a 0,45.

legame tra le previsioni formulate dalla rete neurale e la dinamica comportamentale dell'agente cognitivo (il coefficiente di correlazione tra le decisione di acquisto/vendita dell'agente BPCTAgentB in *t* e le previsioni della rete neurale per il giorno *t+10* è pari, infatti, a 0.1211)[143].

Occorre, infine, sottolineare come la strategia di investimento dell'agente cognitivo - autonomamente sviluppata – abbia importanti effetti sulla dinamica del mercato; il comportamento 'anticiclico' dell'agente cognitivo tende, infatti, a 'smorzare' le tendenze a eccessivi rialzi o ribassi del mercato.

5.7 Conclusioni

Gli esperimenti effettuati e illustrati in questo capitolo dimostrano che l'obiettivo di creare mondi artificiali con agenti semplici - secondo il principio KISS ('*keep it simple, stupid*") di Axelrod - è perseguibile e offre risultati decisamente soddisfacenti; anche dall'interazione di agenti completamente casuali emergono, infatti, fenomeni complessi estremamente interessanti come le bolle e i crash.

Il modello SUM è in grado di ricreare fenomeni simili a quelli riscontrati nei mercati reali - ad esempio i processi di autorealizzazione delle aspettative - come conseguenza della semplice interazione tra agenti eterogenei[144]; il modello mostra, inoltre, l'influenza determinante delle strutture del mercato (il book) sulla dinamica del prezzo e pone l'attenzione sull'importanza del processo di adattamento da parte degli operatori; adattamento - che come si è visto - può essere efficacemente realizzato con l'introduzione di agenti 'cognitivi' sviluppati mediante l'utilizzo delle reti neurali e della metodologia dei cross target (tali strumenti hanno dimostrato di poter essere utilmente utilizzati per dotare gli agenti 'artificiali' del modello di una 'mente' in grado di elaborare comportamenti 'intelligenti').

Gli sviluppi futuri del modello SUM vanno dall'approfondimento del ruolo dell'ambiente sul mercato all'applicazione di una attenta analisi econometrica ai risultati generati; ulteriori interessantissimi sviluppi potranno, poi, venire dall'incremento del numero di titoli negoziati nel mercato artificiale di borsa (il che apre la possibilità di indagare gli effetti dell'interazione su un mercato di titoli derivati) e dall'unione del modello SUM con il filone delle tesi sperimentali in modo da far interagire gli agenti artificiali del modello con agenti 'umani'.

APPENDICE: R

R è un ambiente di programmazione - sviluppato come estensione del programma S-PLUS - particolarmente utile per l'analisi dei dati e dei grafici. Esso, infatti, dispone di:

- strumenti che facilitano la manipolazione delle informazioni;
- un insieme di operatori che gestiscono i calcoli con vettori e matrici;

[143] I valori dei coefficienti di correlazione risultano 'sfumati' poiché risentono delle forti componenti casuali e della complessa dinamica comportamentale frutto dello sviluppo dei CT.
[144] È così possibile superare i principali 'difetti' dei modelli classici: l'assenza di eterogeneità e di interazione tra agenti.

- un linguaggio di programmazione che comprende istruzioni condizionali, *loops*, funzioni ricorsive e strumenti di gestione degli input e degli output;
- una ampia varietà di tecniche statistiche per l'analisi dei dati e la visualizzazione di grafici.

L'utilizzo di R è gratuito in quanto coperto dalla licenza GNU (General Public License).

Di seguito sono elencate una serie di istruzioni particolarmente utili per l'analisi dei dati generati dalle applicazioni del modello SUM:

- library(ts): consente di disporre delle funzioni per la gestione delle serie di tempo;
- scan: consente di caricare i dati contenuti in file; ad esempio, l'istruzione:

  prezzo<-ts(scan(file="c:/Swarm-2.1.1/miaDirectory/Current Price"));

  carica nella variabile 'prezzo' (di tipo time series) i dati del file Current Price contenuto nella directory miaDirectory dalla quale è stato lanciato il modello SUM;
- ts.plot(prezzo,indice, gpars=list(col=c("blue","red"),lwd=c(1,1))): visualizza in un grafico le serie di tempo 'prezzo' (colore blu) e 'indice' (colore rosso);
- cor(prezzo, indice): calcola la correlazione tra 'prezzo' e 'indice';
- lm: permette di calcolare la regressione lineare tra variabili; ad esempio, l'istruzione:

  ris<- lm(prezzo ~ indice)

  calcola la regressione lineare tra 'prezzo' (la variabile dipendente) e 'indice' (la variabile indipendente);

  -summary(x): è la funzione che permette di visualizzare i risultati delle elaborazioni contenute nell'oggetto x; ad esempio, l'istruzione:

  summary(ris) visualizza i risultati della regressione tra 'prezzo' e 'indice'
- arima0: consente di eseguire un'analisi di tipo arima; l'istruzione

  arima0(prezzo,order=c(2,0,0))

  calcola i coefficienti dell'equazione autoregressiva di secondo ordine che meglio descrivono la serie di tempo 'prezzo'.

BIBLIOGRAFIA

- AXELROD R. (1997), *Advancing the Art of Simulation in the Social Sciences* in CONTE R., HEGSELMANN R. & TERNA P. (eds*)* Simulating Social Phenomena, Berlin: Springer.

- TERNA P. (2001), *Cognitive agents behavings in a simple stock market structure*, in corso di pubblicazione.