UNIVERSITA' DEGLI STUDI DI TORINO
Facoltà di Economia
Corso di laurea in economia e commercio

TESI DI LAUREA

ALGORITMI GENETICI PER L'ECONOMIA

Relatore: prof. **PIETRO TERNA**
Correlatore: prof. **SERGIO MARGARITA**

CANDIDATO

GIANLUIGI FERRARIS

ANNO ACCADEMICO 1998/99

Indice

Introduzione

La computazione automatica, normalmente utilizzata nell'esame dei risultati delle sperimentazioni, offre, oggi, opportunità d'impiego più ampie, legate alle tecniche di simulazione. Sfruttando questi paradigmi, è possibile costruire oggetti virtuali, capaci di riprodurre le caratteristiche delle entità studiate, utili per l'esame di molti fenomeni tipici del mondo reale.

Gli aggregati virtuali consentono indagini più semplici ed approfondite di quelli reali: tempo e casualità sono determinati dallo sperimentatore, le caratteristiche ambientali possono essere modificate, la rilevazione dei dati è più precisa e l'esperimento può essere ripetuto a piacere. La possibilità di modificare la velocità di scorrimento del tempo contribuisce a ridurre la durata della ricerca che, nella realtà, può richiedere la ripetizione periodica delle osservazioni, impegnando tempi considerevolmente lunghi; il controllo sulla manifestazione degli eventi casuali consente di produrre dati immediatamente utilizzabili, senza bisogno di procedere ad elaborazioni preventive necessarie per eliminare le componenti di disturbo quando si utilizzassero dati reali. Nelle scienze sociali, l'osservazione dei fenomeni viene spesso effettuata tramite interviste o valutazione di dati forniti dagli istituti di statistica: in questi casi occorre determinare preventivamente quali dati rilevare e, successivamente, considerare margini di errore derivanti da cattiva comprensione dei formulari o da risposte insincere. Impiegando agenti virtuali, invece, tutti i dati possono essere rilevati facilmente, si evitano errori e non occorre determinare a priori quali grandezze osservare. Casi particolari, o estremi, si manifestano difficilmente in ambiti reali, la ripetizione della rilevazione è onerosa e le variazioni delle condizioni al contorno non sempre rispondono alle esigenze del ricercatore; questi problemi non si hanno con le simulazioni. L'indagine su ambiti virtuali risulta, perciò: meno costosa, più agevole, maggiormente adattabile agli scopi della ricerca e capace di fornire risultati fortemente innovativi.

Una buona simulazione deve essere: realistica, per fornire dati attendibili, e basata su entità fortemente autonome, per consentire l'emergere di comportamenti nuovi o imprevisti. Risultati interessanti sono ottenibili con i modelli basati su agenti, cioè entità dotate di un elevato grado di indipendenza e capaci di adattare il loro comportamento alle caratteristiche dell'ambiente in cui operano. Effetti aggregati, di una certa importanza, risultano emergere già con l'impiego di oggetti capaci di reagire, in modo prestabilito, a stimoli semplici; il comportamento di simili agenti risulta, però, ancora influenzato dalle scelte di chi redige il programma. Quando la determinazione delle regole di comportamento avvenga in base a tecniche di intelligenza artificiale, invece, l'azione di ogni agente risulta essere conseguenza del grado di apprendimento del medesimo, permettendo di garantire l'autonomia sufficiente a promuovere lo sviluppo di fenomeni innovativi. Le tecniche di intelligenza artificiale, più conosciute ed adatte allo scopo, fanno capo ai due paradigmi delle reti neurali artificiali e dei metodi evolutivi: algoritmi genetici, *classifier system* e *genetic programming*.

Il presente lavoro tratta dell'impiego dei metodi evolutivi, con particolare riferimento agli algoritmi genetici ed ai *classifier system*. Queste tecniche si basano sull'imitazione del processo di evoluzione naturale: le regole sono assoggettate a procedimenti di riproduzione ed estinzione, atti a garantire la sopravvivenza di quelle che permettano di ottenere i migliori risultati. Si procede valutando ciascuna regola per stabilire quali, fra esse, dovranno contribuire alla formazione di nuove istanze, cioè riprodursi, e quali saranno sostituite dalle nuove regole. La ripetizione ciclica del procedimento consente un graduale perfezionamento del patrimonio normativo. Mutazioni delle condizioni ambientali, comportando variazioni del "valore" di ciascuna regola, vengono continuamente recepite dal sistema con evoluzioni delle regole stesse.

La codificazione dei programmi, per il governo dei metodi descritti, risulta piuttosto complessa ed onerosa; si è ritenuta utile la redazione di "pacchetti" *software* in grado di svolgere la computazione, relativamente a più insiemi di regole, in modo simultaneo ed indipendente. Tali presidi informatici, basati sull'utilizzo del prodotto *Swarm*, valido e diffuso *software* per l'esecuzione di simulazioni, e strutturati secondo lo schema ERA (Terna 1998), possono essere facilmente utilizzati in un gran numero di modelli. L'adozione di parti comuni e l'adesione a precisi *standard* strutturali aumentano la comprensibilità degli esperimenti e favoriscono la collaborazione fra i ricercatori, permettendo un più agevole scambio di esperienze.

1 – Calcolo automatico e studi economici

In questo capitolo si procede all'illustrazione delle possibilità offerte dall'utilizzo degli elaboratori elettronici, sia per l'attività scientifica in generale, sia per gli studi economici.

Il primo paragrafo tratta dei vantaggi del metodo da un punto di vista generale: vi vengono svolte osservazioni relative alle possibilità di delegare compiti, ripetitivi ed onerosi, all'elaboratore e si accenna alle opportunità offerte dai paradigmi dell'intelligenza artificiale. Si tratta, inoltre, dei vantaggi ottenibili dall'utilizzazione della rete *Internet*, particolarmente nella collaborazione fra studiosi; a commento del potenziale di simili cooperazioni, viene svolto un breve accenno all'estensione del patrimonio di *software* gratuito, disponibile in rete, cosiddetto *freeware*.

Segue la disamina dei benefici, derivanti dall'utilizzazione degli strumenti automatici di computazione, nell'ambito degli studi economici.

Il terzo paragrafo è dedicato all'illustrazione delle tecniche basate su modelli di simulazione. Vi vengono discusse le potenzialità del metodo: esso permette di ricreare, agevolmente, le complesse situazioni scaturenti, nella realtà, dall'interazione di numerosi sottosistemi e componenti. Si esaminano le ragioni che ne consigliano l'utilizzazione: i vantaggi derivanti dalle possibilità di eseguire verificazioni in diversi ambiti simulati, l'opportunità di studiare l'emergere di comportamenti complessi in aggregati di individui semplici, le maggiori occasioni di collaborazione legate alle possibilità di pubblicare, oltre ai risultati, anche il modello da cui sono stati ottenuti.

L'esecuzione di simulazioni con il *computer* richiede la predisposizione di ambienti artificiali complessi; occorre gestire aspetti che, nella realtà, sono impliciti, ad esempio lo scorrere del tempo. Oltre alla codificazione di programmi per emulare il comportamento dei diversi operatori, occorre predisporre gli strumenti *software* per la gestione della memoria dell'elaboratore, la ricezione dei parametri, la rilevazione dei dati ed altri. Nel quarto e quinto paragrafo, dopo breve disamina delle problematiche menzionate, è presentato il pacchetto *Swarm*: un prodotto per la costruzione di modelli basati su agenti, dotato di caratteristiche, di versatilità e diffusione, che ne hanno consigliato l'adozione per i modelli usati nel presente lavoro, e per molti altri.

Il capitolo si chiude con alcuni accenni alle esigenze di chiarezza e comprensibilità dei lavori, contenute nell'ultimo paragrafo, in cui viene presentata la proposta di metodo, riassunta dallo schema *Environment-Rule-Agent* (ERA), riguardante l'organizzazione e la struttura dei modelli.

1.1 – Elaborazioni automatiche e ricerca

I progressi compiuti nella realizzazione di macchine per la computazione permettono, oggi, di utilizzare strumenti capaci di svolgere rapidamente grosse moli di lavoro, caratterizzati da un'elevata affidabilità e da bassi costi. Si è avuta, particolarmente negli ultimi anni, una forte crescita delle capacità di elaborazione, accompagnata da una continua riduzione dei costi; il risultato dei due fenomeni è stata la capillare divulgazione dello strumento computazionale.

Parallelamente a questa crescita di potenziale è andata maturando la capacità di sfruttarlo: le prime esperienze di rappresentazione della conoscenza, basate sul tentativo di riprodurre il funzionamento del cervello umano, sono mutate verso atteggiamenti e, conseguentemente, obiettivi, meno ambiziosi ma più costruttivi. La ricerca della plausibilità nella ricostruzione del ragionamento umano ha lasciato il posto alla serena coscienza di essere giunti alla produzione di un sofisticato meccanismo, ben lontano come struttura e funzionamento dal cervello umano, ma capace di svolgere, utilmente, alcune attività, fino ad ora di esclusiva pertinenza dello stesso. La macchina continua ad essere uno strumento, più raffinato e potente, di cui è possibile valersi per tutti quei compiti che

risultino, per loro natura, inadatti o sgraditi ad un intelletto umano, ansioso, nella sua corsa evolutiva, di affrancarsi dalla ripetizione del "conosciuto" per poter indagare il "nuovo".

L'utilizzo del *computer* si concretizza, oggi, nella delegazione delle parti ripetitive e, solitamente, meno qualificanti, di molte attività nel campo della ricerca e la sfida che, continuamente, si pone sta nell'individuare ambiti sempre più vasti per questo impiego. Il progredire delle capacità del mezzo inanimato opera in sinergia con l'evoluzione delle conoscenze umane: le prime rendono le macchine adatte ai nuovi compiti che la seconda porta a delegare loro. Dopo una fase in cui si attribuiva alla macchina il puro calcolo, si vive, da diversi anni, quella in cui, programmi notevolmente complessi, permettono di utilizzare il *computer* nell'ambito del computo simbolico e della risoluzione di sistemi di equazioni. Valendosi di tecnologie maggiormente innovative, è possibile utilizzare il *computer* nella soluzione di problemi caratterizzati dalla presenza di vincoli e condizionamenti non conosciuti a priori, tramite la produzione automatica di regole operative. Ciò preconizza l'esistenza di ulteriori campi di applicazione che, ancora una volta, permetterebbero di affrancare l'elemento umano da compiti per i quali l'utilizzo di una risorsa preziosa come la sua mente risulterebbe "non economico".

L'ausilio elettronico può, dunque, porsi quale strumento per potenziare le capacità umane in campo scientifico, non già perché realmente più "capace" ma, solo, in quanto, al pari del braccio meccanico di un escavatore in campo fisico, specializzato e dimensionato al raggiungimento di un preciso fine dalla più creativa mente umana. L'analogia con la meccanica risulta confortante anche per quanto concerne i problemi di plausibilità: molte macchine pur risultando buoni surrogati dell'uomo, per specifici lavori, non sono affatto simili a lui nell'aspetto o nel funzionamento, pure, così come *computer* e programmi dissimili dalla mente umana, possono produrre eccellenti risultati.

Il lavoro di ricerca è, per sua natura, attività di gruppi: i suoi risultati vengono qualificati ed arricchiti dagli apporti ottenuti nel confronto con quelli di altri studiosi, sia esperti della materia, sia impegnati in altre discipline. La possibilità di condividere dati ed esperienze è esigenza fortemente sentita e, da sempre, componente importante dell'evoluzione della conoscenza e del pensiero scientifico. Il continuo conoscere e rapportarsi di tutti con tutti costituisce il principale stimolo alla crescita intellettuale e contribuisce, non di rado, allo scaturire delle migliori idee. L'elettronica ha permesso l'enorme potenziamento dei mezzi di comunicazione che tutti conosciamo, rendendo possibile la condivisione, quasi istantanea, di informazioni fra gruppi di lavoro sparsi per tutto il mondo: la rete *Internet*, molto utilizzata in ambito universitario e scientifico, collega quotidianamente studiosi siti in località remote e, soprattutto, permette di mantenere, in linea, un magazzino di informazioni dalle proporzioni mai conosciute in precedenza.

Con una simulazione al *computer* la possibilità di condividere, in modo tanto efficiente, i risultati delle proprie deduzioni si estende all'oggetto stesso dell'osservazione; è possibile trasmettere l'intero modello, ossia l'insieme di programmi che producono la simulazione del sistema reale, oggetto di studio. In questo modo altri scienziati possono rieseguire la simulazione, apportando, eventualmente, le variazioni suggerite dalla loro esperienza personale. Nell'ambito di molte discipline, ciò equivale alla possibilità di condividere l'intero esperimento, non solamente le deduzioni da esso tratte. Quando l'attività sperimentale dovesse estendersi ad un elevato numero di casi, o potesse basarsi su molteplici metodologie, più esperti potrebbero collaborare alla sua conduzione, apportando, ciascuno, il contributo relativo alle parti in cui più forti fossero le sue conoscenze. Si apre, perciò, una opportunità di collaborazione attiva, già a partire dalla fase di predisposizione del modello, cioè dell'esperimento, normalmente superiore a quella offerta da metodologie tradizionali.

La possibilità di ricondurre un'esperienza, senza doverne riprodurre l'apparato sperimentale, implica, inoltre, l'affrancamento da tutte le possibili imprecisioni che potrebbero affliggere la riproduzione del medesimo e dal dispendio di tempo ad essa connesso; il confronto risultante da simili interazioni si presenta maggiormente qualificato e fortemente agevolato, con ovvi benefici in ordine ai risultati da esso ottenibili. Ogni variazione apportata all'apparato può, inoltre, essere resa disponibile anche agli altri ricercatori arricchendo, conseguentemente, le comunicazioni di ritorno.

Come esempio dell'enorme potenzialità di simili collaborazioni valgano le cifre relative al cosiddetto "*freeware*", cioè il *software* gratuitamente disponibile sulla rete, immessovi da studiosi

operanti in tutto il mondo, con lo scopo di permetterne l'utilizzazione e l'arricchimento. Recenti studi (Meo 1998), relativi ad un programma di ricerca del Politecnico di Torino, riportano una stima del valore di questo *software* superiore ai cento milioni di dollari, contro un valore presunto del mercato mondiale dell'informatica, per il 1996, di 603 milioni di dollari. Tutto questo patrimonio, composto da milioni di istruzioni scritte dalle persone più disparate, si è formato nell'arco degli ultimi dieci anni dando luogo, fra le altre realizzazioni, al sistema operativo *LINUX*, clone di *UNIX*, che, di recente, la stessa IBM ha deciso di fornire, su richiesta del cliente, in aggiunta o alternativa al consueto *Microsoft Windows*. La prima versione del sistema, scritta fra il 1990 e il 1991 da uno studente dell'Università di Helsinki che non poteva permettersi l'acquisto della versione per *personal computer* di *UNIX*, era in grado di gestire poche funzioni essenziali; l'apporto di appassionati e studiosi, delle più disparate nazionalità, ha permesso di conseguire il risultato odierno.

In conclusione è possibile rilevare come il calcolo automatico possa costituire un valido presidio per le attività scientifiche in svariati campi, non ultimo, quello economico dove può essere proficuamente sfruttato nella gestione di situazioni complesse, quali quelle descritte dalla teoria dei giochi o quelle scaturenti dai tentativi di superare gli stringenti assunti delle impostazioni classiche. L'ottenimento di queste utilità passa per il tramite di una corretta impostazione del rapporto fra elemento umano e elettronico ben espressa in Hajek (1997):

> (…) in informatica noi abbiamo ancora un mondo dominato da istruzione e programmazione, mentre ormai è evidente che la macchina può diventare uno strumento duttile nelle mani dell'utente. Sembra pertanto particolarmente importante cercare di introdurre in ambiente informatico l'idea che la macchina può diventare strumento dinamico implicato nel processo.

1.2 - Applicazione del calcolo automatico agli studi economici

I vantaggi della computazione automatica, elencati nel precedente paragrafo, possono riguardare anche l'indagine economica. In questo campo, l'importanza del calcolo automatico è generalmente riconosciuta in riferimento alle analisi econometriche; più di recente, però, si è andata affermando la sua utilizzazione diretta nello studio dei fenomeni economici, come dimostrano i buoni risultati ottenuti da un nutrito numero di lavori.

Un gran numero di modelli economici può essere trattato con metodi di computazione automatica; la tecnica risulta particolarmente utile quando applicata a modelli fortemente complessi che, con procedimenti tradizionali, risulterebbero difficilmente computabili. Ci si riferisce, in particolare, ai casi di imperfetta informazione, alla soluzione dei modelli basati su aspettative razionali ed ai giochi basati sul concetto di equilibrio di Nash.

I maggiori vantaggi dei metodi computazionali risaltano dal confronto con i tradizionali sistemi basati sulla deduzione. Questi ultimi introducono, di sovente, convenzioni di base volte a semplificare il modello, onde renderlo trattabile; l'utilizzazione di questi modelli per l'interpretazione di dati empirici può portare all'ottenimento di risultati scarsamente realistici. La maggior potenza computazionale degli elaboratori elettronici permette di evitare queste semplificazioni, rendendo più realistica la dimostrazione.

Le tradizionali ricerche empiriche tendono a verificare l'attinenza della teoria ai dati della situazione economia corrente; con i metodi computazionali, invece, è possibile eseguire una verificazione più ampia, simulando tutte le situazioni, determinate dalle combinazioni dei valori plausibili per i diversi parametri. In questo modo l'analisi del problema risulta più completa ed è possibile stabilire il grado di importanza della relazione investigata, in funzione della frequenza di manifestazione dei suoi effetti in un gran numero di situazioni diverse. L'estensione dell'indagine può, infine, portare alla scoperta di effetti particolari, che potrebbero non emergere nell'analisi dei soli dati empirici.

Nella ricerca di nuove connessioni, i metodi basati sulla computazione permettono di effettuare una esplorazione più approfondita del contesto: solitamente, i metodi deduttivi procedono per il tramite di una catena di implicazioni, basate sull'intuizione del ricercatore e sulle possibilità di dimostrazione; la computazione automatica del modello consente, invece, di analizzare un più esteso numero di implicazioni, conferendo maggior robustezza alle teorie successivamente formulate.

La computazione può costituire un potente completamento dell'investigazione deduttiva tradizionale, come si legge in Judd (1996):

> *The first way in wich computation and theory interact is as strong complements. The goal of theoretical analysis in economics is to describe the nature of equilibrium. A complete characterization is often impossible. However, theoretical analysis can often provide a partial characterization, wich can then be used by computational methods to produce efficient numerical approaches, wich then produce economically substantive results.*

1.3 - I modelli di simulazione basati su agenti

Nell'ambito degli studi economici, (Margarita 1992) le strutture stesse degli strumenti di indagine hanno, recentemente, subito una modificazione: dai "modelli di struttura" si è passati ai "modelli di comportamento". I primi sono, normalmente, basati su strumenti statistici ed econometrici, considerano l'intero sistema economico e si valgono di tecniche di ottimizzazione statica. I modelli comportamentali considerano, invece, i singoli agenti per studiarne, appunto, il comportamento ed indagare gli effetti della loro interazione, valendosi di impostazioni maggiormente dinamiche, sovente coinvolgenti il concetto di razionalità, quale quella tipica della "teoria dei giochi".

I modelli di struttura sono, frequentemente, basati sull'azione di un unico individuo, il cui comportamento rappresenta quello dei vari operatori economici; tale individuo è, solitamente, caratterizzato da: perfetta razionalità ed illimitata capacità di ricercare, ricevere ed elaborare le informazioni. La totale uniformità di risposte e l'associazione rigida fra reazioni e stimoli ambientali, che discende da questa impostazione, rende difficoltoso dare il giusto risalto alle distorsioni che possono condizionare l'azione, effettiva, di singoli operatori: imperfetta informazione, difficoltà di scelta, scarsità del tempo per compierla, incapacità o limitazione nel trattare i dati a disposizione sono tutti elementi che possono influenzare le decisioni dei vari individui. Il tentativo di recepire queste peculiarità, all'interno di modelli matematici, normalmente, formalizzati in sistemi di equazioni, comporta una crescita esponenziale di difficoltà, essenzialmente ricollegabile alla forte complicazione del modello, indotta dal dover recepire ogni turbativa come una variabile del sistema; né il superamento di queste difficoltà risulterebbe esaustivo concretizzandosi, comunque, l'introduzione delle nuove variabili nel modello nella codifica di elementi forniti dall'esterno: parametri che potrebbero, al meglio, essere interpretati quali comportamenti prestabiliti di quote, la cui estensione è decisa a priori, della popolazione osservata.

Nei modelli comportamentali divengono oggetto d'indagine le risposte che l'individuo fornisce a fronte di stimoli ambientali e, soprattutto, i meccanismi attraverso i quali l'interazione dell'operato dei singoli può sfociare in conseguenze aggregate complesse, quali il sorgere e trasformarsi di istituzioni o la convergenza dei comportamenti verso matrici comuni. Questa impostazione concretizza un enorme cambiamento nell'ambito delle scienze economiche: Tesfatsion (1998) nota come ciò implichi l'adozione di un metodo sperimentale, che porti a concepire il fenomeno economico come riproducibile in un qualche tipo di laboratorio, distaccandosi, così, dalla concezione classica che considerava la storia l'unica, possibile, fonte di dati per le scienze sociali. L'economia sperimentale, per conseguenza, non può non proporre impostazioni della ricerca del tutto diverse, superando l'orientamento basato principalmente sull’econometria; né ciò dovrebbe considerarsi un puro cambiamento nelle tecniche di verificazione empirica degli asserti, potendo suffragare l'idea di estendere metodi sperimentali alle scienze sociali che condurrebbe verso una unificazione, sotto il profilo epistemologico, delle medesime con le scienze naturali.

Importanti lavori, in questa direzione, riguardano lo sviluppo di mondi artificiali dove strutture sociali e comportamenti di gruppo emergono dall'interazione degli individui, rappresentati da programmi, operanti essi stessi nell'ambiente artificiale, in base a regole autonomamente sviluppate. Promuovere lo sviluppo, spontaneo, di istituzioni e strutture, in un mondo artificiale, ha lo scopo di indagare la formazione delle medesime, onde tentare di scoprire quali meccanismi, a livello individuale, siano bastevoli per generare comportamenti collettivi di un qualche interesse. In campo economico, l'utilizzazione di queste metodologie è adottata dalla "*Agent based computational economics*" (ACE), di cui si ha una definizione in Tesfatsion (1997):

> *Agent-based computational economics (ACE) is roughly characterized as the computational study of economies modelled as evolving decentralized systems of autonomous interacting agents. A central concern of ACE reasearchers is to understand the apparently spontaneous appearence of global regularities in economic processes, such as the unplanned coordination of trade in decentralized market economies that economists associate with Adam Smith's invisible hand. The challenge is to explain these global regularities from the bottom up, in the sense that the regularities arise from the local interactions of autonomous agents channeled through actual or potential economic institutions rather than through fictitious top-down coordinating mechanisms such as a single representative consumer.*

La ACE collega, senza ricorrere a costruzioni teoriche artificiose, gli ambiti micro e macroeconomici indagando le risultanze dell'interazione dei singoli individui, capace di generare scenari complessi. L'attributo va inteso quale paradigma antiriduzionista, da cui evincere: l'impossibilità di ricondurre la spiegazione delle proprietà di un sistema alla conoscenza delle caratteristiche dei suoi componenti elementari e l'importanza, oltre che della complessità strutturale, della dinamica del sistema stesso, giustificando l'interesse per le leggi che ne descrivono l'evoluzione nel tempo. Un concetto, quindi, da non intendersi quale ostacolo alla comprensione, bensì come caratteristica dell'aggregato di cui occorre tenere conto quando si voglia giungere ad una effettiva cognizione del suo funzionamento. Ciascun agente possiede una marcata autonomia; ciò implica la sua capacità di prendere decisioni, in ordine alle proprie azioni, e di adattare il proprio comportamento alla percezione, che egli sperimenta, delle condizioni ambientali. Diviene centrale il dibattito sulla razionalità che i lavori di Simon (1981) sulla razionalità limitata hanno arricchito di una effettiva alternativa alle impostazioni di Von Neumann e Morgestern: in molti casi la ricerca euristica offre strumenti preferibili a quelli della ricerca operativa. I metodi euristici richiedono, tendenzialmente: minori condizioni al contorno, minori capacità specifiche e sono in grado di superare le difficoltà, connesse alle degenerazioni combinatorie del numero di situazioni, tipiche degli ambienti complessi.

L'accento posto sull'individuo in questo tipo di modelli porta Epstein e Axtell (1996) a parlare di "individualismo metodologico" , specialmente in riferimento all'assenza, in essi, di istituzioni predefinite in grado di orientarne l'evoluzione. Al proposito in Terna (1998) sono riportati alcuni interessanti chiarimenti, nel corso dei quali, viene fornita la definizione di questi modelli ed introdotto il problema del grado di complessità del singolo agente. Partendo da una definizione da Zamagni (1987):

> La posizione secondo cui tutte le proposizioni circa i gruppi sono riconducibili a proposizioni circa il comportamento dei singoli facenti parte di quei gruppi e le loro interazioni, è oggi nota come individualismo metodologico, un'espressione coniata da Schumpeter (…)

si osserva come questa posizione venga criticata sia da coloro che negano l'efficacia, nello spiegare il funzionamento di un sistema sociale, dello studio dei singoli individui, sia dagli studiosi portati a rifiutare la scelta dell'agente "rappresentativo", essenzialmente riduzionista. Su questa discussione si deve, inoltre, innestare anche quella relativa all'uso di agenti dotati di strutture semplici, con poche regole, in opposizione alla scelta di agenti molto complessi, dotati di regole fortemente strutturate. La soluzione proposta, per la diatriba, va nella direzione di ammettere ruoli ex ante di istituzioni e strutture preesistenti, sfruttando l'eterogeneità degli agenti, che diviene essenziale, in Terna (1998):

> (…) per sperimentare la non linearità degli effetti aggregati dei loro comportamenti (cioè: il tutto che non necessariamente corrisponde alla somma delle parti) in presenza di

interazione, con l'emergenza della complessità: è in ciò che consiste l'originalità della sperimentazione artificiale con modelli di simulazione fondati su agenti.

La conclusione è che non vi sia coincidenza fra individualismo metodologico e modelli basati su agenti. Questi ultimi sarebbero deputati allo studio della generazione di istituzioni e comportamenti, non previsti, o precedentemente codificati, in strutture, anche organizzate su diversi livelli, quale risultato della interazione, ammessa a prescindere dalla comune appartenenza ad uno stesso livello, fra le varie entità componenti. La natura atomistica dell'agente non risulterebbe strettamente vincolante: egli potrebbe benissimo risultare composto di parti, come nel caso dell'impresa composta di più elementi, ravvisandosi, anzi, in ciò uno degli spunti di maggior interesse per l'impiego di simili modelli in campo organizzativo. Citando Kirman (1992), si osserva come, pur potendo la coazione dei comportamenti di individui semplici e plausibili generare dinamiche complesse, la rappresentazione di quelle stesse dinamiche in un solo agente potrebbe portare a fondare il medesimo su caratteristiche del tutto non naturali, ricollegandosi, con ciò, al dibattito sulla complessità, per puntualizzare la differenza fra aggregati la cui divisione in parti risulti difficile da capire ma possibile, cosiddetti "complicati", e aggregati veramente "complessi", in cui la comprensione di parti e tutto sarebbe legata da una relazione di autoreferenza. Viene rifiutato sia l'orientamento riduzionista dell'individualismo metodologico, che porterebbe a tentare di includere nell'agente l'intera complessità del sistema, sia l'organicismo, che non riterrebbe percorribile la strada degli agenti. Il suggerimento è di procedere con agenti, ragionevolmente semplici, ricercando nell'interazione e nella presenza di strutture ex-ante i mezzi atti a riprodurre la reale complessità, che potrà essere letta alla scala più adatta, non sempre coincidente con quella del singolo individuo.

Riassumendo: i modelli basati su agenti risultano particolarmente adatti a riprodurre situazioni complesse, poiché permettono la scomposizione degli elementi, da cui la complessità stessa scaturisce, in oggetti che non abbisognano di avere consistenza particolarmente complicata, né, tanto meno, debbono tentare di riassumere al proprio interno la complessità totale, in ordine alla quale sono chiamati a fornire unicamente un contributo. In questi modelli, la loro stessa dinamica, risultante dall'autonomia delle azioni individuali, diviene l'elemento che consente di inferire il comportamento dell'aggregato. Il ricercatore non deve affatto sforzarsi di prevedere tutte le azioni possibili; ciò implica, tra l'altro, un rischio decisamente inferiore di influire sui risultati del modello attraverso la codificazione del medesimo. Insieme al rischio di influenza viene ridotto anche quello di errore, specialmente laddove si abbia l'accortezza di mantenere un elevato livello di semplicità degli agenti. Un ulteriore vantaggio è dato dalla possibilità di evitare il ricorso a formulazioni matematiche complesse, riducendo ulteriormente il margine di errore e permettendo di superare l'esigenza, tipicamente connessa con il calcolo differenziale, di disporre di funzioni continue per esprimere la dinamica degli aggregati; in molti casi, questo requisito può rendere meno plausibile il modello. Il metodo basato sugli agenti comporta benefici sia durante la fase attiva, di esecuzione della computazione, sia durante la fase di predisposizione del modello.

Per quanto concerne la fase di computazione, si riscontrano vantaggi, nei confronti delle tecniche di inferenza statistica, riguardo alla maggior duttilità dei modelli di agenti, derivante, principalmente, dalla possibilità di rieseguire la simulazione variandone, eventualmente, i parametri. In questo modo è possibile sia verificare i risultati teorici su un maggior numero di situazioni, sia disporre di nuovi dati, in tempi e con costi inferiori a quelli che caratterizzerebbero una collezione tramite interviste. E' anche possibile eseguire, inizialmente, una versione minimale del modello dove agenti, estremamente semplici, interagiscano con l'ambiente, da essi stessi formato, in pochi modi, per giungere, tramite l'osservazione dei comportamenti, a decidere in quale direzione proseguire la ricerca, modificando le capacità degli agenti o le caratteristiche dell'ambiente. Questo metodo di affinamento successivo del modello permette di procedere a verificazione continua del medesimo, e della teoria su cui si basa, garantendo la certezza di operare ogni aggiunta su una base solida e funzionante. Se si dovesse condurre una simile sperimentazione nella realtà, con interviste o elaborazioni di dati quantitativi, si dovrebbero stabilire, prima della raccolta dei dati, quali aspetti del comportamento dei soggetti osservati indagare; ogni cambiamento in merito a quelli, pur se connesso con l'emergere di sostanziose novità, correrebbe, quasi sicuramente, il rischio di essere frustrato dal significativo incremento dell'impiego di denaro e tempo connesso con la ripetizione dell'indagine.

Durante il lavoro di predisposizione del modello la versatilità del metodo consente, tramite la progressiva modificazione dell'agente, di giungere a livelli elevati di specifica precisione d'azione; il procedimento potrebbe riguardare i soli ambiti di particolare interesse, emersi, di volta in volta, dalla sperimentazione, condotta con agenti meno specializzati. Il metodo risulterebbe, nuovamente, più economico di quello basato su un procedimento di tipo "*top-down*": in quel caso, tutte le attitudini dell'agente rappresentativo dovrebbero essere descritte, fin dall'inizio, nei minimi particolari, implicando, tra l'altro, l'estensione al più ampio livello di dettaglio della collezione dei dati, da usare nel processo di verificazione. Sempre in relazione alla struttura del modello, derivante dall'adozione del metodo, va citato il vantaggio di poter agevolmente riutilizzare parti del medesimo per la conduzione di altre sperimentazioni: se le conoscenze stanno nell'agente ed i risultati derivano dalla interazione dei singoli individui, la stessa progressione combinatoria, che caratterizza la crescita della complessità in natura, enumera le possibilità di ottenere modelli diversi modificando le modalità di interazione dei medesimi individui artificiali.

Gli agenti, impiegati nei modelli, sono, normalmente, dei programmi in grado di simulare il comportamento di singoli operatori o di istituzioni. Risulta basilare, per il funzionamento del metodo, la capacità di ciascun agente di operare nell'ambiente, cioè di intraprendere comportamenti volti a migliorare i risultati, ottenibili durante l'interazione con esso, o con gli altri agenti, per il suo tramite. Queste capacità possono essere simulate in un programma sfruttando diversi orientamenti, ciascuno dei quali comporta una diversa elasticità di azione del programma stesso. La determinazione dell'azione da effettuare può avvenire in base a regole, precedentemente stabilite, codificate direttamente nelle istruzioni del programma; un secondo metodo consta della redazione di sistemi intelligenti, cosiddetti "esperti": capaci di effettuare la selezione della regola da applicare, data la situazione specifica, in un insieme precedentemente codificato. Decisamente più interessante pare, però, essere la metodologia basata sull'utilizzazione di sistemi in grado apprendere: reti neurali, algoritmi genetici, *classifier system* ed altri; in questo caso gli agenti divengono capaci di sviluppare regole comportamentali proprie, modificando un insieme iniziale, solitamente valorizzato in modo casuale, sulla base delle esperienze acquisite. Sia i sistemi esperti sia i metodi basati sull'apprendimento rappresentano prodotti della "*Artificial Intelligence*" di cui si parlerà nel prossimo capitolo.

Il primo metodo comporta una maggiore semplicità di codificazione, ma questa si accompagna alla notevole limitazione connessa con l'impossibilità di apprendimento; simili programmi possono interagire adeguando il loro comportamento, ma solo nell'ambito dei casi previsti dal programmatore; le loro azioni si modificano solo al raggiungimento di soglie, predeterminate, da parte dei valori di specifiche variabili. Il loro impiego permette comunque di ottenere alcuni risultati non banali, che confortano le osservazioni precedenti sulla possibilità di ottenere effetti complessi dall'interazione di parti molto semplici.

L'utilizzazione di sistemi esperti permette di inserire un maggior grado di autonomia nell'azione del programma, consentendo anche la scoperta di soluzioni nuove, per lo meno dal punto di vista della sequenza di applicazione delle regole. La predisposizione preventiva delle regole, però, comporta ancora il rischio di immettere elementi in grado di influire eccessivamente sull'azione del modello, con la conseguenza di condizionarla, riducendo la significatività dell'esperimento.

Risultati decisamente più qualificanti possono essere ottenuti con l'impiego di programmi in grado di manipolare le proprie regole di comportamento, modificando quelle che non risultassero adatte all'ambiente. Si permette, in questo modo, una evoluzione delle capacità di ciascun agente, dotata del grado di autonomia ritenuto più adatto alla particolare applicazione. Il paradigma dell'evoluzione sta alla base dei metodi detti, appunto, "evolutivi": algoritmi genetici, *classifier system*, programmi genetici, eccetera, su cui si concentra l'interesse della presente trattazione.

Margarita (1992) ravvisa nella applicazione di algoritmi evolutivi, in particolare degli algoritmi genetici, il rovesciamento dei metodi econometrici, delineato nelle righe precedenti fra i vantaggi ottenibili con modelli basati su agenti. Il procedimento classico viene articolato, solitamente, in tre passi principali: scelta del modello, stima dei parametri e valutazione della validità del modello stesso. Il primo passo riguarda, in particolare, la scelta della forma, lineare o meno, delle equazioni rappresentative; la stima dei parametri è, tendenzialmente, basata sui dati a disposizione; la validità

del modello viene verificata in riferimento al profilo, interpretativo o previsionale, a seconda dei fini perseguiti. I metodi evolutivi permetterebbero una modellizzazione a posteriori; essa si esplica, come definito in Margarita (1992):

> (…) nella determinazione e nella costruzione "automatiche" della struttura e dei parametri del modello sulla base dei dati stessi.

Dagli studi di Terna, risulta la possibilità di ottenere ottimi risultati applicando la metodologia delle reti neurali, particolarmente nella gestione di problemi relativi alle previsioni ed alla formazione delle aspettative. I due metodi possono, inoltre, essere impiegati in modo congiunto: il contributo degli algoritmi genetici si avrebbe nel guidare la scelta tra diverse possibili architetture di reti neurali e nella ottimizzazione della struttura di queste ultime, volta a diminuire il numero di connessioni nella rete, migliorarne i tempi di apprendimento e ridurne la complessità. Spears e De Jong (1990) ipotizzano che la tecnica delle reti neurali meglio si adatti a problemi piccoli e semplici, mentre i metodi evolutivi risulterebbero più utili per applicazioni di grandi dimensioni e complessità; la possibilità dovrebbe essere quella di congiungere i due metodi, utilizzando le reti per l'ottimizzazione locale ed i metodi genetici per la ricerca globale. Simile connubio è ipotizzato anche in Margarita (1992) a proposito della ipotesi di operatori artificiali, il cui agire risulterebbe basarsi su procedimenti "*bottom-up*", incorporanti: i meccanismi comportamentali elementari dei metodi genetici, e gli aspetti dell'intelligenza di livello più alto delle reti neurali.

Molti economisti vedono, dunque, nelle possibilità offerte dai procedimenti di determinazione dal basso, in combinazione con l'utilizzazione di programmi "intelligenti" una via, di sicuro interesse, per costruire modelli in cui l'interazione fra gli operatori non debba essere prevista a priori fin nei minimi dettagli, risultando, essa, scaturire dalla elaborazione autonoma di regole comportamentali da parte dei singoli, impegnati nel tentativo di migliorare i risultati del proprio rapportarsi con l'ambiente. Questo comportamento, non necessariamente massimizzante, risulta, per nota esperienza anche di altre discipline, più aderente all'effettiva azione degli operatori e permette, spendendo gli opportuni accorgimenti, di far rientrare nel modello fenomeni sociologici influenti ed importanti come il formarsi e modificarsi dei gruppi di individui. Si verrebbe, inoltre, a disporre di un'ulteriore metodologia per illustrare i comportamenti dei sistemi economici che andrebbe ad affiancarsi alle due usuali, costituite dalla trattazione puramente descrittiva e dai modelli matematici.

Nella accezione più ampia, un modello del tipo citato si qualificherebbe per la presenza di entità, capaci di rappresentare singoli individui o istituzioni, in grado di intraprendere azioni, in risposta a stimoli ambientali, volte al conseguimento di obiettivi autonomamente determinati. Le conseguenze di dette azioni si tradurrebbero nella mutazione della situazione patrimoniale o di una qualche misura di utilità dell'agente, in modo da costituire una informazione capace di guidare l'apprendimento del medesimo in base all'esperienza. Insiemi di agenti, autonomi e dotati di scarse o nulle conoscenze del contesto, nella loro interazione con un ambiente simulato, in grado di riprodurre le caratteristiche conosciute dell'ambito di ricerca, risulterebbero capaci di evincere regole ben più profonde e significative delle conoscenze iniziali, grazie alla meccanica pazienza e caparbietà nel confrontarsi, anche milioni di volte con esso, proponendo di volta in volta soluzioni, in qualche misura, innovative. Lo studio della dinamica di siffatti comportamenti potrebbe essere un valido ausilio per la comprensione del funzionamento, reale, dei sistemi simulati nei modelli.

1.4 - Prodotti per le simulazioni al *computer*: *Swarm*

Nelle simulazioni condotte con il *computer*, un insieme di oggetti, solitamente programmi, popola un ambiente artificiale appositamente predisposto, dove sono riprodotti spazio e tempo. I vari oggetti sono sollecitati ad agire da eventi, generati in base ad orari riferiti allo scorrere del tempo artificiale. Occorre, perciò, predisporre appositi programmi che si occupino di gestire la posizione di ciascun agente nello spazio e di "far scorrere" il tempo artificiale. Sono necessari, inoltre, programmi per recepire i dati iniziali, registrare i risultati ottenuti, aggregare e collezionare risultati parziali, generare ed aggiornare grafici relativi all'andamento delle variabili significative. Quando il modello sia volto ad indagare aspetti connessi alla distribuzione spaziale degli agenti, occorre gestire "mappe" dello spazio,

rappresentando su di esse la posizione di ciascuno. Non bisogna, inoltre, dimenticare gli aspetti tecnici, legati alla gestione della memoria dell'elaboratore, alla comunicazione fra i vari oggetti ed alla simulazione della casualità. In breve occorre gestire tre aspetti peculiari: lo scorrere del tempo, la generazione di numeri per simulare eventi casuali e la distribuzione delle risorse dell'elaboratore fra le varie entità.

La maggior differenza intercorrente fra un esperimento condotto nel mondo reale ed uno simulato al *computer* sta proprio nello scorrere del tempo: nella realtà il tempo passa senza bisogno di interventi e gli agenti si sincronizzano su di esso in base alla misurazione del medesimo; in un mondo artificiale occorre disporre di una componente deputata a gestire lo scorrere del tempo, che si preoccupi anche di mantenere il sincronismo fra le varie altre componenti. In contropartita si ha la possibilità di imprimere al tempo la velocità di scorrimento voluta; occorre, però, rendere perfettamente esplicite le assunzioni in ordine al tempo, onde permettere ad altri studiosi di rieseguire l'esperimento per riprodurne i risultati.

La casualità viene simulata, negli ambienti artificiali, facendo dipendere alcuni eventi dal realizzarsi di condizioni attinenti un numero generato, di volta in volta, in modo pseudo-casuale. L'ottenimento di questi numeri è basato su apposite procedure, piuttosto complicate, che permettono anche di scegliere fra i diversi tipi di distribuzione di probabilità, relative alla generazione di ciascuno dei valori contenuti nell'intervallo voluto. Normalmente questi algoritmi si basano su di un valore iniziale, detto "seme", in modo da consentire di ripetere la sperimentazione, utilizzando gli stessi risultati casuali, più volte; quando si voglia esaminare uno scenario in cui gli eventi casuali siano diversi, il seme dovrà essere variato. Anche questa esigenza è compensata da un vantaggio: nell'osservazione del mondo reale il caso viene "subito" e, molte volte si concretizza in un inquinamento dei dati che genera il bisogno di sottoporre i medesimi a lunghe elaborazioni statistiche; nei mondi artificiali il caso viene "gestito", al pari delle altre variabili del problema, con ciò permettendone l'utilizzazione più idonea.

Ogni componente del mondo artificiale consta di uno o più oggetti *software* ed utilizza una serie di variabili, cioè parti della memoria dell'elaboratore, per memorizzare i dati relativi al suo stato e alle conoscenze acquisite in ordine all'ambiente esterno; è basilare che l'utilizzazione delle risorse della macchina, fatta da ciascuna componente, non risulti influente nei confronti delle altre: occorre evitare che la memorizzazione di un dato da parte di un generico agente, possa, accidentalmente, modificare le variabili di altri, così come ogni programma deve avere a disposizione una precisa parte della memoria, dove risiedere e posizionare le variabili usate. Le comunicazioni fra gli individui del mondo artificiale avvengono tramite interazioni fra i programmi che li rappresentano, quindi la localizzazione di ciascun programma, all'interno della memoria, deve essere registrata e conosciuta onde permettere interazioni corrette.

La predisposizione di tutto il corredo strumentale descritto è propedeutica alla codificazione dei programmi per emulare gli agenti. Questo *software* "di base", inoltre, deve possedere caratteristiche di robustezza, sufficienti a permettere la ripetizione ciclica dell'elaborazione per un gran numero di volte, e consentire l'immissione di parametri, onde rendere agevole l'esecuzione dell'esperimento con presupposti diversi. La codificazione dei programmi costituenti tale apparato costituisce un grosso onere, per il ricercatore, e si accompagna, spesso, con il pericolo di commettere errori che potrebbero compromettere il buon funzionamento del modello.

Esistono pacchetti *software*, generalizzati, in grado di fornire valide prestazioni, che si occupano della gestione di quanto descritto; l'utilizzo di questi permette di evitare l'onere precedentemente descritto e, soprattutto, contribuisce a rendere maggiormente comprensibile il funzionamento del modello. La scelta del prodotto da utilizzare deve tenere conto di alcune caratteristiche: la versatilità, il grado di diffusione, le garanzie in ordine alla manutenzione, particolarmente all'adeguamento continuo ai progressi dei sistemi operativi, la possibilità di ottenere assistenza a fronte di problemi.

Un *software* particolarmente versatile permette di sviluppare il modello con maggior libertà, consentendo di perseguire un forte realismo della simulazione; la sua utilizzazione può essere estesa ad un gran numero di modelli. La diffusione del pacchetto aumenta le possibilità di collaborazione con

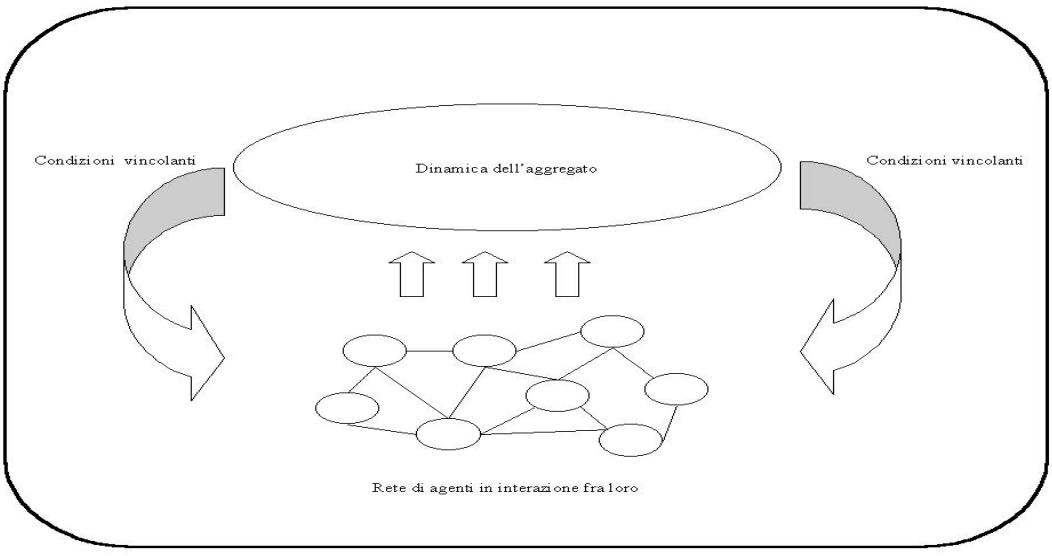
altri ricercatori: i modelli basati sullo stesso pacchetto sono più facilmente trasmissibili e modificabili; notevoli benefici si hanno, inoltre, per la comprensibilità del modello. I prodotti soggetti a continuo sviluppo, permettono di avvantaggiarsi delle novità introdotte e garantiscono la disponibilità di versioni adatte ai più recenti sistemi operativi. L'utilizzazione di questi prodotti può risultare non banale; quindi è importante poter contare su una efficiente assistenza, onde risolvere agevolmente i problemi che potrebbero rallentare o vanificare l'attività di ricerca.

Swarm, oltre a possedere queste caratteristiche, è totalmente gratuito; esso fa parte di quel patrimonio di *freeware* cui si è accennato nel primo paragrafo. *Swarm* costituisce un valido strumento per lo sviluppo di modelli; esso consta di un insieme di librerie, di programmi e protocolli codificati usando il linguaggio *Objective-C*, inteso per la gestione di "sciami" di agenti, che possono a loro volta contenere altri sciami. L'azione avviene sotto la supervisione di un oggetto "osservatore" in grado di rilevare e visualizzare i dati del modello. Una descrizione del pacchetto è contenuta nella definizione, non ufficiale, fornita in previsione del rilascio della prima versione, dagli, allora, curatori del progetto Langton, Minar e Burkhart (1995):

> *Swarm is a multi-agent simulation software platform for the study of complex adaptive systems. In the Swarm system the basic of simulation is the "swarm", a collection of agents executing a schedule of events. Swarm accommodates multi-level modeling approaches in which agents can be composed of swarms of other agents in nested hierarchies. Swarm schedules support hierarchies of time management yielding a natural model of concurrency and a straightforward path to parallel implementation.*

Il prodotto è basato su una struttura a più livelli, in cui ciascun oggetto, può, a sua volta, contenere un intero sciame; il gestore del tempo è in grado di sincronizzare le azioni per tutti i livelli. I vari agenti sono totalmente indipendenti: il loro comportamento risulta dettato soltanto dalla reazione alla situazione ambientale, così come percepita da ciascuno, e dai valori di particolari indicatori di stato. Il concetto di sciame deve essere inteso come metafora di gruppo, ossia di un qualunque insieme, strutturato, di entità interagenti, a prescindere dalla loro natura di api, formiche, batteri, operatori economici o altro. La struttura di *Swarm* si presta alla realizzazione di modelli dove i vari individui risultino estremamente semplici, e tale risulti anche il loro comportamento nella collettività, particolarmente adatta per individuare la soglia di trasformazione dal semplice al complesso; il comportamento dell'insieme scaturisce, in modo non lineare, da quello degli individui componenti; la somma delle loro azioni determina la condotta dell'insieme che, a sua volta, si concretizza quale condizionamento degli individui, in un ciclo senza fine. Il concetto è espresso graficamente nella figura 1.1.

Figura 1.1 - Interazione fra agenti e collettività



Swarm permette di concentrare l'attenzione sulla codificazione dell'esperimento ad un alto livello di astrazione, sollevando l'utilizzatore dalla gran parte degli oneri di programmazione delle componenti infrastrutturali, non direttamente connesse con la specifica realizzazione. Secondo gli autori (Langton, Minar e Burkhart 1995) lo scopo basilare del prodotto è provvedere un contesto di esecuzione in cui un gran numero di oggetti possa "vivere la propria vita", interagendo in modo concorrente e distribuito; condividendo, cioè, le risorse e le informazioni messe a disposizione dall'ambiente. In altre parole l'intenzione è quella di agevolare gli utilizzatori nella loro attività di programmazione, fornendo librerie di programmi, cosiddetti di "utilità", volti a provvedere specifici servizi, fruibili semplicemente indirizzando le opportune richieste agli oggetti ad essi preposti.

L'inserimento di larghe parti del codice in librerie comuni, oltre a ridurre l'onere della programmazione, consente di pervenire allo sviluppo di modelli maggiormente uniformi, più adatti a favorire la collaborazione fra studiosi e, soprattutto, più facilmente comprensibili da numerosi lettori. Questa caratteristica risulta utile nella divulgazione, non solo dei modelli, ma anche dei risultati conseguiti.

I programmi contenuti nelle librerie di *Swarm* sono codificati utilizzando il linguaggio *Objective-C*; nello stesso linguaggio sono normalmente redatti i programmi dell'utilizzatore. *Objective-C* presenta le caratteristiche, di facilità ed immediatezza di codificazione, tipiche dei linguaggi ad oggetti: ogni oggetto consta di un insieme di dati e del codice atto a fornire i servizi richiesti da altri oggetti. La comunicazione fra oggetti è basata su una notazione, eguale per tutti, formata dalla sequenza: indirizzo dell'oggetto, *selector*, parametri. La possibilità di usare variabili, con nome a piacere, sia per memorizzare l'indirizzo dell'oggetto, sia per ospitare i valori dei parametri, permette uno stile di programmazione agile ed immediatamente comprensibile, anche per utilizzatori con scarsa dimestichezza nell'uso di strumenti di calcolo automatico. Il *selector* identifica il servizio richiesto all'oggetto; anch'esso può essere codificato a piacere dell'utilizzatore: è possibile usare nomi che risultino immediatamente esplicativi. La strutturazione del linguaggio, basata sul concetto di funzione, aumenta ulteriormente la comprensibilità del codice che può essere interpretato come sequenza di applicazioni di funzioni, ciascuna delle quali può fornire un valore di ritorno o effettuare determinate azioni. La corrispondenza risulta immediata: la codifica tipica di una funzione

risultato = funzione (valori)

nel linguaggio diviene

```
risultato = [oggetto selector+nome-del-primo-parametro:      valore-del-primo-parametro
                            nome-del-secondo-parametro:      valore-del-secondo-parametro
                                                             …
                            nome-dell'n-esimo-parametro:     valore-dell'n-esimo-parametro]
```

La redazione dei programmi per gli agenti comporta, semplicemente, l'individuazione delle azioni che questi potranno essere chiamati a compiere, ciascuna delle quali diverrà uno dei *selector* per l'oggetto agente e la previsione, per ciascun *selector*, del numero e del tipo di parametri necessari. Il programma, agente, consterà di una serie di blocchi di codice, uno per ogni *selector*, dove saranno indicate le istruzioni necessarie a svolgere una particolare azione. *Swarm* mette a disposizione una ampia serie di oggetti, già costruiti, in grado di fornire servizi disparati, ad esempio: la gestione, totalmente automatica, di finestre grafiche, la collezione di serie di dati, il calcolo di funzioni statistiche, la generazione di numeri pseudo-casuali, la gestione di liste di oggetti. La porzione di codice necessaria a gestire l'azione, connessa ad un *selector*, consta, solitamente, di poche righe di istruzioni. Ogni oggetto può, infine, essere dinamicamente creato o distrutto.

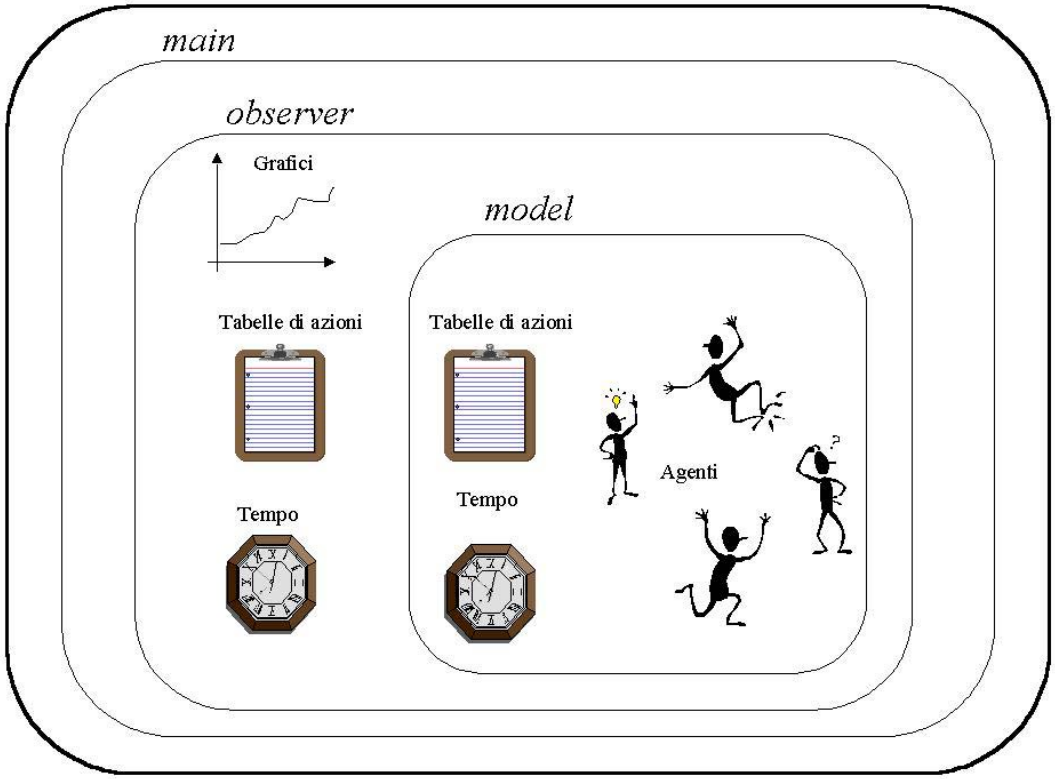
Alla data il mondo *Swarm* conta diverse centinaia di migliaia di utilizzatori, che possono interagire fra loro, e con gli esperti del prodotto, grazie ad una *mailing-list* gestita direttamente dal laboratorio che ha sviluppato il prodotto; in questo modo vengono divulgate sia le questioni relative all'utilizzo del prodotto, sia gli annunci dei nuovi rilasci. Il *software*, come detto, è interamente gratuito e fornito sotto i termini della licenza "GNU": ne sono permesse la modificazione, la duplicazione e la divulgazione, con l'unico obbligo di non apporre termini maggiormente restrittivi,

nei confronti di ulteriori fruitori. *Swarm* può essere installato su svariate piattaforme, fra cui *MS/Windows 95/98/NT, LINUX, SOLARIS* ed altri ancora.

1.5 - Struttura e funzionamento di un modello in *Swarm*

Swarm è in grado di simulare un intero esperimento, completo di un oggetto "osservatore" incaricato di provvedere a gestire: la creazione del modello vero e proprio, il suo avviamento e la collezione dei risultati ottenuti. L'esperimento in *Swarm* è caratterizzato da una struttura gerarchica su due livelli: quello dell'osservatore (*observer*), più esterno, e quello del modello vero e proprio (*model)*; ciascuno dei due contiene oggetti, elenchi di azioni ed orari. L'*observer* è, a sua volta, creato dal programma principale: il cosiddetto "*main*". Al *main* vengono impartiti i comandi relativi ad aspetti generali del lavoro, come: attivare la modalità di *debug* o modificare il seme di generazione dei numeri casuali. Al di sotto del livello del *model* stanno i singoli oggetti che popolano il mondo, cioè gli agenti e le istituzioni, nonché tutti quegli oggetti, genericamente indicati come "di servizio", in larga parte rappresentati dal contenuto delle librerie di *Swarm*. Come già accennato, i vari oggetti possono contenerne altri, a loro volta, divisi in livelli: un singolo oggetto appartenente ad un modello, può, se del caso, contenere un intero esperimento con un proprio *observer*, *model* e la relativa popolazione di agenti; questi, a loro volta, potrebbero contenere un altro esperimento e così via. Un *observer*, inoltre, può controllare diversi *model*, comparandone i risultati e disciplinandone l'esecuzione. Da queste osservazioni risulta palese la grande duttilità e versatilità strutturale del prodotto, che ben si candida quale "*standard*" per lo sviluppo di modelli basati su agenti, offrendo, in addizione, possibilità di intervenire sul comportamento degli stessi a livelli di raffinatezza e complessità, difficilmente riscontrabili in altri *software* per la costruzione di modelli. La figura 1.2 riassume, graficamente, quanto detto.

Figura 1.2 - Struttura di un modello in *Swarm*



Nel linguaggio *Objective-C* il codice di ogni oggetto è contenuto in due *file*: il primo contiene l'elenco dei metodi, il secondo il codice vero e proprio. Un metodo è la parte di codice che serve a

fornire un certo servizio; ciascun metodo è identificato da un *selector*. Durante la compilazione del codice tutti questi *file* vengono tradotti in forma "eseguibile", cioè comprensibile dal processore della macchina, e riuniti in un unico *file*; il nome di questo *file* viene usato, come comando, per ordinare, al sistema operativo, di eseguire il programma. L'esecuzione del programma prevede, innanzitutto la creazione dei vari oggetti, cioè l'assegnazione di parti della memoria, identificate da un indirizzo. E' possibile creare diverse copie di ciascun oggetto, dette "istanze"; una volta terminata la fase di creazione degli oggetti, vengono inviati, a ciascuno, gli ordini relativi alle azioni da compiere.

La fase di creazione degli oggetti viene iniziata dal *main,* creando l'*observer*; questi crea, a sua volta: il *model*, gli oggetti grafici, per visualizzare i dati rilevati, una mappa, o "finestra", per l'immissione dei parametri, ed una serie di "pulsanti", tramite i quali l'utilizzatore può intraprendere azioni sul modello. Il *model* crea: la propria mappa per l'immissione dei parametri, gli agenti e gli, eventuali, oggetti di servizio.

Terminata la creazione degli oggetti, il *main* ordina all'*observer* di creare le "azioni", cioè di preparare una tabella delle rilevazioni di dati, e degli aggiornamenti dei grafici, da effettuare durante l'esecuzione del modello; per ciascuna azione viene stabilito il momento di esecuzione, in base al gestore del tempo contenuto nell'*observer*. Lo stesso ordine è impartito, dall'*observer*, al *model*; anche il *model* crea un propria tabella, corredata degli opportuni orari, scanditi dal proprio gestore del tempo. Il sincronismo fra *model* ed *observer* è automaticamente gestito da componenti del pacchetto *Swarm*.

Da ultimo, il *main* ordina all'*observer* di operare; questi trasmette l'ordine al *model* che comincia ad eseguire le azioni contenute nella propria tabella, inviando gli ordini, ivi codificati, agli agenti ed agli altri oggetti che popolano il modello. Le azioni vengono eseguite ciclicamente dall'*observer* e dal *model*: quando si arriva alla fine della tabella si ricomincia da capo; naturalmente *Swarm* provvede a coordinare l'operato dei due. L'esecuzione del modello può essere arrestata, dall'utilizzatore, in qualunque momento, premendo il pulsante "*stop*"; l'elaborazione può essere, in seguito, ripresa premendo "*start*". Per terminare l'elaborazione occorre premere il pulsante "*quit*". Per eseguire un solo ciclo di elaborazione, cioè per effettuare tutte le azioni, contenute nelle liste, una sola volta, occorre premere "*next*".

Le azioni relative alla visualizzazione dei dati e agli ordini da impartire agli agenti sono codificate dall'utilizzatore; per agevolarne la codificazione gli agenti, o meglio gli indirizzi degli oggetti che li rappresentano, possono essere memorizzati in appositi oggetti detti "liste". Le liste agiscono come intermediari fra il *model* e gli oggetti, così che l'invio di un ordine a tutti gli oggetti, presenti in una lista, possa essere effettuato, semplicemente, inviando lo stesso ordine alla lista, che provvederà a propagarlo, in sequenza, ai vari oggetti. Per evitare di utilizzare sempre la stessa sequenza di attivazione, il contenuto delle liste può essere rimescolato, richiedendo il servizio ad un oggetto "*shuffler*" già predisposto in *Swarm*. Un agente può essere escluso dal modello rimuovendone l'indirizzo dalla lista; l'ingresso di nuovi operatori nel mondo artificiale, viene gestito con la semplice azione di aggiungere i loro indirizzi alla lista.

I grafici, eventualmente previsti, vengono aggiornati dall'*observer* durante l'elaborazione; l'utilizzatore può, in qualunque momento, modificarne il livello di dettaglio in modo dinamico. La posizione degli agenti nel mondo artificiale può essere visualizzata in una apposita finestra grafica, dove ogni agente è rappresentato da un punto; agendo, con il pulsante destro del *mouse,* su uno di questi punti è possibile accedere alle variabili interne dell'agente. Una volta selezionato, nel modo descritto, un agente, tenendo premuto il pulsante del *mouse* e spostando il medesimo su una finestra contenente un grafico, i dati dell'agente vengono automaticamente visualizzati anche sul grafico.

L'onere di programmazione, in relazione a queste possibilità, risulta veramente esiguo: le variazioni da apportare al codice di *main*, *observer* e *model*, sono minime; per la gran parte delle realizzazioni, la loro struttura può essere mutuata direttamente dagli esempi forniti col pacchetto. All'utilizzatore, è richiesto, unicamente, di aggiungere le istruzioni di creazione degli oggetti e di elencare le azioni da compiere, specificando, se necessario, tempi diversi per l'esecuzione delle varie azioni. L'onere maggiore è quello della codificazione dei metodi degli agenti che, si è visto in precedenza, risulta abbastanza semplice.

1.6 - La struttura interna dei modelli: lo schema ERA

La divulgazione dei metodi computazionali e la possibilità di divulgarne i risultati, dipendono in larga misura, dalla possibilità di illustrare chiaramente il funzionamento dei modelli. In Judd (1996) si cita la mancanza di precise convenzioni fra i problemi che rendono difficoltosa la diffusione del metodo:

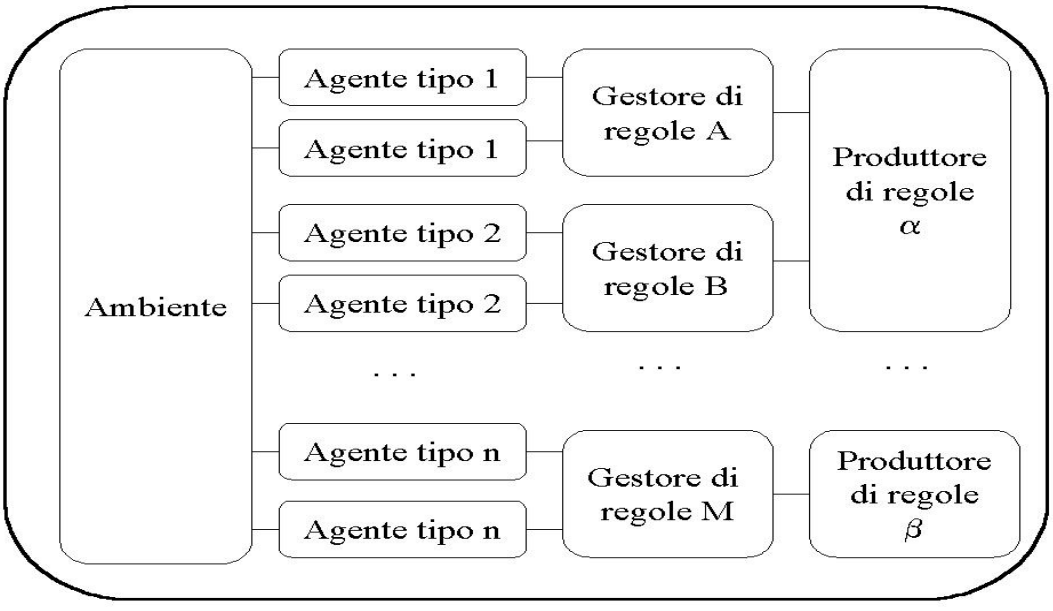
> *Part of the problem is that computational economics has not yet developed the standards, the discipline, and the coherent core of techniques which characterize other subdisciplines in economics, such as econometrics.*

L'adozione di *Swarm* e la scelta, per altro conseguente, di *Objective-C*, come linguaggio di programmazione, permettono di ottenere vantaggi in questo ambito. Nessuna convenzione viene, però, stabilita per quanto riguarda la struttura dell'insieme, di agenti ed istituzioni, che costituisce il modello vero e proprio.

In Terna (1998) si trova una proposta volta a uniformare la struttura dei modelli basati su agenti, con particolare riferimento a quelli sviluppati adottando i protocolli *Swarm* o, comunque, utilizzando linguaggi di programmazione ad oggetti. Nel testo viene suggerita l'adozione di uno schema denominato *Environment-Rule-Agent* (ERA): il lavoro si propone di agevolare sia l'utilizzazione del modello da parte di altri ricercatori, sia la predisposizione del medesimo e l'esecuzione di modificazioni successive.

Lo schema, riprodotto nella figura 1.3, volutamente generalizzato come metodo, prevede una netta separazione fra l'ambiente e gli agenti che, per la gestione delle proprie regole di comportamento, si avvalgono dei servizi forniti da oggetti specializzati, appartenenti ai due livelli successivi. Gli agenti non comunicano direttamente fra loro ma per il tramite dell'ambiente che, ad esempio, potrebbe fornire a ciascuno le informazioni sufficienti a formare una lista dei propri vicini o corrispondenti. Il gestore di regole si occupa di selezionare la norma di comportamento più adatta alla specifica situazione; questa viene descritta dalle informazioni, relative all'ambiente, comunicate dall'agente fruitore dei servizi del gestore di regole. Il produttore di regole è incaricato di ricercare nuove regole, solitamente ottenute a partire da quelle esistenti, così da realizzare una sorta di "apprendimento" da parte dell'agente.

Figura 1.3 - Schema ERA



Sull'architettura iniziale è possibile inserire, due ulteriori elementi, posizionati fra il livello dell'agente e quello del gestore di regole, volti a facilitare le comunicazioni fra questi: un magazzino (*datawarehouse*) per contenere il patrimonio di regole proprio di ciascun agente, ed un interprete (*interface*) capace di tradurre dati espressi nella metrica e nomenclatura propria dell'agente in altri, aderenti alla metrica e denominazione utilizzata dal gestore di regole. L'utilizzo del *datawarehouse* permette, ad agenti diversi, di condividere lo stesso gestore e, normalmente, lo stesso produttore di regole, pur conservando una totale indipendenza comportamentale e sperimentando cammini di apprendimento propri. Gestore e produttore di regole, infatti, agiscono sulle regole contenute nel *datawarehouse*: l'elaborazione, effettuata in risposta a ciascuna richiesta degli agenti, risulta indipendente da quelle precedenti ed ininfluente nei confronti delle successive. L'*interface* consente di semplificare il codice dell'oggetto "agente" e di aumentarne l'indipendenza dal tipo di gestore di regole utilizzato.

La marcata diversificazione degli oggetti, appartenenti ai livelli di ERA, consente di individuare, immediatamente, dove posizionare le singole componenti; si può così procedere sulla base di un robusto impianto strutturale, evitando la redazione di codici confusi e di difficile comprensione e gestione. I vari oggetti sono specializzati per svolgere determinate funzioni: si evita la duplicazione delle istruzioni deputate al presidio di questi servizi, il codice risulta più comprensibile e la sua manutenzione più agevole. L'illustrazione dei vantaggi si ha in Terna (1998):

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

La modularità permette anche l'utilizzazione di componenti diverse, confezionate in precedenza: avendo l'accortezza di adottare la stessa nomenclatura per i *selector* di diversi gestori di regole, la loro sostituzione potrebbe avvenire senza variazioni nel codice degli agenti; le differenze, derivanti dall'adozione di gestori di regole diversi, sarebbero esclusivamente in carico agli oggetti intermedi: *datawarehouse* ed *interface*. In questo modo sono facilitati gli scambi di esperienze: se si volesse sperimentare la metodologia di gestione delle regole utilizzata da un altro studioso, in mancanza di una così precisa impostazione, si dovrebbe, probabilmente, intervenire su gran parte del codice; se anche questi adottasse lo schema ERA, invece, l'intervento comporterebbe poco più della ricompilazione del programma.

La separazione, fra l'agente ed il proprio patrimonio normativo consente, inoltre, di codificare oggetti in grado di sfruttare i servizi di più gestori di regole: è sufficiente dotare ciascun agente di più magazzini e di *interface* separate, specializzati per ciascun gestore di regole.

2 - La rappresentazione della conoscenza (IA)

Dopo il breve esame dell'ambito della ricerca, condotto nel capitolo precedente, in questa parte vengono affrontate le tematiche legate alle tecniche di intelligenza artificiale, utilizzate per dotare gli agenti della capacità di modificare il loro comportamento, durante l'interazione con l'ambiente, verso il conseguimento di propri obiettivi.

La prima parte del capitolo contiene alcune osservazioni generali sul concetto di intelligenza e sulle possibilità di riprodurla con strumenti di computazione automatica. Nel primo paragrafo vengono svolte considerazioni riguardanti la connotazione dell'intelligenza e le attitudini che, indicandone la presenza, permettono di qualificare come intelligente un sistema. Il secondo punto tratta dei progressi nel campo della IA ed elenca le principali realizzazioni con particolare riferimento alla *Artificial Life*.

Il terzo paragrafo contiene una breve disamina delle tecniche tradizionali utilizzate nella risoluzione automatica dei problemi: vengono illustrate le tematiche legate alla rappresentazione ed utilizzazione della conoscenza, le tecniche di ricerca ed il funzionamento dei sistemi di produzioni.

Il tema dell'evoluzione viene introdotto, nel quarto paragrafo, con il richiamo degli aspetti fondamentali delle teorie di Lamarck e Darwin e l'indagine sulle similitudini fra evoluzione naturale e tecniche di ricerca in IA. A seguire vengono presentate le tre principali tecniche evolutive: algoritmi genetici, programmi genetici e sistemi classificatori.

Gli ultimi paragrafi sono dedicati ad una disamina approfondita degli algoritmi genetici: vengono trattate, in modo formale, le basi teoriche del metodo, si procede alla illustrazione del funzionamento e si accenna ad alcune possibilità di estensione. Nell'ultima sezione sono descritti i *classifier system*, basati sull'utilizzazione degli algoritmi genetici per dotare di elevate capacità di apprendimento i tradizionali sistemi di produzioni.

2.1 - Intelligenza naturale ed artificiale

Secondo molte definizioni, l'intelligenza si manifesta per il tramite di comportamenti che sottendono un insieme di attitudini. Si trova in Yam (1999):

> Probabilmente, una definizione esatta di intelligenza è impossibile; tuttavia i dati di cui disponiamo ce ne indicano almeno una - la capacità di gestire la complessità e di risolvere problemi in un contesto utile - indipendentemente dal fatto che si tratti di risolvere un'equazione di secondo grado o di afferrare alcune banane che non siano immediatamente a portata di mano.

La connotazione utilitaristica è presente, con riferimento ampliato dall'individuo alla specie, in studi specifici (D'Allestro 1998) che hanno portato all'ipotesi dell'esistenza di una unica legge di risposta all'ambiente esterno, valevole per tutti gli essere viventi umani, animali o vegetali. Ogni essere reagirebbe all'ambiente con azioni il più possibile atte a garantirne la sopravvivenza; ciò avverrebbe in forza dei meccanismi evolutivi che hanno portato alla selezione dei soli comportamenti ottimali per la continuazione della specie. In forza di queste osservazioni:

> (…) si definisce intelligenza quel comportamento che avviene secondo le leggi di max beneficio globale dell'essere vivente e della sua specie.

L'intelligenza risulterebbe presente in tutti gli organismi viventi dotati di reti neurali, pur se l'azione in ambienti diversi avrebbe portato allo sviluppo di tipi differenti di intelligenza.

Una accezione più restrittiva (Morasso 1998) identifica l'intelligenza come proprietà che si forma nello sviluppo filogenetico e quindi:

> L'intelligenza è la capacità degli organismi di costruirsi rappresentazioni interne del mondo esterno.

L'applicazione di queste definizioni, nella ricerca di manifestazioni di intelligenza in meccanismi di computazione, si basa sulla valutazione dei risultati; quindi (D'Allestro 1998) si potrebbe definire artificialmente intelligente un meccanismo in grado di esprimere comportamenti ottimizzanti. Considerando, invece, l'intelligenza come rappresentazione (Tirassa 1998) risulterebbe difficile immaginare una manipolazione di tipo computazionale della stessa; la computazione deriverebbe infatti da un meccanismo esterno che non pare plausibile operi negli esseri viventi. Un sistema, anche se computazionale, potrebbe comunque esprimere un comportamento intelligente (Cristianini 1998) indipendentemente dal tipo di rappresentazione utilizzata, e quindi la distinzione fra intelligenza e comportamento intelligente potrebbe portare maggior chiarezza. Sarebbe possibile ottenere dalla macchina risultati tipici dell'intelligenza, ad esempio perché ottimizzanti, a prescindere dal fatto che quella operi in modi diversi rispetto ad una mente animale: si potrebbe distinguere l'azione delle neuroscienze, maggiormente volta allo studio dell'elemento umano, da quella della IA più improntata all'ottenimento di risultati di tipo informatico.

Nella attribuzione della qualifica di intelligente ad un sistema artificiale ha operato per lungo tempo una sorta di meccanismo di valutazione opposta, per cui ciò che può essere ottenuto con una macchina, non costituendo più prerogativa umana, verrebbe guardato con sospetto, come rilevato in Hofstadter (1979):

> A questo proposito esiste un "Teorema" sui progressi in IA: non appena si riesce a programmare qualche funzione mentale, immediatamente si smette di considerarla un ingrediente essenziale del "vero pensiero". Il nucleo ineluttabile dell'intelligenza è sempre in quell'altra cosa che non si è ancora riusciti a programmare. Questo teorema mi venne proposto per la prima volta da Larry Tesler, per cui lo chiamo teorema di Tesler: "L'IA è tutto ciò che non è ancora stato fatto".

Nel tentativo di fabbricare una macchina in grado di manifestare una qualche forma di intelligenza, l'uomo tenta di riprodurre peculiarità sue proprie, tuttavia, non appena queste caratteristiche appaiano in una macchina, o nel *software* che presiede al suo funzionamento, cessano di essere tipicamente umane e, conosciuto il meccanismo su cui si basano, si è tentati di pensare che la vera intelligenza sia una cosa altra. La connotazione di intelligente riferita ad uno strumento verrebbe, quindi, influenzata da fattori sociali la cui azione risulterebbe mutare in funzione dell'evoluzione tecnologica. I primi costruttori di macchine per il computo automatico, colpiti dalle loro capacità, definirono "cervelli" quelle strutture; la rapidità e precisione nella computazione era considerata un indice di intelligenza e quindi quelle macchine furono definite tali. In seguito la considerazione per quelle abilità tese a ridursi proprio in forza del fatto che potessero essere svolte in modo meccanico. Di recente il *computer* Deep Blue è riuscito a prevalere, nel gioco degli scacchi, sul campione mondiale Kasparov ma, sebbene le capacità in giochi di strategia siano normalmente assunte come indicatori di intelligenza, si dubita che quella macchina sia veramente tale. Il problema che si delinea è quello, consueto, dei limiti connessi all'autoreferenza dell'attività di un sistema, l'uomo, che ragiona su se stesso.

Il progredire degli studi sul funzionamento della mente umana arricchisce, in continuazione, di significati nuovi la concezione dell'intelligenza e porta a confutare il successo di molti dei tentativi volti ad imitarla artificialmente. Si genera una sorta di rapporto perverso fra neuroscienze e IA: i progressi delle prime forniscono utili idee alla seconda, ma lo svolgersi di quelli, di sovente, confuta i risultati dell'applicazione delle idee da essi stessi ispirate. Nella direzione opposta, si deve osservare che la sperimentazione di tecniche intelligenti permette di scoprire principi algoritmici del pensiero intelligente che risulterebbero difficilmente individuabili con la sola osservazione del comportamento umano. La riproduzione artificiale può divenire un utile laboratorio dove, variando sistematicamente singoli parametri, sia possibile indagare l'interazione dei vari processi mentali. L'osservazione di sistemi artificiali, inoltre, risulta più facile ed affidabile di quella condotta in relazione a comportamenti naturali e l'ispezione del meccanismo è decisamente semplificata.

In conclusione si potrebbe definire la IA come in Ford e Hayes (1999):

> L'IA è la scienza della costruzione di manufatti cognitivi, fondata sulla teorizzazione algoritmica che permea e indirizza l'attuale scienza cognitiva.

2.2 - Gli studi relativi all'intelligenza artificiale

Il termine intelligenza artificiale è usato per definire la disciplina riguardante le teorie e le metodologie che permettono la realizzazione di sistemi artificiali, volti a fornire modelli computazionali dei processi cognitivi e ad ottenere prestazioni tipiche dell'intelligenza. A seconda dell'importanza data a ciascuno dei due obiettivi enunciati, si distingue un paradigma "forte" ed un paradigma "debole" della IA. Il primo fa riferimento alla esatta riproduzione dei meccanismi di funzionamento dell'intelligenza umana mediante sistemi computazionali; il secondo considera maggiormente l'ottenimento di un comportamento intelligente della macchina, prescindendo da preoccupazioni di plausibilità.

Il paradigma forte si colloca in un ambito interdisciplinare di studi e si vale della contribuzione di psicologia, anatomia, biochimica, fisiologia ed altre discipline. L'ottenimento di risultati nel suo dominio necessita, probabilmente, di una conoscenza dei processi, mentali e cognitivi, naturali maggiormente approfondita di quella attuale, accompagnata dalle possibilità di accedere a tecniche fortemente sofisticate. Le ricerche relative all'imitazione dei sistemi biologici hanno contribuito ai progressi dell'IA soprattutto fornendo l'ispirazione per strutture, sfruttate nell'ambito del paradigma debole, per la realizzazione di potenti meccanismi artificiali.

Il paradigma debole si pone, prevalentemente, nel campo dell'informatica e risulta caratterizzato da una più accentuata praticità. A sostegno del paradigma debole, è possibile addurre la scarsità, se non addirittura l'assenza, di evidenze scientifiche che permettano di assimilare la mente umana ad una macchina per il computo automatico e la difficoltà di valutare i risultati del sistema in termini di manifestazione di intelligenza; si rivela, invece, piuttosto semplice accertarne la bontà delle prestazioni in termini di efficacia dei risultati e rapidità di ottenimento degli stessi. Le applicazioni pratiche sfruttano la maggior flessibilità di realizzazione consentita dall'adozione del paradigma debole che, quindi, non va inteso come secondario o meno importante rispetto all'altro.

Si potrebbe datare la nascita della IA attorno agli anni cinquanta di questo secolo; elemento trainante della ricerca fu la convinzione di poter descrivere ogni aspetto e caratteristica dell'intelligenza umana in modo da renderla riproducibile in una macchina in grado di emularla. Si era nella fase di iniziale entusiasmo nei confronti del calcolo automatico: le macchine costruite, come primi esperimenti, presso la Bell Telephone Laboratories ed il Mark I di IBM, basate su un funzionamento elettromeccanico, avevano da poco ceduto il passo allo ENIAC (*Electronic Numerical Integrator and Computer*), realizzato dalla Moore School of Electrical Engineering su progetto della università di Pensylvania, capace di una velocità di calcolo mille volte maggiore grazie all'impiego di circuiti elettronici. Nel 1951 entrava in funzione lo UNIVAC (*Universal Automatic Computer*) realizzato a valvole, in grado di utilizzare nastri magnetici veloci e di svolgere elaborazioni sia scientifiche sia gestionali, che segnava l'inizio della fase di utilizzazione industriale del calcolo automatico. Risulta quindi comprensibile come il tentativo di sfruttare questo, per allora, enorme potenziale di calcolo e la curiosità nei confronti del nuovo mezzo portassero a perseguire obiettivi di così largo respiro. A porsi il fondamentale quesito sulle capacità delle macchine di pensare fu, tra i primi, Turing che propose il noto procedimento di verificazione basato sulla capacità della macchina "intelligente" di emulare il comportamento umano in una conversazione.

Dopo l'iniziale entusiasmo collegato all'ottenimento dei primi risultati, la necessità di disporre di macchine molto potenti e costose ha costituito un freno al progredire della materia: nonostante lo sviluppo, in campo teorico, di ottime proposte, le realizzazioni sono state maggiormente improntate a sfruttare le tecnologie acquisite. Con l'applicazione in campo industriale i programmi intelligenti sono divenuti oggetto d'interesse anche da un punto di vista merceologico e di mercato.

A partire dalla metà degli anni ottanta, la diffusione dei personal *computer* ha favorito la crescita dell'attività di ricerca, grazie, anche, alla possibilità di allargare il numero di addetti, dato il minor costo degli strumenti. Il forte progresso nella realizzazione di piccoli elaboratori ha permesso, nell'ultimo decennio, di dotare queste macchine di notevole potenza, pur riducendone sensibilmente il costo. Ciò ha contribuito alla divulgazione dell'interesse per l'IA, anche verso studiosi delle più disparate discipline che hanno visto nelle tecniche di IA un mezzo per potenziare le possibilità loro offerte dall'ausilio del calcolo automatico, confortati, in aggiunta, dalla "robustezza" dimostrata dalla metodologia nei campi di applicazione industriale. L'IA è una realtà ben consolidata, come dimostra l'osservazione dell'Associazione Italiana per l'Intelligenza Artificiale (1998):

> L'intelligenza artificiale è di interesse strategico, da un punto di vista economico, infrastrutturale e conoscitivo. Le tecnologie dell'intelligenza artificiale infatti contribuiscono allo sviluppo di prodotti informatici centrati sull'utente, adatti alla cultura organizzativa delle aziende e della società moderna.

I campi tradizionali di maggior successo, anche commerciale, dell'IA hanno riguardato ambiti fortemente diversificati quali: la robotica, l'automazione d'ufficio, la comprensione del linguaggio naturale, la creazione di basi di dati "intelligenti", l'utilizzo del *computer* nell'istruzione, la traduzione automatica di testi, l'automazione industriale, la dotazione dei *computer* di capacità espressive vocali e la progettazione guidata da meccanismi automatici. Un ulteriore, fertile, ambito di applicazione è stato, e tuttora permane, quello dei cosiddetti "sistemi esperti", capaci di incorporare le conoscenze di una molteplicità di esseri umani, esperti, appunto, in determinate branche, per utilizzarli nella conduzione di attività di presidio di processi, fornire ausili nella diagnostica, ad esempio in campo medico, e, genericamente, laddove la conoscenza di pochi possa essere utilmente resa fruibile per un numero di interventi incompatibile con le disponibilità temporali e spaziali dei medesimi. Nella sua esperienza lavorativa chi scrive ha avuto occasione di utilizzare il prodotto AOC di IBM, volto a consentire l'automazione delle operazioni delle sale controllo dei grandi elaboratori, essenzialmente basato sul paradigma della risposta ad eventi e situazioni, facilmente configurabile, a suo parere, come un sistema esperto dotato di regole complesse, implicanti la possibilità di attivare combinazioni diverse di processi algoritmici a seconda della situazione contingente.

Nell'ambito della ricerca sono noti gli esperimenti a proposito della dimostrazione di teoremi, della riproduzione artificiale della visione o del linguaggio naturale, della programmazione automatica, delle applicazioni nei giochi di strategia quali la dama e gli scacchi; ultimamente sono disponibili anche programmi "intelligenti" per il gioco del bridge, le cui caratteristiche comportano la generazione di strategie complesse e la gestione di svariate alternative.

Le tecniche IA sono utilizzate nella programmazione di agenti per la conduzione di simulazioni di "vita artificiale", cioè esperimenti in cui automi, reali o simulati, siano in grado di evolvere proprie regole comportamentali che consentano di migliorare i risultati ottenuti durante l'interazione con altri automi e con l'ambiente.

L'idea di vita artificiale venne utilizzata, inizialmente, da Von Neumann e Burks negli esperimenti, basati sugli automi cellulari, volti ad indagare la possibilità di un sistema di replicare se stesso. Viene definito "automa cellulare" un sistema che possa avere un numero finito di stati, operante in condizioni in cui lo stato di ciascun elemento risulti determinato da quello dei suoi vicini: in questo modo il comportamento di ognuno viene condizionato da quello degli altri che, esso, contemporaneamente condiziona. Facendo agire l'automa si perviene allo sviluppo di una rete di interazioni il cui comportamento globale risulta scarsamente prevedibile, pur quando le leggi che governano i passaggi da uno stato all'altro siano estremamente semplici: si assiste, in breve, all'emergere della complessità dall'interazione di molti individui singolarmente semplici e, soprattutto, il sistema diviene capace di replicarsi autonomamente utilizzando le sole informazioni contenute nella propria descrizione. Le informazioni, interpretate, vengono tradotte in azioni da compiere per realizzare il replicante; la copia delle stesse fornisce la descrizione di se stesso al replicante. L'esperimento suscitò particolare scalpore quando, scoperta la struttura ed il funzionamento del DNA, fu possibile notare che la cellula utilizza un meccanismo analogo.

Il lavoro nel campo delle simulazioni di vita artificiale si è spinto particolarmente nella direzione dello studio degli aggregati e delle società, per indagare il formarsi delle istituzioni e dei gruppi. In questi modelli gli agenti risultano dotati di autonomia comportamentale e capacità di apprendimento; essi sono, normalmente, impegnati nel conseguimento di fini propri, ma dalla loro interazione possono emergere comportamenti fortemente sociali. Scaturiscono, dall'esperienza di simili esperimenti, due aspetti basilari, strettamente legati fra loro: l'uno inerente la percezione dell'ambiente da parte dell'individuo e l'altro riguardante le modalità di apprendimento. Nella realtà la percezione del contesto è caratterizzata da un forte grado di incertezza e imprecisione, che, pur parendo un ostacolo, porta, invece, allo sviluppo di un meccanismo di apprendimento maggiormente flessibile e privo di conflitti di precedenza artificiali, che potrebbero essere immessi durante la codificazione di un sistema esterno di risposte agli stimoli. L'ambiente artificiale, perciò, non dovrebbe essere particolarmente "comodo" per l'automa, onde permettere un apprendimento basato su una visione del contesto sviluppata in proprio dall'agente e non su quella fornita dal programmatore: un qualche grado di incertezza risulterebbe importante al fine di migliorare la simulazione. L'apprendimento può avvenire su base genetica, cioè per ereditarietà, o autonoma, cioè utilizzando l'esperienza progressivamente accumulata da ciascuno; oggi è riconosciuto un contributo congiunto dei due tipi che si potrebbe ottenere con l'integrazione delle tecniche basate su algoritmi genetici, per l'ereditarietà, e su reti neurali, per l'esperienza, riprodotta dal meccanismo di propagazione all'indietro per modificare i "pesi" delle connessioni fra neuroni. In questa fase, che si potrebbe considerare iniziale, tende a prevalere l'uso singolo delle due tecniche con maggior ricorso a quella genetica che consentirebbe la selezione delle specie di automi più adatti al compito.

Le ricerche in questo campo sono nate da intenti pratici nell'ambito della robotica: lo scopo era quello di realizzare agenti artificiali, accoppiati con l'ambiente fisico in cui risultassero situati, per simulare il comportamento di meccanismi capaci di apprendere dei compiti e muoversi efficacemente in un ambiente reale. Risulta immediata la possibilità di sostituire l'obiettivo della manovra efficiente in ambienti accidentati o dove agiscano altre entità in movimento, con altri, di maggior interesse economico, come quello di prendere decisioni in relazione alle mutate condizioni di un mercato, o di esprimere strategie di prezzo correlate con quelle dei concorrenti.

2.3 - Le tecniche tradizionali di IA

Prima di procedere, nei prossimi paragrafi, alla illustrazione delle teorie evolutive e dei meccanismi genetici, è opportuna una breve disamina delle tradizionali metodologie di IA che, anche se non utilizzate nel prosieguo, possono, comunque, costituire una buon supporto per la spiegazione del funzionamento degli algoritmi genetici e dei sistemi classificatori. Durante l'illustrazione vengono, inoltre, enunciati principi e problemi della IA comportanti alcuni limiti superati con le tecniche evolutive; ciò permetterà una più estesa individuazione dei benefici connessi con l'utilizzazione di queste ultime.

L'ipotesi di base degli studi in IA è nota col nome di "ipotesi del sistema dei simboli fisici"; essa risulta parzialmente superata dall'impostazione, tipicamente sub-simbolica, delle reti neurali e degli algoritmi genetici, pure conserva una considerevole capacità esplicativa, se si ha cura di non interpretare il simbolo come portatore di una informazione completa, ma di singole parti di quella. La formulazione del teorema e la definizione di sistema dei simboli fisici sono reperibili in Newell e Simon (1976):

> Un sistema di simboli fisici possiede i mezzi necessari e sufficienti per l'agire intelligente.
>
> Un sistema di simboli fisici consiste di un insieme di entità, dette simboli, che sono schemi fisici che possono comparire come componenti di entità di un altro tipo, dette espressioni (o strutture di simboli). Una struttura di simboli, dunque, è composta di un certo numero di esemplari di simboli collegati in qualche modo fisico (come per esempio dal fatto che un esemplare è vicino ad un altro). Ad ogni istante di tempo il sistema conterrà una collezione di queste strutture di simboli. Oltre a queste strutture, il sistema contiene una collezione di processi che operano su espressioni per produrre altre

> espressioni: processi di creazione, modifica, riproduzione e distruzione. Un sistema di simboli fisici è una macchina che produce nel tempo una collezione di strutture di simboli. Un tale sistema esiste in un mondo di oggetti più grande di quello costituito da queste sole espressioni simboliche.

I calcolatori possono essere programmati per simulare qualunque sistema di simboli fisici; quindi la verità dell'ipotesi assicurerebbe la possibilità di ottenere da essi un comportamento intelligente. Purtroppo tale verità può unicamente essere desunta come regolarità empirica, derivante dal buon esito di un elevato numero di sperimentazioni; essa costituisce, comunque, un importante fondamento a sostegno della capacità delle macchine per il calcolo automatico nello svolgere attività "intelligenti". Il problema inerente la plausibilità della rappresentazione che, nella macchina, risulterebbe fortemente diversa rispetto a quella di una mente umana, può essere risolto (Cristianini 1998) con una delimitazione precisa fra le nozioni di "intelligenza" e di "comportamento intelligente".

Dall'ipotesi dei simboli risulta (Rich e Knigth 1991) la necessità di disporre di conoscenza per ottenere un comportamento intelligente; la conoscenza è voluminosa, difficilmente caratterizzabile, continuamente soggetta a mutazione e dipendente da una organizzazione intrinseca, diversa a seconda dei modi in cui si esplica la sua utilizzazione. Una tecnica IA deve sfruttare la conoscenza in modo da recepire le generalizzazioni possibili, renderla comprensibile per chi dovesse fornirla, facilmente modificabile, utilizzabile in molte situazioni, anche quando si presentasse incompleta o imprecisa, ed, infine, utile come ausilio per superare la sua stessa massa, capace, cioè, di contribuire a restringere il numero di possibilità da considerare. Partendo da questa osservazione è possibile caratterizzare la risoluzione dei problemi tramite i tre paradigmi: della ricerca, dell'uso della conoscenza e della astrazione. La ricerca consente di affrontare quei casi in cui non siano disponibili metodi diretti, cioè regole in grado di associare alla situazione contingente una risposta in linea con l'obiettivo stabilito, che andranno individuate mediante l'esplorazione di stati intermedi, in modo da indirizzare il cammino verso lo stato congruente con i fini preposti. L'uso della conoscenza contribuisce alla ricerca in situazioni complesse, quando le scelte, necessarie ad indirizzare il passaggio fra stati, assumano connotazioni non banali. L'astrazione è necessaria perché il sistema acquisisca la capacità di individuare le informazioni poco importanti che potrebbero influire sulla ricerca, rendendola inutilmente più onerosa.

Le diverse scelte possono essere individuate partendo dalla rappresentazione dei vari stati possibili dell'ambiente: grazie a questo accorgimento il generico problema può essere caratterizzato come la necessità di stabilire quale stato futuro sia più conveniente dato quello attuale, o quale successione di stati risulti più adatta a giungere all'obiettivo, cioè ad un particolare stato dell'ambiente precedentemente determinato. Tra i vari stati sono identificati come "finali" quelli che non permettono di giungere ad altri ma solo di tornare a quello precedente. La ricerca della soluzione viene condotta esaminando, ogni volta, un insieme di regole formate da due parti: uno "schema", da usare come modello di confronto con la situazione attuale ed una "descrizione dell'azione", che fornisce al sistema le informazioni relative al comportamento da intraprendere quando risulti conveniente attivare, cioè seguire, la regola. I sistemi che operano in base a questa metodologia vengono definiti "sistemi di produzioni": essi constano dell'insieme delle regole, dei dati necessari alla messa in opera delle azioni, di una strategia di controllo per stabilire l'ordine di esplorazione dell'insieme di regole e di un sistema per applicarle. Una volta definito lo spazio di stati del problema, il sistema di produzioni opera al fine di individuare un cammino che congiunga lo stato iniziale ad uno stato obiettivo: la individuazione di questo cammino comporta il successo della ricerca; quando, invece, tutti gli stati finali raggiungibili, o raggiunti in un tempo massimo stabilito per la ricerca, non risultino soddisfacenti, la medesima fallisce.

Le strategia di movimento è chiaramente l'elemento centrale di un sistema di produzioni; essa contiene la esplicitazione dei criteri da adottare nella scelta fra più cammini alternativi. Il successo di un sistema dipende strettamente dalla efficacia della strategia di movimento in esso codificata. Una buona norma è quella di "creare movimento": se si esplorasse l'insieme delle regole seguendo sempre lo stesso ordine alla ricerca di una regola applicabile, si finirebbe col scegliere invariabilmente quella più vicina al punto di partenza; come è intuibile, ciò potrebbe facilmente impedire l'individuazione di cammini ulteriori che, magari, risulterebbero migliori. In quest'ottica, una ricerca basata su decisioni puramente casuali potrebbe risultare efficace, conducendo ad esplorare, dato un numero sufficiente di

passi di ricerca, l'intero insieme di regole. Le strategie puramente casuali comportano il rischio di esplorare più volte uno stesso cammino con conseguente calo di efficienza del sistema: occorre, quindi, condurre la ricerca in modo sistematico, evitando di visitare più di una volta ciascuno stato, il che, per altro, garantisce anche il movimento, sia a livello locale, per un singolo passo, che globale, per l'intero cammino.

Una ricerca sistematica si basa sulla memorizzazione dell'elenco degli stati visitati, per poter escludere l'applicazione delle regole che a quelli ricondurrebbero. Usando un rigido criterio di sistematicità si garantisce l'efficacia con buoni livelli di efficienza, quando, però, non si disponga di criteri specifici per individuare la regola migliore, la probabilità che questa sia collocata in un punto preciso dell'insieme risulta uniformemente distribuita; procedere alla scansione dell'insieme partendo sempre da un punto dato, non importa quale esso sia, potrebbe connotarsi come elemento di riduzione dell'efficienza della ricerca. L'adozione di criteri sistematici, pur garantendo una certa quantità di movimento, dovrebbe essere accompagnata da accorgimenti che aumentino ulteriormente il movimento stesso: potrebbe essere utile ripartire nell'esplorazione, ogni volta, da un punto scelto a caso, nell'insieme delle regole; non dovrà, allora, costituire elemento di perplessità il largo uso che si farà nell'applicazione degli algoritmi genetici di numeri casualmente generati, né ritenersi ciò un elemento di riduzione delle tecniche IA ad un mero procedimento di ricerca casuale per prova e correzione dell'errore.

Lo spazio della ricerca può essere rappresentato tracciando un grafo che descriva il cammino fra i vari stati del problema: in orizzontale sono allineati tutti gli stati raggiungibili in un certo passo, mentre nella direzione verticale vengono disegnati quelli visitati. Procedendo nella ricerca si ottiene il disegno di un albero rovesciato, dove il fusto è costituito dallo stato iniziale e le ramificazioni nascono dall'aumentare del numero di scelte, dato dalle possibili combinazioni degli stati che configurerebbero altrettanti cammini. Nel linguaggio della IA è convenzione accettata indicare come movimento in "ampiezza" quello che abbraccia ambedue le direzioni (Figura 2.1) e movimento in "profondità" quello lungo la direzione verticale (Figura 2.2), dove sono riportate le scelte effettuate ad ogni passo. Una ricerca avviene in ampiezza quando, ad ogni passo, si visitano tutti gli stati raggiungibili da quello attuale, procedendo in modo analogo per ciascuno degli stati successivi. Questo metodo garantisce il raggiungimento dell'obiettivo migliore, fra quelli possibili, dato che si basa sull'esplorazione di tutto gli stati ed i cammini, ma risulta poco efficiente. Procedendo in profondità, si visita, ad ogni passo, uno solo degli stati accessibili da quello dato, si percorre, cioè, una delle ramificazioni dell'albero, fino a giungere ad uno stato finale o già visitato; naturalmente la ricerca si interrompe non appena raggiunto un obiettivo soddisfacente. Quando si raggiunge uno stato finale, diverso dall'obiettivo, occorre tornare, percorrendo il ramo all'indietro, fino ad uno degli stati precedenti, procedimento denominato "ricerca all'indietro" o *backtracking*, che consenta di muovere verso uno stato non ancora visitato e ripetere la procedura. La porzione dello spazio, dei cammini possibili, esplorata, risulta tendenzialmente minore e quindi la ricerca diviene più efficiente; non è più garantita, però, l'individuazione della migliore fra le soluzioni possibili.

Figura 2.1 - Ricerca in ampiezza

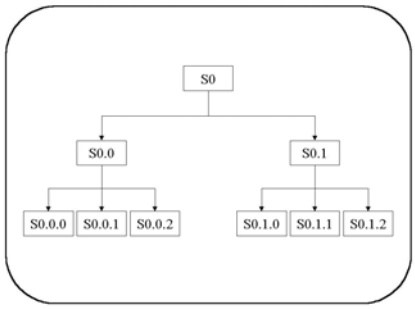
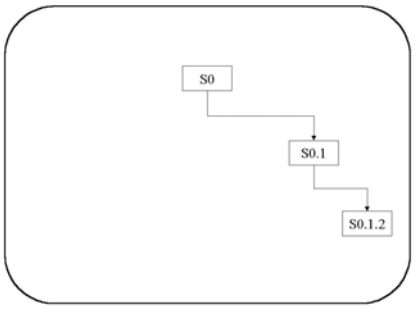


Figura 2.2 - Ricerca in profondità



La ricerca in ampiezza potrebbe simulare il processo di scelta di un investitore che valutasse attentamente diversi titoli prima di decidere quale comprare, mentre quella in profondità ricorderebbe maggiormente il comportamento di uno speculatore che dovesse decidere in tempi brevi.

L'utilizzazione della ricerca in profondità, pur sacrificando quote di efficacia non garantisce un livello di efficienza sufficientemente stabile: nei casi in cui occorra eseguire molte volte la procedura di *backtracking* le prestazioni del programma decadono rapidamente. I tentativi di migliorare la ricerca hanno portato all'adozione delle tecniche cosiddette "euristiche", come quella detta "dell'elemento più vicino": ad ogni passo si seleziona, fra le possibili, l'alternativa localmente, cioè riguardante quel passo, migliore, incrementando, in molti casi, l'efficacia della ricerca pur riducendone i tempi. Normalmente la selezione si basa su un sistema di generazione e verificazione, ottenuto mediante la produzione di soluzioni diverse fino a trovare quella in grado di raggiungere l'ottimo locale. La semplice generazione casuale di soluzioni si rivelerebbe un procedimento piuttosto debole, per questo motivo viene utilizzata una tecnica di "ricerca in salita", in cui la scelta dello stato successivo si basa sul valore di una "funzione euristica", concepita per assegnare a ciascuna descrizione degli stati dei problemi una valutazione in forma numerica.

Un semplice algoritmo di ricerca in salita si basa sulla applicazione, ad ogni passo, di una operazione non ancora sperimentata per quello stato, in modo da muovere verso uno nuovo; se lo stato ottenuto non è l'obiettivo, viene calcolato il valore della funzione euristica per il nuovo stato e si effettua il confronto di questo valore con quello dello stato precedente. La ricerca prosegue, poi, a partire da quello dei due stati che presenta un maggior valore della funzione euristica, ripetendo il procedimento fino al conseguimento dell'obiettivo. Un ulteriore potenziamento del metodo è dato dalla procedura di "ricerca in salita più ripida" o "ricerca lungo il gradiente", dove il confronto del valore ottenuto dalla funzione euristica viene espletato, prima, verso tutti gli altri stati ottenibili da quello di partenza. onde selezionare fra quelli il più desiderabile, e, in seguito, lo stato selezionato viene confrontato con quello di partenza per decidere quale dei due dovrà divenire il nuovo stato di partenza.

E' facile immaginare come la mente umana operi in modi, presumibilmente, molto differenti da quelli descritti; pure un piccolo esempio potrà chiarire come l'applicazione delle procedure illustrate possa fornire risultati in linea con quelli ottenibili da un essere umano. Si consideri di dover ridurre, trasformandola, una espressione algebrica qualsiasi ad una forma maggiormente concisa ed elegante; per eseguire questa attività sono disponibili regole che governano il passaggio fra due stati, cioè fra due forme della espressione. Il problema sta nell'individuare quali regole applicare, per ogni passaggio, per ottenere il risultato voluto nel minor numero possibile di trasformazioni: ciò porta ad escludere subito un procedimento puramente casuale che, pure, sufficientemente ripetuto potrebbe rivelarsi efficace anche se non efficiente; si decide, quindi, di procedere ad una ricerca della soluzione del problema. Nessuno inizierebbe a richiamare alla mente tutte le regole applicabili partendo, ogni volta, dalla prima che ha studiato: quindi il ragionamento si basa sul movimento. Un secondo criterio porta facilmente ad escludere l'applicazione di regole che fornissero, come risultato, una forma già trascritta nei passaggi precedenti: si tende a non rivisitare gli stati, ad essere, perciò, sistematici. E' perfettamente plausibile che ad una certa forma dell'espressione possano essere applicate diverse trasformazioni, le si potrebbero provare tutte per poi ricondurre tale prova sui risultati e così via, ma l'onere sarebbe enorme: è improbabile che si attui una ricerca in ampiezza. Neppure una ricerca in profondità, consistente, ad esempio nello scegliere sistematicamente la prima regola applicabile per ogni espressione ottenuta, fino a giungere ad un livello per cui non siano più applicabili regole, stabilire se questo livello risulti soddisfacente ed effettuare, eventualmente, ulteriori ricerche ripartendo da uno degli stati intermedi, sarebbe una buona idea. E' probabile che si scelga di applicare, ogni volta, la regola di trasformazione che porga come risultato una espressione più concisa, o più maneggevole, rispetto a quella di partenza, attuando così, implicitamente, una ottimizzazione locale. Nella scelta della regola che permettesse l'ottenimento di una nuova espressione, migliore rispetto a quella di partenza, non si procederebbe al confronto di ciascuna trasformazione ottenibile con quella di partenza, ma si procederebbe alla preventiva scelta della regola che apparisse più conveniente per poi stabilire se la sua applicazione fosse opportuna, operando una ricerca lungo il gradiente. L'esercizio continuo porta, naturalmente, ad effettuare la gran parte di queste valutazioni in modo automatico: il ragionamento sotteso non viene esplicitamente effettuato dal soggetto, che si vale della possibilità di memorizzare blocchi di informazione e strategie. Una dimostrazione della presenza di queste memorie può essere facilmente colta nella progressiva acquisizione della capacità di saltare i passaggi elementari, cioè le trasformazioni banali, giungendo più rapidamente alla soluzione.

L'utilizzazione delle tecniche tradizionali di IA comporta alcune esigenze specifiche che metodi più innovativi, come quelli genetici, hanno permesso di superare. La rappresentazione della conoscenza utilizzata è simile a quella del linguaggio umano, il che implica una considerevole estensione della mole dei dati destinati a contenerla; del resto la necessità di fornire al programma conoscenze iniziali, almeno le regole principali e la descrizione degli stati, consiglia di adottare rappresentazioni che agevolino quelle attività. La trattazione dei costrutti di queste informazioni con i normali linguaggi di programmazione, o "imperativi", risulta poco agevole e ciò ha implicato l'adozione di particolari linguaggi "dichiarativi" descritti in Terna (1988):

> I linguaggi dichiarativi (…) descrivono il problema da trattare enunciando le regole che lo governano e i fatti che lo concernono; regole e fatti che saranno combinate anche in modi imprevedibili a priori, ma manifesti al momento dell'uso del programma.

Il metodo di programmazione utilizzato va sotto il nome di "programmazione logica" al proposito della quale si legge in Adorni, Gaglio e Massone (1987):

> La programmazione logica si propone come alternativa ai metodi di programmazione di tipo procedurale usuali. Il suo metodo consiste nello specificare al calcolatore non tanto che cosa fare, quanto ciò che è assunto per vero, cioè degli assiomi, e nel derivarne determinate conseguenze logiche. Ciò significa che un sistema di programmazione logica è essenzialmente un sistema di prova automatica di teoremi, in cui un programma è un insieme di assiomi (o una teoria formale) che può essere interrogato per fornire risposte concernenti il dominio rappresentato. Una computazione è così il tentativo di provare un teorema.

2.4 - Le teorie evolutive

In filosofia il concetto di evoluzione, cioè di graduale derivazione di cose e di esseri, gli uni dagli altri, secondo una perfezione gradatamente maggiore comparve, con la scuola ionica, particolarmente nell'idea del continuo divenire di Eraclito, nel pensiero di Senofane ed in quello di altri atomisti. Nei tempi moderni bisogna rifarsi al concetto di "idea" in Hegel intesa, da quest'ultimo, come sola realtà capace di trasformarsi passando dall'essere Natura inanimata, prima, per divenire Natura animata, Umano spirito ed, infine, Stato, cioè stadio di suprema evoluzione della specie umana. L'idea della mutazione delle specie di animali e piante in conseguenza di progressive modificazioni che diverrebbero ereditarie, si deve a Lamarck. Darwin, ispirato dalla lettura di Malthus, introdusse, in aggiunta, il concetto di selezione naturale: un processo che implica il prevalere, in ogni specie, degli individui più adatti alla lotta per la sopravvivenza, attività imposta a tutti gli esseri viventi dalla ristrettezza della disponibilità di risorse naturali, a sua volta, derivante dalla tendenza alla crescita in proporzione geometrica del numero degli individui stessi. Il concetto di selezione, cioè di scelta, effettuata, secondo Darwin dalla Natura, implica l'esistenza in ogni momento e per ogni specie, di individui con caratteristiche proprie ed emergenti, che producano una fondamentale diversità; acquisisce importanza la variazione a prescindere dal fatto che essa possa, in alcuni casi, risultare sfavorevole: le variazioni vantaggiose tenderebbero ad affermarsi, mentre quelle sfavorevoli sarebbero selezionate per l'estinzione in uno con gli individui di esse portatori. Le scoperte, dell'ultima metà di questo secolo, in campo biologico e, particolarmente, l'isolamento del DNA con i successivi studi sulla sua struttura e sui meccanismi di riproduzione della cellula hanno portato a spiegare l'ereditarietà.

Il metodo, di rappresentazione della conoscenza, adottato in natura presenta caratteristiche di semplicità, maneggevolezza e potenza, che hanno motivato l'interesse dei ricercatori nell'ambito dell'IA. Il genotipo, cioè la combinazione dei geni che costituisce il patrimonio di ciascun essere vivente, contiene tutte le informazioni necessarie allo sviluppo del fenotipo, cioè dell'essere completo e formato. Un genotipo può essere inteso come insieme di geni riuniti in filamenti proteici, il DNA; i geni risultano ottenuti dalla combinazione di vari elementi dei quali, però, solo le quattro "basi", adenina, guanina, timina e citosina, variano. In termini IA potremmo dire che il sistema dei simboli fisici usato per codificare la conoscenza, contenuta nel genotipo, consta di quattro soli elementi, risultando perciò estremamente ristretto ed efficiente; i vari geni, potrebbero esser intesi come regole,

estremamente complesse, capaci di contenere una grande quantità di conoscenza. Data la limitata estensione del sistema dei simboli, il significato di cui è portatrice ciascuna delle quattro basi costituisce una frazione molto piccola della conoscenza di cui è portatore il gene, scaturendo questa dalla diversa combinazione delle prime; da ciò discendono due grossi benefici: il rischio di perdita totale dell'informazione viene fortemente ridotto e le operazioni necessarie per modificare l'informazione risultano estremamente semplificate.

Muovendo dall'ipotesi di Lamarck, si potrebbe individuare nel processo evolutivo una ricerca sistematica; le specie estinte non vengono rigenerate e quindi non si rivisitano inutilmente stati già considerati, ottenuta tramite la memorizzazione delle informazioni, relative ai passi già compiuti, all'interno del patrimonio genetico. La natura opererebbe uno "schiacciamento", incorporando la storia del suo cammino evolutivo, l'albero della ricerca, in una unica descrizione dello stato attuale, capace di rappresentare la somma delle esperienze precedenti e di influire su quelle successive. A proposito dello schiacciamento si ha in Hofstadter (1979):

> (…) L'accorgimento di Samuel per valutare ogni posizione data era di usare sia il metodo dinamico (analisi dell'albero delle mosse) sia quello statico (situazione della scacchiera). Il metodo statico richiedeva una funzione matematica semplice di parecchie quantità caratterizzanti ogni posizione della scacchiera che poteva essere calcolata praticamente all'istante, mentre il metodo dinamico di valutazione richiedeva la creazione dell'"albero" su cui sono rappresentate le possibili mosse, le risposte ad esse, le risposte alle risposte, e così via. Nella funzione statica vi erano alcuni parametri che potevano variare; se ad essi venivano dati valori diversi, si otteneva come conseguenza un insieme di diverse posizioni possibili della funzione di valutazione statica. (…) L'effetto era di codificare parzialmente nei valori dei parametri della valutazione statica la conoscenza ottenuta mediante la ricerca dinamica sull'albero. In breve, l'idea era di "schiacciare" il metodo complesso della valutazione dinamica nella funzione di valutazione statica, molto più semplice ed efficace. (…) Quindi nello schema di Samuel, si attua un tipo intricato di retroazione, nel quale il programma cerca costantemente di "schiacciare" le valutazioni dinamiche in una ricetta statica più semplice; e questa ricetta a sua volta svolge un ruolo chiave nella valutazione dinamica. I due metodi sono dunque intimamente legati e ciascuno beneficia dei progressi dell'altro, in modo ricorsivo.

Il passaggio verso nuovi stati avverrebbe in forza di quelle modificazioni dei caratteri derivanti dall'agire delle varie entità nell'ambiente il che, ammesso che le variazioni avvengano in modo da portare un grado di miglioramento delle prestazioni della specie, potrebbe configurare una ricerca di tipo euristico basata sulla individuazione di ottimi locali col semplice confronto fra il nuovo stato e quello precedente.

Nell'ipotesi di Darwin, basata sul concetto di selezione, sono riscontrabili le connotazioni di sistematicità di quella precedente e si fa uso dell'ereditarietà in modo analogo, ma l'euristica della ricerca risulta potenziata dal confronto, imposto dalla lotta per la sopravvivenza che permetterebbe solo ai migliori di prevalere: i possibili stati futuri verrebbero, implicitamente, confrontati fra loro prima di compiere il passo verso quello di essi che fosse risultato vincente, il paradigma risultante potrebbe essere quello della ricerca lungo il gradiente.

L'efficienza del sistema naturale di rappresentazione della conoscenza ed il forte potenziale, derivante dal parallelismo implicito del metodo evolutivo, hanno motivato il tentativo di imitare il procedimento naturale, dando luogo ad un nuovo filone di sviluppo per le tecniche di IA e per quelle di costruzione dei modelli.

Nelle simulazioni basate sul paradigma evolutivo, gli agenti operano in un ambiente che ha la funzione di promuovere la generazione di elementi portatori di caratteristiche nuove tramite un meccanismo di selezione, fondato sulla valutazione dei risultati collegabili al comportamento di ciascun individuo durante l'interazione con l'ambiente stesso. La possibilità che il comportamento dei singoli influisca sulle variabili ambientali promuove una forma di interazione fra i vari agenti, passante per il tramite dell'intermediazione dell'ambiente, indipendente dall'esistenza di possibilità di comunicazione diretta. L'idoneità evolutiva viene misurata sulla base del raggiungimento di propri obiettivi, autonomamente determinati e modificabili; in questo modo si promuove lo sviluppo di abilità diverse e variabili. Una misura di idoneità evolutiva, *fitness*, viene utilizzata per stabilire quali

individui debbano ricevere maggiori possibilità di riprodursi; i nuovi individui possono sostituire i genitori o altri individui scelti in modo casuale, oppure soggetti selezionati per l'estinzione in funzione della loro *fitness*. Nel primo caso il paradigma evolutivo utilizzato può essere collegato alle teorie di Lamarck; nel secondo, l'atteggiamento maggiormente selettivo rimanda alla lotta per la sopravvivenza teorizzata in Darwin. Le scelte riguardanti la destinazione degli individui vengono effettuate con metodi stocastici, onde garantire la generazione di movimento. La *fitness* assegnata ai nuovi nati viene, solitamente, calcolata come media di quella dei genitori, poiché le loro capacità derivano da un contributo di ambedue.

Gli agenti non conoscono il modello e le loro opinioni rispetto ai suoi parametri possono differire; essi constano di un genotipo, formato dai loro processi mentali, e di un fenotipo, consistente di tutti gli aspetti fisici dell'operato degli agenti; l'attributo fisico va inteso in senso lato, in quanto, in questa sede, trascende la corporeità dell'essere per estendersi ad aspetti legati al suo status sociale come il patrimonio o la posizione in una gerarchia. Nel processo di apprendimento avviene una variazione del genotipo tramite l'imitazione, eventualmente accompagnata dall'incrocio, e la mutazione. Le prime due operazioni hanno funzione di riprodurre la trasmissione ereditaria della conoscenza genetica che avviene nel normale processo di generazione di nuovi individui: negli ambienti simulati il genotipo dei nuovi individui viene ottenuto come copia di parti, incrocio, di quello dei genitori. La porzione dell'apporto di ciascun genitore viene determinata, di volta in volta, con criteri probabilistici, vincolati da appositi parametri, ed è ammesso che l'apporto di un singolo genitore sia nullo. La mutazione consta dell'alterazione di parti minime del patrimonio genetico del nuovo nato ed è utilizzata sia per simulare possibili errori di copia sia per favorire l'emergere, durante la sperimentazione, di tipi diversi.

L'azione del modello comporta l'acquisizione e l'organizzazione di nuove conoscenze: l'agente opera in un contesto ambientale in continua evoluzione, basandosi, perciò, su nuove inferenze, ed influenza l'ambiente in modi diversi che richiederanno un adeguamento anche degli altri agenti. Il numero di comportamenti che ciascun individuo è abilitato ad esprimere nell'ambiente può essere limitato, pur senza tema di introdurre staticità: ciò che determina le strategie è la loro combinazione e ristrutturazione. L'imitazione, realisticamente, dovrebbe basarsi sulle effettive possibilità di osservare o conoscere i comportamenti degli altri agenti; le strategie altrui potrebbero essere inferite dall'osservazione di comportamenti apparentemente slegati: in riferimento ad una attività industriale, l'acquisto di piccole quantità di materie prime specifiche potrebbe rivelare che una azienda sta sperimentando la fabbricazione di nuovi prodotti. La valutazione delle conseguenze derivanti dalle proprie azioni viene effettuata individualmente, mediante funzioni differenti per i vari agenti, rispettando il principio della autonomia nella percezione dell'ambiente.

Il metodo risulta fondamentalmente pluralistico, capace di esprimere un'elevata dinamicità ed indipendente dall'esistenza di una teleologia d'ambiente. L'accento viene posto sulla natura descrittiva e previsionale del pensiero degli operatori, veri o simulati, senza escludere le azioni deliberate e razionali dei medesimi, che potrebbero derivare dall'acquisizione di una conoscenza profonda dell'ambiente, pur situandole in ambiti di forte complessità ed incertezza. Risulta chiaro come i processi, ad esempio sociali ed economici, simulati con queste tecniche, verrebbero ad affinarsi per il tramite di un meccanismo progressivo, capace di operare anche in assenza di sistemi di valutazione forniti dall'esterno, riproducendo in modo più fedele la realtà. I mercati sono regolati da diverse leggi ed usi, ma il loro insieme non è inteso a formare un piano per incrementarne l'efficienza o perseguire obiettivi specifici, ruolo svolto, invece, da ciascuna norma per un particolare ambito di riferimento. L'emergenza del piano risulta solo dalla ricerca di una spiegazione globale, condotta per il tramite di una artificiosa osservazione dall'esterno, del funzionamento del sistema; gli esperimenti relativi allo studio dello scaturire di comportamenti complessi dall'interazione di un gran numero di unità semplici hanno dimostrato come l'effetto osservato possa, invece, risultare dalla concorrenza dei comportamenti elementari degli individui operanti nel sistema.

Il procedimento di evoluzione, adottato nel modello, può essere interpretato in base ad uno schema "mentale" o "individuale". Nel primo caso si considera il processo mentale di ogni singolo agente e l'evoluzione viene utilizzata per produrre quelle idee, rappresentate da genotipi, che risultino capaci di determinare i comportamenti maggiormente efficaci dell'agente nell'ambiente. Nello schema individuale, il procedimento evolutivo riguarda l'intero insieme di agenti che popolano il modello,

ciascuno rappresentato da un singolo genotipo. Nello schema mentale l'importanza dei processi di imitazione e comunicazione, nel determinare l'evoluzione, viene mediata dal grado di influenza che il comportamento di ciascun agente ha verso l'ambiente e dalle regole che presiedono alle comunicazioni esplicite fra agenti, direttamente o per il tramite dell'ambiente; con l'interpretazione individuale, invece, può avvenire uno scambio diretto di conoscenze genetiche fra due agenti, che incrementa notevolmente l'influenza reciproca nel procedimento evolutivo.

L'impostazione mentale si basa sul fatto che ogni individuo possa avere, nella sua mente e con riferimento anche ad uno stesso problema, diverse strategie di soluzione reputate praticabili; l'algoritmo evolutivo interviene sia nella formazione della scelta su quale applicare sia nella produzione di nuove strategie. La selezione viene effettuata assumendo come indicatore della bontà di ciascuna strategia il numero di copie presenti nell'insieme posseduto dall'agente e si basa su metodi non deterministici, onde creare il sufficiente movimento. La riproduzione semplice, quella ottenuta con la copia senza incrocio, corrisponde al processo che porta una idea ad affermarsi come migliore delle altre. L'incrocio consiste nell'inserimento di parti di strategie in altre e genera novità, in sinergia con la mutazione. In funzione dei risultati ottenuti con l'applicazione, può essere assegnato a ciascun genotipo un valore di *fitness,* da utilizzare nella individuazione delle strategie destinate a riprodursi o ad essere rimpiazzate.

L'impostazione mentale presenta caratteri di plausibilità quando si osservi come ciascun agente, reale, non sia in grado di stimare una strategia in termini assoluti, se non per problemi estremamente banali, ma solo di effettuare osservazioni comparative, relativamente alle conseguenze derivanti dalle diverse strategie sperimentate, che lo portino ad acquisire conoscenze sulla base delle quali effettuare le scelte. Meno realistica appare la possibilità di memorizzare parti di strategie da usarsi per comporne delle nuove, attività che risulterebbe implicita nell'incrocio di genotipi: la mente umana è, solitamente, più abile nel memorizzare strategie complete che non singoli elementi delle medesime. L'attribuzione di valori di *fitness* ai nuovi nati, potrebbe essere interpretata come una stima delle aspettative connesse all'attuazione della nuova strategia che verrebbero, in seguito, rivedute alla luce dei risultati conseguiti. La valutazione di ciascun individuo, per il tramite dei risultati ottenuti applicando la strategia di cui è portatore, permette di rimarcare l'esistenza di una chiara differenziazione fra genotipo e fenotipo: il primo incorpora l'aspetto sintattico della procedura ed il secondo quello semantico; in pratica l'algoritmo evolutivo è in grado di manipolare dati che non sa interpretare. L'algoritmo evolutivo non interagisce direttamente con l'ambiente: è l'agente che, in base a valutazioni proprie in ordine ai risultati, assegna i valori di *fitness* che divengono dipendenti dalla sua conoscenza dell'ambiente. Agenti diversi possono valutare in modo differente la stessa situazione e assegnare rilievi dissimili agli individui presenti nel proprio patrimonio genetico: l'evoluzione delle idee di ciascun agente percorre cammini peculiari e ciò aumenta il realismo della simulazione.

Dall'incapacità dell'algoritmo evolutivo di trattare da un punto di vista semantico le strategie che manipola, discende che, durante la riproduzione, potrebbero ottenersi individui aberranti, portatori di strategie assurde o impossibili: è d'uso comune predisporre accorgimenti volti ad evitare l'applicazione di queste strategie nell'ambiente, sebbene la presenza di comportamenti fortemente devianti potrebbe comunque presentare un elemento di maggior realismo della simulazione. Come in natura, l'estinzione degli individui di questo tipo potrebbe avvenire in conseguenza della loro scarsa adattabilità all'ambiente; in un ambiente simulato, però, occorre prevenire quelle azioni che potrebbero danneggiare la simulazione stessa e, quindi, eliminare quelle strategie che potessero implicare azioni tali da compromettere il funzionamento del programma di simulazione. Nell'impostazione mentale questa cernita potrebbe essere giustificata come attività di eliminazione delle idee non compatibili con i vincoli del problema; più difficile risulta la sua motivazione nel caso in cui ciascun genotipo sia considerato come un agente.

L'impostazione individuale è caratterizzata dalla difficoltà di distinzione fra genotipo e fenotipo: l'evoluzione opera direttamente sugli agenti rappresentati da un singolo genotipo. Problemi di plausibilità possono emergere specialmente quando gli agenti rappresentino entità inanimate: nel caso delle aziende l'evoluzione potrebbe essere vista unicamente come azione dell'ambiente che imporrebbe l'eliminazione degli agenti meno bravi e favore della crescita del numero dei migliori. In questo procedimento l'eliminazione delle aziende potrebbe essere ricondotta al fallimento, ma risulterebbe difficile motivare il fatto che le nuove aziende sarebbero portatrici delle migliori strategie

fra quelle esistenti sul mercato: la spiegazione basata sull'operato di forti processi imitativi non sarebbe sufficiente a motivare la mancata adozione di comportamenti simili da parte delle aziende fallite. Se gli agenti fossero intesi a riprodurre esseri animati, il loro comportamento dovrebbe, a rigore, rimanere lo stesso per tutta la loro vita, essendo plausibile uno scambio di informazioni a livello genetico solo al momento del concepimento, salvo che non si ammetta che ciascuno sia in grado di venire a conoscenza della valutazione delle capacità e dei pensieri degli altri in modo telepatico: il problema starebbe, in pratica, nel giustificare l'abilità dei singoli di correlare la strategia dell'evoluzione con le proprie decisioni.

La minor plausibilità dell'impostazione individuale non implica, necessariamente, l'ottenimento di risultati finali diversi o meno interessanti di quelli ottenibili con l'impostazione mentale; i modelli di impostazione individuale sono più semplici da realizzare e possono risultare più efficienti. La semplicità permette sia la riduzione del rischio di commettere errori nella costruzione del modello sia una maggiore comprensibilità dello stesso.

2.5 - Gli algoritmi evolutivi

I metodi di evoluzione, usati negli ambienti artificiali, agiscono, in modo iterativo, su una popolazione di individui con lo scopo di aumentare le prestazioni espresse dai singoli nella interazione con l'ambiente. La struttura degli individui ed il tipo di operazioni effettuate possono variare a seconda del tipo di algoritmo. La popolazione iniziale viene, solitamente, generata in modo casuale ed automatico dal programma e le operazioni compiute sui dati sono piuttosto semplici. Una misura del grado di adattabilità di ciascuna struttura permette di indirizzare le operazioni di ricambio generazionale: assegnando maggior possibilità di riproduzione agli individui migliori e riservando le più alte probabilità di estinzione alle strutture peggiori, gli algoritmi evolutivi sono in grado di esprimere una generale capacità di apprendimento basato sull'esperienza. Tutte le selezioni vengono operate con criteri non deterministici allo scopo di creare movimento, cioè di mantenere un certo grado di eterogeneità della popolazione tale da garantire una esplorazione sufficientemente accurata dello spazio di risoluzione del problema. In forza delle loro caratteristiche, gli algoritmi evolutivi permettono di evitare l'onere di prevedere e codificare a priori tutta una serie di dati relativi alla popolazione iniziale, agli obiettivi, anche intermedi, ed alla conoscenza dell'ambiente, presentando in ciò caratteristiche di elevata comodità di utilizzazione. I principali metodi di IA di tipo evolutivo sono: gli algoritmi genetici, i programmi genetici ed i sistemi classificatori (*classifier system*).

Gli algoritmi genetici hanno costituito uno dei primi esempi di utilizzazione del paradigma evolutivo nella ricerca delle soluzioni di problemi. Il metodo è basato sulla rappresentazione di ciascun genotipo come combinazione dei due valori zero ed uno, in numero sufficiente a contenere tutta la conoscenza necessaria a descrivere una qualche strategia. In termini pratici, data la scarsa capacità delle macchine per il calcolo automatico dell'epoca, questo metodo consentiva di sfruttare una rappresentazione "a *bit*", cioè basata sulla minima unità descrittiva utilizzata dagli elaboratori elettronici, capace di assumere unicamente i valori zero e uno: il genotipo constava di una sequenza, in termine tecnico "stringa", di *bit*. Ad ogni individuo è associato un valore rappresentativo della sua *fitness*, calcolato in base ad una funzione di misurazione del grado di adattamento del singolo all'ambiente. Durante ogni iterazione una certa quota della popolazione viene selezionata per la riproduzione in modo da privilegiare, in ciò, gli individui con valori di *fitness* più elevati; un'altra parte viene selezionata per l'estinzione. L'identità del numero di generazioni ed estinzioni non è strettamente richiesta, ma l'utilizzazione di popolazioni formate da un numero costante di individui presenta vantaggi per la redazione dei programmi. La riproduzione avviene mediante copia ed incrocio delle stringhe genitori, per simularne la compartecipazione al conferimento del patrimonio genetico dei discendenti. L'incrocio, *crossover*, si basa sulla individuazione, in modo casuale, di una posizione qualunque all'interno della stringa e sul successivo scambio delle parti a sinistra di quella. La mutazione consta della semplice inversione del valore di una o più posizioni prese a caso, cioè della sostituzione di uno zero con un uno o viceversa. La popolazione iniziale può essere generata casualmente, bastando, per iniziare il processo, che la *fitness* di almeno un individuo si differenzi da quella degli altri.

La convergenza di larghe quote della popolazione, ben oltre il novanta per cento, verso un unico tipo, vale a dire la presenza di una elevata percentuale di stringhe eguali, osservata dopo un congruo numero di riproduzioni, permette di affermare che la strategia incorporata in quella stringa risulti, se non la migliore, comunque "buona" per l'applicazione nell'ambiente dato. Nell'ambito della singola stringa non è possibile distinguere un genotipo ed un fenotipo.

I programmi genetici usano una procedura del tutto simile a quella degli algoritmi genetici, basata sugli stessi operatori di riproduzione, *crossover* e mutazione. Gli individui sono rappresentati da programmi, codificati con un linguaggio semplificato, in cui ogni unità deriva dall'unione di due elementi: un "operatore" ed un "terminale". Dominio di definizione dei possibili operatori, è, normalmente, l'insieme delle quattro operazioni aritmetiche di: somma, sottrazione, prodotto e divisione; i terminali possono essere i consueti valori zero ed uno. Utilizzando questo semplice linguaggio si perviene alla produzione di "alberi", che sottendono sequenze di semplici operazioni aritmetiche, dove terminali ed operatori rappresentano azioni a valere sull'ambiente. L'algoritmo agisce producendo, di volta in volta, nuovi programmi e, nuovamente, un valore di *fitness* funge da guida per il processo di evoluzione. Rispetto agli algoritmi genetici è qui possibile distinguere il genotipo, cioè il programma genetico vero e proprio, dal fenotipo rappresentato dal processo di esecuzione del programma stesso. Un altro vantaggio, nei confronti degli algoritmi genetici, può ravvisarsi nella maggiore scomponibilità dei programmi genetici e nella possibilità di rappresentare interi rami dell'albero con il risultato della esecuzione delle operazioni in esso contenute: un ramo del tipo "(+ (+ 1 1))" può essere riassunto in +2, una stringa di zero ed uno ha significato solo se accompagnata dall'indicazione delle specifiche posizioni e non può essere ulteriormente ridotta.

I *classifier system* utilizzano la rappresentazione in stringhe di zero ed uno di regole del tipo di quelle usate nei sistemi di produzioni: le informazioni provenienti dall'ambiente vengono tradotte nella stessa metrica per mezzo di componenti specializzate, *detector*, le emissioni verso l'ambiente sono correlate alle azioni da eseguire per mezzo di un secondo tipo di componenti, *effector*. Come nei sistemi di produzioni ogni regola è attivata dal *match* fra il suo schema, *condition part*, e l'indicazione delle condizioni ambientali ottenuta dai *detector*. Tutte le regole attivate concorrono, fra loro, per ottenere il diritto di immettere un messaggio, corrispondente alla descrizione della azione, *action part*, in una lista di messaggi che verrà letta dagli *effector* per decidere quale azione proporre nell'ambiente. La partecipazione all'asta comporta costi, così come la immissione di messaggi nella lista, che vanno ad incidere sull'ammontare del patrimonio di crediti di ciascuna regola; tale patrimonio viene rigenerato mediante la ricezione di una ricompensa, proporzionale ai risultati ottenuti dalle azioni proposte. Un algoritmo di scoperta di nuove regole viene fatto agire, di tanto in tanto, sull'insieme delle regole esistenti, per ottenere, con le consuete operazioni di riproduzione, *crossover* e mutazione, regole nuove; la selezione per la riproduzione o l'estinzione è basata sul valore del patrimonio di crediti di ciascuna regola.

Rispetto agli algoritmi genetici ed ai programmi genetici i *classifier system* producono strategie in modo maggiormente esplicito, consistenti nell'insieme di regole che vengono fatte evolvere. Il paradigma evolutivo contenuto nei *classifier system* potrebbe essere riferito all'intera specie, più che al singolo individuo; la capacità di adattamento del sistema dipende dallo sviluppo di individui, semplici e diversi fra loro, specializzati nell'affrontare singole situazioni. Sfruttando la divisione in specie è possibile evolvere contemporaneamente regole capaci di affrontare problemi diversi.

2.6 - Gli algoritmi genetici

Gli algoritmi genetici (Margarita 1992) stanno ricevendo una attenzione particolare da parte di studiosi delle più disparate discipline; questo interesse può imputarsi sia a fattori intrinseci al metodo stesso sia a motivazioni esterne. Fra i primi è possibile elencare: la possibilità di agire in situazioni di scarsa conoscenza della natura e struttura del problema da risolvere, la facilità di integrazione con tecniche di ottimizzazione tradizionali o la possibilità di sfruttare vantaggi derivanti dall'agire in

collaborazione con altri paradigmi dell'intelligenza artificiale come le reti neurali. Le motivazioni della seconda categoria riguardano il superamento delle tendenze basate sull'aspetto strettamente numerico dei problemi e la possibilità di evitare i limiti, insiti in altre tecniche di ottimizzazione, che possono costituire ostacolo all'utilizzazione in applicazioni non convenzionali. Come in Grefenstette (1991):

> (…) il crescente interesse verso gli algoritmi genetici può essere prevalentemente attribuito alla generalità del loro approccio. Gli algoritmi genetici possono essere utilizzati per problemi di ottimizzazione numerica e di ricerca combinatoria.

Oppure, in Goldberg (1989):

> (…) nuovi utenti si avvicinano agli algoritmi genetici per via del fascino di una metodologia basata sulle procedure naturali di ricerca e di scelta.

Per quanto attiene alle scienze economiche, (Margarita 1992) esistono punti specifici di interesse nelle caratteristiche "quasi ottimali", delle soluzioni offerte dagli algoritmi genetici, che risultano particolarmente coerenti con le teorie di razionalità limitata e con i modelli comportamentali degli agenti economici:

> (…) dotati di un sistema decisionale approssimativo ma rapido piuttosto che rigoroso ed esaustivo (…)

Il ricorso a processi genetici favorisce il ritorno ad una economia basata sull'uso di strumenti più naturali, dotata di maggiore affinità con l'economia qualitativa, in contrapposizione con l'eccessiva infiltrazione matematica che caratterizza le trattazioni quantitative.

Per quanto attiene alla struttura ed alla definizione degli algoritmi genetici si trova in Goldberg (1989):

> Gli algoritmi genetici sono algoritmi di ricerca basati sui meccanismi di selezione naturale e sulla genetica. Integrano un principio di sopravvivenza della struttura maggiormente adattata all'ambiente con un meccanismo di scambio di informazioni, strutturato ma stocastico, per formare un algoritmo di ricerca che presenta un po' del fiuto innovativo insito nella ricerca umana. Ad ogni generazione un nuovo insieme di creature artificiali (sequenze) è generato sulla base degli individui migliori dell'insieme precedente.

Negli studi economici è ricorrente il problema della ottimizzazione, normalmente inteso come ricerca del massimo valore di utilità per un insieme dato di argomenti che concorrerebbero a definire lo spazio delle possibili soluzioni. La risoluzione di un simile problema non presenta, solitamente, difficoltà qualora le leggi che vincolano le varie grandezze siano note ed esprimibili in una funzione che presenti caratteristiche, tipicamente di continuità e derivabilità, tali da permettere di applicare ad essa gli strumenti del computo differenziale. Complicazioni possono, invece, emergere quando le leggi suddette non siano perfettamente conosciute, ad esempio quando ci si trovi ad operare in situazioni di incertezza o a disporre di informazioni incomplete, oppure quando la funzione derivante risulti discontinua, situazione tipicamente riscontrabile nella realtà del processo di fissazione dei prezzi, o della pianificazione di produzioni ed acquisti caratterizzati dalla variazione delle quantità per lotti.

Quando si faccia ricorso agli algoritmi genetici (Holland 1975), la ricerca della soluzione di un problema, si caratterizza come il tentativo di esprimere una struttura, che rappresenti azioni in grado di portare il sistema ad interagire al meglio con l'ambiente in cui opera; in pratica l'obiettivo è stabilire quali organismi meglio si adattino ad un dato contesto. Quando i vincoli ambientali non siano noti od emergenti, l'osservazione attenta dell'evolversi delle strutture permette di trarre utili conclusioni sulle regole di funzionamento del sistema ambientale in oggetto. Immaginando di poter condurre sperimentazioni volte a misurare il grado di adattamento di ogni singolo individuo, è possibile definire "piano" la strategia di movimento all'interno dell'universo delle alternative possibili. Il più semplice piano è quello detto "enumerativo": tutte le possibili alternative vengono valutate in un ordine non influenzato dai risultati degli esperimenti già effettuati; detto procedimento è

sicuramente in grado di individuare la struttura migliore, memorizzando, ad ogni esperimento, quella con il grado di adattamento più alto. Il limite di applicabilità di una siffatta strategia sta nella cardinalità dell'insieme delle possibili alternative: per un sistema non particolarmente complesso, tale insieme potrebbe contenere circa 10^{100} diverse strutture, immaginando di poter effettuare 10^{12} esperimenti al secondo, la ricerca richiederebbe un tempo ben superiore all'età stimata dell'universo. Risulta evidente l'obbligo di garantire un'efficienza del procedimento, in termini di tempi di esecuzione, non disgiunta dalla salvaguardia dell'universalità dell'enumerazione. Simile obiettivo è materia della teoria riguardante i piani "adattivi": agenti in base ai risultati delle precedenti interazioni in modo da recepire gli elementi di conoscenza dell'ambiente che, grazie a quelle, fossero divenuti disponibili; essi devono ritenere i progressi fatti e usare le conoscenze acquisite per incrementare la quantità di strutture maggiormente "adattate" da sperimentare. La loro azione riguarda un insieme iniziale di entità che vengono progressivamente modificate, tramite appositi operatori, per generarne di nuove, capaci di meglio operare nell'ambiente di riferimento.

2.6.1 - Trattazione formale del metodo

Un sistema in fase di adattamento (Holland 1975) è caratterizzato dalla compresenza di operatori agenti su strutture ad ogni passo evolutivo; il piano determina quali strutture debbano crescere, cioè riprodursi, e quali scomparire, in funzione delle caratteristiche, fino a quel momento conosciute, del contesto ambientale, applicando tutte le sequenze possibili di operatori. In questo processo agiscono tre componenti principali: il contesto ambientale (E), il piano di adattamento (P) ed un insieme di sistemi di misurazione dei risultati, cioè della adattabilità delle strutture (M), naturalmente dipendente dal contesto. Non è richiesta una completa conoscenza delle caratteristiche dell'ambiente essendo sufficiente la cognizione, inerente natura e comportamento, di quelle che potrebbero determinare delle alternative nello sviluppo del piano. Ciascuna combinazione di caratteristiche ambientali diverse origina un diverso contesto, così che l'iniziale incertezza, riguardante le conoscenze, può essere formalizzata designando una classe ε di possibili contesti E; il dominio di azione del piano risulta allora formato dalle azioni attinenti ai diversi contesti, raggruppate in insiemi del tipo di A, tutti appartenenti alla classe α degli insiemi di azioni possibili; gli insiemi M dei sistemi di misurazione, ciascuno relativo ad uno specifico contesto (E), formano anch'essi una la classe, μ. Il piano di adattamento modificando progressivamente le strutture, per il mezzo degli operatori presenti nell'insieme O, determina le risposte al contesto cosicché: l'insieme O degli operatori ed il dominio A del piano concorrono a delineare le sue opzioni, mentre l'obiettivo finale del piano è quello di produrre strutture che operino correttamente nel contesto E. Poste queste premesse risulta agevole identificare le principali difficoltà che potrebbero affliggere l'applicazione di simili procedure, cioè: l'eccessiva ampiezza dell'insieme A delle azioni possibili, la particolare complessità delle strutture in A, implicante difficoltà nel determinare a quali di esse fossero attribuibili i migliori risultati, le peculiarità delle funzioni in M, che potrebbero presentare ottimi locali, discontinuità e comportamenti non lineari, la variabilità dei valori ottenuti con la misurazione dei risultati nel tempo e nello spazio, l'eccessiva verbosità di quei contesti che presentassero, per ogni passo del piano P, una grande quantità di informazioni da filtrare e valutare.

La ricerca in A dovrebbe proseguire fino a quando fossero possibili miglioramenti significativi, provando ed incorporando le strutture che fornissero i migliori risultati; il conseguimento di un punto di convergenza si manifesta nell'emergere di un elevato grado di omogeneità, all'interno della popolazione di strutture, tale da far concludere che il processo di selezione abbia ampiamente agito, lasciando in vita solo gli individui veramente capaci di operare correttamente nel contesto. La soluzione ottenuta non è, necessariamente, quella ottima, ma comunque garantisce un livello di prestazioni superiore a quello raggiungibile utilizzando la popolazione iniziale di cromosomi, in diretta dipendenza dalla corretta applicazione del processo.

In questo scenario il dominio di azione del piano P risulta estendersi all'intera collezione delle possibili azioni, rappresentata, al livello massimo di astrazione, da un insieme (A) arbitrario, non vuoto, di cromosomi, cioè strutture, formati, ciascuno, da m geni; ognuno di questi ultimi presenta, a sua volta, una collezione (V) di k valori denominati alleli (v): in pratica l'i-esimo gene presenta una

collezione $V_i = \{v_{i1} , \dots , v_{ik}\}$ e l'insieme dei cromosomi può essere visto come l'insieme delle possibili combinazioni di alleli:

$$(1) \qquad A = V_1 \, X \, V_2 \, X \dots X \, V_m = \prod_{i=1}^{m} V_i \, .$$

Il piano P produce una sequenza di strutture, cioè una traiettoria attraverso A, operando successive selezioni, in base agli operatori contenuti nell'insieme O, avvalendosi delle informazioni ottenute dal contesto E. P, tipicamente, genererebbe differenti traiettorie quando applicato ad ambienti diversi, pur se la capacità di distinguere contesti dissimili risulta limitata dall'intervallo (I) di stimoli e segnali che possono essere ricevuti ed elaborati. Lo stato del piano al tempo t è rappresentato dall'insieme A al tempo t, cioè A(t) ma, per raffigurare anche le informazioni acquisite, occorre pensare A(t) come formato dalle strutture presenti al tempo t (A'(t)) e dalla "memoria" S(t) delle conoscenze acquisite; quindi P risulta dall'applicazione degli insiemi I (degli *input*) ed A su A:

$$(2) \qquad P : I \, X \, A \rightarrow A \, .$$

La struttura da sottoporre a verificazione al tempo t+1 diviene:

$$(3) \qquad (A'(t+1),S(t+1)) = A(t+1) = P(I(t),A(t)) = P(I(t),(A'(t),S(t))) \, .$$

e la forma iniziale, espressa nella (2), può essere interpretata, senza perdita di generalità, come:

$$(4) \qquad P : I \, X \, (A' \, X \, S) \rightarrow (A' \, X \, S) \, .$$

Utilizzando un metodo stocastico, per determinare una struttura A(t+1) partendo da I(t) e A(t), P assegna diverse probabilità alle strutture di un insieme per selezionare A(t+1) fra esse: data la possibilità di ottenere da I(t) X A(t) diverse strutture $A'_1, A'_2, \dots , A'_j \dots$ la A'_j sarà selezionata con probabilità d'_j ;posto D quale insieme rappresentativo della possibile distribuzione di probabilità su A si ottiene:

$$(5) \qquad P : I \, X \, A \rightarrow D \, .$$

La (5) non basta, da sola, a rappresentare il procedimento di selezione da A degli elementi di A(t+1), occorre considerare anche l'applicazione degli operatori appartenenti all'insieme O; le precise operazioni effettuate in un generico passo P' risultano determinate dalla funzione:

$$(6) \qquad P' : I \, X \, A \rightarrow O.$$

Il processo di verificazione, e successivo aggiustamento, delle strutture prosegue finché vengono rilevati miglioramenti nel livello dei risultati ottenuti; per salvaguardare i livelli già raggiunti devono essere provate ed incorporate, nei nuovi cromosomi, quelle proprietà strutturali che risultino associabili ai più alti livelli di adattamento.

Diviene, in quest'ottica, basilare dotarsi di un metodo per confrontare strutture ed identificare le parti simili in esse, cioè riconoscere dei sottoinsiemi in A che abbiano attributi comuni. Ogni struttura $a \in A$ può essere descritta dal proprio insieme ordinato di m attributi (a_i) e possiamo dire che, definito $c' = \{n \in N : 0 < n < m + 1\}$:

$$(7) \qquad a' \text{ e' uguale ad } a'' \text{ se } \forall \, i \in c' \rightarrow a'_i = a''_i \, .$$

La similitudine di due strutture può essere vista in funzione del numero di attributi corrispondenti: dato un criterio di indifferenza per il valore di particolari attributi e definito c" come l'insieme dei numeri naturali che indicano le posizioni non indifferenti, c" sarà sottoinsieme in senso stretto di c', quindi:

$$(8) \qquad a' \text{ e' simile ad } a'' \text{ se } \forall \, i \in c'' \rightarrow a'_i = a''_i \text{ , dove } c'' \subset c'.$$

Data la convenzione per la quale il valore "#" indica indifferenza, è agevole capire come la scrittura (v_j,#,#) designi il sottoinsieme di un insieme A, con strutture lunghe 3 posizioni, formato da tutte le strutture che abbiano il valore v_j in prima posizione. Ogni struttura definita su $\{V_i \cup \{\#\}\}$ è detta "schema"; essa e' caratterizzata da due importanti attributi: l'ordine e la lunghezza definitoria. Il primo attributo è il valore espresso dal numero di posizioni fisse presenti nella struttura stessa, cioè di valori diversi da #; il secondo è il valore della distanza tra la prima e l'ultima posizione fissa, misurata in termini di posizioni (Lo schema "af#4##", ad esempio, ha ordine 3 e lunghezza 4-1=3). L'insieme dei possibili schemi ottenibili utilizzando combinazioni di caratteri di indifferenza e valori è dato da:

$$(9) \qquad R = \prod_{i=1}^{m} \{V_i \cup \{\#\}\} .$$

2.6.2 - Aspetti pratici e schemi

Il funzionamento del processo, si basa sulla codificazione dei parametri del problema in strutture: nelle applicazioni che si avvalgono dell'uso di un elaboratore ciò può essere facilmente ottenuto utilizzando stringhe di caratteri, lunghe a piacere, solitamente definite su un alfabeto binario formato dai caratteri "0" e "1"; l'insieme di queste stringhe costituirà la popolazione iniziale dell'algoritmo genetico, mentre le singole stringhe ne saranno gli individui. Motivazioni di semplicità nella scrittura dei programmi consigliano di operare con popolazioni formate da stringhe della stessa lunghezza ed in numero fisso, selezionando, cioè, ad ogni ciclo, un numero di individui per l'estinzione, pari a quello dei nuovi nati. La generazione iniziale è, normalmente, condotta in modo casuale a meno che non si voglia partire da una popolazione già selezionata in base ad altre elaborazioni.

Ad ogni ciclo ciascun individuo viene valutato in termini di idoneità ad operare nell'ambiente, *fitness*, calcolata in base ai valori assunti dalla funzione di misurazione dei risultati; tale valore influenza la probabilità assegnata a ciascuna stringa di essere selezionata per la riproduzione e la possibilità di estinzione di ciascuno. La riproduzione viene effettuata copiando, una o più volte, una stringa genitore, selezionata in modo da garantire che gli individui migliori abbiano maggior probabilità di ricevere un numero elevato di copie; a fronte delle nuove nascite alcune stringhe devono scomparire, in dipendenza della tecnica di gestione della popolazione adottata (numerosità costante o variabile). Gli individui da eliminare possono essere scelti, casualmente, fra tutti quelli della vecchia generazione o fra quelli appartenenti al sottoinsieme dei genitori, concretizzando, quindi, una evoluzione di tipo maggiormente lamarckiano; in alternativa, la selezione può avvenire, come d'uso, con metodo stocastico basato sull'assegnazione di elevate probabilità di estinzione alle stringhe con più bassi valori di *fitness*, secondo una impostazione più tipicamente darwiniana. Il ricambio generazionale può riguardare l'intera popolazione, tutti gli individui vengono rimpiazzati ad ogni ciclo (*generational replacement*), oppure essere limitata ad alcuni individui (*steady-state reproduction*).

Con la riproduzione si attua solo una scelta degli individui migliori all'interno della popolazione iniziale; per scoprire nuove strutture è necessario avvalersi degli appositi operatori appartenenti all'insieme O. Ciascuno di quelli realizza una manipolazione delle stringhe in base a metodi propri; e' possibile immaginare un numero pressoché infinito di operatori ma i due basilari sono il crossover e la mutazione. Il crossover è articolato in due fasi: si scelgono, casualmente, due individui della nuova generazione ed una posizione all'interno della stringa (un numero naturale casuale fra 1 ed m-1), le parti a sinistra della posizione scelta delle due stringhe vengono scambiate tra loro, in modo da ottenere due individui diversi da quelli di partenza. Naturalmente è possibile immaginare crossover più complicati: scegliendo due posizioni si possono scambiare le porzioni di stringhe dislocate agli estremi o la porzione centrale; con 3 si possono incrociare le porzioni dispari o le pari ecc. Quando il crossover agisce in collaborazione con la riproduzione, partendo da due genitori che fondono il loro patrimonio genetico contribuendo ciascuno a quello dei nuovi nati, si parla di riproduzione "sessuata". La mutazione simula gli errori nella trasmissione del patrimonio genetico, la si attua modificando il valore di una posizione, scelta, casualmente, in base ad una probabilità di applicazione dell'operazione molto bassa; essa può contribuire all'esplorazione dello spazio delle

soluzioni, prevenendo le possibilità di perdita definitiva di una informazione che fosse contenuta in tipi di individui ormai definitivamente estinti.

La convergenza della popolazione su un certo schema rivela, solitamente, il raggiungimento di una configurazione di strutture che, pur potendo non essere quella ottimale, sicuramente risulterebbe migliore di quella di partenza. In dipendenza di questa osservazione l'azione dell'algoritmo viene, normalmente, interrotta al raggiungimento di un grado stabilito, piuttosto elevato, di similitudine o omogeneità della popolazione; l'interpretazione del genotipo prevalente fornisce la soluzione ricercata del problema. Può comunque essere utile osservare la popolazione in stadi diversi del processo onde evincere quale sia l'operato del meccanismo di selezione; dal procedere di esso è facile ottenere informazioni sul funzionamento dell'ambiente del tutto nuove rispetto a quelle utilizzate, e quindi conosciute, relative alla misurazione della *fitness*. In questo senso, particolarmente, il metodo presenta interessanti possibilità di utilizzazione per le indagini relative a quegli ambiti dove le connotazioni di complessità, del mondo reale, comportino commistioni fra diversi fenomeni rendendone difficoltoso un loro isolamento a priori.

Il flusso di trasmissione della conoscenza che si attua nella riproduzione rimanda al concetto di schema, la cui utilizzazione, oltre ad agevolare i confronti fra stringhe volti a stabilirne il grado di similitudine, permette di fornire un robusto supporto metodologico nella interpretazione delle medesime. Uno schema può essere inteso come generalizzazione di un insieme di possibili combinazioni, ottenuta per il tramite dell'utilizzo di un simbolo di indifferenza: esso può rappresentare tutte le stringhe che risultino avere, nelle rispettive posizioni, valori uguali a quelli dello schema, limitatamente a quelli diversi da # (Lo schema "#01#1101" rappresenta l'insieme delle stringhe: "10101101", "10111101", 00101101" e "00111101") . Dal punto di vista semantico uno schema risulta formato dall'insieme dei comportamenti codificati nell'ambito delle stringhe da esso rappresentate. Data una popolazione formata da individui di lunghezza m definiti su di un alfabeto di k simboli, risultano ottenibili $(k + 1)^m$ schemi, mentre da ogni singola stringa possono ricavarsi 2^m schemi, quindi una popolazione di N individui potrà contenere un numero compreso fra 2^m e $N * 2^m$ schemi, a seconda del grado di omogeneità della popolazione stessa.

Un particolare problema riguarda gli effetti degli operatori genetici sugli schemi: la riproduzione tende a fare aumentare le similarità riducendo il numero di schemi; il crossover può distruggere uno schema quando il punto di sezione della stringa cada all'interno di esso: schemi corti ed incorporati in stringhe con forte probabilità di essere selezionate per la riproduzione avranno maggior possibilità di sopravvivere. Immaginando che esistano, ad un dato momento t, n esempi di un certo schema h all'interno di una popolazione A'(t), formata da q individui, e ricordando che la stringa j-esima viene selezionata per la riproduzione con probabilità $d'_j = M(j) / \sum M(i)$, dove M rappresenta l'insieme delle funzioni di misurazione della *fitness*, data una popolazione a generazioni non sovrapponibili, formata, come detto, da q individui, si ottiene che il numero di schemi h al tempo t+1 risulterà da:

$$(10) \qquad n(h,t+1) = n(h,t) * q * M'(h) / \sum M(i) \ .$$

dove M'(h) è la media dei valori delle funzioni obiettivo delle stringhe contenenti lo schema h al tempo t; ponendo $M' = \sum M(i) / q$, cioè pari alla media dei valori delle funzioni obiettivo dell'intera popolazione , è possibile riscrivere la (10) come:

$$(11) \qquad n(h,t+1) = n(h,t) * M'(h) / M' \ .$$

E' palese che gli schemi con prestazioni oltre la media cresceranno di numero, mentre quelli con capacità inferiori alla media tenderanno a sparire; supponendo che uno schema h, preso a piacere, esibisca un valore della funzione di misurazione superiore alla media di una proporzione costante c, si avrà:

$$(12) \qquad n(h,t+1) = n(h,t) * [M' + c * M'] / M' \text{ cioè } n(h,t+1) = n(h,t) * (1 + c) \ .$$

Data la formulazione precedente è facile concludere che l'effetto della riproduzione sugli schemi consta del permettere una crescita esponenziale di quelli che presentino valori della funzione obiettivo superiori alla media.

Il *crossover* tende a distruggere gli schemi più lunghi ed il punto di crossover è scelto in modo uniformemente casuale: risulta possibile definire la probabilità che uno schema sia distrutto come rapporto fra la sua lunghezza definitoria (z) e la lunghezza totale della stringa diminuita di una unità' (non ha senso fissare il punto di crossover a fine stringa). Il crossover avviene, in base ad una distribuzione di probabilità; definita d_c la probabilità che lo schema sia sottoposto a crossover, si otterrà una probabilità di sopravvivenza del medesimo (d_s):

$$(13) \qquad d_s \geq 1 - d_c * [z(h) / (m - 1)] \ .$$

sommando gli effetti di riproduzione e crossover, supposte indipendenti le probabilità che le due azioni agiscano sullo stesso schema, si ottiene:

$$(14) \qquad n(h,t+1) = n(h,t) * [M'(h) / M'] * \{1 - d_c * [z(h) / (m - 1)] \} \ .$$

Definendo d_m la probabilità di mutazione di un singolo allele, se ne può dedurre che uno schema sopravviverà quando ognuna delle o(h) posizioni fisse al suo interno sopravviveranno e, quindi, la probabilità di sopravvivenza ad una mutazione sarà pari a $(1 - d_m)^{o(h)}$; ricordando, però, che d_m ha, solitamente, un valore tendente a 0^+, si giungerà a:

$$(15) \qquad (1 - d_m)^{o(h)} \approx 1 - o(h) * d_m \ .$$

In conclusione risulta possibile affermare che uno schema h può ricevere in ciascuna generazione successiva, data l'applicazione degli operatori riproduzione, crossover a punto singolo e mutazione, un numero di copie pari a:

$$(16) \qquad n(h,t+1) = n(h,t) * [M'(h) / M'] * \{ 1 - d_c * [z(h) / (m -1)] - o(h) * d_m \} \ .$$

Questa espressione prende il nome di "equazione fondamentale del teorema dello schema" e significa che gli schemi con risultati oltre la media, lunghezza definitoria piccola e di basso ordine, tendono a crescere esponenzialmente di numero nel corso dell'applicazione dell'algoritmo genetico.

2.6.3 - Operatori speciali ed estensioni del metodo

Il funzionamento degli algoritmi genetici è basato sulla riproduzione, sul crossover e sulla mutazione ma sono osservabili, anche in natura, operazioni più complesse, la cui azione non sempre risulta facilmente comprensibile, per le quali è stata tentata una riproduzione artificiale. In particolare il riferimento va verso: la dominanza in strutture diploidi, la segregazione e trasposizione in strutture con più cromosomi, la duplicazione e cancellazione di un gene, la differenziazione sessuale delle strutture, il formarsi di nicchie e la specializzazione, l'inversione, e le forme di restrizione delle possibilità di accoppiamento.

La natura fornisce diversi esempi di strutture diploidi, cioè dotate di due cromosomi contenenti informazioni per la stessa funzione; quando queste informazioni risultano mutuamente esclusive il meccanismo della dominanza prevede che un allele, dominante, prevalga sul suo corrispettivo, recessivo, appartenente all'altro cromosoma. Una struttura genetica simile può ricordare alleli o combinazioni di alleli che hanno dato buoni risultati in precedenza, preservandoli dall'estinzione in una situazione ostile. Naturalmente anche i rapporti di dominanza si modificano in accordo con l'evoluzione genetica. Una buona rappresentazione artificiale della dominanza è data nello schema a tre alleli (Hollstien 1971) basato sull'alfabeto {0, 1, 2} e su due semplici regole: il 2, se selezionato, è interpretato come un 1; lo zero domina l'uno. (Ad esempio date le due stringe: a = "012012012" e a' = "000111222" il conflitto sarà risolto con a* = "001011111").

Può essere utile valersi di strutture formate da più cromosomi (Holland 1975), rappresentabili come liste di k coppie di stringhe, accompagnate dall'utilizzo di due speciali operatori: segregazione, che implementa il processo di scelta casuale di uno dei diversi cromosomi, e trasposizione, che potrebbe essere vista come un particolare *crossover*. Duplicazione e cancellazione di singoli geni possono rendere adattivo il meccanismo di mutazione: se il numero di copie di un gene aumenta a causa della duplicazione anche la probabilità di mutazione di quel gene, o meglio la probabilità che una copia di quel gene sia mutata, aumenta; in senso inverso opera la cancellazione.

La discriminazione sessuale, in un algoritmo genetico, essenzialmente, attua una bipartizione della popolazione, incoraggiando, sotto determinate condizioni, la collaborazione fra elementi di sesso diverso.

Una nicchia può essere intesa come un compito specifico di un organismo ed una specie come una classe di organismi con caratteristiche comuni; la distribuzione dei ritorni dall'ambiente in modo proporzionale all'apporto di ciascuno aiuta la formazione di gruppi specializzati: ogni individuo ha convenienza a cambiare gruppo se il ritorno ambientale di un'altra azione, diviso per il numero degli individui che la consigliano, supera l'eguale rapporto relativo al suo attuale gruppo di appartenenza. In questo modo si formano aggregati la cui numerosità rappresenta la bontà dell'azione in cui sono specializzati gli individui appartenenti al gruppo stesso. La preselezione (Cavicchio 1970) può essere utile per promuovere la formazione di nicchie: ogni nuovo individuo rimpiazza il genitore se la sua *fitness* risulta maggiore e la diversità è mantenuta poiché ogni individuo tende a rimpiazzarne altri a lui simili. Una generalizzazione del metodo (De Jong 1975) può essere ottenuta sostituendo stringhe simili indipendentemente dal rapporto genitore figlio: nello schema *crowding*, un individuo è confrontato con una popolazione casuale formata da un numero di altri dipendente dal *crowding factor*; questo metodo salvaguarda la diversità e riserva posto per due o più specie nella popolazione. Per stabilire l'appartenenza ad una specie può essere utilizzato un *external schemata* (Perry 1984), cioè una maschera di confronto.

L'operazione di inversione si basa sulla scelta di due interi nell'intervallo chiuso [0,m], la struttura viene, quindi, tagliata nei punti detti e gli estremi della porzione centrale della stringa vengono scambiati di posto, infine la stringa viene nuovamente ricomposta. In natura una simile operazione non altera il significato degli alleli scambiati, ma nella rappresentazione degli algoritmi genetici l'interpretazione di ciascun gene si basa sulla sua posizione: per poter utilizzare l'inversione occorre adottare una rappresentazione estesa del cromosoma, basata su due stringhe di cui una conservi memoria della posizione originale dei singoli geni e la seconda riporti i rispettivi alleli. L'inversione permette di codificare diversamente un messaggio e può essere d'aiuto nel preservare determinati schemi dalla distruzione; il suo utilizzo implica l'adozione di particolari regole per il *crossover,* che potrà essere permesso solo fra stringhe omologhe o dovrà avvenire con particolari tecniche (Goldberg 1989) di: *partially matched crossover*, *order crossover* o *cycle crossover.*

L'introduzione di limitazioni alla possibilità di accoppiamento degli individui (Hollstien 1971) richiede che gli individui di una famiglia si incrocino fra loro finché la *fitness* totale della famiglia cresce; esistono diversi criteri per determinare le similitudini fra individui richieste per l'accoppiamento, tutti imperniati su confronti in base ad una maschera facente parte di una stringa ampliata. Esistono, naturalmente, anche diversi criteri di accoppiamento: *bidirectional match*, *unidirectional match*, *best partial match*. Il meccanismo permette di introdurre un comportamento adattivo, non più casuale, nelle preferenze di accoppiamento degli individui che aiuta a migliorare le capacità dei nuovi nati.

Ulteriori possibilità di arricchimento del metodo possono essere reperite negli studi condotti sulle ricerche di ottimizzazione basate su più obiettivi, sulla ibridazione degli algoritmi genetici e sulla conduzione in parallelo del procedimento.

Quando il criterio di ottimizzazione si basa su più obiettivi, occorre modificare la condizione di adattamento ottimale, fino ad ora utilizzata, rappresentata dalla massimizzazione della funzione di misurazione della *fitness* di ciascun individuo; diverse funzioni devono essere massimizzate e quindi i risultati devono essere confrontati con criteri complessi. Una delle prime proposte (Schaffer 1984) fu

il VEGA (*Vector Evaluated Genetich Algorithm*) basato sull'utilizzazione di diverse partizioni della popolazione globale, egualmente numerose, in cui selezionare i migliori individui per ciascun criterio, onde disporre di una popolazione “abile” per ulteriori indagini.

A proposito dell'ibridazione, non bisogna dimenticare che molti organismi, specie quelli umani, formulano le loro decisioni sfruttando conoscenze già acquisite: una possibilità in tal senso risulterebbe da meccanismi di algoritmi genetici ibridi, dove l'algoritmo individua i "picchi" e le singole conoscenze forniscono i meccanismi di calcolo per stabilire come giungere "in vetta". Un metodo di questo tipo (Goldberg 1989) è il *G-bit improvement* (*gradientlike-bitwise improvement*) basato sui tre passi seguenti: selezione di uno o più fra i migliori elementi della popolazione, scansione, bit per bit, di ciascuna stringa operando modifiche di singoli bit e conservando la migliore delle due alternative, introduzione nella popolazione delle stringhe così ottenute. Una seconda tecnica prende il nome di *knowledge-Augmented Operators* e si avvale della possibilità di influire sulle operazioni di mutazione, o anche sulle altre, utilizzando informazioni diverse da quelle risultanti dalla funzione di misurazione dei risultati. Un terzo metodo prevede l'impiego di funzioni approssimative dell'ottimo per ridurre i tempi del processo di evoluzione.

Gli algoritmi genetici ben si prestano ad implementazioni basate su processi paralleli, (Grefenstette 1981) tipizzabili come:

- *Synchronous master-slave:* in cui un *master* coordina k processi schiavi controllando selezioni, accoppiamenti ed evoluzione genetica, mentre gli schiavi provvedono solo alla valutazione della funzione; questo algoritmo risulta piuttosto oneroso quando le valutazioni possano cambiare e non e' perfettamente "portabile".
- *Semi-synchronous master-slave:* dove la popolazione e' più dinamica.
- *Asynchronous, concurrent Genetic Algorithm:* dove k processi agiscono in modo indipendente, attuando tutte le operazioni, su dati condivisi.
- *Network prototype:* dove k processi agiscono indipendentemente su dati privati ma si notificano le informazioni, relative ai migliori individui di volta in volta scoperti.

Un'idea di parallelismo fondato sulla simulazione (Goldberg 1989) è contenuta nel *Communitiy model:* ciascuna famiglia evolve nella propria casa ma i figli vengono inviati in un luogo centrale dove incontrano altri individui; le coppie che si formano si rivolgono ad un agente per trovare una casa e competono per ottenerla, in modo da poter riavviare il processo di riproduzione. Se la città è troppo affollata le coppie possono emigrare in altre comunità.

2.6.4 - Vantaggi

L'analogia con i metodi tradizionali di ricerca, pur se affievolita, compare, nel ruolo della valutazione delle singole stringhe, a configurare un tipo di euristica basato sulla ricerca secondo il gradiente e nella forte creazione di movimento derivante dall'utilizzazione di metodi stocastici di selezione. Le differenze, rispetto a quelle tecniche, intervengono, invece, a definire gli elementi di novità e peculiarità del metodo (Goldberg 1989):

- Un algoritmo genetico lavora su una codifica dei parametri e non sui parametri stessi, ciò lo rende maggiormente flessibile e “portabile”.
- La ricerca non muove per singoli punti ma coinvolge l’intero insieme di punti rappresentato dalla popolazione, riducendo notevolmente la possibilità di incappare in ottimi locali determinati da "falsi picchi".
- Viene utilizzata direttamente la funzione obiettivo non sue trasformazioni come derivate e similari, risulta quindi notevolmente ampliato, rispetto ad altri metodi, l’insieme delle funzioni utilizzabili.
- La transizione da una popolazione all’altra avviene in base a regole probabilistiche e non deterministiche.

La astrattezza, derivante dall'operare sulla codificazione dei parametri, contribuisce a rendere applicabili gli algoritmi genetici in ambiti fortemente differenziati e consente la scrittura di programmi

generalizzati, utilizzabili per la conduzione di un gran numero di esperimenti. I programmi possono essere redatti con gli usuali linguaggi la cui conoscenza è, solitamente, più diffusa: ciò facilita la comprensione dei medesimi e delle simulazioni basate sul loro utilizzo.

Il parallelismo intrinseco del metodo (Holland 1975) deriva dalla potenza del concetto di schema: la promozione di ciascuna stringa equivale a circoscrivere una considerevole area di interesse, costituita da tutte le soluzioni affini; ogni stringa risulta appartenere a tutte quelle regioni dello spazio delle soluzioni in cui compaia almeno uno dei suoi componenti. Procedendo in base alla valutazione della *fitness*, l'algoritmo deduce buone combinazioni da buone soluzioni, evitando una esplorazione troppo concentrata su particolari aspetti della soluzione, e garantisce un buon rapporto fra esplorazione dello spazio e sfruttamento delle conoscenze ottenute.

Il riferimento ai risultati, ottenuti dalla applicazione delle strategie proposte, permette una correlazione diretta con l'ambiente e consente di adattare le misurazioni ai suoi mutamenti senza gravare il metodo di pesanti costrutti matematici, volti a recepire quelle modificazioni come ulteriori parametri.

L'adozione di criteri stocastici garantisce la creazione di movimento ed una esplorazione sufficientemente vasta dello spazio delle soluzioni possibili, anche quando le dimensioni di questo non siano note.

2.7 - I *classifier system*

Il metodo può connotarsi come applicazione degli algoritmi genetici al problema di conferire capacità di apprendimento ai tradizionali sistemi esperti. La notevole complessità sintattica delle regole, utilizzate da questi ultimi, rende difficoltosa la predisposizione degli algoritmi di trattazione necessari per la generazione di nuovi costrutti. Con la codificazione tipica degli algoritmi genetici, invece, le operazioni risultano notevolmente semplificate ed è possibile sfruttare i vantaggi dei metodi evolutivi. La definizione della tecnica è reperibile in Goldberg (1989):

> *A classifier system is a machine learning system that learns syntactically simple string rules (called classifiers) to guide its performance in an arbitrary environment*.

I sistemi di produzioni sono, già di per sé, in grado di superare la propria massa di conoscenze tramite l'elaborazione di strategie: la diversa sequenza di applicazione delle regole, fornite dall'esterno, permette al sistema di modificare il proprio comportamento e di affrontare situazioni non preventivamente codificate. L'utilizzazione in situazioni che richiedano elevata capacità di apprendimento è comunque poco consigliata, data la necessità di utilizzare costrutti grammaticali complessi la cui trattazione appesantisce notevolmente l'elaborazione. I *classifier system*, adottando strutture sintattiche elementari, avulse dal significato semantico delle regole, consentono di operare su di esse tramite un algoritmo genetico; ciò agevola considerevolmente la produzione di regole nuove a partire da quelle esistenti sia riducendo, praticamente annullando, i rischi di ottenere strutture prive di significato o aberranti, sia rendendo le operazioni estremamente semplici e veloci.

Un altro vantaggio deriva dal metodo di selezione delle regole: nei sistemi esperti la scelta della regola da applicare viene fatta ad ogni passo, ogni scelta vincola lo sviluppo della ricerca; nei *classifier system* la selezione viene posposta dopo la verificazione di tutti i *match,* così viene incrementato il parallelismo del metodo e resa possibile la gestione di problemi diversi nell'ambito di una medesima applicazione.

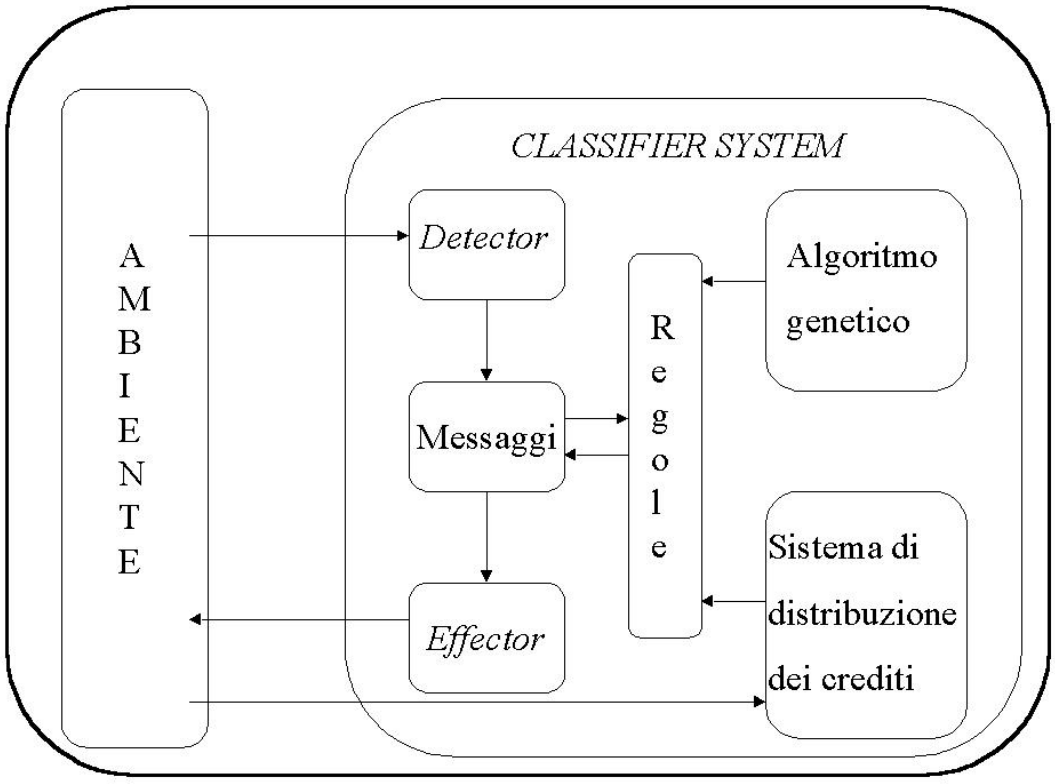
Il valore di ciascuna regola di un sistema di produzioni viene assegnato in modo fisso dall'esperto che si occupa della sua istruzione iniziale; nei sistemi evolutivi, al contrario, l'originale valore viene modificato in proporzione ai risultati ottenuti dalla regola, divenendo oggetto di apprendimento. L'importanza delle varie regole risente, in questo modo, della loro idoneità ad operare nell'ambiente e può modificarsi in seguito a variazioni intervenute nel contesto di applicazione; lo stesso valore viene usato dall'algoritmo genetico per decidere quali regole dovranno riprodursi e quali

scomparire. La gestione del valore delle regole costituisce l'elemento più importante di un *classifier system;* l'algoritmo che presiede a questa attività può assumere connotazioni estremamente variate, ma quello normalmente utilizzato è il *bucket brigade algorithm* (Holland 1975) basato sulla competizione, dei singoli *classifier,* per ottenere il diritto di inserire il proprio messaggio nell'insieme relativo, e su un meccanismo di ripartizione dei valori tra le varie regole che concorrono alla decisione finale.

2.7.1 - Struttura e funzionamento di un *classifier system*

In un *classifier system* è dato di individuare tre componenti principali: un insieme di regole e messaggi, un sistema di attribuzione dei crediti ed un algoritmo genetico. Due componenti aggiuntivi, *detector* ed *effector,* assicurano le comunicazioni con l'ambiente: i primi trasformano i dati provenienti dal contesto in stringhe simili alle regole, gli altri traducono i messaggi emanati dalle regole in azioni per l'ambiente. I messaggi sono stringhe di caratteri codificati su un qualunque alfabeto, normalmente {0,1}. Le regole sono stringhe formate da una parte di condizione e da una di descrizione dell'azione; ciascuna regola possiede un patrimonio di crediti che rappresenta la sua valutazione. Il sistema di attribuzione dei crediti si occupa di gestire il flusso di trasferimenti dall'ambiente e fra regole. L'algoritmo genetico produce nuove regole mediante l'evoluzione di quelle presenti nell'insieme.

Figura 2.3 - *Learning-classifier-system*



La figura 2.3 contiene la rappresentazione grafica di un sistema tipico: *detector* ed *effector* costituiscono i legami del *classifier system* con il mondo esterno, cioè con l'ambiente in cui opera, il sistema di selezione delle regole è basato sul valore di ciascuna, rappresentato dal patrimonio di crediti, i trasferimenti sono gestiti da un apposito algoritmo, lo scambio di informazioni avviene per il tramite di messaggi immessi in una apposita lista, l'algoritmo genetico viene attivato dopo un certo numero di cicli di selezione per ricercare nuove regole.

La popolazione iniziale, dell'insieme di regole, può essere ottenuta in modo casuale, il valore assegnato a ciascuna viene stabilito in misura fissa tramite un parametro del sistema. Le regole sono definite su un insieme di tre valori: 0, 1, #, dove # rappresenta il carattere di indifferenza, in relazione al quale il confronto risulta sempre verificato, a prescindere dal valore del termine con cui viene paragonato. Questo accorgimento permette di lavorare, durante i primi cicli di apprendimento con regole, maggiormente generalizzate, in grado di garantire un *match* in mancanza di regole specifiche: si sfrutta, in questo modo, la potenza definitoria degli schemi. L'azione proposta dalle regole generalizzate risulta piuttosto imprecisa e viene corretta con la produzione di regole specifiche durante l'apprendimento. Assume particolare importanza il concetto di specificità di una regola, quantificabile, in linea di massima, contando il numero di posizioni contenenti valori diversi da # nella stringa. Un buon accorgimento consiste nel rapportare questo valore alla lunghezza totale della stringa, limitandolo all'intervallo]0,1[: risulta, così, possibile confrontare le specificità a prescindere dalle lunghezze e la loro, eventuale, utilizzazione computazionale viene semplificata. La presenza del carattere # può essere limitata alla parte di condizione della regola (Goldberg 1989) oppure essere estesa anche alla parte di azione.

Le informazioni provenienti dall'ambiente vengono tradotte, dai *detector*, in stringhe, definite su {0,1}, che sono inserite nella lista dei messaggi. Ciascun messaggio viene confrontato con la condizione di ogni regola: si verifica la corrispondenza di valori a parità di posizione, tenuto presente che il valore # può valere sia come 1 sia come 0; se la verificazione risulta positiva la regola interessata può concorrere per ottenere il diritto di emanare la propria azione. Al termine dell'accertamento dei *match* viene attivato un algoritmo di scelta per stabilire quali regole potranno agire, cioè attivare un'altra regola o proporre una azione tramite gli *effector*. Per concorrere con gli altri ed emanare la propria azione ogni individuo sostiene un costo che va ad incidere sul suo patrimonio di crediti.

Le azioni vengono comunicate all'ambiente per il tramite degli *effector* e questo fornisce una valutazione della bontà di ciascuna sotto forma di ricompensa (*reward*). L'ammontare del *reward* viene, successivamente, ripartito fra le varie regole che hanno concorso a proporre l'azione; la ricompensa viene, normalmente, mantenuta superiore o uguale a zero per motivi computazionali.

La procedura è ripetuta ciclicamente, così le regole in grado di proporre azioni buone possono ricostituire il proprio patrimonio, intaccato dal costo affrontato per proporre l'azione, mentre le regole portatrici di azioni inutili o sbagliate vedono rapidamente ridursi le proprie disponibilità di crediti: ad ogni ciclo le regole inesatte risultano sempre meno in grado di sostenere i costi necessari per ottenere il diritto di agire, il loro impoverimento, inoltre, tende a provocarne l'estinzione.

L'evoluzione del sistema si basa sull'azione dell'algoritmo genetico che opera sulle diverse regole in funzione della loro *fitness*, stabilita dalla misura del patrimonio di ciascuna: in questo modo le regole portatrici delle azioni migliori vengono riprodotte mentre quelle di scarso valore sono selezionate per l'estinzione. L'algoritmo genetico viene attivato, in modo stocastico, dopo un certo numero di cicli di selezione delle regole.

La ripetizione di cicli di selezione e di apprendimento, basato sull'algoritmo genetico, permette di ottenere popolazioni formate da individui progressivamente più abili nell'interazione con l'ambiente, fino a raggiungere il livello voluto di risultati. Non è richiesta, per i *classifier system,* la convergenza degli individui verso un tipo omogeneo, poiché la diversità viene gestita dal sistema di *matching* onde permettere l'evoluzione, contemporanea, di elementi specializzati nell'affrontare singoli problemi.

2.7.2 - L'attribuzione dei crediti con l'algoritmo *bucket brigade*

L'attribuzione dei crediti (Holland 1975), in un *classifier system*, è basata sulla competizione: le regole, che hanno ottenuto il diritto di agire, pagano una quota del proprio patrimonio a quelle che ne hanno permesso l'attivazione, cioè che hanno prodotto una azione in grado di verificare il *match* con quella regola. Ogni regola viene ad avere, in senso metaforico, fornitori e clienti: i primi sono le

regole che hanno concorso alla sua attivazione, i secondi sono quelle attivate dalla sua azione. Quando una regola ottiene il diritto di agire, il suo patrimonio viene ridotto dell'ammontare pagato; tale somma va ad incrementare il patrimonio dei suoi fornitori. Le regole possono recuperare le somme pagate sia perché hanno attivato altre regole vincenti, sia perché ricevono una ricompensa dall'ambiente; le migliori ricevono alte ricompense, quindi pagano forti somme ai propri fornitori, questi ultimi ai loro e così via. Il patrimonio di ciascuna regola tende a crescere quando questa riceva ricompense ambientali superiori alle somme pagate oppure quando sia in grado di attivare regole più "ricche" di lei. Se una regola propone una azione sbagliata riceve una ricompensa nulla, quindi il suo patrimonio si riduce; se, invece, attiva un'altra regola, a sua volta errata, provoca un impoverimento del suo cliente che, successivamente, le trasmetterà un minor ammontare di crediti: anche in questo caso il suo patrimonio tenderà a ridursi. Le regole possono sopravvivere solo se fanno parte di una catena, di fornitori e clienti, in grado di proporre le azioni corrette nell'ambiente. Questo algoritmo di distribuzione dei crediti viene chiamato *bucket brigade*: esso richiede che ciascuna regola interagisca esclusivamente con i propri clienti e fornitori diretti, quindi permette di evitare l'onere di memorizzare l'intera catena di regole che lega il messaggio, proveniente dall'ambiente, all'azione, proposta tramite l'*effector*. L'algoritmo *bucket brigade* consente di gestire, in modo efficiente, i trasferimenti patrimoniali, indipendentemente dal numero di regole utilizzate.

L'algoritmo *bucket brigade* può agire, in modo semplificato, anche quando ciascuna regola possa esclusivamente proporre azioni nell'ambiente: in questo caso non vi sono trasferimenti patrimoniali fra regole. L'analisi di questo algoritmo semplificato può agevolare la comprensione di quello generale. Il sistema funziona come un mercato (Goldberg 1989) dove gli operatori, singole regole, comprano e vendono il diritto di trattare le informazioni; esse costituiscono una catena di intermediari che collega i *detector*, produttori di informazione, agli *effector*, consumatori. Questo mercato è basato su due istituzioni: un'asta ed una stanza di compensazione; la prima permette di selezionare i *classifier* cui sarà attribuito il diritto di agire, la seconda di regolare i pagamenti. Le regole attivate effettuano un'offerta (O) per un'asta i cui vincitori potranno emettere le proprie azioni, dietro pagamento di un certo ammontare (P) alla stanza di compensazione; le regole possono, inoltre, ricevere dei compensi (C) dall'ambiente e devono pagare alcune tasse (T). Dato un generico individuo (i) il valore del suo patrimonio (F), cioè della sua forza, in un certo tempo t+1, può essere ottenuto sommando algebricamente al valore posseduto al tempo t il flusso di entrate per compensi e di uscite per pagamenti:

$$(1) \qquad F_i(t+1) = F_i(t) - P_i(t) - T_i(t) + C_i(t) .$$

L'ammontare di P è determinato dalla somma che la regola, se vincitrice dell'asta, deve versare alla stanza di compensazione, quindi dall'offerta fatta, O; quest'ultima è proporzionale al patrimonio, o forza, al tempo t, e dipende da un coefficiente α:

$$(2) \qquad P_i(t) = O_i(t) = F_i(t) * \alpha$$

Il vincitore dell'asta potrebbe, facilmente, individuarsi nella regola che ha effettuato l'offerta maggiore; il metodo risulterebbe, però, (De Groot 1970) troppo deterministico, con conseguenze negative sui risultati. Può essere opportuno l'inserimento di un elemento capace di creare un disturbo casuale: un valore addizionale calcolato in base ad una funzione di distribuzione (N), caratterizzata da una deviazione standard dipendente da un parametro (σ) del sistema:

$$(3) \qquad O'_i(t) = F_i(t) * \alpha + N(\sigma) .$$

L'asta viene vinta dai *classifier* che hanno effettuato le più alte offerte, O', questi versano alla stanza di compensazione l'ammontare O per ottenere il diritto di eseguire la propria azione.

La tassazione può essere ispirata a diversi criteri e articolata in una pluralità di imposte, per semplicità si considera il caso di una unica imposta proporzionale, il cui ammontare risulta determinato dal parametro τ:

$$(4) \qquad T_i(t) = F_i(t) * \tau$$

Aggregando i dati dei *classifier* attivi al tempo t è possibile scrivere:

(6) $\Sigma F_i(t+1) = \Sigma F_i(t) - \Sigma F_i(t) * \alpha - \Sigma F_i(t) * \tau + \Sigma C_i(t)$.

Dalla (6), ponendo $\gamma = \alpha + \tau$ e sottintendendo l'operazione di sommatoria si ha:

(7) $F(t+1) = (1 - \gamma) F(t) + C(t)$.

Trascurando, per ora, i compensi è possibile scrivere la 7 come:

(8) $F(t+1) = (1 - \gamma) F(t)$.

Considerando un intervallo di n periodi, dalla (8) è possibile trarre:

(9) $F(n) = (1 - \gamma)^n F(0)$.

L'espressione (9) risulta stabile per qualunque valore iniziale di F solo se $0 \leq \gamma \leq 2$; la condizione deve essere ulteriormente ristretta, per evitare che F assuma valori negativi, in: $0 \leq \gamma \leq 1$.

Le regole che propongono un'azione ricevono, nel ciclo successivo, una ricompensa in funzione dei risultati ottenuti nell'ambiente, quindi in n periodi il sistema riceverà n-1 ricompense ambientali e la forza accumulata risulterà da:

(10) $F(n) = (1 - \gamma)^n F(0) + \sum_{j=0}^{n-1} C(j) (1 - \gamma)^{n-j-1}$.

Il sistema raggiunge uno *steady-state* quando, posto C_{ss} il valore costante ricevuto dall'ambiente e F_{ss} la forza del sistema:

(11) $F_{ss} = C_{ss} / \gamma$.

La somma delle offerte in *steady-state* risulta da:

(12) $O_{ss} = (\alpha / \gamma) * C_{ss} = [\alpha / (\alpha + \tau)] C_{ss}$.

Normalmente, il coefficiente di tassazione risulta decisamente inferiore a quello per determinare l'ammontare dell'offerta, quindi è possibile considerare sufficiente, per la stabilità del sistema, l'eguaglianza fra corrispettivi pagati dall'ambiente e offerte in asta:

(13) $O_{ss} \approx C_{ss}$.

Ogni regola trae un vantaggio dalla partecipazione all'asta che può rifondere l'ammontare pagato alla stanza di compensazione: i *classifier* sostengono un costo proporzionale al loro patrimonio e ottengono ricompense in funzione dei risultati conseguiti. Se un *classifier* immette messaggi capaci di attivare un'altra regola si applica l'algoritmo *bucket brigade* completo, il meccanismo risulta meno diretto ma egualmente efficace: la ricompensa sarà proporzionale al patrimonio della regola attivata e quindi dipenderà, indirettamente, dai risultati da questa ottenuti. Diverse regole possono contribuire a formare una catena di trasformazione dell'informazione che giunga a determinare l'azione corretta.

2.7.3 - Ulteriori considerazioni

I *classifier system* sono basati su due tipi di apprendimento: uno a breve termine ed uno più profondo; ambedue i sistemi sono guidati dalla variazione del valore del patrimonio di ciascun elemento. Il primo permette di assegnare possibilità di azione proporzionali ai risultati, ottenuti da ciascuna regola, e potrebbe essere interpretato come selezione nell'ambito delle soluzioni note; esso viene realizzato per il tramite dell'asta e del meccanismo di gestione dei trasferimenti. Il secondo è ottenuto attraverso

le azioni di riproduzione, incrocio e mutazione, affidate all'algoritmo genetico; esso risulta, chiaramente, influenzato dai trasferimenti dall'ambiente e fra regole ed ha come scopo la generazione di soluzioni nuove. La sua azione potrebbe assimilarsi ad uno studio maggiormente approfondito del problema. Poiché l'apprendimento profondo si basa sulle risultanze di quello a breve termine, è importante che possa agire su una popolazione sufficientemente sperimentata, il cui patrimonio abbia subito un congruo numero di modificazioni in forza dell'agire dei singoli: l'algoritmo genetico deve essere attivato con frequenza tale da permettere l'accumulazione di patrimoni.

L'attivazione delle regole, in dipendenza del *match* con i messaggi, favorisce la formazione di nicchie di specializzazione: il patrimonio degli appartenenti a razze diverse è preservato dall'azione dell'algoritmo di ridistribuzione quando la situazione da affrontare esuli i loro compiti. In una popolazione, generata casualmente, potrà aversi la presenza di individui portatori di una condizione molto generalizzata, quindi atti a proporre azioni in un gran numero di situazioni, o di individui inutili, la cui condizione risulta sistematicamente non soddisfatta. I *classifier* del primo tipo risultano avvantaggiati, nella fase di *match,* rispetto a quelli più specifici; per compensare questo privilegio può essere opportuno calcolare l'offerta per l'asta anche in funzione della specificità. L'idea di base è che le regole generiche debbano intervenire solo in mancanza di altre, più specifiche, atte ad agire a fronte della particolare condizione ambientale; esprimendo la specificità in termini relativi si ha:

(14) s = numero di posizioni $\in \{0,1\}$ / numero di posizioni nella stringa .

I valori di O e O' possono essere calcolati come:

(15) $O_i = F_i * s_i * \alpha$.
(16) $O'_i = F_i * s_i * \alpha + N(\sigma)$.

Le regole specifiche avrebbero valori di s prossimi all'unità, mentre quelle più generali avrebbero valori tendenti a zero; le offerte delle regole generali risulterebbero sistematicamente inferiori, con l'effetto di limitarne l'applicazione ai soli casi di effettiva necessità. L'introduzione del valore di specificità, nel calcolo dell'offerta per l'asta, potrebbe simulare la naturale diffidenza per soluzioni troppo generiche che, molto probabilmente, sarebbero poco utili per il raggiungimento di risultati ottimali.

L'evoluzione di regole specifiche potrebbe, inoltre, essere favorita dall'imposizione di una tassa sulle offerte in asta, *bidTax*; il suo ammontare dovrebbe essere proporzionale al patrimonio della regola, senza riferimento alla specificità. Una simile tassa impoverirebbe ulteriormente le regole generali che, in presenza di antagonisti maggiormente specializzati, effettuerebbero comunque una offerta e pagherebbero la *bidTax*, ma avrebbero scarsa probabilità di recuperare la spesa perché difficilmente potrebbero vincere l'asta. Si noti che l'effetto risulterebbe, invece, annullato in assenza di regole specifiche.

Gli individui inutili permangono nel sistema fino a quando la staticità del loro patrimonio non comporti una povertà, relativa, sufficientemente marcata da provocarne l'estinzione. Questi *classifier*, non risultando mai attivati, non subiscono gli effetti della *bidTax*: occorre, allora, introdurre una *lifeTax,* cioè una imposta prelevata, ad ogni ciclo, su tutti i patrimoni, per accelerare l'impoverimento e la scomparsa delle regole inutili. Mantenendo un'aliquota sufficientemente bassa, le regole attive possono facilmente recuperare l'ammontare della tassa con i compensi ricevuti, mentre quelle che non vengono mai attivate perdono, ad ogni ciclo, una parte del proprio patrimonio, divenendo buone candidate per l'estinzione durante l'azione dell'algoritmo genetico.

3 - Programmi per gestire l'apprendimento

In questo capitolo viene svolto l'esame del *software* realizzato: l'obiettivo dei vari programmi è quello di fornire uno strumento utilizzabile per simulazioni relative a varie tematiche. Data l'enfasi posta sulla possibilità di impiego per diversi studi, l'architettura del progetto, basato sull'utilizzo di *Swarm*, viene rapportata allo schema ERA (Terna 1998) non solo per questioni formali, ma, particolarmente, per dotarla di quelle caratteristiche, stilistiche e strutturali, la cui importanza è stata illustrata agli inizi della trattazione.

I pacchetti-programma realizzati sono due: *Classifier Workbench* (CW), basato sul metodo dei *classifier system,* e *Genetic Manipulator* (GM), volto alla realizzazione di algoritmi genetici. Nelle sezioni loro dedicate, vengono trattati: struttura, funzionamento, modalità di personalizzazione, possibilità di utilizzo e strumentazione fornita a corredo; quest'ultima è specificamente intesa per favorire attività di controllo: indagine della sequenza di operazioni effettuate ed esplorazione delle aree di memoria.

A conclusione del capitolo, piuttosto impegnativo dal punto di vista tecnico, viene presentato un lavoro volto a dimostrare come un *classifier system* possa "apprendere", cioè modificare le sue regole di comportamento verso il raggiungimento di un certo fine. La scelta di muovere dall'esperimento, già noto, della "formica di Langton", permette di richiamare i vantaggi dei modelli basati su agenti in ordine alla agevole riproduzione di situazioni complesse. Nella simulazione, l'utilizzo di *classifier* non evolutivi, accanto ad un *learning-classifier*, oltre a dimostrare la duttilità del *software* realizzato, sottolinea l'analogia fra *classifier-system* e sistemi esperti.

3.1 - Gli obiettivi del progetto

La redazione dei due pacchetti muove verso l'obiettivo di fornire uno strumento per simulazioni condotte utilizzando *Swarm*. Particolare importanza è stata data alle caratteristiche, di generalità e chiarezza, ritenute indispensabili per rendere effettivamente utile il prodotto finale. Nell'intenzione del progetto, destinatari di CW e GM dovrebbero essere ricercatori impegnati in discipline diverse, particolarmente in quelle economiche e sociali, che, pur non avendo interesse né tempo da dedicare alla redazione di *software*, necessitano di conoscere, nel dettaglio, il comportamento dello strumento utilizzato, onde ottenere garanzie, in ordine alla corretta predisposizione del proprio modello, indispensabili per poter usare i risultati della simulazione nel successivo processo di inferenza di leggi e teorie. In quest'ottica, è stata posta la dovuta attenzione nel predisporre gruppi di programmi facilmente adattabili alle esigenze: sia perché in grado di operare in base ad un ampio insieme di parametri, capaci di influire sul comportamento del sistema nella direzione voluta dall'utilizzatore, sia perché richiedenti il minor numero possibile di interventi nel codice, tanto del prodotto che dei diversi modelli realizzati. L'idea è che CW e GM possano essere usati: sia nella costruzione di modelli totalmente nuovi, sia nella modificazione di altri, già realizzati, operanti in base a regole comportamentali fisse o a meccanismi di inferenza diversi da quelli evolutivi.

Ciascuno dei due obiettivi citati è stato perseguito per il tramite della sua strutturazione in sub-obiettivi; ciò ha consentito di fornire precise linee guida per il successivo lavoro di redazione dei programmi. La generalità è stata vista come derivante da caratteristiche di: fungibilità del prodotto in situazioni diverse, adattabilità del comportamento dello strumento e possibilità di gestione parallela di diverse popolazioni in evoluzione. Le questioni relative alle caratteristiche tecniche, delle macchine utilizzate, o dei sistemi operativi installati, hanno potuto essere trascurate: *Swarm* provvede a fornire uno strato, compatibile con i sistemi operativi più usati, su cui innestare lo sviluppo del progetto; i sub-obiettivi citati fanno riferimento, esclusivamente, agli aspetti funzionali dei programmi. La fungibilità è stata ricercata nell'informare l'attività di sviluppo a criteri di isolamento e specificazione delle diverse componenti, dividendole in due grosse tipologie identificabili come: interne, *kernel*, o di frontiera, zona di comunicazione con i programmi dell'utilizzatore, *front-end*. L'impostazione del progetto segue la partizione di ruoli tipica delle applicazioni *client-server*. Motivazioni inerenti la

flessibilità operativa dei programmi hanno consigliato di raccogliere una ricca collezione di parametri, in grado di condizionare lo svolgimento dell'elaborazione; per CW è stata anche prevista una classe di oggetti specializzati nella gestione dei valori attribuiti ai diversi parametri. Operazione analoga è stata effettuata per il trattamento dei dati dei singoli utilizzatori. La possibilità di gestire popolazioni diverse, ciascuna ad uno stadio proprio di sviluppo, è intesa a fornire la maggior facilità di predisposizione del modello: si è voluto ridurre al minimo il numero di istanze delle classi del *kernel* necessarie al funzionamento dello stesso.

La chiarezza dell'azione è stata ricondotta a caratteristiche di: semplicità e immediata comprensibilità dei programmi, forte specializzazione funzionale dei componenti, capacità descrittiva dei nomi adottati per le singole variabili, dotazione di strumenti per il tracciamento dell'azione del sistema e per l'esplorazione del contenuto della memoria. Come sarà meglio chiarito nel prossimo paragrafo, l'adozione dello schema ERA ha consentito di impostare il lavoro in base ad una robusta architettura. La comprensibilità dei programmi ha consigliato la scelta di processi estremamente semplici e di una rappresentazione dei dati di immediata leggibilità, anche sacrificando, ove necessario, possibilità di maggior efficienza o eleganza della programmazione. La specializzazione funzionale, pur comportando un certo proliferare dei componenti, ha permesso di rendere modulare e flessibile il prodotto, costituendo un valido aiuto per quei casi in cui l'utilizzatore volesse apportare personalizzazioni anche al *kernel*. L'uso di nomi fortemente descrittivi, pur comportando una estensione dei medesimi, permette di individuare rapidamente natura e funzione del contenuto delle variabili: l'ingrandimento del corpo delle istruzioni è stato considerato un equo prezzo da pagare, in cambio del conseguimento di un così importante beneficio. Gli strumenti di tracciamento possono risultare particolarmente utili per comprendere l'azione del sistema: nel progetto è stata posta notevole attenzione ad ottenere la possibilità di estendere in misura diversa la loro azione, permettendo di operare a più livelli di dettaglio, nell'ottica di soddisfare le esigenze dell'utilizzatore senza "appesantire" gli *output,* con informazioni non richieste o inutili. Lo strumento di visualizzazione dello stato della memoria, pur risultando, per sua natura, di utilizzo non intuitivo, è pensato per agevolare gli utilizzatori con interessi maggiormente tecnici.

Un'ulteriore fase di affinamento potrà riguardare, in modo più specifico, obiettivi attinenti le prestazioni dello strumento, particolarmente la velocità, ed una maggiore generalizzazione, ottenibile fondendo i due pacchetti in un solo prodotto capace di gestire i due tipi di evoluzione. Le funzioni di *tracing* potranno essere potenziate per il tramite di un *internal log,* i cui contenuti potrebbero costituire la collezione delle informazioni utili per una componente di *debug* specifica. La metodologia adottata potrà, inoltre, essere estesa recependo paradigmi più complessi quali quelli dei *parallel genetic algorithm*. Questi argomenti sono trattati in modo più ampio nelle conclusioni del presente lavoro.

3.2 - Cenni sulla programmazione ad oggetti

L'utilizzo di *Swarm*, interamente scritto in *Objective-C*, rende opportunamente vantaggiosa l'adozione dello stesso linguaggio per la redazione dei programmi per le varie applicazioni. *Objective-C* fa parte della famiglia dei linguaggi per la programmazione "ad oggetti": un insieme di tecniche che portano ad ottenere programmi maggiormente modulari, affidabili e utilizzabili in diversi contesti.

La *Object Oriented Programming* (OOP) permette di basare la programmazione sui dati invece che sulle operazioni: questi vengono "incapsulati" insieme con le funzioni, metodi, che li manipolano, in entità denominate, appunto, oggetti. Ogni oggetto è indipendente e comunica con gli altri attraverso "messaggi". Un programma consta di una rete di oggetti in interazione reciproca. Le caratteristiche dei vari oggetti, struttura dei dati e codice per l'accesso ai medesimi, sono definiti in tipi astratti detti classi. Gli oggetti sono creati come istanze di una classe: essi si differenziano fra loro per il valore dei dati contenuti. Una convenzione di programmazione, molto diffusa, stabilisce che classi ed oggetti, istanze, vengano indicati con lo stesso nome, differenziando le prime per l'iniziale, obbligatoriamente, maiuscola; tale convenzione è adottata anche nell'esposizione delle pagine a seguire. Si noti, inoltre, come la metafora individuale, sottesa dal concetto di interazione fra gli

oggetti, renda la OOP particolarmente adatta alla redazione dei programmi per la realizzazione di modelli di agenti.

L'attività di programmazione consta della redazione del codice per definire le classi, mentre le singole istanze, oggetti, saranno "create", durante l'elaborazione, nella quantità richiesta dai fini della stessa. Le classi sono, concettualmente, collocate in una precisa gerarchia, il cui vertice è costituito da una classe principale, *root*, caratteristica del linguaggio per la OOP utilizzato. Ogni classe dipende da un'altra, al limite dalla *root*, ed eredita, automaticamente, tutte le caratteristiche di quella. Grazie al meccanismo dell'ereditarietà, ciascun oggetto diviene, automaticamente, in grado di effettuare tutte le operazioni codificate, oltre che nella classe di cui rappresenta un'istanza, in tutte le classi da cui dipende. In questo modo il codice di ciascuna classe può essere agevolmente riusato: ciascun programmatore può concentrarsi sulla redazione delle funzioni di sua competenza, si evita la duplicazione di parti del codice e si abbatte, notevolmente, l'entropia.

Altro concetto importate è quello di polimorfismo: esso permette di conferire alla OOP una forte duttilità, pur in presenza di rigide dipendenze fra classi, realizzando una completa indipendenza nella definizione dei metodi. I metodi sono identificati da un nome, *selector*, che deve essere univoco all'interno di ciascuna classe, ma può essere definito in più classi: quando una classe contiene un *selector*, già presente in una delle entità da cui dipende, quest'ultimo prevale, permettendo, perciò, di modificare, con validità limitata alla classe, il contenuto del metodo.

3.3 - L’architettura del progetto in base allo schema ERA

Lo schema *Environment Rule Agent* (Terna 1998) prevede la distinzione di quattro livelli funzionali diversi all'interno del modello. Il primo è quello dell'ambiente, dove sono contenuti gli oggetti rappresentativi di istituzioni centrali o incaricati della gestione delle regole ambientali e della interazione fra gli agenti. Il secondo contiene gli agenti veri e propri, cioè gli oggetti scritti per simulare il comportamento delle singole entità operanti nell'ambiente reale. Le componenti di questo livello possono rappresentare singole persone, come consumatori o investitori, oppure aziende, società di intermediazione e simili. Gli agenti operano in base a regole comportamentali gestite da entità appartenenti al terzo livello dello schema: i gestori di regole (*Rule Master*); quando le regole comportamentali degli agenti siano soggette ad evoluzione durante l'esecuzione della simulazione, lo schema prevede la presenza di oggetti produttori di regole (*Rule Maker*), che costituiscono il quarto livello.

E' facile notare come il *software* attinente prodotti dell'intelligenza artificiale si collochi, per sua natura, nel terzo e quarto livello, mentre nei primi due vengano a trovarsi gli elementi che descrivono il modello. Risulta, perciò, naturale la partizione del modello stesso in: applicazione di gestione del comportamento degli agenti, *kernel,* ed elementi del modello la cui codificazione è tipicamente a carico dell’utilizzatore, *user*. Il raccordo fra le due parti viene affidato ad uno strato intermedio, di *front-end* dell’applicazione di gestione delle regole. Quest'ultimo risulta, in base a quanto previsto in ERA, composto da due classi di oggetti denominate, rispettivamente, *Interface* e *DataWarehouse*. Le istanze della prima classe sono deputate alla traduzione dei dati del modello nella metrica tipica dello strumento di inferenza, inserito nei livelli successivi, mentre quelle della seconda risultano essere gli oggetti, contenitori, in cui ciascun agente conserva il proprio patrimonio normativo. Uno degli obiettivi del progetto è quello di limitare l'esigenza di interventi di personalizzazione a queste due classi ed alla codificazione dei parametri relativi al funzionamento dell'algoritmo prescelto, da effettuarsi intervenendo sull'apposito *file* di *setup* iniziale dell'oggetto gestore dei parametri.

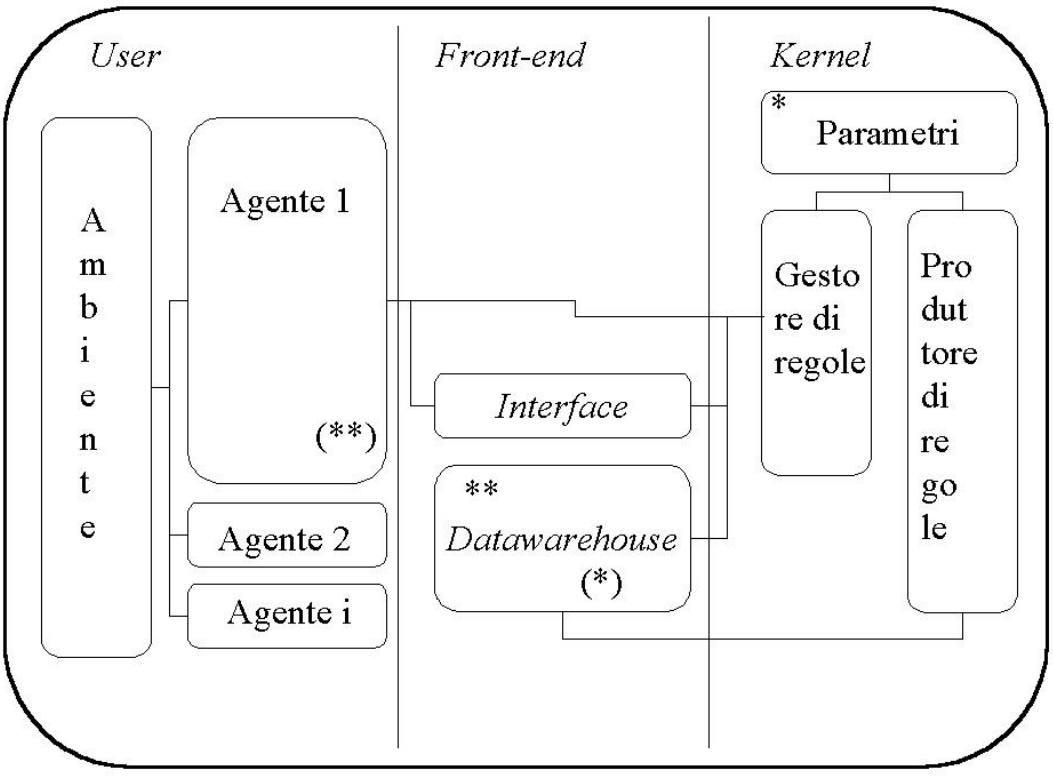
Come risulterà meglio dalla descrizione dettagliata di ciascun pacchetto, degli oggetti di alto livello è sufficiente la creazione di un'unica istanza; la pluralità di comportamento risulta scaturire dall'esistenza di istanze diverse delle classi *Interface* e *DataWarehouse*. Al proposito, pur risultando naturale creare un oggetto *interface* ed un *dataWarehouse* per ogni agente, questo dettaglio non è tecnicamente richiesto, potendo tutti gli agenti appartenenti ad un gruppo, caratterizzato

dall'uniformità del comportamento degli individui, condividere lo stesso patrimonio normativo e, quindi, lo stesso *dataWarehouse.* Le possibilità di condivisione dell'*interface* risultano ancora più elevate. Gli agenti possono anche possedere più *dataWarehouse*, *interface* e oggetti di gestione dei parametri: in questo modo ogni agente potrebbe valersi di diversi sistemi evolutivi, relativi a gruppi differenti di regole, per decidere relativamente a questioni di natura specifica.

Discorso a parte va fatto per la classe degli oggetti gestori di parametri, il cui utilizzo è specifico delle applicazioni presentate. La loro differenziazione risponde all'esigenza di limitare il numero di gestori e produttori di regole, pur garantendo la possibilità di condurre sistemi diversi anche nelle modalità di azione del *kernel*, oltre che perché operanti su popolazioni differenti. Per questo motivo viene memorizzato, nel *dataWarehouse*, in aggiunta all'insieme delle regole, anche l'indirizzo dell'oggetto di gestione dei parametri utilizzato da ciascun agente che, per altro, potrà essere condiviso anche da più agenti.

In conclusione, la struttura del progetto, così come presentata nella figura 3.1, prevede la possibilità di operare, tramite una sola istanza di *Rule Master* ed una di *Rule Maker*, in parallelo sui patrimoni genetici di agenti: caratterizzati da stadi diversi di sviluppo, volti a risolvere problemi differenti, operanti con metriche specifiche di ciascun agente. Tutto questo è ottenibile utilizzando più istanze di *interface* e *dataWarehouse*. Quando si vogliano utilizzare *classifier system* o algoritmi genetici "tecnicamente" diversi, dovranno essere moltiplicate anche le istanze del gestore di parametri, realizzando così una differenziazione metodologica, oltre che evolutiva e finale, dei diversi agenti. Nella figura, le linee di connessione indicano la presenza di una comunicazione fra le componenti, mentre gli asterischi indicano che un oggetto conosce l'indirizzo di un altro, pur senza comunicare direttamente con quello.

Figura 3.1 - Struttura generale dei modelli derivante dall'adozione dello schema ERA



3.4 - *Classifier Workbench*

Il pacchetto consta di un modello *Swarm* completo, dove lo strato *user* contiene un'applicazione di *workbench*, scritta per permettere di simulare problemi diversi, onde sperimentare il comportamento del *kernel* senza bisogno di sviluppare un modello specifico. Il *kernel* contiene le classi di oggetti per condurre l'elaborazione tipica di un *classifier system*, il cui comportamento risulta influenzato dai parametri specificati in un apposito *file.* L'utilizzatore può, quindi, sperimentare diverse configurazioni di parametri: per individuare quella che gli permetterà di ottenere i migliori risultati nella applicazione concreta o, semplicemente, per effettuare esperienze sul funzionamento del meccanismo. L'intero pacchetto può, in seguito, costituire lo scheletro da cui sviluppare il modello, sostituendo le componenti dello strato *user* con quelle specifiche della simulazione voluta, e personalizzando quelle dello strato *front-end*.

CW è basato sul paradigma mentale degli algoritmi evolutivi: i singoli cromosomi, in questo caso regole, sono considerati come idee che popolano il pensiero di un agente. Il *RuleMaster* provvede a selezionare, di volta in volta, l'idea migliore da applicare, data la descrizione delle condizioni ambientali fornita dall'agente, per suggerire, a quest'ultimo, quale fra le azioni ammesse effettuare nell'ambiente. Dopo ogni azione l'agente deve comunicare al *RuleMaster* una valutazione della stessa, che influirà sulle probabilità di sopravvivenza e riproduzione della regola responsabile del suggerimento. Per evolvere la popolazione di idee, il *RuleMaster* si vale dei servizi del *RuleMaker*, capace di effettuare la manipolazione genetica delle singole idee. E' possibile usare CW nell'ambito di una concezione individuale, dove ogni cromosoma rappresenta un intero individuo: i singoli agenti dovrebbero possedere un solo cromosoma; i diversi cromosomi confluirebbero tutti nello stesso *dataWarehouse*, condiviso da tutti gli agenti.

Il *kernel* di CW è un *learning-classifier-system*, cioè un sistema di inferenza, basato su regole, in grado di evolvere, modificando le stesse, verso il conseguimento di determinati obiettivi, il cui grado di raggiungimento è comunicato, per il tramite della valutazione delle singole azioni suggerite, dall'agente dopo ogni interazione. Ogni idea viene così ad accumulare un patrimonio di crediti, il cui ammontare permette al sistema di conoscerne l'efficacia. L'apprendimento avviene: sia per il tramite della selezione delle idee più efficaci, sia attraverso la riproduzione e l'incrocio delle medesime per ottenere nuove idee. Queste azioni possono essere inibite azzerando il valore di alcuni parametri e, quindi, CW può essere configurato, senza modificazioni del *software*, come: *learning-classifier*, *non-learning-classifier*, in grado di effettuare solo la selezione delle idee migliori partendo dalla popolazione iniziale, e sistema esperto tradizionale, capace di applicare singole regole, predeterminate, in risposta alle diverse situazioni. Quanto detto vale, mutatis mutandis, anche nel caso di utilizzo secondo il paradigma individuale: oggetto della selezione non sarebbero più le idee di ogni agente, ma gli agenti stessi.

La classe *DataWarehouse*, contenuta in CW, possiede i metodi per generare, automaticamente, in modo pseudo-casuale, una popolazione iniziale o per effettuare il caricamento, *load,* della medesima da *file.* Risulta, quindi, possibile far interagire entità che partano da stadi di evoluzione diversi e siano dotate di insiemi di idee differentemente numerosi, concepiti per codificare condizioni ed azioni proprie di ciascun agente o comuni a più agenti.

Tutti i nomi dei *file* vengono comunicati ai vari oggetti utilizzatori al momento della loro creazione, solitamente nel *modelSwarm*, quindi nomi e numero dei *file* utilizzati possono essere agevolmente modificati dall'utilizzatore.

In conclusione, CW, utilizzando un'unica istanza di *RuleMaster* e di *RuleMaker*, può gestire, parallelamente, le decisioni relative ad agenti, variamente adattivi, secondo i tre tipi seguenti: agenti operanti in base ad un *learning classifier,* agenti esclusivamente capaci di selezionare il comportamento migliore fra una serie di possibilità, *non learning classifier,* agenti puramente reattivi, operanti in base ad un sistema esperto rigido, capace solamente di associare una predeterminata azione alla condizione contingente. Ciascuna delle tre tipologie va vista come sistema autonomo, derivante dall'adozione di parametri diversi per l'erogazione dei servizi ai vari oggetti *client,* da parte dell'unica coppia di oggetti *server*, *ruleMaster* e *ruleMaker*, in base a quanto specificato nel *dataWarehouse* di

ciascun *client*. Dotando *client*, o gruppi di *client*, di specifici oggetti gestori di parametri, *classifierParm*, è possibile differenziare ulteriormente il comportamento dei singoli, all'interno delle tre tipologie, giungendo, al limite, a dotare ciascun agente di un proprio, differente, sistema decisivo. Nel prosieguo del paragrafo si procede ad un'illustrazione maggiormente dettagliata di questi aspetti, fornendo sia la descrizione precisa del funzionamento di CW, sia le istruzioni per il suo utilizzo.

3.4.1 – Struttura ed interazioni in un modello con CW

Aderendo all'impostazione mentale, ogni agente deve disporre di un proprio patrimonio di idee, soggetto ad un'evoluzione autonoma. Non sono ammesse comunicazioni dirette fra gli agenti, seguendo in ciò il consiglio di Terna (1998), volto a favorire la chiarezza del modello. Ogni agente deve essere proprietario di un *dataWarehouse*, dove sono contenute tutte le informazioni relative al suo stato attuale di evoluzione: indirizzi dei gruppi di oggetti rappresentanti gli individui che formano il suo genotipo, indirizzo della regola attiva, che ha proposto l'ultima azione, indirizzo dell'oggetto contenente i parametri di funzionamento del *classifier* per quell'agente, contatori delle varie azioni evolutive effettuate, utili a fini statistici, eventuale nome del *file* da cui effettuare il *load* iniziale della popolazione.

Ogni agente conosce l'indirizzo di un'*interface,* che potrà risultare specifica di quell'agente o condivisa con altri. La condivisione non è possibile fra agenti che debbano trattare problemi diversi, poiché le metriche interne risultano differenti. In questo caso sarebbe meglio diversificare anche i rispettivi *classifierParm*, pur se la classe risulta dotata dei metodi, necessari per cambiare dinamicamente il valore dei parametri, volti a consentire variazioni durante l'elaborazione.

Da ultimo, l'agente deve conoscere l'indirizzo del *ruleMaster*, uno solo può bastare per tutto il modello, ed essere in grado di ottenere informazioni dall'ambiente, che comunicherà al *ruleMaster* per il tramite dei servizi di traduzione dell'*interface*. Gli stessi servizi saranno usati, in senso contrario, per interpretare il consiglio ricevuto dal *ruleMaster*. In questo modo il *classifier* ha una visione dell'ambiente che risulta mediata dalle percezioni dell'agente che ne richiede i servizi: ciò permette di rendere ulteriormente caratteristico il comportamento dei vari agenti, innalzando il grado di fedeltà della simulazione e, conseguentemente, quello di plausibilità del modello.

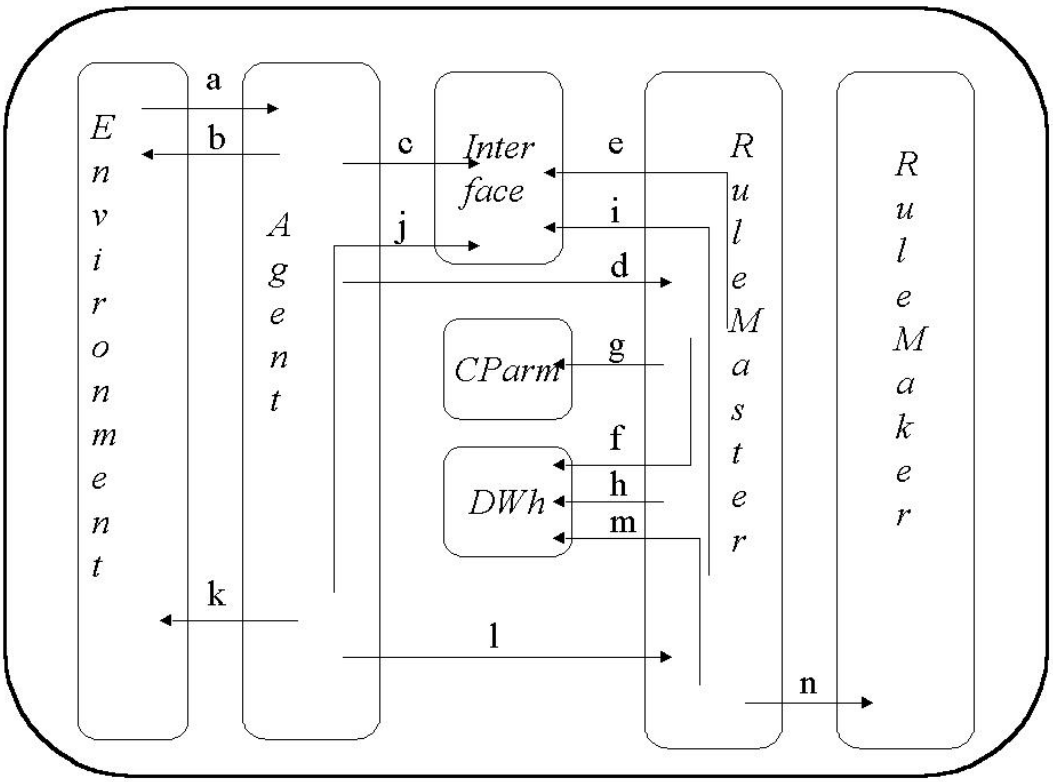
Lo schema, in figura 3.2, riporta la sequenza delle interazioni, fra i vari oggetti, che potrebbero configurare quattro fasi operative: i) reperimento e codificazione delle informazioni, a cura dell'agente, ii) elaborazione, da parte del *ruleMaster*, iii) decodificazione e messa in pratica del consiglio ricevuto, da parte dell'agente, iiii) valutazione delle conseguenze dopo l'interazione con l'ambiente, calcolo e comunicazione al *ruleMaster* della valutazione assegnata al consiglio ricevuto, ancora a cura dell'agente.

Innanzitutto l'agente riceve (a) , dall'ambiente, l'ordine di compiere un'azione; in risposta a ciò questi richiede (b), all'ambiente, le informazioni necessarie a formare la sua percezione del problema. I dati ottenuti vengono comunicati (c) all'*interface,* perché li traduca nella metrica adatta allo specifico *classifier*, dati i parametri di funzionamento adottati dall'agente. Avendo terminato la fase di codificazione della richiesta l'agente può, a questo punto, richiedere (d) un consiglio al *ruleMaster,* avendo cura di comunicare l'indirizzo dell'*interface* e del *dataWarehouse* da usare.

Nella seconda fase il *ruleMaster* interroga (e, f, g) un'*interface,* un *dataWarehouse*, quelli indicati dall'agente, ed un *classifierParm*, quello indicato nel *dataWarehouse*, per ottenere: le informazioni ambientali, gli indirizzi dei dati su cui operare ed i valori dei parametri da applicare durante l'elaborazione. In questo modo viene ricostruito il *classifier* specifico dell'agente, così che il programma possa operare su di esso. Durante il suo funzionamento il *ruleMaster* provvede a modificare (h) il contenuto delle variabili statistiche ed a variare l'indirizzo della regola attiva, contenuti nel *dataWarehouse*: viene aggiornata l'immagine del livello evolutivo dell'agente. Stabilita l'azione da effettuare questa viene comunicata (i) all'*interface* e il controllo dell'elaborazione torna all'oggetto agente.

La terza fase inizia con la richiesta (j) dell'agente, verso l'*interface,* per ottenere la traduzione dell'azione suggerita dal *ruleMaster*. Successivamente quell'azione viene effettuata (k) nell'ambiente ed il controllo dell'elaborazione torna all'ambiente, che potrà attivare un altro agente.

Figura 3.2 - Interazione fra oggetti in un modello con CW.



La quarta fase può prendere le mosse da uno specifico ordine dell'ambiente o essere effettuata prima di riavviare il ciclo, relativo ad un nuovo *step*, in dipendenza dalle caratteristiche dello specifico modello. L'agente valuta le conseguenze dell'azione, solitamente questa comporta la modificazione di alcuni valori di stato dell'agente stesso, ad esempio il patrimonio, e comunica (l) il proprio apprezzamento al *ruleMaster*. Quest'ultimo, ricostruito il *classifier* specifico interrogando *dataWarehouse* ed *interface* indicati dall'agente, provvede ad alterare i valori dei patrimoni delle regole in base alla ricompensa ricevuta; i dati nel *dataWarehouse* vengono aggiornati (m). In dipendenza dai parametri utilizzati e dallo stato dell'evoluzione, il *ruleMaster* decide se richiedere (n) i servizi del *ruleMaker,* per evolvere l'insieme di individui contenuto nel *dataWarehouse* che, terminata l'attività, sarà ulteriormente aggiornato a cura del *ruleMaker* stesso.

3.4.2 – La gestione dei *classifier system*

Nel capitolo precedente sono state individuate le tre componenti principali di un *classifier system*: insieme di regole e messaggi, sistema di distribuzione dei crediti, algoritmo genetico. CW gestisce più *classifier* contemporaneamente, derivanti dalla combinazione di istanze diverse di *DataWarehouse* e *ClassifierParm*. Il *kernel* è in grado di effettuare le operazioni di distribuzione dei crediti e di evolvere la popolazione, in base ad un algoritmo genetico, agendo su insiemi diversi di regole e messaggi e utilizzando configurazioni differenti di parametri. Ogni combinazione di parametri e insiemi di regole caratterizza un diverso *classifier*, pur se gli oggetti incaricati delle operazioni restano sempre gli stessi.

Gli insiemi di regole e messaggi sono rappresentati da gruppi di oggetti, organizzati in base ai protocolli delle “liste” di *Swarm*; tale tipologia presenta vantaggi in ordine alle dinamiche di inserimento, interazione ed eliminazione di oggetti. Tutti gli elementi di una lista possono essere acceduti inviando appositi messaggi alla lista stessa e quindi non è necessario memorizzare gli indirizzi dei singoli oggetti.

Il sistema è basato sull'interazione delle istanze di quattro classi: *Rule*, *Message*, *Effector, Treasury*. Ogni *dataWarehouse,* di ciascun agente, contiene una o più istanze di queste classi, variamente raggruppate in liste. Sono utilizzate sei liste: *ruleList*, *messageList*, *effectorList*, destinate a contenere gli indirizzi degli oggetti nominati, *matchList*, per le regole, presenti anche nella *ruleList*, che soddisfanno le condizioni di *match* con le informazioni ambientali comunicate dall'agente, *winnerList*, per le regole che hanno vinto l'asta, *workList*, di appoggio per l'elaborazione. Altre liste, puramente transitorie, sono utilizzate da altri oggetti come verrà illustrato in seguito. L'utilizzo delle liste evita il bisogno di disporre di *flag* di stato, all'interno di ciascun oggetto, per ricordare quale sia il suo contributo nell'elaborazione corrente; consente, inoltre, di ottenere insiemi omogenei, ad esempio quello di tutte le regole che hanno soddisfatto il *match*, senza dover effettuare scansioni diverse della *ruleList* e permette una facile gestione del ricambio generazionale.

Ciascuna *rule* contiene un cromosoma formato da due genomi, *array* di caratteri, che rappresentano, rispettivamente, la *condition-part* e la *action-part*. La loro estensione, numero di posizioni utilizzate, può variare fino ad un massimo codificato, facilmente estensibile. Ogni posizione può assumere i valori: "0", "1" o "#", quest'ultimo rappresenta il carattere di indifferenza, *wildcard*. Apposite variabili contengono l'ammontare del patrimonio di crediti di ciascuna regola, *strength*, il valore di specificità della stessa, *specificity,* e l'ammontare dell'offerta, fatta dalla regola nell'asta per determinare quali individui avranno il diritto di immettere il proprio messaggio nella *messageList*. Ogni regola contiene una *partnerList* dove memorizzare gli indirizzi delle entità autrici dei messaggi che hanno permesso l'attivazione, conseguente al *match*, della regola stessa.

I messaggi possiedono un cromosoma formato da un solo genoma, contenente il testo del messaggio, cioè la traduzione delle informazioni ambientali o la copia della *action-part* di una regola; in ciascun messaggio è indicato, inoltre, l'indirizzo del suo autore, o proprietario, *owner*.

Gli *effector* sono simili alle regole, ma la loro *action-part* contiene la codifica, binaria, di un numero identificante l'azione da consigliare all'agente. La traduzione di questo valore è eseguita *dall'interface*. Anche gli *effector* possiedono una *partnerList,* per memorizzare quali regole li abbiano attivati; un'apposita variabile è utilizzata per registrare il valore di *support*, cioè la somma delle offerte fatte dalle varie regole che hanno attivato l'*effector*, utile per decidere quale, fra gli *effector* attivi, debba prevalere e suggerire l'azione, di cui è portatore, all'agente.

La *treasury* è la controparte dei movimenti di crediti effettuati durante l'elaborazione: fornisce il numerario per pagare le ricompense, incassa le imposte e le ricompense verso la *routine* di *detecting*. In questa versione il suo ruolo è fortemente passivo ma è previsto un ampliamento del medesimo in versioni successive di CW.

Quando viene richiesto un consiglio al *ruleMaster,* questi provvede a caricare i dati dal *dataWarehouse* indicato dall'agente, in particolare: gli indirizzi delle sei liste presenti nel *dataWarehouse*, l'indirizzo del *classifierParm* e, interrogando quest'ultimo, i valori dei parametri di funzionamento, che saranno descritti in seguito. Subito dopo interroga l'*interface*, il cui indirizzo è il valore del secondo parametro ricevuto dall'agente, per conoscere i dati relativi all'ambiente; con questi dati, costruisce un messaggio e lo immette nella *messageList* dopo quelli, provenienti dalle regole, residuati dal ciclo precedente. Il terzo passo è costituito dalla ricerca delle regole che soddisfano il *match* con almeno uno dei messaggi della lista; queste vengono immesse nella *matchList*, precedentemente svuotata. Se nessuna regola soddisfa il *match*, cioè se la *matchList,* dopo il confronto di tutte le regole con tutti i messaggi, risulta vuota, un componente specifico, *cover detector*, provvede alla modifica della regola più vicina al *match* e con patrimonio di minor ammontare, così da garantire l'avviamento del meccanismo. In questo modo viene garantita l'attivazione di almeno una regola immettendo il minimo turbamento nel sistema, sia perché si modificano poche posizioni della regola, sia perché l'individuo modificato è, probabilmente, poco importante. Per immettere movimento

nel sistema le liste, di regole e messaggi, vengono mescolate casualmente prima di ricercare i *match*.

Il quarto passo consiste nella conduzione dell'asta: a tutte le regole presenti nella *matchList* viene chiesto di formulare un'offerta, *bid*, in base a parametri letti nel *classifierParm;* di seguito si procede a selezionare la regola che ha effettuato l'offerta maggiore; una volta individuata, questa viene rimossa dalla *matchList* ed immessa nella *winnerList*, precedentemente svuotata. La procedura viene ripetuta fino a selezionare un numero di vincitori pari alla quantità massima specificata dall'oggetto di gestione dei parametri. Per ridurre la possibilità di avere *bid* di eguale ammontare, ogni regola viene selezionata in base ad un valore, *effective bid*, ottenuto sommando al *bid* una quantità casuale, fra zero e uno, mantenuta sufficientemente piccola moltiplicandola per un valore specificato nei parametri.

A questo punto le regole vengono tassate di una quota proporzionale del loro patrimonio. I tipi d'imposta sono due: una riguarda tutte le regole, *life tax*, la seconda è pagata dalle sole regole che partecipano all'asta, *bid tax*. Le aliquote sono determinate interrogando l'oggetto di gestione dei parametri. Il gettito di queste imposte viene incassato dalla *treasury*.

Il sesto passo consiste nello svuotare la *messageList*, distruggendo i messaggi in essa contenuti. Questi ultimi sono infatti divenuti inutili: il loro testo ha già partecipato ad una sessione di *matching* e le regole attivate hanno già registrato, nella propria *partnerList*, l'indirizzo degli autori dei messaggi. Viene trasmesso, a tutte le regole presenti nella *winnerList,* l'ordine di pagare i rispettivi *bid* agli autori dei messaggi che ne hanno consentito l'attivazione, cioè alle regole presenti nella loro *partnerList*, e di emettere i propri messaggi. Ciascuna di queste regole comunica il contenuto della propria *action-part,* che viene immesso in un messaggio appena costruito, da aggiungere alla *messageList*.

In base al contenuto della nuova *messageList,* viene ricercato il *match* con gli *effector:* ciascuno di quelli che soddisfano il *match* provvede a registrare l'indirizzo della regola, proprietaria del messaggio, nella propria *partnerList* ed a sommare, al proprio valore di *support,* il *bid* della regola. Gli *effector* attivati dal *match* vengono immessi nella *matchList*, precedentemente svuotata. Se questa dovesse risultare vuota al termine del processo, verrebbe applicata un'apposita *routine* (*cover effector*), volta a garantire almeno un *match*. Tutti i messaggi utilizzati dagli *effector* vengono rimossi dalla *messageList* in modo che non possano, accidentalmente, attivare altre regole.

L'ultimo passo consiste nell'effettuare la scelta dell'*effector,* che avrà il diritto di consigliare la propria azione, in base al valore di *support*. L'indirizzo di questo *effector* viene registrato nel *dataWarehouse*. Viene richiamata l'*interface* affinchè registri il valore dell'azione, espresso, nella metrica interna, dall'*action-part* dell'*effector* vincitore.

Durante tutte queste fasi sono stati aggiornati i contatori statistici, inviando appositi messaggi al *dataWarehouse*.

3.4.3 – *Cover detector*, *cover effector* e regole cooperative

I primi due sono componenti del sistema incaricati di garantire i *match* necessari al funzionamento del medesimo. Il *cover detector* interviene quando nessun individuo risulta in grado di agire, a fronte dei messaggi provenienti dall'ambiente: come già descritto, si provvede a variare la *condition-part* di una regola onde garantire l'avviamento della procedura. CW fornisce, nelle statistiche, l'indicazione del numero di volte in cui la *routine* è stata applicata: un ricorso troppo frequente al *cover detector* potrebbe indicare un'errata impostazione della codifica del problema o la penuria di regole generiche, cioè contenenti *wildcard*. Queste regole sono estremamente utili nelle fasi iniziali dell'apprendimento perché permettono di ottenere comunque risposte, pur se imprecise; la gestione dei patrimoni e gli algoritmi di determinazione delle offerte garantiscono la loro progressiva sostituzione, mano a mano che regole più specifiche vengono individuate e sperimentate.

Il *cover effector* ha funzioni analoghe al *cover detector*, la sua azione, però, concerne il *match* delle condizioni degli *effector*. E' normale che esso venga utilizzato nelle fasi iniziali

dell'apprendimento, specie quando vi sia un alto rapporto fra numero di regole e numero di *effector* o quando il cromosoma, *array* contenuto nei vari oggetti, sia molto lungo. La funzione fondamentale è quella di garantire la risposta all'interrogazione dell'agente, onde permettere l'avviamento del processo.

Nell'esempio fornito da Goldberg (1989), la *action-part* delle regole contiene direttamente l'indicazione dell'azione da effettuare: non sono richiesti *effector* e, tanto meno, un *cover effector*. Quell'impostazione non permette, però, lo sviluppo di atteggiamenti cooperativi fra le regole: due regole cooperano quando il messaggio, *action-part*, immesso da una regola, contribuisce all'attivazione di un'altra. Con regole cooperative si vengono a formare legami fra gli individui che contribuiscono alla formazione della risposta; la conoscenza viene ripartita fra più regole in modo da conferire maggiore elasticità al meccanismo di reazione, che si forma con l'evoluzione della popolazione. Dalle sperimentazioni effettuate risulta che *classifier* con regole cooperative possono giungere più rapidamente a configurare comportamenti ottimali in ambiti complessi. La ripartizione della conoscenza fra più individui riduce il rischio di perdita totale della medesima; pur tuttavia, è bene dotare il sistema di un adeguato numero di regole, sufficiente a garantire una bassa probabilità di distruzione degli individui facenti parte di una catena.

3.4.4 - L'evoluzione delle regole

L'attività è affidata ad un oggetto della classe *RuleMaker*: anche questo, come il *ruleMaster*, è in grado di trattare diverse popolazioni, ricostruendo, di volta in volta, lo specifico algoritmo genetico da applicare, in base ai parametri utilizzati da ciascun *client*. Il *ruleMaker* interagisce esclusivamente con i *dataWarehouse*, leggendo e modificando i dati in essi contenuti. L'attivazione del programma è operata dal *ruleMaster,* che provvede anche a comunicare l'indirizzo del *dataWarehouse* su cui operare.

Dal *dataWarehouse* il programma ottiene gli indirizzi della lista delle regole e dell'oggetto di gestione dei parametri, per lo specifico *classifier*. I genitori sono selezionati a coppie: da ciascuna coppia vengono prodotti due nuovi individui, che vanno a sostituire i due peggiori, e più simili ai nuovi nati, scelti in un campione della popolazione. La procedura viene ripetuta fino a raggiungere la quota di ricambio generazionale specificata nei parametri. L'ampia dotazione di parametri permette di personalizzare, in modo diverso per ciascun agente, anche la fase evolutiva.

Il primo passo consta dell'acquisizione delle notizie relative alla popolazione su cui operare. Una volta ottenute le informazioni, viene calcolato il numero di riproduzioni da effettuare e l'ammontare della somma dei patrimoni di tutte le regole, ricchezza totale. A questo punto si avvia l'iterazione della procedura, ripetuta per ciascuna coppia; al termine di ogni riproduzione, i nuovi nati vanno a rimpiazzare due individui della generazione corrente.

I genitori vengono selezionati in base al proprio patrimonio, o forza, in modo stocastico. Si procede sommando i patrimoni delle regole fino ad ottenere un ammontare superiore ad un valore, *random*, compreso fra zero e la ricchezza totale; viene scelta l'ultima regola che ha partecipato alla somma. In questo modo ogni regola riceve una probabilità di riprodursi proporzionale all'ammontare del suo patrimonio: più la regola è ricca, cioè forte, più grande è la possibilità che l'addizione del suo valore comporti il superamento della soglia, casualmente stabilita. La generazione pseudo-casuale del valore di soglia avviene in base ad una distribuzione di probabilità uniforme. Il procedimento viene ripetuto per la selezione del secondo genitore, avendo cura di evitare che la scelta ricada sulla regola già individuata come primo genitore.

I patrimoni genetici dei due genitori vengono incrociati e assoggettati a mutazione, in base ai parametri validi per il *dataWarehouse* utilizzato; l'attività viene condotta in modo da preservare la distinzione fra *action-part* e *condition-part*: si incrociano separatamente le due parti. Il *crossover* viene effettuato generando, in modo pseudo-casuale, due interi, uno per la *condition-part* ed uno per la *action*, di valore compreso fra uno e la lunghezza di ciascuna parte, diminuita di un'unità. In base a questo valore i due genomi possono essere divisi, a loro volta, in due parti, sinistra e destra: ciascun

figlio riceve un cromosoma ottenuto copiando quello dei genitori e scambiando le parti sinistre dei due genomi. La copia viene effettuata per singole posizioni, prima di ogni copia si sottopone a mutazione l'allele, data la probabilità specificata in modo parametrico. La mutazione consiste nel cambiare il valore della singola posizione: da zero in uno, da uno in *wildcard* oppure da *wildcard* in zero.

Può accadere che nuovi individui risultino eguali ad altri già esistenti; in questo caso l'aggiunta della regola non comporterebbe nessuna novità effettiva. I due nuovi cromosomi vengono, perciò, sottoposti a verificazione, cioè a confronto con gli altri esistenti che ne certifichi l'originalità: se l'esito è negativo la procedura di generazione viene ripetuta, a partire dalla ricerca dei genitori, fino ad ottenimento di effettive novità. Solo a questo punto è possibile creare due nuovi oggetti della classe *Rule*, dotati dei nuovi cromosomi, con patrimonio iniziale pari alla media dei patrimoni dei genitori.

L'ultima fase consta della selezione dei due individui da rimpiazzare. Viene sostituita la regola che risulti, contemporaneamente, dotata di minor patrimonio e maggiormente simile a quella destinata a prendere il suo posto. La selezione viene condotta su una porzione della popolazione, ottenuta estraendo casualmente, dalla popolazione globale, il numero di individui determinato dal valore dei parametri *crowdingFactor* e *crowdingRate.* Moltiplicando per *crowdingFactor* il numero di regole, si ottiene la numerosità di un primo campione; ciascun individuo di questo gruppo viene confrontato con tutti quelli appartenenti ad un secondo campione, generato in base a *crowdingRate*. In ogni confronto viene scelta la regola dotata del minore patrimonio; le regole selezionate vengono, poi, confrontate con l'individuo da inserire nella popolazione; la più simile a quest'ultimo viene rimossa dalla *ruleList* e distrutta. La regola nuova viene accodata nella lista procedendo, in conseguenza, ad aggiornare il valore della sommatoria dei patrimoni. Il procedimento viene ripetuto per l'altro nuovo individuo.

Potrebbe accadere che una regola, fra quelle da rimuovere, risultasse proprietaria di un messaggio, ancora presente nella *messageList* del *classifier*: se questo messaggio dovesse contribuire ad attivare una regola, quella tenterebbe di ricompensare il proprietario del messaggio e, non trovando più l'oggetto, ormai distrutto, si verificherebbe un errore di programma. Per questo motivo, nei messaggi di proprietà della regola distrutta, l'indirizzo dell'*owner* viene sostituito con quello della *treasury*.

3.4.5 - I parametri

La versatilità di CW è legata alla possibilità di differenziare, per il tramite dei valori contenuti nei *file* di parametri, il comportamento dei diversi *classifier system* gestiti. A CW devono essere forniti tre tipi diversi di parametri: generali, del sistema di distribuzione dei crediti e dell'algoritmo genetico.

Fanno parte dei parametri generali: il numero di regole presenti nella popolazione, il numero di *effector*, la lunghezza del genoma, cioè di un singolo *array*. Il numero di regole deve essere sufficiente a garantire flessibilità e completezza di azione: poiché il numero di individui viene mantenuto costante durante l'evoluzione, un valore troppo ridotto potrebbe comportare una esplorazione troppo lenta dell'insieme delle soluzioni; inoltre, le nuove regole potrebbero finire per ricoprire regole valide costringendo il sistema ad imparare, nuovamente, cose che aveva già appreso in precedenza. Per quanto concerne la completezza, poiché regole altamente specifiche possono attivare, al massimo, un *effector*, dovranno essere previste almeno tante regole quante sono le azioni che ciascun agente deve poter compiere ad apprendimento avvenuto. Gli inconvenienti legati ad un numero troppo elevato di regole sono essenzialmente prestazionali: in diverse fasi occorre effettuare scansioni della *ruleList* e dialogare con tutti gli individui.

Il numero di *effector* deve essere pari al numero di azioni che ciascun agente è in grado di effettuare; si noti che si parla di azioni in generale e non di azioni corrette: quando i termini del problema da indagare fossero poco conosciuti, potrebbe essere conveniente abilitare un gran numero di azioni osservando, poi, quali verrebbero effettivamente utilizzate ad apprendimento avvenuto; in questo caso si potrebbero avere più *effector* che regole. Il numero di *effector* dipende

dall'impostazione del modello: agenti che debbano decidere se comprare o vendere un singolo bene potrebbero essere guidati da un *classifier* con due soli *effector;* se la decisione dovesse riguardare un particolare titolo, scelto fra quelli disponibili, il numero di *effector* dovrebbe essere pari al doppio del numero di titoli.

La lunghezza del genoma dipende dal numero e dalla tipologia dei dati da gestire e dal protocollo di codificazione adottato. Poiché tutti gli *array*, contenuti nelle regole, nei messaggi e negli *effector,* hanno eguale estensione, occorrerà determinare il parametro tenendo conto della maggior lunghezza necessaria. Gli elementi dell'*array* possono essere utilizzati per rappresentare lo stato di singole caratteristiche ambientali o informazioni relative al verificarsi di determinati eventi: zero indicherebbe che la condizione è falsa o l'evento non si è verificato, uno il contrario e *wildcard* la mancanza di informazioni. Ogni regola rappresenterebbe una strategia, più o meno complessa, e la lunghezza del genoma dovrebbe essere pari al numero massimo di condizioni che ciascuna regola dovesse gestire. Un caso diverso si avrebbe quando l'informazione ambientale fosse una grandezza: il valore potrebbe essere tradotto in un numero naturale rappresentabile, nella codifica binaria, come sequenza di zero ed uno, quindi la lunghezza del cromosoma dipenderebbe dalla massima estensione delle grandezze trattate e dal livello di precisione voluto. Le *wildcard*, in questo caso, servirebbero per identificare, con un solo genoma, un sottoinsieme del dominio delle possibili grandezze: un genoma formato di sole *wildcard* seguite, in ultima posizione, da uno zero, identifica tutte le grandezze pari, espresse nella metrica del *classifier*.
CW è predisposto per accettare una lunghezza massima di 256 posizioni per ogni *array*, il che significa gestire circa $1{,}16 * 10^{77}$ stati diversi dell'ambiente, fornendo un numero di risposte estensibile fino alla stessa quantità; all'atto pratico possono essere sviluppate e verificate $1{,}34 * 10^{154}$ regole diverse, quando ogni posizione indichi lo stato di una condizione ambientale o l'esecuzione di una azione. Nei casi in cui il numero risultasse eccessivamente restrittivo, l'utilizzatore potrà elevare il limite, intervenendo nei *file* con estensione "h" delle classi contenenti *array:* l'intervento si riduce alla sostituzione della grandezza 256 con il nuovo limite, nelle istruzioni di definizione degli *array*.

Nella categoria di parametri generali sono previsti altri cinque valori, usati per influenzare la conduzione dell'apprendimento: dimensione della *message list*, *effectorsFlag*, proporzione di *wildcard* nella popolazione iniziale, livello di confidenza, forza iniziale degli individui.

La dimensione della *message list* specifica il numero massimo di messaggi che può essere prodotto in ogni ciclo: se tale grandezza viene fissata ad uno la collaborazione fra regole risulta, implicitamente, inibita. La componente *cover effector,* infatti, garantisce che almeno un messaggio venga utilizzato da un *effector*, questo messaggio sarà rimosso dalla *message list* e, quindi, all'inizio di ciascun ciclo, la medesima risulterà sempre vuota: ciò impedisce che una regola ne possa attivare un'altra. Permettendo la produzione di un solo messaggio per ciclo, le regole, che verranno sviluppate, saranno specializzate nel fornire risposte dirette, date le situazioni descritte nella loro *condition-part*; il sistema sarà caratterizzato da estrema semplicità di azione, ma richiederà regole piuttosto lunghe per affrontare situazioni complesse: i tempi di evoluzione e apprendimento potrebbero risultare maggiori, come già detto, rispetto ad un sistema basato sulla cooperazione di più individui semplici. La produzione di un gran numero di messaggi, per contro, favorirà la formazione di catene fortemente ramificate, la cui interpretazione potrà risultare piuttosto difficoltosa.

L'*effectorsFlag* può assumere i valori zero o uno: uno indica che il *classifier* deve agire secondo l'esempio di Goldberg, citato a proposito delle regole cooperative. Se l'*effectorsFlag* vale uno, durante la generazione iniziale della popolazione ad ogni regola viene assegnata, come *action-part*, la *condition-part* di un *effector*: in questo modo il sistema si comporta come se la *action-part* contenesse direttamente l'indicazione dell'azione da suggerire. Il r*uleMaker*, in questo caso, esegue le operazioni di *crossover* e mutazione solo sulla *condition part* della regola: le *action-part* vengono ereditate direttamente dai genitori senza modificazioni. In questo caso il programma formula regole di associazione fra stati dell'ambiente ed una specifica azione: la *condition-part* può contenere *wildcard* mentre la *action* no. Con *effectorsFlag* a zero, anche la *action-part* può essere generalizzata con *wildcard* e, quindi, è possibile associare più stati a più azioni, la cui esecuzione risulti apportatrice di risultati equivalenti . Con valore uno di *effectorsFlag* la cooperazione fra regole è inibita, in quanto tutti i messaggi sono utilizzati dagli *effector*, e quindi rimossi dalla *messageList*, nel corso di ciascun ciclo. Questa modalità di utilizzo fornisce risultati interessanti nella ricerca di soluzioni precise, da

applicare in ambienti piuttosto stabili; le regole ottenute sono di immediata interpretazione ed il sistema, ad apprendimento avvenuto, somiglia molto ad un sistema esperto basato su regole pre-codificate. La superiorità rispetto ai tradizionali sistemi esperti sta sia nella capacità iniziale di adattamento, sia nella possibilità di modificare le regole a fronte delle mutazioni ambientali che dovessero intercorrere successivamente. A fronte di problematiche complesse, o quando si dispongano di scarse informazioni sul funzionamento dell'ambiente, è consigliabile ammettere maggior libertà d'azione del sistema, valorizzando a zero il parametro.

La proporzione di *wildcard* serve a stabilire il grado di generalità della popolazione iniziale; il suo valore può essere fissato fra zero ed uno. Durante la generazione casuale della popolazione, viene assegnato, ad un gene, l'allele "#" ogni volta che la generazione di un numero casuale, compreso fra zero ed uno, fornisce un valore inferiore a quello del parametro. Negli altri casi viene assegnato, con eguale probabilità, zero oppure uno.
La presenza di *wildcard* nella popolazione iniziale aumenta la probabilità di *match* e agevola l'avviamento del processo evolutivo, CW è, però, in grado di evolvere popolazioni dove non figuri nessuna *wildcard,* sfruttando il *cover detector* ed il *cover effector*: è, quindi, ammesso fissare a zero il parametro in oggetto. Il sistema potrebbe, teoricamente, evolvere anche con un valore uno: tutte le regole risulterebbero capaci di effettuare il *match* con qualunque condizione ed immetterebbero messaggi capaci di attivare qualunque *effector;* le risposte del sistema sarebbero totalmente casuali, ma, con la mutazione, potrebbero essere introdotti, nei genomi, valori zero ed uno che potrebbero, progressivamente, ridurre la casualità delle risposte. Valori estremi sono, naturalmente, del tutto sconsigliati: buoni risultati sono ottenibili, per un gran numero di applicazioni, con proporzioni nell'ordine del venti o trenta percento.

Il livello di confidenza esprime la quota di errore considerata trascurabile, il *ruleMaster* considera errate le azioni che abbiano fruttato una ricompensa pari a zero: quando il rapporto fra azioni consigliate e azioni errate, nel periodo di osservazione, risulti inferiore al livello di confidenza, viene sospesa la ricerca di nuove regole, cioè viene inibita la chiamata al *ruleMaker*. La limitazione verrebbe rimossa, automaticamente, qualora il rapporto dovesse, nuovamente, superare il livello di confidenza: in questo modo si conferisce carattere di maggior stabilità all'azione del *classifier* che, una volta raggiunto un buon livello di apprendimento, continua ad applicare le regole collaudate. Un simile comportamento risulta giustificabile, in termini di plausibilità, immaginando come, anche un essere umano, tenda, una volta raggiunto il livello di risultati desiderato, a mantenere la strategia adottata, pur restando, sensibile alle mutazioni delle condizioni a contorno, pronto ad intraprendere quelle azioni correttive che garantissero il mantenimento dei risultati. L'accorgimento descritto costituisce un elemento di novità nella gestione dei *classifier*; esso risulta utile, soprattutto, per le applicazioni volte a determinare il comportamento di agenti: la possibilità di ammettere un livello di errore, della dimensione voluta, consente di realizzare entità, dotate di razionalità limitata, il cui comportamento appare maggiormente realistico di quello di agenti perfettamente ottimizzanti. Nel caso si volesse operare, tradizionalmente, con un sistema in continua evoluzione, sarà sufficiente specificare un valore negativo: essendo zero il minimo numero di errori raggiungibile, la condizione di inibizione della chiamata al *ruleMaker* non verrebbe mai raggiunta e, quindi, il meccanismo di limitazione non potrebbe mai essere attivato.

L'ultimo parametro generale specifica l'ammontare del patrimonio iniziale delle regole: il valore viene usato solo nel caso di generazione casuale delle medesime e, quindi, nell'impossibilità di stabilire a priori, quali individui siano dotati di maggior potenzialità, lo stesso valore viene assegnato a tutti. Quando la popolazione iniziale venga codificata in precedenza, è possibile assegnare valori diversi a ciascun individuo.

3.4.6 - I parametri del sistema di distribuzione dei crediti

La seconda tipologia di parametri riguarda, specificamente, la distribuzione dei crediti; essa comprende: le aliquote unitarie della *bidTax* e della *lifeTax* ed i coefficienti di determinazione dell'offerta effettuata, da ciascuna regola, nell'asta per ottenere il diritto di immettere il proprio messaggio nella lista.

Ad ogni ciclo vengono riscossi due tipi di tributi: *bidTax*, a carico di tutte le regole che partecipano all'asta, e *lifeTax*, cui è soggetta l'intera popolazione. Scopo della prima imposta è quello di rappresentare un costo di partecipazione all'asta, onde accelerare il decadimento del patrimonio delle regole sbagliate; la seconda serve a far decadere quelle regole che, possedendo una condizione che non trova riscontro nelle configurazioni ambientali, non vengono mai applicate e sono, perciò, inutili. Le due imposte sono proporzionali: l'applicazione consiste nel trasferimento della quota di patrimonio, indicata dal parametro, dalle singole regole alla *treasury*. Normalmente l'aliquota della *bidTax* è superiore a quella della *lifeTax*: presupposto dell'applicazione della prima è l'attivazione della regola, questa, perciò, ha la possibilità di recuperare, tramite la ricompensa ricevuta, l'ammontare sottratto; se ciò non avviene è perché la regola non rappresenta una buona soluzione del problema e, quindi, un rapido decadimento del suo patrimonio, aumentando le probabilità che l'individuo venga rimpiazzato, contribuisce ad elevare le capacità della popolazione. Nell'ipotesi di regole cooperative, la ricompensa può provenire da un'altra regola, invece che dall'agente; ogni regola paga, ai proprietari dei messaggi che ne hanno consentito l'attivazione, un ammontare proporzionale al proprio patrimonio: un individuo, allora, riesce a recuperare la *bidTax* se in grado di attivare regole corrette, che a loro volta ricevano una ricompensa elevata dall'agente. L'informazione, implicita nell'ammontare della ricompensa, relativa alla bontà della regola si propaga, per il tramite del meccanismo di distribuzione dei crediti, a tutta la catena di cooperazione.
La *lifeTax* agisce in modo da favorire il decadimento delle regole inutili, ma tale caratteristica emerge esclusivamente durante l'elaborazione; risulta, perciò, opportuno mantenere un'aliquota, piuttosto ridotta, tendenzialmente di ammontare inversamente proporzionale al numero di possibili stati dell'ambiente: per stabilire, sia pure in modo stocastico, che una regola non verrà mai attivata, occorrere offrire a ciascun individuo la possibilità di confrontarsi con un congruo numero di situazioni. Un'aliquota troppo elevata, inoltre, potrebbe comportare un eccessivo impoverimento di regole che, pur corrette, venissero applicate di rado, perché atte a rispondere a situazioni non particolarmente frequenti. In conclusione la *lifeTax* dovrebbe essere tanto più ridotta quanto più alta fosse la dinamica dell'ambiente e quanto più irregolare fosse la distribuzione dei vari accadimenti nel tempo.

Ogni regola attivata partecipa all'asta, con un'offerta, in ragione delle proprie disponibilità patrimoniali e della propria specificità. Il patrimonio costituisce un indicatore della bontà della regola, la specificità fornisce una misura della sua precisione. La funzione fondamentale di regole poco specifiche, contenenti molte *wildcard*, è quella di assicurare i *match* e, conseguentemente, di permettere al *classifier* di fornire una risposta, pur se relativamente imprecisa; mano a mano che vengono prodotte regole più specifiche, queste devono essere favorite nell'azione e, conseguentemente, nella riproduzione. Un simile effetto può essere ottenuto penalizzando le regole con bassa specificità nella conduzione dell'asta: l'offerta di ciascuna regola viene calcolata in base ad una trasformazione lineare, basata sul valore di specificità, della misura del patrimonio della regola stessa. Questo algoritmo, perfettamente deterministico, può essere migliorato, introducendo ulteriore movimento, tramite l'aggiunta di una piccola quantità casuale all'ammontare calcolato per ciascuna regola; l'accorgimento citato permette, inoltre, di evitare che vengano presentate offerte di eguale entità. Nella versione presente di CW, la distribuzione di probabilità della componente casuale è rigidamente normale ma, nell'ottica di implementazioni future sono stati previsti i parametri per influire sulla distribuzione citata.
L'offerta in asta costituisce l'ammontare che le regole vincitrici devono pagare agli autori dei messaggi che ne hanno consentito l'attivazione, cosiddetti "*partner*"*;* volendo conferire al prodotto elevata versatilità, sono stati predisposti accorgimenti affinchè questo importo possa differire da quello considerato nella conduzione dell'asta.
Ogni regola è capace di calcolare un *effectiveBid*, valido per l'asta, come segue:

$$EB = \beta * (\alpha + \delta * s) * S + R * \sigma + \mu$$

EB rappresenta l'ammontare offerto in asta, s la specificità relativa della regola, numero di *wildcard* diviso lunghezza totale del cromosoma, S è l'ammontare del patrimonio, o *strength*, ed R è un numero pseudo-casuale compreso fra zero e uno, generato in base ad una distribuzione uniforme di probabilità. Le lettere greche rappresentano i valori assegnati ai parametri: *bidRatio*, *effectiveLinearBid1*, *effectiveLinearBid2*, *bidSigma*, *bidMu*.

L'ammontare della ricompensa, da pagare ai *partner*, quindi il costo effettivo sostenuto dalla regola per acquisire il diritto di immettere un messaggio nella lista, è dato da:

$$B = \beta * (\alpha' + \delta' * s) * S$$

Le misure di α' e δ' sono fornite valorizzando, rispettivamente, i parametri: *linearBid1* e *linearBid2*.

I parametri del sistema di distribuzione dei crediti devono essere valorizzati, in modo significativo, quando si voglia utilizzare CW come *learning* o *non-learning classifier*; per utilizzare CW come motore di inferenza per sistemi esperti tradizionali, il valore di tutti questi parametri deve essere posto a zero.

3.4.7 - I parametri dell'algoritmo genetico

L'ultimo gruppo di parametri serve per condizionare l'azione dell'algoritmo genetico, gestito dal *ruleMaker*. Essi permettono di stabilire: la probabilità di attivazione dell'algoritmo, la quota della popolazione soggetta a *turnover*, le probabilità di *crossover* e di mutazione, la numerosità dei campioni di individui da utilizzarsi nella ricerca delle regole, appartenenti alla generazione corrente, da sostituire con le nuove.

Il *ruleMaker* viene attivato dal *ruleMaster* con la probabilità specificata tramite il parametro *evolutionRate,* che può assumere i valori reali compresi nell'intervallo]0,1[. I valori interni all'intervallo sono utilizzati come indicatori della probabilità, unitaria, di attivazione dell'algoritmo di ricerca di nuove regole: un numero *random* viene confrontato con la soglia prescelta e, se risulta inferiore, viene attivato il *ruleMaker*. Specificando zero l'algoritmo genetico non viene mai attivato e CW opera come *non-learning classifier*. Il valore uno ha un significato convenzionale: il *ruleMaker* viene attivato dopo un numero di cicli di selezione pari al numero delle regole contenute nel *dataWarehouse* utilizzato; in questo modo, se la probabilità che le diverse condizioni ambientali si presentino è uniformemente distribuita, tutte le regole, appartenenti ad una generazione, hanno la possibilità di essere attivate prima dell'evoluzione successiva.

Nel decidere la probabilità di attivazione, è importante ricordare che l'algoritmo genetico agisce in base all'ammontare del patrimonio delle singole regole, misura della bontà di ciascuna; tale valore viene stabilito, all'atto della generazione di un nuovo individuo, come media dei corrispondenti valori dei genitori e diviene effettiva misura della correttezza della regola solo dopo un congruo numero di cicli di elaborazione del *ruleMaster*. Se l'evoluzione avvenisse troppo di frequente, l'algoritmo genetico, operando su valori poco significativi, potrebbe portare all'ottenimento di regole, complessivamente, meno adattate all'ambiente.

In ogni evoluzione vengono prodotti un numero di nuovi individui, e conseguentemente rimpiazzate altrettante regole delle vecchie generazioni, determinato applicando il *turnoverRate* al numero di individui presenti nel *dataWarehouse*; per ragioni di elaborazione tale numero viene mantenuto, con opportuni arrotondamenti, pari e superiore a zero.

Il ricambio della popolazione deve essere effettuato in misura tale da conservare le conoscenze acquisite, pur arricchendo, con nuove regole, il patrimonio normativo contenuto nei vari *dataWarehouse*: è importante tener conto della quantità di situazioni ambientali diverse da fronteggiare e della dinamica che caratterizza la loro variazione. Un ricambio eccessivo potrebbe portare alla perdita di regole atte a rispondere ad eventi che si manifestassero raramente. In quest'ottica, può essere opportuno prevedere che ogni *dataWarehouse* contenga un numero di regole superiore al numero di situazioni da gestire: in questo modo si viene a disporre di un maggior spazio per l'immissione di nuovi individui, compatibile con un tasso di rinnovamento della popolazione sufficientemente elevato, capace di garantire una rapida esplorazione dell'insieme delle soluzioni.

Crossover e mutazione determinano la differenziazione dei nuovi individui, rispetto ai

genitori; osservazioni, svolte nel capitolo precedente, hanno illustrato come le due operazioni influiscano sulle probabilità di sopravvivenza di uno schema e, perciò, le probabilità di applicazione, di ambedue, andranno fissate tenendo conto del *trade-off* tra necessità di innovazione e importanza di preservare gli schemi apportatori di buoni risultati. Solitamente la probabilità di mutazione viene tenuta a livelli molto bassi, nell'ordine del millesimo; il *crossover* è applicato, invece, con probabilità tendenzialmente superiori al cinquanta percento.

La selezione degli individui per l'estinzione è basata sull'identificazione della regola dotata di minor patrimonio e, contemporaneamente, più simile a quella destinata a prendere il suo posto. Tale ricerca viene condotta su un campione della popolazione, formato in modo casuale, la cui numerosità è determinata applicando i coefficienti *crowdingRate* e *crowdingFactor* al numero di individui presenti nel *dataWarehouse*: per ogni individuo, del campione determinato in base a *crowdingRate.* viene effettuato il confronto con tutti gli individui di un altro campione, di ampiezza dipendente da *crowdingFactor*, per scegliere quello che possieda il minor patrimonio. Fra le regole selezionate, viene destinata ad estinzione quella più simile alla nuova regola. Il procedimento tende a divenire molto oneroso, a parità dei valori dei due parametri, quando il numero di regole risulti piuttosto elevato: *crwdingRate* e *crowdingFactor* devono essere specificati in ragione del numero di individui da trattare.

Volendo conferire all'algoritmo genetico un comportamento maggiormente lamarkiano, è possibile valorizzare a zero i due parametri: in questo caso, il programma sceglie in modo casuale la regola da sopprimere per far posto alla nuova.

3.4.8 - Valori di *default* e variazione dei parametri durante l'elaborazione

La classe *ClassifierParm* provvede ad assegnare, automaticamente, ai vari parametri, valori di *default*; ciò consente di utilizzare immediatamente il prodotto, rinviando l'attività di valorizzazione dei parametri a quando sia stata acquisita sufficiente esperienza e dimestichezza col metodo. L'utilizzatore dotato di poca confidenza può limitarsi ad impostare il numero di *effector* e la lunghezza del genoma. E' facoltativa anche la predisposizione del *file* dei parametri: i due citati possono essere valorizzati inviando appositi messaggi, di "*set*", all'oggetto *classifierParm*. Con i valori di *default,* CW opera come *learning classifier*, non limita l'evoluzione, attiva l'algoritmo genetico in modo probabilistico ed effettua la manipolazione di ambedue le parti, *condition* ed *action*, delle regole.

La classe *ClassifierParm* è dotata di tutti i metodi di "*set*", cioè di valorizzazione, necessari per modificare qualunque parametro: è possibile variare il comportamento del prodotto durante l'elaborazione; se viene modificato il numero di regole, o il numero di *effector* o la lunghezza del genoma, occorre rigenerare la popolazione; il *dataWarehouse* permette di eseguire tale attività durante l'elaborazione. Sfruttando questi servizi è possibile effettuare interessanti sperimentazioni, in cui il livello di autonomia degli agenti può essere innalzato fino a dotarli di algoritmi per modificare, metaforicamente, oltre che il proprio pensiero, anche il proprio modo di formare il medesimo. Agenti particolarmente abitudinari, che utilizzano un *classifier* con bassa frequenza di esecuzione dell'algoritmo genetico e tassazione delle regole particolarmente mite, potrebbero divenire, a fronte di determinati eventi, maggiormente dinamici innalzando i parametri suddetti. Altri, che non ritraessero buoni risultati dalle strategie elaborate, potrebbero decidere di effettuare un cambiamento radicale, rigenerando l'intera popolazione di regole. Queste azioni dovrebbero essere codificate nell'*interface* onde salvaguardare l'indipendenza fra agenti e *ruleMaster*; la decisione di attuarle potrebbe essere presa in base al suggerimento di un altro *ruleMaster:* dotato di regole fisse, operante con metodi genetici, con reti neurali o con quant'altro offerto dalle tecniche di intelligenza artificiale.

Si potrebbe anche assegnare ad ogni agente: un *dataWarehouse* normativo, contenente regole relative alla valorizzazione dei parametri di funzionamento del *classifier,* un *dataWarehouse* operativo, contenente le regole per determinare i comportamenti dell'agente nei confronti dell'ambiente, ed un *classifierParm* operativo, dove fossero riportati i parametri di funzionamento, del *classifier* funzionante col *dataWarehouse* operativo. Naturalmente occorrerebbe anche un *classifierParm* normativo che, però, potrebbe essere condiviso da più agenti. L'agente opererebbe,

normalmente, richiedendo al *ruleMaster* consigli, basati sul *dataWarehouse* operativo, per sapere quali azioni effettuare nell'ambiente; con una qualche cadenza, verrebbe richiesto al *ruleMaster* di agire sul *dataWarehouse* normativo, per stabilire quali variazioni apportare ai parametri del *classifierParm* operativo. Si noti che, anche in questo caso, il modello funzionerebbe con una sola istanza di *ruleMaster* e *ruleMaker*, bastando la differenziazione dei *dataWarehouse* a separare compiti e azioni dei due *classifier*.

3.4.9 - Metrica del *classifier*, metrica dell'agente e ricompense

Usando CW, l'azione di traduzione eseguita dall'*interface* permette di distinguere agevolmente la metrica esterna, o dell'agente, da quella interna, o del *classifier*. Utilizzando metodi genetici, ciò assume una notevole importanza, poiché algoritmi genetici e *classifier system* operano sulla rappresentazione dei dati, a prescindere dal significato semantico attribuito dall'agente che, nella struttura ERA, è noto solo all'*interface*. In questo modo l'agente continua ad operare in base alla propria percezione degli eventi ambientali, indicandoli con i costrutti grammaticali preferiti dal programmatore, ed esegue azioni codificate in modo analogo; l'*interface* si preoccupa di tradurre queste informazioni nella forma di parole, sequenze di simboli, definite sull'alfabeto {0, 1, #}. Si immagini, ad esempio, di avere agenti che debbano decidere se acquistare o vendere titoli, nella conduzione di speculazioni sui corsi dei medesimi. Ogni agente sia in possesso di stime relative all'andamento a breve, prezzi di apertura della borsa il giorno successivo, e a termine, prezzi previsti per il prossimo giorno dei riporti. Le stime riguardino esclusivamente i tre casi di aumento, riduzione o stabilità dei corsi. Queste informazioni potrebbero essere tradotte, dall'*interface,* in parole di quattro caratteri, i primi due rappresenterebbero le stime a breve e gli altri quelle a termine: "00" potrebbe indicare stabilità, "01" aumento e "10" riduzione. Quando l'agente prevedesse riduzione a breve seguita da un aumento a termine, l'*interface* comunicherebbe l'informazione al *classifier* con la parola "1001", contenuta in un *array* di 4 elementi, ciascuno lungo un carattere.

Le azioni possibili siano, per semplicità, tre: non operare con quel titolo, azione zero, acquistare a termine, azione uno, o vendere a termine, azione due, tutte eseguite al prezzo della borsa del giorno seguente. Se il *classifier* a fronte di "1001" consigliasse l'azione "0010", tradotta dall'*interface* in "vendere a termine" (si ricordi che il numero binario 0010 corrisponde al numero decimale 2), l'agente subirebbe una perdita, quindi fornirebbe al *classifier* una ricompensa nulla: la regola "1001:0010" sarebbe destinata a decadere. Risulta intuitivo come la regola "1001:0001" sarebbe destinata ad aumentare il suo patrimonio come, probabilmente, la: "0110:0010". In pratica il *classifier* impara ad associare due genomi, rappresentanti condizioni ed azioni correlate, in modo da massimizzare le ricompense ottenute. Se l'*interface*, per un altro agente, venisse codificata in modo da associare ad uno l'azione vendere e, a due, acquistare, oppure a"01" riduzione del corso e a "10" aumento, il *classifier* imparerebbe ad associare "0001" a "0110" e "0010" a "1001". E' anche possibile che risultino corrette regole generalizzate, contenenti caratteri *wildcard*: con riferimento all'esempio, è palese l'inutilità di speculare su un titolo per il quale si abbiano attese di stabilità del corso a termine, indipendentemente dall'andamento a breve. In questo caso è probabile che la regola "##00:0000" risulti buona ed il *classifier* non abbia bisogno di affinare il suo apprendimento fino a scoprire "0000:0000", "0100:0000" e "1000:0000".

Si potrebbe affermare che il metodo di ricerca del *classifier* consente di raggiungere un buon risultato, indipendentemente dal significato semantico del patrimonio genetico della popolazione su cui opera, grazie all'informazione costituita dal valore della ricompensa. Ciò ribadisce l'importanza della funzione di calcolo della ricompensa: essa deve garantire che l'ammontare attribuito al *classifier* rifletta il grado di soddisfazione dell'agente o la misura del raggiungimento dei suoi obiettivi. L'esperienza pratica porta a sconsigliare la codificazione di funzioni molto complesse, che rischierebbero di introdurre elementi in grado di influenzare il sistema, evitando, però, l'eccesso opposto, dato dall'utilizzo diretto dei valori assoluti di determinati aggregati, utile, patrimonio o altro, che, specie se molto variabili, potrebbero creare eccessive sperequazioni nella distribuzione della ricchezza delle regole, con conseguenti fenomeni di dominanza da parte degli individui più favoriti. Quanto più la ricompensa dipende dalle variazioni del grado di soddisfazione dell'agente, che siano stretta conseguenza del suo comportamento, tanto più rapido e preciso è l'apprendimento;

l'introduzione di elementi di disturbo della percezione conferisce, tuttavia, un maggior grado di realismo alla simulazione. La presenza di questi elementi risulta, infine, inevitabile quando la simulazione sia utilizzata allo scopo di ottenere informazioni sul funzionamento di un sistema poco conosciuto.

3.4.10 - Il *workbench*

CW contiene un'applicazione volta ad agevolare la sperimentazione di diverse configurazioni di parametri e la realizzazione dei modelli; da queste funzioni la denominazione di *workbench.* Tramite l'applicazione, è possibile simulare problemi di complessità diversa e, quindi, sperimentare il comportamento del *classifier* prima di scrivere il modello nella sua completezza.

Precedentemente si è osservato come un *classifier* impari, al livello della sua metrica, ad associare sequenze di zero ed uno, la *wildcard* serve essenzialmente a raggruppare queste sequenze in classi rappresentabili con un solo individuo, ad una azione; l'associazione passa per il tramite degli *effector* e, in caso di regole cooperative, anche per il tramite di altre regole, ma ciò non modifica i termini del ragionamento. Le varie regole, possono essere lette come coppie di numeri interi, rappresentati in forma binaria: "1001:0001" può essere letta come "se 9 allora 1". Problemi molto complessi necessitano, naturalmente, di genomi piuttosto lunghi, per poter codificare tutti gli eventi, o le grandezze, che configurano le diverse condizioni. Una situazione di utilizzo, di complessità voluta, può essere approssimata usando numeri, generati in modo pseudo-casuale, da associare ad altri numeri, prestabiliti, in base ad una qualche legge: il *classifier* impara le associazioni fra i numeri, non la legge, ogni numero rappresenta un insieme di informazioni, non un valore, per il *classifier*. Variando la grandezza dei numeri è possibile modificare l'onerosità del compito assegnato al sistema.

Il *workbench* di CW utilizza direttamente operazioni su numeri naturali, generati in modo pseudo-casuale: i risultati delle prime fungono da termini di raffronto per stabilire la correttezza della risposta, ricevuta dal *classifier;* i secondi costituiscono la descrizione delle condizioni, comunicata al *classifier*. Il *classifier* impara ad associare condizioni, espresse dalla rappresentazione binaria del numero casuale, con azioni, rappresentazione binaria del risultato della trasformazione. Il fatto che, data la metrica usata dall'agente, i dati vengano tradotti dall*'interface* in numeri naturali è strettamente legato a motivazioni di comodità di gestione: un *classifier* capace di associare, correttamente, ai primi n numeri naturali, m risultati della trasformazione dei medesimi, è genericamente capace di consigliare m' azioni a fronte di n' stati dell'ambiente, dove i valori di m' ed n' dipendono dalla metrica dell'agente. Nell'esempio fatto in precedenza, le combinazioni dei dati, relativi agli andamenti attesi, possono generare nove stati diversi dell'ambiente, cioè delle aspettative riguardanti il corso di un certo titolo: "0000", "0001", "0010", "0100", "0101", "0110", "1000", "1001", "1010"; il *classifier* è chiamato ad associare, a quelli, tre azioni diverse: "0000", "0001","0010". Nella metrica del *workbench* ciò significherebbe che i numeri zero, uno, due, quattro, cinque, sei, otto, nove e dieci devono essere trasformati nei numeri zero, uno e due, in base ad una o più regole: l'inutilità della speculazione su titoli a corso stabile, ad esempio, aveva portato ad ipotizzare la regola "##00:0000" che potrebbe essere simulata, nel *workbench*, trasformando i numeri divisibili per quattro in zero.

Nel *workbench* la simulazione è basata sull'interazione fra un numero variabile di agenti, ciascuno dotato di un proprio *dataWarehouse*, e un'istituzione centrale denominata *lottery*. L'azione dell'agente consiste nel richiedere informazioni alla *lottery*, ottenute sotto forma di un numero naturale, scelto, in modo casuale, nell'intervallo stabilito dall'utilizzatore; questo numero viene fornito, tramite la traduzione eseguita dall'*interface*, al *ruleMaster* come informazione ambientale. Il *ruleMaster* decide quale azione effettuare tra quelle possibili, anche il loro numero è stabilito dall'utilizzatore, e la comunica all'*interface* che la traduce in un numero da comunicare all'agente. L'agente utilizza il numero ricevuto per "scommettere" che il risultato della trasformazione, operata dalla *lottery*, coincide con esso; naturalmente, riceve un premio in caso di vittoria e, allora, comunica una valutazione positiva al *ruleMaster*.

La *lottery*, quando riceve la richiesta di informazioni dell'agente, genera un numero, *random*, compreso fra zero e il numero di situazioni possibili, calcola il resto della divisione del numero

generato, aumentato di una quantità di *bias*, per il numero di azioni, registra il risultato nell'elemento, relativo all'agente, di un apposito *array*, e restituisce il valore del numero generato. Al momento della scommessa, la *lottery* confronta il numero, proposto dall'agente, con quello registrato nell'*array* dei risultati e paga la ricompensa se i due numeri coincidono.
La *lottery* tiene il conto delle vincite consecutive, effettuate da ciascun agente: quando questo supera una soglia, decisa dall'utilizzatore, modifica il valore del *bias*, da zero in uno o viceversa; il cambiamento costringe gli agenti, e quindi i vari *classifier*, a modificare le proprie regole per adattarsi nuovamente all'ambiente.

Immaginando che ogni posizione del genoma, risultante dalla traduzione del numero fornito dalla *lottery*, rappresenti un evento, per il quale il valore uno potrebbe indicare che il fatto è avvenuto, mentre zero significherebbe il contrario, è possibile affermare che il *workbench* permette di simulare applicazioni in cui un *classifier* deve prendere decisioni, relativamente ad un insieme di condizioni, stati dell'ambiente, rappresentate dalla combinazione del verificarsi, o meno, di un numero dato di eventi. In altri casi, si potrebbe leggere il genoma come formato dall'accostamento di un certo numero di dati quantitativi, ad esempio prezzi, espressi in forma binaria e, quindi, il *workbench* simulerebbe un modello in cui gli agenti dovessero decidere basandosi sulla conoscenza di serie storiche. Se il genoma venisse letto come risultante dall'accostamento di informazioni quantitative sul valore nutritivo di certi beni e sul loro prezzo, la simulazione potrebbe riguardare una decisione di consumo. Le diverse interpretazioni possibili, molto più numerose degli esempi forniti, possono essere utilizzate in riferimento a parti di uno stesso genoma che, allora, potrebbe includere informazioni, del tipo “vero o falso”, relative ad accadimenti o condizioni, e dati quantitativi. Per il *kernel* di CW, a variare sono semplicemente la lunghezza del genoma ed il numero di azioni. Ciò non esclude che l'ottenimento di buoni risultati sia condizionato al corretto dimensionamento dei vari parametri, primi fra tutti, numero di regole e frequenza di chiamata dell'algoritmo genetico.

Le differenze fra gli esempi di simulazione, citati in precedenza, emergono vigorosamente al livello del *front-end* e dello *user*, cioè nella diversità di azione delle componenti *interface*, agenti e ambiente. Il *workbench* può, allora, servire per effettuare una prima esplorazione dei risultati ottenibili utilizzando CW, simulando il livello di complessità del modello specifico, e per affinare, tramite sperimentazioni dirette, la configurazione di base dei parametri, particolarmente di quelli dimensionali come: numero di regole, di azioni, e così via. Una volta ottenuto un buon livello di risultati dal *workbench*, si potrà procedere alla sostituzione dello strato *user* con quello del modello specifico e a modificare le componenti dello strato di *front-end* di CW. L'esecuzione del modello potrà fornire risultati, anche in relazione alle prestazioni del *kernel*, non coincidenti con quelli ottenuti col *workbench,* che richiederanno ulteriori interventi di regolazione "fine" dei valori dei parametri. Si consideri, in particolare, che la transizione tra i vari stati ambientali, rappresentati, per il *workbench*, da un numero casuale, avviene in base ad una distribuzione di probabilità uniforme: i vari stati tendono a presentarsi con una stessa frequenza; ciò non è assolutamente garantito in un modello di simulazione di specifiche realtà. In relazione a questa diversità, potrebbe risultare necessario, nel passaggio dalla simulazione col *workbench* al modello vero e proprio, ridurre, ad esempio, la *lifeTax* per preservare regole, corrette, che, poiché volte a rispondere ad una condizione infrequente, verrebbero attivate di rado.

3.4.11 - Parametri e servizi del *workbench*

L'utilizzo del *workbench* richiede la specificazione di alcuni parametri volti a descrivere il problema da riprodurre, per comodità sono stati aggiunti tre parametri generali di CW: numero di regole, massimo numero di messaggi ed *effectorsFlag*. Il *workbench* provvede, inoltre, a variare, automaticamente, la lunghezza del genoma ed il numero di *effector*. In aggiunta è possibile specificare il numero di agenti che dovranno essere presenti nel modello e la soglia di vincite che farà scattare il meccanismo di variazione del *bias*; quest'ultimo può essere inibito ponendo a zero il valore del parametro. Per quanto concerne gli *output* è possibile scegliere la frequenza di aggiornamento dei grafici e stabilire se stampare il contenuto dei *dataWarehouse* di ciascun agente o meno.

Ogni agente possiede un proprio *dataWarehouse* e, quindi, viene gestito un *classifier*

specifico per ogni agente; il numero di agenti presenti nel modello può essere fissato a piacere, compatibilmente con le disponibilità di risorse; i tempi di esecuzione, con molti agenti, potrebbero dilatarsi notevolmente, specie su macchine poco potenti. La presenza di diversi agenti permette di notare come, nonostante l'apparente semplicità del modello, le capacità di adattamento di ciascuno tendano ad evolvere percorrendo cammini diversi, anche in presenza di eguali parametri di conduzione per i diversi *classifier*.

Il *workbench* considera ogni posizione del genoma condizione come rappresentazione dello stato di un evento, quindi il numero di eventi viene utilizzato per adeguare la lunghezza del genoma, modificando l'eventuale valore letto nel *file* dei parametri.

Il numero di azioni rappresenta il numero di possibili comportamenti dell'agente a fronte delle possibili situazioni e, perciò, il numero di consigli che il *classifier* può fornire: il numero di *effector* viene automaticamente aggiornato con questo valore.

Il numero di situazioni è il numero di stati dell'ambiente che si vogliono simulare; esiste un legame fra lunghezza del genoma e numero di stati rappresentabili: Numero di stati $= 2^{\text{numero di eventi}}$. Può, inoltre, essere interessante dimensionare un numero di eventi sovrabbondante, in modo da studiare il comportamento del *classifier* nei confronti di informazioni che non mutano mai. Si immagini, ad esempio, di rappresentare 8 stati, i numeri da 0 a 7, con un cromosoma di 4 posizioni: la prima posizione riporterebbe, invariabilmente, il valore zero e, in un caso simile, il *classifier* potrebbe sviluppare regole recanti il carattere di indifferenza nella prima posizione, dimostrando di aver "capito" che quella informazione non è utilizzata o influente. L'emergenza del fenomeno potrebbe essere favorita limitando ad otto il numero di regole.

Durante l'elaborazione viene visualizzato un diagramma, riportante il numero di vincite di ciascun agente, cioè il numero di risposte corrette fornito da ciascun *classifier*. La frequenza di aggiornamento è decisa dall'utilizzatore valorizzando l'apposito parametro, *displayFrequency*, che indica ogni quanti cicli deve essere aggiornato il *display*; con la stessa cadenza il dato viene anche stampato su *standard output*, normalmente il video e, se il valore di *printDataWarehouse* è stato fissato ad uno, viene stampato il contenuto del *dataWarehouse* di ciascun agente. Sempre su *standard output*, viene data notizia dell'avvenuta modificazione del *bias*.

I *classifier* gestiti dal *workbench* utilizzano tutti lo stesso insieme di valori per i parametri, letti nel *file classifierParm.dat*; i valori dei parametri gestiti dal *workbench* prevalgono su quelli del *file*. In questo modo è possibile sperimentare, in principio, modificazioni di parametri di facile utilizzo, senza intervenire sul *file,* ed agire sugli altri, solo dopo aver acquisito una certa dimestichezza col metodo.

3.4.12 - La stampa del *dataWarehouse* e degli oggetti del *kernel*

La classe *DataWarehouse* risponde all'ordine di stampare il suo contenuto. L'azione può essere utile sia per effettuare controlli sullo stato dell'apprendimento e sulla dinamica del medesimo, sia per disporre dei dati utili per trarre le conclusioni riguardanti le regole emerse dall'esecuzione della simulazione. A fronte dell'ordine "*print*", ciascun *dataWarehouse* provvede a stampare il contenuto delle liste di regole e messaggi: ogni oggetto, lista, singola regola, *effector*, messaggio eccetera, viene identificato con il suo indirizzo nella memoria dell'elaboratore.

La prima riga fornisce l'indirizzo del *dataWarehouse*, di seguito viene stampato l'indirizzo della *ruleList*, seguito dal numero di regole contenute. A questo punto inizia l'esposizione delle regole: viene stampato l'indirizzo della regola, l'ammontare del suo patrimonio, *strength*, ed il valore della sua specificità relativa; le due righe successive contengono i dati della *action-part* e della *condition-part*. Da ultimo viene stampato il numero di oggetti presenti nella *partnerList* della regola, seguito dagli indirizzi di tutti gli oggetti contenuti.

Con lo stesso formalismo viene stampata la lista degli *effector* e, di seguito, la lista dei

messaggi, con indicazione, per ogni messaggio, dell'oggetto proprietario, o "autore". Segue la stampa della *matchList,* e delle regole in essa contenute, della lista delle regole vincitrici, *winnerList*, e della *workList*.

In ultimo, vengono stampati il saldo contabile della *treasury* e l'ultimo *effector* attivato, cioè che ha proposto la propria azione. Seguono i contatori, progressivi a partire dall'inizio della simulazione, delle azioni effettuate dal *ruleMaster* utilizzando quel *dataWarehouse*: numero di selezioni, cioè di aste, numero di applicazioni degli operatori speciali *coverDetector* e *coverEffector*, e numero di applicazioni dell'algoritmo genetico, cioè di chiamate al *ruleMaker*.

Tutti gli oggetti del *kernel* sono in grado di rispondere al comando "*print*", quindi il metodo del *dataWarehouse* può essere facilmente modificato dall'utilizzatore, semplicemente rimuovendo o modificando la posizione delle singole chiamate. Risulta, inoltre, abbastanza semplice scrivere metodi di stampa a misura dell'utilizzatore stesso.

3.5 - *Genetic Manipulator*

L'attenzione, in GM, è posta essenzialmente sul *kernel*: un *ruleMaker* capace di evolvere una popolazione di cromosomi, formati da un unico genoma, interagendo con oggetti della classe *Agent* o della classe *Interface*. Il paradigma genetico seguito è quello individuale: ogni agente si identifica con un cromosoma; la distinzione fra genotipo e fenotipo è riconducibile a quella fra aspetto lessicografico e interpretazione semantica di ciascun cromosoma. Nel modello distribuito gli strati *user* e *front-end* sono ridotti al minimo indispensabile: dato che ogni agente possiede un solo cromosoma, questo viene registrato direttamente nell'*interface*, ed il *ruleMaker* interagisce direttamente con le *interface* e viene richiamato dall'ambiente, nel caso specifico dal *modelSwarm*.

La struttura dei modelli, che utilizzeranno GM, risulterà, naturalmente, più ricca e rigorosa, inoltre, l'adozione di un *dataWarehouse* e di pochi accorgimenti nell'interazione fra *ruleMaker* ed agente, potrà permettere, a discrezione dell'utilizzatore, di dotare ciascun agente di un proprio patrimonio genetico, caratterizzato da uno sviluppo autonomo, secondo il paradigma mentale.

Nel rispetto dello schema ERA, l'agente non decide quando debba avvenire l'evoluzione, né conosce la presenza del *ruleMaker*: esso risulta titolare di un'*interface*, o si identifica con l'*interface* stessa, che evolve in relazione ad eventi, per lui, ambientali. La classe *Interface* deve, quindi, essere sempre codificata e ne deve essere creato un certo numero di istanze, anche quando l'interazione del *ruleMaker* avvenga con oggetti della classe *Agent*.

Un algoritmo genetico puro è in grado di evolvere popolazioni capaci di agire su singole strategie: non è richiesto, come nei *classifier*, il meccanismo di asta, e l'evoluzione può essere effettuata ad ogni chiamata. Ogni individuo contiene la rappresentazione di una strategia; tutti gli individui devono essere valutati, cioè tutte le strategie devono essere "provate" nell'ambiente, prima di ogni ciclo di evoluzione. La misura della bontà di ciascuna strategia è data dal valore di *fitness,* assegnato a ciascun individuo dopo ogni "prova": non è richiesto un sistema di distribuzione dei crediti. Queste attività sono svolte direttamente dall'ambiente, nel caso di GM dal *modelSwarm*.

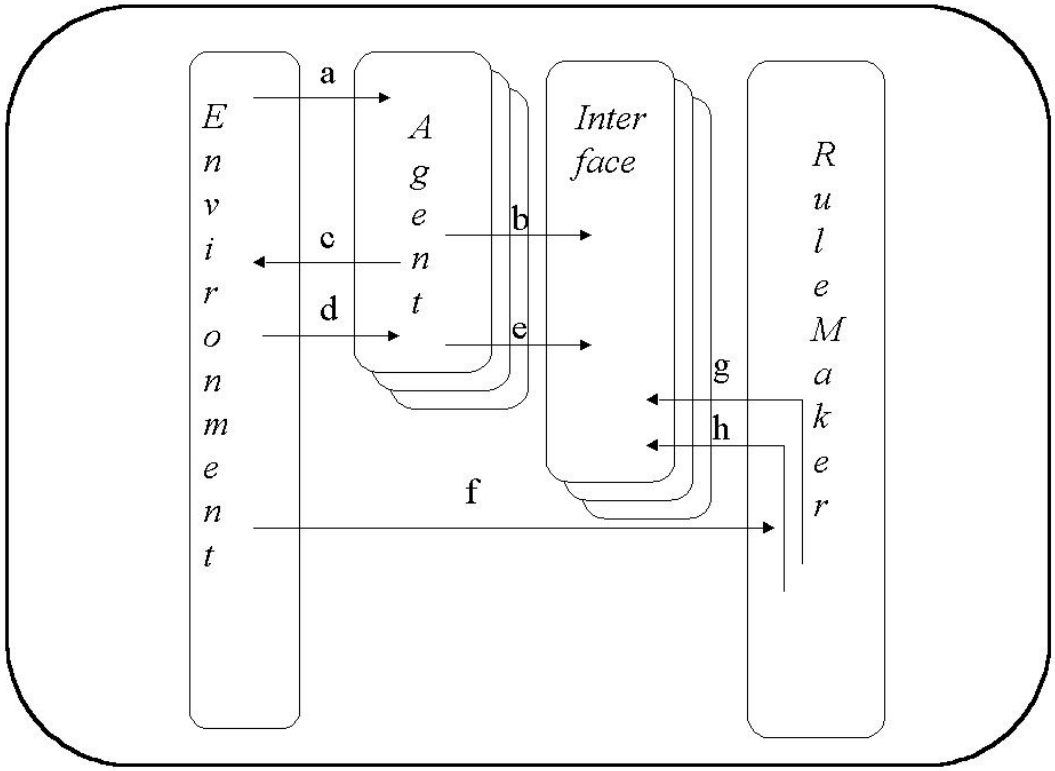
Durante l'evoluzione, tende a propagarsi il genotipo maggiormente "adatto" all'ambiente, cioè quello che codifica il comportamento foriero di migliori risultati. Un elevato grado di omogeneità della popolazione, superiore al novanta percento, indica che il tipo dominante rappresenta una buona soluzione del problema.

3.5.1 - Struttura ed interazioni in un modello con GM

Il *RuleMaker*, contenuto in GM, è in grado di effettuare le operazioni di riproduzione, *crossover* e

mutazione relative ad una popolazione di cromosomi, formati da un solo genoma, rappresentato come *array* di caratteri, definiti sull'alfabeto {0,1}; ad ogni cromosoma è associato un valore di *fitness*, che misura la sua adattabilità all'ambiente. Tale azione si basa sul dialogo con oggetti della classe *Interface*, contenuti in una lista denominata *populationList*. La *populationList* può, come detto, contenere anche indirizzi di oggetti della classe *Agent*: in questo caso ciascuno risulterà proprietario di un'*interface* e dovrà essere in grado di comunicare al *ruleMaker* l'indirizzo della propria *interface*, rispondendo al messaggio *getInterface*. L'interazione con oggetti della classe *Agent*, pur non escludendo la creazione di *interface*, permette di utilizzare la lista di agenti normalmente impiegata per gestire le azioni del modello. Aderendo al paradigma individuale, la lista è unica per tutto il modello; il *ruleMaker* è, però, in grado di recepire, dinamicamente, l'indirizzo di nuove liste: ciò permette di passare al paradigma mentale, inviando al *ruleMaker*, prima dell'ordine di evoluzione, metodo "*step*", l'apposito messaggio di "*set*" della *populationList* appropriata. In quest'ultimo caso, l'agente dovrebbe conoscere l'indirizzo del *ruleMaker* o, meglio, dovrebbe valersi dei servizi di un *ruleMaster* che, oltre ad attivare l'evoluzione, deciderebbe anche quale individuo rappresentasse l'azione migliore da suggerire, tramite *interface*, all'agente. Sarebbe, naturalmente, richiesto un *dataWarehouse* per ogni agente, destinato a contenere l'indirizzo della lista di *interface* ed i contatori statistici; in questo caso ogni agente dovrà essere proprietario di più *interface.*

Figura 3.3 - Struttura di un modello con GM



Come in figura 3.3, l'azione del modello parte dall'ambiente che ordina (a) a ciascun agente di effettuare una azione. L'agente richiede (b) all'*interface* di inferire, dal genoma, indicazioni relative al comportamento da tenere; di seguito esegue (c) le azioni consigliate nell'ambiente. Dopo l'esecuzione delle azioni, agli agenti viene ordinato (d) di computare le conseguenze delle proprie azioni e, successivamente, di trasmettere (e) la propria valutazione all'*interface*: il valore ricevuto andrà a sostituire la misura della *fitness* del cromosoma.

Al termine di questa prima fase, l'ambiente ordina al *ruleMaker* di evolvere (f) le *interface* contenute nella *populationList*. Il *ruleMaker* richiede (g) a ciascuna *interface* la trasmissione del genoma; in seguito, trasmette (h), ad ognuna, il nuovo genoma, risultante dall'azione dell'algoritmo genetico.

3.5.2 - La gestione dell'algoritmo genetico

L'attività è affidata al *ruleMaker,* che la conduce interagendo con le singole *interface*. A supporto dell'attività, sono predisposte quattro liste: *workList*, per copiare la *populationList*, *parentList*, dove vengono registrati gli indirizzi delle *interface* i cui cromosomi sono destinati a riprodursi, *dyingList,* per i cromosomi destinati all'estinzione ed *embryoList*, per contenere i genomi, embrioni, dei nuovi nati.

La decisione di evolvere la popolazione viene presa, dal *ruleMaker*, in base al valore di un parametro che permette di ottenere due comportamenti diversi: evoluzione a cadenza fissa, cioè dopo un certo numero di chiamate, oppure con frequenza calcolata in modo stocastico, data una probabilità di evoluzione stabilita. L'evoluzione a cadenza fissa può essere richiesta con intervallo uno, quando si voglia effettuare una evoluzione ad ogni chiamata del *ruleMaker*. Grazie alla gestione della frequenza di evoluzione, l'utilizzatore può decidere la medesima senza variare la successione delle azioni all'interno del modello, riducendone, quindi, la complessità; risulta, inoltre, più agevole inserire GM in un modello già predisposto.

All'inizio della procedura di evoluzione, il programma provvede a copiare gli indirizzi delle varie *interface* nella *workList* e nella *dyingList*, calcolando, contemporaneamente, la sommatoria delle *fitness* di tutti gli individui e determinando l'ammontare del più basso valore di *fitness* presente nella popolazione. Dove la *populationList* contenesse indirizzi di agenti, a ciascuno verrebbe, preventivamente, richiesto l'indirizzo dell*'interface*, per popolare la *workList*. In questa fase viene calcolato il numero di individui da destinare alla riproduzione, in ragione dei parametri di funzionamento del sistema.

Nella seconda fase, vengono selezionati gli individui destinati alla riproduzione: si procede, come in CW, a sommare la *fitness* di ciascun cromosoma, fino a raggiungere un ammontare superiore ad un valore, *random*, compreso fra zero e la *fitness* totale; viene selezionato l'individuo che ha determinato il superamento. La *interface*, proprietaria del cromosoma, viene rimossa dalla *workList* e accodata alla *parentList,* viene aggiornato il valore della *fitness* totale; la procedura viene ripetuta, fino ad ottenimento del numero stabilito di genitori. La rimozione, di ogni *interface* selezionata, dalla *workList* garantisce che un individuo non possa essere scelto più di una volta; ciò assicura che ad ogni riproduzione concorrano sempre due individui diversi e che un individuo non possa partecipare a più di una riproduzione per ogni ciclo.

La terza fase consta della selezione degli individui destinati all'estinzione: si procede come per la scelta dei genitori ma, questa volta, gli individui vengono rimossi dalla *dyingList* e la procedura viene ripetuta per un numero pari a quello degli individui destinati a sopravvivere; al termine la *dyingList* contiene gli indirizzi degli individui, probabilmente, peggiori.

In ultimo viene effettuata la riproduzione: i cromosomi, appartenenti alle *interface* presenti nella *parentList*, vengono presi a coppie ed incrociati, in accordo con le probabilità di *crossover*; per ciascun elemento dell'*array*, viene verificata l'opportunità di effettuare una mutazione, in base ad apposito parametro, generando un numero, *random*, fra zero e uno, e confrontandolo con il parametro. La mutazione consiste nel sostituire all'allele "0" l'allele "1" e viceversa. Da ogni coppia vengono ottenuti due nuovi genomi, embrioni, che saranno, momentaneamente, immessi nella *embryoList*; in seguito gli embrioni verranno comunicati alle *interface*, presenti nella *dyingList*, affinchè provvedano a sostituirli ai cromosomi posseduti. Agli embrioni e, quindi, alle *interface* che li riceveranno, può essere conferito un valore iniziale di *fitness*: fisso e stabilito a priori oppure derivante dalla media ponderata dei valori dei genitori.

La gestione degli embrioni, interna al *ruleMaker*, è totalmente automatica: il programma provvede a generare il numero di istanze necessarie, a variare, dinamicamente, la lunghezza del genoma in esse contenuto ed a gestire la *embryoList* in modo da riutilizzare le istanze già create. L'utilizzatore non ha necessità di creare istanze di *Embryo*, né deve includere il *file Embryo.h* in alcuna delle classi del modello.

3.5.3 - I parametri e la funzione di valutazione della *fitness*

Per il funzionamento di GM sono richiesti pochi parametri; inoltre, l'aderenza al paradigma individuale rende superflua l'adozione di classi specializzate nella gestione dei medesimi: si utilizza un unico algoritmo genetico, quindi, una sola versione di parametri. Il valore dei vari parametri è fissato, all'atto della generazione, inviando al *ruleMaker* gli appropriati messaggi di "*set*"; tutti i valori possono essere modificati, anche durante l'elaborazione, con i medesimi messaggi.

L'utilizzatore deve specificare: lunghezza del genoma, tasso unitario di ricambio generazionale, probabilità di esecuzione del *crossover* e probabilità di mutazione. Al *ruleMaker* deve essere comunicato anche l'indirizzo della *populationList*. Volendo utilizzare GM secondo il paradigma mentale, tali parametri saranno contenuti nel *dataWarehouse* di ciascun agente.

Il tasso di ricambio generazionale, *turnoverRate*, può assumere valori compresi fra zero ed uno: moltiplicando la numerosità della popolazione per *turnoverRate* si ottiene il numero di individui destinati a riprodursi. Le riproduzioni avvengono sempre con il concorso di due individui e producono due nuovi nati; i nuovi nati rimpiazzano altrettanti elementi della generazione precedente. Valori particolarmente alti, tendenti ad uno, del tasso di ricambio generazionale possono costituire un forte ostacolo per la omogeneizzazione della popolazione, impedendo la convergenza del sistema; tassi così elevati devono essere intesi come casi particolari di utilizzo; normalmente un valore nell'ordine di 0.5 garantisce buoni risultati in tempi accettabili.

Crossover e mutazione servono ad introdurre novità, onde permettere un'esplorazione dell'insieme delle soluzioni possibili, rappresentate dai diversi genomi ottenibili, sufficientemente estesa. Un valore troppo elevato dei due parametri potrebbe ostacolare la convergenza della popolazione verso il tipo migliore; nel contempo, valori eccessivamente ridotti potrebbero determinare convergenze, premature, verso tipi rappresentativi di ottimi "locali". In linea di principio, è possibile affermare che questi valori andrebbero dimensionati in modo direttamente proporzionale alla complessità della situazione affrontata.

Con gli algoritmi genetici non si presenta il problema di permettere la convivenza di diversi cromosomi, sollevato al proposito dei *classifier system*: i valori forniti dalla funzione di computo della *fitness* possono essere fortemente differenziati. Occorre, però, prestare attenzione alla distribuzione di questi valori: la presenza di "picchi" isolati potrebbe comportare la convergenza del sistema su livelli non ottimali. In questi casi, l'esplorazione dello spazio delle soluzioni potrebbe essere potenziata tramite l'adozione di più alte probabilità di mutazione e *crossover*.

Sulla gestione della *fitness* sono state introdotte due possibilità innovative: assegnazione di un valore iniziale di *fitness* ai nuovi individui e utilizzazione, nelle selezioni, di valori differenziali, anziché assoluti, della *fitness*.

Il parametro *childrenFitness* permette di specificare un valore, reale, compreso fra –1 e il massimo valore attribuibile ad una variabile di tipo *float*. Se il valore fornito è nullo o positivo, esso viene inteso come ammontare di *fitness* da assegnare, indistintamente, a tutti nuovi nati. I valori negativi sono utilizzati come peso ponderale per mediare la contribuzione di ciascun genitore alla *fitness* del nuovo nato: si presuppone che ciascun nuovo individuo erediti una quota della *fitness* di ciascun genitore così calcolata:

$$c1 = \lambda * f1 + (1 - \lambda)\, f2$$
$$c2 = \lambda * f2 + (1 - \lambda)\, f1$$

dove c1 e c2 indicano i valori assegnati ai nuovi nati, f1 ed f2 le *fitness* dei genitori e λ è il valore di *childrenFitness*, il cui segno viene considerato positivo.

Durante l'azione dell'algoritmo genetico, mano a mano che il livello globale dei risultati aumenta, le *fitness* dei singoli individui tendono a differenziarsi sempre meno, particolarmente in termini relativi; la selezione degli individui risulta, perciò, meno efficace che all'inizio. L'algoritmo

genetico è, tipicamente, capace di fornire buone soluzioni, ma l'ottenimento di risultati molto precisi può comportare, per l'utilizzatore, la predisposizione di complicate funzioni di valutazione della *fitness*. Per aumentare l'efficacia della selezione, è stata prevista la possibilità di basare l'attività sulla differenza fra la *fitness* di ciascun individuo e quella del peggiore fra essi; in questo modo vengono poste in risalto anche piccole differenze, specie quando il livello di *fitness* tenda a convergere su valori simili per tutti gli individui. L'accorgimento illustrato permette, inoltre, di recepire valori negativi di *fitness*, assicurando una automatica traduzione dei valori forniti nella metrica più adatta all'elaborazione. Si noti, infine, che la manipolazione dei valori avviene esclusivamente ai fini della selezione: l'ammontare contenuto nelle singole *interface* non viene mai modificato. L'utilizzatore può scegliere se far agire l'algoritmo in base al valore totale di *fitness*, cioè utilizzare il sistema classico, oppure sfruttare la possibilità illustrata, valorizzando l'apposito parametro *useDeltaFitness,* rispettivamente, a zero oppure uno.

Come già descritto, il *ruleMaker* è in grado di gestire la frequenza dell'evoluzione, svincolando l'utilizzatore dalla codificazione di istruzioni a tal fine. E' possibile richiamare, ad ogni ciclo, il *ruleMaker*, delegando ad esso la decisione sull'opportunità di evolvere la popolazione, dato il valore del parametro *evolutionFrequency*. Se viene specificato un numero naturale, il *ruleMaker* effettua l'evoluzione solo dopo un numero di chiamate pari ad *evolutionFrequency*; se il valore è uno l'evoluzione avviene ad ogni chiamata. Con valori di *evolutionFrequency* reali e compresi fra zero e uno, il *ruleMaker* confronta un numero *random*, generato nello stesso intervallo, con il suddetto valore ed esegue l'evoluzione solo quando il primo risulti inferiore al secondo: la decisione viene presa, perciò, in modo stocastico. L'evoluzione può, infine, essere inibita valorizzando *evolutionFrequency* a zero.

La possibilità di modificare tutti i parametri, anche durante l'elaborazione, permette di realizzare modelli caratterizzati da forte dinamismo, dove, ad esempio, si potrebbe scegliere di variare la frequenza di evoluzione in relazione a particolari accadimenti, modificare il tipo di manipolazione genetica, cambiare il tasso di ricambio generazionale e così via. La sola attenzione da spendere riguarda la rigenerazione delle *interface* a fronte di variazioni della lunghezza del genoma.

3.5.4 - Le macro, le statistiche ed il modello dimostrativo

Il pacchetto GM contiene un *file* di macro, utilizzate dal compilatore dell'*Objective-C*: si tratta di istruzioni che influiscono sul codice che verrà, successivamente, letto dal compilatore, modificandolo. Questo tipo di codifica permette di agevolare la variazione di parti del codice: le modifiche vengono eseguite principalmente nel *file* delle macro, pur valendo per tutti i *file* del progetto, e si evitano errori di sintassi.

GM utilizza tre macro per codificare: il limite di estensione del genoma, cioè il numero massimo di elementi dell'*array* di caratteri destinato a contenerlo, la scelta della classe di oggetti con cui il *ruleMaker* dovrà interagire, *Agent* o *Interface*, e l'opzione di riorganizzazione casuale della *parentList* prima di iniziare la fase di riproduzione, attività del tutto facoltativa. La prima macro, MAXGL, serve a determinare la massima estensione di ciascun genoma. nel *file* distribuito, il suo valore è 1024. A proposito di questo valore occorre segnalare l'esigenza di eseguire, unitamente alle modifiche di MAXGL, la variazione delle istruzioni di dimensionamento dell'*array*, destinato ad ospitare il genoma, nei *file Interface.h* ed *Embryo.h*; in quelle istruzioni occorre, infatti, riportare il valore di MAXGL, a causa di un vincolo insito nella struttura del linguaggio di programmazione adottato. Lo stesso vincolo impone di inserire, nel *file RuleMaker.h*, le istruzioni di import del *file Agent.h* e la dichiarazione della variabile *agent* di tipo *Agent**, quando si decida di far interagire il *ruleMaker* con oggetti della classe *Agent*. Nessuna variazione deve essere effettuata nei *file*, contenenti le istruzioni di programma vere e proprie, caratterizzati dal suffisso ".m"; l'utilità delle macro è correlata proprio alla maggior complessità di intervento che caratterizza questo tipo di *file*.

Il *ruleMaker* conta il numero di operazioni eseguite, a partire dal momento della creazione, accumulando il risultato in quattro contatori relativi a: numero di evoluzioni, cioè al numero di volte in cui la popolazione è stata manipolata, numero totale di riproduzioni, ognuna coinvolge una coppia e

produce due figli, numero totale di *crossover*, ognuno coinvolge una coppia ed agisce su ambedue i figli, numero di mutazioni, cioè numero di posizioni mutate, uno stesso genoma può, se molto lungo, subire più mutazioni durante una riproduzione. Questi dati sono accessibili tramite appositi metodi "*get*" o possono essere stampati su *standard output* con il metodo "*print*".

A corredo del *ruleMaker,* viene fornito un piccolo modello, dove, per semplicità, le *interface* stesse sono gli agenti, non esiste *ruleMaster* e l'evoluzione è decisa direttamente dal *modelSwarm,* che provvede a fornire, al *ruleMaker,* i valori dei parametri e l'indirizzo della lista delle *interface, populationList*. Ad ogni ciclo il modello trasmette: all'*interface*, l'ordine di modificare, in modo casuale, la propria *fitness*, simulando il processo di valutazione dei vari genomi che avverrà nelle applicazioni reali, e al *ruleMaker* l'ordine di evolvere la popolazione. In base alla cadenza di *display,* stabilita dall'utilizzatore, vengono stampate, su *standard output*, le *interface* contenute nella *populationList*; per ciascuna si stampa, nell'ordine: indirizzo, contenuto del genoma, inserendo un punto ogni quattro posizioni per agevolare la lettura, e valore corrente di *fitness*. Per ultime vengono fornite le statistiche. Il modello non produce risultati, diversamente da quello di CW, il suo scopo è solo di illustrare l'azione del *ruleMaker*; pur tuttavia, esso può costituire, per l'utilizzatore, la base da cui sviluppare lavori specifici; i valori usati per i parametri di funzionamento del sistema possono essere usati come *default* per le prime esperienze di utilizzo.

3.6 - Gli strumenti a corredo di CW e GM

Ambedue i pacchetti contengono due strumenti, rispettivamente, per il tracciamento delle azioni svolte dai programmi e per l'indagine sul contenuto della memoria dell'elaboratore. Il loro utilizzo permette sia di meglio comprendere come avviene l'elaborazione, sia di ottenere dati utili per le fasi di *debug* dei nuovi modelli. L'azione delle due componenti può essere facilmente estesa ai programmi redatti dall'utilizzatore.

Il primo strumento, denominato *trace*, è basato sull'utilizzo di istruzioni per il compilatore, volte a permettere l'inibizione di istruzioni di programma, senza intervenire sul codice e senza inserire in esso istruzioni condizionali che appesantirebbero l'elaborazione. Il secondo, denominato *dump*, è una vera e propria classe, di cui si utilizza solitamente un'unica istanza, contenente i metodi necessari a stampare, su *standard output*, il contenuto di parti della memoria, a partire da un indirizzo dato e per un'estensione decisa dall'utilizzatore.

3.6.1 - *Trace*

La maggior parte dei prodotti di *debug* permette di sospendere l'esecuzione del programma dopo ogni istruzione e di visualizzare il contenuto delle variabili dello stesso; alcuni consentono di fissare, preventivamente, i punti di arresto, in modo da poter eseguire interi blocchi di istruzioni ogni volta. L'utilizzo di queste tecniche risulta piuttosto oneroso in casi non banali: indagine riguardante il comportamento di grossi programmi o di progetti formati da molti moduli in interazione, errori la cui manifestazione non sia sistematica, errori che compaiono solo dopo l'esecuzione di diversi cicli di elaborazione, eccetera. Per questo motivo, è pratica usuale porre nel codice istruzioni di *display* che forniscano dati sull'elaborazione; queste istruzioni devono, però, essere rimosse nella versione definitiva dei programmi, per evitare di inquinare gli *output*; una via alternativa consiste nel condizionarne l'esecuzione al valore di determinati parametri, ma ciò comporta un appesantimento dell'elaborazione e, solitamente, una minor chiarezza del codice.

Trace permette di suddividere le istruzioni di *display* in famiglie, ciascuna specializzata nell'azione su un determinato componente o su una precisa funzione; tutte le istruzioni delle varie famiglie possono essere attivate, o inibite, agendo in un unico punto del *file Trace.h*. La gestione delle istruzioni avviene a livello di compilazione, quindi, le istruzioni possono rimanere nel codice, inibite, senza comportare aggravi per l'elaborazione. Le istruzioni, presenti nel codice, sono individuate da

uno specifico verbo, *TRACE*, che ne permette l'immediata identificazione: con la pratica d'uso diviene automatico ignorarle durante la lettura dei programmi, come accade per i commenti. Queste caratteristiche permettono di conservare le istruzioni di *tracing* anche nella versione definitiva, disponendo, così, di uno strumento immediatamente utilizzabile a fronte di errori. Avendo cura di dirigere lo *standard output* su *file*, è possibile far agire il programma per un certo periodo, collezionando dati che potranno essere: esaminati in un secondo momento, sottoposti all'attenzione di collaboratori, non presenti al momento dell'elaborazione, o inviati agli autori dei programmi, per meglio descrivere un problema. I *file* così prodotti, possono risultare anche un utile supporto per spiegare, ad altri, il funzionamento dei programmi stessi e, quindi, dei modelli.

Il funzionamento di *trace* è basato sull'utilizzo dell'istruzione, per il compilatore, di definizione, presente con nomi diversi nella più parte dei linguaggi di programmazione: con questo verbo si ordina al compilatore di sostituire una determinata macro-istruzione, presente nel codice dei programmi, con un certo testo; l'ordine ha effetto per tutte le ricorrenze della macro-istruzione, presenti nei programmi. Nel caso di *trace* la macro-istruzione, da immettere nel codice, ammette, come argomento, l'intera istruzione di *display*; gli ordini per il compilatore, riguardanti tutte le macro-istruzioni, sono contenuti nel *file Trace.h*, che deve essere incluso in tutti i programmi. L'inibizione del *display* viene effettuata ordinando al compilatore di sostituire la macro con un commento; l'attivazione avviene ordinando di sostituire la macro con il proprio argomento. In questo modo all'atto della compilazione le istruzioni inibite saranno ignorate, non verranno incluse nel *file* eseguibile, mentre quelle attive consteranno di normali istruzioni codificate nella sintassi del linguaggio adottato. L'argomento della macro-istruzione potrà, perciò, essere qualunque istruzione di programma e trattare qualsiasi variabile dello stesso; il metodo può essere usato anche per predisporre punti di interruzione, in cui l'elaborazione viene sospesa in attesa di un ordine da parte dell'utilizzatore.

Supponendo di aver codificato, in diversi punti di un programma redatto in *Objective-C*, la macro:

```
TRACE(printf ("valore di x: %d\n",x);)
```

tutte le istruzioni di stampa potranno essere attivate codificando, in *Trace.h*,

```
#define TRACE(A) A
```

che ordina di sostituire a "*TRACE(A)*" il suo argomento; in questo caso, ai fini della compilazione, l'istruzione sarà stata trasformata in:

```
printf("valore di x %d\n",x);
```

Codificando:

```
#define TRACE(A) //
```

l'istruzione verrebbe trasformata in "//", che in *objective-c* identifica il commento, e, conseguentemente, ignorata ai fini della compilazione.

Si noti che *TRACE* è esclusivamente il nome dato alla macro-istruzione, quindi, è possibile codificare macro con nomi diversi (magari aggiungendo semplicemente un suffisso, in modo da conservare la possibilità di identificare, immediatamente, la funzione della macro durante la lettura del codice) per raggruppare le istruzioni di *tracing* in famiglie; agendo, poi, sul *file Trace.h* l'utilizzatore potrà decidere, in base ai dati di cui necessita, quali famiglie attivare. La semplicità del metodo rende agevole la predisposizione di *trace* fortemente personalizzate.

3.6.2 - Le *trace* in CW e GM

Durante la redazione di CW e di GM, è stata predisposta una struttura *standard* di *tracing*: ogni

metodo contiene una macro iniziale, di *entry*, ed una finale, di *exit;* per ogni classe è stata predisposta una apposita macro, in modo che l'utilizzatore possa effettuare indagini per singole tipologie di oggetti. All'inizio dell'elaborazione del metodo vengono forniti, nell'ordine: indirizzo dell'oggetto, istanza della classe che sta eseguendo il metodo, la dicitura "*entry*" seguita dal nome del metodo, *selector*, e da nome e valore degli argomenti ricevuti; al termine viene ripetuto l'indirizzo dell'oggetto, seguito dalla dicitura "*exit*", dal *selector* e dal valore restituito all'oggetto chiamante.

La lettura dei dati prodotti attivando tutte le *trace*, permette di seguire esattamente le interazioni fra gli oggetti e, se nel caso, di individuare immediatamente: punti di interruzione, errori nel passaggio dei dati fra oggetti, errori di indirizzamento degli oggetti stessi.

3.6.3 - *Dump*

I prodotti di *debug* forniscono, normalmente, i valori di specifiche variabili interpretando, dato il formato delle medesime, il contenuto delle zone di memoria ad esse riservate; risultato analogo può aversi con istruzioni di *print* codificate nel corpo del programma. Quando il contenuto della memoria non risultasse conforme al formato suddetto, l'*output* sarebbe privo di significato. In questi casi, l'individuazione della causa dell'errore, risulta fortemente agevolata dalla possibilità di conoscere l'esatto contenuto dell'area di memoria indagata. *Dump* è un oggetto capace di stampare, in un formato *standard*, il contenuto di qualunque area di memoria, indipendentemente dal formato e dalla lunghezza delle variabili a cui l'area è stata destinata.

Utilizzando puntatori o *array*, non è infrequente che un'istruzione agisca su aree, diverse da quelle riservate alle variabili su cui il programmatore intendeva agire. Questo tipo di errore, in casi particolari, può indurre comportamenti fuorvianti del programma, che rendono difficoltosa l'identificazione precisa della loro causa. Nella programmazione ad oggetti, i destinatari dei messaggi sono identificati dall'indirizzo delle aree di memoria loro riservate: può essere utile stampare il contenuto delle aree indirizzate, per individuare errori nella gestione delle liste di oggetti.

Per utilizzare lo strumento occorre che nel modello sia stata generata un'istanza della classe *Dump*; tale oggetto potrà essere richiamato, in qualunque momento dell'elaborazione, fornendo un indirizzo ed un valore, lunghezza dell'area che si vuole visualizzare. Di seguito sono riportate alcune righe tratte da un *output* di *dump*, ognuna contiene la stampa di un blocco di sedici *byte*: il primo valore è l'indirizzo di partenza del blocco, seguono i valori dei sedici *byte*, divisi per agevolare la lettura in gruppi di quattro (il valore di un *byte* viene rappresentato con due caratteri, nella codifica esadecimale), ed, in ultimo, viene stampata la traduzione in formato carattere, dove questa non sia possibile viene visualizzato un punto.

```
497FF78  94AB4B00 884B9304 524D4B52 00000000   ..K..K..RMKR....
497FF88  00000000 40E39704 F0F09704 00000000   ....@...........
497FF98  00000000 00000000 00000000 00000000   ................
497FFA8  00000000 00000000 00000000 00000000   ................
```

La stampa è stata ottenuta fornendo a *dump* l'indirizzo del *ruleMaker*, durante il *run* di un modello con CW; l'identificazione dell'oggetto stampato è facilitata dalla presenza delle quattro lettere "RMKR", visibili all'inizio dell'area. Tutti gli oggetti di CW contengono una apposita variabile, *eyeCatcher*, valorizzata nel metodo di creazione, per contenere una etichetta, che ne faciliti l'identificazione durante l'esplorazione della memoria. Questo accorgimento risulta particolarmente utile nei casi di errore nella gestione delle variabili destinate a contenere indirizzi.

3.7 - Dalle formiche di Langton al formichiere di Ferraris

Il modello è stato concepito per esemplificare un utilizzo di CW sia come *learning-classifier-system*,

sia come sistema esperto tradizionale. Lo studio di Langton illustra la possibilità di ottenere comportamenti complessi, per il tramite dell'interazione di unità, estremamente semplici, il cui comportamento obbedisce a regole precise; il successo del *classifier* permette, quindi, di dimostrare la sua idoneità ad operare in ambienti dinamici, dove possano svilupparsi comportamenti imprevedibili a priori, strutturalmente simili agli ambiti di applicazione delle scienze sociali. Tale somiglianza è stata rinforzata con l'inserimento, in aggiunta al modello originale, della possibilità di riproduzione delle formiche. Il paradigma genetico adottato è quello mentale: le regole rappresentano idee che si sviluppano nella mente di ciascun agente. L'estrema semplicità delle regole di movimento delle formiche permette di verificare, in modo intuitivo, il livello di apprendimento raggiunto dal *learning-classifier*, consentendo un'immediata percezione della meccanica e delle potenzialità del metodo.

3.7.1 - La formica di Langton

La metafora è rappresentata da un punto, colorato diversamente dallo sfondo, che si muove sullo schermo di un *computer*, concepito come un insieme di caselle, obbedendo a due semplici regole: se la casella di partenza è bianca, la si colora di nero, si ruota, di novanta gradi, verso destra e si avanza di una casella; se è nera, la si colora di bianco, si ruota a sinistra e si avanza. Le caselle possono essere inizialmente tutte bianche o alcune, casualmente selezionate, possono essere colorate di nero. L'effetto interessante consiste nel fatto che, dopo aver seguito percorsi, la cui complessità cresce rapidamente al crescere del numero di passi compiuti, ed indipendentemente dalla presenza delle caselle colorate e dalla loro distribuzione, il punto intraprende, comunque, un sentiero che porta, direttamente, verso uno dei quattro angoli del video. Al proposito si legge in Galante (1997):

> (…) Siamo di fronte a un esempio di caos deterministico; in natura esistono numerosi sistemi fisici che si comportano così, ma il fenomeno è forse meno preoccupante del problema di Langton, dove, se ben ricordate le condizioni iniziali del moto non erano affette da errori!

3.7.2 – Struttura e dinamica del modello

Su una superficie toroidale, tale per cui i confini in ciascuna direzione toccano quelli della direzione opposta, si muove una collettività di formiche di Langton, alcune delle quali, le regine, possono generarne altre; fra queste vi possono essere anche delle nuove regine. Il movimento delle formiche avviene secondo le regole di Langton: ogni formica avanza di un passo alla sua destra se incontra una traccia di feromone, casella bianca, o alla sua sinistra se la traccia è assente; se nella posizione di partenza esiste una traccia, questa viene rimossa mentre, se la traccia è assente, ne viene lasciata una.

Contro la colonia agisce un formichiere che, date le dimensioni, è in grado di vedere l'intero spazio e, quindi, di localizzare una qualunque formica; per riuscire a mangiarla, però, ne deve anticipare le mosse, posizionando la sua proboscide sulla casella che la formica andrà ad occupare. Il problema del formichiere è, quindi, quello di capire quali regole stanno alla base dell'agire delle formiche. La sua azione si articola in:

a) Localizzazione di una formica, cioè acquisizione delle informazioni relative alla sua posizione e direzione, rilevazione della presenza di feromone e calcolo del punto in cui la formica verrà a trovarsi dopo una mossa, per tentarne la cattura.

b) Cattura della formica, quando questa muova verso la casella individuata.

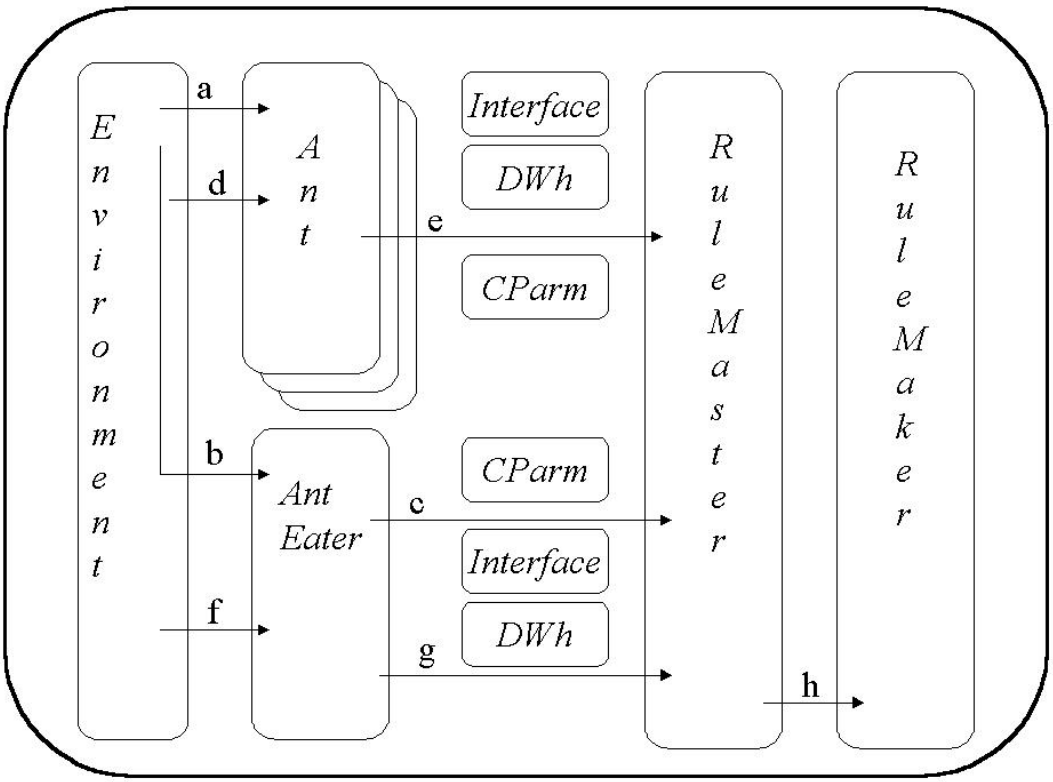
Formiche e formichiere si avvalgono dei servizi erogati dalla stessa coppia di oggetti *ruleMaster* e *ruleMaker*. Questi, agendo su patrimoni normativi diversi, *dataWarehouse*, ed in base a parametri differenti, *classifierParm*, assumono comportamenti tipici, rispettivamente, di un sistema esperto e di un *learning-classifier-system*.

Dopo un certo numero di interazioni, il *dataWarehouse* del formichiere, inizialmente

popolato in modo casuale, giunge a contenere regole perfettamente uguali a quelle del *dataWarehouse* delle formiche; in pratica il formichiere, quindi il *classifier*, riesce ad inferire e riprodurre, osservandone il comportamento, le regole che determinano l'azione delle formiche. Lo stesso programma, operando sulla simulazione di un mercato, dovrebbe, allora, inferire le regole che ne presidiano il funzionamento.

Lo schema in figura 3.4 rappresenta la struttura del modello e le più importanti interazioni fra gli oggetti. L'ambiente si identifica con il *modelSwarm* che, oltre a gestire la dinamica del modello, provvede anche a fornire tutte le informazioni relative agli agenti, in modo da evitare comunicazioni dirette fra gli stessi.

Figura 3.4 – Il formichiere di Ferraris



Gli agenti sono rappresentati da istanze delle classi *Ant* ed *Anteater*: nel modello vengono create diverse istanze della prima ed una sola della seconda; tutte le formiche utilizzano lo stesso *dataWarehouse*, e la stessa *interface*, in quanto le regole, che presidiano il loro comportamento, non sono soggette ad evoluzione. Il contenuto del *dataWarehouse* delle formiche viene codificato dal programmatore. Il formichiere possiede un proprio magazzino di regole ed una propria *interface*.

L'azione viene avviata dall'ambiente che ordina (a) a tutte le formiche di riprodursi: a ciascuna regina viene assegnata una probabilità di riproduzione, decisa al momento dell'esecuzione del modello; le operaie ricevono sempre zero e, quindi, non si riproducono. Le nuove operaie vengono posizionate nelle immediate vicinanze della regina, mentre alle nuove regine è assegnata una posizione casuale, in modo da simulare la fondazione di un nuovo formicaio.

Terminata la riproduzione, il *modelSwarm* ordina (b) al formichiere di ricercare una formica, cioè di memorizzare la posizione attuale di una di esse, ottenuta interrogando l'ambiente; di seguito il formichiere richiede (c) i servizi del *ruleMaster* fornendo, tramite l'*interface* i dati relativi alla posizione della formica ed alla presenza di feromone nella casella; il *ruleMaster* deve stabilire quale direzione prenderà la formica. In base al consiglio ricevuto, viene calcolata la posizione futura dell'insetto. Ambedue si valgono dei servizi di *interface*, *dataWarehouse* e *classifierParm*.

A questo punto, l'ambiente ordina (d) alle formiche di muovere un passo: ogni formica interpella (e) il *ruleMaster*, fornendo i dati relativi alla propria posizione e alla presenza di feromone, per sapere in che direzione muovere. Quest'ultimo interagisce, come di consueto, con gli oggetti *dataWarehouse*, *classifierParm* ed *interface,* di proprietà della colonia. Poiché l'insieme delle regole delle formiche non è soggetto ad evoluzione, nessuna ricompensa viene comunicata al *ruleMaster*; durante la sua azione le regole non vengono tassate, né le vincitrici sono soggette a pagamenti verso gli autori dei messaggi. Non viene mai chiamato il *ruleMaker*. Questo comportamento è ottenuto ponendo a zero i valori dei parametri per la gestione della distribuzione dei crediti e per l'algoritmo genetico; non è richiesto alcun intervento sui programmi.

Da ultimo viene ordinato (f) al formichiere di tentare la cattura: questi richiede all'ambiente la nuova posizione della formica e, se le coordinate calcolate precedentemente, risultano uguali a quelle ottenute, comunica all'ambiente che ha mangiato l'insetto. Dopo ogni tentativo di cattura viene trasmessa (g) al *ruleMaster* una ricompensa, in dipendenza del risultato conseguito. In base ai parametri specificati nel *classifierParm*, il *ruleMaster* richiede (h) l'evoluzione della popolazione di regole al *ruleMaker*.

3.7.3 – Visualizzazione dei risultati

Le formiche sono rappresentate come punti, in movimento all'interno di una finestra grafica, colorati di rosso per le operaie e di bianco per le regine. La finestra simboleggia il campo visivo del formichiere, quindi, questi non compare. Il progresso dell'apprendimento è rappresentato dal numero di formiche mangiate durante ogni periodo di osservazione; il dato è riportato, insieme al numero di formiche, totali e regine, presenti nel modello, in un apposito diagramma.

Il *modelSwarm* provvede a stampare le regole, contenute nel *dataWarehouse* delle formiche e del formichiere all'inizio dell'esperimento e, ad apprendimento completato o ad estinzione delle formiche, a stampare, nuovamente, l'insieme di regole del formichiere. Nell'ipotesi di completo apprendimento si noterà che, la seconda volta, le regole del formichiere risulteranno uguali a quelle delle formiche. In alcuni casi la presenza di *wildcard*, nella prima posizione della *condition-part*, risulta dovuta al fatto che tale posizione non è effettivamente utilizzata. Le condizioni, infatti, contengono due dati: la direzione, codificata su due posizioni consecutive, e lo stato del feromone; la lunghezza, quattro, di ciascun genoma è, però, capace di contenere sedici combinazioni di "0" e "1", quindi la prima posizione non viene utilizzata. Per ogni intervallo di verificazione viene stampato, inoltre, il numero di tentativi, coronati da successo.

3.7.4 – Personalizzazione del modello

Tramite le *probe standard* di *Swarm*, è possibile stabilire: la frequenza di *display*, cioè il numero di cicli intercorrenti fra due aggiornamenti dell'*output*, la dimensione dell'ambiente, la probabilità della presenza di tracce di feromone, il tasso di riproduzione, il numero di cicli di generazione da effettuarsi prima di far agire il formichiere. Ogni regina, in ogni ciclo, genera un altro individuo, in base al tasso di riproduzione: effettuare dieci generazioni con fecondità 0.5 porterà, di norma, a creare cinque formiche; la probabilità che una di queste sia una regina è pari al quadrato del tasso di riproduzione. La riproduzione viene inibita dal sovraffollamento dell'ambiente, rappresentato dal raggiungimento del numero massimo di formiche, specificato dall'utilizzatore. Oltre al massimo numero di formiche, deve essere stabilito anche il massimo numero di regine.

In presenza di tassi di riproduzione troppo bassi, o di un numero ridotto di regine, il formichiere potrebbe estinguere la colonia, senza aver completato il proprio processo di apprendimento. Allo scopo di garantire il raggiungimento di elevati livelli di conoscenza, quando viene selezionato un numero massimo di regine pari ad uno, il *modelSwarm* provvede, automaticamente, a nascondere l'unica regina al formichiere, onde evitare che una prematura

scomparsa di questa possa portare all'estinzione delle operaie, prima che il formichiere abbia appreso.

Nell'interazione fra un formichiere ed una sola regina, avviene, normalmente, l'estinzione totale della colonia, salvo la regina; tale risultato può presentarsi anche con più regine, dato un congruo tasso di riproduzione. In presenza di forti tassi di riproduzione e diverse regine, si osserva comunque un completo apprendimento del formichiere, pur se, data l'elevata fertilità delle formiche, non si addiviene alla loro totale estinzione.

3.7.5 – Una piccola variante

Per aumentare la difficoltà del formichiere, si sono cambiate le regole di movimento delle regine: esse oscillano attorno alla posizione iniziale. L'introduzione della modifica porta a sedici il numero di regole necessario, saturando la capienza del genoma. Dato il minor numero di regine, risulta ridotta la possibilità di osservazione, del loro comportamento, da parte del formichiere; ciò ne aumenta le probabilità di salvezza, contribuendo a stabilizzare la simulazione.

3.8 - Alcuni esempi di utilizzazione di CW

Nella prima parte del paragrafo, vengono presentati i risultati di alcune elaborazioni fatte con il *workbench* di CW: gli esempi si riferiscono a situazioni astratte, servono a verificare le capacità di apprendimento dello strumento e forniscono la possibilità di commentare gli effetti delle variazioni dei parametri. Ogni caso, oltre che illustrato per il mezzo della stampa degli *output* prodotti dal programma, è brevemente commentato: ne vengono discusse le peculiarità, si procede alla spiegazione del significato dei parametri, all'interpretazione dei risultati e, nei casi più semplici, alla disamina delle regole prodotte durante l'evoluzione. Nella maggior parte dei casi viene usato un solo agente, per motivi di semplicità, ma, come dimostrato con il quarto esempio, il numero dei medesimi può variare senza creare problemi.
La seconda parte riguarda il modello del "formichiere di Ferraris": sono illustrati i risultati del modello di base e di quello modificato, secondo la variante proposta nel paragrafo precedente.

3.8.1 - Esempio 1

La simulazione riproduce il caso in cui occorra selezionare un'azione, fra le quattro ammesse, a fronte di sedici possibili stati ambientali; questi ultimi derivano dalle diverse combinazioni date dal verificarsi di quattro eventi. In realtà solo due degli eventi condizionano la bontà delle azioni, gli altri due sono posti nell'intento di riprodurre la presenza di informazioni fuorvianti, tipica dei casi reali. (cosiddetto "rumore di fondo dell'ambiente").

Quando l'agente sperimenti per almeno 3000 interazioni un comportamento corretto, viene immessa una variazione nell'ambiente, che non influisce sugli stati, tale da modificare gli effetti delle azioni proposte. In pratica si assume l'esistenza di un quinto fattore, significativo e poco dinamico, sconosciuto all'agente. La sua presenza permette di stabilire le capacità del *classifier* di rielaborare la propria conoscenza per raggiungere, nuovamente, livelli ottimali.

Per l'elaborazione i parametri sono stati valorizzati come segue:

```
numberOfRules       = 32
numberOfEffectors   = 4
geneLength          = 4
maxNumberOfMessages = 1
effectorsFlag       = 0
wildCardRate        = 0.3
```

```
confidence          = 0.001
initialStrength     = 10

bidTaxRate          = 0.25
lifeTaxRate         = 0.02
bidRatio            = 0.25
linearBid1          = 0.25
linearBid2          = 1
effectiveLinearBid1 = 0.25
effectiveLinearBid2 = 1
bidSigma            = 0.075
bidMu               = 0

evolutionRate       = 1
turnoverRate        = 0.1
crossoverRate       = 1
mutationRate        = 0.002
crowdingRate        = 0.2
crowdingFactor      = 0.2
```

I valori elencati non sono ottimali, ma il *classifier* riesce egualmente ad apprendere: ciò dimostra la robustezza e la duttilità d'impiego del metodo. Il numero di regole, ad esempio, è sovrabbondante: probabilmente, otto sarebbero state sufficienti, quattro regole esatte più quattro per evitare la distruzione delle prime durante l'azione dell'algoritmo genetico. Avendo deciso di non promuovere cooperazione fra le regole, il massimo numero di messaggi è stato posto ad uno: si sarebbe potuto usare l'*effectorsFlag* per associare direttamente l'azione alle regole, così, invece, alcune di esse potranno contenere messaggi inutili. Poiché ogni azione può essere, esclusivamente, giusta o sbagliata, ogni regola può essere giudicata, efficacemente, dopo la prima interazione con l'ambiente: *bidTax* e *bidRatio* potrebbero, dunque, essere molto più elevate, in modo che il patrimonio delle regole errate si riduca rapidamente.

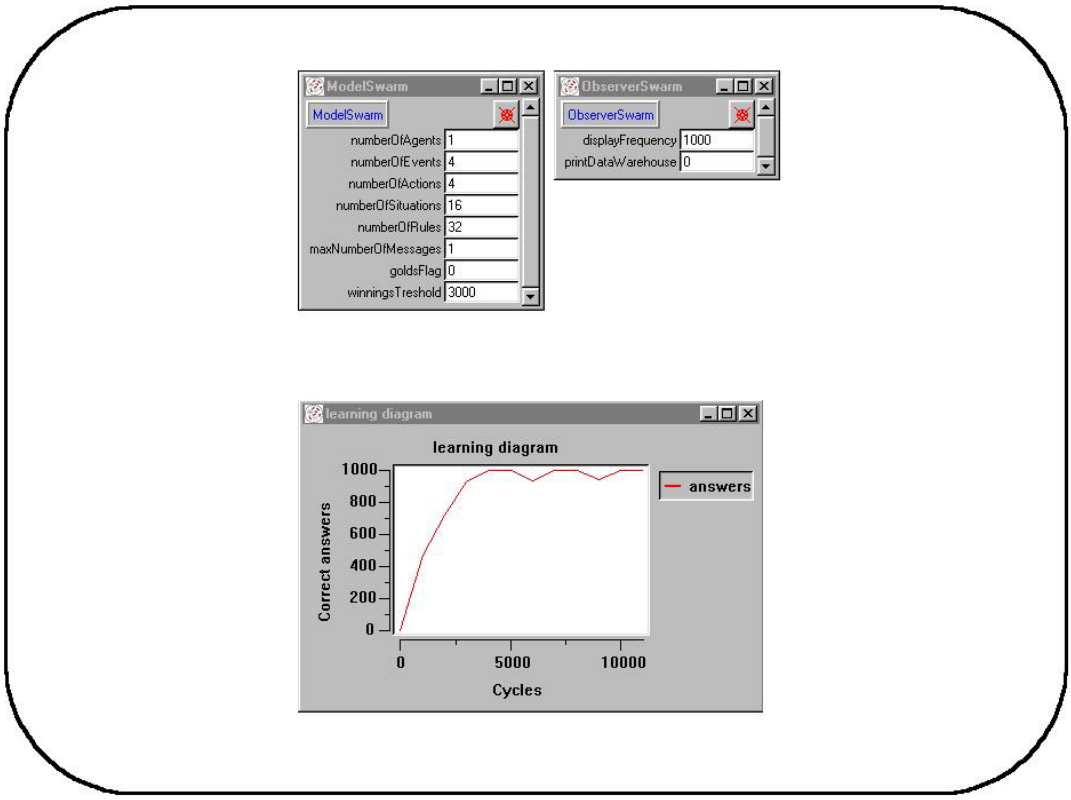
La figura 3.5 mostra un completo apprendimento dell'agente: il grafico riporta il numero di azioni corrette eseguite nell'intervallo di osservazione; l'ampiezza dell'intervallo è stabilita dal valore di *displayFrequency*, fornito nella *probe-map* dell'*observerSwarm*. Ricordando che la *lottery* ha modificato il valore di *bias*, cioè cambiato il giudizio di bontà dell'azione, dopo 3000 risposte corrette consecutive, e poiché ciò è avvenuto, per la prima volta, nell'intervallo fra i 5000 ed i 6000 cicli, è possibile stabilire che tutte le regole corrette sono state elaborate in poco più di 3000 cicli.

Come si nota, il *classifier* è stato in grado di adeguare le regole al nuovo sistema di valutazione; dopo la riduzione di risposte esatte, conseguente alla variazione, esso è tornato, rapidamente, ad esprimere prestazioni ottimali. Ciò dimostra, inoltre, come il meccanismo di inibizione dell'algoritmo genetico, il parametro *confidence* stabilisce che l'algoritmo genetico venga inibito dopo 1000 risposte corrette, non comporti riduzione delle capacità di reazione del *classifier* a mutamenti dell'ambiente. Durante l'elaborazione, CW ha inibito e riavviato il meccanismo evolutivo per tre volte, producendo, dopo ogni variazione, un insieme di regole diverso da quello posseduto.

Di seguito sono riportati i dati, relativi allo stato delle regole dell'agente, forniti da CW in seguito alla richiesta dell'utilizzatore, espressa assegnando uno come valore del parametro *printDataWarehouse* nella *probe-map* dell'*ObserverSwarm*. La stampa è stata effettuata dopo ciascun intervallo di osservazione, il formato adottato è già stato descritto nel paragrafo dedicato a CW. Sono stati sottolineati i dati relativi al numero di successi, ottenuti di volta in volta dall'agente, e le regole corrette, sviluppate ad apprendimento avvenuto. Nella stampa compare anche il messaggio della *lottery,* che notifica l'avvenuta modificazione del valore di *bias*. CW ha stampato il contenuto del *dataWarehouse* ogni 1000 cicli ma, per non appesantire l'esposizione, sono stati riportati solo i dati relativi: allo stato iniziale, contenuto del *dataWarehouse* dopo il primo ciclo di elaborazione, ed alle regole sviluppate ad apprendimento completato; per gli altri intervalli di osservazione sono stati conservati solo il numero di successi dell'agente e le statistiche del *classifier*. A proposito di queste ultime, occorre rammentare che la contatura avviene dall'inizio dell'elaborazione e non dall'inizio del

ciclo di osservazione, come per il numero di successi dell'agente.

Figura 3.5 - Esempio 1




```
Agent 0 has won 0 times in the latest period

      4973988 - DataWarehouse cointains following data:
      49702C8 - RuleList contains 32 rules:
       49750A8 strength: 4.225000 specificity 1.000000
      1001
      0011
      My partnerList contains 1 entries for:
      4970DA8
       4975320 strength: 9.800000 specificity 0.625000
      ##00
      #100
      My partnerList contains 0 entries for:
       4975810 strength: 9.800000 specificity 0.500000
      11##
      1##1
      My partnerList contains 0 entries for:
       4976468 strength: 9.800000 specificity 0.750000
      #0#0
      1111
      My partnerList contains 0 entries for:
       4976958 strength: 9.800000 specificity 0.625000
      #0#0
      1#00
      My partnerList contains 0 entries for:
       4976BD0 strength: 9.800000 specificity 0.625000
      ##1#
```

```
1101
My partnerList contains 0 entries for:
 4976E48 strength: 9.800000 specificity 0.750000
##10
1111
My partnerList contains 0 entries for:
 49770C0 strength: 9.800000 specificity 1.000000
0110
0110
My partnerList contains 0 entries for:
 4977338 strength: 9.800000 specificity 0.750000
0##0
0100
My partnerList contains 0 entries for:
 4978480 strength: 9.800000 specificity 0.500000
111#
##1#
My partnerList contains 0 entries for:
 49790D8 strength: 7.350000 specificity 0.625000
1##1
11#0
My partnerList contains 1 entries for:
 4970DA8
 4975598 strength: 9.800000 specificity 0.625000
0###
0110
My partnerList contains 0 entries for:
 4977828 strength: 9.800000 specificity 0.625000
0#01
#1#0
My partnerList contains 0 entries for:
 49761F0 strength: 9.800000 specificity 0.750000
##10
0100
My partnerList contains 0 entries for:
 4977D18 strength: 9.800000 specificity 0.625000
11#0
0#1#
My partnerList contains 0 entries for:
 4978208 strength: 7.350000 specificity 0.500000
1##1
##11
My partnerList contains 1 entries for:
 4970DA8
 4978970 strength: 7.350000 specificity 0.125000
##0#
####
My partnerList contains 1 entries for:
 4970DA8
 49786F8 strength: 7.350000 specificity 0.625000
#00#
1#10
My partnerList contains 1 entries for:
 4970DA8
 4977F90 strength: 9.800000 specificity 0.625000
1#1#
100#
My partnerList contains 0 entries for:
 4979350 strength: 9.800000 specificity 0.750000
```

```
#111
10#1
My partnerList contains 0 entries for:
 4974E30 strength: 9.800000 specificity 0.875000
00#1
0111
My partnerList contains 0 entries for:
 4975F78 strength: 9.800000 specificity 0.625000
1110
#0##
My partnerList contains 0 entries for:
 49766E0 strength: 7.350000 specificity 0.625000
1#0#
#110
My partnerList contains 1 entries for:
 4970DA8
 4978E60 strength: 9.800000 specificity 0.625000
11##
11#0
My partnerList contains 0 entries for:
 49795C8 strength: 7.350000 specificity 0.500000
1###
#101
My partnerList contains 1 entries for:
 4970DA8
 4974900 strength: 9.800000 specificity 0.875000
0#10
0000
My partnerList contains 0 entries for:
 4975D00 strength: 9.800000 specificity 0.875000
1101
111#
My partnerList contains 0 entries for:
 4977AA0 strength: 9.800000 specificity 0.625000
00#0
0#1#
My partnerList contains 0 entries for:
 4978BE8 strength: 9.800000 specificity 0.875000
101#
0000
My partnerList contains 0 entries for:
 4979840 strength: 9.800000 specificity 0.500000
#0#0
#00#
My partnerList contains 0 entries for:
 4975A88 strength: 9.800000 specificity 0.500000
11#1
1###
My partnerList contains 0 entries for:
 49775B0 strength: 7.350000 specificity 0.875000
1001
1#00
My partnerList contains 1 entries for:
 4970DA8

 49718F0 - EffectorList contains 4 effectors:
 49746A0 suggestion 2
0010
My partnerList contains 0 entries for:
```

```
 4974570 suggestion 1
0001
My partnerList contains 0 entries for:
 49747D0 suggestion 3
0011
My partnerList contains 1 entries for:
 49750A8
 4974120 suggestion 0
0000
My partnerList contains 0 entries for:

 4971CE0 - MessageList contains 0 messages:

 4971D70 - MatchList contains 1 rules:
 49747D0 suggestion 3
0011
My partnerList contains 1 entries for:
 49750A8

 49722B0 - WinnerList contains 1 rules:
 49750A8 strength: 4.225000 specificity 1.000000
1001
0011
My partnerList contains 1 entries for:
 4970DA8

 495A098 - WorkList contains 0 rules:

 4970DA8 - Treasury balance:
Current amount of treasury is 29.125000

current active effector is:
 49747D0 suggestion 3
0011
My partnerList contains 1 entries for:
 49750A8

Statistics:
Rule's selection has been performed 1 times
CoverDetector has been applied 0 times
CoverEffector has been applied 1 times
Rule's genetic evolution has been performed 0 times
```

Agent 0 has won 470 times in the latest period

```
Statistics:
Rule's selection has been performed 1001 times
CoverDetector has been applied 5 times
CoverEffector has been applied 22 times
Rule's genetic evolution has been performed 29 times
```

Agent 0 has won 719 times in the latest period

```
Statistics:
Rule's selection has been performed 2001 times
CoverDetector has been applied 6 times
CoverEffector has been applied 22 times
Rule's genetic evolution has been performed 58 times
```

Agent 0 has won 924 times in the latest period

Statistics:
Rule's selection has been performed 3001 times
CoverDetector has been applied 7 times
CoverEffector has been applied 23 times
Rule's genetic evolution has been performed 72 times

Agent 0 has won 1000 times in the latest period

4973988 - DataWarehouse cointains following data:
49702C8 - RuleList contains 32 rules:
4978BB8 strength: 0.000000 specificity 0.750000
#010
0#11
My partnerList contains 0 entries for:
4974900 strength: 1.734602 specificity 0.750000
##01
0010
My partnerList contains 0 entries for:
4976938 strength: 0.000000 specificity 0.750000
0010
0#1#
My partnerList contains 0 entries for:
4978970 strength: 0.000000 specificity 0.500000
##1#
0#01
My partnerList contains 1 entries for:
4970DA8
4978698 strength: 0.000000 specificity 0.750000
#010
001#
My partnerList contains 0 entries for:
49757A8 strength: 0.000000 specificity 0.625000
##00
000#
My partnerList contains 0 entries for:
49775A0 strength: 0.000000 specificity 1.000000
1101
0010
My partnerList contains 0 entries for:
49965A0 strength: 0.000000 specificity 0.625000
##01
0#00
My partnerList contains 0 entries for:
4978E40 strength: 0.000000 specificity 0.875000
#011
0011
My partnerList contains 1 entries for:
4970DA8
49967E8 strength: 1.914541 specificity 0.625000
##11
0#00
My partnerList contains 1 entries for:
4970DA8
49781D0 strength: 0.000000 specificity 0.750000
##11
0001

```
My partnerList contains 1 entries for:
 4970DA8
 4975030 strength: 0.000000 specificity 0.875000
#000
0001
My partnerList contains 0 entries for:
 4977808 strength: 0.000000 specificity 0.500000
01#1
###0
My partnerList contains 0 entries for:
 4976660 strength: 0.000000 specificity 0.500000
##01
0#1#
My partnerList contains 0 entries for:
 4977CC8 strength: 0.000000 specificity 0.625000
##00
0#01
My partnerList contains 0 entries for:
 4996A30 strength: 0.000000 specificity 0.375000
01##
###1
My partnerList contains 0 entries for:
 4979568 strength: 0.000000 specificity 0.625000
##10
0#00
My partnerList contains 0 entries for:
 4977080 strength: 0.000000 specificity 0.625000
0010
##1#
My partnerList contains 0 entries for:
 4977318 strength: 1.789581 specificity 0.750000
##00
0001
My partnerList contains 0 entries for:
 49797B0 strength: 0.000000 specificity 0.750000
##01
0000
My partnerList contains 0 entries for:
 4977F10 strength: 1.737049 specificity 0.750000
##10
0011
My partnerList contains 0 entries for:
 4977A80 strength: 0.000000 specificity 0.625000
#010
0#1#
My partnerList contains 0 entries for:
 4979098 strength: 0.000000 specificity 0.750000
##01
0001
My partnerList contains 0 entries for:
 4975CE0 strength: 0.000000 specificity 0.750000
0#00
#001
My partnerList contains 0 entries for:
 4996358 strength: 0.000000 specificity 0.750000
0#00
0#11
My partnerList contains 0 entries for:
 4976B80 strength: 0.000000 specificity 0.500000
```

```
##00
0#1#
My partnerList contains 0 entries for:
 4978438 strength: 0.000000 specificity 0.750000
##11
0010
My partnerList contains 1 entries for:
 4970DA8
 4975A28 strength: 0.000000 specificity 0.875000
#011
0010
My partnerList contains 1 entries for:
 4970DA8
 4976190 strength: 0.000000 specificity 0.625000
##10
0#01
My partnerList contains 0 entries for:
 4976DC8 strength: 0.000000 specificity 0.875000
#010
0011
My partnerList contains 0 entries for:
 49752F8 strength: 0.000000 specificity 0.625000
##00
0#00
My partnerList contains 0 entries for:
 4975F28 strength: 0.000000 specificity 0.875000
#010
0010
My partnerList contains 0 entries for:

 49718F0 - EffectorList contains 4 effectors:
 49746A0 suggestion 2
0010
My partnerList contains 0 entries for:
 4974120 suggestion 0
0000
My partnerList contains 1 entries for:
 49967E8
 49747D0 suggestion 3
0011
My partnerList contains 0 entries for:
 4974570 suggestion 1
0001
My partnerList contains 0 entries for:

 4971CE0 - MessageList contains 0 messages:

 4971D70 - MatchList contains 1 rules:
 4974120 suggestion 0
0000
My partnerList contains 1 entries for:
 49967E8

 49722B0 - WinnerList contains 1 rules:
 49967E8 strength: 1.914541 specificity 0.625000
##11
0#00
My partnerList contains 1 entries for:
 4970DA8
```

495A098 - WorkList contains 0 rules:

4970DA8 - Treasury balance:
Current amount of treasury is 494.866651

current active effector is:
4974120 suggestion 0
0000
My partnerList contains 1 entries for:
49967E8

Statistics:
Rule's selection has been performed 4001 times
CoverDetector has been applied 7 times
CoverEffector has been applied 23 times
Rule's genetic evolution has been performed 72 times

Agent 0 has won 1000 times in the latest period

Statistics:
Rule's selection has been performed 5001 times
CoverDetector has been applied 7 times
CoverEffector has been applied 23 times
Rule's genetic evolution has been performed 72 times

Lottery is switching manipulation bias = 0

Agent 0 has won 937 times in the latest period

Statistics:
Rule's selection has been performed 6001 times
CoverDetector has been applied 7 times
CoverEffector has been applied 23 times
Rule's genetic evolution has been performed 80 times

Agent 0 has won 1000 times in the latest period

4973988 - DataWarehouse cointains following data:
49702C8 - RuleList contains 32 rules:
49965A0 strength: 0.000000 specificity 0.875000
#010
0000
My partnerList contains 0 entries for:
4978BB8 strength: 0.000000 specificity 0.750000
#010
0#11
My partnerList contains 0 entries for:
4976B80 strength: 0.000000 specificity 0.875000
#001
0010
My partnerList contains 0 entries for:
4977080 strength: 0.000000 specificity 0.625000
0010
##1#
My partnerList contains 0 entries for:
4976938 strength: 0.000000 specificity 0.750000
0010
0#1#

```
My partnerList contains 0 entries for:
 4996C78 strength: 0.000000 specificity 0.750000
##00
0010
My partnerList contains 0 entries for:
 4976660 strength: 0.000000 specificity 0.500000
##01
0#1#
My partnerList contains 0 entries for:
 49781D0 strength: 0.000000 specificity 0.625000
##11
0#10
My partnerList contains 1 entries for:
 4970DA8
 49967E8 strength: 0.000000 specificity 0.625000
##11
0#00
My partnerList contains 1 entries for:
 4970DA8
 4977CC8 strength: 0.000000 specificity 0.750000
##10
0000
My partnerList contains 0 entries for:
 4977318 strength: 0.000000 specificity 0.750000
##00
0001
My partnerList contains 0 entries for:
 49752F8 strength: 0.000000 specificity 0.625000
##00
0#00
My partnerList contains 0 entries for:
 49757A8 strength: 0.000000 specificity 0.625000
##00
000#
My partnerList contains 0 entries for:
 4979098 strength: 1.785028 specificity 0.750000
##01
0001
My partnerList contains 0 entries for:
 4978E40 strength: 0.000000 specificity 0.875000
#011
0011
My partnerList contains 1 entries for:
 4970DA8
 49797B0 strength: 1.616694 specificity 0.750000
##10
0010
My partnerList contains 0 entries for:
 4996A30 strength: 0.000000 specificity 0.375000
01##
###1
My partnerList contains 0 entries for:
 49775A0 strength: 0.000000 specificity 1.000000
1101
0010
My partnerList contains 0 entries for:
 4978438 strength: 0.000000 specificity 0.625000
##01
0#01
```

My partnerList contains 0 entries for:
 4996358 strength: 0.000000 specificity 0.750000
##10
0001
My partnerList contains 0 entries for:
 4976190 strength: 1.775873 specificity 0.750000
##00
0000
My partnerList contains 0 entries for:
 4996EC0 strength: 0.000000 specificity 0.750000
##11
0000
My partnerList contains 1 entries for:
 4970DA8
 49763F8 strength: 1.841016 specificity 0.750000
##11
0011
My partnerList contains 1 entries for:
 4970DA8
 4975CE0 strength: 0.000000 specificity 0.875000
#001
0011
My partnerList contains 0 entries for:
 4978680 strength: 0.000000 specificity 0.875000
#000
0000
My partnerList contains 0 entries for:
 4979568 strength: 0.000000 specificity 0.750000
##01
0011
My partnerList contains 0 entries for:
 4978970 strength: 0.000000 specificity 0.625000
##10
0#00
My partnerList contains 0 entries for:
 49759F0 strength: 0.000000 specificity 0.750000
##00
0011
My partnerList contains 0 entries for:
 4977808 strength: 0.000000 specificity 0.500000
01#1
###0
My partnerList contains 0 entries for:
 4975F28 strength: 0.000000 specificity 0.875000
#010
0010
My partnerList contains 0 entries for:
 4977F10 strength: 0.000000 specificity 0.750000
##10
0011
My partnerList contains 0 entries for:
 4974900 strength: 0.000000 specificity 0.750000
##01
0010
My partnerList contains 0 entries for:

 49718F0 - EffectorList contains 4 effectors:
 49746A0 suggestion 2
0010

My partnerList contains 0 entries for:
 4974570 suggestion 1
0001
My partnerList contains 0 entries for:
 4974120 suggestion 0
0000
My partnerList contains 0 entries for:
 49747D0 suggestion 3
0011
My partnerList contains 1 entries for:
 49763F8

 4971CE0 - MessageList contains 0 messages:

 4971D70 - MatchList contains 1 rules:
 49747D0 suggestion 3
0011
My partnerList contains 1 entries for:
 49763F8

 49722B0 - WinnerList contains 1 rules:
 49763F8 strength: 1.841016 specificity 0.750000
##11
0011
My partnerList contains 1 entries for:
 4970DA8

 495A098 - WorkList contains 0 rules:

 4970DA8 - Treasury balance:
Current amount of treasury is 518.342916

current active effector is:
 49747D0 suggestion 3
0011
My partnerList contains 1 entries for:
 49763F8

Statistics:
Rule's selection has been performed 7001 times
CoverDetector has been applied 7 times
CoverEffector has been applied 23 times
Rule's genetic evolution has been performed 80 times

Agent 0 has won 1000 times in the latest period

Statistics:
Rule's selection has been performed 8001 times
CoverDetector has been applied 7 times
CoverEffector has been applied 23 times
Rule's genetic evolution has been performed 80 times

Lottery is switching manipulation bias = 1

Agent 0 has won 940 times in the latest period

Statistics:
Rule's selection has been performed 9001 times
CoverDetector has been applied 7 times

```
      CoverEffector has been applied 24 times
      Rule's genetic evolution has been performed 88 times

Agent 0 has won 1000 times in the latest period

      Statistics:
      Rule's selection has been performed 10001 times
      CoverDetector has been applied 7 times
      CoverEffector has been applied 24 times
      Rule's genetic evolution has been performed 88 times

Agent 0 has won 1000 times in the latest period

      Statistics:
      Rule's selection has been performed 11001 times
      CoverDetector has been applied 7 times
      CoverEffector has been applied 24 times
      Rule's genetic evolution has been performed 88 times
```

Il valore iniziale del *bias* è uno; poiché le risposte corrette sono calcolate come resto della divisione di un numero *random*, rappresentativo dello stato dell'ambiente, aumentato del *bias* per il numero di azioni possibili, le regole corrette sono: '0000:0001', '0001:0010', '0010:0011', '0011:0000', '0100:0001', '0101:0010', '0110:0011', '0111:0000', '1000:0001' e così via. Leggendo le due parti come numeri binari, il meccanismo di calcolo è palese: se 1 allora 2, cioè il resto della divisione di 1 + 1 per quattro fornisce 2, se 2 allora 3 eccetera.

CW ha sviluppato solo quattro regole, tutte recanti due caratteri di indifferenza, #, nelle prime due posizioni, che, nella codifica binaria, sono poste ad uno solo per i multipli di quattro: si potrebbe, quindi, dire che ha "capito" che quelle due informazioni non influivano sul risultato.

Dopo i primi 5000 cicli, dato il livello di apprendimento raggiunto dall'agente, la *lottery* ha modificato il valore del *bias*: le risposte corrette sono diventate: se 0 allora 0, se 1 allora 1, se 2 allora 2, se 3 allora 3, se 4 allora 0 eccetera. A fronte di questa novità, il *classifier* è stato capace di esprimere, rapidamente, nuove regole, conservando la conoscenza relativa all'indifferenza dei primi due valori; come si evince dalle statistiche, il *ruleMaker*, che negli ultimi mille cicli non aveva più agito, è stato nuovamente attivato, per scoprire le regole più "adatte" a guidare l'operato dell'agente nel nuovo scenario.

3.8.2 - Esempio 2

Il caso studiato è quello presentato a proposito dell'esempio precedente, ma sono stati variati alcuni parametri: la *lifeTax* è stata raddoppiata, da 0.002 a 0.004, il *turnoverRate* è stato portato a 0.2 e, soprattutto, *crowdingRate* e *crowdingFactor* sono stati elevati a 0.9. Le variazioni, quindi, hanno riguardato, in prevalenza, la conduzione dell'algoritmo genetico; l'aumento di *lifeTax* ha poco effetto in un caso come questo: il messaggio che codifica la percezione ambientale dell'agente usa tutte le posizioni del genoma e, quindi, è improbabile che si sviluppino regole sistematicamente incapaci di *match*.

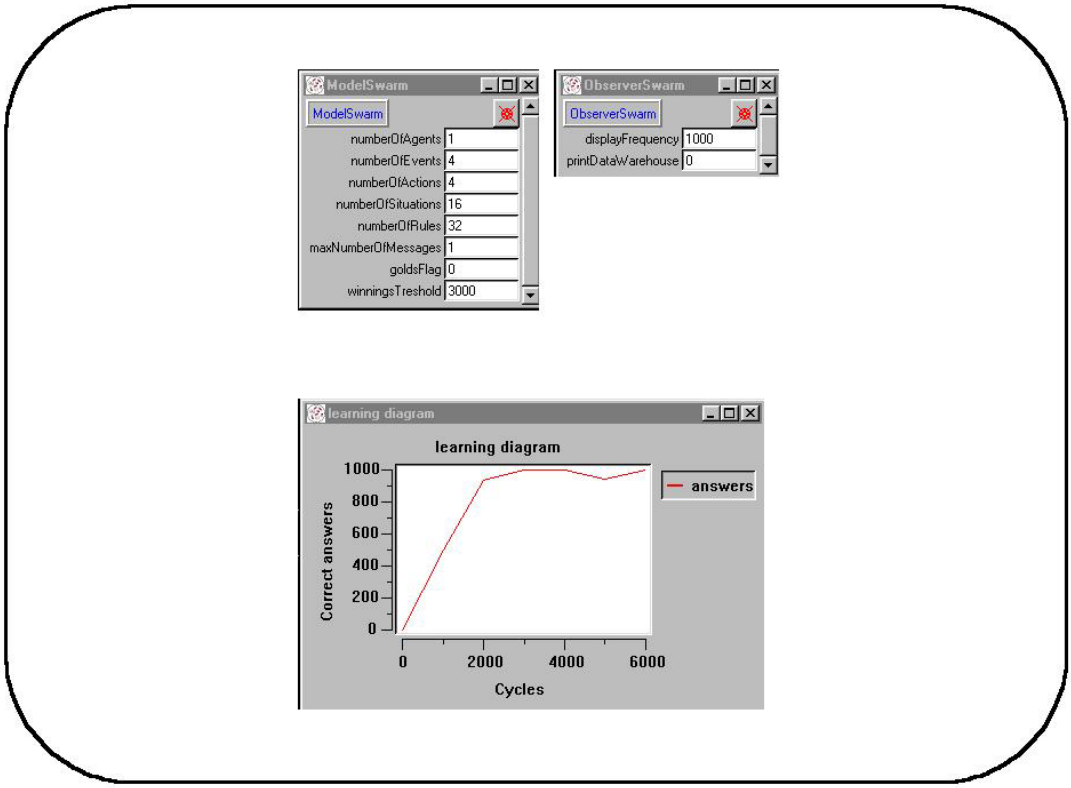
Il più forte ricambio generazionale, derivante dall'aumento di *turnoverRate*, ha contribuito a rendere più rapida la ricerca; valori così alti sono, però, sostenibili solo in presenza di un basso rapporto, inferiore ad uno, fra numero di stati dell'ambiente e numero di cicli fra due evoluzioni consecutive. Se l'ambiente potesse assumere molti stati diversi e l'evoluzione avvenisse troppo frequentemente, le nuove regole rischierebbero di non poter agire, prima dell'evoluzione successiva, perché volte a rispondere ad uno stato non verificatosi; in questo caso il loro patrimonio risulterebbe ridotto, in forza dell'applicazione della *lifeTax*, senza che esse avessero avuto la possibilità di

incrementarlo proponendo un'azione. Si potrebbero ottenere, in una evoluzione, regole corrette che, poiché non si è verificata la condizione di attivazione, verrebbero trattate come regole errate dal *ruleMaker* durante l'evoluzione successiva. La frequenza di evoluzione è, in questo caso, 1/32 mentre gli stati sono sedici, quindi è tollerabile una *lifeTax* piuttosto elevata.

La maggior estensione del campione, nella ricerca delle regole da eliminare, aumenta la selettività dell'evoluzione; ciò, pur allungando i tempi di elaborazione per le singole riproduzioni, permette di rendere il ricambio più efficace. Valori così alti, per altro, potrebbero risultare poco adatti sia in casi di patrimoni genetici formati da molte regole, per il notevole aumento dell'onere di elaborazione, sia quando fossero utilizzate regole cooperative.

Il diagramma in figura 3.6 rivela il compimento di alcuni progressi, rispetto all'esempio precedente, sia nel raggiungimento del livello massimo di successi, sia nel cammino verso di esso: il *classifier* giunge ai 1000 successi, nel periodo di osservazione, con un anticipo di circa 1000 cicli e consegue, fin dalla prima osservazione risultati migliori.

Figura 3.6 - Esempio 2



In questo caso non è stata richiesta la stampa delle regole, *printDataWarehouse* vale 0, quindi CW ha prodotto, oltre ai grafici, solo le indicazioni relative al livello di successi raggiunto in ogni ciclo di osservazione, come di seguito riportato:

```
Agent 0 has won 0 times in the latest period
Agent 0 has won 500 times in the latest period
Agent 0 has won 936 times in the latest period
Agent 0 has won 1000 times in the latest period
Agent 0 has won 1000 times in the latest period

Lottery is switching manipulation bias = 0

Agent 0 has won 940 times in the latest period
```

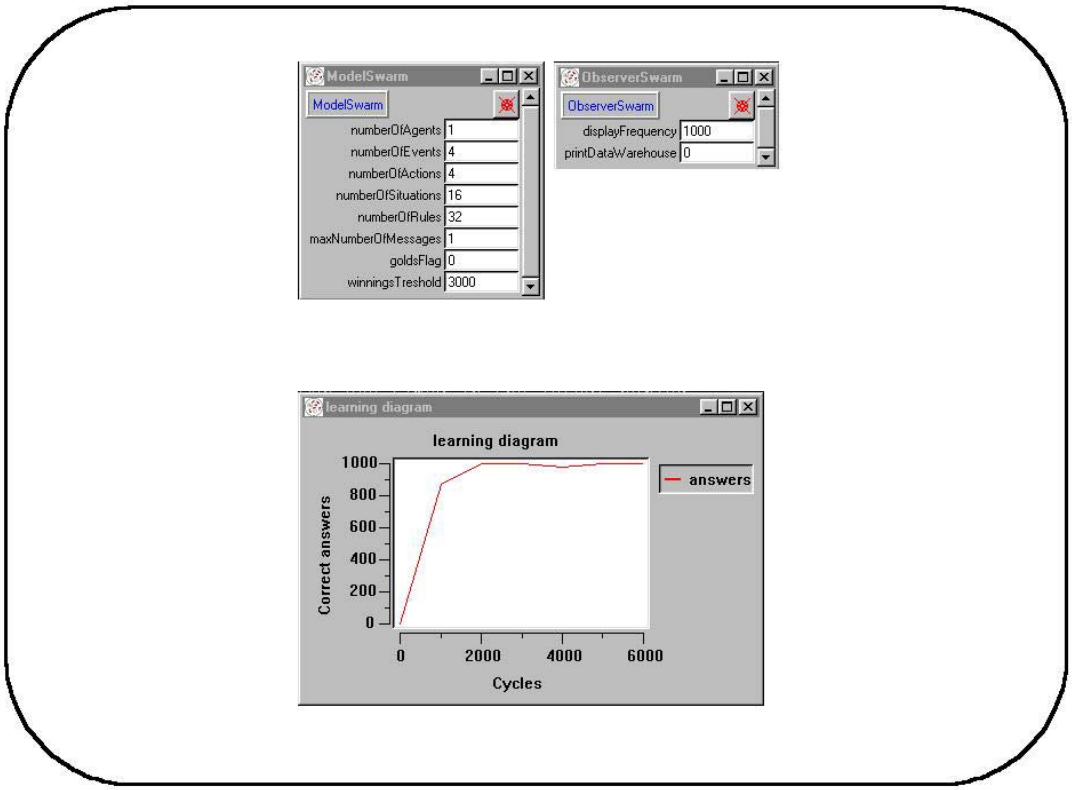
```
Agent 0 has won 1000 times in the latest period
```

Anche dalla disamina dei dati numerici, risulta che la maggior selettività ha permesso un più rapido conseguimento dei risultati ottimali; il dato, però, va interpretato tenendo conto delle peculiarità del caso: le regole sono molto semplici e la loro valutazione è immediata, tanto che il *reward* assegnato dall'agente assume unicamente i valori zero ed uno. Particolarmente quando la valutazione delle azioni risulti piuttosto complessa o difficoltosa, una selettività elevata può andare a discapito della ricerca di novità rallentando, se non addirittura impedendo, l'apprendimento: la configurazione di parametri, sperimentata con il *workbench*, va quindi affinata, in seguito, adattandola al modello vero e proprio.

3.8.3 - Esempio 3

Visti i buoni risultati ottenuti con un algoritmo genetico più selettivo, agendo sempre sul caso dell'esempio 1, si è reso maggiormente aggressivo anche il comportamento del meccanismo di distribuzione dei crediti. Riprendendo le osservazioni fatte a proposito dei valori usati per l'esempio 1, si sono sfruttate le possibilità, allora sottolineate, di inasprire il sistema fiscale, favorendo il rapido decadimento delle regole errate.

Figura 3.7 - Esempio 3



Ai ritocchi effettuati per l'esempio 2, si sono aggiunti il raddoppio della *bidTax* e della *bidRatio*. La *bidTax*, rappresentando il costo di partecipazione all'asta, permette, tramite il suo inasprimento, di accelerare l'impoverimento delle regole portatrici di messaggi inutili o assurdi, incapaci, cioè, di attivare *effector* od altre regole. La sua azione è particolarmente importante quando, come in questo caso, le regole non siano portatrici di un'*action-part* direttamente rappresentativa di una azione, l'*effectorsFlag* (indicato nella *probe-map* del *modelSwarm* come *goldsFlag* in riconoscimento del lavoro di Goldberg) vale zero, e, quindi, possono svilupparsi individui portatori di

indicazioni inutili. Utilizzando regole alla Goldberg, invece, prevarrebbe il secondo effetto della *bidTax*: l'impoverimento delle regole molto generalizzate. Regole generali, portatrici di molte *wildcard*, sono in grado di soddisfare facilmente il *match* con i messaggi ambientali; sono, però, penalizzate dal meccanismo di calcolo dell'offerta per l'asta, basato sulla specificità: esse hanno alta probabilità di partecipare all'asta ma ridotta possibilità di vincere. Con valori elevati di *bidTax* questa loro caratteristica ne provoca un rapido impoverimento ed il *classifier* tende a produrre regole maggiormente "precise". Nel caso specifico il *classifier* ha sperimentato minor capacità di riassumere, in sole quattro regole contenenti due *wildcard* iniziali, la conoscenza acquisita.

L'aumento della *bidRatio* ha contribuito ad accentuare la perdita di patrimonio delle regole portatrici di messaggi errati. Nel caso in esame la scelta è stata corretta: dati i valori dei parametri, la regola, per immettere il proprio messaggio nella lista, è stata portata a perdere quasi l'ottanta percento del proprio patrimonio; il mancato recupero del patrimonio, derivante dall'attribuzione di una ricompensa pari a zero in caso di errore, ha permesso di ridurre la possibilità che la stessa regola sopravvivesse o potesse nuovamente contribuire a determinare le azioni del *classifier*.

Vale, anche in questo caso, l'osservazione, fatta a proposito dell'esempio 2, in merito all'esigenza di adeguare la configurazione dei parametri alle particolarità dei casi di utilizzo: valori alti di *bidTax* e *bidRatio* si adattano bene a funzioni di *reward* precise e fortemente discontinue; in casi diversi, valori eccessivi possono costituire un grave ostacolo all'apprendimento.

I risultati ottenuti appaiono chiari nel grafico in figura 3.7: già nei primi 1000 cicli il sistema ha fornito quasi 900 risposte adeguate; è, inoltre, migliorata la capacità di rispondere alle modificazioni del *bias*.

A conferma della rappresentazione grafica sono riportati i consueti dati numerici:

```
Agent 0 has won 0 times in the latest period
Agent 0 has won 870 times in the latest period
Agent 0 has won 1000 times in the latest period
Agent 0 has won 1000 times in the latest period

Lottery is switching manipulation bias = 0

Agent 0 has won 978 times in the latest period
Agent 0 has won 1000 times in the latest period
Agent 0 has won 1000 times in the latest period
```

Rispetto ai casi precedenti, si è avuto un notevole incremento delle risposte corrette, già nel primo periodo di osservazione: da 470 e 500 a 870; è, poi, aumentato anche il numero di successi dopo la modificazione del *bias*, da 937 e 940 a 978.

3.8.4 - Esempio 4

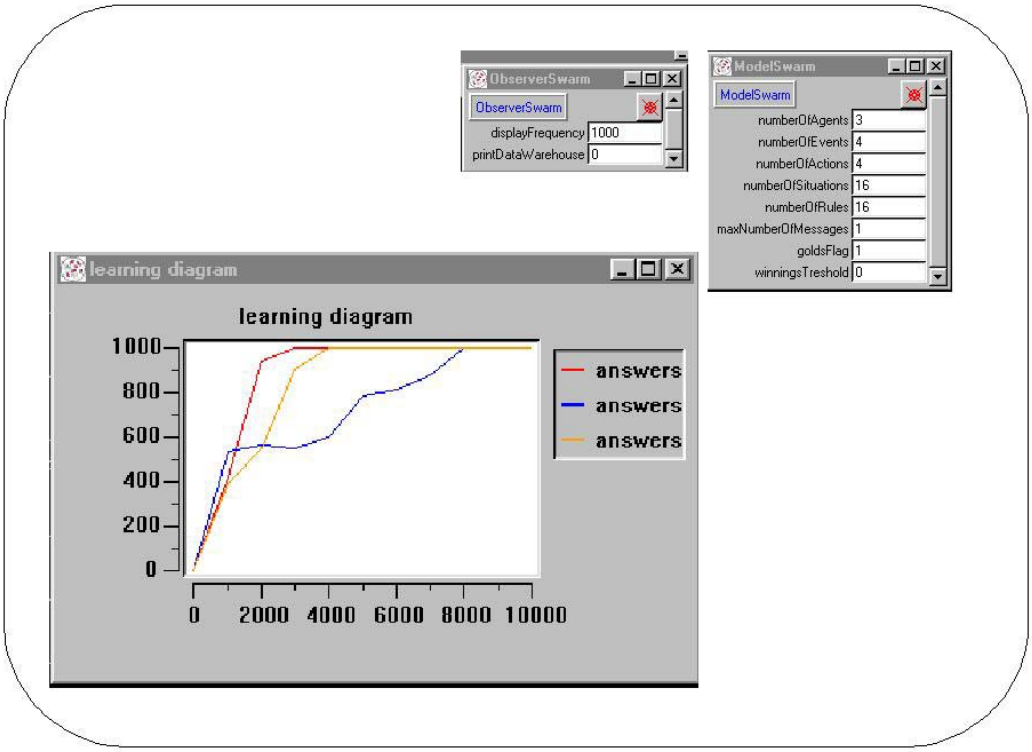
L'azione avviene ancora secondo il caso descritto nell'esempio 1, ma ad interagire con la *lottery* sono, ora, tre agenti. La distribuzione di frequenza delle varie informazioni, ottenute da ciascun agente, viene alterata dalla pluralità di essi: ogni agente richiede un numero diverso in ogni ciclo di azione, quindi ciascuno esamina un terzo dei casi che erano comunicati all'agente unico. Il meccanismo di interazione, tipico del *workbench*, è stato predisposto proprio per poter modificare, aumentando il numero di agenti, la distribuzione degli eventi percepiti da ciascuno, permettendo di conferire alla simulazione maggiore aderenza ai casi concreti di utilizzo di CW.

Data l'alterazione immessa, uno stesso stato dell'ambiente, sintetizzato nel numero *random*, può presentarsi con minore frequenza al singolo agente; la regola adatta a rispondere ad esso potrebbe, allora, essere eccessivamente indebolita da una *lifeTax* elevata: per questo motivo l'aliquota è stata ridotta all'uno percento.

Altra novità consiste nell'utilizzo dell'*effectorsFlag* e nella riduzione del numero di regole a sedici: grazie al meccanismo di verificazione della novità, che impedisce la produzione di regole identiche ad altre già esistenti, CW è in grado di operare con poche regole. La riduzione del numero di regole, inoltre, ben si accompagna all'utilizzo dell'*effectorsFlag* che, come illustrato in precedenza, rende impossibile la cooperazione di più regole.

La figura 3.8 permette di cogliere notevoli differenze nel processo di apprendimento dei vari agenti, confermando come l'utilizzo dello stesso *ruleMaster,* e *ruleMaker*, non influisca sulla indipendenza di ciascuno di essi. Ciò comprova, nei fatti, quanto detto, in precedenza, relativamente alla capacità di CW di condurre *classifier system* sostanzialmente diversi per ciascun agente. Tale diversità emerge, come in questo caso, pur con parametri di conduzione del *classifier* uguali per tutti gli agenti; ciò garantisce la possibilità di aumentare la differenziazione comportamentale intervenendo sul modello. Nel caso esaminato, inoltre, l'azione di ciascuno non influisce sui risultati altrui: ciò porta a prevedere un ulteriore arricchimento della complessità in modelli, dove le azioni dei singoli possano avere influenza sull'ambiente o sui risultati delle azioni di altri.

Figura 3.8 - Esempio 4



Tutti gli agenti giungono, sia pure in tempi diversi, al livello ottimale di apprendimento che viene mantenuto, data la disattivazione del meccanismo di variazione del *bias*, richiesta valorizzando a zero il parametro *winningsTreshold* nella *probe-map* del *ModelSwarm*. Il fenomeno è chiaramente leggibile anche nei dati numerici, riportati di seguito:

```
Agent 0 has won 0 times in the latest period
Agent 1 has won 1 times in the latest period
Agent 2 has won 0 times in the latest period

Agent 0 has won 422 times in the latest period
Agent 1 has won 537 times in the latest period
Agent 2 has won 393 times in the latest period
```

```
Agent 0 has won 945 times in the latest period
Agent 1 has won 567 times in the latest period
Agent 2 has won 547 times in the latest period

Agent 0 has won 1000 times in the latest period
Agent 1 has won 551 times in the latest period
Agent 2 has won 909 times in the latest period

Agent 0 has won 1000 times in the latest period
Agent 1 has won 598 times in the latest period
Agent 2 has won 1000 times in the latest period

Agent 0 has won 1000 times in the latest period
Agent 1 has won 782 times in the latest period
Agent 2 has won 1000 times in the latest period

Agent 0 has won 1000 times in the latest period
Agent 1 has won 814 times in the latest period
Agent 2 has won 1000 times in the latest period

Agent 0 has won 1000 times in the latest period
Agent 1 has won 879 times in the latest period
Agent 2 has won 1000 times in the latest period

Agent 0 has won 1000 times in the latest period
Agent 1 has won 1000 times in the latest period
Agent 2 has won 1000 times in the latest period

Agent 0 has won 1000 times in the latest period
Agent 1 has won 1000 times in the latest period
Agent 2 has won 1000 times in the latest period

Agent 0 has won 1000 times in the latest period
Agent 1 has won 1000 times in the latest period
Agent 2 has won 1000 times in the latest period
```

E' stato, inoltre, verificato che la ripetizione dell'elaborazione, con modificazione del valore di base usato nella generazione dei numeri casuali, cosiddetto "seme di randomizzazione", porta a cammini evolutivi diversi, dove a prevalere può essere indifferentemente uno dei tre agenti. Testimonianza dell'indipendenza dei tre può essere ravvisata anche nel "sorpasso" da parte dell'agente due che, inizialmente più lento nell'apprendere rispetto all'agente uno, è giunto, in questa elaborazione, al risultato ottimale prima di quello.

3.8.5 - Esempio 5

Il caso di studio è un po' più complesso di quello finora utilizzato: otto eventi, o condizioni, contribuiscono a configurare 250 stati diversi dell'ambiente; l'agente, per semplicità uno solo, è chiamato ad assumere sedici diversi comportamenti, in risposta ai vari stati. La scelta del numero di stati è stata effettuata in modo da non saturare tutte le possibili configurazioni ottenibili: in questo modo potranno formarsi regole "inutili", la cui *condition-part* non troverà corrispondenza in nessuno degli stati possibili dell'ambiente, e alcune azioni saranno adatte a rispondere ad un numero di stati dell'ambiente inferiore a quello di altre.

Il sistema deve selezionare le regole corrette nell'ambito di 43.046.721 schemi, teoricamente ottenibili: ciascun cromosoma è lungo otto posizioni che possono assumere valore '0', '1', '#', quindi gli

schemi ottenibili sono 3^8 per ciascuna parte della regola, risultando questa dall'associazione di due parti le possibilità divengono 3^{16} pari, appunto, a 43.046.721. Poiché dall'ambiente possono giungere 250 messaggi diversi e le azioni ammesse sono 16, solo un massimo di 4000 schemi potranno risultare corretti.

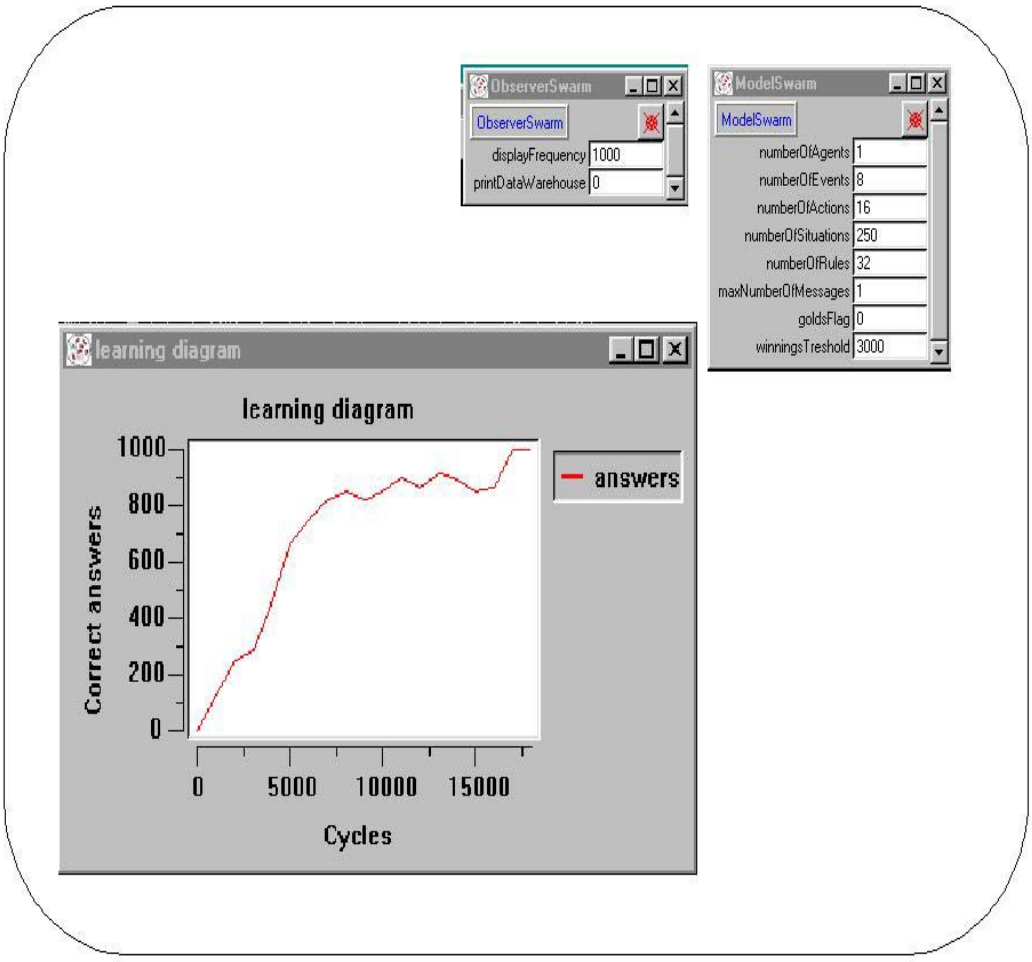
I parametri adottati sono quelli dell'esempio 1, con le seguenti eccezioni:

EvolutionRate = 0.05
CrowdingRate = 0.9
CrowdingFactor = 0.1

I maggiori valori di *crowdingRate* e *crowdingFactor*, come già descritto, servono ad aumentare la selettività dell'algoritmo genetico. Il valore di evolutionRate costituisce, invece, una novità: l'evoluzione avveniva, negli esempi precedenti, dopo un numero di cicli pari al numero di regole, in questo caso, invece, la decisione viene presa in modo stocastico, data una probabilità di evoluzione del cinque percento; ciò contribuisce a creare maggior movimento nella ricerca.

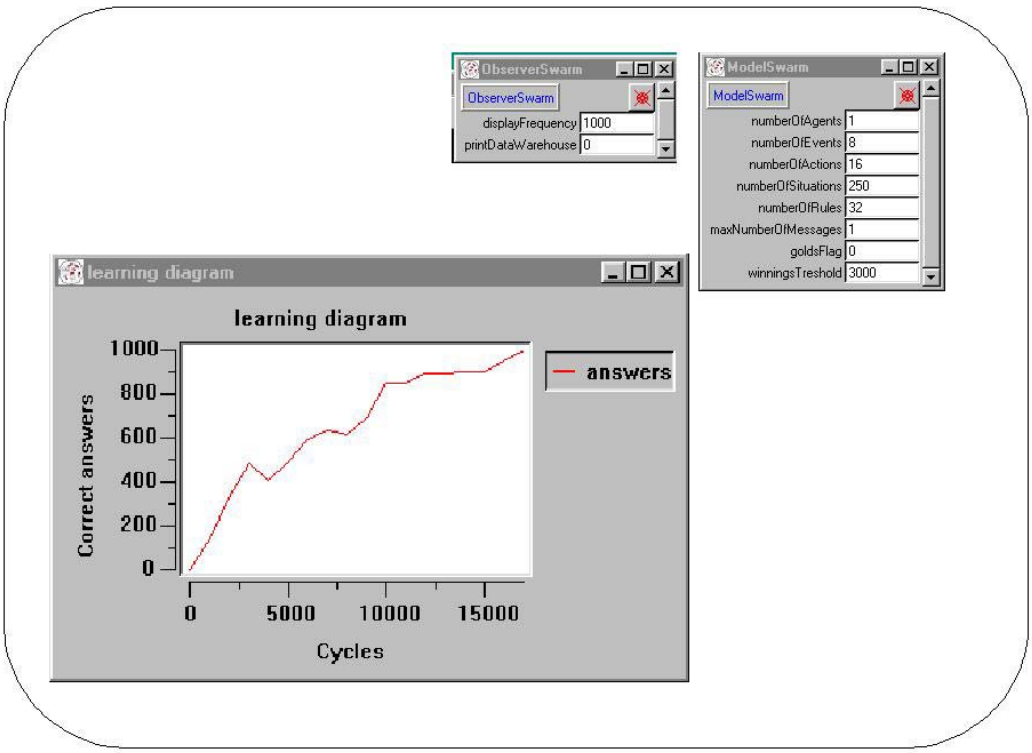
Come visibile nel grafico in figura 3.9, il *classifier* giunge a produrre regole corrette in poco meno di ventimila cicli.

Figura 3.9 - Esempio 5



I risultati migliorano leggermente adottando un *crowdingFactor* pari a 0.9, come illustrato dal grafico nella successiva figura 3.10

Figura 3.10 - Esempio 6



3.8.6 - Considerazioni sull'utilizzo di *workbench*

Da un punto di vista strettamente tecnico, gli esempi descritti dimostrano le caratteristiche di versatilità e robustezza del metodo. Queste emergono sia dalla possibilità di trattare problemi di complessità e difficoltà diverse, sia dall'ottenimento di risultati corretti, a prescindere, entro certa misura, dalla accuratezza delle scelte effettuate nell'attribuire i valori dei vari parametri.

Nei primi tre esempi si è giunti ad ottenere un completo "apprendimento" con diverse aliquote di tassazione e attraverso l'impiego di campioni variamente estesi per la selezione degli individui. L'esempio 1, pur essendo basato su valori dichiaratamente non ottimali dei parametri, ha fornito risultati di pieno soddisfacimento: l'utilizzo del metodo, perciò, può risultare proficuo anche per chi possedesse scarsa confidenza con esso.

I progressi ottenuti, variando pochi parametri, dimostrano la ricchezza delle possibilità offerte dalla semplice azione su di essi; tali opportunità risultano di facile utilizzo, poiché sfruttabili senza interventi sul codice dei programmi.

L'utilizzo di *workbench,* illustrato nell'esempio quattro, dimostra l'elevata indipendenza dei singoli *classifier*, il che permette di elevare il grado di plausibilità delle simulazioni basate su di essi.

Il rapporto fra numero di schemi ottenibili e numero di regole corrette, relativo all'esempio 5, permette di illustrare la superiorità delle ricerche basate sui metodi genetici, rispetto ad altri tipi. Inoltre, i significativi progressi ottenuti con l'intervento sui parametri del sistema di distribuzione dei crediti, che hanno caratterizzato l'esempio 3, dimostra l'importanza dell'arricchimento apportato dal metodo dei *classifier system* ai tradizionali algoritmi genetici.

L'azione del *workbench* è fortemente astratta; pure il suo impiego permette di esplorare le possibilità di applicazione del metodo a casi estremamente specifici. Il paradigma eventi, o condizioni, contro azioni, usato nell'interpretazione degli esempi, è solo uno dei tanti adottabili: adattando la funzione di calcolo della ricompensa, gli esempi potevano essere interpretati come espressione delle decisioni di domanda di un consumatore a fronte di un certo livello di prezzi, come politica di *pricing* di un monopolista a fronte della variazione delle condizioni di mercato, come meccanismo di reazione di un investitore a fronte di determinate aspettative sui corsi o sugli indici di borsa, ed altro ancora. I singoli cromosomi possono contenere un insieme di indicatori relativi a condizioni o eventi, allo stesso modo in cui possono contenere la traduzione binaria di determinati valori. Nel caso specifico il meccanismo di associazione di 4 azioni a 16 stati ambientali, può funzionare come meccanismo di associazione di quattro livelli di domanda a 16 livelli di un indice dei prezzi o a 16 prezzi diversi. L'associazione fra condizioni ed azioni può anche essere biunivoca, in modo da studiare le possibili reazioni, ad un stesso numero di condizioni o di valori di determinati indicatori. La simulazione condotta con il *workbench* può, perciò, costituire una buona base di partenza per la redazione di un modello, fortemente aderente alla realtà.

3.8.7 – Il formichiere di Ferraris

In questo modello CW viene usato sia come sistema esperto, sia come *learning classifier*. Le formiche si muovono in base a regole, codificate a priori, la cui applicazione è gestita dal *ruleMaster*, in guisa di motore d'inferenza; lo stesso oggetto *ruleMaster*, cooperando con il *ruleMaker*, provvede alla conduzione del *learning classifier* che guida le azioni del formichiere. La differenza di comportamento del programma è ottenuta codificando insiemi di parametri diversi, ciascuno dei quali risulta associato ad un *dataWarehouse*.

I parametri usati sono indicati di seguito, nella prima colonna quelli per le formiche e, accanto, quelli per il formichiere:

	Formiche	Formichiere
numberOfRules	= 8	8
numberOfEffectors	= 4	4
geneLength	= 4	4
maxNumberOfMessages	= 1	1
effectorsFlag	= 0	1
wildCardRate	= 0	0.3
confidence	= 0	0.001
initialStrength	= 10	10
bidTaxRate	= 0	0.2
lifeTaxRate	= 0	0.04
bidRatio	= 0	0.25
linearBid1	= 0	0.25
linearBid2	= 0	1
effectiveLinearBid1	= 0	0.25
effectiveLinearBid2	= 0	1
bidSigma	= 0	0.075
bidMu	= 0	0
evolutionRate	= 0	1
turnoverRate	= 0	0.1
crossoverRate	= 0	1
mutationRate	= 0	0.002
crowdingRate	= 0	0.4
crowdingFactor	= 0	0.4

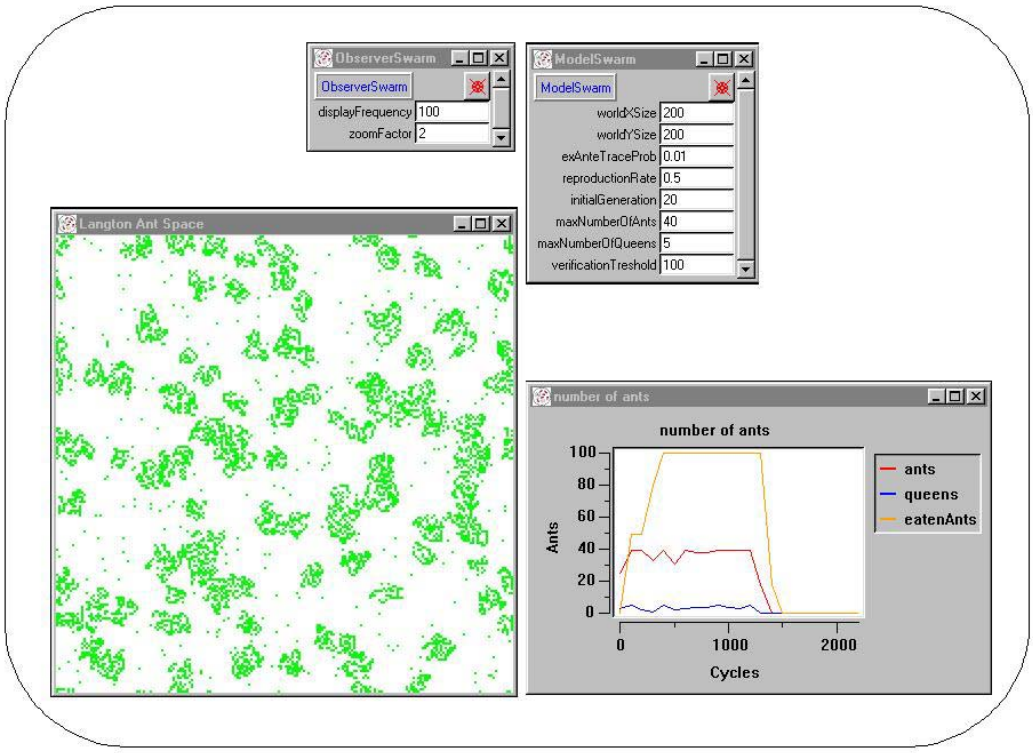
A proposito dei valori utilizzati per le formiche si noti che: l'azzeramento delle aliquote

d'imposta e della *bidRatio* inibiscono il meccanismo di ripartizione dei crediti, quindi il patrimonio delle regole non subisce variazioni. Il valore, zero, assegnato ad *evolutionRate* impedisce l'attivazione dell'algoritmo genetico, quindi le regole non vengono mai modificate. L'azzeramento degli altri valori è, in realtà, superfluo. Le regole sono codificate in modo che una sola di esse risulti compatibile con ciascuna condizione ambientale e, quindi, non occorre che queste effettuino offerte significative in asta, poiché una sola regola partecipa ogni volta. Essendo, poi, tutte le regole codificate in modo corretto, non serve applicare l'algoritmo genetico per evolvere la popolazione e, quindi, neppure serve variare il patrimonio di ciascuna regola per misurarne l'efficacia.

I valori per il formichiere sono simili a quelli usati per i vari esempi di utilizzazione del *workbench*; la maggior differenza è costituita dal valore, uno, di *effectorsFlag* che consente di ridurre il numero di regole ad otto, rendendo più agevole la verificazione delle medesime.

Gli agenti del modello, formiche o formichiere, codificano, tramite la propria *interface*, le informazioni ambientali secondo la seguente convenzione: la prima posizione non è utilizzata, la seconda vale zero se nella casella non vi sono tracce di feromone ed uno nel caso opposto, le ultime due sono usate, come numero binario, per rappresentare la direzione della formica. Il sistema di codificazione è, dunque, misto: due condizioni ed un valore. L'azione è rappresentata da un numero, da zero a tre, associato alla direzione che la formica deve assumere prima di muovere un passo. Il formichiere deve simulare il comportamento della formica, per stabilire in quale casella essa si troverà; le sue regole, una volta terminato il processo di apprendimento, dovranno risultare perfettamente identiche a quelle delle formiche.

Figura 3.11 – Estinzione delle formiche di Langton



Come riportato nella stampa della *probe-map* del *modelSwarm*, la simulazione è stata condotta con cinque regine, ognuna delle quali ha probabilità di riprodursi, in ogni ciclo, pari a 0.5. La probabilità che, fra i nuovi nati vi sia una nuova regina è pari, a sua volta a 0.5. Prima di avviare il modello sono stati effettuati venti cicli riproduttivi, onde disporre di una popolazione iniziale su cui far agire il formichiere. La riproduzione viene, inoltre, inibita dal sovrappopolamento, raggiunto quando nel modello siano presenti più di quaranta formiche; nell'ambito di queste non possono

coesistere più di cinque regine.

La mappa del territorio riporta la situazione ad estinzione avvenuta, dove sono ben visibili i percorsi effettuati dalle formiche prima di essere catturate; il movimento di queste ultime è stato reso meno regolare immettendo, prima di avviare la simulazione, tracce di feromone in posizioni casuali, in ragione dell'un percento degli spazi disponibili.

Il diagramma riporta: numero di formiche totali, numero di regine e numero di formiche catturate nell'intervallo di osservazione; il raggiungimento del livello di 100 formiche catturate ogni 100 tentativi, dimostra che il *classifier* è riuscito ad apprendere le regole di movimento delle formiche in circa 400 tentativi, cioè 400 cicli, poiché il formichiere effettua un tentativo per ogni ciclo.

Di seguito è riportata la stampa dei contenuti dei due *dataWarehouse* (si ricordi che tutte le formiche condividono lo stesso insieme di regole e, quindi, lo stesso *dataWarehouse*) e dei risultati conseguiti dal formichiere nei vari intervalli di osservazione. Il *dataWarehouse* delle formiche viene stampato solo all'inizio dell'elaborazione, poiché il suo contenuto non è destinato a cambiare; quello del formichiere è stampato ad inizio elaborazione e, nuovamente, ad apprendimento avvenuto.

```
Ant's rules are the following:

 49F7B60 - DataWarehouse cointains following data:
 49A6B70 - RuleList contains 8 rules:
 49F8730 strength: 10.000000 specificity 1.000000
0000
0001
My partnerList contains 0 entries for:
 49F8CE0 strength: 10.000000 specificity 1.000000
0001
0010
My partnerList contains 0 entries for:
 49F8F48 strength: 10.000000 specificity 1.000000
0010
0011
My partnerList contains 0 entries for:
 49F91C0 strength: 10.000000 specificity 1.000000
0011
0000
My partnerList contains 0 entries for:
 49F9438 strength: 10.000000 specificity 1.000000
0100
0011
My partnerList contains 0 entries for:
 49F96B0 strength: 10.000000 specificity 1.000000
0101
0000
My partnerList contains 0 entries for:
 49F9928 strength: 10.000000 specificity 1.000000
0110
0001
My partnerList contains 0 entries for:
 49F9BA0 strength: 10.000000 specificity 1.000000
0111
0010
My partnerList contains 0 entries for:

 49A6788 - EffectorList contains 4 effectors:
 49F9E18 suggestion 0
0000
My partnerList contains 0 entries for:
```

```
 49FA348 suggestion 1
0001
My partnerList contains 0 entries for:
 49FA4A8 suggestion 2
0010
My partnerList contains 0 entries for:
 49FA608 suggestion 3
0011
My partnerList contains 0 entries for:

 49A67A8 - MessageList contains 0 messages:

 49A67C8 - MatchList contains 0 rules:

 49A52E0 - WinnerList contains 0 rules:

 49A5300 - WorkList contains 0 rules:

 49A5320 - Treasury balance:
Current amount of treasury is 0.000000

current active effector is:

Statistics:
Rule's selection has been performed 0 times
CoverDetector has been applied 0 times
CoverEffector has been applied 0 times
Rule's genetic evolution has been performed 0 times
```

Anteater's rules before learning are the following:

```
 49FBF28 - DataWarehouse cointains following data:
 49FBEB8 - RuleList contains 8 rules:
 49FC7A0 strength: 10.000000 specificity 0.875000
001#
0000
My partnerList contains 0 entries for:
 49FCA18 strength: 10.000000 specificity 0.875000
0#11
0001
My partnerList contains 0 entries for:
 49FCC90 strength: 10.000000 specificity 1.000000
0010
0010
My partnerList contains 0 entries for:
 49FCF08 strength: 10.000000 specificity 0.750000
0#0#
0011
My partnerList contains 0 entries for:
 49FD180 strength: 10.000000 specificity 0.875000
10#0
0000
My partnerList contains 0 entries for:
 49FD3F8 strength: 10.000000 specificity 0.750000
01##
0001
My partnerList contains 0 entries for:
 49FD670 strength: 10.000000 specificity 0.875000
0#01
```

```
0010
My partnerList contains 0 entries for:
 49FD8E8 strength: 10.000000 specificity 0.875000
00#1
0011
My partnerList contains 0 entries for:

 49FBED8 - EffectorList contains 4 effectors:
 49FC230 suggestion 0
0000
My partnerList contains 0 entries for:
 49FC380 suggestion 1
0001
My partnerList contains 0 entries for:
 49FC4E0 suggestion 2
0010
My partnerList contains 0 entries for:
 49FC640 suggestion 3
0011
My partnerList contains 0 entries for:

 49FC190 - MessageList contains 0 messages:

 49FC1B0 - MatchList contains 0 rules:

 49FC1D0 - WinnerList contains 0 rules:

 49FC1F0 - WorkList contains 0 rules:

 49FC210 - Treasury balance:
Current amount of treasury is 0.000000

current active effector is:

Statistics:
Rule's selection has been performed 0 times
CoverDetector has been applied 0 times
CoverEffector has been applied 0 times
Rule's genetic evolution has been performed 0 times

49 of the latest 101 attempts were successfull
49 of the latest 100 attempts were successfull
80 of the latest 100 attempts were successfull
100 of the latest 100 attempts were successfull

Anteater's rules after learning are the following:

 49FBF28 - DataWarehouse cointains following data:
 49FBEB8 - RuleList contains 8 rules:
 4A21058 strength: 1.474236 specificity 1.000000
0110
0001
My partnerList contains 0 entries for:
 4A206A8 strength: 0.595666 specificity 1.000000
0000
0001
My partnerList contains 0 entries for:
 4A18778 strength: 1.501719 specificity 1.000000
0010
```

```
0011
My partnerList contains 0 entries for:
 4A1C6A0 strength: 1.166006 specificity 1.000000
0101
0000
My partnerList contains 0 entries for:
 4A1FD18 strength: 1.187388 specificity 1.000000
0111
0010
My partnerList contains 0 entries for:
 4A1E308 strength: 1.661862 specificity 1.000000
0001
0010
My partnerList contains 1 entries for:
 49FC210
 4A20E10 strength: 1.285472 specificity 1.000000
0100
0011
My partnerList contains 0 entries for:
 4A1F318 strength: 0.680483 specificity 1.000000
0011
0000
My partnerList contains 0 entries for:

 49FBED8 - EffectorList contains 4 effectors:
 49FC380 suggestion 1
0001
My partnerList contains 0 entries for:
 49FC230 suggestion 0
0000
My partnerList contains 0 entries for:
 49FC4E0 suggestion 2
0010
My partnerList contains 1 entries for:
 4A1E308
 49FC640 suggestion 3
0011
My partnerList contains 0 entries for:

 49FC190 - MessageList contains 0 messages:

 49FC1B0 - MatchList contains 1 rules:
 49FC4E0 suggestion 2
0010
My partnerList contains 1 entries for:
 4A1E308

 49FC1D0 - WinnerList contains 1 rules:
 4A1E308 strength: 1.661862 specificity 1.000000
0001
0010
My partnerList contains 1 entries for:
 49FC210

 49FC1F0 - WorkList contains 0 rules:

 49FC210 - Treasury balance:
Current amount of treasury is 105.645348
```

```
current active effector is:
 49FC4E0 suggestion 2
0010
My partnerList contains 1 entries for:
 4A1E308

Statistics:
Rule's selection has been performed 401 times
CoverDetector has been applied 49 times
CoverEffector has been applied 0 times
Rule's genetic evolution has been performed 26 times

100 of the latest 100 attempts were successfull
100 of the latest 100 attempts were successfull
100 of the latest 100 attempts were successfull
100 of the latest 100 attempts were successfull
100 of the latest 100 attempts were successfull
100 of the latest 100 attempts were successfull
100 of the latest 100 attempts were successfull
100 of the latest 100 attempts were successfull
100 of the latest 100 attempts were successfull
```

Ad apprendimento completato, le regole del formichiere sono identiche a quelle delle formiche, quindi il *classifier* è stato capace di "clonare" il programma che regola il movimento delle formiche. Il risultato è interessante poiché dimostra la possibilità di riprodurre il funzionamento di un sistema non conosciuto a priori; il formichiere, infatti, inizia ad operare con un patrimonio di regole generate casualmente.

Leggendo, poi, le regole contenute nel *dataWarehouse* è possibile effettuare congetture sui meccanismi che regolano tale funzionamento. In questo caso, la conoscenza preventiva delle regole delle formiche è resa necessaria dagli scopi del modello, volto ad illustrare il funzionamento del metodo e a permettere una immediata verificazione di quanto sopra. Un modello simile potrebbe simulare le reazioni di un monopolista che fronteggiasse un insieme di compratori, ognuno guidato da una specifica funzione di utilità, fissa o in evoluzione, per indagare i meccanismi di formazione dei prezzi in un mercato. Anche se la funzione di utilità dei consumatori non variasse, la diversa combinazione delle loro decisioni potrebbe comportare una variabilità accentuata della domanda e, in conseguenza, una ricerca non banale del livello ottimale di prezzi, da parte del monopolista. Lo studio delle variazioni di prezzo, a fronte dei vari livelli di domanda aggregata costituirebbe una via, alternativa alle due tradizionali nominate nel primo capitolo, per descrivere il funzionamento dei mercati monopolistici.

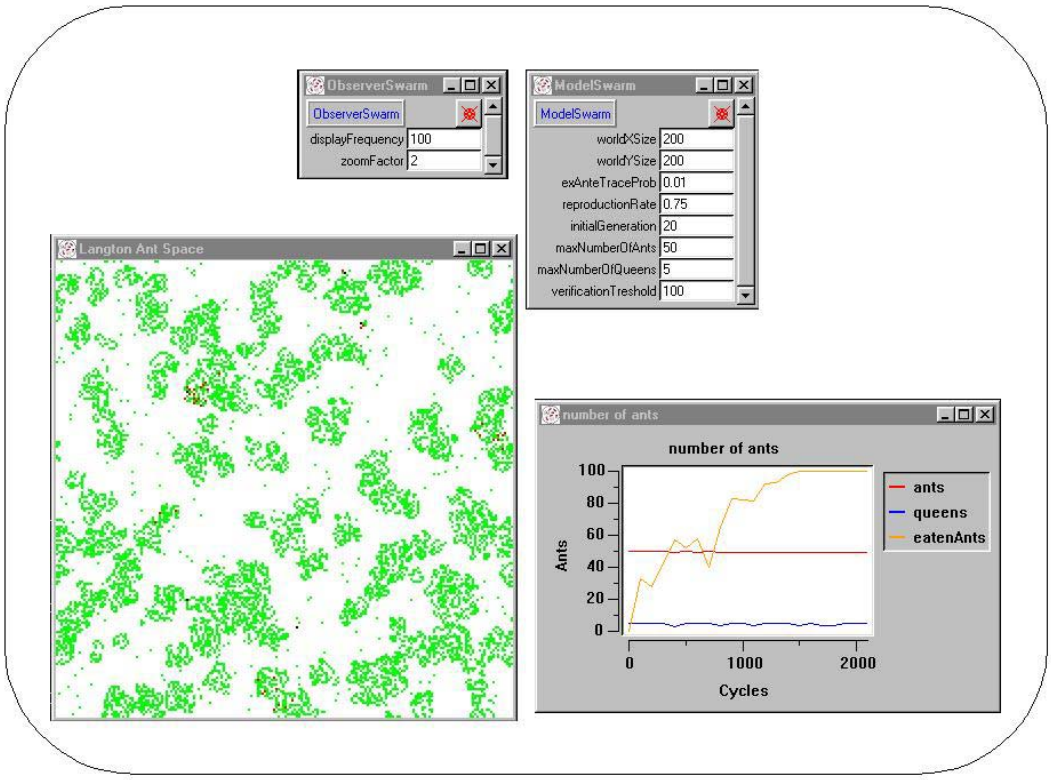
3.8.8 - Introduzione della variante di movimento delle regine

Per differenziare il comportamento delle formiche, si è stabilito che le regine muovano secondo regole diverse: mentre le operaie effettuano cambi di direzione di 90 gradi, le regine ruotano di 180 gradi, prima di effettuare il passo. Il risultato grafico è che le regine “oscillano” attorno alla posizione iniziale, e, poiché le nuove operaie generate vengono poste accanto alla regina, vengono a formarsi gruppi di formiche, simili a diverse colonie.

Il minor numero di regine, rispetto a quello di operaie, introduce una irregolarità nella percezione del formichiere: esso osserva, per la maggior parte delle volte, insetti che ruotano di 90 gradi e, solo poche volte, insetti che ruotano di 180 gradi; rilevando il colore, diverso, dei due tipi, il formichiere è in grado di associare il diverso comportamento agli insetti di colore diverso, ma la possibilità di studiare le regine è ridotta. Occorre, quindi, mantenere un basso livello di *lifeTax*, per evitare che le regole relative alle regine, poiché poco applicate, possano veder ridotto il loro patrimonio, pur essendo corrette: l'aliquota è stata ridotta all'uno percento.

Il *dataWarehouse* delle formiche è stato arricchito di otto nuove regole, per gestire il movimento delle regine, quindi, anche il *dataWarehouse* del formichiere è stato popolato con sedici regole. Le regine applicano le regole che presentano un "1" nella prima posizione, non utilizzata nell'esempio precedente. Il formichiere comunica al *ruleMaster*, sempre tramite il valore di quella posizione, l'informazione relativa al ruolo della formica oggetto di caccia. Il maggior numero di cicli di evoluzione, prevedibilmente necessario, ha consigliato di aumentare la fecondità delle formiche, onde impedire che l'estinzione della colonia potesse avvenire prima del completo apprendimento da parte del formichiere. E' stato anche elevato il numero massimo di formiche.

Figura 3.12 - Variante del "formichiere di Ferraris"



Il *classifier* è riuscito, anche se in un numero di cicli più elevato rispetto al caso precedente, ad ottenere il totale dei successi nel periodo di osservazione. La maggior fecondità delle formiche ha impedito l'estinzione della colonia e, quindi, sono ancora visibili, sulla mappa del territorio, i gruppi formatisi attorno alle varie regine.

Dall'osservazione del diagramma è percepibile una maggior irregolarità del cammino di apprendimento; in certi periodi il numero di successi si riduce, rispetto all'osservazione precedente, il che potrebbe derivare dall'estinzione di regole corrette, troppo indebolite dalla *lifeTax* perché poco applicate. Il sistema riesce, comunque, a raggiungere l'obiettivo prefissato e ciò conferma, ulteriormente, la robustezza del metodo.

I dati stampati durante l'elaborazione, riportati di seguito, confermano le osservazioni relative ai grafici

```
Ant's rules are the following:

        49F7B18 - DataWarehouse cointains following data:
        49A6B50 - RuleList contains 16 rules:
        49F8760 strength: 10.000000 specificity 1.000000
       0000
```

```
0001
My partnerList contains 0 entries for:
 49F8D10 strength: 10.000000 specificity 1.000000
0001
0010
My partnerList contains 0 entries for:
 49F8F78 strength: 10.000000 specificity 1.000000
0010
0011
My partnerList contains 0 entries for:
 49F91F0 strength: 10.000000 specificity 1.000000
0011
0000
My partnerList contains 0 entries for:
 49F9468 strength: 10.000000 specificity 1.000000
0100
0011
My partnerList contains 0 entries for:
 49F96E0 strength: 10.000000 specificity 1.000000
0101
0000
My partnerList contains 0 entries for:
 49F9958 strength: 10.000000 specificity 1.000000
0110
0001
My partnerList contains 0 entries for:
 49F9BD0 strength: 10.000000 specificity 1.000000
0111
0010
My partnerList contains 0 entries for:
 49F9E48 strength: 10.000000 specificity 1.000000
1000
0010
My partnerList contains 0 entries for:
 49FA0C0 strength: 10.000000 specificity 1.000000
1001
0011
My partnerList contains 0 entries for:
 49FA338 strength: 10.000000 specificity 1.000000
1010
0000
My partnerList contains 0 entries for:
 49FA5B0 strength: 10.000000 specificity 1.000000
1011
0001
My partnerList contains 0 entries for:
 49FA828 strength: 10.000000 specificity 1.000000
1100
0010
My partnerList contains 0 entries for:
 49FAAA0 strength: 10.000000 specificity 1.000000
1101
0011
My partnerList contains 0 entries for:
 49FAD18 strength: 10.000000 specificity 1.000000
1110
0000
My partnerList contains 0 entries for:
 49FAF90 strength: 10.000000 specificity 1.000000
```

```
1111
0001
My partnerList contains 0 entries for:

 49A6E00 - EffectorList contains 4 effectors:
 49FB208 suggestion 0
0000
My partnerList contains 0 entries for:
 49FB738 suggestion 1
0001
My partnerList contains 0 entries for:
 49FB898 suggestion 2
0010
My partnerList contains 0 entries for:
 49FB9F8 suggestion 3
0011
My partnerList contains 0 entries for:

 49A6E20 - MessageList contains 0 messages:

 49A6E40 - MatchList contains 0 rules:

 49F8020 - WinnerList contains 0 rules:

 49F8040 - WorkList contains 0 rules:

 49F8060 - Treasury balance:
Current amount of treasury is 0.000000

current active effector is:

Statistics:
Rule's selection has been performed 0 times
CoverDetector has been applied 0 times
CoverEffector has been applied 0 times
Rule's genetic evolution has been performed 0 times
```

Anteater's rules before learning are the following:

```
 49FE7E8 - DataWarehouse cointains following data:
 49FE730 - RuleList contains 16 rules:
 49FEFF0 strength: 10.000000 specificity 0.750000
#11#
0000
My partnerList contains 0 entries for:
 49FF268 strength: 10.000000 specificity 0.750000
#00#
0001
My partnerList contains 0 entries for:
 49FF4E0 strength: 10.000000 specificity 0.750000
0##1
0010
My partnerList contains 0 entries for:
 49FF758 strength: 10.000000 specificity 0.750000
0#0#
0011
My partnerList contains 0 entries for:
 49FF9D0 strength: 10.000000 specificity 0.875000
1#01
```

```
0000
My partnerList contains 0 entries for:
 49FFC48 strength: 10.000000 specificity 1.000000
1101
0001
My partnerList contains 0 entries for:
 49FFEC0 strength: 10.000000 specificity 0.875000
#111
0010
My partnerList contains 0 entries for:
 4A00138 strength: 10.000000 specificity 1.000000
0101
0011
My partnerList contains 0 entries for:
 4A003B0 strength: 10.000000 specificity 0.875000
#101
0000
My partnerList contains 0 entries for:
 4A00628 strength: 10.000000 specificity 0.750000
0##1
0001
My partnerList contains 0 entries for:
 4A008A0 strength: 10.000000 specificity 0.875000
#111
0010
My partnerList contains 0 entries for:
 4A00B18 strength: 10.000000 specificity 1.000000
0001
0011
My partnerList contains 0 entries for:
 4A00D90 strength: 10.000000 specificity 0.875000
1#11
0000
My partnerList contains 0 entries for:
 4A01008 strength: 10.000000 specificity 0.875000
011#
0001
My partnerList contains 0 entries for:
 4A01280 strength: 10.000000 specificity 0.875000
01#1
0010
My partnerList contains 0 entries for:
 4A014F8 strength: 10.000000 specificity 0.875000
00#1
0011
My partnerList contains 0 entries for:

 49FE750 - EffectorList contains 4 effectors:
 49FEA90 suggestion 0
0000
My partnerList contains 0 entries for:
 49FEBE0 suggestion 1
0001
My partnerList contains 0 entries for:
 49FED30 suggestion 2
0010
My partnerList contains 0 entries for:
 49FEE90 suggestion 3
0011
```

```
My partnerList contains 0 entries for:

 49FE770 - MessageList contains 0 messages:

 49FE790 - MatchList contains 0 rules:

 49FE7B0 - WinnerList contains 0 rules:

 49FEA50 - WorkList contains 0 rules:

 49FEA70 - Treasury balance:
Current amount of treasury is 0.000000

current active effector is:

Statistics:
Rule's selection has been performed 0 times
CoverDetector has been applied 0 times
CoverEffector has been applied 0 times
Rule's genetic evolution has been performed 0 times

33 of the latest 101 attempts were successfull
28 of the latest 100 attempts were successfull
43 of the latest 100 attempts were successfull
57 of the latest 100 attempts were successfull
52 of the latest 100 attempts were successfull
58 of the latest 100 attempts were successfull
40 of the latest 100 attempts were successfull
65 of the latest 100 attempts were successfull
83 of the latest 100 attempts were successfull
82 of the latest 100 attempts were successfull
81 of the latest 100 attempts were successfull
92 of the latest 100 attempts were successfull
93 of the latest 100 attempts were successfull
98 of the latest 100 attempts were successfull
100 of the latest 100 attempts were successfull

Anteater's rules after learning are the following:

 49FE7E8 - DataWarehouse cointains following data:
 49FE730 - RuleList contains 16 rules:
 4A2C7A0 strength: 1.515215 specificity 1.000000
0100
0011
My partnerList contains 0 entries for:
 4A37640 strength: 1.574913 specificity 1.000000
0011
0000
My partnerList contains 0 entries for:
 4A22790 strength: 1.292732 specificity 1.000000
0000
0001
My partnerList contains 0 entries for:
 4A5ADD0 strength: 0.379659 specificity 1.000000
1000
0010
My partnerList contains 0 entries for:
 4A4EDB8 strength: 0.917372 specificity 1.000000
1110
```

```
0000
My partnerList contains 0 entries for:
 4A5A040 strength: 0.592815 specificity 1.000000
1101
0011
My partnerList contains 0 entries for:
 4A59DF8 strength: 0.606123 specificity 1.000000
1100
0010
My partnerList contains 0 entries for:
 4A4CBD0 strength: 1.636444 specificity 1.000000
0101
0000
My partnerList contains 0 entries for:
 4A57430 strength: 0.168700 specificity 1.000000
1001
0011
My partnerList contains 0 entries for:
 4A360C8 strength: 1.575471 specificity 1.000000
0111
0010
My partnerList contains 1 entries for:
 49FEA70
 4A46298 strength: 1.728077 specificity 1.000000
0110
0001
My partnerList contains 0 entries for:
 4A500F8 strength: 1.529700 specificity 1.000000
1111
0001
My partnerList contains 0 entries for:
 4A5B018 strength: 0.593813 specificity 1.000000
1010
0000
My partnerList contains 0 entries for:
 4A35E80 strength: 1.805518 specificity 1.000000
0001
0010
My partnerList contains 0 entries for:
 4A58AB0 strength: 1.490572 specificity 1.000000
1011
0001
My partnerList contains 0 entries for:
 4A35590 strength: 1.867444 specificity 1.000000
0010
0011
My partnerList contains 0 entries for:

 49FE750 - EffectorList contains 4 effectors:
 49FEE90 suggestion 3
0011
My partnerList contains 0 entries for:
 49FEA90 suggestion 0
0000
My partnerList contains 0 entries for:
 49FEBE0 suggestion 1
0001
My partnerList contains 0 entries for:
 49FED30 suggestion 2
```

```
0010
My partnerList contains 1 entries for:
 4A360C8

 49FE770 - MessageList contains 0 messages:

 49FE790 - MatchList contains 1 rules:
 49FED30 suggestion 2
0010
My partnerList contains 1 entries for:
 4A360C8

 49FE7B0 - WinnerList contains 1 rules:
 4A360C8 strength: 1.575471 specificity 1.000000
0111
0010
My partnerList contains 1 entries for:
 49FEA70

 49FEA50 - WorkList contains 0 rules:

 49FEA70 - Treasury balance:
Current amount of treasury is 315.791085

current active effector is:
 49FED30 suggestion 2
0010
My partnerList contains 1 entries for:
 4A360C8

Statistics:
Rule's selection has been performed 1501 times
CoverDetector has been applied 122 times
CoverEffector has been applied 0 times
Rule's genetic evolution has been performed 71 times

100 of the latest 100 attempts were successfull
100 of the latest 100 attempts were successfull
100 of the latest 100 attempts were successfull
100 of the latest 100 attempts were successfull
100 of the latest 100 attempts were successfull
100 of the latest 100 attempts were successfull
```

3.9 - Un esempio di utilizzazione di GM

Viene qui illustrato un semplice modello volto alla ricerca del massimo di una funzione data. Si vuole, inoltre, fornire una comparazione fra le due possibilità di trattazione della *fitness* codificate in GM: quella tradizionale, che considera il valore totale, e l'altra, introdotta con GM, basata sulle differenze fra la *fitness* dell'individuo peggiore e quella di ciascun altro. La funzione scelta per lo studio è utilizzata da Freeman (1994) nella trattazione delle metodologie evolutive con l'impiego del prodotto *Mathematica*; essa presenta peculiarità che rendono la ricerca del massimo assoluto particolarmente difficoltosa per un algoritmo genetico. Si vuole dimostrare la possibilità di ottenere buoni risultati nell'applicazione del metodo basato sulle differenze di *fitness*: esso permetterebbe una maggior precisione di azione quando, con il miglioramento della popolazione, le differenze fra le *fitness* dei vari individui tendessero a risultare minime, ponendole in grado di influenzare con maggior efficacia l'evoluzione ulteriore. Beneficio accessorio, ma interessante, legato a questo metodo sarebbe la possibilità di assegnare valori negativi di *fitness*, comportamento non ammesso utilizzando

algoritmi genetici tradizionali.

L'elaborazione è stata condotta, per le prime cinquanta volte, utilizzando il metodo delle differenze e, successivamente, ripetuta con metodo tradizionale. Per ciascuna elaborazione sono stati collezionati i dati relativi ai progressi e risultati conseguiti. In base a questi dati viene svolta una comparazione fra i due metodi: essa permette di rilevare un comportamento nettamente più efficiente, e maggiormente efficace, da parte dell'algoritmo basato sulla valutazione delle differenze di *fitness.* Con il metodo innovativo si sono ottenuti risultati più precisi, in un numero minore di cicli evolutivi.

3.9.1 - La funzione usata da Freeman

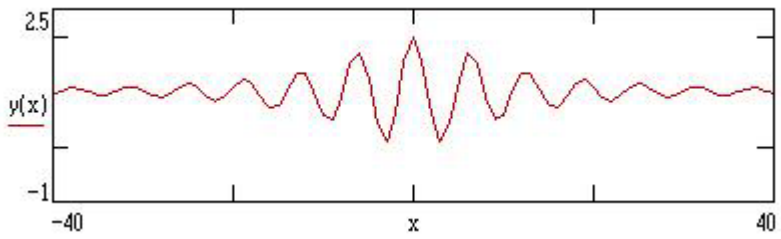
Per verificare la validità di una ricerca in tutto lo spazio, viene impiegata, in Freeman, una particolare funzione, definita sull'intero insieme dei numeri reali: i valori del codominio presentano un andamento periodico, dove massimi e minimi relativi si alternano in continuazione, contribuendo a delineare una espressione grafica in cui cuspidi, sempre più accentuate e ravvicinate, intornano l'unico punto di massimo assoluto, sito in corrispondenza dell'origine degli assi. La forma analitica della funzione è data da:

$$y = 1 + \cos(x) / (1 + 0.01 * x \wedge 2)$$

dove x è espresso in radianti.

Lo studio è stato condotto, come in Freeman, sull'intervallo]-40,+40[del dominio, per il quale si fornisce, in figura 3.13, la rappresentazione grafica dell'andamento dei valori nel codominio:

Figura 3.13 - Grafico della funzione di Freeman nell'intervallo]-40,+40[



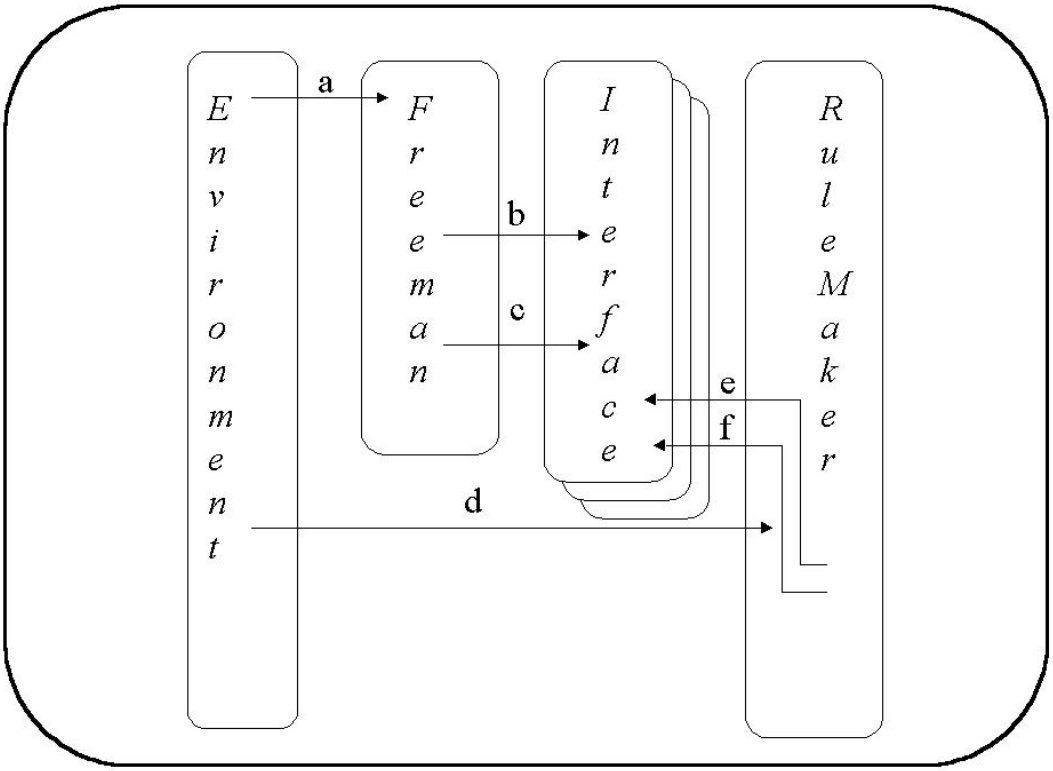
La presenza di un gran numero di massimi locali comporta un'elevata possibilità che l'algoritmo genetico tenda ad arrestarsi su uno di essi. Utilizzando individui che codifichino, in forma binaria, un valore, poi trasformato in un numero reale, appartenente all'intervallo studiato, la presenza di un massimo locale può contribuire a determinare una differenza di *fitness*, a favore dell'individuo cui corrisponde quel valore di ascissa, sufficiente a consentire il prevalere di quel tipo durante l'evoluzione. I picchi risultano più ravvicinati ed elevati nell'intorno del massimo assoluto; ciò contribuisce a rendere minima la differenza di *fitness*, in termini relativi, a favore di un eventuale individuo capace di esprimere l'ascissa zero; analogamente piccolo risulterebbe il vantaggio nella selezione per la riproduzione di quest'ultimo. L'algoritmo genetico può tendere, perciò, a convergere rapidamente su un tipo, buono, ma non ottimo. In alternativa, possono manifestarsi grosse difficoltà di convergenza della popolazione su livelli di omogeneità soddisfacenti: quando le frequenze dei tipi venissero a ripartirsi, in modo quasi uniforme, tra pochi schemi portatori di valori simili di *fitness,* difficilmente uno di quelli riuscirebbe a prevalere. I vari picchi costituirebbero, in conclusione, una sorta di "trappole" per l'algoritmo genetico.

Con il metodo delle differenze di *fitness*, queste ultime vengono amplificate, in termini relativi, in modo da consentire una selezione più efficace dei vari individui.

3.9.2 - Il modello

Si tratta di una struttura costruita a partire dal pacchetto GM; l'interazione avviene direttamente con le *interface*, senza impiego di oggetti della classe *Agent*. L'unica addizione consta della classe *Freeman*, capace di calcolare il valore della funzione, data l'ascissa rappresentata dal genoma contenuto in ciascuna *interface*. Il valore di ascissa è ottenuto trasformando linearmente il numero naturale espresso da ciascun genoma, considerato come numero binario, in un valore reale.

Figura 3.14 - Schema del modello



Come in figura 3.14, l'azione parte dall'ambiente, cioè dall'oggetto *modelSwarm*, che ordina (a) all'oggetto *freeman* di effettuare un passo. *Freeman* provvede a richiedere (b) a ciascuna *interface* il valore di ascissa di cui è portatrice ed effettua il calcolo del valore della funzione; tale ammontare viene, poi, comunicato (c) all'*interface* come valore di *fitness*.

Terminata la valutazione delle singole *interface*, da parte di *freeman*, il *modelSwarm* ordina (d) al *ruleMaker* di effettuare un'evoluzione; questi procede, come descritto nella illustrazione del funzionamento di GM, richiedendo (e) a ciascuna *interface*: *fitness* e genoma. Dopo aver effettuato l'evoluzione i dati modificati vengono comunicati (f) alle diverse *interface*. L'azione può essere ripetuta fino al conseguimento del grado di omogeneità desiderato.

L'oggetto *freeman* assolve, inoltre, a compiti accessori quali: determinare, ad ogni passo, quale valore di ascissa risulti più frequentemente rappresentato nella popolazione di *interface*, calcolando l'ammontare di detta frequenza, e stampare, su *standard output*, i dati relativi all'andamento dell'evoluzione.

Utilizzando i dati resi disponibili da *freeman*, l'*observerSwarm* produce un grafico dove sono riportati sia il valore di ascissa più frequente, sia il numero di individui portatori del medesimo. L'osservazione del grafico permette di cogliere la convergenza della popolazione verso il grado di omogeneità voluto, leggendo, quindi, in corrispondenza di esso il valore di ascissa, indicato dall'algoritmo genetico, capace di massimizzare il valore della funzione.

Freeman provvede a stampare, ogni cento passi, partendo da quello in cui, per la prima volta, l'omogeneità della popolazione risulti pari al valore desiderato, nell'ordine: numero di passi compiuti dall'inizio dell'elaborazione, valore reale di ascissa più frequentemente rappresentato nella popolazione, frequenza con cui detto valore si presenta, valore della funzione per l'ascissa suddetta, valori minimo, medio e massimo di *fitness* della popolazione. I dati possono essere preceduti da simboli che ne indicano il significato o prodotti in formato direttamente leggibile da *Excel.*

Per gestire il comportamento di *freeman* sono stati aggiunti, agli usuali parametri di GM, due valori ulteriori: *confidence* ed *excelFormat*.

Confidence: stabilisce il grado di omogeneità prescritto; il suo valore deve essere fornito in termini unitari. Al raggiungimento di detto grado *freeman* stampa i dati statistici; gli stessi dati saranno stampati dopo ogni 100, ulteriori, cicli di elaborazione, per verificare la stabilità del risultato ottenuto. Se una popolazione converge nella misura voluta dopo 35 cicli, le successive stampe avverranno al ciclo 135, 235 eccetera.

ExcelFormat: utilizzato per selezionare il tipo di stampa; la discriminazione avviene in base all'eguaglianza a zero del valore fornito. Con valore nullo viene stampata anche la descrizione dei dati; con valore diverso da zero vengono stampati i singoli dati, separati da un "!"; ciò consente di leggere direttamente da *excel* il *file*, caricandolo in un foglio di lavoro. In quest'ultimo caso l'*output* andrà indirizzato su *file* utilizzando il comando: "*freeman > filename*". Dovendo collezionare i dati di più prove risulta comodo utilizzare il comando "*freeman >> filename*" che permette di accodare i diversi dati in un unico *file*.

La *probe map* del *modelSwarm* consente, naturalmente, di immettere le scelte relative a: frequenza di evoluzione, tasso di ricambio generazionale, probabilità di *crossover*, probabilità di mutazione, numero di individui e numero di *interface* da utilizzare, lunghezza dei genomi e metodo di trattazione della *fitness*.

3.9.3 - Metrica dell'algoritmo, metrica del modello e densità del dominio

I genomi, contenuti nelle *interface*, rappresentano numeri naturali, codificati in forma binaria. Ad ogni numero naturale viene fatto corrispondere uno dei punti appartenenti al sottoinsieme del dominio dato dall'intervallo]-40,+40[: l'analisi della funzione avviene, perciò, in base ad un numero finito di punti del dominio; tanto più lungo è il genoma, tanto più denso risulta l'insieme di punti utilizzato.

La classe *Interface* contiene le istruzioni necessarie a trasformare il valore naturale, espresso da ciascun genoma, in un valore reale. La funzione di trasformazione impiegata consente di utilizzare genomi di lunghezza variabile, onde permettere di effettuare esperimenti caratterizzati da differente densità dell'insieme di punti considerato. La forma analitica della funzione è data da:

$$x = -40 + n * 1/((2^{\text{lunghezza del genoma}} - 2) / 80)$$

dove $x \in R$ indica il valore reale di ascissa, ottenuto a partire da $n \in N$ che corrisponde al valore, decimale, del numero binario espresso dal genoma.

Si tratta di una semplice trasformazione lineare del valore decimale, ottenuto dalla conversione dell'espressione binaria di cui è portatore ciascun genoma, caratterizzata da un coefficiente angolare pari all'inverso del rapporto tra il massimo valore pari, ottenibile dal genoma portatore di tutti "1" salvo uno "0" in ultima posizione, e l'ampiezza dell'intervallo considerato; l'intercetta con l'asse delle ascisse è uguale all'estremo inferiore di tale intervallo. L'accorgimento di calcolare il coefficiente angolare della trasformazione in base al massimo valore pari ottenibile dal genoma, data la sua lunghezza, invece del massimo valore in assoluto, forzatamente dispari, consente di garantire che uno dei numeri naturali, espressi dai vari genomi, corrisponda, dopo la trasformazione, al valore reale zero. Utilizzando il massimo valore in assoluto, tale corrispondenza verrebbe meno in conseguenza della limitata precisione, quindici cifre significative per il tipo *double*,

della rappresentazione dei numeri in *Objective-C*.

Nel caso in osservazione, dove la lunghezza del genoma è pari a dieci, la funzione di trasformazione è esprimibile come:

x = -40 + n * 1/12.775

in base alla quale, il numero naturale 511 viene trasformato nel valore reale zero.

3.9.4 - Prove e risultati

GM è in grado di gestire algoritmi genetici basati sia sulla trattazione tradizionale della *fitness*, sia sull'utilizzazione delle differenze di *fitness*. Agendo sul valore del parametro *useDeltaFitness* è possibile stabilire quale comportamento debba essere tenuto durante l'elaborazione. Grazie a questa possibilità è risultato agevole condurre una serie di cento elaborazioni, utilizzando, per le prime cinquanta, la tecnica più innovativa e per le restanti quella tradizionale. Prima di ogni elaborazione è stato rinnovato il seme di generazione per i numeri casuali: in questo modo le popolazioni iniziali e le scelte, pseudo-casuali, effettuate durante l'evoluzione, sono state diverse per ciascuna elaborazione. La popolazione utilizzata constava di cento *interface*, ciascuna delle quali ospitava un genoma lungo dieci posizioni.

Utilizzando i servizi statistici dell'oggetto *freeman*, sono stati collezionati i dati relativi ad ogni elaborazione, riassunti, poi, nelle due tabelle seguenti:

Tabella 3.1 - Elaborazioni effettuate col metodo del valore differenziale di *fitness*: parametro *useDeltaFitness* = 1.

Step	**x**	**%**	**y**	***average* y**	**abs(0-x)**
32	0,00000000	100	2,00000000	2,00000000	
132	0,00000000	100	2,00000000	2,00000000	
232	0,00000000	99	2,00000000	1,98243400	
332	0,00000000	100	2,00000000	2,00000000	
432	0,00000000	98	2,00000000	1,99061700	0,00000000
98	0,00000000	100	2,00000000	2,00000000	
198	0,00000000	100	2,00000000	2,00000000	
298	0,00000000	100	2,00000000	2,00000000	
398	0,00000000	100	2,00000000	2,00000000	
498	0,00000000	100	2,00000000	2,00000000	0,00000000
18	0,07827789	100	1,99687700	1,99687600	
118	0,07827789	100	1,99687700	1,99687600	
218	0,07827789	100	1,99687700	1,99687600	
318	0,07827789	99	1,99687700	1,98651500	
418	0,07827789	100	1,99687700	1,99687600	0,07827789
38	0,00000000	100	2,00000000	2,00000000	
138	0,00000000	100	2,00000000	2,00000000	
238	0,00000000	100	2,00000000	2,00000000	
338	0,00000000	100	2,00000000	2,00000000	

Step	x	%	y	*average* y	abs(0-x)
438	0,00000000	100	2,00000000	2,00000000	0,00000000
37	0,07827789	100	1,99687700	1,99687600	
137	0,07827789	100	1,99687700	1,99687600	
237	0,07827789	100	1,99687700	1,99687600	
337	0,07827789	100	1,99687700	1,99687600	
437	0,07827789	100	1,99687700	1,99687600	0,07827789
100	-6,34050900	66	1,71208500	1,69790900	
200	-6,26223100	100	1,71815300	1,71815400	
300	-6,26223100	100	1,71815300	1,71815400	
400	-6,26223100	100	1,71815300	1,71815400	
500	-6,26223100	100	1,71815300	1,71815400	6,26223100
51	0,07827789	100	1,99687700	1,99687600	
151	0,07827789	100	1,99687700	1,99687600	
251	0,07827789	100	1,99687700	1,99687600	
351	0,07827789	100	1,99687700	1,99687600	
451	0,07827789	100	1,99687700	1,99687600	0,07827789
32	0,00000000	100	2,00000000	2,00000000	
132	0,00000000	100	2,00000000	2,00000000	
232	0,00000000	99	2,00000000	1,99234200	
332	0,00000000	99	2,00000000	1,99950500	
432	0,00000000	99	2,00000000	1,99807100	0,00000000
45	0,07827789	100	1,99687700	1,99687600	
145	0,07827789	100	1,99687700	1,99687600	
245	0,07827789	100	1,99687700	1,99687600	
345	0,07827789	100	1,99687700	1,99687600	
445	0,07827789	100	1,99687700	1,99687600	0,07827789
36	0,00000000	100	2,00000000	2,00000000	
136	0,00000000	100	2,00000000	2,00000000	
236	0,00000000	100	2,00000000	2,00000000	
336	0,00000000	100	2,00000000	2,00000000	
436	0,00000000	100	2,00000000	2,00000000	0,00000000
53	0,07827789	100	1,99687700	1,99687600	
153	0,07827789	100	1,99687700	1,99687600	
253	0,07827789	100	1,99687700	1,99687600	
353	0,07827789	100	1,99687700	1,99687600	
453	0,07827789	100	1,99687700	1,99687600	0,07827789
30	0,07827789	100	1,99687700	1,99687600	
130	0,07827789	100	1,99687700	1,99687600	
230	0,07827789	100	1,99687700	1,99687600	
330	0,07827789	100	1,99687700	1,99687600	
430	0,07827789	98	1,99687700	1,96859500	0,07827789

Step	x	%	y	*average* y	abs(0-x)
40	0,07827789	100	1,99687700	1,99687600	
140	0,07827789	99	1,99687700	1,99662700	
240	0,07827789	100	1,99687700	1,99687600	
340	0,07827789	99	1,99687700	1,99662700	
440	0,07827789	99	1,99687700	1,99448900	0,07827789
33	0,00000000	100	2,00000000	2,00000000	
133	0,00000000	100	2,00000000	2,00000000	
233	0,00000000	100	2,00000000	2,00000000	
333	0,00000000	100	2,00000000	2,00000000	
433	0,00000000	100	2,00000000	2,00000000	0,00000000
27	0,00000000	100	2,00000000	2,00000000	
127	0,00000000	100	2,00000000	2,00000000	
227	0,00000000	100	2,00000000	2,00000000	
327	0,00000000	100	2,00000000	2,00000000	
427	0,00000000	100	2,00000000	2,00000000	0,00000000
47	0,07827789	100	1,99687700	1,99687600	
147	0,07827789	100	1,99687700	1,99687600	
247	0,07827789	100	1,99687700	1,99687600	
347	0,07827789	100	1,99687700	1,99687600	
447	0,07827789	98	1,99687700	1,98044400	0,07827789
44	0,07827789	100	1,99687700	1,99687600	
144	0,07827789	100	1,99687700	1,99687600	
244	0,07827789	100	1,99687700	1,99687600	
344	0,07827789	100	1,99687700	1,99687600	
444	0,07827789	100	1,99687700	1,99687600	0,07827789
34	0,00000000	100	2,00000000	2,00000000	
134	0,00000000	100	2,00000000	2,00000000	
234	0,00000000	100	2,00000000	2,00000000	
334	0,00000000	100	2,00000000	2,00000000	
434	0,00000000	100	2,00000000	2,00000000	0,00000000
38	0,00000000	100	2,00000000	2,00000000	
138	0,00000000	100	2,00000000	2,00000000	
238	0,00000000	100	2,00000000	2,00000000	
338	0,00000000	100	2,00000000	2,00000000	
438	0,00000000	100	2,00000000	2,00000000	0,00000000
31	0,07827789	100	1,99687700	1,99687600	
131	0,07827789	100	1,99687700	1,99687600	
231	0,07827789	100	1,99687700	1,99687600	
331	0,07827789	100	1,99687700	1,99687600	
431	0,07827789	100	1,99687700	1,99687600	0,07827789
34	0,00000000	100	2,00000000	2,00000000	
134	0,00000000	100	2,00000000	2,00000000	

Step	x	%	y	*average* y	abs(0-x)
234	0,00000000	100	2,00000000	2,00000000	
334	0,00000000	100	2,00000000	2,00000000	
434	0,00000000	100	2,00000000	2,00000000	0,00000000
47	0,00000000	100	2,00000000	2,00000000	
147	0,00000000	99	2,00000000	1,99807100	
247	0,00000000	100	2,00000000	2,00000000	
347	0,00000000	99	2,00000000	1,99074200	
447	0,00000000	100	2,00000000	2,00000000	0,00000000
21	0,07827789	100	1,99687700	1,99687600	
121	0,07827789	99	1,99687700	1,99662700	
221	0,07827789	99	1,99687700	1,99613700	
321	0,07827789	100	1,99687700	1,99687600	
421	0,07827789	100	1,99687700	1,99687600	0,07827789
35	0,07827789	100	1,99687700	1,99687600	
135	0,07827789	100	1,99687700	1,99687600	
235	0,07827789	100	1,99687700	1,99687600	
335	0,07827789	100	1,99687700	1,99687600	
435	0,07827789	100	1,99687700	1,99687600	0,07827789
44	0,07827789	100	1,99687700	1,99687600	
144	0,07827789	100	1,99687700	1,99687600	
244	0,07827789	98	1,99687700	1,98511100	
344	0,07827789	100	1,99687700	1,99687600	
444	0,07827789	100	1,99687700	1,99687600	0,07827789
37	0,07827789	100	1,99687700	1,99687600	
137	0,07827789	100	1,99687700	1,99687600	
237	0,07827789	100	1,99687700	1,99687600	
337	0,07827789	100	1,99687700	1,99687600	
437	0,07827789	100	1,99687700	1,99687600	0,07827789
25	-0,15655580	100	1,98752800	1,98753000	
125	0,00000000	100	2,00000000	2,00000000	
225	0,00000000	100	2,00000000	2,00000000	
325	0,00000000	100	2,00000000	2,00000000	
425	0,00000000	99	2,00000000	1,99987500	0,00000000
68	0,00000000	100	2,00000000	2,00000000	
168	0,00000000	100	2,00000000	2,00000000	
268	0,00000000	100	2,00000000	2,00000000	
368	0,00000000	100	2,00000000	2,00000000	
468	0,00000000	100	2,00000000	2,00000000	0,00000000
31	0,07827789	100	1,99687700	1,99687600	
131	0,07827789	100	1,99687700	1,99687600	
231	0,07827789	100	1,99687700	1,99687600	
331	0,07827789	100	1,99687700	1,99687600	

Step	x	%	y	*average* y	abs(0-x)
431	0,07827789	100	1,99687700	1,99687600	0,07827789
46	0,00000000	100	2,00000000	2,00000000	
146	0,00000000	100	2,00000000	2,00000000	
246	0,00000000	100	2,00000000	2,00000000	
346	0,00000000	99	2,00000000	1,99308200	
446	0,00000000	99	2,00000000	1,98586700	0,00000000
32	0,00000000	100	2,00000000	2,00000000	
132	0,00000000	100	2,00000000	2,00000000	
232	0,00000000	100	2,00000000	2,00000000	
332	0,00000000	99	2,00000000	1,99074200	
432	0,00000000	100	2,00000000	2,00000000	0,00000000
33	0,00000000	100	2,00000000	2,00000000	
133	0,00000000	100	2,00000000	2,00000000	
233	0,00000000	100	2,00000000	2,00000000	
333	0,00000000	100	2,00000000	2,00000000	
433	0,00000000	100	2,00000000	2,00000000	0,00000000
32	0,00000000	100	2,00000000	2,00000000	
132	0,00000000	100	2,00000000	2,00000000	
232	0,00000000	100	2,00000000	2,00000000	
332	0,00000000	100	2,00000000	2,00000000	
432	0,00000000	100	2,00000000	2,00000000	0,00000000
34	0,07827789	100	1,99687700	1,99687600	
134	0,07827789	100	1,99687700	1,99687600	
234	0,07827789	100	1,99687700	1,99687600	
334	0,07827789	100	1,99687700	1,99687600	
434	0,07827789	100	1,99687700	1,99687600	0,07827789
44	0,00000000	100	2,00000000	2,00000000	
144	0,00000000	100	2,00000000	2,00000000	
244	0,00000000	100	2,00000000	2,00000000	
344	0,00000000	100	2,00000000	2,00000000	
444	0,00000000	100	2,00000000	2,00000000	0,00000000
41	-6,26223100	100	1,71815300	1,71815400	
141	-6,26223100	100	1,71815300	1,71815400	
241	-6,26223100	100	1,71815300	1,71815400	
341	-6,26223100	100	1,71815300	1,71815400	
441	-6,26223100	100	1,71815300	1,71815400	6,26223100
58	0,00000000	100	2,00000000	2,00000000	
158	0,00000000	100	2,00000000	2,00000000	
258	0,00000000	99	2,00000000	1,99996900	
358	0,00000000	100	2,00000000	2,00000000	
458	0,00000000	100	2,00000000	2,00000000	0,00000000

Step	x	%	y	*average* y	abs(0-x)
26	0,07827789	100	1,99687700	1,99687600	
126	0,07827789	100	1,99687700	1,99687600	
226	0,07827789	100	1,99687700	1,99687600	
326	0,07827789	100	1,99687700	1,99687600	
426	0,07827789	100	1,99687700	1,99687600	0,07827789
34	0,07827789	100	1,99687700	1,99687600	
134	0,07827789	100	1,99687700	1,99687600	
234	0,07827789	98	1,99687700	1,98577600	
334	0,07827789	99	1,99687700	1,98303500	
434	0,07827789	100	1,99687700	1,99687600	0,07827789
26	0,00000000	100	2,00000000	2,00000000	
126	0,00000000	100	2,00000000	2,00000000	
226	0,00000000	100	2,00000000	2,00000000	
326	0,00000000	100	2,00000000	2,00000000	
426	0,00000000	98	2,00000000	1,98393800	0,00000000
59	0,07827789	100	1,99687700	1,99687600	
159	0,07827789	99	1,99687700	1,99448900	
259	0,07827789	97	1,99687700	1,98700100	
359	0,07827789	100	1,99687700	1,99687600	
459	0,07827789	100	1,99687700	1,99687600	0,07827789
43	0,00000000	100	2,00000000	2,00000000	
143	0,00000000	100	2,00000000	2,00000000	
243	0,00000000	100	2,00000000	2,00000000	
343	0,00000000	99	2,00000000	1,99308200	
443	0,00000000	100	2,00000000	2,00000000	0,00000000
31	0,00000000	100	2,00000000	2,00000000	
131	0,00000000	100	2,00000000	2,00000000	
231	0,00000000	100	2,00000000	2,00000000	
331	0,00000000	100	2,00000000	2,00000000	
431	0,00000000	100	2,00000000	2,00000000	0,00000000
35	0,07827789	100	1,99687700	1,99687600	
135	0,07827789	100	1,99687700	1,99687600	
235	0,07827789	100	1,99687700	1,99687600	
335	0,07827789	100	1,99687700	1,99687600	
435	0,07827789	100	1,99687700	1,99687600	0,07827789
35	0,07827789	100	1,99687700	1,99687600	
135	0,07827789	99	1,99687700	1,98924300	
235	0,07827789	100	1,99687700	1,99687600	
335	0,07827789	100	1,99687700	1,99687600	
435	0,07827789	99	1,99687700	1,98924300	0,07827789
31	0,07827789	100	1,99687700	1,99687600	
131	0,07827789	100	1,99687700	1,99687600	

Step	**x**	**%**	**y**	***average* y**	**abs(0-x)**
231	0,07827789	100	1,99687700	1,99687600	
331	0,07827789	100	1,99687700	1,99687600	
431	0,07827789	99	1,99687700	1,99662700	0,07827789
62	0,00000000	100	2,00000000	2,00000000	
162	0,00000000	100	2,00000000	2,00000000	
262	0,00000000	100	2,00000000	2,00000000	
362	0,00000000	100	2,00000000	2,00000000	
462	0,00000000	100	2,00000000	2,00000000	0,00000000
47	0,07827789	100	1,99687700	1,99687600	
147	0,07827789	99	1,99687700	1,99662700	
247	0,07827789	100	1,99687700	1,99687600	
347	0,07827789	100	1,99687700	1,99687600	
447	0,07827789	100	1,99687700	1,99687600	0,07827789
41	0,00000000	100	2,00000000	2,00000000	
141	0,00000000	100	2,00000000	2,00000000	
241	0,00000000	100	2,00000000	2,00000000	
341	0,00000000	100	2,00000000	2,00000000	
441	0,00000000	100	2,00000000	2,00000000	0,00000000
35	0,00000000	100	2,00000000	2,00000000	
135	0,00000000	100	2,00000000	2,00000000	
235	0,00000000	100	2,00000000	2,00000000	
335	0,00000000	100	2,00000000	2,00000000	
435	0,00000000	100	2,00000000	2,00000000	0,00000000
Sum					14,32485347
Ave	-0,21542075	100	1,98721538	1,98625689	0,28649707
Var	1,53521899	5	0,00304518	0,00306856	1,51975462

Tabella 3.2 - Elaborazioni effettuate con metodo tradizionale: parametro *useDeltaFitness* = 0

Step	**x**	**%**	**y**	***average* y**	**abs(0-x)**
100	-0,39138940	50	1,92296600	1,97040000	
200	-0,07827789	97	1,99687700	1,99697000	
300	-0,07827789	97	1,99687700	1,97826100	
400	-0,07827789	100	1,99687700	1,99687600	
500	-0,39138940	99	1,92296600	1,99613700	0,39138940
100	0,23483370	49	1,97201700	1,98065600	
200	0,07827789	91	1,99687700	1,99588000	
300	0,07827789	72	1,99687700	1,98843000	
400	0,07827789	66	1,99687700	1,99369800	
500	0,07827789	94	1,99687700	1,99631500	0,07827789

Step	x	%	y	*average* y	abs(0-x)
100	6,26223100	41	1,71815300	1,69058400	
200	6,18395300	51	1,71981400	1,71690800	
300	6,10567500	99	1,71699600	1,71291200	
400	6,10567500	85	1,71699600	1,71716900	
500	6,10567500	89	1,71699600	1,71047600	6,10567500
100	0,07827789	64	1,99687700	1,99089400	
200	0,15655580	80	1,98752800	1,96602000	
300	0,15655580	73	1,98752800	1,98766600	
400	0,15655580	90	1,98752800	1,98846400	
500	0,07827789	87	1,99687700	1,98035500	0,07827789
100	-0,23483370	79	1,97201700	1,99423500	
200	-0,07827789	97	1,99687700	1,99185300	
300	0,00000000	57	2,00000000	1,99580100	
400	0,00000000	66	2,00000000	1,97493900	
500	-0,31311150	76	1,95044800	1,98810800	0,31311150
100	0,31311150	34	1,95044800	1,96520000	
200	0,23483370	42	1,97201700	1,97050300	
300	0,07827789	58	1,99687700	1,98245700	
400	0,23483370	50	1,97201700	1,95872100	
500	0,07827789	87	1,99687700	1,99364400	0,07827789
100	0,00000000	26	2,00000000	1,96726000	
200	-0,07827789	32	1,99687700	1,97663100	
300	-0,07827789	33	1,99687700	1,95790700	
400	-0,39138940	45	1,92296600	1,95820600	
500	-0,23483370	46	1,97201700	1,96939200	0,23483370
100	0,31311150	35	1,95044800	1,82846800	
200	0,07827789	33	1,99687700	1,97810100	
300	0,23483370	80	1,97201700	1,99330000	
400	0,07827789	72	1,99687700	1,97176600	
500	0,07827789	75	1,99687700	1,97671400	0,07827789
100	-0,31311150	40	1,95044800	1,95987800	
200	-0,07827789	60	1,99687700	1,99520200	
300	0,00000000	58	2,00000000	1,99794300	
400	0,00000000	100	2,00000000	2,00000000	
500	0,00000000	98	2,00000000	1,99993700	0,00000000
100	0,07827789	70	1,99687700	1,99236500	
200	0,23483370	60	1,97201700	1,98693200	
300	0,23483370	96	1,97201700	1,99539100	
400	0,07827789	100	1,99687700	1,99687600	
500	0,15655580	72	1,98752800	1,99410400	0,15655580
100	0,39138940	59	1,92296600	1,90751300	
200	0,39138940	60	1,92296600	1,95253200	

Step	x	%	y	*average* y	abs(0-x)
300	0,07827789	100	1,99687700	1,99687600	
400	0,07827789	100	1,99687700	1,99687600	
500	0,07827789	100	1,99687700	1,99687600	0,07827789
100	0,15655580	81	1,98752800	1,98048400	
200	0,15655580	98	1,98752800	1,97886300	
300	0,15655580	99	1,98752800	1,97809300	
400	0,15655580	98	1,98752800	1,98771600	
500	0,15655580	100	1,98752800	1,98753000	0,15655580
100	0,00000000	66	2,00000000	1,98310600	
200	0,00000000	100	2,00000000	2,00000000	
300	0,00000000	83	2,00000000	1,99788000	
400	0,00000000	86	2,00000000	1,98927500	
500	-0,15655580	100	1,98752800	1,98753000	0,15655580
100	0,23483370	79	1,97201700	1,97723600	
200	0,07827789	87	1,99687700	1,99321300	
300	0,23483370	88	1,97201700	1,99404800	
400	0,07827789	96	1,99687700	1,99619200	
500	0,15655580	69	1,98752800	1,98756500	0,15655580
100	-0,07827789	45	1,99687700	1,97468400	
200	-0,07827789	60	1,99687700	1,98579600	
300	0,00000000	89	2,00000000	1,99709500	
400	-0,07827789	100	1,99687700	1,99687600	
500	-0,07827789	100	1,99687700	1,99687600	0,07827789
100	-0,15655580	90	1,98752800	1,98268800	
200	-0,15655580	93	1,98752800	1,98840200	
300	-0,15655580	99	1,98752800	1,98470100	
400	-0,15655580	99	1,98752800	1,98765400	
500	-0,15655580	93	1,98752800	1,98840300	0,15655580
100	-0,23483370	49	1,97201700	1,98527500	
200	-0,07827789	59	1,99687700	1,98586900	
300	0,00000000	85	2,00000000	1,99953200	
400	0,00000000	73	2,00000000	1,99915700	
500	0,00000000	88	2,00000000	1,99797700	0,00000000
100	-0,07827789	49	1,99687700	1,99313200	
200	-0,07827789	42	1,99687700	1,98307500	
300	0,00000000	95	2,00000000	1,99757200	
400	0,00000000	96	2,00000000	1,99950100	
500	0,00000000	99	2,00000000	1,99987500	0,00000000
86	-0,15655580	100	1,98752800	1,98753000	
186	-0,15655580	99	1,98752800	1,98470100	
286	-0,15655580	95	1,98752800	1,97338500	
386	-0,15655580	99	1,98752800	1,98655200	

Step	x	%	y	*average* y	abs(0-x)
486	-0,15655580	99	1,98752800	1,98470100	0,15655580
100	0,15655580	60	1,98752800	1,98303600	
200	0,07827789	58	1,99687700	1,98494800	
300	0,15655580	83	1,98752800	1,98911800	
400	0,15655580	66	1,98752800	1,98393400	
500	0,15655580	100	1,98752800	1,98753000	0,15655580
100	0,07827789	56	1,99687700	1,97754000	
200	0,07827789	57	1,99687700	1,98537000	
300	0,07827789	55	1,99687700	1,99173500	
400	0,07827789	68	1,99687700	1,99052000	
500	0,07827789	50	1,99687700	1,99088300	0,07827789
100	-0,15655580	36	1,98752800	1,98663000	
200	-0,15655580	60	1,98752800	1,98471300	
300	-0,15655580	71	1,98752800	1,97374900	
400	-0,15655580	61	1,98752800	1,98259900	
500	0,00000000	44	2,00000000	1,99644400	0,00000000
100	0,00000000	79	2,00000000	1,99052300	
200	0,00000000	81	2,00000000	1,96739800	
300	0,00000000	98	2,00000000	1,98393800	
400	0,00000000	99	2,00000000	1,98957600	
500	0,00000000	100	2,00000000	2,00000000	0,00000000
100	6,88845400	23	1,55771100	1,64749600	
200	6,02739700	71	1,70965100	1,71259900	
300	6,18395300	81	1,71981400	1,69759600	
400	6,02739700	96	1,70965100	1,70362300	
500	6,02739700	89	1,70965100	1,70853700	6,02739700
100	0,00000000	94	2,00000000	1,99925200	
200	0,00000000	100	2,00000000	2,00000000	
300	0,00000000	100	2,00000000	2,00000000	
400	0,00000000	96	2,00000000	1,99876000	
500	0,00000000	99	2,00000000	1,99996900	0,00000000
100	-6,02739700	43	1,70965100	1,70819700	
200	-6,02739700	80	1,70965100	1,70524900	
300	-6,18395300	76	1,71981400	1,71209100	
400	-6,02739700	54	1,70965100	1,69228700	
500	-6,02739700	85	1,70965100	1,71117600	6,02739700
100	0,15655580	49	1,98752800	1,97428500	
200	0,07827789	57	1,99687700	1,98028500	
300	0,07827789	96	1,99687700	1,99588100	
400	0,07827789	97	1,99687700	1,98971500	
500	0,07827789	99	1,99687700	1,99678200	0,07827789

Step	x	%	y	*average* y	abs(0-x)
94	0,00000000	100	2,00000000	2,00000000	
194	0,00000000	51	2,00000000	1,98890900	
294	-0,31311150	58	1,95044800	1,97203100	
394	0,00000000	94	2,00000000	1,96399700	
494	-0,07827789	66	1,99687700	1,99893700	0,07827789
100	0,07827789	92	1,99687700	1,98979900	
200	0,07827789	100	1,99687700	1,99687600	
300	0,07827789	99	1,99687700	1,99448900	
400	0,07827789	100	1,99687700	1,99687600	
500	0,07827789	97	1,99687700	1,99563900	0,07827789
100	0,07827789	42	1,99687700	1,97288000	
200	0,07827789	56	1,99687700	1,98502700	
300	0,07827789	100	1,99687700	1,99687600	
400	0,07827789	100	1,99687700	1,99687600	
500	0,07827789	98	1,99687700	1,99539700	0,07827789
100	-0,15655580	48	1,98752800	1,98363800	
200	0,00000000	39	2,00000000	1,97288600	
300	-0,15655580	49	1,98752800	1,98500900	
400	0,00000000	95	2,00000000	1,99752200	
500	-0,15655580	99	1,98752800	1,99987500	0,15655580
100	0,54794520	38	1,85104200	1,92229800	
200	0,07827789	75	1,99687700	1,99066100	
300	0,07827789	82	1,99687700	1,98504400	
400	0,07827789	99	1,99687700	1,99448900	
500	0,07827789	100	1,99687700	1,99687600	0,07827789
100	0,23483370	79	1,97201700	1,96785900	
200	0,23483370	87	1,97201700	1,99364400	
300	0,07827789	94	1,99687700	1,98420100	
400	0,07827789	100	1,99687700	1,99687600	
500	0,07827789	95	1,99687700	1,97502800	0,07827789
100	0,31311150	70	1,95044800	1,95119500	
200	0,15655580	70	1,98752800	1,97440400	
300	0,15655580	98	1,98752800	1,98187200	
400	0,15655580	97	1,98752800	1,96875000	
500	0,15655580	91	1,98752800	1,98837100	0,15655580
91	0,15655580	100	1,98752800	1,98753000	
191	0,15655580	98	1,98752800	1,98678800	
291	0,15655580	93	1,98752800	1,98206700	
391	0,15655580	99	1,98752800	1,97923400	
491	0,15655580	99	1,98752800	1,98762300	0,15655580
100	-0,15655580	92	1,98752800	1,98742500	
200	-0,15655580	72	1,98752800	1,98606600	

Step	x	%	y	*average* y	abs(0-x)
300	-0,15655580	60	1,98752800	1,99519800	
400	0,00000000	73	2,00000000	1,99663300	
500	-0,15655580	52	1,98752800	1,99420100	0,15655580
100	-0,07827789	45	1,99687700	1,96110000	
200	-0,07827789	91	1,99687700	1,98315300	
300	-0,07827789	87	1,99687700	1,99728200	
400	0,00000000	84	2,00000000	1,98796700	
500	-0,07827789	52	1,99687700	1,99837500	0,07827789
100	0,54794520	83	1,85104200	1,91074100	
200	0,39138940	98	1,92296600	1,92230400	
300	0,07827789	55	1,99687700	1,96361700	
400	0,39138940	56	1,92296600	1,96096900	
500	0,07827789	65	1,99687700	1,96575400	0,07827789
100	-0,07827789	43	1,99687700	1,98891200	
200	-0,07827789	51	1,99687700	1,98877500	
300	-0,15655580	46	1,98752800	1,97746600	
400	0,00000000	63	2,00000000	1,99641400	
500	0,00000000	81	2,00000000	1,99763100	0,00000000
100	-6,34050900	19	1,71208500	1,65716000	
200	-6,10567500	84	1,71699600	1,70941000	
300	-6,10567500	95	1,71699600	1,71713600	
400	-6,10567500	96	1,71699600	1,70552700	
500	-6,10567500	99	1,71699600	1,71643000	6,10567500
100	-0,23483370	69	1,97201700	1,96377600	
200	-0,23483370	94	1,97201700	1,96954000	
300	-0,23483370	98	1,97201700	1,96959600	
400	-0,23483370	77	1,97201700	1,94518800	
500	-0,23483370	93	1,97201700	1,97310200	0,23483370
100	0,31311150	59	1,95044800	1,96301500	
200	0,15655580	32	1,98752800	1,95840500	
300	0,31311150	39	1,95044800	1,96825600	
400	0,15655580	82	1,98752800	1,97501300	
500	0,15655580	74	1,98752800	1,99444600	0,15655580
100	-0,07827789	24	1,99687700	1,78295800	
200	0,23483370	82	1,97201700	1,96558900	
300	0,23483370	100	1,97201700	1,97201600	
400	0,23483370	100	1,97201700	1,97201600	
500	0,23483370	98	1,97201700	1,96551200	0,23483370
100	0,07827789	41	1,99687700	1,96890400	
200	0,23483370	58	1,97201700	1,95818700	
300	0,23483370	97	1,97201700	1,96307800	
400	0,23483370	97	1,97201700	1,97136900	

Step	x	%	y	*average* y	abs(0-x)
500	0,31311150	68	1,95044800	1,95959000	0,31311150
100	-0,31311150	82	1,95044800	1,94550300	
200	-0,31311150	66	1,95044800	1,96729700	
300	0,00000000	70	2,00000000	1,98513500	
400	0,00000000	94	2,00000000	1,99060400	
500	0,00000000	100	2,00000000	2,00000000	0,00000000
100	-0,07827789	33	1,99687700	1,96920400	
200	-0,23483370	66	1,97201700	1,99548900	
300	-0,07827789	96	1,99687700	1,98632900	
400	-0,07827789	98	1,99687700	1,99210200	
500	-0,07827789	100	1,99687700	1,99687600	0,07827789
100	0,07827789	36	1,99687700	1,97100700	
200	0,15655580	64	1,98752800	1,98618600	
300	0,07827789	92	1,99687700	1,99612800	
400	0,07827789	91	1,99687700	1,96101300	
500	0,07827789	98	1,99687700	1,98064900	0,07827789
100	0,23483370	62	1,97201700	1,97535800	
200	0,07827789	84	1,99687700	1,97599300	
300	0,23483370	83	1,97201700	1,97624200	
400	0,23483370	97	1,97201700	1,97276200	
500	0,23483370	99	1,97201700	1,97080600	0,23483370
100	0,07827789	62	1,99687700	1,96590400	
200	0,07827789	88	1,99687700	1,99575500	
300	0,07827789	99	1,99687700	1,99662700	
400	0,07827789	91	1,99687700	1,98153500	
500	0,07827789	88	1,99687700	1,98879300	0,07827789
100	0,15655580	84	1,98752800	1,98159700	
200	0,15655580	95	1,98752800	1,98753300	
300	0,15655580	94	1,98752800	1,98809000	
400	0,15655580	90	1,98752800	1,97621800	
500	0,15655580	99	1,98752800	1,98111600	0,15655580
Sum					29,66731862
Ave	0,03819961	77	1,96380576	1,95909074	0,59334637
Var	3,06338053	470	0,00627576	0,00623816	2,66557834

Ciascuna tabella riporta cinque rilevazioni per ogni elaborazione; la cadenza di esecuzione delle rilevazioni dipende dalla rapidità, espressa dall'algoritmo genetico, nel fornire una popolazione dotata del sufficiente grado di omogeneità, fissato per tutte le prove al novantanove percento. Quando tale livello è stato raggiunto prima della centesima evoluzione, i dati sono stati rilevati immediatamente, proseguendo, poi, nella stampa dopo intervalli costanti di cento evoluzioni. Nei casi in cui il livello di omogeneità non è stato raggiunto entro la centesima evoluzione, la stampa è stata effettuata a partire da quest'ultima con intervalli costanti di cento evoluzioni ciascuno.

Ogni rilevazione ha prodotto, nell'ordine in cui i dati sono riportati nelle colonne delle tabelle, l'indicazione: del numero di *step* trascorsi dall'inizio dell'elaborazione (*step*), del valore di ascissa correlato al tipo prevalente nella popolazione al momento della rilevazione (x), del grado di diffusione del tipo suddetto (%), del valore di *fitness* ad esso assegnato (y), e del valor medio della *fitness* della popolazione in quel momento (*average* y). Sono stati omessi i dati relativi ai valori minimo e massimo di *fitness* per non appesantire l'esposizione; questi dati influiscono sul valore medio della *fitness* e quindi i loro effetti sono comunque presenti.

Per comodità, è stata aggiunta una ulteriore colonna, recante il valore assoluto dello scarto fra il valore di x relativo al tipo più frequente ed il valore zero, ascissa in corrispondenza della quale la funzione raggiunge il valore massimo.

Differenze di segno negativo fra *fitness* del tipo più ricorrente (y) e media delle *fitness* (*average* y), indicano la presenza, nella popolazione, di minoranze formate da individui migliori di quelli più ricorrenti; specie in presenza di un basso tasso di omogeneità (%), ciò può indicare delle forti possibilità di miglioramento. Differenze di segno opposto, se elevate, potrebbero preludere ad una instabilità del risultato, in quanto normalmente accompagnate da bassi tassi di omogeneità. Minime differenze, positive, fra i due valori, possono invece essere considerate normali, in quanto causate dagli effetti della mutazione.

La semplice lettura delle tabelle permette di rilevare la maggior precisione dei risultati ottenuti col metodo innovativo: il valore zero, nella colonna delle x, è presente con maggior frequenza. Tale metodo ha consentito, inoltre, di ottenere i suddetti risultati in un numero minore di evoluzioni: il tasso di omogeneità del 99% è, normalmente, ottenuto con meno di cinquanta evoluzioni, contro le forti difficoltà di convergenza dell'algoritmo gestito in modo tradizionale.

L'analisi dei valori medi rivela la sistematica tendenza ad un errore maggiore da parte dell'algoritmo basato sul metodo tradizionale. Data la particolare forma della funzione, tale tendenza, non provoca grandi differenze nella media dei valori di *fitness*. La maggior rapidità di convergenza dell'algoritmo più innovativo è leggibile sia nella media del grado di omogeneità della popolazione, sia nella differenza fra le medie dei valori medi di *fitness* della popolazione. I dati relativi all'algoritmo tradizionale risultano, inoltre, caratterizzati da una più spiccata varianza. La media degli scarti della x dal valore ottimale, espressi in valore assoluto, conferma e dimostra la maggior efficacia dell'algoritmo innovativo: l'errore medio di quest'ultimo risulta, infatti, inferiore alla metà di quello presentato dall'algoritmo tradizionale.

I risultati ottenuti possono essere riepilogati distribuendo in classi la frequenza degli scarti rispetto al valore corretto, cioè zero, in valore assoluto.

Tabella 3.3 - Riepilogo dei risultati della sperimentazione

	Gestione differenziale della *fitness*	**Gestione tradizionale della *fitness***
Differenza = 0	25	8
$0 \leq$ Differenza < 0.1	23	18
$0.1 \leq$ Differenza < 1	0	20
$1 \leq$ Differenza < 10	2	4
Totale risposte	50	50

Gli algoritmi genetici del primo tipo hanno individuato, perfettamente, il punto di massimo assoluto nel cinquanta percento dei casi, contro un sedici percento attribuibile a quelli tradizionali. Questi ultimi presentano, inoltre, un errore superiore al decimo in venti casi ed all'unità in quattro; i primi presentano solo due casi di errore elevato, superiore all'unità, mentre nei restanti ventitrè casi l'errore è inferiore al decimo.

Gli algoritmi genetici basati sulla gestione tradizionale della *fitness* hanno raggiunto un grado elevato di omogeneità della popolazione solo in 23 casi su 50: in alcuni casi la prosecuzione dell'evoluzione, dopo il raggiungimento di una elevata omogeneità, ha comportato un regresso verso soluzioni scorrette.

Gli algoritmi genetici basati sulla gestione differenziale hanno raggiunto la omogeneità del 99%, per la più parte dei casi, in meno di quaranta cicli: la media esclusi i due casi limite, diciotto e oltre cento cicli, rispettivamente, è trentasette. Questi algoritmi genetici hanno, inoltre, dimostrato, in tutti i casi, capacità di mantenere il livello di correttezza delle risposte, pur se sono state effettuate sempre quattrocento evoluzioni dopo il raggiungimento del grado di omogeneità prescritto. In un caso, una popolazione totalmente uniforme è riuscita ad evolvere, in cento cicli, verso un'altra, di pari omogeneità, passando da uno scarto nell'ordine di 0,15 alla risposta corretta.

La limitazione delle osservazioni, a 500 cicli, non ha permesso di verificare se gli algoritmi genetici gestiti con metodo tradizionale avrebbero potuto migliorare le loro risposte in un maggior numero di cicli; la possibilità non può essere esclusa visto il basso livello di omogeneità della popolazione da loro raggiunto, pur se, nella più parte dei casi, l'emergere di uno stesso tipo dominante porterebbe a considerarla piuttosto remota.

Variando la lunghezza del genoma cambia il numero di punti dell'intervallo del dominio rappresentabili nella popolazione. Al proposito sono state effettuate prove con differenti lunghezze; da queste pare rilevarsi che utilizzando genomi cortissimi si ottenga una immediata convergenza verso il valore ottimale: con lunghezza pari a due, l'intera popolazione suggeriva, durante le prove, il valore esatto di x dopo sole tre evoluzioni. Le diverse prove, condotte con specifiche lunghezze, hanno permesso di rilevare una tendenza alla maggior costanza di comportamento con lunghezze elevate, pur se la più precisa esplorazione del dominio ha richiesto l'effettuazione di un numero di evoluzioni più grande, prima di raggiungere il grado di omogeneità voluto della popolazione.

4 - Alcuni esperimenti in campo economico

In questo capitolo sono presentati due modelli volti a simulare situazioni di mercato. L'obiettivo è quello di dimostrare le possibilità di proficuo utilizzo del metodo; per questo motivo si è ritenuto opportuno basare i modelli su formulazioni teoriche note, utilizzando, inoltre, valori che permettessero un'agevole verificazione dei risultati.

La prima parte del capitolo è dedicata all'analisi dell'interazione fra un insieme di consumatori ed uno di produttori, verso il conseguimento di un livello di equilibrio dei prezzi. I primi sono rappresentati da oggetti operanti in base ad una funzione di utilità prestabilita, definita dall'utilizzatore del modello mediante la codificazione di appositi parametri. I produttori sono, invece, individui portatori di una propria valutazione del prezzo ottimale, che, scambiando le informazioni relative alla bontà di ciascuna strategia, perseguono il raggiungimento del livello massimo di profitto. Lo scambio di informazioni avviene, implicitamente, per il tramite delle operazioni effettuate nell'ambito della conduzione di un algoritmo genetico, affidata al *ruleMaker* di GM. Viene illustrato come la sola modificazione delle modalità di interazione, in particolare del numero di fornitori a cui ogni compratore può rivolgersi, possa portare alla convergenza del modello su equilibri tipici dei casi di monopolio, o di concorrenza atomistica.

Nella seconda parte, dopo un breve accenno alla "teoria dei giochi", viene illustrato un modello di duopolio di Bertrand: le imprese scelgono i prezzi, sperimentando livelli diversi di quantità vendute. Il mercato è rappresentato da un solo oggetto che provvede a calcolare le quantità domandate a ciascuna impresa, dato un certo saggio di sostituzione dei beni. Ad ogni ciclo le imprese fissano simultaneamente il prezzo, in base ai suggerimenti ricevuti dai rispettivi *classifier system;* in seguito, dopo aver calcolato l'ammontare dei profitti realizzati, ciascuna impresa determina il valore della ricompensa, da attribuire alle regole applicate, e la comunica al *classifier system*. L'indagine svolta riguarda gli effetti delle variazioni del saggio di sostituzione fra i due beni.

4.1 - Caratteristiche comuni ai due modelli

Lo scopo della presente trattazione è di illustrare i risultati ottenibili con tecniche di computazione automatica, nella simulazione di attività economiche. I modelli presentati, perciò, risultano basati su presupposti estremamente semplici, volti a mettere in rilievo l'effetto di singole variabili; l'azione avviene in ambienti caratterizzati da scarsa incertezza, onde permettere un rigido confronto dei risultati ottenuti con quelli forniti da modelli teorici noti. Ciò riduce, sicuramente, il grado di realismo dei modelli, ma permette di dimostrare in modo maggiormente preciso, la bontà del metodo.

I due modelli presentati non perseguono la ricerca di nuovi risultati, essendo, invece, pensati per documentare come i risultati, derivanti dall'applicazione di teorie note, possano essere ottenuti con metodologie innovative. Solo a partire da questa dimostrazione, è possibile ipotizzare l'utilizzazione di tali metodi per ricercare risultati affatto nuovi od emergenti. Il lavoro vuole, dunque, costituire un punto di inizio, in cui alla presentazione del metodo, svolta nei capitoli precedenti, si fa seguire la dimostrazione tangibile delle sue potenzialità, onde suffragare la bontà dei risultati che, con il suo impiego, potranno essere ottenuti in altri modelli, maggiormente complessi e dettagliati.

Per i motivi su esposti, le funzioni adottate nei modelli ed il comportamento degli attori sono banalizzati e, volutamente, omogenei: tutti i compratori, ad esempio, agiscono in base ad una stessa funzione di utilità, espressa in forma lineare, e le strutture dei costi di tutte le imprese sono perfettamente uguali. Questa scelta non deriva da "pigrizia intellettuale", ma è stata operata nell'intento di consentire, a chi legge, di cogliere, senza ricorrere a complesse verificazioni, la implicita certificazione della correttezza del metodo.

Un elemento d'interessante innovazione è l'impiego di grandezze non continue, espresse da numeri naturali: tutti i risultati delle computazioni sono ottenuti troncando l'eventuale parte decimale;

si vuole, con ciò, dimostrare la immediatezza di applicazione dei metodi evolutivi, già richiamata, nel capitolo secondo, a proposito dei vantaggi di tali strumenti. Un simile metodo di computazione risulta, inoltre, maggiormente realistico: esso permette di tenere conto dell'esistenza di vincoli legati alle minime quantità trattabili ed alla misura, necessariamente discreta, della più piccola variazione di prezzo praticabile. Tali elementi non possono essere considerati, invece, quando occorra garantire caratteristiche di continuità alla funzione di domanda, indispensabili per trattare la medesima con i metodi tradizionali, tipicamente basati sul calcolo differenziale.

4.2 - Simulazione di mercati

Il modello è volto alla ricerca del prezzo di equilibrio, in una situazione di mercato che viene a crearsi per il tramite dell'interazione, ripetuta, di un numero variabile di acquirenti e fornitori. Gli acquirenti agiscono in base ad una funzione di utilità, eguale per tutti, che li porta ad esprimere livelli di domanda correlati al prezzo. La domanda di ciascuno è definita come funzione lineare del prezzo con coefficiente angolare negativo ed intercetta, con l'asse delle ordinate, pari ad un livello massimo, definito dall'utilizzatore del modello valorizzando un apposito parametro.

I fornitori, rappresentati da agenti in possesso di una propria strategia riassunta nel livello di prezzi praticato, hanno come obiettivo la massimizzazione del profitto, data una funzione, nuovamente lineare, di costo totale, definita fissando, con la valorizzazione dei rispettivi parametri, il livello dei costi fissi totali e l'ammontare del costo variabile unitario, presupposto ad andamento costante. La struttura dei costi è uguale per ogni impresa e non esistono limiti di capacità produttiva. I fornitori non possiedono informazioni relative alla curva di domanda, né conoscono il numero di acquirenti con cui possono interagire. Al termine di ciascun periodo, cioè di ciascun ciclo del modello, valutano la bontà della propria strategia calcolando, semplicemente, l'ammontare del profitto, o della perdita, realizzati.

Il rapporto fra consumatori e produttori è mediato da un oggetto "rivenditore". Tale oggetto non effettua alcuna azione diretta sui prezzi, non si configura perciò come un "banditore", in grado di equilibrare il mercato, ma più come un grosso punto di vendita, dove l'acquirente può acquisire informazioni sul prezzo dei beni, forniti da un certo numero di produttori, prima di decidere quale acquistare ed in quale quantità. La rinuncia all'acquisto si ha solo quando il minor prezzo, conosciuto dall'acquirente, risulti troppo elevato, data la funzione di domanda precedentemente descritta. Per comodità di computazione, il rivenditore, che non possiede magazzino, comunica, di volta in volta, l'ammontare di ciascuna vendita al produttore; l'ammontare totale del venduto è utilizzato dallo stesso per il calcolo dei profitti conseguiti. Il rivenditore non incassa provvigioni né persegue propri scopi di lucro. La funzione dell'oggetto è, in verità, esclusivamente computazionale: la sua presenza permette di evitare interazioni dirette fra compratori e venditori ottenendo due benefici: chiarezza della struttura del modello, quindi maggior facilità di descrizione dell'azione gestita in esso, e maggior semplicità nella definizione dei vari oggetti, con conseguente agevolazione delle attività di redazione e modificazione dei programmi.

Tutti i beni prodotti sono perfetti sostituti; quindi la scelta dei consumatori avviene esclusivamente in base al prezzo. I beni, inoltre, non sono conservabili: non si possono accumulare scorte. Non esistono vincoli di capacità produttiva da parte dei fornitori, né limitazioni alla capacità di spesa degli acquirenti che, naturalmente, tendono a consumare, quindi ad acquistare, in ogni ciclo, la quantità di beni sufficiente a rendere massima la propria utilità, espressa in funzione del livello dei prezzi.

Ad ogni ciclo tutti gli acquirenti si rivolgono al rivenditore per ottenere informazioni relative ai prezzi, incaricandolo di ricercare il fornitore che pratica quello più basso; ricevuta l'informazione, decidono la quantità da acquistare e ne comunicano l'ammontare al rivenditore. Questi trasmette i dati dell'ordine al produttore che provvede alla fornitura e ne contabilizza l'importo. Al termine del periodo, i vari fornitori calcolano l'ammontare del profitto conseguito, dato l'andamento delle vendite.

Tutte le imprese usufruiscono dei servizi di un consulente di *marketing*, rappresentato dal *ruleMaker*, che fornisce, ad ogni impresa, indicazioni sulle strategie di *pricing* da seguire. L'azione del consulente si basa sul confronto fra i risultati della singola impresa e quelli del settore, formato dalle imprese stesse; i dati utilizzati sono, unicamente, il prezzo praticato al pubblico ed il profitto ottenuto. E' verosimile che, per quelle informazioni, non esistano problemi di riservatezza: il prezzo è, solitamente, conosciuto e l'ammontare del profitto è desumibile dal bilancio di ciascuna impresa. Il consulente tende, logicamente, a consigliare strategie alternative alle imprese che presentino minori profitti; quindi quelle, più probabilmente, modificheranno i propri prezzi. Le strategie consigliate sono ottenute emulando, in tutto o in parte, quelle, desumibili dai dati conosciuti, delle imprese migliori. Le imprese con profitti più elevati, invece, sono incoraggiate a perseverare nella politica sperimentata; quindi è probabile che i loro prezzi non mutino. Il consulente effettua le scelte, relative alle strategie, in modo stocastico, permettendo di immettere nel modello un grado di incertezza, tale da renderlo maggiormente aderente alla realtà, e contribuendo alla nascita di atteggiamenti "innovativi", che risultano maggiormente efficaci di quelli puramente "conservativi" nella ricerca del massimo profitto. Il consulente utilizza, per determinare ogni nuova strategia, materiale desunto da solo due delle altre conosciute; una di queste ultime, inoltre, può essere la strategia destinata ad essere sostituita.

La ripetizione ciclica delle azioni elencate produce la convergenza del modello verso uno stabile equilibrio. Il livello dei prezzi di equilibrio tende a variare a seconda delle possibilità di scelta degli acquirenti: ognuno richiede al rivenditore l'ammontare del più basso prezzo e, in base a quello, decide quale quantità acquistare. Il rivenditore effettua la ricerca nell'ambito dei fornitori più vicini all'acquirente, onde ridurre l'onerosità della consegna. Tramite il valore di un apposito parametro, è possibile stabilire l'estensione dell'area di consegna dei fornitori, determinando il numero di possibili scelte per l'acquirente. Nel caso più restrittivo, ogni acquirente può rivolgersi ad un solo fornitore, che, simmetricamente, possiede un solo cliente; aumentando il valore del parametro si configurano aree sempre più grandi di influenza delle varie imprese, dando luogo ad un atteggiamento concorrenziale. Nel primo caso i vari agenti, pur non conoscendo la curva di domanda, giungono ad esprimere livelli di prezzo più alti, tipici di un mercato monopolistico; la semplice estensione delle possibilità di scelta, implicite, dell'acquirente, comporta la riduzione dei prezzi, fino ai livelli tipici del mercato di perfetta concorrenza.

I progressi compiuti dai vari agenti vengono controllati dall'oggetto "osservatore" che provvede a raccogliere e visualizzare i dati relativi. Più in dettaglio, l'osservatore verifica, in base alla cadenza specificata dall'utilizzatore, quale prezzo venga maggiormente applicato e quale sia il grado di omogeneità della popolazione di agenti. L'oggetto provvede a produrre un diagramma dove sono rappresentati i due dati, aggiornandolo ad intervalli di ampiezza determinata dall'utilizzatore; sempre a scelta dell'utilizzatore, l'osservatore può anche produrre la stampa dei dati utili per successive elaborazioni statistiche. L'osservatore, inoltre, dopo aver acquisito i parametri del modello, effettua un'elaborazione preventiva per individuare quale sia il livello ottimale dei prezzi che dovrebbe affermarsi durante l'azione dello stesso. Grazie a questo espediente, è possibile disporre di un controllo automatico dei risultati ottenuti.

4.2.1 - Interpretazione evoluzionista del modello

L'evoluzione delle strategie di *pricing*, espresse dai diversi agenti, basata sulla sopravvivenza di quelle capaci di garantire i profitti più elevati, concorre a determinare l'omogeneizzazione dei comportamenti dei diversi produttori, fino ad individuare un livello di prezzi da ritenersi ottimale.

Il paradigma evolutivo utilizzato partecipa delle caratteristiche di quello individuale, in quanto ogni agente possiede un unico genoma che contiene la codificazione del prezzo praticato; nella formazione di nuove strategie avviene, perciò, un'implicita comunicazione fra agenti. Le remore, relative all'impiego di simili modelli per rappresentare entità inanimate, come le imprese, espresse nel secondo capitolo, vengono superate portando l'azione dell'evoluzione sulle strategie praticate dalle imprese, invece che sulle imprese stesse. Il fenotipo sarebbe, quindi, rappresentato dal prezzo applicato, mentre il genotipo consterebbe della sua codificazione in forma di successione di valori

zero ed uno. Data questa impostazione, l'estinzione di un fenotipo, indispensabile per giustificare le modificazioni del genotipo, comporterebbe semplicemente il rinnovamento della strategia dell'impresa, non il dissesto della medesima con la conseguente sostituzione da parte di una nuova azienda. In questo senso, è possibile rilevare un contributo del paradigma mentale nella configurazione del modello. Tale contributo permette, inoltre, di giustificare la comunicazione, implicata dalle attività di copia e *crossover*, nei termini illustrati nel paragrafo precedente.

Il grado di adattamento di ciascun tipo all'ambiente, *fitness*, è espresso dal livello di profitti ottenuto dall'agente con l'applicazione del fenotipo risultante, cioè del prezzo. L'evoluzione dei vari tipi, risulta perciò, influenzata da tutte le condizioni ambientali.

4.2.2 - Divisione funzionale in base allo schema ERA

La precisa distinzione delle funzioni di ciascun attore permette di garantire un completo rispetto delle convenzioni introdotte con lo schema ERA (Terna 1998). In particolare, l'impiego di oggetti specializzati nelle funzioni di comunicazione e di evoluzione delle regole consente di salvaguardare la rigida separazione fra i diversi livelli del modello.

La presenza dell'oggetto "venditore" permette di evitare comunicazioni dirette fra elementi appartenenti allo stesso livello, quello degli agenti, che si avrebbero, invece, se gli acquirenti dovessero rivolgersi direttamente alle imprese, per avere informazioni relative ai prezzi e comunicare l'ammontare dei propri acquisti.

La delegazione, al *ruleMaker*, dell'attività di evoluzione, garantisce la netta separazione fra livello degli agenti e dei produttori di regole, rendendo agevole la modificazione del modello, per introdurre, eventualmente, procedure di apprendimento basate su tecniche diverse.

Date le caratteristiche del metodo, e del paradigma evolutivo adottato, non è richiesto un gestore di regole: il terzo livello risulta, perciò, vuoto; ciò non deve essere, comunque, inteso come mancata aderenza allo schema suddetto.

4.2.3 - Oggetti ed interazioni nel modello

Il modello è stato sviluppato a partire dal pacchetto GM, aggiungendo il codice relativo alle classi: *Agent*, per le imprese, *Customer*, per gli acquirenti e *Shop*, per il rivenditore. Il ruolo del consulente è svolto da un oggetto della classe *RuleMaker*, già codificata in GM. La qualifica di "agente" è stata attribuita alle imprese, poiché è il comportamento di queste ultime a costituire oggetto di studio.

Delle classi *Agent* e *Customer* vengono create molteplici istanze, i cui indirizzi sono immessi in apposite liste; per ogni *agent* viene creato un oggetto della classe *Interface*, destinato a contenere un genoma, rappresentante la strategia di *pricing*, ed il valore di *fitness* ad esso associato. La classe *Interface* è stata modificata, in modo da renderla capace di trasformare il proprio genoma in un livello di prezzo, per rispondere alla richiesta di decodificazione inviatale dall'agente. Le necessarie variazioni sono state effettuate anche nel codice delle classi *ModelSwarm* ed *ObserverSwarm*.

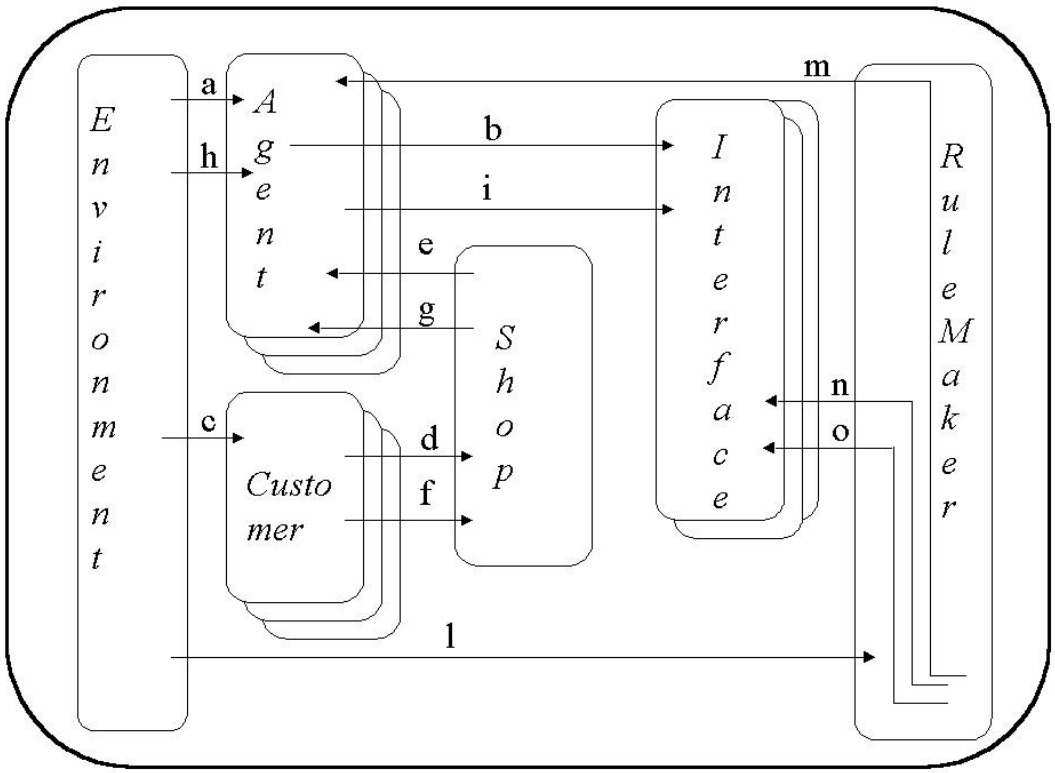
L'azione del modello è, come di consueto, coordinata dal *modelSwarm*, che provvede ad inviare, per ogni ciclo, la sequenza di ordini codificata durante la redazione del modello, ad imprese ed acquirenti. Il *modelSwarm* si occupa anche di contattare il *ruleMaker* affinché effettui l'evoluzione delle strategie, cioè esegua un passo di computazione dell'algoritmo genetico sull'insieme delle *interface*.

Il *ruleMaker* comunica con gli agenti, poiché ne è il consulente, per richiedere a ciascuno l'indirizzo della propria *interface*, acquisendo, così, tutte le informazioni necessarie per eseguire il

proprio compito. Imprese ed acquirenti non interagiscono direttamente, come consigliato in ERA, ma si valgono dei servizi di intermediazione dell'oggetto *shop*.

Le interazioni fra i vari oggetti sono rappresentate, con maggior dettaglio, nella figura 4.1.

Figura 4.1 - Modello di mercato



L'azione parte dal *modelSwarm* che ordina (a) ad ogni agente di fissare un prezzo; per far ciò ciascun agente richiede alla propria *interface* di decodificare (b) il genoma, in essa contenuto. Il genoma viene valorizzato, all'atto della creazione dell'*interface*, in modo casuale; in seguito viene modificato in base ai dettami del *ruleMaker*.

A questo punto il *modelSwarm* ordina (c) a tutti gli acquirenti di comprare. Ciascun acquirente interpella *shop*, richiedendo (d) di ricercare l'impresa che applica il minor prezzo, in un sottoinsieme, della lista degli agenti, di ampiezza definita dall'utilizzatore del modello. Una volta individuata l'impresa che applica il prezzo più favorevole, *shop* ne memorizza l'indirizzo e restituisce, all'acquirente, l'ammontare del prezzo. L'attività di ricerca è condotta interpellando (e) uno o più agenti, per conoscere quale prezzo intendano praticare.
L'acquirente calcola quante dosi del bene acquistare e comunica (f) la sua decisione a *shop*. L'ammontare di tale vendita è immediatamente notificato (g), da *shop*, all'agente.

Dopo che l'ultimo cliente ha effettuato i suoi acquisti, il *modelSwarm* ordina (h) a tutti gli agenti di calcolare i profitti conseguiti; una volta effettuata la computazione, ciascuna impresa comunica (i) alla propria *interface* il risultato della medesima, in guisa di *fitness*.

Ancora il *modelSwarm* ordina (l) al *ruleMaker* di procedere ad un'evoluzione. Questi interroga (m) ciascun agente per avere l'indirizzo della sua *interface*, onde formare la lista delle *interface* con cui interagire. Il *ruleMaker* richiede (n) a ciascuna *interface* notizie riguardanti il genoma e la *fitness*; procede, poi, all'evoluzione e comunica (o), alle *interface* prescelte, il contenuto del nuovo genoma.

A cura del *modelSwarm*, l'intero ciclo viene ripetuto fino a quando l'utilizzatore decida di arrestare l'azione del modello. Il *modelSwarm* prevede la possibilità di immettere tutti i parametri necessari a modificare la situazione simulata nel modello; per alcune di queste grandezze vengono proposti valori pseudo-casuali di *default*, in modo da permettere la generazione automatica di una elevata quantità di casi.

L'*observerSwarm* provvede, dopo la creazione del *modelSwarm*, a determinare i valori ottimali di prezzo, dati i parametri della simulazione ottenuti dal *modelSwarm* stesso: eventuali variazioni ai valori proposti come *default*, vengono, così, recepite. Durante l'azione del modello l'*observerSwarm* effettua una costante rilevazione dei prezzi e dei profitti, interrogando i diversi agenti, e computa il livello di omogeneità della popolazione. In base alle specifiche richieste dell'utilizzatore, i dati ottenuti vengono visualizzati in un apposito diagramma e possono, inoltre, essere stampati su *standard-output* per venire utilizzati in successive elaborazioni statistiche.

4.2.4 - I parametri dell' *observerSwarm*

La *probeMap* dell'*observerSwarm* permette di selezionare la frequenza di aggiornamento del grafico, prodotto durante l'esecuzione del modello, e della stampa, su *standard-output*, dei dati relativi ai progressi compiuti dagli agenti. Un terzo valore riguarda il grado di omogeneità, della popolazione, ritenuto significativo del raggiungimento di un risultato valido.

Il valore del parametro *displayFrequency* stabilisce il numero di cicli, del modello, che dovranno intercorrere fra due aggiornamenti consecutivi del grafico.

Il parametro *printFrequency* è utilizzato per determinare la cadenza della stampa dei risultati; i medesimi vengono stampati, inoltre, al raggiungimento del livello di omogeneità della popolazione stabilito e quando il prezzo, maggiormente praticato dagli agenti, corrisponde a quello corretto. Assegnando valore zero a *printFrequency* tutte le stampe vengono soppresse.

Il parametro *confidence* permette di stabilire il grado di omogeneità della popolazione, da ritenersi significativo. Il valore è espresso in termini unitari: indicando uno, si stabilisce che è da ritenersi significativa solo la completa omogeneità dei comportamenti.

Per ciascun parametro viene proposto, all'utilizzatore, un valore di *default*: cento per *displayFrequency*, cento per *printFrequency* ed uno per *confidence*. Se l'utilizzatore non effettua personalizzazioni, perciò, ogni cento passi viene aggiornato il grafico e vengono stampati i dati relativi all'evoluzione. Detti dati vengono, inoltre, stampati, una sola volta, quando tutte le imprese giungono a praticare lo stesso prezzo.

4.2.5 - I parametri del *modelSwarm*

La ricca dotazione di parametri, in grado di influire sull'azione di questo oggetto, permette di personalizzare il modello, onde sperimentarne il funzionamento in situazioni diverse. Sono previste due tipologie di valori: i primi riguardano le caratteristiche del modello, gli altri il funzionamento dell'algoritmo genetico.

Per quanto riguarda il modello è possibile stabilire, nell'ordine di presentazione dei parametri: numero di imprese, numero di acquirenti, ammontare dei costi fissi, valore del costo variabile unitario, quantità massima domandata dagli acquirenti, inclinazione della curva di domanda, numero di imprese cui ciascun compratore può rivolgersi.

Per l'algoritmo genetico sono previsti: lunghezza del genoma, grado di ricambio generazionale, probabilità di *crossover*, probabilità di mutazione, frequenza di evoluzione, valore, o

metodo di determinazione, della *fitness* assegnata ai nuovi individui, indicatore di scelta tra metodo tradizionale di selezione o trattazione in base alle differenze di *fitness*. Il significato di questi parametri è stato trattato, nel capitolo precedente, durante l'illustrazione del metodo.

In merito alla valorizzazione di questi parametri occorre aver presenti alcune peculiarità; una breve disamina di quelle risulta utile per una più chiara comprensione dell'azione del modello.

La lunghezza del genoma deve essere tale da consentire la rappresentazione, in forma binaria, del massimo livello di prezzo che si desidera ammettere. Come spiegato in precedenza, lo spazio della ricerca è determinato da questo valore; in particolare occorre ricordare che il massimo valore rappresentabile, dato un genoma di lunghezza m, è pari a 2^m-1, mentre il minimo è zero. Intervalli di valori diversi, contenenti anche valori negativi o frazionari, possono essere utilizzati previo intervento sul codice della classe *Interface*.

La funzione di domanda, utilizzata dagli acquirenti, è espressa dalla seguente equazione:

maxDemand + *demandSlope* * prezzo.

Data la decisione di utilizzare unicamente valori naturali, anche *demandSlope* è trattato come tale: non possono essere specificati importi frazionari. Ne consegue che le variazioni di quantità domandate risultano discrete; tale caratteristica si estende, inoltre, all'andamento dei profitti. I prezzi in grado di massimizzare una simile funzione potranno, perciò, differire, in alcuni casi, da quelli calcolati in base ad una funzione continua.

Il numero di oggetti della classe *Agent* determina la numerosità della popolazione, di *interface*, su cui agisce l'algoritmo genetico: per il buon funzionamento del metodo è bene che questo numero non risulti troppo esiguo. Ogni oggetto delle classi *Agent* e *Customer* riceve un numero progressivo all'atto della creazione, corrispondente alla sua posizione nella rispettiva lista. Il criterio di territorialità viene applicato in base a quel numero: sono considerati vicini gli oggetti che portano lo stesso numero. Quando riceve la richiesta, da parte di un compratore, di ricercare il minor prezzo, *shop* interroga un numero di fornitori pari al valore del parametro *searchLevel*, partendo da quello che occupa analoga posizione, nella lista delle imprese, rispetto a quella occupata, nella lista dei compratori, dal richiedente. La ricerca prosegue, poi, incrementando di una unità alla volta la posizione dell'impresa consultata; quando viene raggiunta l'ultima impresa, la ricerca prosegue ripartendo dalla prima. Se il numero di acquirenti è inferiore a quello delle imprese, aumentato di *searchLevel* a meno di un'unità, tutte le imprese con numero superiore a quell'ammontare non vengono consultate da *shop*; il loro livello di vendite è nullo per qualunque prezzo, data la mancanza di acquirenti. Sperequazioni nella distribuzione delle possibilità di vendita possono derivare, per motivi analoghi, anche quando il numero di compratori, pur risultando superiore a quello delle imprese, non sia multiplo di quest'ultimo. In relazione a quanto esposto, è consigliabile fissare sempre un numero di compratori pari ad un multiplo del numero di imprese, o, al limite, uguale.
Il comportamento dell'oggetto *shop* è motivato dall'applicazione del criterio di territorialità: si considerano tanto più vicini due oggetti, quanto minore è la differenza intercorrente, fra il numero di posizione del secondo e quello del primo.

Il *modelSwarm*, come l'*observerSwarm*, prevede valori di *default* per ciascun parametro; la peculiarità sta nella determinazione, pseudo-casuale, dei "*default*" per i parametri: *maxDemand*, *demandSlope*, *fixedCost* e *directCost*. Tale accorgimento permette all'utilizzatore di ottenere, automaticamente, la simulazione di casi diversi, semplicemente specificando l'opzione "-s", all'atto dell'esecuzione del programma. Il valore assegnato, casualmente, a ciascun parametro viene generato in modo da garantirne l'appartenenza agli specifici intervalli sotto riportati:

200	≤	*MaxDemand*	≤	256
-4	≤	*demandSlope*	≤	-1
51	≤	*fixedCost*	≤	100
1	≤	*directCost*	≤	50

L'accorgimento descritto, in addizione alle possibilità computazionali inserite

nell'*observerSwarm*, permette di rendere completamente automatici la generazione ed il controllo di un'ampia serie di casi.

4.2.6 - Monopolio

Stabilendo che ogni acquirente possa rivolgersi, dato il vincolo di vicinanza, ad una sola impresa, ciascuna di queste viene ad operare in regime di monopolio. I risultati, forniti dal modello, dimostrano che il metodo è stato capace di rilevare questa peculiarità del mercato, giungendo a consigliare la fissazione del prezzo migliore. Tale prezzo viene individuato, normalmente, per il tramite dell'analisi della curva, presupposta continua, dei profitti: in particolare, attesa l'inclinazione negativa della curva di domanda, la funzione di profitto presenta valori negativi della derivata seconda; quindi il suo massimo è individuabile come quel valore di ascissa che garantisce l'azzeramento della derivata prima.

La curva di domanda è rappresentata dall'equazione:

$q = maxDemand + demandSlope * p$

dove q rappresenta la quantità venduta dato il prezzo, indicato con p. L'andamento dei costi marginali è presupposto costante; il loro ammontare coincide con quello dei costi variabili unitari, espresso dal valore del parametro *directCost*. Si ottiene, perciò, una curva di profitto data da:

$U = (maxDemand * p + demandSlope * p^2) - (directCost * maxDemand + directCost * demandSlope * p + fixedCost)$.

la cui derivate prima e seconda risultano, rispettivamente:

$U' = maxDemand + 2 * demandSlope * p - directCost * demandSlope$.

$U'' = 2 * demandSlope$

Il prezzo, capace di garantire il massimo profitto è determinato, perciò, dall'espressione:

$p = (maxDemand - directCost * demandSlope) / (2 * (-1) * demandSlope)$.

La figura 4.2 riporta il grafico dei prezzi praticati durante l'azione del modello, data una possibilità nulla di scelta, valore di *searchLevel* pari ad uno, da parte del compratore. Il livello massimo di domanda è stato fissato a 256, i costi diretti a 50 e *demandSlope* a -1. Da quanto esposto sopra ci si attende la fissazione di un prezzo pari a:

$(256 + 50) / 2 = 153$.

Per comodità di lettura, oltre al grafico originale, riportato in alto a sinistra, sono esposti due ingrandimenti che permettono di rilevare il numero di cicli, occorsi per giungere ad un alto livello di omogeneità della popolazione, ed il livello di prezzo determinato.

Come risulta evidente, le imprese, pur non conoscendo direttamente l'andamento della curva di domanda, giungono, con l'esperienza, ad individuare la miglior strategia di *pricing*: viene fissato un prezzo simile a quello determinabile in base alla teoria dei mercati monopolistici. L'effetto si produce a prescindere dall'applicazione di regole conosciute a priori; esso scaturisce unicamente dall'interazione fra gli operatori del modello.

A sostegno di questa affermazione sono state effettuate varie simulazioni, basate sull'impiego di parametri diversi per la determinazione delle quantità domandate e dei costi totali; prima di ciascuna elaborazione è stato variato il valore di base per la generazione dei numeri pseudo-casuali. Ciascuna elaborazione è stata basata su un modello costituito da duecento imprese ed altrettanti

compratori; ogni impresa risultava proprietaria di un'*interface*, portatrice di un genoma e del relativo valore di *fitness*. L'evoluzione della popolazione di genomi è stata eseguita ad ogni ciclo del modello, adottando un *turnover* del 50%, una eguale probabilità di *crossover* e una probabilità di mutazione pari ad un millesimo; per la gestione delle selezioni è stato adottato il metodo basato sulle differenze di *fitness*, illustrato nel capitolo precedente. I risultati ottenuti sono riepilogati nella tabella 4.1.

Figura 4.2 - Monopolio

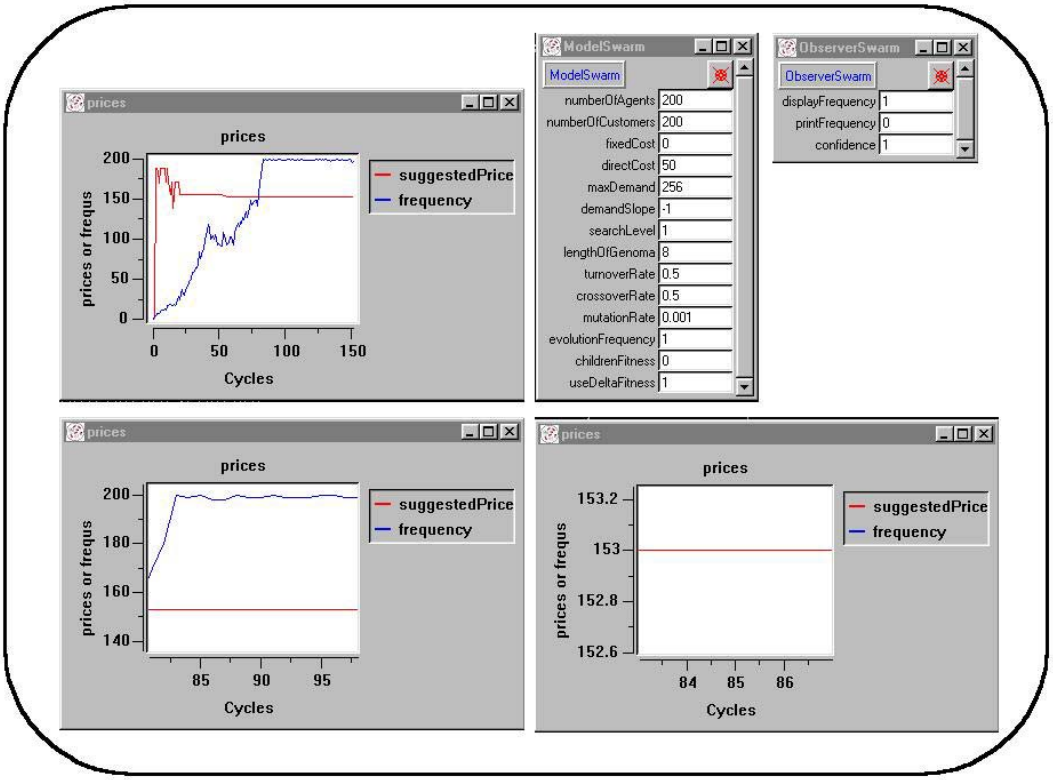


Tabella 4.1 - Prezzi in regime monopolistico

N	sL	mD	dS	fC	dC	rP	P	S	F	sP	delta
1	1	216	-3	99	4	38	3369	119	200	38	0
2	1	209	-3	71	42	56	503	83	200	56	0
3	1	228	-1	66	40	134	8770	105	200	134	0
4	1	208	-3	100	30	50	1060	87	200	50	0
5	1	211	-1	63	2	106	10857				
5	1	211	-1	63	2	107	10857	411	200	107	0
6	1	235	-3	71	18	48	2659	65	200	48	0
7	1	231	-2	99	44	80	2457	58	200	79	-1
8	1	223	-3	98	30	52	1376	91	200	52	0
9	1	233	-4	72	3	31	2980	43	200	31	0
10	1	221	-4	98	5	30	2427	88	200	30	0
11	1	221	-2	95	43	77	2183	65	200	77	0
12	1	249	-3	85	35	59	1643	64	200	59	0
13	1	242	-3	73	18	49	2872	139	200	49	0
14	1	237	-3	72	12	45	3294	109	200	45	0

N	sL	mD	dS	fC	dC	rP	P	S	F	sP	delta
14	1	237	-3	72	12	46	3294				
15	1	234	-2	76	33	75	3452	132	200	75	0
16	1	241	-1	64	43	142	9737	109	200	142	0
17	1	237	-2	57	6	62	6271	75	200	62	0
18	1	228	-3	86	32	54	1366	84	200	54	0
19	1	236	-4	95	30	44	745				
19	1	236	-4	95	30	45	745	238	200	45	0
20	1	253	-3	76	2	43	5008	93	200	43	0
21	1	226	-3	76	16	46	2564	109	200	46	0
22	1	247	-4	55	40	51	418	112	200	51	0
23	1	223	-3	51	27	51	1629	49	200	51	0
24	1	228	-2	68	32	73	3294	52	200	73	0
25	1	249	-2	88	24	74	4962	68	200	74	0
26	1	223	-2	56	41	76	2429	124	200	76	0
27	1	242	-2	73	38	79	3371				
27	1	242	-2	73	38	80	3371	100	200	80	0
28	1	215	-4	98	50	52	-84	173	200	52	0
29	1	237	-2	52	14	66	5408	132	200	66	0
30	1	230	-4	97	13	35	1883	88	200	35	0
31	1	229	-3	56	2	39	4088	74	200	39	0
32	1	245	-3	81	11	46	3664	66	200	46	0
33	1	206	-4	76	3	27	2276	71	200	27	0
34	1	228	-3	62	35	55	1198	55	200	55	0
34	1	228	-3	62	35	56	1198				
35	1	250	-1	67	28	139	12254	127	200	139	0
36	1	213	-3	58	12	41	2552				
36	1	213	-3	58	12	42	2552	112	200	42	0
37	1	248	-3	91	21	52	2761	101	200	52	0
38	1	242	-1	94	45	143	9608	69	200	143	0
38	1	242	-1	94	45	144	9608				
39	1	223	-4	71	26	41	814	95	200	41	0
40	1	251	-1	95	48	149	10207				
40	1	251	-1	95	48	150	10207	103	200	150	0
41	1	213	-2	96	30	68	2830	104	200	68	0
42	1	237	-2	96	42	80	2830	88	200	80	0
43	1	203	-3	76	33	50	825	59	200	50	0
44	1	211	-4	76	35	44	239	106	200	44	0
45	1	224	-2	56	30	71	3306	91	200	71	0
46	1	214	-3	85	50	61	256	94	200	61	0
47	1	231	-3	63	46	61	657	111	200	64	3
48	1	200	-1	92	2	101	9709	94	200	101	0
49	1	205	-3	51	26	47	1293	59	200	47	0
50	1	226	-4	72	43	50	110	97	200	50	0
Media1							3659	101			
Media2								92			

La tabella riporta i dati relativi a cinquanta elaborazioni, ciascuna delle quali è contrassegnata da un numero progressivo, riportato nella colonna "N". Le colonne "sL", "mD", "dS", "fC" e "dC"

recano, rispettivamente, l'indicazione del valore dei parametri, *searchLevel*, *maxDemand*, *demandSlope*, *fixedCost* e *directCost*, utilizzati per la simulazione. In base a questi valori, l'*observerSwarm* ha provveduto a determinare il livello di prezzo capace di garantire il conseguimento del massimo profitto; tale valore è riportato nella colonna "rP". In relazione a quest'ultimo, è bene osservare come, in conseguenza di arrotondamenti, il suo ammontare possa differire dalla grandezza ottenibile con la semplice applicazione della formula:

p = (*maxDemand* - *directCost* * *demandSlope*) / (2 * (-1) * *demandSlope*).

La ricerca del prezzo ottimale viene effettuata confrontando l'ammontare dei profitti, ottenibili applicando tutti i prezzi compresi nell'intervallo]0,256], in base ad un procedimento puramente enumerativo. Tale procedura risulta applicabile solo in presenza di un esiguo numero di alternative possibili; la sua utilizzazione nel modello risponde a puro fine di verificazione.

Accanto al valore del prezzo ottimale, colonna "rP", viene riportato l'ammontare del profitto ottenuto, colonna P. La successiva colonna, contrassegnata da "S", riporta il numero di *step*, cioè di evoluzioni, compiute al fine di ottenere il grado voluto di omogeneità della popolazione: in questo caso il 100%, come mostrato dal valore riportato nella colonna adiacente, "F", dove è indicato il numero di imprese che assumono lo stesso comportamento in un dato momento. L'ammontare del prezzo praticato è riportato nella colonna "sP", cui segue l'indicazione, colonna "delta", della eventuale differenza fra il prezzo ottimale e quest'ultimo.

Il formato della tabella deriva da quello, utilizzato dall'*observerSwarm*, per la stampa dei dati, i valori di alcune colonne risultano costanti, in quanto, per semplicità espositiva, sono state omesse le righe relative al conseguimento di risultati intermedi.

L'analisi dei dati permette di rilevare l'ottenimento del risultato corretto in quarantotto casi, sui cinquanta documentati, dimostrando, così, la sistematica emergenza del fenomeno descritto o, se si preferisce, la capacità del modello di esprimere la corretta strategia, data la situazione di mercato derivante dalle regole di interazione e dai valori dei parametri adottati.

Il numero di evoluzioni, mediamente impiegate, risulta decisamente inferiore al numero di tentativi necessario, per l'esplorazione dello spazio dello soluzioni, utilizzando un piano puramente enumerativo: immaginando che ogni agente dovesse sperimentare, per ogni ciclo, uno dei valori dell'intervallo]0,256], memorizzando il livello di profitto ottenuto, per stabilire quale prezzo garantisse il conseguimento del miglior risultato, la convergenza di tutte le duecento imprese si sarebbe avuta, forzatamente solo dopo 255 "passi" di evoluzione. La convergenza del modello in un numero medio di 101 passi, consente di apprezzare, pur in presenza di poche alternative, la maggior potenza del metodo di ricerca basato sugli algoritmi genetici. La differenza fra piani enumerativi ed evolutivi per l'esplorazione dello spazio di soluzioni possibili tende, naturalmente, ad aumentare fortemente in presenza di un maggiore numero di alternative; oltre determinati livelli, l'applicazione di piani enumerativi risulterebbe, come descritto nel secondo capitolo, praticamente impossibile.

La maggior efficienza dei procedimenti evolutivi risulta ancor più marcata se si ha cura di correggere la media delle evoluzioni impiegate, tenendo conto della difficoltà comportata dall'esistenza di casi in cui valori diversi di prezzo implicano l'ottenimento di eguali profitti; tale evento si è manifestato durante le elaborazioni contrassegnate dai numeri: 5, 14, 19, 27, 34, 36, 38, 40. In casi simili, la convergenza della popolazione, verso un unico tipo, risulta ostacolata dall'assegnazione dello stesso valore di *fitness* a tipi diversi: nella fattispecie la popolazione tende a ripartirsi in due gruppi, ciascuno formato dagli individui facenti capo ad uno dei due tipi recanti i prezzi ottimali; il numero di evoluzioni, necessarie a determinare la prevalenza di uno dei due, può risultare molto elevato. Nella tabella, due elaborazioni, caratterizzate dalla peculiarità descritta, presentano valori fortemente devianti: la numero cinque, che ha richiesto ben 411 passi, e la numero diciannove, con 238. Dopo la correzione, consistente nell'esclusione dei casi suddetti dal calcolo, la media scende a 91, cioè a quasi un terzo degli *step* richiesti da un piano enumerativo.

Un caso particolarmente interessante è rappresentato dalla elaborazione numero 28, dove l'algoritmo genetico riesce ad esprimere un livello di prezzi in gradi di minimizzare le perdite. Tale

caso costituisce un ottimo esempio di utilizzazione della gestione differenziale dei valori di *fitness*: grazie a questo accorgimento, l'algoritmo riesce ad operare correttamente pur se i diversi individui risultano portatori di valori negativi di *fitness*.

4.2.7 - Concorrenza

Concedendo possibilità di scelta ai vari acquirenti, si nota una tendenza delle imprese ad esprimere prezzi tipici di un mercato concorrenziale. L'effetto si manifesta quando ogni compratore possa scegliere fra, almeno, il dieci percento delle imprese presenti sul mercato: il criterio di vicinanza geografica risulta, perciò, ancora rispettato. Come nel caso precedente, le imprese non trattano direttamente con i compratori, ma si limitano a fissare un prezzo e contabilizzare le vendite comunicate da *shop*. Ciascuna impresa, pur agendo, di fatto, in concorrenza con un certo numero di altre, non è al corrente di questa situazione, né conosce i propri acquirenti. Il prezzo praticato concorre a determinare le quantità, vendute da ciascuna impresa, in due modi: un livello sufficientemente basso può comportare la cattura di quote di mercato, la cui estensione minima riguarda l'intera domanda di un singolo compratore; il livello di quest'ultima è determinato dall'ammontare del prezzo, in base alla relazione, descritta al punto precedente:

$$q = maxDemand + demandSlope * p$$

dove p rappresenta il prezzo, praticato dall'impresa, e q la quantità, domandata da ciascuno dei compratori che si rivolge a quell'impresa.

L'andamento della domanda, rivolta a ciascuna impresa, risulta molto più irregolare che nel caso precedente; analogamente irregolare può presentarsi l'andamento dei profitti. Quando una impresa pratica un prezzo sufficientemente basso essa percepisce una domanda data da:

$$\sum (maxDemand_i + demandSlope_i * p)$$

dove p è ancora il prezzo praticato e la sommatoria è intesa estendersi ai valori relativi a tutti i "clienti" dell'impresa; dato il meccanismo codificato in *shop*, il numero di potenziali clienti risulta pari al doppio del numero di possibili imprese che ciascuno può consultare ridotto di un'unità. Inizialmente, perciò, la singola impresa trova molto conveniente procedere a riduzioni del prezzo: fino a quando non venga imitata da altre, essa cattura consistenti quote di domanda, sperimentando profitti molto elevati. La percezione della curva di domanda, che è rimasta immutata a livello individuale, da parte dell'impresa viene, perciò, alterata dalla variabile estensione del mercato dei propri prodotti. Anche altre imprese possono tendere a ridurre i prezzi, per catturare quote di mercato: si origina, così, una vera e propria competizione al ribasso che porta a fissare un prezzo appena superiore al livello di copertura dei costi variabili.

La competizione avviene fra tutte le imprese: il meccanismo di ricerca, codificato nell'oggetto *shop*, evita le segmentazioni del mercato. Ogni compratore, in pratica, si rivolge alle stesse imprese cui si indirizzano i propri vicini, con l'eccezione di una che risulta, sistematicamente, "nuova"; diremo allora che ogni compratore si rivolge ad m + 1 imprese. Ciascuna delle m imprese viene, perciò, a concorrere, per ogni nuovo compratore, con una nuova impresa; la scelta da parte di quest'ultimo, relativa alla decisione di acquisto, comporta un arricchimento delle informazioni, implicite, per ciascuna delle m imprese. In un simile mercato, le riduzioni competitive di prezzo tendono a propagarsi, per effetto delle scelte dei compratori: le imprese concorrono, direttamente, per catturare la domanda di un singolo compratore, e indirettamente per consolidare la propria posizione rispetto al gruppo.

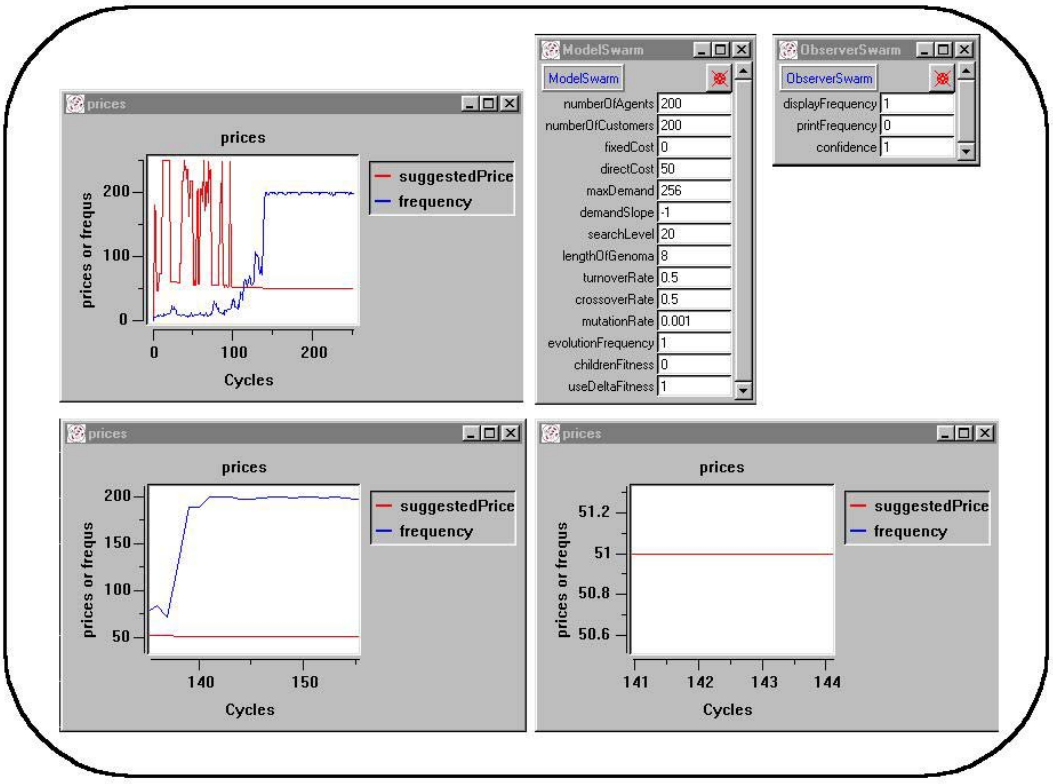
Il concetto può essere meglio chiarito con un esempio: siano inseriti nel modello cinque imprenditori e cinque compratori; ciascun compratore possa rivolgersi, come risulta nella tabella 4.2, a due imprese, cioè a quella che occupa, nella lista delle imprese, la stessa posizione occupata dal compratore, nella opportuna lista, o a quella seguente. Il compratore uno sceglierà, allora, l'impresa che pratica il minor prezzo fra la uno e la due; il compratore due fra la due e la tre e così via. Il

compratore cinque potrà rivolgersi all'impresa cinque o alla uno. La decisione dell'impresa quattro, ad esempio, di ridurre il prezzo, condizionerà sia l'impresa cinque, sia l'impresa tre; la decisione della tre condizionerà, a sua volta, le vendite della due, mentre quella della cinque agirà sulle opportunità della uno. Con un maggior numero di imprese, pur con un più ampio numero di scelte, il meccanismo di propagazione agirà in modo analogo.

Tabella 4.2 - Interazione fra compratori e imprese

Compratore	1° Impresa	2° Impresa
Uno	Uno	Due
Due	Due	Tre
Tre	Tre	Quattro
Quattro	Quattro	Cinque
Cinque	Cinque	Uno

Figura 4.3 - Concorrenza



Il livello dei prezzi in mercati perfettamente concorrenziali, quando si ipotizza di operare su quantità continue, tende all'eguaglianza con il valore del costo marginale; operando, però, con quantità e prezzi discreti, è ragionevole attendere l'affermarsi di un prezzo, appena superiore all'ammontare del costo marginale: lo scarto fra i due dovrebbe risultare pari all'ammontare minimo di variazione del prezzo. Nel caso presentato, l'ammontare del prezzo è espresso da un numero naturale, quindi parrebbe lecito qualificare come "concorrenziale" il comportamento di quella impresa che fissasse il suo prezzo al livello del costo marginale aumentato di una unità. Ricordando che l'andamento dei costi marginali è stato ipotizzato costante e coincidente con quello dei costi variabili unitari, il livello dei prezzi emergente dall'elaborazione dovrebbe risultare da:

$$p = directCost + 1$$

indipendentemente dal valore di *maxDemand*, *demandSlope* e *fixedCost*.

La figura 4.3 illustra l'elaborazione del modello precedentemente utilizzato; l'unico valore modificato, rispetto al caso di monopolio, è quello di *searchLevel*. In pratica l'unica differenza, rispetto al caso precedente, sta nella possibilità del compratore di scegliere fra più imprese.

Dal grafico in basso a destra, risulta la scelta del livello di prezzo di concorrenza, date, naturalmente, le considerazioni svolte in precedenza sull'utilizzo di quantità discontinue. Il numero di evoluzioni, necessarie per ottenere la convergenza della popolazione, risulta maggiore a quello del caso monopolio ma, come verrà dimostrato in seguito, ciò è incidentale.

Anche il caso concorrenziale è stato approfondito con l'esecuzione di prove ripetute; le modalità di esecuzione sono del tutto simili a quelle esposte a proposito del caso di monopolio: i valori dei parametri sono decisi casualmente dal *modelSwarm* e prima di ogni elaborazione è stato variato il "seme" di generazione dei numeri casuali. La tabella 4.3 riporta il risultato della varie elaborazioni.

Tabella 4.3 - Prezzi in regime concorrenziale

N	sL	mD	dS	fC	dC	rP	P	S	F	sP	delta
1	20	232	-2	79	41	42	237	100	200	42	0
2	20	241	-1	81	33	34	194	57	200	34	0
3	20	208	-4	90	9	10	158	34	200	10	0
4	20	255	-2	69	6	7	200	124	200	7	0
5	20	210	-1	78	4	5	137	41	200	5	0
6	20	222	-3	72	32	33	249	38	200	33	0
7	20	210	-1	93	4	5	122	64	200	5	0
8	20	247	-4	68	2	3	191	29	200	3	0
9	20	206	-1	76	50	51	181	102	200	51	0
10	20	214	-4	77	37	38	289	76	200	38	0
11	20	220	-1	96	1	2	126	13	200	2	0
12	20	230	-4	59	35	36	315	49	200	36	0
13	20	224	-4	52	19	20	252	109	200	20	0
14	20	237	-1	95	18	19	161	200	200	19	0
15	20	216	-1	80	21	22	158	194	200	22	0
16	20	202	-2	79	19	20	163	90	200	20	0
17	20	218	-2	84	38	39	212	87	200	39	0
18	20	232	-4	80	1	2	160	13	200	2	0
19	20	225	-1	100	47	48	173	88	200	48	0
20	20	206	-3	64	30	31	235	19	200	32	1
21	20	210	-3	83	3	4	139	23	200	4	0
22	20	224	-2	90	5	6	146	81	200	6	0
23	20	247	-4	82	49	50	365	65	200	50	0
24	20	239	-1	71	47	48	216	73	200	48	0
25	20	234	-4	81	17	18	225	64	200	18	0
26	20	213	-3	64	33	34	251	31	200	34	0
27	20	214	-2	73	29	30	201	22	200	32	2
28	20	207	-4	65	43	44	318	44	200	44	0
29	20	203	-1	94	24	25	134	75	200	25	0
30	20	246	-1	97	48	49	198	68	200	49	0
31	20	211	-3	68	4	5	158	168	200	5	0
32	20	212	-2	92	2	3	126	26	200	3	0

N	sL	mD	dS	fC	dC	rP	P	S	F	sP	delta
33	20	219	-4	88	38	39	287	106	200	39	0
34	20	227	-2	54	37	38	249	82	200	38	0
35	20	252	-3	73	1	2	185	30	200	2	0
36	20	229	-2	63	34	35	236	42	200	35	0
37	20	214	-2	89	24	25	175	74	200	25	0
38	20	217	-3	99	35	36	226	68	200	36	0
39	20	214	-3	91	10	11	156	49	200	11	0
40	20	216	-2	63	35	36	225	33	200	36	0
41	20	232	-4	72	23	24	256	64	200	24	0
42	20	229	-1	98	45	46	177	107	200	46	0
43	20	223	-4	98	15	16	189	46	200	16	0
44	20	207	-1	65	12	13	155	157	200	13	0
45	20	246	-3	71	33	34	277	55	200	34	0
46	20	242	-4	66	48	49	372	64	200	49	0
47	20	210	-2	100	3	4	118	26	200	4	0
48	20	201	-2	96	5	6	117	34	200	6	0
49	20	205	-2	73	50	51	234	158	200	51	0
50	20	234	-1	56	1	2	180	11	200	2	0
Media							204	69			

La struttura della tabella è uguale a quella usata per il caso di monopolio; si può notare, in questa, che il valore di *searchLevel*, colonna "sL", è stato portato a venti. Il prezzo di riferimento, quello della colonna "rP", in base al quale si calcola l'errore, colonna "delta", è stato ottenuto, dall'*observerSwarm*, applicando rigidamente la formula:

p = *directCost* + 1

Una esplorazione di tutto lo spazio delle soluzioni con individuazione dei massimi livelli di profitto, simile a quella utilizzata nel caso precedente, non avrebbe fornito risultati tangibili; in questo caso l'effetto scaturisce dalla interazione, indiretta, dei vari agenti, per il tramite dei movimenti di quote di domanda: nessun agente possiede strumenti per stabilire quale sia il prezzo ottimale. Ogni impresa, pur perseguendo la massimizzazione del profitto, viene costretta, dalla concorrenza delle altre, a stabilire un prezzo che le permetta di restare sul mercato. Il prezzo finale viene, perciò, a formarsi solo con il contributo di tutti gli agenti, sia per il tramite dello scambio di strategie, materiale genetico, sia per i condizionamenti reciproci che ognuna immette nell'attuare la propria strategia. Questa seconda interazione, particolarmente, scaturisce esclusivamente dal modello e ne qualifica il realismo.

Un punto di considerevole interesse economico è dato dallo scaturire dell'effetto esclusivamente dall'interazione delle parti componenti il modello; non esistono meccanismi che equilibrano il mercato, il semplice sforzo di ottimizzazione esplicato sia dai compratori, sia dalle aziende, risultante dai banali obiettivi di acquistare al minor prezzo e di massimizzare i profitti, è risultato sufficiente a determinare un equilibrio.

Il livello di prezzi corretto, cioè compatibile con gli assunti teorici riguardanti la concorrenza atomistica, viene raggiunto in quarantotto elaborazioni e fortemente avvicinato nelle due restanti. Ciò dimostra, sia la correttezza del modello, sia l'abilità dell'algoritmo genetico: la prima si traduce nella efficacia nel simulare un mercato concorrenziale, la seconda permette di gestire con successo la competizione. La concorrenza delle due componenti parrebbe agire addirittura a favore dell'efficienza: il numero di evoluzioni mediamente impiegate, in ciascuna elaborazione, risulta decisamente inferiore che nel caso di monopolio.

Da un punto di vista puramente tecnico, l'algoritmo genetico risulta facilitato nella ricerca,

data la forte differenza di *fitness* fra i genomi esprimenti prezzi tali da tenere l'impresa sul mercato e gli altri. Non occorre, però, sottovalutare la difficoltà derivante dal livellamento della *fitness* di questi ultimi: dato l'annullamento delle vendite, conseguente all'applicazione del prezzo da loro consigliato, il valore di *fitness* assegnato risulterebbe sistematicamente pari all'opposto dell'ammontare dei costi fissi. Si consideri, inoltre, che l'appartenenza di un genoma ad uno dei due tipi, dipende dalla composizione della popolazione: un genoma recante un prezzo capace di catturare domanda può, dopo l'evoluzione, essere "spiazzato" dalla nascita di un altro, recante un prezzo inferiore e compatibile con il livello dei costi diretti.

4.3 - Cenni sulla teoria dei giochi

Ambito di applicazione della teoria è l'analisi delle decisioni coinvolgenti più individui, specialmente quando le decisioni di ciascuno influiscano sulle conseguenze verso altri, o verso l'intera collettività di giocatori. In questi casi, la razionalità di ciascun giocatore si basa anche sulla conoscenza delle, analoghe, capacità degli altri, determinando l'insorgere di costrutti congetturali estremamente complessi. Sylos Labini (1982), con riferimento alle indagini di tipo "psicologico" sui comportamenti in mercati oligopolistici, basate sulle curve di reazione di ciascun operatore, scrive:

> (…) La verità è che sulla via delle "variazioni congetturali" (Cred'io ch'ei credette ch'io credesse), non ci si ferma mai. Le soluzioni possono essere aumentate all'infinito ed il proporre siffatte ipotesi e soluzioni può divenire un sorta di mestiere.

Proprio in questi contesti la teoria ha trovato le prime applicazioni di interesse economico. L'utilizzazione del paradigma del "gioco" può, però, risultare proficua anche in altri ambiti: nella microeconomia, ad esempio, è utile per la formalizzazione di modelli di scambio. In campo macroeconomico sono stati proposti modelli basati sugli effetti dell'interazione fra le autorità monetarie e le altre istituzioni incaricate della fissazione di prezzi e salari. La teoria è stata sfruttata anche per indagini finanziarie e per studi sul mercato del lavoro. In campo internazionale, utili congetture possono essere tratte riguardo alla collusione, fra paesi, nella scelta di tariffe e politiche commerciali. Non bisogna, infine, dimenticare ambiti interni all'azienda quali: la competizione fra le varie divisioni per ottenere fondi da investire o risorse da impiegare.

L'interesse per la presente trattazione riguarda, in particolare, un modello di collusione fra oligopolisti appartenente ai cosiddetti "giochi statici con informazione completa". La caratterizzazione della tipologia può essere rinvenuta in Gibbons (1992):

> (…) i giocatori scelgono simultaneamente le azioni e successivamente ricevono i *payoff* i quali dipendono dalle particolari azioni scelte. All'interno di questa classi di giochi statici (o con mosse simultanee) concentriamo l'attenzione sui giochi con informazione completa, cioè quei giochi in cui la funzione dei *payoff* di ogni giocatore è nota a tutti i giocatori.

Si intende, solitamente, per *payoff* l'espressione quantitativa del livello di beneficio, tratto da ciascun giocatore, da ogni singola mossa. L'esempio classico è quello del noto "dilemma del prigioniero": a due indiziati, posti nell'impossibilità di comunicare, viene notificato che potranno avere sensibili riduzioni di pena se confesseranno; la riduzione, inoltre, sarà maggiore, per quello dei due che avrà confessato, se l'altro non avrà ammesso il crimine. In caso di confessione simultanea la pena risulterà di medio rigore; la stessa sarà, invece, inasprita nei confronti di colui che, non confessando, verrà incriminato in base alla confessione dell'altro. L'alternativa per ciascun giocatore è fra tacere o confessare; la combinazione dei comportamenti dei due indiziati configura quattro casi: se ambedue tacciono ottengono una pena lieve, se ambedue confessano subiscono una pena media, se uno confessa viene rimesso in libertà ma l'altro deve scontare una pena severa. Il *payoff* può essere rappresentato dal numero di periodi di detenzione comminati: se ambedue confessano ricevono un *payoff*, negativo, pari a -2, cioè scontano due mesi di carcere, se tacciono ricevono -1, chi confessa tradendo il compagno riceve 0 mentre l'altro riceve -3. L'azione di ciascuno, quindi, influisce sul risultato dell'azione dell'altro.

La situazione descritta, viene espressa in modo formale nella "rappresentazione in forma normale del gioco"; al proposito è riportata in Gibbons (1992) la seguente definizione:

> La rappresentazione in forma normale di un gioco con n giocatori specifica lo spazio delle strategie dei giocatori $S_1 \ldots S_n$ e le loro funzioni di *payoff* $u_1 \ldots u_n$. Questo gioco è indicato con $G = \{ S_1 \ldots S_n; u_1 \ldots u_n \}$.

La rappresentazione in forma normale specifica: quali sono i partecipanti al gioco, quali sono le strategie a disposizione di ciascuno, quali *payoff* sono ricevuti, da ognuno, per ciascuna delle combinazioni di strategie originate dai diversi comportamenti individuali. Si perviene, così, alla redazione di una "bimatrice", dove ogni posizione reca, separati da virgole, i valori di *payoff* per i vari partecipanti (da cui il prefisso bi), simile alle seguente:

Tabella 4.4 - Esempio di bimatrice

	Il prigioniero B tace	Il prigioniero B confessa
Il prigioniero A tace	-1,-1	-3,0
Il prigioniero A confessa	0,-3	-2,-2

Per convenzione viene indicato a sinistra della virgola il *payoff* del giocatore di riga e a destra quello del giocatore di colonna. S_i rappresenta l'insieme delle strategie a disposizione del giocatore i-esimo, mentre s_i indica una delle strategie appartenenti ad S_i.

Nell'esempio, data l'incertezza sul comportamento dell'altro, a ciascun giocatore conviene confessare: chi confessa viene scarcerato, se l'altro tace, oppure subisce una detenzione di due mesi, se anche l'altro confessa. Scegliendo di tacere, invece, egli subisce comunque una detenzione: di un solo mese, se l'altro tace, e di tre mesi, se l'altro confessa. I valori del *payoff,* associati alla scelta di tacere, risultano sistematicamente inferiori a quelli derivanti dalla decisione di confessare, a parità di comportamento da parte dell'altro giocatore. Si dice, in casi simili, che la strategia "confessare" domina la strategia "tacere". Il concetto è espresso, in modo generalizzato, in Gibbons (1992):

> Nel gioco in forma normale $G = \{ S_1 \ldots S_n; u_1 \ldots u_n \}$ siano s'_i e s''_i due strategie ammissibili per il giocatore i (cioè s'_i e s''_i sono elementi di S_i). La strategia s'_i è strettamente dominata dalla strategia s''_i se, per ogni combinazione ammissibile di strategie degli altri giocatori, il *payoff* che i riceve giocando s'_i è strettamente inferiore a quello che riceve giocando s''_i:
>
> $$u_i(s_1,\ldots, s_{i-1}, s'_i, s_{i+1},\ldots, s_n) < u_i(s_1,\ldots, s_{i-1}, s''_i, s_{i+1},\ldots, s_n)$$
>
> per ogni $(s_1,\ldots, s_{i-1}, s_{i+1},\ldots, s_n)$ ottenuto dagli spazi di strategie degli altri giocatori $S_1,\ldots, S_{i-1}, S_{i+1},\ldots, S_n$.

Conseguenza naturale della definizione è che giocatori razionali non giochino, mai, strategie strettamente dominate. La risoluzione del gioco procederebbe, perciò, per eliminazione iterata delle strategie strettamente dominate; per sfruttare tale possibilità occorre, però, presupporre che, oltre ad essere razionale, ciascun giocatore sappia che anche gli altri sono tali. Il procedimento, inoltre, non risulta applicabile a qualunque gioco.

Un concetto di soluzione più forte è fornito dall'equilibrio di Nash, definito, dall'autore già citato, nei termini seguenti:

> Nel gioco in forma normale con n giocatori, $G = \{ S_1 \ldots S_n; u_1 \ldots u_n \}$, le strategie $(s^*_i \ldots s^*_n)$ sono un equilibrio di Nash se, per ogni giocatore i, s^*_i è la miglior risposta del giocatore i alle strategie specificate per gli altri n-1 giocatori, $(s^*_i,\ldots, s^*_{i-1}, s^*_{i+1},\ldots, s_n)$:
>
> $$u_i(s^*_1,\ldots, s^*_{i-1}, s^*_i, s^*_{i+1},\ldots, s^*_n) \geq u_i(s^*_1,\ldots, s^*_{i-1}, s_i, s^*_{i+1},\ldots, s^*_n)$$

per ogni strategia ammissibile s_i in S_i; cioè se s^*_i risolve il problema

$$\max u_i (s^*_1, \ldots, s^*_{i-1}, s_i, s^*_{i+1}, \ldots, s^*_n).$$

Se esistesse una soluzione migliore di quella indicata dalle condizioni di Nash, almeno un giocatore sarebbe incentivato a deviare dalla prescrizione teorica e quindi, perché una convenzione di gioco, cioè un equilibrio, si affermi, occorre che le strategie, in essa contenute, costituiscano un equilibrio di Nash.

4.4 - Il duopolio di Bertrand

Si tratta di un gioco relativo all'interazione di due oligopolisti, in cui l'equilibrio è basato sul concetto di Nash. Le due imprese fissano i prezzi, subendo le decisioni del mercato per quanto concerne le quantità domandate. I prodotti sono differenziati ma parzialmente sostituibili.

La quantità domandata a ciascuna impresa risulta dall'espressione:

$$q_i(p_i,p_j) = a - p_i + bp_j$$

dove q e p rappresentano, rispettivamente quantità domandate e prezzi praticati; b esprime la misura in cui il prodotto dell'impresa i è sostituto di quello dell'impresa j, naturalmente il suo valore deve essere superiore a zero.

Si presume che i costi marginali siano costanti e pari a $c < a$, che non vi siano costi fissi e che le imprese stabiliscano i prezzi simultaneamente. La fissazione simultanea dei prezzi non implica che l'attività debba essere effettuata nello stesso momento da parte delle due imprese, bastando, per soddisfare il requisito, che ciascuna impresa, nel momento in cui stabilisce il nuovo prezzo, non conosca quanto, eventualmente, già deciso da parte dell'altra. L'impresa j può fissare il prezzo per il periodo uno, anche dopo che l'impresa i ha fissato il suo, mantenendo il requisito della simultaneità, purché il prezzo di i per il tempo uno non le sia noto; nulla vieta che j conosca, ad esempio, il prezzo praticato da i nel periodo zero.

Le strategie a disposizione di ogni giocatore sono i diversi livelli di prezzo, quindi lo spazio può essere rappresentato da $S_i = [0,\infty [$. L'ammontare del *payoff* è dato dal profitto conseguito da ciascuna impresa:

$$P_i(p_i,p_j) = q_i(p_i,p_j) [p_i - c] = [a - p_i + bp_j] [p_i - c].$$

La coppia (p^*_i,p^*_j) individua un equilibrio di Nash se, per ogni impresa, p* risolve il problema max $[P_i(p_i,p_j)]$.

Per l'impresa i, la soluzione di max(P) è data da:

$$p_i = (1/2) * (a + bp_j + c)$$

simmetricamente, per l'impresa j, si ha:

$$p_j = (1/2) * (a + bp_i + c).$$

Risolvendo il sistema costituito dalle due equazioni si ottiene:

$$p^*_i = p^*_j = (a + c) / (2 - b).$$

L'espressione è valida a condizione che $0 \leq b \leq 2$. La funzione di domanda adottata, inoltre, comporta che, per b = 1, la miglior strategia di collusione consista nella fissazione del massimo prezzo

ammissibile da parte di ambedue le imprese; con un saggio di sostituzione unitario, infatti, le quote di mercato trasferite fra imprese risultano esattamente uguali a quelle perdute per effetto dell'aumento dei prezzi. Generalizzando l'osservazione, è possibile affermare, data una funzione di domanda del tipo: $q_i(p_i,p_j) = a - xp_i + bp_j$, che, vista la limitata possibilità di scelta degli acquirenti, per b = x le imprese ritraggono il massimo profitto dalla simultanea adozione del più alto livello di prezzo possibile. La domanda rivolta a ciascuna impresa si compone, infatti, di due parti: una diretta, negativamente correlata al prezzo praticato dall'impresa, ed una indiretta, che è funzione crescente dei prezzi praticati dal concorrente: se i due coefficienti di correlazione risultano eguali, aumenti di prezzo lasciano invariata la domanda totale, poiché le riduzioni della domanda diretta sono esattamente compensate da aumenti di quella indiretta.

4.5 - Il modello basato sul duopolio di Bertrand

Il gioco esposto nel paragrafo precedente ha ispirato la redazione di un modello in cui viene applicato il metodo dei *classifier system*. Il meccanismo di interazione è simile a quello descritto; sono, però, state variate alcune assunzioni di partenza: prezzi e quantità sono trattati come grandezze discrete, la conoscenza delle funzioni di *payoff* viene acquisita, dai due agenti, durante l'interazione e l'insieme delle strategie, per esigenze computazionali, è stato limitato ad un numero finito di prezzi. La razionalità degli attori, impegnati nel tentativo di perseguire il massimo profitto, emerge dai progressi ottenuti dal *classifier system* di ciascuno, durante l'elaborazione del modello. Il gioco viene ripetuto per un elevato numero di volte, in modo che l'accumulazione di esperienza consenta l'apprendimento da parte del *classifier system*.

Le variazioni apportate, particolarmente il riferimento a quantità discrete, non permettono di valutare le prestazioni degli agenti in base alla equazione: $p^*_i = p^*_j = (a + c) / (2 - b)$; la validità dei risultati viene determinata, per ogni caso, in base ad una bimatrice appositamente costruita. L'abbandono del requisito di continuità della funzione di domanda comporta la necessità di effettuare arrotondamenti nel computo delle quantità; in conseguenza di detti, il livello di equilibrio dei prezzi può risultare, in certi casi, diverso da quello ottenibile applicando l'equazione suddetta. A seconda del saggio di sostituzione adottato, possono aversi casi in cui risulti indifferente l'applicazione di prezzi simili, ad esempio differenti di una sola unità; è, perciò, possibile ottenere uno stabile equilibrio anche se le due imprese adottano prezzi diversi.

Le differenze immesse tendono ad introdurre elementi di maggior difficoltà per i *classifier system*, quindi le modifiche risultano utili dato il fine di dimostrare l'efficacia del metodo. A tal proposito è stato deciso di includere nell'insieme delle strategie, cioè dei prezzi possibili, anche il prezzo zero; l'accorgimento permette di disporre di almeno una strategia assurda, compensando, sia pur parzialmente, la facilitazione conseguente alla limitazione dell'insieme delle stesse.

La limitazione dell'insieme delle strategie consente, infine, di ottenere sperimentalmente l'effetto paradossale illustrato in chiusura del paragrafo precedente.

Nel modello, due imprese, costituenti l'intero settore economico, concorrono a soddisfare la domanda, espressa da un insieme, indistinto, di acquirenti. Il rapporto fra imprese e clienti è gestito da un oggetto "intermediario" che provvede, inoltre, a computare i dati medi settoriali.

I beni prodotti dalle due imprese sono sostituibili nella misura specificata dall'utilizzatore; non sono conservabili né la loro domanda è soggetta a fenomeni di saturazione. La domanda formulata dagli acquirenti è unicamente funzione dei prezzi delle due imprese: il loro ammontare concorre a determinare sia la quantità totalmente richiesta sia la sua ripartizione fra i beni, forniti dalle due imprese, in base alla funzione mutuata dal gioco, precedentemente descritto. Gli acquirenti non hanno memoria né sono previsti atteggiamenti di fedeltà verso un particolare fornitore; il loro potere di spesa è considerato illimitato. Essi decidono, di volta in volta, la composizione del *mix* domandato unicamente in funzione del rapporto fra i prezzi dei beni e del saggio di sostituzione. In presenza di forti differenze di prezzo è possibile che le vendite di una impresa risultino nulle. Le imprese sono in

grado di fornire qualunque quantità domandata, non vengono accumulate scorte di prodotti e non sono previste limitazioni relative all'approvvigionamento ed all'impiego dei fattori produttivi.

Per elaborare le proprie strategie di *pricing,* le imprese si avvalgono di un *classifier system*, la cui gestione, pur se svolta da due oggetti comuni, risulta indipendente per ciascun agente. Le imprese non conoscono la funzione di domanda, né interagiscono direttamente con i consumatori. Ciascuna impresa richiede all'intermediario quale prezzo è praticato dal proprio concorrente, indi stabilisce il proprio prezzo, in base al consiglio ricevuto dal *classifier system*. Solo dopo che ambedue le imprese hanno elaborato il nuovo prezzo, viene loro ordinato di comunicarne l'ammontare all'intermediario; in questo modo l'unico dato a disposizione di ciascuna impresa è il prezzo praticato nel periodo precedente dal proprio concorrente. La disponibilità di questa informazione è sicuramente plausibile, riguardando un dato, solitamente, noto anche nella realtà.

Dato l'ammontare dei profitti realizzati, ciascuna impresa valuta la bontà della propria strategia confrontando questi ultimi, sia con i livelli conseguiti nel passato, sia con i livelli medi di settore. Anche questo comportamento risulta plausibile: imprese, operanti nella realtà, con scarsa conoscenza del mercato trovano, usualmente, nel confronto con l'esperienza propria e con l'andamento del settore, un efficace riferimento per valutare i risultati conseguiti; i dati relativi all'andamento del settore sono, inoltre, facilmente reperibili.

L'obiettivo di ciascuna impresa è il conseguimento del massimo profitto o, per lo meno, di un livello in linea con gli andamenti del settore. Dall'elaborazione emerge, però, anche un tacito accordo sui prezzi, i cui livelli tendono ad essere uguali per le due imprese, data anche la perfetta eguaglianza della struttura dei costi; questo effetto costituisce il principale motivo di interesse, non essendo l'accordo sul prezzo codificato in alcun modo fra gli obiettivi dei due agenti.

4.5.1 – Interpretazione evoluzionista del modello

L'impostazione del modello è rigorosamente mentale: i vari genomi, rappresentanti, in questo caso, le regole di reazione al comportamento del concorrente, possono essere assimilati alle diverse ipotesi di soluzione del problema, cioè alle "idee", che popolano il cervello di ciascun imprenditore. Ogni agente possiede un proprio patrimonio di regole, costituente la popolazione di un *classifier system*; durante la trattazione da parte del gestore, e del produttore, di regole, i patrimoni normativi dei due agenti sono conservati rigidamente distinti.

L'evoluzione agisce in modo da promuovere la formazione di regole, idee, la cui applicazione risulti capace di produrre un elevato livello di profitto; la bontà della singola regola viene, perciò, valutata in base alla differenza fra il profitto ottenuto e quelli realizzati in precedenza.

Il paradigma di reazione alla condizione ambientale, rappresentata dal livello conosciuto, relativo al periodo precedente, dei prezzi dell'altra impresa, ben si adatta all'applicazione di regole formate da un antecedente, la condizione, ed un conseguente, l'azione, quali quelle gestite dai *classifier system*. Il progresso della popolazione si realizza, praticamente, per il tramite dell'apprendimento relativo alla miglior azione da associare alla condizione corrente. All'inizio dell'elaborazione, i due agenti non conoscono la funzione di *payoff*, né possiedono elementi in relazione alle azioni che potranno essere intraprese dal loro concorrente: il loro agire non è razionale. Tale caratteristica viene a formarsi durante l'evoluzione e, dato particolarmente interessante, in modo cooperativo: ciascun agente raggiunge più facilmente buoni livelli di conoscenza, quanto più elevata è la razionalità del concorrente. Le conseguenze delle azioni dell'uno dipendono anche da quelle dell'altro; quindi i due agenti possono esprimere comportamenti razionali solo se ambedue progrediscono; in questo caso ciascun agente impara sia a rapportarsi correttamente con la funzione di domanda, sia a prevedere efficacemente le reazioni del proprio concorrente.

La presenza della mutazione risulta fondamentale per promuovere il sorgere di comportamenti nuovi, tali da garantire una esplorazione sufficientemente accurata dello spazio delle soluzioni possibili. Tale esplorazione deve essere effettuata molte volte, poiché, col progredire delle

conoscenze di ciascuna impresa, le reazioni dei due attori cambiano, contribuendo a vanificare l'efficacia di azioni, fino ad allora, buone; è, dunque, necessario un elevato numero di cicli evolutivi.

La simulazione risulta abbastanza realistica: l'iniziale incertezza potrebbe rispecchiare sia la diffidenza, naturale, di ciascuna impresa nei confronti di un ambito non ancora conosciuto, sia la relativa inesperienza, nella collusione, di imprese nuove. Né bisogna dimenticare la scarsa conoscenza delle reazioni della domanda che caratterizza l'ingresso in nuovi mercati o, il che è lo stesso, il lancio di prodotti fortemente innovativi. Col passare del tempo, le imprese acquisiscono informazioni ed esperienze che permettono loro di attuare comportamenti sempre più efficaci; nel contempo l'emergere di un'elevata razionalità, consente a ciascun concorrente di immaginare, se non prevedere, quali saranno le reazioni dell'altro. Dopo una sufficiente esplorazione delle diverse alternative i due concorrenti tendono a privilegiare quella che garantisca un elevato e stabile profitto; in altre parole tendono a concordare, implicitamente, un livello di prezzi capace di apportare un soddisfacente livello di profitti. L'emergere di questi forme di tacito accordo arreca una notevole stabilità dei prezzi; nella realtà i singoli prezzi potranno essere differenti; nel modello, invece, data l'identità delle funzioni di costo, essi tendono a convergere verso un eguale ammontare.

4.5.2 - Divisione funzionale in base allo schema ERA

Nell'ambiente rappresentato, come di consueto, dall'oggetto "modello" agiscono tre tipi di operatori: le due imprese, o agenti in senso stretto, un oggetto "cliente", rappresentativo dell'intera popolazione di acquirenti, ed un oggetto "intermediario". La funzione di quest'ultimo è precipuamente quella di gestire le comunicazioni fra aziende e mercato: in questo modo si evita la comunicazione diretta fra agenti e compratori, nel rispetto delle raccomandazioni di chiarezza e trasparenza. L'oggetto svolge, inoltre, un servizio di contabilità delle vendite, per ciascuna impresa, e provvede a calcolare il livello dei profitti del settore. L'intermediario non persegue fini propri, la sua azione è limitata alla fornitura di servizi agli altri due tipi; tale posizione è sottolineata dal fatto che l'oggetto viene unicamente interpellato dagli altri, senza mai prendere iniziative autonome.

L'insieme dei clienti agisce in base ad una regola fissa, data dalla funzione di domanda, già illustrata in precedenza. Tutti i beni acquisiti sono consumati, non esistono comportamenti di fedeltà alla marca o preferenze nei confronti di un particolare fornitore; non sono previsti casi di saturazione dei consumi. In ogni ciclo, il livello della domanda dipende unicamente dai prezzi praticati dalle due imprese e dal saggio di sostituzione dei beni.

Le imprese decidono, in ogni ciclo, il livello dei prezzi da praticare in risposta a quello adottato dall'impresa concorrente nel ciclo precedente; la fissazione dei prezzi avviene, perciò, in modo simultaneo. L'ammontare di questi ultimi è stabilito in base ai consigli ricevuti dal *classifier system*. Ogni impresa possiede un proprio patrimonio di regole, soggette ad evoluzione, quindi, pur utilizzando i servizi degli stessi oggetti di gestione del *classifier system*, ogni agente risulta dotato di un proprio sistema indipendente. Le regole di ciascun agente sono memorizzate in un apposito oggetto "magazzino"; ogni agente possiede un proprio magazzino di regole. Oltre al magazzino, ogni agente, possiede un proprio oggetto per la traduzione dei dati dalla metrica del modello a quella del *classifier system* e viceversa.

I vari *classifier system* sono gestiti da due oggetti, appartenenti, rispettivamente, al livello dei gestori di regole e a quello dei produttori di regole. Le funzionalità di selezione della regola da applicare e distribuzione delle ricompense sono affidate al gestore di regole, che viene interpellato ad ogni ciclo da ciascun agente. Il gestore provvede, data una certa probabilità di evoluzione, a richiedere i servizi del produttore di regole per rinnovare la popolazione di ciascun magazzino.

Rispetto al modello precedente, sono qui presenti tutti e quattro i livelli funzionali previsti dallo schema ERA: ambiente, agenti, gestori di regole e produttori di regole. La comunicazione avviene, in modo diretto, solo fra livelli diversi e contigui, mentre all'interno del livello degli agenti è mediata da appositi oggetti. Il modello rispetta, quindi, sia i requisiti di ERA, sia la rigida indipendenza dei patrimoni genetici, derivante dall'adozione del paradigma mentale nell'utilizzo del

metodo evolutivo. La precisa distinzione funzionale permette di ipotizzare ulteriori sviluppi del modello, verso una maggior differenziazione degli agenti che, ad esempio, potrebbero decidere di richiedere i servizi del gestore solo in relazione a livelli insoddisfacenti di profitto, o con cadenze diverse. Un secondo esperimento potrebbe riguardare l'adozione di agenti differentemente dotati: con magazzini diversamente popolosi, con *classifier system* caratterizzati da differenti parametri di gestione, con gestori di regole operanti in base ad altri paradigmi dell'intelligenza artificiale.

4.5.3 - Oggetti ed interazioni nel modello

Il modello è basato sul gestore di *classifier system*, rappresentato dal *kernel* di CW; tutte le classi del *kernel* di CW sono state incluse nel progetto senza necessità di modificazioni. Nell'ambito dello strato di *front-end*, mentre la classe *DataWarehouse* non ha richiesto interventi, è stata personalizzata la classe *Interface*. Nello strato utilizzatore, sono state aggiunte le classi relative all'intermediario ed agli acquirenti; le classi *Agent*, *ModelSwarm* ed *ObserverSwarm* sono state completamente rivedute.

Oltre alle consuete istanze, una per tipo, delle classi *ObserverSwarm* e *ModelSwarm*, vengono creati, nel modello: due oggetti agenti, ciascuno corredato dal proprio *dataWarehouse* e dalla propria *interface*, un oggetto intermediario, denominato *shop*, ed un oggetto rappresentativo della collettività degli acquirenti, denominato *customer*. La gestione delle regole contenute nei due *dataWarehouse* è affidata alla coppia di oggetti *ruleMaster*, gestore di regole, e *ruleMaker*, produttore di regole, ottenuti a partire dalle rispettive classi. Ciascun *dataWarehouse* provvede, inoltre, a creare le istanze di *Rule*, *Effector*, *Message* e *Treasury* nel numero specificato dai parametri di gestione del *classifier system* dell'agente. I *classifier system* dei due agenti hanno parametri di conduzione eguali, quindi, viene creata una sola istanza di *ClassifierParm*.

La dinamica del modello è basata sulle iniziative intraprese dal *modelSwarm* verso gli altri oggetti. Esso ordina agli agenti di decidere un livello di prezzi, dopo che ambedue hanno eseguito l'azione, trasmette loro l'ordine di pubblicare i prezzi, cioè di comunicarli all'oggetto *shop*. Poiché ciascun agente richiede a quest'ultimo, prima di decidere quale prezzo praticare, il prezzo applicato dal suo concorrente, la separazione delle azioni di elaborazione e pubblicazione del prezzo garantisce la simultaneità della sua fissazione.

Una volta che i prezzi sono stati notificati a *shop*, il *modelSwarm* ordina al *customer* di comprare; in seguito viene ordinato agli agenti di calcolare i profitti, che saranno, inoltre, comunicati, a cura di ciascun agente, a *shop*. Terminata la fase di computazione dei profitti, *shop* riceve l'ordine di aggiornare le statistiche dei dati settoriali. In ultimo, viene ordinato, agli agenti, di effettuare la valutazione del proprio livello di soddisfazione; tale valutazione influisce sull'ammontare della ricompensa che ciascun agente conferisce al proprio *classifier system*.

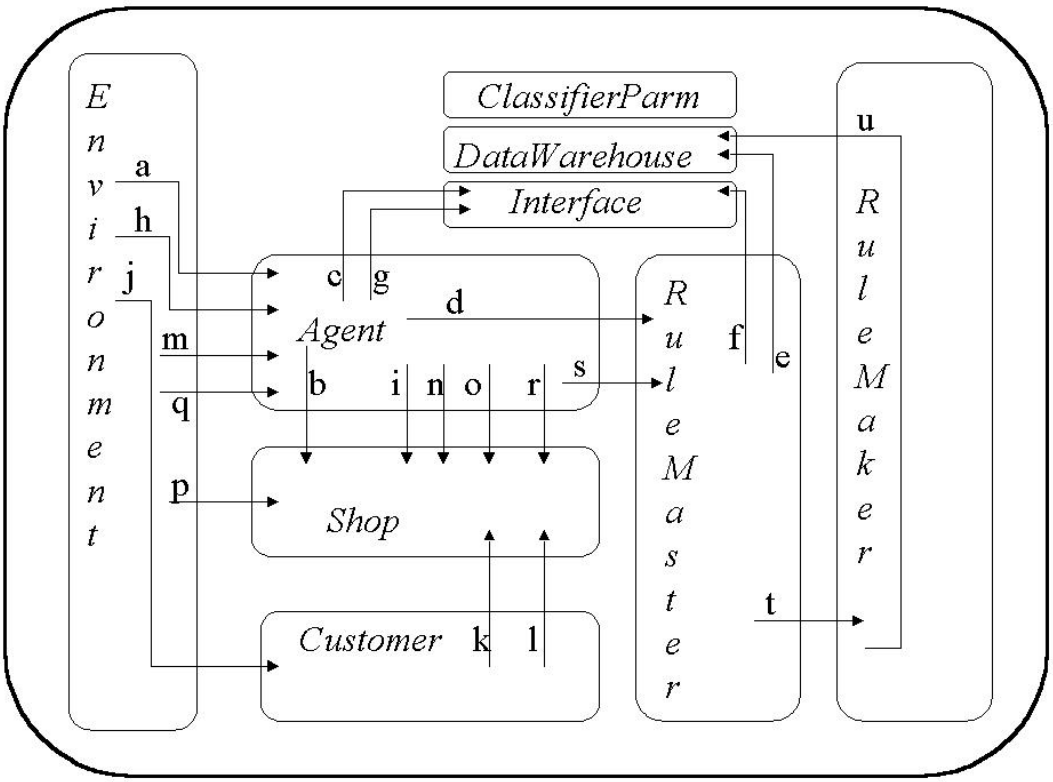
L'esecuzione iterata dei passi descritti permette, ai *classifier system,* di generare e selezionare regole, tali da consentire il raggiungimento di profitti elevati. Il processo deve essere svolto un gran numero di volte: il profitto derivante dall'applicazione di una regola dipende, anche, dalla reazione del concorrente e l'apprendimento, di ciascun agente, è funzione dell'analogo processo del concorrente; la maggior capacità di reazione, acquisita mano a mano da ciascuno, comporta una continua revisione delle regole.

La figura 4.4 permette di esaminare in maggior dettaglio la dinamica del modello: l'azione parte dall'ambiente che ordina (a) a ciascun agente di elaborare una nuova strategia di prezzo. A fronte della richiesta ogni agente richiede (b) a *shop* quale prezzo sta praticando l'altra impresa, ordina (c) alla propria *interface* di tradurre l'informazione nella metrica del *classifier system* ed, infine, richiede (d) al *ruleMaster* quale prezzo dovrà essere adottato nel futuro, indicando gli indirizzi del proprio *dataWarehouse* e della propria *interface*.

Il *ruleMaster* esegue l'elaborazione, già descritta nel precedente capitolo, richiedendo (e) al *dataWarehouse* ed (f) all'*interface*, rispettivamente, i dati relativi alla popolazione di regole da utilizzare e le informazioni ambientali, in questo caso il prezzo del concorrente. Dal *dataWarehouse* il

ruleMaster ottiene anche l'indirizzo dell'oggetto *classifierParm*, dove sono registrati i parametri di funzionamento del sistema classificatore di ciascun agente. Terminata l'elaborazione il *ruleMaster* comunica all'*interface* la propria risposta. Il meccanismo permette di condurre la gestione di vari *classifier system* indipendenti, utilizzando un solo oggetto *ruleMaster*.

Figura 4.4 – Modello di duopolio



Il controllo dell'elaborazione torna, a questo punto, all'agente che richiede (g) all'*interface* di decodificare la risposta ricevuta, traducendola in un valore di prezzo; tale valore viene memorizzato in una variabile dell'agente.

Terminata la fase di elaborazione delle strategie, il *modelSwarm* ordina (h) a ciascun agente di comunicare il proprio prezzo a *shop*. Ogni agente comunica (i) il nuovo prezzo, precedentemente memorizzato, a *shop*.

Il *modelSwarm* prosegue ordinando (j) ai compratori, rappresentati dall'unico oggetto *customer*, di effettuare i propri acquisti. *Customer* richiede (k) a *shop* informazioni relative ai prezzi praticati dalle due imprese e decide, data la funzione di domanda in esso codificata, quali quantità richiedere a ciascuna impresa. Tale decisioni vengono comunicate (l) a *shop,* che provvede a contabilizzare le vendite di ciascuna impresa.

L'azione del modello prosegue con l'ordine (m), inviato dal *modelSwarm* a ciascun agente, di calcolare i profitti; ogni agente richiede (n) l'ammontare delle proprie vendite a *shop*, quindi calcola il profitto conseguito e comunica (o) il dato a *shop* che lo utilizzerà per le computazioni relative al settore.

Shop, ricevuto (p) l'ordine di aggiornare i dati settoriali, provvede al calcolo e ne memorizza i risultati in apposite variabili

Il *modelSwarm*, infine, ordina (q) agli agenti di effettuare la valutazione dei risultati. Ciascun agente confronta l'ammontare dell'ultimo profitto conseguito con la media mobile dei precedenti,

l'ampiezza del periodo considerato dipende da un parametro specificato dall'utilizzatore. Il profitto viene, inoltre, confrontato con il più alto profitto medio di settore, ottenuto interrogando (r) *shop*. Terminata la valutazione, l'agente comunica (s) l'ammontare della ricompensa, da attribuire alla regola attiva, al *ruleMaster*. Questi, dopo le consuete interazioni (e,f) con *interface* e *dataWarehouse*, per stabilire su quale *classifier system* operare, provvede alle necessarie elaborazioni. Immediatamente dopo, in base ai parametri evolutivi utilizzati per quello specifico *classifier system*, decide se richiedere (t) al *ruleMaker* di compiere un ciclo di evoluzione, cioè di produrre nuove regole, comunicando l'indirizzo del *dataWarehouse* da utilizzare. Anche il *ruleMaker*, infatti, interagisce (u) con *dataWarehouse* per identificare la popolazione da evolvere.

L'*observerSwarm* provvede, dopo la creazione del *modelSwarm*, a calcolare i valori da immettere nella bimatrice dei *payoff*, dati i parametri specificati dall'utilizzatore nella *probeMap* del *modelSwarm*; tali valori possono essere stampati, in formato *excel*, a richiesta dell'utilizzatore, su *standard-output*. Durante l'azione del modello, inoltre, l'oggetto produce ed aggiorna due diagrammi rappresentanti, rispettivamente, l'ammontare dei prezzi e dei profitti conseguiti delle due imprese. Ad ogni ciclo viene verificato il raggiungimento dei livelli ottimali, quelli cioè che garantiscono il massimo profitto per le due imprese, a parità di prezzo praticato. L'oggetto è in grado di stampare i dati relativi all'andamento del modello, con la cadenza scelta dall'utilizzatore, ed, una sola volta, al raggiungimento della strategia ottimale; l'utilizzatore può, inoltre, stabilire la frequenza di aggiornamento dei grafici.

4.5.4 - I parametri dell'*observerSwarm*

La *probeMap* dell'*observerSwarm* permette di valorizzare i parametri: *displayFrequency* e *printFrequency*. Il primo stabilisce la frequenza di aggiornamento dei diagrammi relativi a prezzi e profitti: il valore specificato stabilisce quanti cicli del modello devono intercorrere fra due aggiornamenti consecutivi. Il secondo stabilisce ogni quanti cicli devono essere stampati, su *standard-output*, i dati del modello, completati con l'indicazione dei prezzi praticati dai due agenti e del profitto conseguito. Il valore zero inibisce l'esecuzione delle stampe, con qualunque altro valore vengono stampati, in aggiunta ai dati menzionati, i valori della bimatrice dei *payoff* e, quando i due agenti fissano uno stesso prezzo compatibile col raggiungimento del miglior *payoff*, i dati precedentemente menzionati.

4.5.5 - I parametri del *modelSwarm* e dei *classifier system*

L'azione del *modelSwarm* è influenzata dal valore assegnato ai seguenti parametri: *numberOfAgents*, *numberOfCustomers*, *maxPrice*, *substitutionRate*, *directCost*, *fixedCost* e *memory*.

numberOfAgents specifica il numero di imprese che dovranno operare nel modello; in questa versione l'unico valore possibile è due. Il parametro è stato previsto in vista di future implementazioni.

numberOfCustomers permette di stabilire quante istanze della classe *Customer* dovranno agire nel modello. Come illustrato, una sola istanza è sufficiente; un numero più elevato di *customer* viene correttamente gestito e provoca un aumento della domanda e dei profitti. Anche questo parametro è pensato nell'ottica di future implementazioni: immettendo elementi di disturbo, pseudo-casuali, nel comportamento dei *customer* e creandone un numero elevato, sarà possibile simulare situazioni maggiormente realistiche. Dato lo scopo dimostrativo della presente trattazione non si è ritenuto utile procedere in tal senso.

Il valore di *maxPrice* stabilisce, in uno, la massima quantità domandata ed il massimo livello di prezzi praticabile. Se vengono assegnati valori superiori a 15, deve essere aumentato anche il valore di lunghezza del genoma, contenuto nel *file ClassifierParm.dat*, onde garantire che l'informazione, relativa ai prezzi, venga correttamente recepita dal *classifier system*.

substitutionRate specifica il saggio di sostituzione dei beni; il valore assegnabile oscilla, normalmente fra zero ed uno; il modello è in grado di gestire tutti i valori reali, zero compreso, ma i risultati possono essere scarsamente significativi. Un valore negativo nell'intervallo [-1,0[, ad esempio, potrebbe essere utilizzato per studiare il caso di beni complementari, ma ciò invaliderebbe i presupposti di base del modello.

directCost e *fixedCost* permettono, rispettivamente, di stabilire il livello dei costi variabili e dei costi fissi, onde impostare la funzione di costo totale, eguale per ambedue le imprese.

memory consente di specificare il numero di risultati precedenti che gli agenti devono usare nella valutazione del profitto corrente. In pratica ciascun agente confronta l'ultimo profitto conseguito con la media dei profitti precedenti: *memory* specifica il numero di periodi di cui bisogna tenere memoria, quindi il numero di osservazioni che concorrono a determinare il valore della media. Gli agenti sono capaci di ricordare fino ad un massimo di 256 risultati pregressi; questo limite può, però, essere facilmente modificato intervenendo sul codice della classe *Agent*.

I parametri relativi alla gestione dei *classifier system*, già descritti nel capitolo precedente, sono contenuti nel *file ClassifierParm.dat*; i loro valori sono utilizzati per ambedue gli agenti.

Come nel modello precedente, per agevolare la simulazione di situazioni diverse, il *modelSwarm* provvede ad assegnare valori di *default* pseudo-casuali ad alcuni parametri. Eseguendo il modello con l'opzione "-s" tali valori vengono automaticamente variati, nell'ambito degli intervalli, di seguito specificati:

0,1	<	*substitutionRate*	<1	(reale)
1	≤	*directCost*	≤ 5	
0	≤	*fixedCost*	≤ 20	

Questi valori possono, naturalmente, essere modificati dall'utilizzatore, prima di avviare l'esecuzione del modello.

4.5.6 - Caso con saggio di sostituzione unitario

Si parte dal presupposto che i beni siano perfetti sostituti: in questo caso, data l'inclinazione unitaria della curva di domanda, gli agenti potrebbero massimizzare il loro profitto accordandosi per praticare, ambedue, il prezzo più elevato. Come osservato in precedenza: se ambedue le imprese praticano la stessa politica di prezzi, le riduzioni della domanda diretta, conseguenti a ciascun aumento dei prezzi, sono perfettamente compensate da aumenti di quella indiretta.

Come nella formulazione di Bertrand, l'ammontare dei costi fissi è nullo; il valore del costo variabile unitario, presupposto coincidente con quello del costo marginale, è stato fissato a tre. Per valutare i profitti, gli agenti confrontano ciascun risultato con la media dei profitti conseguiti negli ultimi cento periodi.

La tabella 4.5 riporta i valori di *payoff* dei due agenti, calcolati dall'*observerSwarm*. In ogni cella sono indicati i profitti, conseguiti dai due agenti, data l'adozione dei prezzi indicati in testa alla riga ed alla colonna cui la cella appartiene. Il primo valore, *payoff* del giocatore di riga, è il risultato della espressione:

(*maxPrice* - p_r + *substitutionRate* * p_c) * (p_r - *directCost*) - *fixedCost.*

dove p_r rappresenta il prezzo praticato dall'agente di riga e p_c quello praticato dall'agente di colonna; le altre variabili assumono i valori specificati dall'utilizzatore per i parametri omonimi. Il secondo valore è calcolato applicando la formula:

(*maxPrice* - p_c + *substitutionRate* * p_r) * (p_c - *directCost*) - *fixedCost*.

Il massimo profitto compatibile con l'adozione di prezzi eguali, da parte delle due imprese, è riportato in grassetto.

Tabella 4.5 - Bimatrice dei *payoff* con saggio di sostituzione unitario, costi fissi nulli e costi variabili pari a 3.

	0	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**	**11**	**12**	**13**	**14**	**15**
0	-45	-48	-51	-54	-57	-60	-63	-66	-69	-72	-75	-78	-81	-84	-87	-28
	-45	-28	-13	0	11	20	27	32	35	36	35	32	27	20	11	0
1	-28	-30	-32	-34	-36	-38	-40	-42	-44	-46	-48	-50	-52	-54	-56	-13
	-48	-30	-14	0	12	22	30	36	40	42	42	40	36	30	22	12
2	-13	-14	-15	-16	-17	-18	-19	-20	-21	-22	-23	-24	-25	-26	-27	0
	-51	-32	-15	0	13	24	33	40	45	48	49	48	45	40	33	24
3	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	11
	-54	-34	-16	0	14	26	36	44	50	54	56	56	54	50	44	36
4	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	20
	-57	-36	-17	0	15	28	39	48	55	60	63	64	63	60	55	48
5	20	22	24	26	28	30	32	34	36	38	40	42	44	46	48	27
	-60	-38	-18	0	16	30	42	52	60	66	70	72	72	70	66	60
6	27	30	33	36	39	42	45	48	51	54	57	60	63	66	69	32
	-63	-40	-19	0	17	32	45	56	65	72	77	80	81	80	77	72
7	32	36	40	44	48	52	56	60	64	68	72	76	80	84	88	35
	-66	-42	-20	0	18	34	48	60	70	78	84	88	90	90	88	84
8	35	40	45	50	55	60	65	70	75	80	85	90	95	100	105	36
	-69	-44	-21	0	19	36	51	64	75	84	91	96	99	100	99	96
9	36	42	48	54	60	66	72	78	84	90	96	102	108	114	120	35
	-72	-46	-22	0	20	38	54	68	80	90	98	104	108	110	110	108
10	35	42	49	56	63	70	77	84	91	98	105	112	119	126	133	32
	-75	-48	-23	0	21	40	57	72	85	96	105	112	117	120	121	120
11	32	40	48	56	64	72	80	88	96	104	112	120	128	136	144	27
	-78	-50	-24	0	22	42	60	76	90	102	112	120	126	130	132	132
12	27	36	45	54	63	72	81	90	99	108	117	126	135	144	153	20
	-81	-52	-25	0	23	44	63	80	95	108	119	128	135	140	143	144
13	20	30	40	50	60	70	80	90	100	110	120	130	140	150	160	11
	-84	-54	-26	0	24	46	66	84	100	114	126	136	144	150	154	156
14	11	22	33	44	55	66	77	88	99	110	121	132	143	154	165	0
	-87	-56	-27	0	25	48	69	88	105	120	133	144	153	160	165	168
15	0	12	24	36	48	60	72	84	96	108	120	132	144	156	168	**180**
	-28	-13	0	11	20	27	32	35	36	35	32	27	20	11	0	**180**

Durante l'elaborazione l'*observerSwarm* ha prodotto i grafici dei prezzi e dei profitti ed ha stampato, ogni mille cicli, i dati del modello. Una stampa ulteriore è stata eseguita al conseguimento di un accordo sui prezzi da parte degli agenti. La figura 4.5 riporta i grafici e le *probeMap* di *observerSwarm* e *modelSwarm*; la tabella 4.6 contiene i dati stampati dall'*observerSwarm*.

Dai grafici risulta il raggiungimento di una strategia ottimale, con accordo implicito sui prezzi, dopo una fase di concorrenza durata circa 7000 cicli. La crescita tendenziale dei profitti rivela l'esistenza di un processo di apprendimento: i due agenti esprimono strategie migliori, mano a mano che acquisiscono esperienza. La dinamica dei profitti è, in realtà, molto irregolare, ma una

interpolazione lineare porgerebbe una retta con coefficiente angolare positivo; ciò consente di escludere che l'effetto derivi dall'esecuzione di prove casuali: se così fosse, dato l'alto numero di prove, il coefficiente angolare della retta risulterebbe pressoché nullo.

Figura 4.5 - Duopolio tipo Bertrand con saggio di sostituzione unitario

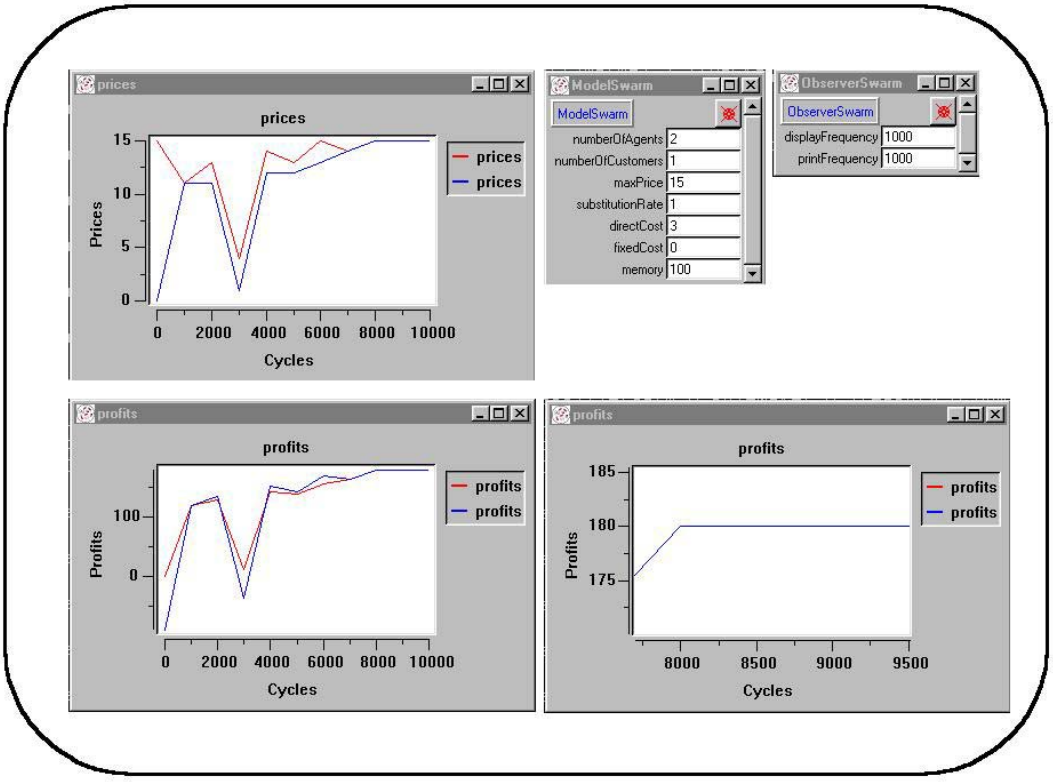


Tabella 4.6 - Duopolio tipo Bertrand con saggio di sostituzione unitario

S	sR	mP	dC	fC	rP	mPr	aN	aP	aPr	delta
1000	1	15	3	0	15	180	0	11	120	60
1000	1	15	3	0	15	180	1	11	120	60
2000	1	15	3	0	15	180	0	13	130	50
2000	1	15	3	0	15	180	1	11	136	44
3000	1	15	3	0	15	180	0	12	162	18
3000	1	15	3	0	15	180	1	15	144	36
4000	1	15	3	0	15	180	0	14	143	37
4000	1	15	3	0	15	180	1	12	153	27
5000	1	15	3	0	15	180	0	13	140	40
5000	1	15	3	0	15	180	1	12	144	36
6000	1	15	3	0	15	180	0	15	156	24
6000	1	15	3	0	15	180	1	13	170	10

S	sR	mP	dC	fC	rP	mPr	aN	aP	aPr	delta
7000	1	15	3	0	15	180	0	14	165	15
7000	1	15	3	0	15	180	1	14	165	15
7106	**1**	**15**	**3**	**0**	**15**	**180**	**0**	**15**	**180**	**0**
7106	**1**	**15**	**3**	**0**	**15**	**180**	**1**	**15**	**180**	**0**
8000	1	15	3	0	15	180	0	15	180	0
8000	1	15	3	0	15	180	1	15	180	0
9000	1	15	3	0	15	180	0	15	180	0
9000	1	15	3	0	15	180	1	15	180	0
10000	1	15	3	0	15	180	0	15	180	0
10000	1	15	3	0	15	180	1	15	180	0

La tabella contiene i valori derivanti dalle osservazioni, effettuate ogni mille cicli; per comodità sono riportati, per ogni riga, anche i parametri del modello, i valori ottimali di prezzo e la differenza fra il profitto massimo conseguibile e quello ottenuto dall'agente in quel ciclo. Più in dettaglio sono indicati: il numero di cicli compiuto, colonna "S", ed i valori di *substitutionRate*, *maxPrice*, *directCost* e *fixedCost*, colonne "sR","mP","dC","fC". Seguono: l'indicazione del valore ottimale di prezzo, colonna "rP", e del massimo profitto conseguibile, colonna "mPr". I dati relativi agli agenti sono: numero dell'agente, colonna "aN", prezzo praticato, colonna "aP", profitto conseguito, colonna "aPr", e differenza fra quel profitto e l'ammontare massimo ottenibile, colonna delta.

I dati di ogni osservazione sono divisi su due righe, una per agente; le righe in grassetto riportano i valori ottenuti al raggiungimento della strategia ottimale.

4.5.7 - Caso con saggio di sostituzione inferiore ad uno

La presente elaborazione è stata effettuata utilizzando gli stessi dati di quella precedente; l'unica variazione riguarda il valore di *substitutionRate*, qui fissato a 0.5. Il valore di equilibrio, calcolato in base ad una funzione continua, è dato da:

$$p_0 = p_1 = (15 + 3) / (2 - 0{,}5) = 12$$

dove p_0 e p_1 rappresentato i prezzi praticati dai due agenti. Il calcolo effettuato con quantità e prezzi discreti, però, fornisce un valore pari a 14, come risulta dalla tabella 4.7.

I dati relativi all'elaborazione sono riepilogati nella tabella 4.8, compilata in base alle segnalazioni prodotte dall'*observerSwarm* durante l'esecuzione del programma.

La figura 4.6 riporta i grafici e le *probe map*, da cui risulta il raggiungimento del livello ottimale dopo circa 3500 cicli. La dinamica di prezzi e profitti rivela come ambedue gli agenti abbiano dovuto accettare riduzioni del proprio livello di profitto, conseguenti al miglioramento della strategia del rispettivo concorrente. L'accordo implicito sul prezzo 13, inoltre, risultando non efficiente, si è rivelato instabile. Come nel caso precedente, la prosecuzione dell'elaborazione, fino ai diecimila cicli, consente di dimostrare la stabilità della situazione ottimale.

Tabella 4.7 - Bimatrice dei *payoff* con saggio di sostituzione pari 0.5, costi fissi nulli e costi variabili pari a 3.

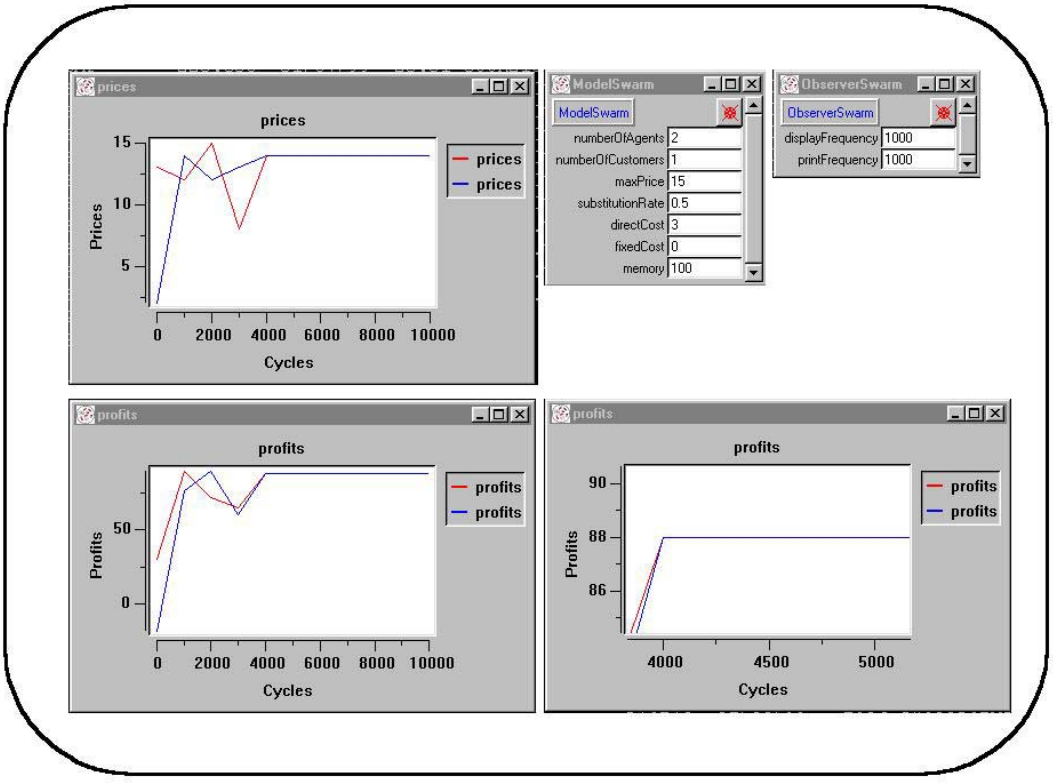
	0	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**	**11**	**12**	**13**	**14**	**15**
0	-45	-45	-48	-48	-51	-51	-54	-54	-57	-57	-60	-60	-63	-63	-66	-28
	-45	-28	-13	0	11	20	27	32	35	36	35	32	27	20	11	0
1	-28	-28	-30	-30	-32	-32	-34	-34	-36	-36	-38	-38	-40	-40	-42	-13
	-45	-28	-13	0	11	20	27	32	35	36	35	32	27	20	11	0
2	-13	-13	-14	-14	-15	-15	-16	-16	-17	-17	-18	-18	-19	-19	-20	0
	-48	-30	-14	0	12	22	30	36	40	42	42	40	36	30	22	12
3	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	11
	-48	-30	-14	0	12	22	30	36	40	42	42	40	36	30	22	12
4	11	11	12	12	13	13	14	14	15	15	16	16	17	17	18	20
	-51	-32	-15	0	13	24	33	40	45	48	49	48	45	40	33	24
5	20	20	22	22	24	24	26	26	28	28	30	30	32	32	34	27
	-51	-32	-15	0	13	24	33	40	45	48	49	48	45	40	33	24
6	27	27	30	30	33	33	36	36	39	39	42	42	45	45	48	32
	-54	-34	-16	0	14	26	36	44	50	54	56	56	54	50	44	36
7	32	32	36	36	40	40	44	44	48	48	52	52	56	56	60	35
	-54	-34	-16	0	14	26	36	44	50	54	56	56	54	50	44	36
8	35	35	40	40	45	45	50	50	55	55	60	60	65	65	70	36
	-57	-36	-17	0	15	28	39	48	55	60	63	64	63	60	55	48
9	36	36	42	42	48	48	54	54	60	60	66	66	72	72	78	35
	-57	-36	-17	0	15	28	39	48	55	60	63	64	63	60	55	48
10	35	35	42	42	49	49	56	56	63	63	70	70	77	77	84	32
	-60	-38	-18	0	16	30	42	52	60	66	70	72	72	70	66	60
11	32	32	40	40	48	48	56	56	64	64	72	72	80	80	88	27
	-60	-38	-18	0	16	30	42	52	60	66	70	72	72	70	66	60
12	27	27	36	36	45	45	54	54	63	63	72	72	81	81	90	20
	-63	-40	-19	0	17	32	45	56	65	72	77	80	81	80	77	72
13	20	20	30	30	40	40	50	50	60	60	70	70	80	80	90	11
	-63	-40	-19	0	17	32	45	56	65	72	77	80	81	80	77	72
14	11	11	22	22	33	33	44	44	55	55	66	66	77	77	**88**	0
	-66	-42	-20	0	18	34	48	60	70	78	84	88	90	90	**88**	84
15	0	0	12	12	24	24	36	36	48	48	60	60	72	72	84	84
	-28	-13	0	11	20	27	32	35	36	35	32	27	20	11	0	84

Tabella 4.8 - Duopolio tipo Bertrand con saggio di sostituzione pari a 0.5

S	**sR**	**mP**	**dC**	**fC**	**rP**	**mPr**	**aN**	**aP**	**aPr**	**delta**
1000	0,5	15	3	0	14	88	0	12	90	-2
1000	0,5	15	3	0	14	88	1	14	77	11
2000	0,5	15	3	0	14	88	0	15	72	16
2000	0,5	15	3	0	14	88	1	12	90	-2
3000	0,5	15	3	0	14	88	0	13	80	8
3000	0,5	15	3	0	14	88	1	13	80	8

S	sR	mP	dC	fC	rP	mPr	aN	aP	aPr	delta
3489	**0,5**	**15**	**3**	**0**	**14**	**88**	**0**	**14**	**88**	**0**
3489	**0,5**	**15**	**3**	**0**	**14**	**88**	**1**	**14**	**88**	**0**
4000	0,5	15	3	0	14	88	0	14	88	0
4000	0,5	15	3	0	14	88	1	14	88	0
5000	0,5	15	3	0	14	88	0	14	88	0
5000	0,5	15	3	0	14	88	1	14	88	0
6000	0,5	15	3	0	14	88	0	14	88	0
6000	0,5	15	3	0	14	88	1	14	88	0
7000	0,5	15	3	0	14	88	0	14	88	0
7000	0,5	15	3	0	14	88	1	14	88	0
8000	0,5	15	3	0	14	88	0	14	88	0
8000	0,5	15	3	0	14	88	1	14	88	0
9000	0,5	15	3	0	14	88	0	14	88	0
9000	0,5	15	3	0	14	88	1	14	88	0
10000	0,5	15	3	0	14	88	0	14	88	0
10000	0,5	15	3	0	14	88	1	14	88	0

Figura 4.6 - Duopolio tipo Bertrand con saggio di sostituzione pari a 0.5



4.5.8 - Risultati di elaborazioni con parametri pseudo-casuali

Sfruttando le possibilità del *modelSwarm*, l'elaborazione del modello è stata ripetuta, per cinquanta volte, con valori casuali di: *substitutionRate*, *directCost* e *fixedCost*. Il seme di generazione dei numeri pseudo-casuali è stato rinnovato prima di ogni elaborazione. I risultati ottenuti sono riepilogati nella tabella 4.9; per semplicità sono riportati solo i dati relativi al ciclo in cui è stato raggiunto il massimo livello di profitto. Le singole elaborazioni sono, comunque, state condotte per un numero di cicli sufficiente a verificare la stabilità dei risultati ottenuti.

Tabella 4.9 - Elaborazioni ripetute del modello di duopolio

N	S	sR	mP	dC	fC	rP	P	aN	aP	aPr	delta
1	2336	0,418481	15	2	9	12	71	0	12	71	0
	2336	0,418481	15	2	9	12	71	1	12	71	0
2	17267	0,59978	15	2	20	14	88	0	14	88	0
	17267	0,59978	15	2	20	14	88	1	14	88	0
3	8362	0,289312	15	5	13	14	32	0	14	32	0
	8362	0,289312	15	5	13	14	32	1	14	32	0
4	13796	0,624834	15	4	6	15	93	0	15	93	0
	13796	0,624834	15	4	6	15	93	1	15	93	0
5	18019	0,266559	15	1	1	12	65	0	12	65	0
	18019	0,266559	15	1	1	12	65	1	12	65	0
6	23452	0,723251	15	2	10	14	122	0	14	122	0
	23452	0,723251	15	2	10	14	122	1	14	122	0
7	3307	0,260904	15	5	20	12	22	0	12	22	0
	3307	0,260904	15	5	20	12	22	1	12	22	0
8	41757	0,58017	15	2	17	14	91	0	14	91	0
	41757	0,58017	15	2	17	14	91	1	14	91	0
9	18943	0,423445	15	4	5	15	61	0	15	61	0
	18943	0,423445	15	4	5	15	61	1	15	61	0
10	8217	0,855128	15	2	7	15	149	0	15	149	0
	8217	0,855128	15	2	7	15	149	1	15	149	0
11	4945	0,358032	15	5	13	14	41	0	14	41	0
	4945	0,358032	15	5	13	14	41	1	14	41	0
12	11973	0,420807	15	3	9	12	63	0	12	63	0
	11973	0,420807	15	3	9	12	63	1	12	63	0
13	4000	0,35582	15	4	12	12	44	0	15	43	1
	4000	0,35582	15	4	12	12	44	1	15	43	1

N	S	sR	mP	dC	fC	rP	P	aN	aP	aPr	delta
14	35142	0,341082	15	1	0	12	77	0	12	77	0
	35142	0,341082	15	1	0	12	77	1	12	77	0
15	21333	0,702508	15	5	18	15	82	0	15	82	0
	21333	0,702508	15	5	18	15	82	1	15	82	0
16	13582	0,45386	15	2	20	14	64	0	14	64	0
	13582	0,45386	15	2	20	14	64	1	14	64	0
17	21870	0,596985	15	2	5	14	103	0	14	103	0
	21870	0,596985	15	2	5	14	103	1	14	103	0
18	7762	0,818459	15	5	15	15	105	0	15	105	0
	7762	0,818459	15	5	15	15	105	1	15	105	0
19	5274	0,157648	15	3	5	9	37	0	10	37	0
	5274	0,157648	15	3	5	9	37	1	10	37	0
20	16859	0,770557	15	3	5	15	127	0	15	127	0
	16859	0,770557	15	3	5	15	127	1	15	127	0
21	477	0,230386	15	2	19	9	37	0	10	37	0
	477	0,230386	15	2	19	9	37	1	9	37	0
22	20483	0,182242	15	2	14	11	40	0	11	40	0
	20483	0,182242	15	2	14	11	40	1	11	40	0
23	13390	0,410103	15	2	14	15	64	0	15	64	0
	13390	0,410103	15	2	14	15	64	1	15	64	0
24	25616	0,397155	15	1	16	13	68	0	13	68	0
	25616	0,397155	15	1	16	13	68	1	13	68	0
25	106	0,828731	15	3	13	15	131	0	15	131	0
	106	0,828731	15	3	13	15	131	1	15	131	0
26	7047	0,870033	15	4	2	15	141	0	15	141	0
	7047	0,870033	15	4	2	15	141	1	15	141	0
27	10057	0,754803	15	1	9	15	145	0	15	145	0
	10057	0,754803	15	1	9	15	145	1	15	145	0
28	2450	0,811334	15	5	10	15	110	0	15	110	0
	2450	0,811334	15	5	10	15	110	1	15	110	0
29	6943	0,550478	15	3	15	15	81	0	15	81	0
	6943	0,550478	15	3	15	15	81	1	15	81	0
30	1657	0,122274	15	3	2	9	40	0	10	40	0

N	S	sR	mP	dC	fC	rP	P	aN	aP	aPr	delta
	1657	0,122274	15	3	2	9	40	1	9	40	0
31	30740	0,873961	15	3	2	15	154	0	15	154	0
	30740	0,873961	15	3	2	15	154	1	15	154	0
32	40793	0,705451	15	2	18	15	112	0	15	112	0
	40793	0,705451	15	2	18	15	112	1	15	112	0
33	61885	0,868839	15	2	7	15	162	0	15	162	0
	61885	0,868839	15	2	7	15	162	1	15	162	0
34	14601	0,315165	15	1	17	10	55	0	13	55	0
	14601	0,315165	15	1	17	10	55	1	13	55	0
35	5720	0,162286	15	1	3	8	53	0	9	53	0
	5720	0,162286	15	1	3	8	53	1	9	53	0
36	4078	0,181073	15	5	5	12	30	0	12	30	0
	4078	0,181073	15	5	5	12	30	1	12	30	0
37	6079	0,114786	15	4	5	10	31	0	10	31	0
	6079	0,114786	15	4	5	10	31	1	10	31	0
38	7778	0,367054	15	3	0	14	66	0	14	66	0
	7778	0,367054	15	3	0	14	66	1	14	66	0
39	949	0,386437	15	1	7	13	77	0	13	77	0
	949	0,386437	15	1	7	13	77	1	13	77	0
40	20161	0,114482	15	1	14	9	42	0	9	42	0
	20161	0,114482	15	1	14	9	42	1	9	42	0
41	39341	0,299349	15	1	4	11	66	0	11	66	0
	39341	0,299349	15	1	4	11	66	1	11	66	0
42	10897	0,548745	15	5	16	15	64	0	15	64	0
	10897	0,548745	15	5	16	15	64	1	15	64	0
43	11558	0,338703	15	4	16	12	40	0	12	40	0
	11558	0,338703	15	4	16	12	40	1	12	40	0
44	712	0,227379	15	2	12	9	44	0	10	44	0
	712	0,227379	15	2	12	9	44	1	10	44	0
45	29058	0,8933	15	4	1	15	142	0	15	142	0
	29058	0,8933	15	4	1	15	142	1	15	142	0
46	8758	0,77114	15	5	9	15	101	0	15	101	0
	8758	0,77114	15	5	9	15	101	1	15	101	0

N	S	sR	mP	dC	fC	rP	P	aN	aPr	aP	delta
47	164	0,970015	15	3	18	15	150	0	15	150	0
	164	0,970015	15	3	18	15	150	1	15	150	0
48	2759	0,882497	15	2	3	15	166	0	15	166	0
	2759	0,882497	15	2	3	15	166	1	15	166	0
49	30861	0,708482	15	4	3	15	107	0	15	107	0
	30861	0,708482	15	4	3	15	107	1	15	107	0
50	30811	0,877432	15	3	10	15	146	0	15	146	0
	30811	0,877432	15	3	10	15	146	1	15	146	0

Dalla disamina della tabella, risulta che solo in un caso, il numero 13, gli agenti non sono stati in grado di raggiungere il massimo livello di profitti; la differenza risulta comunque lieve. Al raggiungimento di elevati risultati economici, si è accompagnato un accordo implicito sui prezzi 47 casi su 50, 48 se si conta anche il caso tredici. I due casi di discordanza, numero 21 e numero 30, sono, probabilmente, dovuti al fatto che l'adozione di prezzi diversi risultava ininfluente dal punto di vista dei profitti; simili situazioni sono imputabili all'adozione di piccoli intervalli di escursione delle quantità: il meccanismo di arrotondamento, in questi casi, può rendere indifferente l'applicazione di prezzi moderatamente diversi. Fenomeno analogo può spiegare i casi in cui sia presente una differenza fra il prezzo ottimale, calcolato dall'*observerSwarm*, e quello praticato dagli agenti; l'anomalia è rilevabile a proposito dei casi numero: 19, 21, 30, 34, 35 e 44. Al proposito occorre tener conto che la ricerca condotta dall'*observerSwarm* è basata sulla enumerazione delle combinazioni possibili: l'enumerazione viene effettuata partendo dai valori di prezzo più bassi e, in seguito, viene individuato, quale prezzo ottimale, il primo che garantisce il raggiungimento del massimo profitto. In questo modo, in presenza di più valori di prezzo in grado di assicurare lo stesso livello di profitti, l'*observerSwarm* considera come ottimale il più basso.

Alla luce delle osservazioni svolte, è possibile formulare un positivo giudizio sulla capacità del modello di ottimizzare il livello di profitto nelle diverse situazioni considerate. Risulta, quindi, dimostrata l'efficacia del metodo dei *classifier system*: gli agenti hanno imparato ad attuare la strategia migliore, pur partendo da una situazione di assoluta ignoranza delle caratteristiche ambientali.

Per quanto concerne l'aspetto economico, si può notare come l'accordo sui prezzi sia risultato emergere dal perseguimento dell'obiettivo di massimo profitto; tale effetto, inoltre, è stato ottenuto a prescindere dalla precisa conoscenza della curva di domanda, né può dirsi scaturire da comportamenti semplicemente imitativi. Le imprese, infatti, oltre a non conoscere inclinazione della curva e saggio di sostituzione, disponevano, durante le elaborazioni, di informazioni paritetiche e, oltremodo, limitate: la sola conoscenza del prezzo praticato, fino a quel momento, dal concorrente e l'andamento dei migliori profitti settoriali.

L'attività di continua ricerca del livello ottimale di prezzi, esperita da ciascun agente, per il tramite del proprio *classifier system*, non si discosta in modo sostanziale dal comportamento di una impresa che fosse soggetta agli stessi vincoli di informazione. Un modello maggiormente plausibile potrebbe, per altro, considerare tutte le politiche accessorie e complementari alla pura strategia di *pricing*; esistono, perciò, possibilità per futuri lavori.

5 – Conclusioni

A conclusione del lavoro, si presentano brevi considerazioni sui risultati ottenuti, accompagnate dall'illustrazione di alcuni spunti per futuri sviluppi della ricerca.

Il commento dei risultati è organizzato in tre paragrafi dedicati, rispettivamente: alla computazione automatica, con particolare riferimento alle simulazioni basate su modelli con agenti, ai metodi evolutivi, con richiami ai programmi realizzati per il presente lavoro, ed al significato economico dei risultati ottenuti con i due modelli di simulazione di situazioni di mercato.

Gli ultimi due paragrafi sono dedicati all'illustrazione delle possibilità di approfondimento della ricerca: il primo tratta del progetto di unificazione dei due pacchetti realizzati, con ampliamento delle loro potenzialità, nel secondo viene fornita una sintetica illustrazione degli algoritmi genetici paralleli, interessante paradigma per il potenziamento del metodo.

5.1 – Utilità della computazione automatica

Le sperimentazioni condotte hanno permesso di verificare la possibilità di riprodurre, senza soverchie difficoltà, situazioni di mercato, in simulazioni caratterizzate da un buon grado di realismo. L'impiego di agenti artificiali, dotati di elevata autonomia decisionale e di razionalità limitata, ha contribuito a determinare un elevato grado di plausibilità dei comportamenti simulati.

Dall'azione dei modelli sono emersi risultati globali in linea con gli assunti teorici di base, pur in assenza delle stringenti assunzioni, relative al comportamento degli agenti, che caratterizzano le impostazioni tradizionali. L'osservazione della dinamica di azione dei modelli ha contribuito a spiegare l'emergere, nella realtà, di comportamenti aggregati complessi, a prescindere dall'esistenza di una teleologia specifica a livello di sistema. Nei due modelli economici presentati nel capitolo quarto, ad esempio, l'azione di agenti, impegnati unicamente nella massimizzazione dei propri profitti, ha prodotto l'affermazione di livelli comuni di prezzo. L'ammontare dei prezzi è risultato dipendere unicamente dalle caratteristiche delle situazioni riprodotte, come dimostrato dall'ottenimento di effetti differenti, per i vari casi studiati, con l'impiego di agenti perfettamente eguali. La modificazione dei comportamenti, scaturita dalla variazione dei parametri della simulazione, dimostra l'efficacia del metodo nella riproduzione della complessità di sistemi costituiti da entità in interazione.

L'esperienza condotta ha consentito di verificare la correttezza dei risultati ottenibili e fornisce, perciò, valida motivazione per l'impiego di procedimenti computazionali nell'esplorazione di sistemi poco conosciuti o nella ricerca di spiegazioni alternative di fenomeni noti. I modelli utilizzati nel presente lavoro costituiscono un esempio, volutamente semplice, della ricerca di procedimenti, diversi dai modelli matematici, per fornire descrizioni quantitative dei fenomeni indagati. Essi risultano basati sull'utilizzazione congiunta dei sistemi tradizionali e computazionali descritta, a proposito dei ruoli reciproci dei due paradigmi, in Judd (1996):

> (…) In some activities, they are clearly complements with their complementary strengths and weaknesses indicating that they can be very successfull as partners. Deductive theory is necessary in reducing an economic question to a finite set of mathematical expressions which a computer can then analyze to produce economically useful results.

Quanto sopra non esclude che i metodi computazionali possano, comunque, produrre risultati propri, a prescindere dall'apporto di precedenti studi condotti in modo tradizionale; in molti casi, essi permettono di individuare spiegazioni difficilmente rilevabili con metodi puramente deduttivi. Nelle simulazioni effettuate, ad esempio, l'osservazione del modello ha consentito di cogliere aspetti

relativamente nuovi od emergenti, descritti, più avanti, nella trattazione dei significati economici dei risultati ottenuti dai modelli.

A questa peculiarità i metodi computazionali aggiungono, di frequente, un minor costo della ricerca: mentre i procedimenti tradizionali possono richiedere un elevato impiego di risorse umane, quelli computazionali sono basati sul prevalente utilizzo di macchine, molto meno costose.

Simili osservazioni sono rinvenibili anche in Judd (1996):

> (…) computation can also be, and will sometimes be, a substitute for deductive theory. First, computation can inform us of patterns which analytical theory would have great difficulty discerning or expressing. Second, it may be cheaper to use computationally intensive methods instead of theorem-proving to analyze a theory.

L'autore citato giunge a concludere nei termini seguenti:

> (…) Computation cannot achieve its potential without the use of theory, and theory will become increasingly dipendent on computation to answer theoretical questions and guide it in directions of greatest economic value.

Pur non essendo la valutazione dello schema ERA fra gli scopi della presente ricerca, è doveroso riscontrare il contributo alla chiarezza e immediata comprensibilità del modello apportato dall'adozione di quel paradigma; la struttura derivante ha, inoltre, offerto notevoli vantaggi nelle fasi di sviluppo del *software*, rivelando l'estesa validità dello schema.

5.2 - Considerazioni sull'impiego dei metodi evolutivi

I modelli realizzati, sia quelli generali proposti nel capitolo terzo sia quelli economici del quarto capitolo, hanno rivelato buone possibilità di sfruttamento del paradigma evolutivo nella gestione dell'apprendimento dei singoli agenti.

I *Classifier system* hanno manifestato doti di robustezza ed elevata duttilità. Il metodo si è dimostrato in grado di evolvere, contemporaneamente, regole atte a fronteggiare situazioni ambientali diverse, coniugando ragguardevole potenza ed elevata precisione di azione; queste caratteristiche tendono, purtroppo, ad accompagnarsi ad una immediatezza di utilizzo non sempre elevata.

L'impiego del metodo richiede un maggior livello di attenzione, rispetto a quello di semplici algoritmi genetici, nella valorizzazione dei parametri di conduzione e nel fornire valutazioni dei risultati particolarmente curate. Il semplice utilizzo dei valori zero ed uno, ad indicare successi, o fallimenti, per l'esperimento basato sulle formiche di Langton, è risultato insufficiente nel caso del duopolio di Bertrand; in quel modello la funzione ha dovuto essere modificata con l'inserimento di valori che permettessero di meglio differenziare i risultati. I *classifier system* possono essere utilizzati anche in assenza di informazione sugli aspetti specifici del problema avendo, però, cura di porre particolare attenzione nel fornire valutazioni dei risultati sufficientemente specifiche.

L'esperimento, incluso nel *workbench* di CW, ha consentito di apprezzare il funzionamento dei *classifier system* nella ricerca di risposte a stimoli ambientali di varia complessità; dalle verificazioni, sono emerse utili congetture sulla valorizzazione dei parametri che ne hanno rivelato l'importanza, specialmente quando si vogliano ottenere prestazioni efficienti oltre che efficaci. Nei vari esempi è illustrata la possibilità di ottenere un completo "apprendimento" anche con valori non ottimali dei parametri; la codificazione di valori più adatti ha consentito di ridurre il numero di cicli di elaborazione necessari.

Il modello del "formichiere di Ferraris" ha consentito la verificazione delle capacità di inferire le regole di comportamento di altre entità simulate: il *classifier*, che guidava le azioni del

formichiere, è stato in grado di riprodurre perfettamente le regole del sistema esperto, che condizionava il comportamento delle varie formiche.

In ambedue i casi, inoltre, si è assodato che i *classifier system* sono in grado di adeguare il proprio livello di apprendimento, incorporando le istanze emergenti dalle modificazioni dell'ambiente: regole obsolete, o comunque non più efficaci, sono state agevolmente sostituite da nuove istanze, prodotte dall'algoritmo evolutivo, mantenendo elevato il livello dei risultati. Questa capacità permette di ottenere buoni risultati anche in ambiti fortemente instabili: nel modello basato sul duopolio di Bertrand, ad esempio, due *classifier system* hanno operato condizionando reciprocamente i comportamenti dell'opponente; le regole di ciascuno erano, perciò, soggette ad una rapida obsolescenza, legata ai progressi ottenuti dal sistema avversario. In quel modello ogni *classifier system* è stato costretto a modificare continuamente le proprie regole, fino al conseguimento di una situazione che risultasse simultaneamente soddisfacente per ambedue. Nonostante le difficoltà enunciate, nell'elaborazione di cinquanta esempi diversi di duopolio di Bertrand, i *classifier system* hanno conseguito il massimo livello di risultati per ben quarantotto volte, sperimentando, inoltre, nelle due volte residue, "scostamenti" piuttosto ridotti.

Gli algoritmi genetici si sono qualificati come buon metodo di ricerca di soluzioni ottime, o significativamente prossime all'ottimo, in spazi estesi e, soprattutto, in contesti del tutto sconosciuti. Nelle ricerche condotte con il modello di simulazione di mercati, il metodo si è dimostrato decisamente migliore delle tecniche di ricerca tradizionali: risultati ottimali sono stati raggiunti in un numero di cicli mediamente inferiore al numero di passi richiesti da una procedura di ricerca in ampiezza. Il margine di errore, emerso nella trattazione della funzione utilizzata in Freeman (1994), è risultato adeguato alla difficoltà connessa al problema: il modello era stato volutamente concepito come caso estremo. Nel modello di simulazione di mercati, l'algoritmo genetico ha rivelato una affidabilità elevata accompagnata da una capacità considerevole di adattamento al contesto. In ambedue le simulazioni, infine, il metodo ha dimostrato la propria semplicità di utilizzo: l'unico dato messo a disposizione dell'algoritmo è stata la misura dei risultati ottenuti, consistente nel semplice calcolo della funzione nel punto indicato o nell'ammontare del profitto realizzato, data l'applicazione del prezzo consigliato.

I due pacchetti *software*, CW e GM, si sono dimostrati in grado di fornire buone prestazioni, adattandosi facilmente ad impieghi fortemente diversificati: tutti i modelli presentati sono stati realizzati a partire da uno dei due pacchetti, senza mai dover procedere a modificazioni del *kernel*. CW è stato in grado di condurre, contemporaneamente e senza problemi: nel quarto esempio di utilizzo del *workbench*, più *classifier system*, nell'esperimento del formichiere, un sistema esperto ed un *classifier system*, e, nell'esperimento del duopolio di Bertrand, due *classifier system* che influenzavano reciprocamente i loro risultati. In GM l'inserimento della gestione differenziale della *fitness* ha permesso di ottenere risultati di elevata precisione; l'accorgimento, dando facoltà di assegnare valori negativi di *fitness*, ha consentito, inoltre, di semplificare notevolmente le funzioni di valutazione degli individui, rendendo maggiormente immediato l'utilizzo del metodo. L'utilità del sistema di gestione differenziale della *fitness* è emersa, particolarmente, nella comparazione con il metodo tradizionale, svolta a proposito della ricerca del massimo per la funzione utilizzata in Freeman (1994).

5.3 – Interpretazione economica dei risultati ottenuti con i modelli

Le due simulazioni presentate nel capitolo quattro, pur avendo per principale obiettivo la verificazione della correttezza dei risultati forniti dai metodi evolutivi, hanno permesso di cogliere effetti di un certo interesse per lo studio della dinamica dei mercati.

L'affermarsi di livelli di prezzo di tipo concorrenziale, nel modello di simulazione di mercati, permette di evincere come la differenziazione territoriale non consenta una efficace segmentazione del mercato. Nel modello ogni compratore poteva rivolgersi solo ad un numero ristretto di imprese, considerate come localizzate entro determinate distanze dal cliente; la limitazione porterebbe quindi a

non escludere la possibilità dell'affermarsi di prezzi diversi nelle varie aree. Dall'azione del modello è emersa l'importanza del comportamento dei compratori siti ai confini di ciascun intorno; nel raggio di azione di questi ultimi cadevano aziende appartenenti, sistematicamente, a diverse aree: le scelte effettuate da questi compratori hanno comportato l'estensione della concorrenza fra le imprese al di fuori dei confini delle singole aree, determinando la diffusione di un unico livello di prezzo.

La simulazione del duopolio di Bertrand ha consentito di cogliere un meccanismo di formazione di accordi impliciti sui prezzi: le due aziende, dopo una fase di sperimentazione di strategie diverse, hanno sistematicamente trovato, nell'adozione di un prezzo comune, il modo di ottenere il massimo profitto compatibile con le condizioni del mercato.

In ambedue i modelli la ricerca dell'accordo sui prezzi non era contemplata come obiettivo per gli agenti che, invece, erano impegnati unicamente nella ricerca del massimo profitto. La convergenza dei vari produttori su strategie comuni di *pricing* è, perciò, risultata emergere quale comportamento spontaneo, ed accessorio alla massimizzazione dei profitti; tale effetto si è manifestato pur in assenza di entità incaricate di equilibrare il mercato.

5.4 – Note sul *software* realizzato e sui possibili sviluppi futuri

I buoni risultati ottenuti con l'utilizzo dei pacchetti GM e CW motivano la previsione di ulteriori sviluppi. CW e GM si sono dimostrati utili nella conduzione di simulazioni molto diverse tra loro: il loro impiego ha permesso di conferire un buon livello di realismo ai modelli, pur senza richiedere la codificazione di programmi specifici.

CW include un algoritmo genetico del tipo di quello realizzato con GM; la codificazione di due prodotti separati ha consentito di sperimentare conduzioni diverse dell'algoritmo genetico: quello di CW è basato sul sistema "*crowding*" come in Goldberg (1989), mentre quello di GM è stato ispirato dalla trattazione di Holland (1975). In GM, inoltre, è stato aggiunto il procedimento di selezione basato sulle differenze di *fitness*.

Si intende, nel prossimo futuro, riunire i due pacchetti in un unico prodotto che assumerà la denominazione di *General Genetic Instrument*, in sigla GGI. La struttura del pacchetto, sviluppata nel rispetto dello schema ERA, si baserà su di un *kernel* composto dalla coppia *ruleMaster* e *ruleMaker*: il primo ospiterà le componenti di selezione delle regole e di attribuzione dei crediti, tipiche di un *classifier system*, il secondo sarà deputato alla gestione di un algoritmo genetico. L'elemento di novità sarà costituito dalla possibilità di indirizzare, al *ruleMaster,* richieste volte ad ottenere azioni diverse sulle popolazioni di cromosomi, contenute in oggetti della classe *dataWarehouse,* propri di ciascun agente, o di gruppi di agenti. In questo modo un solo oggetto *ruleMaster*, eventualmente accompagnato da un singolo *ruleMaker*, sarà sufficiente per la conduzione, simultanea, di elaborazioni relative a molteplici: sistemi esperti tradizionali, *non learning classifier system*, *learning classifier system* ed algoritmi genetici.

Gli agenti fruitori dei servizi di GGI potranno scegliere dinamicamente, anche durante l'elaborazione, quale procedimento applicare agli oggetti contenuti nei propri *dataWarehouse*, inviando gli opportuni messaggi al *ruleMaster*. Questi effettuerà azioni diverse, a seconda del tipo di servizi richiesto. La conduzione di un sistema esperto verrà realizzata attivando il solo algoritmo di selezione delle regole, in base al *match* con le condizioni ambientali. Le operazioni tipiche di un algoritmo genetico verranno effettuate richiamando direttamente il *ruleMaker*, in questo caso il *ruleMaster* agirà esclusivamente da intermediario o, al più, verificherà il raggiungimento del grado di omogeneità della popolazione. La conduzione dei *classifier system* potrà avvenire in modo "*learning*", attivando, se del caso, il *ruleMaker*; per i *non learning classifier* il *ruleMaster* non si avvarrà della collaborazione del *ruleMaker*.

I vantaggi di una simile architettura sono riassumibili nei seguenti punti:

- Più elevata versatilità degli agenti: i programmi potranno utilizzare paradigmi inferenziali diversi con modificazioni minime.
- Maggior dinamicità del comportamento degli agenti: sarà possibile dotare gli agenti di meta-regole, di scelta del procedimento di ricerca adatto alle singole decisioni.
- Possibilità di utilizzare uno dei quattro paradigmi per controllare le decisioni in merito all'utilizzazione degli altri.
- Possibilità di impiegare più procedimenti per la stessa decisione, confrontando, in seguito i risultati.
- Maggior uniformità dei modelli, basati sugli stessi programmi di gestione e produzione di regole, pur utilizzando paradigmi evolutivi diversi.
- Possibilità di sfruttare gli algoritmi genetici nell'ambito dell'impostazione mentale senza modificazioni del *software*.
- Possibilità di delegare al *ruleMaster* le verificazioni in ordine al grado di omogeneità raggiunto dalla popolazione di un algoritmo genetico.
- Possibilità di sfruttare le varie metodologie di selezione presentate, sistema *crowding*, metodo tradizionale basato sul valore della *fitness*, metodo basato sulle differenze di *fitness*, sia per i *classifier system*, sia per gli algoritmi genetici.

Nella predisposizione del nuovo pacchetto, si intende, infine, porre particolare attenzione alla ricerca di maggior efficienza, mettendo in atto tutte le semplificazioni e gli affinamenti dei programmi che l'esperienza maturata potrà consigliare. A tal fine si intende valutare l'opportunità di modificare, radicalmente, i programmi di gestione, onde sfruttare le possibilità offerte dalle metodologie "parallele", brevemente illustrate nel successivo paragrafo.

5.5 - Algoritmi genetici paralleli

Le metodologie genetiche si prestano particolarmente bene a sviluppi paralleli: esse operano su insiemi di individui agendo, però, in modo indipendente sulle singole entità. Quanto si intende presentare non va, però, inteso come semplice conduzione parallela di diverse attività, bensì come tentativo di ottenere sistemi organizzati, formati da molteplici algoritmi genetici, il cui comportamento risulti migliore della somma dei comportamenti espressi dai singoli algoritmi che li compongono.

Operare in questa direzione costituisce una stimolante opportunità di approfondimento della ricerca appena presentata; tale sviluppo permetterebbe di ottenere notevoli miglioramenti prestazionali del metodo, ben accompagnandosi ai futuri obiettivi trattati nel punto precedente.

L'innovazione di metodo è trattata in Alba e Troya (1999), dove sono riprese, con maggior dettaglio, anche alcune possibilità già nominate in Goldberg (1989). L'attributo "paralleli" può sottendere due aspetti diversi riguardanti: la conduzione dell'elaborazione o le caratteristiche della medesima.

Nel primo caso i vantaggi deriverebbero dalla possibilità di ripartire l'onere della computazione su più processori o su più elaboratori che opererebbero, appunto, in "parallelo". La possibilità di effettuare elaborazioni di questo tipo, sfruttando contemporaneamente più unità di computazione, è strettamente legata alle caratteristiche dell'*hardware* e dei linguaggi impiegati, quindi non costituisce elemento di particolare interesse per questa trattazione. La ripartizione dell'onere computazionale fra più elaboratori, inoltre, può essere realizzata attraverso tecniche, abbastanza semplici, di comunicazione; tali procedimenti risultano convenienti solo in presenza di un alto rapporto fra l'onere della computazione e quello della comunicazione; in caso contrario i benefici della ripartizione del carico verrebbero compensati, quando non superati, dagli oneri della comunicazione. L'applicazione di questi metodi sarebbe utile per trattare, ad esempio, funzioni, di valutazione dei singoli individui, particolarmente complesse, richiedenti un rilevante impiego di risorse computazionali.

Il maggior interesse va verso le tecniche che afferiscono le caratteristiche del metodo. Queste si basano, fondamentalmente, sulla individuazione di aree dove avvengono evoluzioni indipendenti; le aree possono comunicare tra loro per il tramite dello scambio di individui "migranti". Anche in natura gli accoppiamenti, all'interno della specie, tenderebbero a risentire di limitazioni dovute all'appartenenza a gruppi diversi e a problemi di distanza, quindi il criterio di isolamento risulterebbe plausibile. Allo stesso modo la funzione di divulgazione culturale, ascrivibile ai movimenti migratori ed ai contatti fra gruppi, renderebbe corretto l'interscambio di individui. Diviene basilare inserire nell'algoritmo una "geografia" che consenta di individuare precisamente le varie aree e regolamentare i flussi migratori.

A proposito della plausibilità del metodo, particolarmente con riferimento all'imitazione della selezione naturale, si legge in Alba e Troya (1999):

> If we mimic natural evolution, we would not operate on a single population in which a given individual has the potential to mate with any other partner in the entire population (panmixia). Instead, species would tend to reproduce within subgroups or within neighborhoods. A large population ditributed among a number of semi-isolated breeding groups is known as polytipic. A PGA introduces the concept of interconnected demes. The local selection and reproduction rules allow the species to evolve locally, and diversity is enhanced by migrations of strings among demes.

Gli algoritmi di questo tipo sono distinti in base alla maggior importanza della computazione o della comunicazione: si definiscono *coars-grain algorithm* quelli per cui è più importante la computazione e *fine-grain algorithm* gli altri. I primi vengono anche denominati *distribuited Genetic Algorithm* (DGA), mentre i secondi sono chiamati *cellular Genetic Algorithm*(CGA).

Un DGA consta di una disposizione di gruppi di individui: ad anello, ipercubo od altra. L'evoluzione della popolazione viene condotta, in autonomia, all'interno di ciascun gruppo; fatti migratori, più o meno ricorrenti, contribuiscono a movimentare il sistema globale, prevenendo il verificarsi di convergenze eccessivamente rapide. La struttura si adatta bene anche all'utilizzazione di una rete di elaboratori, dove le singole macchine operino su uno o più gruppi e le migrazioni avvengano via rete.

I *cellular Genetic Algorithm* operano su una griglia toroidale di individui, ciascuno dei quali interagisce solo con i propri vicini. L'evoluzione viene condotta rimpiazzando gli individui prescelti, con altri ottenuti dall'incrocio con i propri vicini: la comunicazione fra loci diviene molto importante. In un DGA ciascun gruppo conterrebbe diversi individui mentre in un CGA ogni individuo farebbe gruppo a sé.

La distinzione potrebbe essere basata su molte altre caratteristiche essendo, in realtà, i due tipi soltanto gli estremi di un continuo di soluzioni, proponibili tramite diverse combinazioni delle tecniche descritte. Strutture complesse, capaci però di buone prestazioni, potrebbero ottenersi dalla combinazione di più DGA e CGA, dando luogo a disposizioni articolate in diversi livelli.

La computazione di un algoritmo genetico tradizionale può essere scomposta in tre fasi: generazione iniziale di una popolazione, ottimizzazione della stessa e verificazione dei criteri di terminazione del processo. Negli algoritmi genetici distribuiti queste fasi sono condotte da singoli processi, agenti in parallelo, ognuno dei quali include una quarta fase di "comunicazione", caratterizzata da cadenze periodiche di espletamento, nei confronti di un insieme di "vicini" data una determinata topologia. La comunicazione consta, normalmente, dello scambio di gruppi di individui, pur nulla vietando che oggetto di essa siano altre informazioni come, ad esempio, statistiche relative all'evoluzione. I singoli processi operano, usualmente, in base ad uno stesso piano riproduttivo, pur se algoritmi fortemente eterogenei potrebbero rappresentare interessanti sviluppi.

In un algoritmo genetico cellulare ogni individuo interagisce con i propri vicini posti ai quattro punti cardinali: su ciascun gruppo, di cinque individui, viene eseguito lo stesso tipo di evoluzione ed ogni nuovo cromosoma sostituisce quello selezionato solo se migliore. Ogni gruppo è ottenuto, data una lista degli individui, prendendo quelli che occupano le cinque posizioni contigue a

partire da una assegnata; incrementando di un'unità la posizione di partenza, si ottengono i diversi gruppi, in numero pari a quello degli individui che compongono la popolazione.

Le risultanze del fondamentale "teorema dello schema", sulle possibilità di sopravvivenza di uno schema all'interno di una popolazione, sono generalmente confermante anche in riguardo agli algoritmi genetici paralleli.

L'evoluzione può essere condotta, in ciascun gruppo, seguendo criteri di *generational replacement*, cioè creando una nuova popolazione ad ogni ciclo, o in base ad una *steady-state evolution*, ottenendo un solo nuovo individuo per ogni ciclo, che sarà inserito nella popolazione in base alla politica di rinnovamento adottata. Nel primo caso l'algoritmo risulta più lento, la diversità viene rapidamente a decadere ed i nuovi individui sono utilizzabili solo al termine dell'intero processo. Con le tecniche di *steady-state* si ottengono procedimenti fino a dieci volte più rapidi e decisamente più efficaci; i nuovi nati, convivendo con i genitori, sono immediatamente disponibili per ulteriori operazioni.

Nella selezione si sperimenta un *trade-off* tra rapidità di convergenza e probabilità di ottenere una soluzione altamente ottimale. I due tipi fondamentali sono quella proporzionale, basata sulla *fitness* di ciascun individuo, e quella attinente la posizione di ciascun individuo in una popolazione ordinata. Politiche di sostituzione diverse possono essere adottate per rimpiazzare i vecchi individui con i nuovi, creati nell'ambito del gruppo o immigrati da altri. Esempi di queste politiche sono riconducibili ai paradigmi della *elitist generational replacement* o dello *extinctive replacement*.

A proposito del *crossover*, si distinguono procedimenti di *pure crossover*, intendendo il metodo normalmente usato, e di *hybrid crossover* e *problem dependent operators,* capaci di considerare conoscenze precedentemente acquisite, in ordine al problema trattato. Potrebbero essere anche utilizzati metodi basati su incroci multipli, ottenuti scambiando più sezioni dei due cromosomi “genitori”.

La mutazione, normalmente utilizzata per aggiungere "movimento" ed impedire la perdita definitiva di materiale genetico, risulterebbe, in un algoritmo parallelo, agire in rinforzo della diversificazione, ottenuta tramite la migrazione.

Con i metodi paralleli si utilizza, inoltre, la migrazione: essa è intesa come l'operatore che guida lo scambio di individui fra gruppi. L'azione dell'operatore dipende dal *migration gap* e dalla *migration rate*. Il primo valore rappresenta il numero di passi fra due migrazioni consecutive, il secondo la quota della popolazione coinvolta nell'operazione.

L'ammontare del *migration gap* può essere determinato in modo rigido o stocastico, data, cioè, una probabilità media di migrazione. Un ulteriore criterio è quello del *sigma-exchange algorithm,* secondo il quale la migrazione avverrebbe solo quando la deviazione standard, della *fitness* di ciascun gruppo, raggiungesse un determinato valore di soglia, solitamente piuttosto basso. L'adozione di quest'ultimo criterio tende a ridurre considerevolmente il numero di migrazioni.

La quota della popolazione soggetta a migrazione, *migration rate*, può essere espressa in percentuale o fissata ad un numero dato di individui; pare logico che, comunque, il numero di individui coinvolti debba essere piuttosto basso, ad esempio non superiore al 10% dell'ammontare della popolazione. La *migration rate* parrebbe essere strettamente correlata al tipo di problema, per cui il suo valore dovrebbe essere soggetto a sperimentazione per singoli casi. Utilizzando gruppi di numerosità eterogenea, è ragionevole ipotizzare un dimensionamento del fenomeno migratorio per numero di individui. Operando in base a quote proporzionali della popolazione di partenza, i movimenti da grossi gruppi potrebbero implicare influenze eccessive se avvenissero verso gruppi più piccoli, concretizzando vere e proprie colonizzazioni.

Emigrazione ed immigrazione sono normalmente condotte in modo sincrono, cioè le due operazioni vengono effettuate nello stesso ciclo di elaborazione; maggior efficienza risulterebbe caratterizzare migrazioni asincrone. Processi di quel tipo potrebbero basarsi sulla disponibilità a ricevere nuovi individui in qualunque passo, in modo che la decisione di emigrazione divenisse del

tutto indipendente per ciascun gruppo. Con migrazioni asincrone, potrebbe verificarsi l'immissione di individui caratterizzati da livelli di evoluzione diversi da quello medio della popolazione ricevente; ciò comporterebbe due tipi di problemi: *non-effect problem*, dato dall'inserimento di individui meno sviluppati della media, e *super individual problem*, nel caso opposto.

La selezione degli individui da sottoporre a migrazione può essere condotta con criteri analoghi a quelli per la riproduzione. Si noti, però, che l'invio dei migliori individui dovrebbe accompagnarsi con elevati valori del *migration gap*, cioè con migrazioni piuttosto rare, onde prevenire una prematura convergenza della popolazione.

Nella realizzazione di questo tipo di algoritmi genetici assume particolare importanza la topologia adottata. I migliori modelli appaiono essere gerarchici e basati su anelli o ipercubi; strutture maggiormente connesse tendono ad avere una comunicazione troppo stretta. Con riferimento a questi aspetti è d'uso distinguere algoritmi *stepping-stones*, dove ogni individuo è libero di migrare verso qualunque altro gruppo, o *island models*, dove la migrazione è ristretta ai gruppi vicini, cioè all'"isola".

I vari algoritmi, utilizzati all'interno dei diversi gruppi, possono risultare omogenei, cioè agire in base agli stessi parametri su diverse popolazioni iniziali, o eterogenei, cioè agire in base a parametri diversi. Nel secondo caso ciascun algoritmo di gruppo può essere incaricato di ricercare la soluzione di uno degli aspetti del problema totale. Il *gradually distribuited genetic algoritm* crea due piani di ricerca, tramite l'utilizzo di *crossover* particolari; la migrazione provvede a scambiare individui fra i due piani dello stesso ipercubo. Gli algoritmi basati sulla *co-operating population* adottano regole simili ma modificano le probabilità di *crossover* e mutazione per ciascun gruppo. Una terza via per ottenere eterogeneità consta dell'adozione di algoritmi, evolutivi e non, messi in competizione per la risoluzione di un medesimo problema.

In Alba e Troya (1999) sono citate diverse realizzazioni basate sulla metodologia in oggetto: istruzione di reti neurali artificiali, ottimizzazione dei "pesi" delle loro connessioni, ottimizzazione di funzioni, ricerca di massimi in presenza di vincoli, distribuzione di elaborazioni su più nodi ed altre. Gli algoritmi genetici paralleli risultano, quindi, essere una realtà operativa che merita un certo grado di interesse.

I benefici, ottenibili dall'impiego del metodo, sono dati dalla somma di quelli tipici degli algoritmi genetici tradizionali e di alcuni peculiari degli algoritmi paralleli; in dettaglio:

- Si opera sulla codificazione del problema e non sui dati grezzi, ciò permette di predisporre algoritmi generalizzati utilizzabili per diverse applicazioni.
- Gli algoritmi sono fortemente indipendenti dal problema, perciò costituiscono un metodo robusto.
- E' possibile ottenere soluzioni alternative per uno stesso problema.
- La ricerca viene condotta, simultaneamente, su una molteplicità di punti appartenenti allo spazio delle soluzioni.
- La computazione può essere, facilmente, condotta in parallelo per gruppi o "isole".
- La ricerca risulta migliore anche in assenza di strumenti *hardware* paralleli.
- Gli algoritmi paralleli risultano maggiormente efficaci e più efficienti di quelli tradizionali, normalmente sequenziali.
- E' facilitata la cooperazione con altre tecniche di ricerca.

Per l'utilizzazione del metodo, in una struttura basata sullo schema ERA, si potrebbe dotare ogni agente di un proprio insieme di cromosomi, dei valori dei parametri di conduzione dell'algoritmo e delle regole di traduzione dei dati, dalla metrica dell'agente a quella dell'algoritmo. In questo modo ogni agente potrebbe evolvere autonomamente proprie strategie, relativamente a problemi sia comuni, sia specifici di ciascuno. La migrazione di individui, singole idee, potrebbe giustificarsi con l'ipotesi di comunicazione fra i vari agenti, realizzata in appositi incontri di approfondimento del problema.

Dotando ciascun agente di informazioni incomplete ma, in qualche modo, complementari si dovrebbe ottenere una interessante eterogeneità di sviluppo per ogni insieme di cromosomi. Ogni

agente potrebbe, inoltre, avere sensibilità diversa per i consigli, quindi accogliere un numero più o meno elevato di idee provenienti dai patrimoni di altri. Per ottenere un maggior isolamento potrebbero essere stabilite limitazioni alla comunicazione fra agenti, basate, ad esempio, sulla loro dislocazione nell'ambiente simulato, sull'appartenenza a gruppi cooperativi, sulla nazionalità, eccetera.

Il pacchetto GGI potrebbe includere la gestione di algoritmi genetici paralleli. La conduzione di tutte le operazioni verrebbe demandata al *kernel* del prodotto; l'isolamento dei singoli algoritmi sarebbe garantito dalla capacità, di *ruleMaster* e *ruleMaker*, di agire su aggregati diversi di regole. Le dotazioni di ciascun agente sarebbero contenute nei consueti oggetti: *dataWarehouse*, *classifierParm*, ed *interface*.

Bibliografia

ALBA E., TROYA J.M.(1999), A Survey of Parallel Distributed Genetic Algorithms - Complexity vol. 4 n. 4, pp. 31-52, John Wiley & Sons

ADORNI G., GAGLIO S., MASSONE L. (1987), Manuale di intelligenza artificiale, Roma, NIS

ASSOCIAZIONE ITALIANA PER L'INTELLIGENZA ARTIFICIALE (1998), Web page (reperibile su: http//www.di.unito.it/gopher-data/assoc/aiia/,brochure.html)

BURKHART R., MINAR N. e LANGTON C.(1998,) Documentation Set for Swarm 1.3.1 - Santa Fe Institute (reperibile su http://www.santafe.edu./projects/swarm).

CAVICCHIO D.J. (1970), Adaptive Search using Simulated Evolution
(Dd Michigan Universitiy)

CHATTOE E. (1996), The Simulation Of Budgetary Decision Making and Mechanism Of Social Evolution - paper for the ISA '96, Colchester

CHATTOE E. (1998) Just how (Un)realistic are Evolutionary Algorithms as Representation of Social Processes - Journal of Artificial and Social Simulations 1,3
(reperibile su http://www.soc.surrey.ac.uk/JASSS/1/3/2.html)

CRISTIANINI N. (1998), Sulla definizione di intelligenza - Sistemi Intelligenti, 3, p. 361, Bologna, Il Mulino

D'ALLESTRO S. (1998), Sulla definizione di intelligenza - Sistemi Intelligenti, 3, pp. 356-357, Bologna, Il Mulino

DAVIS I. (1991), Handbook of Genetic Algorithms, New York, Van Nostrand Reinhold.

DE GROOT M. H. (1970), Optimal Statistical Decisions, New York: McGraw-Hill

DE JONG K. A. (1975), An Analisys of the Behavior of a Class of Genetic Adaptive Systems
(Dd Michigan University)

EPSTEIN J.M. e AXTELL R. (1996), Growing Artificial Societies - Social Science from the Bottom Up, Cambridge MA, MIT Press.

FORD M. e HAYES P.J. (1999), Su ali algoritmiche: ripensare gli obiettivi dell'intelligenza artificiale - Le Scienze Dossier, 1, pp. 88-93, Milano, Le Scienze.

FREEMAN J.A. (1994), Simulating Neural Networks with Mathematica, Reading MA, Addison Wesley.

GALANTE L. (1997), La formichina che abita sullo schermo del computer, in TuttoScienze del 29/10/1997, p. 2

GIBBONS R. (1992), Teoria dei giochi, Bologna, Il Mulino Prentice Hall International

GOLDBERG D.E. (1989), Genetic Algorithms in Search, Optimization and Machine Learning , Addison Wesley Longman Inc.

GREFENSTETTE J.J. (1981), Parallel Adaptive Algorithms for Function Optimization
(Tecnical Report Vanderbilt University)

GREFENSTETTE J.J: (1991), Strategy Acquisition with Genetic Algorithms, in Davis (1991)

HAJEK S. (1997) in Forum Teorema - Intelligenza e adattamento - Intervista a J. Holland (reperibile su http://www.logikos.it/teorema/Interview.htm)

HOFSTADTER D.R. (1979) Gödel, Escher, Bach: un'Eterna Ghirlanda Brillante, Milano, Adelphi

HOLLAND J.H. (1975), Adaptation in Natural and Artificial Systems, Cambridge MA, MIT Press.

HOLLSTIEN R.B. (1971), Artificial Genetic Adaptation in Computer Control Systems (Dd Michigan University)

HOLZNER S., NORTON P. (1992) Programmazione in C++, Milano 1997, Jackson

JUDD K.L. (1996), Computational Economics and Economic Theory: Substitutes or Complements? (reperibile su http://bucky.stanford.edu/)

KIRMAN A. (1992), Whom or What Does the Representative Agent Represent? - Journal of Economic Perspectives, 6, pp. 126-139.

LANGTON C., MINAR N. e BURKHART R. (1995,) Web Site of Swarm - Santa Fe Institute (reperibile su http://www.santafe.edu./projects/swarm).

MARGARITA S. (1992), Verso un "robot oeconomicus" algoritmi genetici ed economia - Sistemi Intelligenti, 3, pp. 421-459.

MEO A. R. (1998), Bozza di proposta di un programma nazionale di ricerca sul tema freeware - Politecnico di Torino

MORASSO P. (1998), Sulla definizione di intelligenza - Sistemi Intelligenti, 3, p. 357 e 359, Bologna, Il Mulino

NEWELL A., SIMON H.A. (1976), Computer Science as Empirical Inquiry: Symbols and Search - Communications of the ACM, 3, pp. 113-126.

PERRY Z. A. (1984), Experimental Study of Speciation in Ecological Niche Theory Using Genetic Algorithms
(Dd Michigan University)

RICH E., KNIGHT K.(1991), Intelligenza Artificiale, Milano, MCGraw-Hill

SCHAFFER J.D. (1984), Some Experiments in Machine Learning using Vector Evaluated Genetic Algorithms
(Dd Vanderbilt University)

SIMON H. A. (1981), "Economic Tationality: Adaptive Artifice - The Sciences of the Artificial, Cambridge MA, MIT Press.

SPEARS W.M., DE JONG K. (1990), Using Neural Networks and Genetic Algorithms as Heuristic for NP-Complete Problems, in Proceedings of the International Joint Conference on Neural Networks, Hillsdale N. J., Lawrence Erlbaum Associates.

SYLOS LABINI P. (1982), Oligopolio e progresso tecnico, Torino, Giulio Einaudi Editore

TERNA P. (1998), Creare mondi artificiali: una nota su Sugarscape e due commenti - Sistemi Intelligenti, 3/98, pp.489-496, Bologna, Il Mulino.

TERNA P., MARGARITA S. (1988), Rassegna di strumenti informatici, Torino, G. Giappichelli

TESFATSION L. (1997), Old Web Site for Agent-Based Computational Economics (ACE)
(reperibile su http://www.econ.iastate.edu/tesfatsi/abe.htm)

TESFATSION L. (1998), The Labor Market: a Simulative Perspective
(reperibile su http://www.econ.iastate.edu/tesfatsi/evlab.ps)

TIRASSA M. (1998), Sulla definizione di intelligenza - Sistemi Intelligenti, 3/98, p. 360, Bologna, Il Mulino

VARI (1989), Artificial Life
(reperibile su:
http//www.liberliber.it/biblioteca/tesi/apprendimento_di_significato_in_reti_competitive.html)

VARI (1999), Modelli evolutivi - l'algoritmo genetico
(reperibile su http://www.logikos.it/teorema/Modevo.htm)

ZAMAGNI S. (1987), Economia politica - Teoria dei prezzi, dei mercati e della distribuzione, Roma, NIS.

YAM P. (1999), Considerazioni sull'intelligenza - Le Scienze Dossier, 1, pp.6-11, Milano, Le Scienze.