L’ottimizzazione dell’impiego di filatoi mediante simulazione: il caso Filatura Marchi

SIGILL·VNIVERS·AVGVSTAE·TAVRINORVM·

Desidero esprimere la mia riconoscenza a Massimo Marchi, Presidente della Filatura Marchi Giovanni S.p.A., *per l'infinita disponibilità e per la fiducia che mi ha voluto dimostrare, tollerando la mia presenza presso la sede dell'impresa che dirige e permettendomi di accedere a tutte le informazioni che sono state necessarie al completamento di questo lavoro.*

Ringrazio inoltre Gustavo Cametti, che mi ha saputo consigliare saggiamente, orientandomi nelle fasi iniziali e guidandomi pazientemente anche nel seguito del mio lavoro. L'Ing. Luca Cinguino mi è stato indispensabile in ogni momento, competente supervisore delle mie ricerche e vero alter ego del Presidente quando si è trattato di sviscerare tutti gli aspetti della realtà aziendale. Per tutte queste ragioni e per la sua pazienza senza limiti penso di essergli debitore in maniera particolare.

La mia gratitudine più profonda va al Dott. Gianluigi Ferraris, preziosissimo amico che mi ha concesso di utilizzare e di integrare nel mio lavoro il suo formidabile pacchetto software Genetic Manipulator, *e che con la sua intelligenza critica mi ha permesso di affrontare con successo molte difficoltà.*

Matteo Morini

Indice

1 Introduzione

In questo lavoro viene affrontato un problema comune a numerose aziende di quasi tutti i settori: la programmazione della produzione. Il caso di una produzione molto diversificata, la cui lavorazione avviene su di un parco macchine numericamente poco consistente, è diffuso in una moltitudine di aziende, tra cui vale la pena ricordare le tessiture, le tintorie, le filature, per limitarsi alla filiera tessile. Da questa situazione nasce l'esigenza di adattare (riattrezzare) di volta in volta gli strumenti produttivi (telai, vasche, filatoi), in base alle caratteristiche tecniche dei prodotti in lavorazione.

Nel caso specifico è stato affrontato il problema dal punto di vista di una filatura: i filatoi installati (una quindicina) sono estremamente versatili, e ciascuno di essi può potenzialmente produrre una qualunque delle tipologie di prodotto in catalogo (centinaia), se opportunamente predisposto. Le operazioni necessarie a predisporre un filatoio, gli eventuali ritardi di consegna e la gestione del magazzino sono alcune delle componenti che determinano il costo finale della produzione per l'azienda. Lo scopo del programma realizzato è quello di ottimizzare la disposizione dei lotti sui filatoi disponibili, minimizzando i costi totali risultanti.

Nella prima parte del lavoro vengono presentate le fondazioni teoriche della metodologia di simulazione utilizzata, il cosiddetto *Agent–Based Modelling* (ABM); in particolare si fa riferimento a lavori specificamente incentrati sul *Multi–Agent Enterprise Modelling*, sul concetto di *Supply Chain Network* (SCN), sui *Multi–Agent Information Systems* (MAIS).

Si mostra inoltre come le librerie Swarm, sviluppate inizialmente presso il Santa Fe Institute of Complexity ed ora manutenute da un gruppo di sviluppatori volontari indipendenti, mettano a disposizione un ambiente di sviluppo ideale per la realizzazione di modelli di SCN.

La seconda parte presenta nei dettagli l'azienda oggetto dello studio, che ha accettato la proposta di collaborazione ed ha ospitato la ricerca, mettendo a disposizione cospicue risorse in termini di tempo ed informazioni. Sono descritte le diverse fasi della produzione ed i reparti nei

quali queste sono svolte. Si accenna al ciclo produttivo dei filati, al passaggio delle materie prime attraverso la fase di mistatura, alle successive lavorazioni a cui le fibre semilavorate vengono sottoposte: cardatura e stiro. Si descrive infine, più in dettaglio, il momento della filatura, senza trascurare il ruolo delle componenti sostituibili dei filatoi: *cardíne* e *rotori*. Per ultime sono descritte le operazioni di vaporizzo ed imballaggio.

Ci si sofferma in particolare sui numerosi fattori dei quali si deve tenere conto nel momento della programmazione della produzione, e si spiega come l'assegnazione dei lotti ai singoli filatoi sia un'operazione cruciale a causa della moltitudine di componenti di costo che ne sono coinvolte.

Tra i fattori che vanno tenuti sotto controllo, in quanto generatori di costi reali o figurativi, sono analizzati i ritardi di consegna, i costi di riattrezzaggio, nelle relative componenti di costo orario delle squadre di attrezzisti, tempi di fermo macchine ed eventuali interferenze causate da *setup* simultanei; inoltre vi sono costi da ri-roccatura, costi di magazzino (dovuti ad un vincolo sul titolo medio) e costi di mistatura causati da una non omogenea distribuzione nel tempo di prodotti in funzione delle loro caratteristiche cromatiche.

La terza ed ultima parte della tesi introduce il programma realizzato, ed è la più importante. Il modello della filatura, realizzato in un dialetto ad oggetti del C (l'*Objective-C*), è basato su agenti, ciascuno dei quali ha il ruolo di un elemento costituente l'azienda. Esistono agenti con corrispondenti reali: `SpinnerManager`, `Spinner`, `Order` fanno rispettivamente le veci del dispositore aziendale, dei filatoi, dei lotti di prodotto. Altri oggetti, come `DBInterface`, `DataWarehouse` od `OrderList` hanno una mera funzione di servizio, e non si possono definire agenti in senso proprio. Altri oggetti ancora sono stati introdotti in aderenza allo standard per la realizzazione di modelli "ERA" ([GT00]). Si tratta di `SpinnerRuleMaster`, `SpinnerManagerRuleMaster` e `SpinnerManagerRuleMaker`.

Nel seguito si approfondiscono alcuni aspetti tecnici della simulazione, come la gestione del calendario e l'accesso alla base di dati aziendale.

Le modalità operative del modello sono presentate in riferimento ai possibili modi di agire dell'agente dispositore, che può effettuare programmazioni casuali, replicare programmazioni suggerite dall'esterno od effettuare ricerche di programmazioni ottimali. Quest'ultima modalità di funzionamento si basa sull'applicazione di un motore evolutivo fondato su di un algoritmo genetico. La funzione che guida la ricerca è il costo totale che risulta da ciascuna programmazione, funzione che l'AG cerca di minimizzare.

Alcuni cenni sono dedicati al funzionamento degli AG ed alla transcodifica tra programmazioni di ordini e stringhe binarie.

Infine sono presentati i risultati dell'applicazione del modello ad alcuni casi reali, con confronti tra i costi ottenuti senza nessuna ottimizzazione (programmazioni casuali), i costi minimi ottenuti da un programmatore autentico ed i costi ottenuti grazie al modello.

La migliore performance del programma si è ottenuta nel caso di un'assegnazione di 32 ordini su 14 filatoi: il costo medio di una programmazione random è stato normalizzato a **100**; si è ottenuto un risultato pari a circa **25** (il 75% in meno) con la programmazione effettuata dal dispositore umano e circa **10** (il 90% in meno della programmazione random, circa il 60% in meno della programmazione "umana") grazie all'uso dell'AG.

Parte I

Le catene di fornitura

2 Simulazioni al computer e modelli ad agenti

I fenomeni fisici, sociali ed economici sono alcuni degli oggetti di studio nei quali il ricorso alla simulazione è sempre più diffuso. Alcune delle ottime ragioni per le quali si impiega tale metodologia, specialmente in abbinamento alla modellizzazione basata su agenti, sono trattate in [Axt00].

La simulazione al computer consiste essenzialmente nell'uso (oppure nella creazione *ex-novo*) di *software* che, funzionando, misura i valori e le variazioni dei valori delle variabili scelte per rappresentare il fenomeno oggetto di studio. L'esperimento consiste nell'osservare un ambiente artificiale, nel quale un sistema reale è riprodotto in tutti gli aspetti analiticamente rilevanti, ed eventualmente interagire con esso, variando le condizioni iniziali del sistema, riprogettando le interazioni fra le varie componenti, aggiungendo od eliminando elementi all'architettura. La macchina diventa, in questa prospettiva, *interprete duttile degli obiettivi dell'utente* ([Haj97]), e lo sperimentatore diviene il creatore di veri e propri "mondi artificiali": al riguardo si vedano ad es. le applicazioni sviluppate in [Axt99, Eps96], ma anche [AE94, Eps99a, MT00a, Ter01].

Un modello realizzato sulla base di questo paradigma permette allo studioso-sperimentatore di condividere i risultati del suo lavoro al massimo livello possibile, trasmettendo cioè l'oggetto stesso del suo esperimento, che diventa agevolmente e compiutamente ripetibile.

La simulazione può rivelarsi particolarmente indicata per affrontare lo studio di sistemi complessi[1], e la metodologia alla quale è più opportuno rifarsi in questi casi è la modellistica cosid-

[1] Il termine "complesso" non va inteso nel senso di "complicato", ma serve a connotare sistemi in cui interazione tra componenti numerose ed eterogenee e non linearità dei fenomeni implicati possono dare luogo a comportamenti inaspettati, tecnicamente detti *emergenti*, come quelli che si incontrano tipicamente nelle scienze sociali.

detta “ad agenti”, nella quale il problema viene scomposto nelle sue componenti elementari, e ci si propone di osservare a livello macroscopico il risultato di azioni ed interazioni che sono state descritte al solo livello microscopico (si parla a proposito di modelli sviluppati “dal basso verso l’alto”).

Molto spesso la modellizzazione ad agenti è l’unica via percorribile per trattare problemi esplicitabili analiticamente in forma di sistemi di equazioni, ma non risolvibili con le tecniche tradizionali; in questo caso un modello basato su agenti può rivelarsi molto utile: *“. . . [the ABM] can shed significant light on the solution structure, illustrate dynamical properties of the model, serve to test the dependence of results on parameters and assumptions”* (da [Axt00]).

In alcuni casi la possibilità di affrontare analiticamente il problema è esclusa a priori, essendo impossibile impostarne la soluzione su di un tradizionale sistema di equazioni (problema *intrattabile*); realizzare un modello ad agenti può rivelarsi molto produttivo.

In [FD96] si sostiene, citando [Kuh62] al riguardo dell’imprevedibilità di certi risultati, che è proprio grazie ad essi che si può migliorare la propria comprensione delle relazioni empiriche sottostanti, ed è dallo studio di questi che ci si può aspettare di ottenere risultati originali in grado di generare progresso ed innovazione. Conclusioni “interessanti” sono spesso dovute a ciò che viene definito in [FD96] come *“the unique and contingent mix of insight coupled with chance”* o, in una sola parola, “*serendipity*”.

3 Multi–Agent Enterprise Modelling

Un’azienda è un esempio particolarmente calzante di realtà “complicata” e “complessa” allo stesso tempo; in [LTS96] è compiuta un’operazione di smembramento indirizzata ad individuare le singole componenti che, combinate nella gerarchia aziendale, costituiscono una catena produttiva.

L’impresa, infatti, viene ridotta ad una rete di entità comunicanti, il cui scopo ultimo è quello

di portare a compimento tutte le fasi necessarie alla produzione di un “ordine”.

Un aspetto particolarmente importante dei modelli di impresa è il livello di astrazione al quale ci si vuole porre nel rappresentare ciascuna delle componenti: esistono livelli di aggregazione distinti, come il livello dell’industria (costituito dalle diverse imprese che hanno rapporti tra di loro), il livello del singolo impianto (nel quale i rapporti si hanno tra diverse linee produttive), fino al livello delle singole macchine.

I concetti sui quali si basa l’intera metodologia di modellizzazione aziendale basata su agenti sono descritti nel seguito:

Supply Chain Network (*SCN*): è una rappresentazione della gerarchia aziendale in forma di rete di entità (economiche) autonome o semi-autonome. Ciascuna delle unità che costituiscono la SCN dispone di collegamenti verso l’alto ed il basso rispetto alla catena produttiva. Attraversando i nodi che rappresentano diverse attività, o processi, il prodotto finale (bene o servizio) viene completato e messo a disposizione del cliente.

Order Fulfillment Process (*OFP*): L’attività fondamentale della SCN è la produzione dell”’ordine”, e l’ottimizzazione dell’OFP è uno degli obiettivi che si ricercano attraverso la simulazione ad agenti di un’impresa.

MultiAgent Information System (MAIS): Il MAIS è il software che viene sviluppato allo scopo di simulare l’OFP, rappresentandolo in un modello schematico.

Enterprise Integration (*EI*): La gestione delle interazioni tra i partecipanti ai processi che devono essere seguiti nell’impresa (spesso lunghi ed articolati) può migliorare sensibilmente la performance aziendale. L’EI tenta di migliorare la coordinazione tra le organizzazioni, gli individui ed i macchinari. Le tre tipologie di integrazione alle quali le imprese tendono sono:

Inter-company: Concerne l’integrazione tra *partner* della stessa dimensione o verso for-

nitori e terzisti. Si definisce anche *vertical partnering*, ed interessa le informazioni sui prodotti.

Intra-company: All'interno di una singola impresa si parla di *horizontal partnering* o *process information based integration*: lo scopo è quello di realizzare un'infrastruttura di processo integrata per la produzione.

Value chain integration: Si tratta di un'integrazione flessibile, detta anche *business based integration*, e comprende imprese che hanno simultaneamente il ruolo di fornitori e clienti dei rispettivi prodotti.

Enterprise Integration Models: Lo scopo degli EIM è la descrizione delle operazioni svolte all'interno delle imprese in termini di "funzionalità" e "comportamento dinamico". Ne sono importanti requisiti la fedeltà di rappresentazione della realtà e la possibilità di osservarne le operazioni stesse esercitando una funzione di controllo. In letteratura sono stati presentati numerosi EIM, i più importanti dei quali sono:

CIMOSA: Questa architettura di modello (*Open System Architecture for Computer Integrated Manufacturing*) definisce tra livelli separati ma comunicanti: *requirement definition*, *design specification* ed *implementation description*, tre strati nel modello: *generic building blocks*, *partial models* e *particular models*, e quattro punti di vista: *function*, *information*, *resource* ed *organization*.

ARIS: È uno dei modelli di riferimento per l'industria: l'"*Architecture of Integrated Information Systems* ricorre a quattro punti di vista diversi: *function*, *organization*, *data* e *control*. Sulla base della loro vicinanza alle risorse del sistema informativo aziendale queste "viste" sono suddivise in tre livelli descrittivi: *requirement definition*, *design specification* ed *implementation description*.

GIM: Questa architettura (*GRAI Integrated Methodology*) definisce nel modello due dimensioni differenti: i livelli delle "viste" ed il livelli delle "astrazioni".

PERA: Nella *"Purdue Enterprise Reference Architecture"* sono previste cinque fasi di sviluppo: *concept phase*, *definition phase*, *design phase*, *construction and installation phase* ed *operation phase*.

IEM-GAM: L'*"Integrated Enterprise Modelling"* è orientato verso la modellizzazione orientata agli oggetti; un'impresa manifatturiera viene suddivisa in *prodotti*, *ordini* e *risorse*; esistono due punti di vista: la vista *funzionale* che descrive gli aspetti funzionali degli oggetti implicati nel modello della produzione, e la vista dell'*informazione*, che ne descrive le caratteristiche.

Per approfondimenti relativi ai diversi modelli di EIM si rimanda alla sostanziosa bibliografia presentata in [LTS96].

3.1 Un modello concettuale per l'Enterprise Modelling

Lo scopo della realizzazione di un modello di impresa è migliorare la propria comprensione dei suoi meccanismi interni; in un'azienda reale ognuna delle componenti, pur operando con uno scopo comune, ha obiettivi e programmi di lavoro diversi. Si intende rappresentare l'*interdipendenza* tra le diverse entità, comprendendo anche le attività che hanno luogo all'interno dell'impresa.

Un'altro requisito del modello è la capacità di rappresentare diversi livelli di dettaglio, corrispondenti ai diversi livelli di astrazione possibili.

Le componenti di un modello di impresa sono presentate nel seguito:

***business unit*:** ad un alto livello di astrazione (es. di industria) si intende un'organizzazione; ad un livello inferiore (es. un impianto) si potrebbe riferire ad un reparto od una squadra responsabile per un compito specifico;

risorsa fisica: all'interno di un'organizzazione, indica i macchinari o gli impianti, a livello di

industria si riferisce alla capacità di produrre; a livello di impianto indica il numero di linee produttive;

processo: ciascuna attività all'interno di un'impresa corrisponde ad un *processo*, che può essere ulteriormente suddiviso in sequenze di *sub-processi*;

***input* ed *output*:** i processi ricevono un input e restituiscono un output; se ne distinguono due tipi, fisici e di informazione.

relazione ed interazione: ogni business unit ha relazioni con una o più altre business unit; se l'output di una di queste *unit* diventa l'input di un'altra si ha una relazione, ossia le unit interagiscono tra di loro;

flusso di informazioni e flusso di comunicazione: esistono flussi materiali e flussi di informazioni (comunicazioni); flussi di informazione efficienti permettono alle business units di ottenere le informazioni necessarie nel momento giusto;

decisione e strategia: i processi di una business unit sono guidati da decisioni e strategie, che ne influenzano i processi e la performance;

fonte di conoscenza: la sopravvivenza di un'impresa dipende in modo critico dalla disponibilità di informazioni attendibili; in una SCN diversi strati filtrano le informazioni sulla domanda prima che dai clienti giunga ai fornitori;

obiettivo aziendale: tutte le decisioni e le operazioni di un'impresa sono pilotate dagli obiettivi aziendali; la misura delle performance rispetto agli obiettivi serve a valutarne il buon funzionamento;

misura di performance: la performance di un sistema è misurata sulla base dei metodi e delle variabili identificate come significative;

adattabilità: le business units nel mondo reale non hanno comportamenti statici, ma si adattano ai cambiamenti dell'ambiente; questi possono essere affrontati nel migliore dei modi anticipandoli.

3.2 Componenti aggiuntivi del modello, orientati alla simulazione

Altri elementi possono entrare a far parte di un modello di impresa con lo scopo di sperimentare strategie differenti:

linguaggio e piattaforma: il linguaggio della simulazione deve poter rappresentare gli obiettivi del modello: l'ambiente Swarm si presta particolarmente per le sua caratteristiche di ambiente *agent-based*. Ciascuna *business unit* viene modellizzata come gruppo (*swarm*) di agenti, ed ogni agente corrisponde ad una risorsa fisica che esegue un insieme di azioni;

evento/stato/flusso: ogni azione in una simulazione è modellizzata come un "evento", o una serie di eventi. Una azione consiste nel modo in cui un agente si comporta nei confronti del mondo, sulla base della funzione di decisione che esso contiene e che valuta gli *input* provenienti dal mondo esterno e gli stati cognitivi dell'agente stesso.

***schedule*:** la sequenza di eventi che costituiscono la simulazione è detta *schedule*, e coordina le interazioni tra gli agenti;

formazione di agenti ed oggetti: ogni agente può essere dotato di variabili di stato interne, di funzionalità e di conoscenza, il che mette a disposizione un sistema molto flessibile per realizzare individui che possono essere organizzati in società;

scambio di informazioni tra agenti: la capacità che gli agenti hanno di comunicare tra di loro.

3.3 Value–chains e Process–chains

Nel contesto della simulazione aziendale si può definire un'impresa come complesso di *business entities* che collaborano con il fine di consegnare ai clienti prodotti o servizi. Tali entità possono appartenere ad una singola impresa od essere compagnie individuali separate.

Dal punto di vista del prodotto questo processo è una *value–chain*, dove l'output finale è il bene o servizio pronto per la consegna. Ogni business unit aggiunge valore al proprio output che, al termine della "catena di valore", costituisce il prodotto finito.

Dal punto di vista del processo, la catena di valore si può considerare una sequenza di processi che successivamente trasformano degli input in degli output.

In una *process–chain* ogni output intermedio diventa l'input dell'unità del processo successiva. Non necessariamente la catena è lineare: si può definire una *gerarchia di processi* che trasforma diverse materie prime (più input) in un solo output, viceversa un solo input può, seguendo processi diversificati, essere trasformato in diverse tipologie di prodotti finiti.

3.4 Proprietà di tipo "OO" del modello

Il modello può beneficiare molto dalle proprietà "object–oriented" che sono prerogativa del metodo di modellizzazione prescelto.

Le proprietà di un modello "OO" sono:

***encapsulation*:** i metodi e le variabili di ciascun oggetto sono definiti localmente ed "incapsulati" all'interno dell'oggetto stesso (per quanto riguarda i metodi, è la loro implementazione ad essere nascosta nell'oggetto; le interfacce sono invece esposte verso l'esterno);

ereditarietà: in una gerarchia di oggetti e possibile definire degli oggetti "figli" come estensione di oggetti "padri", dei quali ereditano le proprietà (metodi e variabili), aggiungendone altre ancora;

concetto di classe: gli oggetti funzionalmente equivalenti (ma differenti quanto a stati interni e valori delle variabili di stato) sono raggruppati in famiglie di oggetti omogenei dette classi.

3.5 La piattaforma Swarm per la simulazione

Swarm è un pacchetto software di portata molto generale, in grado di simulare mondi artificiali "concorrenti" (multi agente); è stato sviluppato per lo studio di sistemi adattivi complessi.

Swarm mette a disposizione un'architettura orientata agli oggetti nella quale è possibile definire i comportamenti degli agenti e degli altri oggetti che interagiscono nel corso della simulazione. In Swarm ogni agente è costruito in modo tale che comprende: (1) variabili di stato organizzate in una struttura di dati, (2) una funzione che ad ogni passo lo fa operare, (3) funzioni che vengono richiamate da messaggi provenienti dall'esterno dell'agente (da altri agenti o dagli oggetti che controllano la simulazione: ad es. gli oggetti di tipo `Schedule`).

La modellizzazione è basata su individui; le azioni di ogni agente sono determinate dalle variabili di stato interne che gli sono proprie; ogni individuo ha una propria visione locale del mondo; la combinazione di comportamenti individuali determina il comportamento collettivo dell'intero gruppo.

Ogni agente in swarm può consistere di uno sciame (swarm) di agenti; il comportamento di un tale agente è determinato dalle azioni compiute dai suoi agenti costituenti; si parla di "proprietà gerarchica nidificata" insita.

Ogni sciame di agenti comprende un'"agenda" di azioni (schedule), indipendente dagli altri sciami. Tutti gli agenti appartenenti ad uno sciame condividono la stessa agenda di azioni. Le azioni sono ordinate secondo un riferimento temporale globale, con la possibilità di eccezioni a questa regola.

3.5.1 Swarm come Multi-Agent Information System (MAIS)

Presentato in questa prospettiva, Swarm comprende quattro componenti: agenti, compiti, organizzazioni ed infrastruttura delle informazioni.

agente: è un oggetto attivo dotato di capacità di eseguire determinati compiti; comunica con altri agenti, seguendo la struttura gerarchica prevista, allo scopo di cooperare nel raggiungimento di scopi;

compiti: sono le azioni che ciascun agente deve svolgere una volta che gli sono state assegnate; i compiti possono essere scomposti da un agente ed assegnati ad altri agenti, al contrario compiti svolti da diversi agenti possono essere raggruppati in un fine globale;

organizzazioni: le relazioni tra agenti in un MAIS formano delle organizzazioni; esistono controlli delle relazioni centralizzati e decentrati, ed i controllo delle relazioni tra agenti può essere di uno dei due generi citati ovvero di tipo ibrido;

infrastuttura delle informazioni: costituisce il fondamento della comunicazione tra agenti; l'informazione può essere condivisa tra agenti tramite lo scambio di messaggi o attraverso l'accesso a basi di dati comuni.

3.5.2 Modellizzare un'impresa come SCN con Swarm

Una SCN può venire rappresentata in Swarm in maniera molto soddisfacente: le diverse "business entities" della SCN hanno agenti di Swarm costruiti con variabili interne e funzioni diverse come corrispettivi; la descrizione multilivello della SCN è adatta ad essere catturata dalla struttura "inerentemente gerarchica nidificata" di Swarm; flussi di informazioni sono rappresentati dai messaggi che gli agenti si scambiano; ogni attività della SCN viene simulata come "evento discreto", ed inserita in uno degli step temporali possibili, dunque richiamata quando necessario;

la performance globale della SCN, dovuta ai processi eseguiti a livello individuale dalle entità aziendali, in Swarm è misurata dal risultato del comportamento collettivo dello (dei) sciame(i).

Le proprietà di un buon prototipo di MAIS sono:

1. una definizione efficiente degli agenti e delle (loro) azioni: agente $\Leftrightarrow$ oggetto; azione $\Leftrightarrow$ metodo;
2. uso flessibile dei protocolli di comunicazione disponibili: variabili condivise o messaggi;
3. presenza di meccanismi di controllo, centralizzati o decentrati in base alla struttura dell'organizzazione;
4. configurazione statica o dinamica delle strutture della SCN, in base al tipo di ambiente aziendale simulato;

Esistono agenti *fisici* ed agenti *logici*: per la simulazione dei primi si può usare un approccio "ad eventi discreti", per i secondi, più attivi, una agenda di azioni da compiere passo a passo. Agenti fisici che fanno parte di una SCN sono: fornitori, magazzini, assemblatori (a loro volta costituiti da agenti della linea del flusso, cioè sciami di macchinari ed agenti che assemblano), distributori, dettaglianti. I componenti del sistema informativo sono un sistema di ingresso degli ordini, pianificatore delle capacità, pianificatore dei materiali, programmatore della produzione e controllore del magazzino.

Un agente è composto da:

agent id: un'etichetta che distingue ogni agente da tutti gli altri;

agent state: descrive lo stato interno dell'agente (occupato, libero...);

action engine: stabilisce il (i) compito(i) da compiere e la procedura necessaria al loro svolgimento;

database: memorizza le informazioni relative ai compiti dell'agente, per facilitare i processi interni o per condividerle con altri agenti;

knowledge base: memorizza le conoscenze necessarie ai compiti dell'agente, conoscenze rivedibili in base alla capacità di apprendimento;

learning mechanisms: pongono l'agente nella condizione di adattare le sue azioni a mutate condizioni;

In aggiunta agli agenti ed ai processi relativi all'OFP, nel MAIS viene creato un agente osservatore (o più di uno): gli observer agents registrano le statistiche degli agenti durante la simulazione: questo rende disponibili dati come tempo medio di riordino, tasso di evasione degli ordini, costo medio di immagazzinamento. Un'altra possibilità offerta da Swarm è l'effettuare probing a livello di singolo agente, per ispezionarne le variabili interne.

3.5.3 Simulare l'OFP nelle SCN con Swarm

L'OFP-ModelSwarm è composto da una matrice di entità appartenenti alla SCN; le proprietà di ciascuna di queste sono introdotte nella fase di creazione; le azioni dell'OFP sono composte dalle azioni individuali di ogni entità della SCN; queste azioni vengono attivate quando l'OFP-ModelSwarm è attivato.

Gli agenti creati sono:

1. Agente di gestione degli ordini `[OrdM]`: stima le scadenze da rispettare per gli ordini ricevuti (dall'SCN Swarm), la disponibilità di prodotti in magazzino (tramite richieste all'agente di gestione del magazzino) ed esegue ordinazioni ai fornitori che lo precedono nella catena produttiva.

2. Agente di gestione del magazzino `[InvM]`: mantiene dei registri delle merci immagazzina-

te, determina il punto di riordino e risponde alle richieste di disponibilità a magazzino da parte di altri agenti (1., 5., 6.)

3. Agente pianificatore della produzione `[PrdP]`: riceve gli ordini da (1.) e, in base alle scadenze, alla capacità produttiva ed alla disponibilità di materie prime, genera programmi di produzione; questi saranno usati da (5. e 6.) per eseguire il processo produttivo;

4. Agente pianificatore della capacità produttiva `[CapP]`: tiene conto dell'utilizzo della capacità produttiva dei sistemi produttivi, e fornisce informazioni sulla disponibilità di capacità produttiva a 3. per la pianificazione della produzione;

5. Agente pianificatore dei materiali `[MatP]`: calcola le necessità di materiali per la produzione (da magazzino o da acquisti da fornitori); inoltre fornisce informazioni a 3. per la produzione di programmi di produzione.

6. Agente di controllo del magazzino `[ShopC]`: spedisce materiali dal magazzino ai sistemi produttivi, in base al piano di lavoro deciso sulla base dei piani di produzione; inoltre fornisce informazioni sull'utilizzo della capacità produttiva;

7. Agente produttore `[ManuS]`: esegue il processo produttivo utilizzando componenti come input e consumando capacità produttiva;

8. Agente di gestione della SCN `[ScnM]`: sceglie i fornitori per le entità che devono acquisire componenti per la produzione o prodotti finiti per la distribuzione.

Esempio: `Swarm A` riceve un ordine da `Swarm C`. `OrdM` riceve l'ordine e gli assegna una scadenza (si basa su `InvM`, `PrdP`, `CapP`); se il prodotto è in stock, l'ordine è evaso consegnando il prodotto già pronto in magazzino; se il prodotto è in fase di arrivo, la data di consegna prevista è fissata di conseguenza. Altrimenti, se l'entità ha capacità produttive, l'ordine viene girato a `PrdP`, che lo inserisce in agenda, pianificandone la produzione riferendosi anche a `CapP` e `MatP`.

`ScnM` infine riceve le informazioni sull'ordine per allocare le richieste di materiali e trasferirle ad un'altra entità (es. `Swarm B`). Se questa è un distributore od un dettagliante senza capacità produttive,i prodotti vengono richiesti da altri fornitori come prodotti da fornire a chi li ha richiesti; se l'entità è in grado di produrre, i prodotti ordinati sono forniti da `ManuS`.

Parte II

La Filatura

La *Filatura Marchi Giovanni S.p.A.* fu fondata nel 1969 da Giovanni Marchi, padre dell'attuale titolare e Presidente, Massimo Marchi. L'attività, inizialmente, era di filatura cardata tradizionale; la produzione consisteva di filati rigenerati e filati misti cotone.

A pochi anni dalla fondazione venne acquistato il primo filatoio a tecnologia *open–end* a rotore, tecnologia sulla quale oggi è basata la totalità della produzione. I filatoi attualmente installati hanno il loro punto forte nella flessibilità; una opportuna configurazione ne permette l'impiego nella filatura dei materiali più disparati: lana, cotone, viscosa, lino, fibra acrilica, poliestere ed altre fibre ancora.

La produzione della Filatura Marchi, che attualmente consiste di filati semplici e ritorti, greggi e tinti (in filo su rocche od in fibra su fiocco), viene esportata per il 40% circa, in prevalenza verso il continente europeo.

L'azienda impiega circa 80 persone, tra impiegati ed operai; di alcune delle lavorazioni si occupano terzisti: tra queste le principali sono sfilacciatura, tintura in filo su rocca, roccatura e ritorcitura.

4 Il ciclo produttivo ed i reparti

Il processo produttivo della filatura comprende quattro reparti: nell'ordine mistatura, cardatura, stiro e filatura, attraverso i quali transitano le materie prime ed i semilavorati che sono infine trasformati in filati.

Le prime fasi della lavorazione, che si trovano a monte del reparto filatura, non hanno un contenuto tecnologico (e costi di impianto) particolarmente elevati; sono svolte per mezzo di

impianti sovradimensionati. Si ha invece una “strozzatura” nella quarta ed ultima fase produttiva, quella in cui si effettua la filatura vera e propria: è infatti la capacità del relativo reparto a limitare la capacità produttiva dell’azienda. Il costo degli impianti nel reparto filatura ha un peso notevole: ciascuno dei filatoi, macchine ad altissimo contenuto tecnologico, deve essere utilizzato in maniera ottimale, sempre ai limiti della capacità produttiva; ciò è cruciale per l’economia dell’azienda.

Per inquadrare opportunamente gli aspetti tecnici della produzione sono presentati, nei paragrafi successivi, i dettagli di ciascuna fase di lavorazione; tutte le informazioni sono state tratte da [Ghe01].

4.1 Mistatura

In questa fase le materie prime subiscono la prima lavorazione: premettiamo che tanto le fibre tessili naturali quanto le fibre sintetiche presentano normalmente grande eterogeneità tra lotti di provenienza diversa ma anche, fermo restando il fornitore, variabili in funzione del momento. Le caratteristiche di un lotto di filato, al contrario, devono essere quanto più uniformi sia possibile, anche nel caso di produzioni attuate a distanza di tempo.

La “mischia” è l’operazione che, dosando opportunamente ed accuratamente lotti di materia prima poco uniformi, permette di ottenere una massa semilavorata sufficientemente omogenea. Le due tecniche impiegate sono presentate nel seguito.

Al reparto mistatura in effetti compete anche gran parte dell’operazione di pulizia della materie prime, che al loro arrivo contengono frequentemente sostanze estranee. Questo aspetto è particolarmente importante a riguardo delle balle di cotone, meno evidente nel caso delle fibre di origine chimica. La pulitura, attraverso l’azione di potenti correnti d’aria, di nastri e di rulli chiodati, elimina dai fiocchi sostanze che vanno dai residui di involucro alla sabbia, a frammenti di foglie o semi. In questo reparto si riescono ad eliminare all’incirca i $2/3$ delle impurità; il terzo restante viene eliminato successivamente, dalla carda.

4.2 Carderia

La seconda lavorazione alla quale le fibre provenienti dal reparto di mistatura consiste nella cardatura; questa operazione si articola nelle fasi seguenti:

- Apertura del materiale, fino allo stato di singola fibra;
- Allontanamento delle residue impurità, sfuggite durante la pulizia preliminare;
- Eliminazione delle fibre più corte che finirebbero per peggiorare le caratteristiche finali di resistenza e qualità del filato;
- Rimozione dei cascami e dei “neps” (piccoli grovigli);
- Ulteriore mistatura delle fibre;
- Formazione di un “nastro”, ossia di una grossa corda appiattita di fibre non ritorte;

Il nastro risultante viene depositato in appositi vasi di carda in modo automatico prima di lasciare il reparto.

4.3 Stiro

I materiali fibrosi in arrivo al reparto stiro richiedono, prima di essere destinati alla filatura, un’operazione di preparazione ulteriore: lo “stiro” ha lo scopo di diminuire la sezione, aumentando nel contempo la lunghezza, dei materiali sottoposti a tale lavorazione. Ne risulta una migliore disposizione delle singole fibre, le quali sono raddrizzate e rese parallele tra di loro, nel senso della lunghezza.

4.4 Filatura

Presso la Filatura Marchi viene impiegato il sistema di filatura *open–end* a rotore, che è, tra i procedimenti non convenzionali, quello più affermato. Il termine tecnico usato per indicare tale metodologia è *break–spinning*.

Ciascuna delle teste dei filatoi è alimentata con un nastro di fibre, risultante dalle lavorazioni precedenti, contenuto in un capiente “vaso di stiratoio” e richiamato da un sistema ad uno o più cilindri.

La filatura è un procedimento che non si innesca spontaneamente, in nessun caso: è necessario avviare il sistema introducendo nella macchina un filato preformato, al quale le fibre iniziano ad unirsi; in poco tempo il meccanismo può continuare ad operare autonomamente.

I due parametri che si controllano, velocità di richiamo del nastro e velocità di torsione delle fibre, caratterizzano il filato prodotto determinando il cosiddetto rapporto *di torsione*.

4.4.1 Separazione

Nella prima fase della lavorazione di filatura avviene una disaggregazione completa del nastro, che risulta nella *separazione* delle singole fibre, ad opera di una *cardína*. Il nastro viene dissolto per consentire all’elemento torcente, mantenuto in rotazione ad una velocità di diverse migliaia di giri al minuto primo, di conferire al filato la torsione prevista. Un ulteriore effetto stirante è prodotto sulle fibre da una potente corrente d’aria prodotta all’interno della testa.

4.4.2 Condensazione

Le fibre, una volta separate meccanicamente e per via pneumatica, sono convogliate in una parte della testa del filatoio detta “zona di condensazione”; i fattori che determinano le caratteristiche peculiari del filato sono il numero di fibre convogliate e la struttura fibrosa risultante; il risultato

finale è determinato anche dalla caratteristiche del *rotore* impiegato, la cui gola interna è profilata in maniera particolare.

All'uscita dal rotore il filato è guidato da un meccanismo automatico che lo avvolge su rocche; un braccio robotizzato provvede a rimuovere le rocche complete, a convogliarle su di un nastro (che le trasporta verso della grosse ceste) ed a rimpiazzarle con delle rocchette vuote.

4.5 Vaporizzo

Il filato, già avvolto su rocche, viene sottoposto per 30' ad un procedimento di stabilizzazione, che si effettua in una camera ipobarica; la depressione facilita la penetrazione del vapore fino ai livelli più interni della rocca. Tale operazione ha lo scopo di eliminare le tensioni residue dovute alla torsione imposta alle fibre cheratiniche dal processo di filatura.

4.6 Imballaggio ed etichettatura

L'imballaggio è completamente automatizzato; mentre le rocche sono inscatolate agli imballaggi vengono apposte etichette che riportano le informazioni più importanti: il numero della partita, il tipo di filato (e la relativa composizione), il titolo ed il colore.

5 L'obiettivo centrale: minimizzare i costi di produzione dovuti alla programmazione

L'esigenza principale della Filatura, comune ad una moltitudine di industrie diverse, è quella di programmare la propria produzione, nel rispetto di una moltitudine di vincoli, nel migliore dei modi possibili. L'operazione è attualmente svolta, non senza qualche problema, da un "dispositore" che sfruttando la sua esperienza e il suo intuito organizza il "lancio" dei diversi lotti

abbinandoli ai filatoi nelle sequenze che ritiene migliori. La vastità dello spazio delle soluzioni e l'oggettiva complessità dell'operazione fanno sospettare che esistano margini di miglioramento.

Lo scopo del modello è quello di ricercare le soluzioni migliori, in termini economici ma non solo, al citato problema della programmazione.

6 Analisi dettagliata dei costi

La programmazione della produzione costituisce un aspetto particolarmente rilevante, dal punto di vista economico: le esigenze che vanno contemperate sono numerose e disparate; l'allontanamento dai parametri di funzionamento ideali finisce sempre con il ripercuotersi sui costi finali del prodotto. L'azienda ha l'esigenza di rispettare una moltitudine di vincoli espressi in forma di parametri, alcuni dei quali possono (talvolta devono) essere violati al prezzo di una produzione meno efficiente (e quindi più costosa). Evidentemente esigenze opposte possono risolversi solo agendo in modo da minimizzare il "danno" totale.

Nel seguito della trattazione compaiono, espressi in termini monetari, dei costi; essi non vanno interpretati tanto come valori assoluti quanto come valori indicativi dell'ordine di grandezza della causa collegata ai costi stessi.

6.1 Ritardo di consegna

Allo stato attuale delle cose, nel momento in cui l'azienda riceve la conferma di un ordine, attraverso uno dei canali esistenti (agenti commerciali, fax, posta elettronica...), è già stata stabilita, in forma di "promessa", una data di consegna. Non sempre è possibile programmare la produzione in modo tale che siano soddisfatte tutte le promesse di consegna; nel caso che una consegna non possa avvenire puntualmente si genera nel cliente una insoddisfazione tangibile, eppure difficilmente quantificabile monetariamente.

In accordo con quanto emerso dai numerosi incontri svoltisi presso l'azienda, ho ritenuto

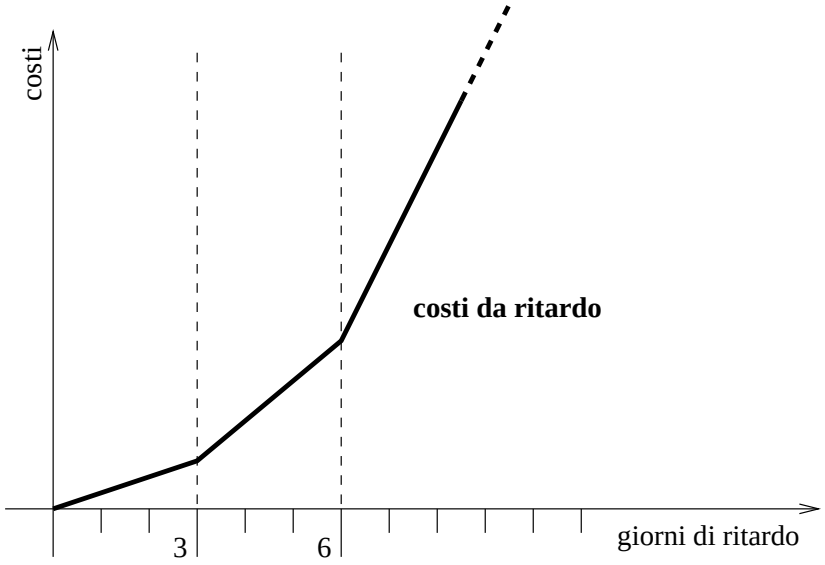

Figura 1: Andamento qualitativo dei costi imputabili a ritardi nelle consegne

opportuno ricorrere all'espediente di associare dei "costi figurativi" a tali eventualità, allo scopo di penalizzare le programmazioni nelle quali le date di consegna eccedano le scadenze stabilite. Si possono calcolare tali costi moltiplicando la dimensione dell'ordine, espressa in chilogrammi, per un "costo giornaliero di ritardo". Come tutte le altre componenti di costo, anche queste sono parametriche, e quindi modificabili discrezionalmente; i valori utilizzati per default, giudicati ragionevoli in sede aziendale, prevedono i seguenti casi:

- ritardo di consegna inferiore ai tre giorni lavorativi ($r \leq 3$): 5 Lit. / giorno / Kg.;
- ritardo di consegna compreso tra i 4 ed i 6 giorni ($4 \leq r \leq 6$): 12.5 Lit. / giorno / Kg.;
- ritardo di consegna superiore ai 7 giorni inclusi ($r \geq 7$): 50 Lit. / giorno / Kg.

Appare evidente come, ferma restando l'intenzione di mantenere i tempi di consegna quanto più possibile entro i termini stabiliti, ai ritardi modesti vengano associati costi poco consistenti; i ritardi più gravi vengono invece sanzionati con costi rilevanti.

6.2 Costi di riattrezzaggio

L'azienda ha la necessità di produrre una grande varietà di filati (nel momento della stesura della tesi il database prodotti ne contiene 200 circa), ciascuno dei quali ha caratteristiche tecniche e merceologiche differenti, su un numero ridotto di filatoi. Questo fa nascere l'esigenza di "attrezzare" i filatoi nella maniera appropriata per la produzione di ogni lotto; le operazioni di attrezzaggio (che prevedono talvolta la sola riprogrammazione, talaltra la sostituzione di alcune parti) richiedono l'intervento di squadre di operai che, intervenendo tempestivamente, cercano di mantenere minimo il tempo di fermo macchina.

I tipi di intervento che possono essere necessari richiedono da un minimo di 60 minuti (che è il tempo fisso di setup, ineliminabile, legato alla necessità di predisporre i contenitori con i semilavorati e di tarare i filatoi) fino ad un massimo di 2 ore e 30 minuti (nel caso peggiore in cui molte componenti dei filatoi debbano essere sostituite).

A ciascuno dei prodotti sono associati particolari parametri tecnici (molti dei quali sono mantenuti riservati), due dei quali incidono sensibilmente sul tempo di riattrezzaggio che intercorre tra la produzione di due lotti successivi: il tipo di "cardína" e di "rotore" impiegato. I filatoi possono infatti utilizzare diverse tipologie di cardíne e rotori per lavorare prodotti dalle caratteristiche diverse: esistono tre diversi tipi di cardíne[2] e cinque tipi di rotori[3]. La sostituzione di tali pezzi è stata valutata in 30' per le cardíne, 60' per i rotori; 90' nell'eventualità che entrambi vadano sostituiti.

È ragionevole tentare di programmare la produzione in modo tale che a susseguirsi sui singoli filatoi siano lotti con caratteristiche (tecniche) simili: in questo modo le sostituzioni di cardíne e rotori sono ridotte al minimo. Questo non sempre è possibile, ovviamente; normalmente si dovrà tenere conto di altri fattori (eventualmente preponderanti) e la sostituzione andrà dunque eseguita. L'azienda ha stimato un costo orario per le operazioni di riattrezzaggio pari a 25.000

[2] I codici usati in azienda (e nel modello) sono "AA", "BB" e "CC".
[3] I codici dei rotori sono "T1", "T2", "S1", "S2" ed "S3".

Lit. per ogni operaio dedicato.

6.2.1 Fermo macchine

Si deve tenere conto, ogniqualvolta un filatoio viene fermato per un riattrezzaggio, che il tempo di inattività trascorso farà slittare la produzione e quindi la consegna di tutti i lotti futuri assegnati al filatoio; questo potrebbe generare ritardi nella consegna, di cui si tiene conto come accennato sopra.

6.2.2 Costo delle squadre di attrezzisti

Un fattore che potrebbe influire negativamente sulla produzione è la concomitanza di riattrezzaggi nello stesso periodo di una giornata lavorativa: le squadre di operai ad essi addetti sono disponibili in numero limitato, ed in certi casi potrebbe richiedersi lo spostamento provvisorio di manodopera addetta da altri reparti, o addirittura un arresto temporaneo della produzione. Per penalizzare le programmazioni nelle quali una tale situazione si dovesse verificare, viene assegnato a questa eventualità un costo figurativo piuttosto elevato. Tale costo viene calcolato in 100.000 Lit. per ogni setup simultaneo eccedente il numero di squadre disponibili.

6.3 Costi di roccatura

I filati vengono consegnati ai clienti su rocchette che hanno caratteristiche peculiari in base all'uso che se ne deve fare: le rocchette hanno normalmente forma troncoconica per i prodotti destinati alla tessitura ed alla maglieria; alternativamente si hanno rocchette cilindriche (destinate alle tintorie).

Le operazioni di setup necessarie affinché un filatoio produca rocchette di genere diverso da quelle per le quali è stato predisposto sono talmente laboriose e costose in termini di tempo che di fatto i filatoi sono stati specializzati una volta per tutte per un tipo specifico di produzione. Allo

stato attuale i filatoi sono distinti secondo questo criterio, che tecnicamente è detto *confezione*, in 12 di tipo M/S (per tessitura/maglieria) e 3 di tipo T (tintoria).

Può accadere che la concomitanza di tempi di consegna particolarmente stringenti e filatoi disponibili predisposti per la confezione "sbagliata" renda necessario produrre un lotto di filato sulle rocchette non adatte alla sua destinazione. Questo comporta, per il cliente, la necessità di trasferire il filato sulle rocchette appropriate (pena un risultato non ottimale - ad esempio una tintura potrebbe risultare imperfetta su rocchette troncoconiche); il costo figurativo relativo a questa eventualità è tarato per penalizzare sensibilmente assegnazioni inopportune, ed è stato quantificato in 350 Lit. per ogni chilogrammo.

6.4 Costi di deriva dal titolo medio

Il magazzino aziendale, pur non interessando direttamente il reparto filatura, per funzionare nel migliore dei modi deve essere sfruttato in modo razionale: il magazzino è stato dimensionato per smaltire le operazioni relative ad imballaggio e spedizione di un determinato numero di colli. Un flusso di filati eccessivo potrebbe rendere impossibile ai magazzinieri sostenere un ritmo di lavoro troppo elevato, saturando il magazzino; similmente una produzione esageratamente ridotta si ripercuoterebbe sul magazzino che, sottoutilizzato, genererebbe dei costi inutili, e sui reparti a monte (carderia e stiro), nei quali si accumulerebbero eccessive scorte di preparato.

Per i reparti a monte in effetti la situazione è ancora più critica (ma al riguardo si veda anche il punto 6.5): in essi il vincolo sulla capacità media dipende dalla portata dei macchinari installati, piuttosto che dal numero di operai disponibili (i quali, costituendo una risorsa più versatile, possono essere all'occorrenza adibiti ad altri compiti o chiamati a cooperare da altri reparti). Una richiesta eccessiva di semilavorati da parte del reparto di filatura non potrebbe essere soddisfatta, bloccando di fatto la lavorazione; la situazione opposta, come accennato sopra, creerebbe seri problemi di stoccaggio dei semilavorati in eccesso.

Il parametro più adatto a misurare l'entità del flusso di prodotti finiti tra il reparto filatura ed

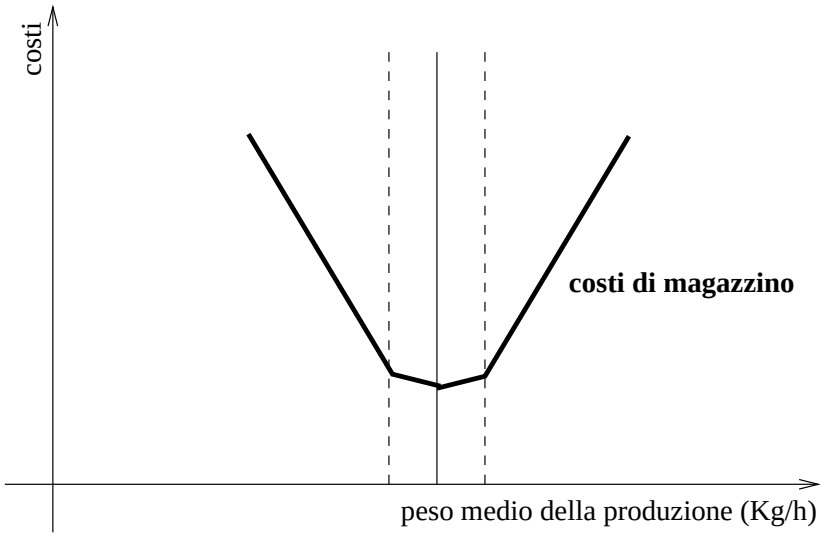

Figura 2: Andamento qualitativo dei costi dovuti alla violazione della regola del titolo medio

il magazzino è il cosiddetto "titolo medio[4]". Il titolo di un filato ne esprime la finezza: quanto più elevato (nel caso del titolo *tex*; al contrario per quanto riguarda il titolo *Ne*) è il titolo, tanto meno fine sarà il filo. Un grande volume di filato a sezione elevata potrà essere prodotto in un tempo relativamente breve, viceversa per filati a basso titolo.

È necessario che il volume di filati che fluiscono dal reparto filatura al magazzino sia il più costante possibile: per ottenere questo risultato è necessario che non siano messi in produzione simultaneamente troppi filati a titolo molto basso o molto elevato, e che la produzione sia sempre bilanciata, in maniera che in media il titolo dei filati prodotti si attesti intorno ad un valore prestabilito. Il dato che si ha a disposizione per ogni filato è la resa oraria in grammi per testa (di filatoio): questo permette di calcolare la produzione oraria e di rapportarla al valore calcolato come ideale di circa 500 Kg. /ora.

Il calcolo viene effettuato su base giornaliera[5], e ad eventuali scostamenti vengono associati

[4]Tra i diversi metodi di titolazione esistenti, il c.d. *titolo universale tex* è espresso come la massa p in g di $l = 1$ Km. di filo: $tex = p/l \qquad p = l \cdot tex \qquad l = p/tex$. Molto usato è anche il *titolo inglese Ne*, che indica il numero di matassine (*hank*) da 840 yards (768 m) necessarie per ottenere la massa di una libbra (453,59 g). [Hoe85, sez. L pag. 151]

[5]La formula utilizzata, che tiene conto anche dei periodi in cui informazioni sui carichi sono sconosciute (fasi

dei costi figurativi calcolati secondo i seguenti criteri:

- scostamenti inferiori ai 50 Kg. / ora in più o in meno ($450 \leq p \leq 550$): 5 Lit. / Kg. / ora di scostamento;
- scostamenti superiori ($p < 450 \vee p > 550$): 20 Lit. / Kg. / ora di scostamento.

6.5 Costi di percentuale colore

Nella catena produttiva a monte del reparto filatura si trovano la mistatura[6] e la carderia: per come sono dimensionate, e per come è organizzato il lavoro, le condizioni ideali si hanno quando le percentuali di mescole preparate (colore chiaro, coloro medio, colore scuro) si equivalgono. Un numero di macchine cardatrici di capacità complessiva uguale è stabilmente assegnato alla

iniziali) od inesistenti (fasi finali), è:

$$\tilde{W}_i = \sum_{j=0}^{23} \left(w_{i,j} \cdot \frac{S}{s} \right)$$

dove, essendo $\tilde{W}_i$ il prodotto totale stimato per il giorno i, $w_{i,j}$ rappresenta il prodotto orario totale espresso in Kg. nel giorno i all'ora j, S il numero totale dei filatoi ed s il numero dei filatoi sui quali si hanno informazioni relativamente all'ora considerata. Il fattore $\frac{S}{s}$ permette di correggere l'informazione relativa al carico tenendo conto della mancanza di informazione relativa ad alcuni dei filatoi. L'assunzione sottostante è che in media il carico sia uniformemente distribuito su tutte le macchine; se 10 filatoi su 15 (i 2/3) in un dato momento producono 300 Kg. si può supporre che il prodotto totale sia di 450 Kg. ($10 : 300 = 15 : x \rightarrow x = 450$). Escludendo dal computo i giorni per i quali non si hanno informazioni sufficienti (la soglia è fissata in 18 ore), si calcola, per prima cosa, il "delta" medio giornaliero:

$$\bar{\Delta}_i = \frac{\left| 24\mathbf{w} - \tilde{W}_i \right|}{24}$$

In cui $\mathbf{w}$ è il carico orario ideale espresso in Kg. Quindi si calcolano i costi giornalieri C_i secondo il seguente criterio:

$$\begin{cases} C_i = \chi_L \cdot 24\bar{\Delta}_i \cdot \frac{h_i}{H} & \text{se} \quad \bar{\Delta}_i < \delta \\ C_i = \chi_H \cdot 24\bar{\Delta}_i \cdot \frac{h_i}{H} & \text{se} \quad \bar{\Delta}_i \geq \delta \end{cases}$$

dove χ_L e χ_H rappresentano il costo al Kg. generato dallo scostamento dal carico ideale $\mathbf{w}$, nei casi rispettivamente di differenza inferiore o superiore alla soglia δ, anch'essa espressa in Kg. Il fattore $\frac{h_i}{H}$, in cui h_i è il totale delle ore lavorate da ciascun filatoio nel giorno i ed H il prodotto tra il numero totale di filatoi e le 24 ore costituenti una giornata lavorativa. La sua funzione consiste nel ridurre l'impatto dei costi al diminuire delle informazioni disponibili al riguardo di ogni giornata: la stima del prodotto totale, mancando dati su alcuni dei lotti in produzione, potrebbe non essere realistica, e pare opportuno abbattere proporzionalmente l'entità del costo generato.

[6] La mistatura è costituita da tre diverse linee, distinte in base alle tonalità di colore gestite e rigidamente separate, a differenza della carderia nella quale è possibile ridistribuire all'occorrenza i carichi di lavoro.

lavorazione di una specifica tonalità di colore: i volumi di colori chiari, medi e scuri prodotti, di conseguenza, si equivalgono.

Nella programmazione va tenuto conto anche di questa peculiarità della catena produttiva: dal momento che tutto ciò che avviene al di fuori del reparto "filatura" è da considerarsi esogeno e non controllabile, si deve agire localmente in modo da non interferire con il buon funzionamento degli altri reparti. I lotti prodotti devono, in un dato arco di tempo, rispettare questo equilibrio, e la programmazione non deve concentrare troppi filati della stessa tonalità (chiara, media, scura) in un periodo breve.

Avendo stabilito che le proporzioni ideali da rispettarsi (il calcolo viene applicato all'arco delle 24 ore) corrispondono proprio ad $1/3$ per ogni tonalità (prodotto chiaro: $p_B = 33.3\%$; prodotto medio: $p_M = 33.3\%$; prodotto scuro: $p_D = 33.3\%$), la formula impiegata per penalizzare gli scostamenti eccessivi calcola la distanza euclidea da tale punto:

$$C_{\%c} = k \cdot \sqrt{(0.333 - p_B)^2 + (0.333 - p_M)^2 + (0.333 - p_D)^2}$$

Il coefficiente k serve a trasformare lo "scostamento" dalla situazione ideale, espresso come un numero puro, in un costo figurato espresso in termini monetari.

Dalle prime prove è risultato che il contributo al totale dei costi apportato dalla voce "costi di percentuale colore" è irrisorio per la quasi totalità dei casi esaminati; si tenga anche nel debito conto l'impatto non trascurabile di questo calcolo sul peso computazionale dell'applicazione. Alla luce di queste considerazioni si è ritenuto accettabile omettere questa componente di costo dal computo totale.

Dalla sperimentazione si è comunque potuto constatare come gli ordini in arrivo siano generalmente ben bilanciati, conformemente alla capacità dei reparti interessati, il che mostra come questi ultimi siano stati correttamente dimensionati. Ovviamente casi straordinari sono sempre possibili, ma in nessun modo la programmazione può ovviare a situazioni simili. L'unica uti-

lità della simulazione in tal caso sarebbe quella di rivelare e quantificare costi generati da una anomala distribuzione degli ordini.

Parte III

Il modello Swarm

7 La programmazione ad oggetti, la modellizzazione mediante agenti e l'ambiente Swarm

Per la realizzazione del modello è stato impiegato l'*Objective-C*, un linguaggio di programmazione che è un dialetto del più noto *C* (che è attualmente il più diffuso e conosciuto dei linguaggi imperativi). L'Objective-C, esaurientemente documentato in [App99], ha una sintassi in parte ispirata a quella dello *Smalltalk-80*, che è il capostipite dei linguaggi ad oggetti.

I linguaggi di programmazione "orientati agli oggetti" (Object-Oriented, "*OO*") impongono all'autore un rigore ed una pulizia che si rispecchiano in progetti, anche molto complessi, leggibili, agevolmente manutenibili e ben strutturati.

Caratteristica della programmazione ad oggetti è la rigorosa distinzione delle diverse componenti funzionali del programma in "entità" separate, ciascuna delle quali comprende variabili accessibili solo *localmente* (proprietà di *encapsulation*, incapsulazione). Lo scambio delle informazioni tra le diverse parti (oggetti) che costituiscono il programma avviene tramite interfacce esplicitamente definite (*metodi*): oggetti diversi interagiscono tra di loro mediante lo scambio di messaggi di questo genere.

La programmazione ad oggetti, in abbinamento alle funzioni messe a disposizione dalle librerie del progetto Swarm, è la scelta tecnica più naturale da adottare per la realizzazione di modelli di simulazione basati su agenti, in cui entità indipendenti interagiscono secondo regole precise, ed il controllo della gerarchia nella quale sono situate è un aspetto fondamentale.

8 Il modello della Filatura Marchi

Realizzare un modello che rappresentasse fedelmente almeno un reparto della Filatura Marchi è stato agevole, una volta che la struttura interna e le relazioni che legano ciascun componente sono state intese a fondo; dopo avere individuato il livello di dettaglio ideale per la rappresentazione, e le semplificazioni ammissibili, è stato possibile realizzare oggetti "software" in grado di rappresentare funzionalmente ciascuno dei componenti identificati come significativi per il processo produttivo dell'impresa.

8.1 Gli agenti e gli oggetti costituenti il modello

Il modello di filatura che è stato realizzato e presentato in questo lavoro è costituito da entità funzionalmente indipendenti che nella gerarchia "orientata agli oggetti" proposta hanno significati diversi.

Alcuni degli oggetti hanno nella realtà un corrispettivo dotato di capacità decisionali: in questo caso si parla di *agenti* veri e propri; altri rappresentano una componente fisica od immateriale della filatura modellizzata, altri ancora hanno mere funzioni di servizio. L'ultima categoria di oggetti inclusi discende dall'adozione di un rigoroso paradigma progettuale che impone di mantenere separati gli agenti e le relative regole di funzionamento (schema ERA, tratto da [GT00]).

8.1.1 Gli oggetti "istituzionali" di Swarm

I modelli realizzati in Swarm prevedono diversi livelli concettuali ai quali gli oggetti si situano ed operano: l'oggetto principale, all'interno del quale si situano tutti i restanti, è `ObserverSwarm`.

L'oggetto `ObserverSwarm` è presente in tutti i modelli Swarm di una certa complessità: lo si può considerare un contenitore all'interno del quale si trovano, allo stesso livello della gerarchia ad oggetti, il "mondo artificiale" che si vuole osservare (costituito da `ModelSwarm` e da tutti gli

oggetti ad esso subordinati gerarchicamente) e gli "strumenti di misura" dello sperimentatore, in forma di oggetti specializzati per la rappresentazione grafica (un esempio dei quali compare in fig. 7, a pag. 78). Un oggetto di tipo `Schedule` permette di gestire il succedersi degli eventi e di effettuare gli aggiornamenti dei grafici in maniera eventualmente asincrona[7] rispetto allo svolgimento degli eventi simulati.

L'oggetto `ModelSwarm`, situato in posizione gerarchica immediatamente inferiore rispetto ad `ObserverSwarm`, rappresenta idealmente il "contenitore" dell'esperimento che si vuole effettuare tramite simulazione.

`ModelSwarm` comprende uno *scheduler* indipendente da quello che opera al livello dell'`ObserverSwarm`, ma ad esso sincronizzato per quanto riguarda ogni singolo ciclo (o "*step*") della simulazione.

8.1.2 Gli agenti e gli oggetti con un corrispondente reale

Gli oggetti software che sono stati realizzati con l'intento di simulare le azioni compiute nella realtà, o di rappresentare un'informazione che entra a fare parte del processo produttivo e al suo interno può liberamente circolare, sono `SpinnerManager`, `Spinner` ed `Order`.

Di queste entità, ciascuna delle quali costituisce una parte indipendente del programma, solo `SpinnerManager` costituisce un agente vero e proprio: la sua funzione è quella di organizzare la produzione assegnando ciascun lotto da produrre (gli oggetti `Order` ne sono la fedele rappresentazione) ad un filatoio (ciascuno dei quali ha un corrispettivo *software* negli oggetti `Spinner`).

Gli oggetti di tipo `Order` incapsulano al loro interno una struttura di dati corrispondente sotto ogni aspetto alle informazioni contenute in un "ordine" di produzione vero e proprio: in forma

[7] La gestione dei grafici è una delle funzioni più costose in termini di tempo macchina, tanto che per sveltire l'esecuzione delle simulazioni spesso si preferisce limitare gli aggiornamenti a periodi di tempo (simulato) distanti tra di loro.

di variabili in ogni `Order` sono memorizzati un numero di serie univoco, un "codice prodotto", il peso in Kg., la data obiettivo di consegna e diversi parametri tecnici.

Similmente ciascun oggetto `Spinner` corrisponde ad uno dei filatoi realmente installati presso l'azienda, ed ognuno presenta le caratteristiche tecniche del filatoio suo corrispondente "vero".

8.1.3 Lo schema ERA e gli oggetti di servizio

Fanno parte del modello numerosi oggetti privi di un corrispondente nella realtà da modellizzare: la loro ragion d'essere deriva dall'esigenza di realizzare software leggibile, manutenibile e soprattutto dalla necessità di seguire un criterio di standardizzazione progettuale. Esempi di tali criteri si hanno nei lavori di Gilbert e Terna ([GT00]) e Gilbert e Troitzsch ([GT99]). Lo schema seguito in questo lavoro è quello proposto in [GT00], che suddivide in più livelli comunicanti il modello: ambiente, agenti, regole, da cui l'acronimo ERA (Environment, Rules, Agents).

Alcuni degli oggetti hanno ruoli esclusivamente di servizio, mettendo a disposizione funzioni utili a più agenti (`TimeFilter`, ad esempio, provvede alla gestione del calendario[8]; l'oggetto `OrderList` funge invece da "polmone" nonché da filtro tra gli ordini provenienti da `DBInterface` e l'agente dispositore (`SpinnerManager`), permettendo a quest'ultimo di disinteressarsi dei dettagli relativi al formato ed alla fonte degli ordini da programmare. `DataWarehouse` mette a disposizione degli altri agenti informazioni sulle caratteristiche dei prodotti (resa oraria, tipo di cardíne e rotori necessari); `SpinnerSetupper` predispone i filatoi, sulla base delle ultime lavorazioni svolte, in modo che la programmazione inizi a partire da uno stato definito.

Altri oggetti, nella fattispecie `SpinnerRuleMaster`, `SpinnerManagerRuleMaster`, `SpinnerManagerRuleMaker`, sono stati realizzati in aderenza allo standard definito in [GT00], che prevede di mantenere rigidamente separati non solo concettualmente, ma anche a livello di codi-

[8] Il formato usato internamente al programma per gestire il tempo reale è, per questioni di praticità, espresso con variabili di tipo `time_t` (dettagli al punto 8.2). I dati relativi a date ed ore ricevuti dall'esterno, al contrario, sono espressi secondo le convenzioni umane: mesi, giorni, ore, minuti; `TimeFilter` mette a disposizione un metodo (`getAbsoluteTime`) che converte tali dati nel formato più opportuno, ed a tale risorsa possono fare ricorso tutti gli agenti.

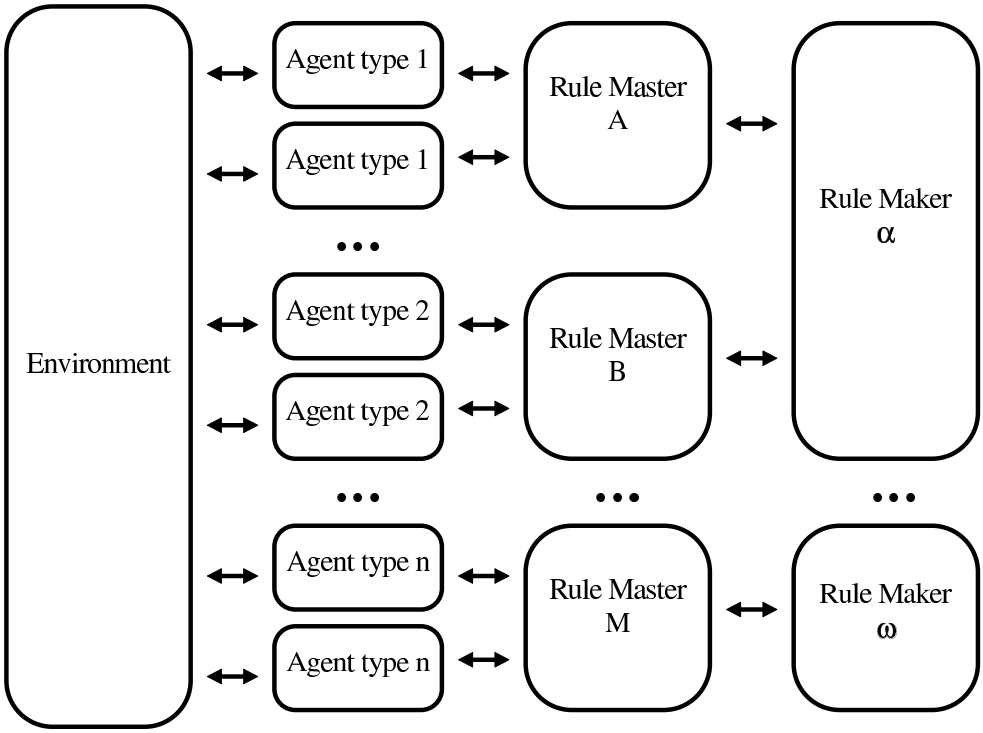

Figura 3: Lo schema Environment–Rules–Agents (da [Ter00])

ce, gli agenti (ed i dati in essi incapsulati) e le regole sulle quali il comportamento degli agenti si fonda. Secondo questa metodologia, chiamata ERA (data “Environment–Agents–Rules”, rappresentata schematicamente in fig. 3), le azioni e le decisioni degli agenti sono determinate da oggetti ad essi esterni (i *Rule Master*): in questo modo la progettazione e lo sviluppo del codice risulta estremamente semplificato, e si ha un guadagno in termini di leggibilità a lungo termine.

Ad un livello concettuale ancora diverso si collocano i *Rule Maker*: le regole nel corso della simulazione possono variare (esistono meta-regole che generano a loro volta regole), ed è compito di questi oggetti comunicare ai Rule Master ad essi abbinati quali siano le regole che questi ultimi devono a loro volta fare seguire agli agenti.

In realtà anche l’oggetto `DataWarehouse` fa parte dello schema ERA, fungendo da contenitore di dati, ma nell’architettura di questo modello non compete strettamente ad un solo agente; sono invece due (`SpinnerSetupper` nella fase iniziale e `DBInterface` durante la programmazione) gli agenti che vi fanno riferimento. Si tratta di un’eccezione che comunque non ne snatura

il significato.

La struttura presentata, proprio per la sua rigidità, presenta indubbi vantaggi di chiarezza, e la sua modularità permette di agire sul comportamento degli agenti intervenendo sulla minor quantità di codice nel modo più sicuro e robusto possibile.

8.2 La gestione del calendario reale: considerazioni

L'introduzione degli ordini ed il setup iniziale dei filatoi si basa sulla possibilità di accedere al database aziendale, che contiene tutte le informazioni necessarie. Il dato meno agevole da gestire, tecnicamente, è il tempo reale nel quale sono collocati tutti gli eventi relativi alla produzione, e sono almeno due le situazioni in cui è stato necessario gestire tali informazioni: date ed ore di inizio e fine lavorazione per quanto riguarda la fase di setup iniziale; data di consegna attesa per quanto riguarda ogni singolo ordine.

Il calendario reale si presta molto male ad effettuare direttamente ed agevolmente operazioni come il calcolo della distanza in ore su archi di tempo che non siano minimi (e che coinvolgano situazioni irregolari come un giorno bisestile od un passaggio ad ora legale). La necessità di gestire correttamente e puntualmente anche situazioni simili è stata soddisfatta ricorrendo a particolari funzioni, formati e strutture di dati messi a disposizione da apposite librerie del linguaggio C[9], facenti comunque parte della dotazione standard.

La procedura estremamente farraginosa che avrebbe richiesto un'operazione tipica come il calcolo del termine di una lavorazione diviene addirittura banale, com'è evidente dall'esempio che segue: un ordine viene messo in lavorazione (al netto dei tempi di setup) alle 16:00 di Martedí

[9]Nella fattispecie si è fatto ricorso alla libreria `time.h`; gli strumenti che essa mette a disposizione semplificano di molto il lavoro quando si debbano conciliare il tempo reale ed il tempo simulato di un modello. Il tempo reale viene opportunamente misurato, nel formato `time_t`, come *"the number of seconds elapsed since 00:00:00 on January 1, 1970, Coordinated Universal Time (UTC)"* ([Pro00, CTIME(3)]); la conversione dal formato "umano" (anno, mese, giorno, ore, minuti, secondi) a `time_t` viene effettuata grazie alla funzione `mktime (struct tm *timeptr)`, l'operazione inversa grazie a `localtime(const time_t *timep)`, che tiene conto anche del fuso orario di appartenenza. Dettagli ed ulteriore documentazione si trovano in [Pro00]. Si veda anche il codice allegato (11.2 e 11.3).

15 Maggio 2001 (`time_t` 989935200); la produzione sarà completata in 41 ore (pari a 147600 secondi). La lavorazione terminerà quindi in `time_t` 990082800 ($989935200 + 147600$), che corrisponde alle ore 09:00 di Giovedí 17 Maggio 2001. Nessuna preoccupazione per i casi di cambio di regime orario o di presenza di un giorno bisestile; nel formato prescelto tali problemi non si pongono affatto.

8.2.1 Il lavoro straordinario e la gestione dei fine settimana

Uno dei problemi che nascono quando il tempo simulato di un modello ed il tempo reale entrano in così stretto contatto come in questo modello è quello delle festività, in particolare quando tali occasioni hanno qualche effetto sulla produzione, che potrebbe venire fermata. L'impiego di lavoro straordinario, che ha effetti importanti sulla durata totale delle lavorazioni, riguarda per l'appunto le festività, ed è quindi opportuno trattare congiuntamente le due questioni, legate molto intimamente. I giorni della settimana interessati sono il Sabato e la Domenica, e le ore che li costituiscono possono essere del tutto, in parte, od affatto lavorative. Il programma usa un parametro apposito[10] per determinare l'ammontare di ore lavorative impiegate nell'arco dei fine settimana. Tale valore entra a fare parte del calcolo che determina il momento in cui, terminata una lavorazione, un filatoio può considerarsi "libero". La possibilità che la lavorazione di un lotto si protragga lungo l'arco di uno o addirittura più fine settimana è gestita da una algoritmo piuttosto sofisticato, che opera in due passaggi.

In figura 4 è rappresentata a titolo di esempio una situazione non banale da gestire: un ordine particolarmente importante la cui lavorazione richiede ben 264 ore (pari ad 11 giorni); ordini di questa entità costituiscono eventualità infrequenti ma reali, ed il diagramma di Gantt ne permette una visualizzazione agevole.

Di seguito sono descritte le fasi che permettono di calcolare con esattezza la "scadenza" dell'ordine; sono previste 12 ore lavorative durante il fine settimana (la prima metà del Sabato):

[10] `weekendWorkingHours`, con range $[0, 48]$

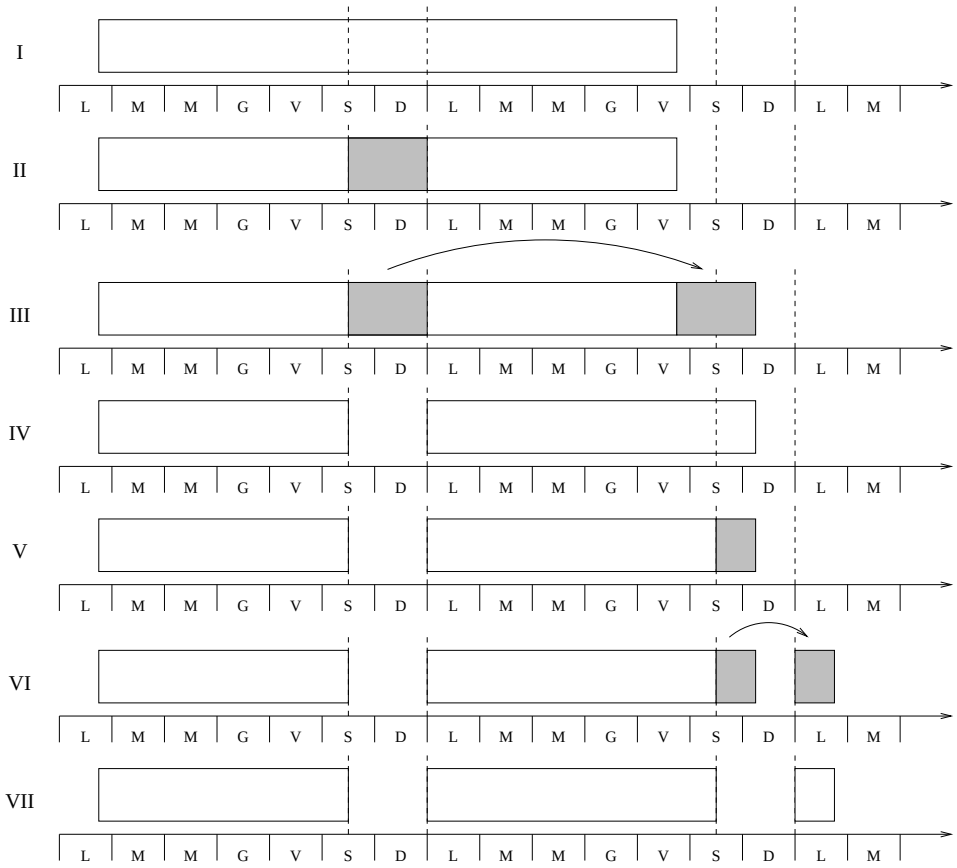

Figura 4: Ordini a lavorazione lunga: il problema dei fine settimana

- *Step I:* L'ordine di 264 ore viene "lanciato" alle ore 18:00 di un Lunedí; la sua scadenza non corretta risulta essere alle 18:00 del Venerdí della settimana successiva, esattamente 11 giorni dopo;

- *Step II:* L'ordine attraversa completamente un fine settimana; 36 ore di lavorazione "saltano";

- *Step III:* Le 36 ore di lavorazione che non si possono svolgere nel fine settimana vengono accodate alla scadenza non corretta dell'ordine;

- *Step IV:* Dopo la prima correzione la scadenza dell'ordine risulta essere alle 06:00 di una Domenica;

- *Step V:* La scadenza della lavorazione non può avvenire in un periodo non lavorativo: altre 18 ore di lavorazione vanno ricollocate;
- *Step VI:* Le 18 ore di lavorazione residue vengono accodate a partire dal primo momento utile;
- *Step VII:* Dopo la seconda correzione la scadenza (reale) dell'ordine risulta essere alle 18:00 del secondo Lunedí successivo.

8.3 La base di dati aziendale

Ottenere l'accesso ai dati aziendali è allo stesso tempo un'operazione delicata e complessa ma fondamentale: al di là delle comprensibili riserve espresse da chi è preoccupato dal rischio di rendere pubbliche informazioni vincolate o segrete, vi sono i tipici problemi informatici. Il sistema informativo aziendale, estremamente sofisticato, è stato realizzato su misura per la Filatura da una società esterna che ne ha messe a disposizione le specifiche.

Il formato dei file usati internamente dal database, che è stato sviluppato in *Clipper*[11], è il non recentissimo ma comune (e piuttosto affermato) *dBase III*. Mi sono state messe a disposizione copie complete di tali file, depurate dei dati ritenuti inopportuni da divulgare, ricavate direttamente dal database server aziendale.

8.3.1 Conversione dei dati

La procedura di importazione di tali dati nel modello, una volta constatata l'assenza di problemi dovuti al formato, ne ha richiesta la trasformazione in un formato il cui accesso fosse agevole

[11] Il Clipper è un linguaggio di programmazione derivato dall'*xBase*, che ne costituisce un sottoinsieme. Per le sue caratteristiche tecniche è particolarmente indicato per lo sviluppo di applicazioni aziendali che devono accedere a basi di dati. L'ampia diffusione di cui attualmente Clipper gode è dovuta anche alla disponibilità di numerosi compilatori distribuiti con licenza di tipo *free*, il più noto dei quali è probabilmente *Harbour* (`http://www.harbour-project.org/`).

tramite le normali funzioni del linguaggio C[12].

Il formato dei dati utilizzato in azienda, seppur molto datato (in termini informatici) è diventato, per le sue caratteristiche di diffusione e robustezza, uno standard industriale piuttosto comune, e non è stato difficile procurarsi documentazione e strumenti adatti a gestirlo.

I file sui quali lavorare, di tipo *dBase III* ad estensione `.DBF`, sono stati filtrati grazie ad un programma reperito sull'internet[13] e trasformati in file ASCII, strutturati con un *record* su ogni linea e separazione dei campi indicata da un carattere convenzionale (#).

In una fase preliminare l'accesso ai file `DBF` avveniva direttamente, appoggiandosi alle funzioni offerte dalla una libreria *ad hoc*[14]; tuttavia la necessità di leggere e modificare frequentemente, anche a scopo di test, i file ha fatto preferire un formato accessibile direttamente e con più immediatezza, quale è l'ASCII.

Il risultato finale dell'operazione preliminare di conversione, effettuata non appena i file originali sono stati resi disponibili dall'azienda, consiste in alcuni file direttamente leggibili, i cui campi sono chiaramente distinguibili.

8.3.2 Formato dei file da sottoporre a parsing

La procedura detta *parsing* ha una base teorica in [Cho59] e si è sviluppata in parallelo con i linguaggi di programmazione; in generale essa permette, applicata ad una stringa, di riconoscerne una struttura, e di suddividerla in componenti elementari, secondo regole definite.

Il programma, durante il suo funzionamento, accede ad alcuni file dai quali vengono ricavate tutte le informazioni necessarie. Di seguito è presentata le struttura ed il significato di ogni

[12] Pur disponendo Swarm di metodi di accesso ai file piuttosto sofisticati, come la gestione nativa del formato `NCSA HDF V.5` od i metodi `getInt` e `getWord` impiegati dalla classe `InFile`, il programma ricorre alla grezza ma versatile funzione `fscanf()` (vedere [Pro00, SCANF(3)])

[13] `dbview-1.0.3`, messo a disposizione e manutenuto da Martin Schulze (`http://www.infodrom.north.de/~joey/Linux/Debian/dbview.html`)

[14] `Xbase DBMS v.1.8.1`, distribuita con licenza GPL v.2 dalla StarTech (`http://www.startech.keller.tx.us/xbase/xbase.html`, progetto ora migrato ad `http://linux.techass.com/projects/xdb/`)

campo; i campi presenti ma vuoti contenevano in origine informazioni riservate e comunque inutili per gli scopi del programma. A fungere da delimitatore tra i diversi campi è il carattere #.

KMAC.txt: Il file nel quale sono elencati i filatoi installati, con le relative caratteristiche tecniche. I campi di interesse sono, nell'ordine e con il relativo numero indicato tra parentesi: descrizione del filatoio (2), numero di teste (3), numero del filatoio (4). Seguono alcune righe di esempio, nelle quali compaiono anche i campi vuoti ed i campi inutilizzati.

```
 3   7#FILATOIO SCHLAFHORST ACO              #120#   7# # # # # # # # #I#S# 7#
 3   8#FILATOIO SCHLAFHORST ACO              #120#   8# # # # # # # # #I#S# 8#
 3   9#FILATOIO SCHLAFHORST ACO              #168#   9# # # # # # # # #I#S# 9#
```

KMPROD1.txt: I diversi tipi di filato prodotti dalla Filatura sono elencati in questo file; da ciascun *record* sono ricavati i campi che seguono: codice filato (1), resa oraria (4), tipo di cardína (10), tipo di rotore (11). Alcuni dei campi, come già spiegato, contengono informazioni inutilizzate. Il file si presenta così:

```
1/054150       # 55.0# 60# 550.0# 60#  0.0#  0#  15.00# 390#AA#S1#
1/380200/  11  # 50.0# 60# 258.0# 60#  0.0#  0#  20.00#   0#AA#S1#
1/750260       #  0.0#  0# 137.5# 60#  0.0#  0#  26.00# 800#AA#S1#
```

KMGANTT.txt: Il modello, prima di iniziare ad assegnare nuovi ordini ai filatoi, esegue un *setup* preliminare: i filatoi devono trovarsi in una condizione coerente con quella reale, a partire dalla quale programmare la produzione. Nel file in questione sono elencati i lotti "lanciati", che si trovano già in produzione; al termine delle relative lavorazioni i filatoi si troveranno in uno stato preciso, in termini di cardíne e rotori montati; è necessario anche calcolare il momento in cui ciascun filatoio sarà libero di iniziare una nuova lavorazione.

I lotti saranno programmati, a partire dai momenti calcolati, sui filatoi attrezzati in funzione dell'ultima lavorazione effettuata.

I campi presi in considerazione sono: data di fine lavorazione (1), ora di fine lavorazione (2), filatoio interessato (6), codice prodotto (10), reparto di competenza (11). In fase di setup preliminare, così come durante la programmazione vera e propria, si cerca di abbinare al codice prodotto (campo 10) un codice prodotto corrispondente, presente nel database dei prodotti (cfr. file KMPROD1.txt). Non sempre si ha identità tra i codici, e si deve ricorrere ad una logica più raffinata[15]. È necessario verificare anche il reparto di competenza di ciascun ordine, dal momento che il database aziendale tiene traccia della "produzione in corso" in tutti i reparti in questa stessa tabella. I lotti con numero di reparto diverso dal 3 vengono ignorati dal modello.

Seguono alcuni *record*, trascritti a titolo di esempio:

```
#20010221#13:06#20010218#00:08#     #   9#          #   85.00#5869/1  /MO#1/802135/6189  # 3#6400C8#
#20010221#21:57#20010218#03:14#     #  11#          #   92.00#5869/1  /MO#1/802135/6189  # 3#6400C8#
#20010224#16:23#20010219#22:52#     #  13#          #  110.00#5868/1  /MA#1/250150/ 263  # 3#6400C8#
```

KMPIAN1.txt: I lotti di filato che devono essere prodotti sono elencati in questo file; le caratteristiche di ogni ordinazione sono descritte nei 6 campi utilizzati degli 11 disponibili. I campi interessanti sono: data di consegna (1), Kg. da produrre (4), filatoio (5), partita (da cui si ricava la *confezione*[16]) (7), codice prodotto (9), reparto (10).

Il campo relativo al "filatoio" indica, quando contiene un numero, che l'ordine si trova già in lavorazione (ed è duplicato nel file `KMGANTT.txt`), e che quindi può essere ignorato. Il

[15] I codici dei prodotti sono formati da tre campi numerici separati da *slash*, ed hanno strutture del tipo `x/yyyyyy/zzzz`. L'operazione di *matching* vero e proprio compete all'oggetto `DataWarehouse`, che esegue una ricerca in due passaggi: durante il primo viene verificata l'eventuale corrispondenza perfetta tra il codice richiesto ed i codici registrati (ricerca di *eccezione*); fallendo questa, viene attuato un secondo passaggio in cui si cerca un codice corrispondente almeno nei primi due valori (`x/yyyyyy`, ricerca *generica*). Nel caso che nemmeno il secondo passaggio abbia successo l'elaborazione si ferma e viene segnalato un errore di prodotto sconosciuto, cosa che può essere dovuta ad un errore nell'immissione dei dati od alla necessità di aggiornare la tabella dei prodotti.

[16] Con il termine *confezione* si indica il tipo di rocchetta sulla quale il filato viene avvolto e consegnato. Per gli scopi di questo lavoro è sufficiente specificare che esistono due diversi tipi di rocchetta: rocchetta troncoconica (destinata alla filatura) e rocchetta cilindrica (per lavori di tintura). Dal codice della partita si può ricavare il tipo di confezione (è indicata dal decimo carattere, "T", "M" o "S").

numero di reparto dovrebbe essere sempre il 3, che indica il reparto filatura; è comunque controllato per rilevare eventuali errori nei dati.

La struttura del file è visibile nell'estratto che segue:

```
#20010225#        #      0.00#   1056.00#    #    0#5564/1  /MA#           #1/250150/ 296  # 3#     # #
#20010225#        #      0.00#   2400.00#    #    0#5568/1  /MA#           #1/250150/ 384  # 3#     # #
#20010219#20010321#      0.00#   5244.00#    #    0#5605/1  /SQ#5605/1  /00#1/001100/4400  # 3#     # #
```

`PROGRAM.txt` [impiegato nei soli modi `1/GA` e `2/file`]: Una possibile programmazione di *n* ordini identificati da un numero univoco può essere descritta univocamente impiegando *n* coppie ordinate di numeri interi (a,b), dove *a* è il numero dell'ordine e *b* il numero del filatoio sul quale l'ordine viene prodotto. L'ordine in cui diversi lotti vengono prodotti su di un unico filatoio dipende dall'ordine delle coppie in cui tali ordini compaiono: nel caso $(1,1)(2,2)(3,1)$ sul primo filatoio saranno prodotti gli ordini 1 e 3 in questo ordine, l'ordine 2 sul filatoio 2. Al contrario, l'ordine 3 sarà lavorato in anticipo sull'ordine 1, sempre sul filatoio 1, ad esempio nel caso $(2,2)(3,1)(1,1)$. Evidentemente tutte le combinazioni possibili, in tutte le sequenze possibili, possono essere rappresentate con questa notazione; alcune rappresentazioni possono essere differenti ma equivalenti a livello di risultato: $(3,1)(1,1)(2,2)$ è perfettamente equivalente a $(2,2)(3,1)(1,1)$.

Il file `PROGRAM.txt` contiene una coppia di valori per ogni linea, e le linee sono tante quanti gli ordini la cui programmazione si vuole rappresentare. Nel file seguente 5 ordini sono assegnati in un ordine precisamente determinato a 2 filatoi; le lavorazioni avverranno nelle sequenze $1-3-4$ e $2-5$ rispettivamente sul filatoio 1 e 2.

```
1 # 1
2 # 2
3 # 1
5 # 2
4 # 1
```

Un file PROGRAM.txt può essere preparato dall'utente (allo scopo di esaminare i risultati di una programmazione effettuata da un programmatore umano) o creato direttamente dal programma, che nel modo di funzionamento "1/GA" può salvare a richiesta dell'utente la migliore programmazione ottenuta fino a quel momento[17].

La modalità di funzionamento 2 prevede per l'appunto la lettura e l'esecuzione della programmazione codificata nel file, con calcolo dei relativi costi.

8.4 Il ruolo cruciale dell'agente `SpinnerManager` (dispositore)

La programmazione della produzione compete al dispositore, che ha nell'agente SpinnerManager il suo corrispettivo simulato. L'operazione consiste nell'assegnare ad uno dei filatoi disponibili ciascuno dei lotti che è stato richiesto di produrre, nell'ordine ritenuto più opportuno; è necessario seguire criteri organizzativi ragionevoli: si è mostrato quante diverse tipologie di costo dipendano dalla programmazione, e come tali costi possano incidere sensibilmente sui costi totali di produzione. Lo scopo della programmazione consiste proprio nel cercare di mantenere al minimo i costi sulle cui cause si ha un qualche controllo.

Il processo decisionale seguito dal dispositore aziendale è tutt'altro che banale da descrivere, tanto più che qualunque tentativo di esplicitare in forma di regole la strategia seguita non ha avuto successo. Si tratta di un tipico caso di conoscenza implicita e non esplicitabile; si tornerà sull'argomento poco oltre.

L'implementazione in software del comportamento dell'agente dispositore–spinnerManager è andata evolvendosi passando attraverso diverse fasi. Inizialmente la versione simulata del dispositore è stata realizzata nel modo più semplice (ma anche più irrealistico) concepibile: l'agente - cioè - operava in modo casuale. Un primo tentativo - fallito - di simulare un comportamento

[17] Il programma mette a disposizione il metodo printBestEver per salvare la migliore programmazione trovata (nel file convenzionalmente chiamato PROGRAM_test.txt per evitare accidentali sovrapposizioni ad un eventuale file PROGRAM.txt preesistente). L'accesso a tale funzione è possibile, durante un *run* in modalità 1/GA, aprendo una sonda sull'agente GMRuleMaker (pulsante "Create a GM RuleMaker Probe" sul pannello probeGenerator, cfr. fig. 10). La sonda che viene visualizzata comprende il pulsante printBestEver.

realistico avrebbe dovuto basarsi sull'applicazione di un insieme di regole, tali da permettere di percorrere un albero decisionale in tempi ragionevoli, arrivando a soluzioni per lo meno sensate.

Il metodo che si è rivelato vincente ha richiesto il ricorso agli algoritmi genetici (AG) ed un certo stravolgimento dell'architettura del modello.

Per garantire la possibilità di confrontare risultati "sperimentali" e risultati "reali" è stato introdotto anche un metodo di funzionamento guidato, che si basa su scelte già programmate.

8.4.1 Il tentativo fallito di realizzare un sistema a regole

Un requisito fondamentale per ottenere una rappresentazione fedele del comportamento del dispositore è la conoscenza dei processi che lo guidano nelle sue scelte. L'ipotesi sottostante era che il dispositore, nel suo operato, seguisse delle regole più o meno complesse; applicando tali regole, e seguendo una qualche strategia, il dispositore dovrebbe essere guidato verso le scelte migliori e ottimizzanti la produzione, nel rispetto dei diversi vincoli imposti.

Una volta esplicitata in qualche modo la strategia seguita dal dispositore, noto il sistema di regole applicate ed un'eventuale "funzione di valutazione", si sarebbe potuto applicare uno dei collaudati algoritmi decisionali usati dai programmi scritti per competere in giochi di strategia come la dama, il backgammon, gli scacchi. In [Dew84] e [Fel97] si sono incontrati numerosi spunti che sarebbero stati utili ad adattare le logiche dei programmi di gioco (generatore di mosse, valutatore di scacchiera, procedure *minimax*) al problema della disposizione degli ordini.

Rappresentando lo spazio delle scelte possibili in forma di albero decisionale si sarebbe avuta la possibilità di percorrerne i rami più promettenti, valutando ciascuna scelta in base alle regole estrapolate dal dispositore umano, fino a raggiungere le soluzioni più valide.

Sfortunatamente ci si è resi conto che l'operato del dispositore umano non segue delle regole chiaramente esplicitabili; ci si è scontrati con un tipico problema di *conoscenza tacita*, in cui le azioni compiute non sono state del tutto razionalizzate; a maggior ragione non esiste la possibilità

di formalizzarle né di includerle in un algoritmo finalizzato alla ricerca di soluzioni ottimali mediante la loro applicazione.

8.5 Modalità di funzionamento del modello

Il modello, nato inizialmente con il solo scopo di rappresentare fedelmente una parte del processo produttivo della Filatura, avrebbe dovuto permettere almeno la simulazione di situazioni difficilmente riscontrabili nella realtà, o non convenienti, o comunque troppo costose da mettere in pratica con fini puramente speculativi.

Disporre di un modello realistico dell'azienda offre la possibilità di effettuare, a costi irrisori, verifiche ed esperimenti di vario genere: si pensi alla possibilità di verificare l'effetto dell'acquisto di un nuovo filatoio sui ritardi di consegna, o viceversa di osservare quanto grave potrebbe essere l'impatto di un evento catastrofico come la simultanea rottura di più macchine sull'organizzazione. La possibilità di influenzare il funzionamento del modello attraverso una moltitudine di parametri offre la possibilità di verificare la sensibilità della performance aziendale a situazioni non banali da esaminare in pratica: il ricorso ad ore di straordinario avrebbe avuto sui tempi di consegna ricadute sufficientemente importanti da giustificarlo? Ha senso modificare uno dei filatoi perché possa lavorare con rocchette di tipo diverso? Anche situazioni al limite della realtà o del tutto irrealistiche possono essere studiate, grazie ad un modello del genere: un importante aumento dei costi orari delle squadre di attrezzisti, la presenza di clienti che esigono consegne puntuali ad ogni costo (situazione che si può simulare gonfiando artificialmente i costi "da ritardo"), una drastica riduzione della capacità dei magazzinieri a far fronte al normale carico di lavoro dovuta alle cause più varie.

La modalità di funzionamento del modello si può controllare modificando all'avvio del programma il valore di default (0) che compare nella casella `managerMode` sul pannello `ModelSwarm`. I valori accettabili sono 0, 1, 2; valori inesistenti causano l'interruzione dell'esecuzione.

8.5.1 Modalità 0: *programmazione random*

Nella prima fase ci si è dunque accontentati di fare operare un'azienda simulata in modo non realistico: ad un comportamento deterministico di tutte le componenti "fisiche" si contrapponeva l'azione casuale dell'unico agente "logico" (conformandosi alla terminologia usata in [LTS96, LS00b, SLS98] ed introdotta nella parte iniziale di questo lavoro).

Il risultato è stato sufficiente a verificare il corretto funzionamento del meccanismo di *"message passing"* che governa l'interazione di tutti gli oggetti software. La modalità di funzionamento, che in realtà riguarda non tanto l'intero modello quanto l'agente dispositore, prevede da parte dell'oggetto `SpinnerManager` la richiesta all'oggetto `DBInterface` di trasmettere il primo ordine presente in coda e l'assegnazione di tale ordine ad uno qualsiasi dei filatoi disponibili[18].

L'obiettivo principale del lavoro svolto fino a questo livello di sviluppo era quello di ottenere la massima stabilità possibile dal programma, in vista degli sviluppi futuri. La complessità del progetto e la quantità di codice scritto hanno richiesto numerosi e prolungati test, per ridurre al minimo la possibilità di ottenere risultati errati: aumentando la complessità del progetto la ricerca di errori diviene esponenzialmente più difficoltosa.

Le verifiche effettuate hanno previsto un esame dei comportamenti di ogni componente del modello alla ricerca di anomalie od incongruenze, operazione facilitata dalla possibilità di consultare un *log* nel quale ogni operazione viene registrata (dettagli alla nota 30 del par. 9) e dalla disponibilità di un doppio meccanismo di generazione degli ordini[19].

[18] L'assegnazione casuale si basa sulla generazione di numeri interi casuali distribuiti uniformemente sull'intervallo $[1, nS]$, dove nS è il numero totale dei filatoi disponibili.

[19] L'oggetto `DBInterface` normalmente funge, come il nome stesso suggerisce, da interfaccia verso la base di dati aziendali: gli ordini vengono letti, decodificati e messi a disposizione, immagazzinati in una lista, nell'attesa che vengano richiesti dall'oggetto `OrderList`, immediatamente successivo nella gerarchia del modello. È però possibile attivare un generatore casuale di ordini da utilizzare a scopi diagnostici, disponendo di ordini reali in quantità limitata. Tale funzione si può attivare tramite il pannello di controllo `DBInterface` (cfr. 12), la cui comparsa è controllata dal pulsante `"Create a DBI Probe"` presente sul pannello `probeGenerator` (fig. 10; il metodo `toggleFakeOrdersGenerator` consente di impiegare alternativamente ordini reali letti da file od ordini generati casualmente

8.5.2 Modalità 1: *programmazione GA*

L'integrazione del modello della filatura e dell'algoritmo genetico GM (dettagli al punto 8.6) è risultata in un potente sistema in grado di ricercare le soluzioni migliori al problema dell'assegnazione di un insieme di ordini.

Eseguendo ogni volta una programmazione diversa, ottenuta dalla decodifica di una stringa binaria proposta dall'AG, si riduce l'intero modello della filatura ad una funzione con innumerevoli parametri che fornisce in uscita un numero reale. Tale risultato misura in un certo senso la bontà della programmazione effettuata, valutando i "costi totali" ad essa associati descritti analiticamente al punto 6 e successivi.

Il programma, una volta avviato in modalità 1, inizia, attraverso l'evoluzione operata dall'AG, la ricerca delle programmazioni migliori. La ricerca può essere interrotta in qualsiasi momento, essendo la convergenza delle soluzioni verso valori simili un buon indicatore della bontà del risultato ottenuto.

Esiste la possibilità di visualizzare la migliore soluzione ottenuta fino ad un determinato momento e di memorizzarla in un file utilizzabile nella modalità 2, presentata nel punto seguente: la procedura è presentata nella nota 17 a pag. 52.

8.5.3 Modalità 2: *programmazione da file*

Allo scopo di confrontare le programmazioni ottenute dal programma (random od ottimizzate) con le programmazioni decise dal dispositore umano è stata prevista la possibilità di leggere da un file una programmazione e di farla eseguire al modello, che forza il dispositore simulato ad eseguire la programmazione impostagli.

Il formato del file, descritto a pag. 51 al punto `PROGRAM.txt`, è piuttosto semplice e si riduce ad un elenco ordinato dei lotti, ciascuno dei quali è abbinato ad un filatoio; tanto i lotti quanto i filatoi sono identificati dal numero univocamente assegnato a ciascuno di essi.

8.6 Il motore evolutivo: Genetic Manipulator

La realizzazione di un modello della filatura Marchi, per quanto dettagliato, non avrebbe potuto considerarsi riuscita senza che la programmazione della produzione avvenisse in modo realistico. Attivando il modello in modalità "GA" (codice 1) il meccanismo che controlla l'intero sistema risulta profondamente modificato: il modello aziendale viene a dipendere nel suo funzionamento dal motore evolutivo, che si assume la conduzione delle fasi di assegnazione degli ordini e di calcolo dei costi. In un certo senso l'intero modello della filatura si può considerare un singolo agente, e molteplici copie di tale agente vengono usate in parallelo per esplorare lo spazio delle soluzioni da parte di Genetic Manipulator (GM).

L'algoritmo genetico usato in questo programma, GM per l'appunto, è stato realizzato e messo a disposizione da Gianluigi Ferraris (si rimanda a [Fer00] e [Fer01] per approfondimenti). Gli oggetti che lo costituiscono e che sono stati inclusi nel mio programma sono `GMRuleMaker`, `Embryo` ed `Interface`; nella classificazione usata sopra ricadono nell'insieme degli "oggetti di servizio" (8.1.3).

8.7 Gli algoritmi genetici (cenni)

In natura l'evoluzione degli organismi ha luogo grazie all'applicazione di una regola semplice e rigorosa: gli organismi inadatti all'ambiente in cui si vengono a trovare muoiono; quelli adatti, o comunque dotati di un qualche vantaggio competitivo, sopravvivono più a lungo e si riproducono. Come è suggerito in [RN98, pag. 667], *"Si dà il caso che ciò che è buono per la natura lo è anche per i sistemi artificiali"*: gli algoritmi genetici applicano ad un insieme iniziale di "individui" una selezione ed un meccanismo di riproduzione, in modo che gli individui più "adatti" (nel senso descritto più rigorosamente poco oltre) si evolvano di generazione in generazione.

Cercando di superare l'analogia zoologica–biologica, si possono vedere delle corrispondenze tra "individui" e "soluzioni", tra "riproduzione biologica" e "regole genetiche di riproduzione".

L'adeguatezza di un organismo a sopravvivere nell'ambiente in cui è immerso corrisponde all'adeguatezza di una soluzione a risolvere il problema; alle generazioni di individui si sostituiscano insiemi di soluzioni in competizione per massimizzare una funzione: l'affinità tra le due situazioni è a questo punto evidente.

A misurare la bontà di una soluzione provvede la funzione di *fitness* (funzione di adeguatezza in italiano): "dipende dal problema, ma in ogni caso è una funzione che prende in ingresso un individuo e restituisce un numero reale in uscita" ([RN98, pag. 668]); l'algoritmo genetico ricerca l'individuo che massimizzi tale funzione.

La corrispondenza individuo–soluzione è evidente nella rappresentazione dell'individuo come stringa[20] : la stringa è formata da uno o più *geni*, a ciascuno dei quali corrisponde un elemento che può essere uno stato logico, un numero naturale, la rappresentazione di un numero reale (dettagli alla nota 21).

Le affinità tra la selezione naturale e la strategia di selezione propria degli algoritmi genetici sono altrettanto evidenti: negli AG la selezione avviene in modo stocastico ma guidato, e le probabilità che gli individui hanno di riprodursi sono proporzionali alla loro *fitness*.

Anche il meccanismo di riproduzione è evidentemente ispirato a ciò che accade in natura: la generazione di "discendenti" avviene infatti per incrocio e mutazione. L'incrocio, che avviene tra due individui scelti a caso tra quelli selezionati precedentemente, dà luogo ad individui il cui patrimonio genetico è una combinazione di quello degli individui "genitori". Questa operazione, detta *crossover*, una volta scelto un *punto di incrocio* a caso spezza le stringhe in due parti, una iniziale ed una finale, e le ricombina scambiandole: la procedura è mostrata in fig. 5.

La mutazione, che consiste nella modifica casuale di un carattere della stringa, serve ad aggiungere variabilità alla popolazione: normalmente il tasso di mutazione viene mantenuto molto basso, in modo che l'evoluzione non ne sia eccessivamente disturbata. Analogamente a quanto avviene per gli esseri viventi una occasionale mutazione nel patrimonio genetico può portare

[20]Nell'approccio classico si usa l'alfabeto binario costituito dalle cifre $\{0,1\}$; quando la rappresentazione è più complessa si parla di *programmazione evolutiva*.

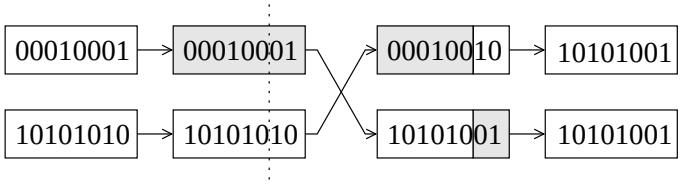

Figura 5: *Crossover* di stringhe binarie: la linea tratteggiata indica il punto di incrocio.

l'individuo colpito ad essere selezionato per l'eliminazione, ma in rari casi può determinare un vantaggio. È questo uno dei meccanismi che permettono alle specie viventi di adattarsi all'ambiente sviluppando caratteristiche originali, e che fanno in modo che l'algoritmo genetico esplori molte zone dello spazio delle soluzioni, evitando di finire intrappolato in un massimo locale.

Una definizione sintetica ma completa, tratta da [DG88] (in [BMT96]), descrive gli algoritmi genetici come

> *"[...] procedure di ricerca probabilistiche progettate per funzionare su spazi ampi che comprendono stati che si possono rappresentare come stringhe. Questi metodi sono intrinsecamente paralleli, dal momento che utilizzano un insieme distribuito di campioni dello spazio (una popolazione di stringhe) per generare un nuovo insieme di campioni"*.

Le caratteristiche fondamentali degli algoritmi genetici, presentate sempre in [BMT96], sono quattro, e consistono in:

- Codifica dei parametri: i parametri oggetto dell'ottimizzazione, negli AG, devono essere codificati in forma di una stringa di caratteri, solitamente binari. Esempio: il numero naturale 7 si può rappresentare come 0111 in una codifica a 4 bit. I problemi di rappresentazione di numeri reali si possono altrettanto facilmente superare[21].

[21] Ad esempio un intervallo continuo come $[-10,10]$ può essere campionato con risoluzione grande a piacere: ricorrendo ad un intero $i \in [0,255]$ rappresentato da una stringa di 8 bit, $x \in [-10,10]$ può assumere uno dei 256 valori risultanti dall'operazione $x = ((i/255) \cdot 20) - 10$: $\{-10, -9.921, -9.843, -9.764 \ldots 9.921, 10\}$. All'aumentare del numero di bit utilizzato cresce la raffinatezza della rappresentazione e la precisione della soluzione che si può ottenere.

Più parametri possono essere rappresentati come concatenazione delle loro codificazioni binarie: due parametri con valori 7 (0111) e 2 (0010) costituiscono una stringa di 8 bit: 01110010.

- Parallelismo: Gli algoritmi di ottimizzazione standard esplorano lo spazio delle soluzioni un punto per volta; gli AG, al contrario, prendono in considerazione una quantità di punti distribuiti in regioni diverse dello spazio di ricerca delle soluzioni; i vantaggi in termini di efficienza di esplorazione sono evidenti.

- Informazioni necessarie al calcolo: Gli AG necessitano solamente della funzione obiettivo: utilizzando il valore della funzione corrispondente a ciascuno dei punti del dominio esplorati orientano la ricerca senza che sia necessario calcolare nessuna derivata. Questo rende ammissibile il ricorso a funzioni obiettivo discontinue, non differenziabili, non numeriche.

- Regole di transizione: Contrariamente a quanto normalmente accade nell'algoritmo standard di discesa del gradiente, l'esplorazione dello spazio delle soluzioni procede stocasticamente. L'insieme delle soluzioni potenzialmente buone in un certo stadio contribuisce a determinare l'insieme di "candidati" presi in considerazione nello stadio successivo. La selezione avviene sulla base di un'estrazione casuale con probabilità dipendente dal contributo di ciascun soggetto alla soluzione obiettivo. Non si verifica una salita monotonica dei gradienti; può benissimo verificarsi un salto all'indietro verso punti con valori peggiori della funzione obiettivo. L'algoritmo in questo modo può peggiorare la sua performance di breve termine al fine di esplorare, in tempi più lunghi, regioni dello spazio delle soluzioni più promettenti, evitando di rimanere intrappolato in punti di massimo locale.

8.8 La codifica binaria della programmazione

Gli AG operano su stringhe che rappresentano degli "stati" possibili; sottoponendo tali stati alla valutazione della funzione di *fitness* accade che le stringhe corrispondenti agli stati maggiormente

	Ordine 1	Ordine 2	Ordine 3	Ordine 4	Ordine 5
Filatoio 1	0	1	0	0	0
Filatoio 2	1	0	1	0	0
Filatoio 3	0	0	0	1	1

Tabella 1: Matrice filatoi–ordini

desiderabili (nel senso di "aventi un valore della funzione di fitness maggiore"), combinandosi ed occasionalmente subendo una mutazione casuale, generino "stati" sempre migliori, o soluzioni sempre migliori al problema oggetto di studio.

Un'operazione estremamente delicata è l'ottenere la corrispondenza tra le soluzioni possibili e le stringhe che costituiscono ciascun individuo della popolazione soggetta ad evoluzione, vale a dire "mappare" gli stati possibili sulle stringhe rappresentative.

Si è detto che le stringhe fatte evolvere dall'algoritmo genetico in questo lavoro sono di tipo binario, costituite dai soli simboli $\{0,1\}$. Si tratta di una scelta dettata in parte dal desiderio di non travalicare i limiti dell'ortodossia (i testi di riferimento sono [Hol75] e [DG88]), ed in parte emersa come soluzione più ragionevole dopo una serie di tentativi che, per approssimazioni successive, hanno portato alla decisione ritenuta alla fine soddisfacente.

Segue una breve rassegna di tutte le codifiche che sono state prese in considerazione nel corso del lavoro.

8.8.1 Una codifica ortodossa: valori booleani; ordinamento filatoi/ordini ed ordini/filatoi

Nel tentativo di aderire nel massimo grado alla semantica "binaria" degli AG, il primo tentativo di codifica è consistito nel rappresentare in una matrice di ordini e filatoi le associazioni desiderate, come rappresentato nella tabella 1.

La matrice è formata da numeri binari che indicano l'assegnazione (1) o meno (0) di ciascun ordine ad ognuno dei filatoi disponibili: nella matrice mostrata come esempio l'ordine 2 compete al filatoio 1, gli ordini 1 e 3 al filatoio 2, gli ordini 4 e 5 al filatoio 3.

Tale codifica impiega n bit di informazione ($n = filatoi \cdot ordini$), 15 nel caso specifico; rappresentando la matrice come un vettore formato dall'accostamento delle relative righe ne risulta la stringa seguente:

0	1	0	0	0	1	0	1	0	0	0	0	0	1	1

La correttezza formale di tale rappresentazione è rigorosamente rispettata: ciascun ordine è stato programmato, e nessuno è stato duplicato (in ciascuna delle colonne della matrice compare uno ed un solo 1). Individuare le stringhe non valide perchè viziate da una situazione del genere è un'operazione semplice e veloce, anche riferendo tali termini ad un contesto informatico.

La soluzione, apparentemente accettabile, presenta però un vizio sostanziale: viene persa l'informazione relativa alla sequenza delle lavorazioni. Continuando a fare riferimento al caso presentato, è evidente come non sia possibile rappresentare la situazione nella quale, ad es., gli ordini 1 e 3 siano eseguiti sul filatoio 2 in ordine diverso dal loro ordine naturale (numerico); la situazione è analoga per quanto riguarda gli ordini 4 e 5.

L'ordine di lavorazione a livello di singoli filatoi, particolarmente quando sono coinvolti numerosi ordini ed i tempi di consegna acquisiscono importanza cruciale, riveste un ruolo fondamentale; la primitiva codifica appena presentata deve considerarsi inaccettabile, e non è quindi utilizzabile per gli scopi di questo lavoro.

Una codifica in un certo senso simmetrica a quest'ultima prevede la rappresentazione dei filatoi in colonne e degli ordini in righe (la situazione è l'esatto opposto della della precedente: cfr. la tabella 2). Sono impiegati bit nello stesso numero, ma resta il problema relativo all'impossibilità di specificare un ordine di esecuzione per ciascun filatoio. Anche questa possibilità è stata scartata.

	Filatoio 1	Filatoio 2	Filatoio 3
Ordine 1	0	1	0
Ordine 2	1	0	0
Ordine 3	0	1	0
Ordine 4	0	0	1
Ordine 5	0	0	1

Tabella 2: Matrice ordini–filatoi

8.8.2 Una codifica (meno ortodossa) con numeri naturali impacchettati

Se si accetta di lasciare cadere la condizione (eccessivamente stringente) di rappresentare delle sole variabili booleane e si introduce la codifica di numeri naturali in forma di (sotto)stringhe binarie, si può costruire una stringa formata dalla successione dei numeri degli ordini.

Si ipotizzi di voler gestire un massimo di 255 ordini: ciascuno dei numeri naturali corrispondente ad un ordine può essere codificato in un pacchetto di 8 cifre binarie (bit), escludendo la sola sequenza [0000.0000] (il punto centrale è inserito per facilitarne la lettura), che sarebbe associata alla condizione di “macchina ferma”.

Rifacendosi ad uno degli schemi matriciali già presentati, ad es. “filatoi/ordini”, è possibile rappresentare anche l’informazione relativa alle sequenze di “lancio” degli ordini. Si costruisca, in analogia alla programmazione precedentemente proposta, una matrice simile (in cui si è omessa la stringa [0000.0000] per questioni di leggibilità):

	Ordine 1	Ordine 2	Ordine 3	Ordine 4	Ordine 5
Filatoio 1		[0000.0010]			
Filatoio 2	[0000.0001]		[0000.0011]		
Filatoio 3				[0000.0100]	[0000.0101]

Si muti il significato delle colonne della matrice, attribuendo alla posizione in colonna di ogni ordine un significato di “priorità” nella produzione. È necessario disporre di un numero di colonne pari al numero totale degli ordini (per tenere conto del caso limite in cui tutti gli ordini

sono associati ad uno stesso filatoio). Le caselle vuote possono essere ignorate; nella situazione rappresentata nella matrice seguente tutti gli ordini sono idealmente scivolati a sinistra. Le caselle vuote sono sistematicamente ignorate, quindi la situazione è in tutto e per tutto analoga alla precedente.

	1°ordine	2°ordine	3°ordine	4°ordine	5°ordine
Filatoio 1	[0000.0010]				
Filatoio 2	[0000.0001]	[0000.0011]			
Filatoio 3	[0000.0100]	[0000.0101]			

È del tutto evidente che questa programmazione permette di rappresentare correttamente anche uno specifico ordinamento da seguire per la produzione, a livello di singolo filatoio.

Per quanto riguarda il secondo filatoio è possibile invertire l'ordine di lavorazione, a titolo di esempio: gli ordini 1 ([0000.0001]) e 3 ([0000.0011]), la cui sequenza di lancio non era stata finora specificata (meglio, era stata definita l'unica possibile per le primitive programmazioni precedenti), possono essere programmati come sopra (prima l'ordine 1, quindi il 3):

Filatoio 2	[0000.0001]	[0000.0011]	…

ovvero nell'ordine inverso (prima il 3, quindi l'1):

Filatoio 2	[0000.0011]	[0000.0001]	…

La matrice, completa di tutti i campi, si presenterebbe come segue:

	1°ordine	2°ordine	3°ordine	4°ordine	5°ordine
Filatoio 1	[0000.0010]	[0000.0000]	[0000.0000]	[0000.0000]	[0000.0000]
Filatoio 2	[0000.0001]	[0000.0011]	[0000.0000]	[0000.0000]	[0000.0000]
Filatoio 3	[0000.0100]	[0000.0101]	[0000.0000]	[0000.0000]	[0000.0000]

La stringa relativa a tale rappresentazione, formata da $n = 120$ bit ($n = filatoi \cdot ordini \cdot bpo$[22]) sarebbe "000000100000000000000000000000000000000000000010000001100000000000000000-0000000000000100000001010000000000000000000000".

Si noti come la stringa associata anche ad un caso volutamente semplificato come questo (5 ordini per 3 filatoi) abbia una lunghezza notevole: un caso realistico quale potrebbe essere il problema di assegnare 150 ordini a 15 filatoi richiederebbe l'impiego di stringhe di ben 18.000 bit[23], evidentemente al di là del limite della trattabilità, tenuto conto delle risorse informatiche a disposizione (e forse esistenti).

8.8.3 Una codifica eterodossa: stringhe alfanumeriche

L'ultimo abbozzo di soluzione al problema della codifica, presentato sopra, ha un grave (e per ora insormontabile) limite nell'utilizzo di stringhe dalle proporzioni mastodontiche. In alcune applicazioni ([Dol00], o nella *programmazione genetica* in generale) si è visto come sia possibile ricorrere a stringhe alfanumeriche (ad esempio basate su di un alfabeto a 256 caratteri come l'estensione ad 8 bit dell'ASCII).

Affrontare il problema in questo modo, operazione discutibile prima di tutto dal punto di vista dell'ortodossia metodologica, riducendo in effetti la lunghezza delle stringhe di quasi un ordine di grandezza[24], potrebbe dare l'impressione di rendere il problema più trattabile dal punto di vista informatico. In realtà, la mancata disponibilità di software collaudato, e da questo la necessità di dover implementare da zero un algoritmo genetico alfanumerico da utilizzare in questo lavoro, ha fatto scartare anche questa possibilità.

[22] *bpo* (bits per order) indica il numero di bit necessari ad identificare univocamente ogni ordine: ferma restando la necessità di riservare una combinazione alla situazione di "macchina ferma", si ottengono 7 bit per 127 ($2^7 - 1$), ordini, 8 per 255 ($2^8 - 1$)...

[23] I bit necessari a rappresentare 150 ordini sono 8 ($2^7 - 1 = 127 < 150 < 2^8 - 1 = 255$), da cui si calcola $150\,ordini \cdot 15\,filatoi \cdot 8\,bit = 18.000$.

[24] L'impiego di caratteri alfanumerici appartenenti al succitato insieme "ASCII-esteso", 256 in tutto, avrebbe permesso di rappresentare 8 caratteri della stringa binaria originaria con un solo carattere. La ingestibile stringa da 18.000 bit dell'esempio precedente sarebbe ridotta a "soli" 2.250 caratteri, lunghezza tutto sommato trattabile.

8.8.4 Altre proposte per una codifica delle soluzioni

Il Dott. Ferraris ha proposto una codifica estremamente compatta, che prevede che ogni filatoio sia rappresentato da un pacchetto di bit. Quattro cifre binarie sono ad esempio sufficienti a rappresentare (gestire) fino a 16 macchine. La posizione di ciascuno di questi pacchetti nella successione interna alla stringa rappresenterebbe l'ordine corrispondente, *in una lista già ordinata in base alle priorità degli ordini*.

Ordine 1				Ordine 2				...				Ordine 200			
0	0	0	1	1	1	1	1	...	...	...	...	0	0	1	1

Nell'esempio l'ordine 1 competerebbe al filatoio 1 (in binario 0001), l'ordine 2 al filatoio 15 (1111), fino all'ordine 200, "lanciato" sul filatoio 3 (0011).

Come già si è accennato, questo tipo di codifica ha un grande vantaggio nel ridotto consumo di bit in rapporto alla quantità di informazioni che può rappresentare. Nel plausibile caso dell'assegnazione di 200 ordini a 15 filatoi si otterrebbe una stringa di soli 800 bit.

La fantastica parsimonia che caratterizza questa codifica, però, si sconta nel fatto che la "lista già ordinata" degli ordini purtroppo non esiste, ne può esistere. Uno dei gradi di libertà del dispositore consiste proprio nella possibilità di modificare la sequenza di "lancio" degli ordini, tanto a livello globale che di singolo filatoio, al fine di ottenere buone performance. È proprio cambiando e ricambiando sia di posizione (ossia di filatoio) che di priorità gli ordini che il dispositore tenta di minimizzare i costi di riattrezzaggio. La codifica appena descritta, prevedendo un "preordinamento" degli ordini limita le possibilità di scelta del dispositore in un aspetto troppo importante, cosa che la rende, purtroppo, inutilizzabile.

La codifica che segue è chiaramente ispirata a quella proposta dal Dott. Ferraris; l'idea è quella che, fermo restando il significato "posizionale" dei pacchetti di bit (il cui ordine di riferimento si ricava dalla posizione all'interno della stringa), si assegnino ad ogni ordine due distinte caratteristiche: filatoio di assegnazione (come nella codifica Ferraris) e priorità.

Gli ordini si susseguono lungo la stringa in base al loro numero progressivo (che non ha niente a che fare con la priorità), e compaiono due pacchetti di bit per ogni ordine. 4 bit sono sufficienti a distinguere fino a 16 filatoi, 8 bit bastano per 256 ordini. La lunghezza della stringa risultante per 200 ordini e 15 filatoi (caso standard già considerato sopra) sarebbe di 2.400 bit ($(4+8)\cdot 200\,bit$).

Ordine 1		Ordine 2		Ordine 3		...		Ordine 200	
priorità	filatoio	priorità	filatoio	priorità	filatoio	...	...	priorità	filatoio

Il concetto di priorità in questo concetto assume un significato “locale”: la precedenza di un ordine su di un altro si verifica a livello di singolo filatoio. Se gli ordini 1, 2, 3 e 4 hanno priorità rispettive 250, 255, 1 e 2 non si può dire nulla sull’ordine del loro lancio se non sapendo che sono assegnati, per es., al filatoio 1 gli ordini 1 e 2, al filatoio 2 gli ordini 3 e 4. In questo caso si avrebbe la sequenza $1-2$ sul filatoio 1, $3-4$ sul filatoio 2.

Alternativamente, assegnando gli ordini 1 e 3 al filatoio 1 e gli ordini 2 e 4 al filatoio 2, le sequenze risulterebbero essere $3-1$ (filatoio 1) e $4-2$ (filatoio 2). Nella non frequente eventualità in cui due ordini “lanciati” sullo stesso filatoio abbiano la stessa priorità si avrebbe però una stringa dal significato ambiguo, che dovrebbe essere eliminata dalla popolazione assegnandole fitness nulla.

È inoltre necessario disporre almeno di tanti gradi di libertà quanti sono gli ordini, per poter tenere conto dell’irrealistico caso limite in cui tutti gli ordini venissero assegnati ad un solo filatoio. I 256 possibili livelli di priorità assunti nell’esempio ci permetterebbero quindi di lavorare con un massimo di 256 ordini.

La codifica appena presentata garantisce la possibilità di rappresentare univocamente ogni assegnazione possibile, in ogni sequenza possibile, assicurando al dispositore la libertà di azione che lo caratterizza nella realtà.

Sfortunatamente anche quest’ultima codifica presenta alcune caratteristiche non del tutto desiderabili: in effetti esistono intere famiglie di stringhe che a genotipi diversi fanno corrispondere

fenotipi (programmazioni) identici, e questo non è un problema particolarmente grave; più seria è la difficoltà collegata alla ricerca di stringhe inaccettabili perchè aventi priorità duplicate a livello di singolo filatoio. La procedura di ricerca e di eliminazione sarebbe lunga e costosa in termini di risorse computazionali.

Il Prof. Pietro Terna ha proposto un'ulteriore sistema di codifica, nel quale si rappresentano, in successione, pacchetti di bit che indicano il numero dell'ordine ed il filatoio di assegnazione; la priorità in questo caso emerge naturalmente dalla sequenza in cui gli ordini compaiono, codificati, all'interno della stringa:

1°Ordine		2°Ordine		3°Ordine		...		200°Ordine	
# ordine	filatoio	# ordine	filatoio	# ordine	filatoio	...	...	# ordine	filatoio

Si tratta senz'altro di una soluzione semplice e leggibile, la stringa è interpretabile con immediatezza, l'informazione in essa contenuta è completa e permette di rappresentare correttamente la priorità degli ordini.

Sfortunatamente una codifica strutturata in questo modo presenta un grave difetto di ordine pratico, piuttosto che formale o concettuale: il rapporto tra il totale delle stringhe che è possibile costruire ("stringhe possibili") **S** e le stringhe accettabili (nel senso definito poco oltre) S_a è piuttosto basso, e diminuisce molto rapidamente al crescere del numero degli ordini trattati.

Il requisito fondamentale che una stringa deve soddisfare per essere considerata "accettabile", nel senso di "rappresentare una delle possibili soluzioni del problema" è che ogni ordine (il numero che lo rappresenta) compaia una ed una sola volta in una delle posizioni disponibili nella stringa, e che di conseguenza compaiano tutti gli ordini, nessuno escluso, e che nessun ordine sia duplicato.

Il rapporto $\frac{S_a}{\mathbf{S}}$ tende velocemente allo zero[25] quando si inizi ad aumentare appena il numero degli ordini gestiti, ben prima di avvicinare valori interessanti per questo lavoro.

[25]Esempi di individui accettabili, in un caso ipersemplificato che contempla soli 3 ordini, sono: $(1*2*3*)$, $(1*3*2*)$, $(2*1*3*)$, nei quali ai numeri di filatoio (non interessanti in questo contesto) è stato sostituito un simbolo $(*)$. Stringhe inaccettabili sarebbero invece $(1*1*2*)$, $(1*2*2*)\ldots$, nelle quali è evidente come certi ordini siano duplicati ed altri mancanti. È possibile conoscere il rapporto tra il primo ed il secondo tipo di stringhe ricorrendo al

Nella teoria il funzionamento dell'AG non sarebbe influenzato da una situazione simile, dal momento che l'esplorazione parallela stocastica dello spazio delle soluzioni interesserebbe, prima o poi, le poche stringhe accettabili.

Nella pratica d'altra parte ciò comporterebbe una dilatazione insostenibile dei tempi di ricerca: gli eventuali, pochi individui accettabili che dovessero comparire nella popolazione molto difficilmente potrebbero dare luogo, per mutazione o crossover, ad individui parimenti accettabili. Sarebbe un evento estremamente raro, a meno di impiegare popolazioni numerosissime (e dunque rallentare ancora di più l'elaborazione). In questa situazione si assisterebbe con ogni probabilità solo alla fortuita generazione occasionale di individui accettabili, che potrebbero avvantaggiarsi di un miglioramento solo in casi molto poco probabili. Il tutto si ridurrebbe ad una ricerca stocastica a forza bruta, essendo di fatto inutili ed inapplicabili le regole evolutive tipiche dell'AG.

Si è anche pensato di delegare all'AG stesso il compito di selezionare le stringhe inaccettabili per l'eliminazione, considerandole come programmazioni imperfette (errori del dispositore), ed associando loro ingenti costi che le avrebbero dovute penalizzare: un ordine programmato più volte avrebbe rappresentato uno spreco di materie prime e di risorse aziendali, mentre ad un ordine omesso dal programma di produzione sarebbe corrisposto un gravissimo ritardo nella consegna. Modificare il modello affinché potesse gestire tali eventi anomali avrebbe comportato modifiche estremamente importanti e delicati interventi sul codice già esistente. Simili operazioni fortunatamente non sono state necessarie per il motivo che si è trovata una soluzione che non

calcolo combinatorio: se si continua a trascurare l'elemento "filatoi", che non ha importanza a questo riguardo, noto come N il numero di ordini da lanciare, si ottiene un totale di N^N permutazioni possibili. Di queste permutazioni abbiamo definito come accettabili esclusivamente quelle *senza* ripetizione degli ordini, che sono "solo" $N!$.

Partendo dal caso precedente di soli 3 ordini si avrebbero $3^3 = 27$ permutazioni con ripetizione possibili ($\mathbf{S}$); di queste solo $3! = 6$ sarebbero accettabili (S_a) come soluzione al problema della programmazione dei 3 ordini. Una percentuale del $6/27 = 22\%$ sembra tutto sommato ragionevole, ma si osservi l'andamento di tale valore all'aumentare degli ordini:

4 ordini:	$\mathbf{S} = 4^4 = 256$	$S_a = 4! = 24$	$\rightarrow$	$S_a/\mathbf{S} \simeq 9\%$
5 ordini:	$\mathbf{S} = 5^5 = 3125$	$S_a = 5! = 120$	$\rightarrow$	$S_a/\mathbf{S} \simeq 4\%$
10 ordini:	$\mathbf{S} = 10^{10} = 10000000000$	$S_a = 10! = 3628800$	$\rightarrow$	$S_a/\mathbf{S} \simeq 0.03\%$

La percentuale di individui accettabili crolla vistosamente, divenendo infinitesima poco oltre la decina di ordini.

presenta tale problema.

Nel paragrafo 8.8.5 si mostrerà infatti come, grazie ad un'intuizione del Dott. Ferraris, sia stato possibile conservare ed applicare quest'ultima codifica, sfruttandone tutti gli aspetti positivi ed aggirando il problema dell'esiguo rapporto tra stringhe accettabili ed inaccettabili.

8.8.5 La codifica prescelta

La soluzione definitiva al problema della codifica in stringhe binarie delle possibili soluzioni al problema è il risultato dell'integrazione della codifica "base" proposta dal Prof. Terna e di un'accorta gestione delle liste degli ordini suggerita dal Dott. Ferraris.

Nell'esposizione che segue si farà riferimento ad un caso molto semplice, nel quale 4 bit sono impiegati per rappresentare un massimo di 16 ordini (10 nell'esempio), mentre 2 bit bastano per 3 filatoi.

L'ultima delle codifiche descritte, già rappresentata nella tabella di pag. 68, diventa perfettamente utilizzabile avendo l'accortezza di interpretarla in modo originale: il numero che veniva associato a ciascun ordine deve essere letto come "posizione dell'ordine" in una lista con caratteristiche peculiari.

La "lista degli ordini" contiene un elenco degli ordini, ciascuno dei quali è rappresentato come al solito da un numero (cfr. la prima colonna sulla sinistra in fig.6). La stringa binaria, una volta interpretata e ridotta ad una sequenza di coppie di numeri (posizione ordine, numero filatoio) viene usata per selezionare gli ordini dalla suddetta lista secondo un meccanismo semplice ma potente: facendo ancora riferimento alla fig. 6, si supponga di incontrare come prima coppia i valori $(2,2)$: l'ordine 2 risulta assegnato al filatoio 2. Prima di proseguire però la lista subisce una modifica: l'ordine appena assegnato (il 2) viene eliminato dalla lista e tutti gli ordini successivi scivolano verso l'alto di una posizione. La coppia di numeri successiva $(9,1)$ permette di chiarire che, dal momento che i membri della lista diminuiscono di un'unità ad ogni lettura, può capitare un valore corrispondente ad una posizione non occupata da alcun ordine,

perchè successiva all'ultimo ordine ancora presente in lista. Un tale valore viene corretto facendolo corrispondere all'ultimo ordine presente: all'inesistente ordine in posizione 9 viene fatta corrispondere l'ultima posizione disponibile, in questo caso la 8, che è occupata dall'ordine numero 9. È importante tenere presente la distinzione tra numeri di ordine e posizioni nella lista. La seconda coppia, in conclusione, associa al filatoio 1 l'ordine 9. La terza coppia che compare $(4, 1)$ indica una posizione nella lista in cui è effettivamente contenuto un ordine (l'ordine 5 nella posizione 4) e lo associa al filatoio 1.

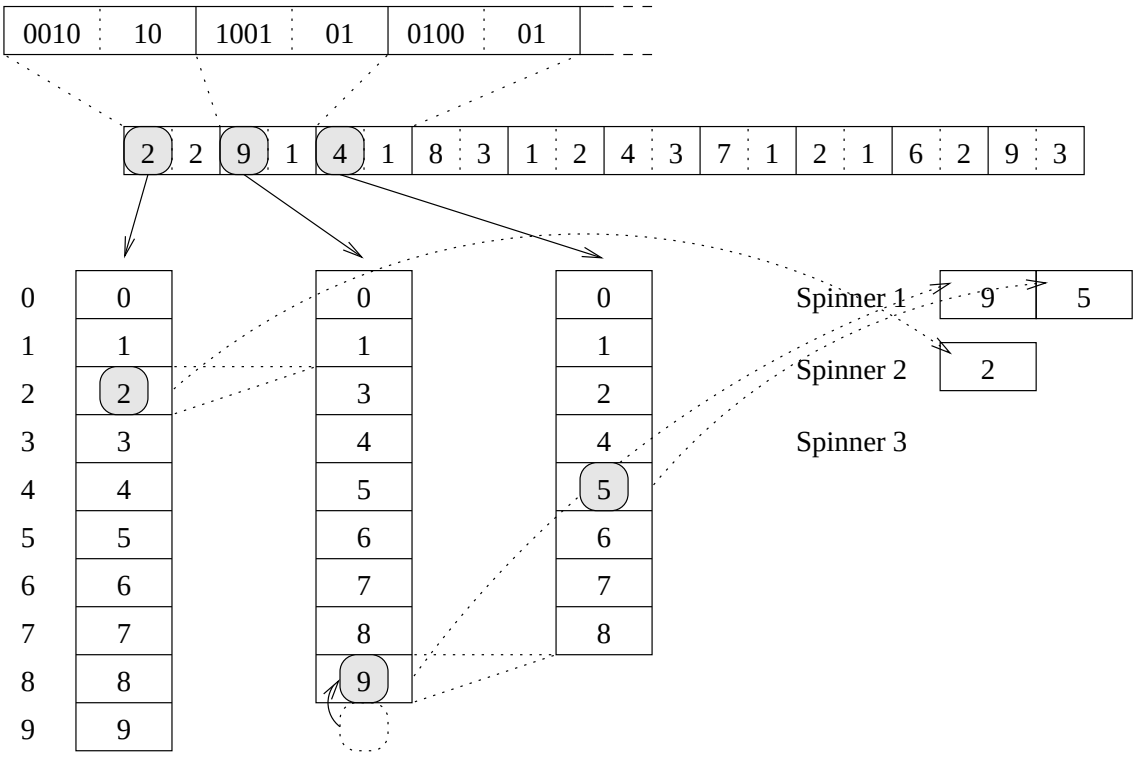

Figura 6: La codifica binaria e la lista dinamica degli ordini

L'applicazione di questo meccanismo permette di ottenere sempre programmazioni formalmente corrette (senza duplicazioni né omissioni di ordini), non esistono stringhe inaccettabili ed è possibile calcolare la *fitness* di ogni individuo, a differenza di quanto accadeva con stringhe "malformate" a cui era associata automaticamente una fitness nulla.

codice		filatoio
00	$\rightarrow$	0
01	$\rightarrow$	1
10	$\rightarrow$	2
11	$\rightarrow$	**2**

Tabella 3: Mappatura di 3 filatoi su 2 bit

codice		ordine
0000	$\rightarrow$	0
0001	$\rightarrow$	1
0010	$\rightarrow$	2
...		
1000	$\rightarrow$	8
1001	$\rightarrow$	9
1010	$\rightarrow$	**9**
...		
1110	$\rightarrow$	**9**
1111	$\rightarrow$	**9**

Tabella 4: Mappatura di 10 ordini su 4 bit

Tanto nel caso degli ordini quanto al riguardo dei filatoi, la necessità di rappresentarli solo in potenze di due (a causa della programmazione binaria) presenta l'inconveniente delle combinazioni in soprannumero: i 2 bit usati sono sufficienti a rappresentare 4 filatoi, mentre ne esistono solo 3. La soluzione ovvia (impiegata anche sulle liste di ordini i cui membri diminuiscono progressivamente) è quella di interpretare ogni valore maggiore del numero dei membri effettivamente presenti con il valore dell'ultimo membro presente in lista. Nella tabella 3 è presentato il caso dei 3 filatoi, numerati da 0 a 2.

Analogamente, per i 10 ordini citati (numerati da 0 a 9) la situazione è quella che compare in tabella 4.

Un simile risultato si ottiene senza dover rinunciare alla più parsimoniosa delle codifiche, che per rappresentare 200 ordini assegnati a 15 filatoi necessita sempre di soli 2.400 bit.

8.9 I parametri relativi all'AG

Il programma permette di inserire i parametri di funzionamento dell'AG desiderati, così come di modificarli nel corso dell'esecuzione, attraverso i pannelli di controllo disponibili[26].

La parte di software che svolge, nella modalità `1/GA`, la funzione di evolvere le soluzioni migliori applicando un algoritmo genetico è basata su GM, un modello *Swarm* completo di cui si è sfruttato il *kernel*, presentato in [Fer01][27]. GM è stato concepito fin dall'inizio per essere facilmente integrabile in altri progetti, o per funzionare autonomamente alla ricerca di soluzioni ottime a problemi di ogni genere; gli algoritmi genetici sono infatti apprezzati per la vastità delle applicazioni in cui si possono rivelare utili.

Segue una breve descrizione dei parametri principali impiegati dal programma ed attinenti il funzionamento dell'algoritmo genetico, che sono anche quelli canonici per ogni AG (tra parentesi quadre compare il *range* dei valori ammissibili).

`numberOfInterfaces` [$\mathbb{N}$]: fissa il numero di stringhe generate; esse costituiscono la popolazione di indivudui che sono fatti successivamente evolvere. Ad un maggiore numero di individui corrisponde una maggiore variabilità della popolazione. La scelta di questo parametro è critica, e deve tenere conto di aspetti inconciliabili come il tempo di elaborazione e la precisione del risultato. Il valore di default, 20, permette di ottenere rapidamente risultati ragionevoli per insiemi di ordini non troppo numerosi.

`turnoverRate` [$\mathbb{R}_{[0,1]}$]: controlla quanta parte degli individui è eliminata da ciascuna ge-

[26] Tutti i parametri possono essere cambiati prima dell'avvio della simulazione, rispetto al *default*, agendo sul pannello `ModelSwarm`, che è sempre visualizzato (vedere fig. 9) automaticamente; nel corso della simulazione le modifiche operate tramite tale pannello sono ininfluenti (per una scelta progettuale legata all'architettura di *Swarm*). È invece necessario per prima cosa visualizzare il pannello `GMRuleMaker` (cfr. fig. 13) agendo sul pulsante "`Create a GM RuleMaker Probe`" del pannello `Probe Generator` (fig. 10), e quindi intervenire sui valori visualizzati nel pannello appena comparso. Alcuni dei parametri relativi all'AG devono tuttavia essere impostati una volta per tutte all'inizio, e non compaiono dunque in questo pannello.

[27] Nel lavoro di Ferraris è indicata la possibilità di reperire GM agli indirizzi seguenti: `http://www.swarm.org/` (nella sezione dei contributi degli utenti) ed `http://eco83.econ.unito.it/swarm/`

nerazione e sostituita da nuovi individui nella generazione successiva. Il valore di default è 0.5.

`crossoverRate` [$\mathbb{R}_{[0,1]}$]: stabilisce la percentuale di individui che saranno interessati da operazioni di *crossover*. Default 0.5.

`mutationRate` [$\mathbb{R}_{[0,1]}$]: indica il tasso di mutazione a cui sono sottoposti gli individui: è la probabilità che ogni singolo bit cambi di stato. Default 0.001 (che risulta in media in una mutazione ogni 1.000 bit).

Altri parametri, specifici di GM ed utilizzati dal programma sono:

`evolutionFrequency` [$\mathbb{N}$]: Indica ogni quanti *step* di simulazione debba avere luogo un'evoluzione della popolazione di stringhe. In questo modello ha senso mantenere sempre il valore di default 1.

`childrenFitness` [$\mathbb{R}$]: come spiegato in [Fer01, pag.12] si tratta di un'innovazione nel campo degli AG; è possibile stabilire un valore iniziale di *fitness* da assegnare ai "nuovi nati". La funzione è disattivata per default.

`useDeltaFitness` [$\{0,1\}$]: anche questa modalità di funzionamento, attivata per default (1), costituisce un'innovazione, ed è anch'essa dovuta a G. Ferraris. La selezione degli individui migliori, che normalmente avviene in base al valore assoluto della fitness, può, ed in questo caso è estremamente opportuno, basarsi sulla differenza tra ciascuna delle fitness e quella dell'individuo peggiore di ogni generazione. Se ne ottiene un guadagno in termini di precisione e velocità di convergenza dell'AG, ed è permesso l'impiego di valori negativi per la fitness[28].

[28] È estremamente utile poter lavorare con fitness negative: in questo programma la fitness corrisponde per l'appunto al costo totale associato ad ogni programmazione (valore per sua natura sempre positivo), cambiato di segno. Le fitness con valore maggiore sono infatti quelle associate ai costi minori: un costo pari a 80 è preferibile ad un costo pari a 100; l'AG, che tenta sempre di *massimizzare* le fitness, valuterà di conseguenza migliore la fitness pari a -80 del primo caso rispetto al valore -100 del secondo, che corrisponde ad un costo più elevato.

`bitsPerSpinner [`$\mathbb{N}$`]:` nel codificare ciascuna programmazione in una stringa binaria si utilizza normalmente, per indicare ogni filatoio, un numero di bit che deve bastare a definirne ciascuno univocamente[29]. Questo parametro indica quanti bit impiegare per la rappresentazione dei filatoi, tenendo conto della loro numerosità.

`bitsPerOrder [`$\mathbb{N}$`]:` il parametro è del tutto analogo al precedente, con la differenza che si riferisce ai bit da impiegare per la rappresentazione degli ordini.

9 Conclusioni

Non appena lo sviluppo del modello è giunto al termine e si è ottenuta dal programma una stabilità soddisfacente, è stato possibile effettuare delle prove di funzionamento impiegando insiemi di ordini reali, ottenuti dalle basi di dati aziendali. Si sono misurati e messi a confronto i risultati di programmazioni effettuate casualmente (modo di funzionamento `0/Random`), di programmazioni ottimizzate grazie all'algoritmo genetico (modo `1/GA`) e di programmazioni studiate da un dispositore umano (modo `2/File`). Sono stati valutati i costi della produzione di diversi insiemi di ordini, in numero variabile tra la ventina ed alcune centinaia.

Durante l'esecuzione ed al termine di ogni programmazione il software realizzato presenta un resoconto dettagliato[30] dal quale si possono estrapolare tutte le informazioni relative al *run* in corso.

I dettagli relativi ai costi generati sono visualizzati a cura dell'agente `SpinnerManager`, e le relative informazioni sono evidenziate all'interno del *log* dall'intestazione `[SpM]` giustapposta

[29] Questo programma lascia all'utente la possibilità di non impiegare necessariamente il valore che sarebbe naturalmente più appropriato (la minima potenza di due maggiore del numero), permettendo anzi interessanti sperimentazioni. Si è notato, ad esempio, come anche utilizzando bit in soprannumero si possa giungere a soluzioni soddisfacenti, anche se a discapito dei tempi di calcolo.

[30] Il programma prevede la possibilità di visualizzare una moltitudine di informazioni diagnostiche sul suo funzionamento direttamente sullo schermo o di reindirizzarle su di un file. La quantità di notizie messe a disposizione ed il loro livello di dettaglio può essere deciso in fase di compilazione del programma, agendo sul file `Macro.h` e "commentando" o "definendo" le linee relative a ciascun agente, come evidenziato all'interno del file stesso (cfr. 11.1).

ad ogni linea. Qui di seguito compare un esempio di resoconto di programmazione, estratto da un tipico file di *log*.

```
[SpM] ------------------------------------------------------
[SpM] Total costs from resulting production program:   40518.80
[SpM] ------------------------------------------------------
[SpM] Details:
[SpM] - costs from setups:                               625.00
[SpM] - costs from delayed deliveries:                 30919.22
[SpM] - costs from wrong packaging:                     8080.80
[SpM] - costs from wrong average output:                 893.78
[SpM] - costs from overlapping setups:                     0.00
```

Nello stralcio riportato si possono distinguere le diverse voci che concorrono a formare i costi totali: nell'ordine, e dopo il totale, si notano i costi di riattrezzaggio (cfr. il punto 6.2), i costi da ritardo nelle consegne (cfr. 6.1), i costi da errata confezione (cfr. 6.3), i costi da deriva del titolo medio (cfr. 6.4), i costi da interferenza nei riattrezzaggi (cfr. 6.2.2).

Com'era prevedibile le programmazioni casuali sono state quelle che hanno dato i risultati peggiori, generando sempre i costi più elevati.

Eseguire simulazioni facendo ricorso a delle programmazioni prestabilite, e decise da un dispositore umano, ha fatto misurare dei costi sensibilmente inferiori: in particolare il dispositore umano ha la capacità di azzerare facilmente i costi "di confezione", applicando una semplice regola.

In questo aspetto è particolarmente evidente la differenza tra il modo di operare di un essere umano e la ricerca di un ottima da parte di un AG: mentre il dispositore umano può assegnare correttamente senza alcuno sforzo i lotti per tintoria ai filatoi appositamente attrezzati (e fare altrettanto con i lotti da filatura), percependo come assolutamente naturale ed intuitiva la scelta di indirizzare appropriatamente gli ordini, l'unica informazione di cui dispone l'AG è la pressione che un'assegnazione errata ha sull'aggregato dei costi totali. L'AG, mancando di una qualsivoglia comprensione semantica delle stringhe che evolve, si limita a penalizzare le soluzioni più costose,

e pur procedendo in maniera per certi versi miope riesce a fare emergere la soluzione migliore, che in quasi tutti i casi consiste nel fare corrispondere confezioni e filatoi.

9.1 Risultati

I risultati presentati nel seguito sono stati normalizzati considerando pari a 100 i costi generati in media da una programmazione casuale; tali costi, che in media sono sempre molto superiori ai costi ottenuti dalle programmazioni "umane" od ottimizzate dall'AG, presentano comunque una variabilità considerevole, con valori estremi anche molto distanti da tale media.

Sono stati svolti innumerevoli esperimenti, nel corso dei quali è stato possibile confrontare le programmazioni ottimizzate dall'AG e le programmazioni random; si è potuto concludere che l'AG in effetti ottiene dei risultati sempre migliori di quanto faccia una disposizione casuale degli ordini. Questo è un primo interessante risultato, che dimostra come l'evoluzione delle stringhe abbia effettivamente luogo grazie all'AG.

La ricerca delle soluzioni migliori non si limita alla selezione degli indivudui più soddisfacenti nell'insieme iniziale di stringhe generate casualmente, ma riesce anche, grazie alle operazioni di crossover e mutazione tipiche degli AG, ad evolvere individui con fitness decisamente migliore della media iniziale. In figura 7 è rappresentato il risultato di un'evoluzione protratta per circa 2.000 generazioni; nel grafico (che in ascissa rappresenta il succedersi delle generazioni ed in ordinata la fitness indicata come costo totale cambiato di segno) compaiono, dal basso verso l'alto, gli andamenti della fitness relativa all'individuo peggiore, la fitness media della generazione e la fitness del migliore individuo. Si nota come, dopo un periodo iniziale di miglioramenti sensibili, la popolazione di stringhe converga verso una buona soluzione, mentre il risultato va asintoticamente stabilizzandosi.

I risultati ottenuti da due esperimenti in particolare sono specialmente rilevanti, dal momento che in questi due casi c'è stata la possibilità di confrontare i risultati delle programmazioni arti-

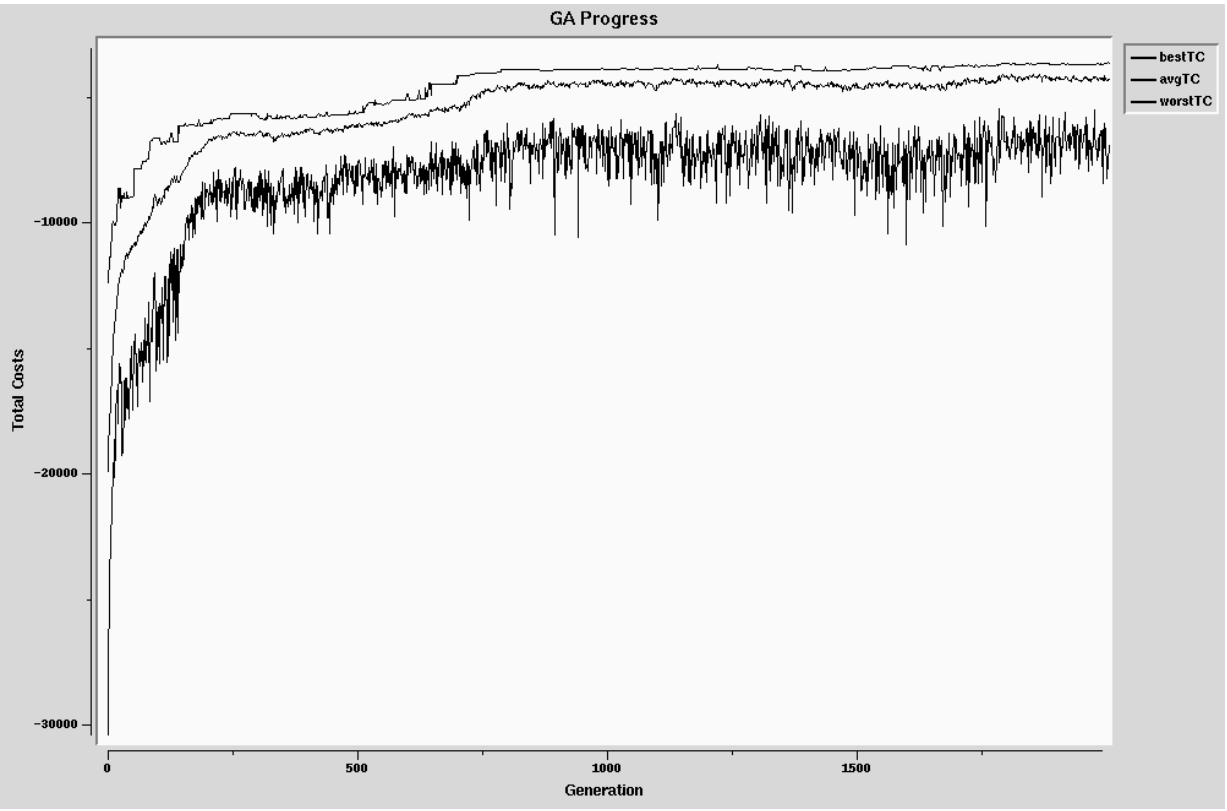

Figura 7: Ricerca di soluzioni tramite AG

ficiali (random e da AG) con le programmazioni che sono state decise e messe effettivamente in produzione dal dispositore aziendale.

Il primo di questi esperimenti ha riguardato la programmazione di 125 ordini su 15 filatoi, con ordini di entità variabile tra i 500 Kg. circa e le 15 Tonnellate. Si considerino pari a 100 i costi medi ottenuti da 100 programmazioni casuali[31]: nella tabella 5 si può notare come il risultato ottenuto dal dispositore umano, che costituisce un indubbio miglioramento, sia comunque peggiore rispetto alla programmazione ottimizzata dall'AG.

RND	**100.00**
Umano	24.75
GA	12.38

Tabella 5: Costi di produzione a confronto: RND, Umano, GA - **125 ordini**

Il dispositore ha ottenuto un buon risultato grazie alla sua capacità di azzerare i costi di confezione (cfr. il punto 6.3), all'aver disposto gli ordini secondo una sequenza ragionevole in vista della date di consegna e alla sua capacità di bilanciare il carico tra filati di titolo diverso in modo da ottenere una produzione media corretta. In particolare si è potuto notare come sulle singole macchine si susseguano per lo più filati simili dal punto di vista tecnico, rendendo necessarie poche operazioni di riattrezzaggio.

È opportuno presentare anche i risultati del secondo esperimento che ha coinvolto il dispositore aziendale. Si notano infatti valori simili per quanto riguarda la performance umana, che riduce ad un quarto i costi delle programmazioni casuali, così com'era avvenuto nell'esperimento su 125 ordini.

In questo secondo caso tuttavia il miglioramento ottenuto grazie all'AG è ancora più marcato:

[31]Il programma mette a disposizione un metodo particolarmente agevole per ottenere il risultato medio di *n* programmazioni casuali: è sufficiente infatti osservare nel grafico delle fitness (in modalità `1/GA`) la media della prima generazione di *n* individui, che sono generati casualmente una volta stabilito quanto numerosa debba essere la popolazione (cfr. 8.9).

mettendo a confronto diretto i risultati del dispositore e quelli ottimizzati la riduzione di costi supera addirittura il 60% (contro il 50% del primo esperimento).

Si tratta di risultati piuttosto netti, per certi versi eclatanti, e alcune riflessioni e considerazioni a questo punto sono necessarie. La prima considerazione riguarda la vera natura di quelli che si sono voluti chiamare "costi totali": l'uso del termine *costi* potrebbe essere per certi versi fuorviante. Sono valori eterogenei quelli che contribuiscono, una volta sommati, a formare il numero usato per misurare la bontà di una programmazione, e non tutti si possono considerare costi a tutti gli effetti.

La componente dovuta, ad esempio, ai costi dei riattrezzaggi (cfr. 6.2) ha un significato chiaro e ben definibile economicamente: corrisponde precisamente al valore delle ore di lavoro consumate in una specifica attività. In analogia a quest'ultima, la componente da "errata confezione" (cfr. 6.3) viene fatta corrispondere al costo reale che l'operazione di ri-rocchettaggio comporta per un lotto di filato che è stato prodotto su rocchette non appropriate.

Altri aspetti, d'altra parte, seppure tenuti sotto controllo, non sono direttamente traducibili in termini monetari: una consegna ritardata non genera alcun costo (a meno del caso limite in cui il cliente non decida di annullare il suo ordine), ma è comunque un'eventualità non desiderabile che va evitata. A tale scopo si fa corrispondere ai ritardi, in proporzione alla loro un gravità, un costo figurativo (cfr. il punto 6.1) che è una sorta di misura della gravità del disagio generato.

La gestione dei cosiddetti "costi da titolo medio" (punto 6.4), "costi da percentuale colore" (punto 6.5) e "costi da interferenza dei riattrezzaggi" (punto 6.2.2) ha luogo in maniera del tutto analoga: si tratta di penalizzare le programmazioni che non soddisfano i vincoli imposti. Si ottiene il risultato desiderato attribuendo a ciascuna delle suddette eventualità un "costo" figurativo,

RND	**100.00**
Umano	25.72
GA	9.81

Tabella 6: Costi di produzione a confronto: RND, Umano, GA - **32 ordini**

RND	**100.00**
Umano	82.27
GA (*run* di $5'$)	68.75
GA (*run* di $30'$)	62.25
GA (*run* di 6*h*)	60.62

Tabella 7: Costi di produzione a confronto: RND, Umano, GA - **120 ordini**

ponderato in base alla criticità di ciascun vincolo.

La scelta di penalizzare le scelte di programmazione che violano certi vincoli piuttosto che di impedire *tout-court* tali programmazioni può permettere all'AG di evolvere soluzioni che mai il dispositore umano prenderebbe in considerazione, peraltro sbagliando. La bontà di ogni programmazione viene considerata a livello "aggregato" di costi, ed è possibile che una programmazione viziata sotto un aspetto (e per questo penalizzata) sia particolarmente vantaggiosa sotto un altro aspetto che potrebbe più che compensare la penalità ricevuta.

9.2 Precisazioni su alcuni risultati preliminari

Il 20 Aprile 2001 presso l'Unione Industriale di Biella si è tenuto un incontro, organizzato dal *Gruppo Giovani Imprenditori*, nell'ambito del progetto `e-biella`. Nel corso del mio intervento ho presentato i risultati dell'applicazione del programma, non ancora nella sua versione definitiva, ad un primo insieme di 120 ordini.

Nella tabella 7 compaiono gli indici[32] relativi alle performance del programmatore umano e dell'AG, concedendo a quest'ultimo, per l'ottimizzazione, un tempo limitato in ciascuno dei tre casi proposti, a 5 minuti, 30 minuti e 6 ore. Una delle conclusioni che già allora si erano potute trarre riguardava la relativa rapidità con cui l'AG è in grado di ottenere una soluzione ragionevole, e come prolungare il tempo di *run* comporti vantaggi via via decrescenti.

Osservando i risultati preliminari presentati in occasione dell'incontro, le differenze di perfor-

[32] La cifra relativa alla media delle programmazioni random è stata normalizzata attribuendole l'usuale valore di 100 e le altre cifre sono state scalate proporzionalmente.

mance tra le tre modalità di programmazione risultano molto meno accentuate rispetto a quanto risulta dai dati definitivi presentati al punto 9.1.

È opportuno chiarire quali siano state le ragioni che hanno causato differenze tanto accentuate tra i dati presentati preliminarmente ed i dati definitivi riportati in questo lavoro.

Una prima spiegazione si trova in un'importante differenza tra il modello `RC-1`, utilizzato per i calcoli definitivi, e la versione anteriore del modello (`v. 0.10`) che ha dato i risultati della presentazione del 20 Aprile: nella versione provvisoria del modello il calcolo di due componenti di costo (costo da "titolo medio" e costo da "percentuale colore") era già presente ma disattivato, in quanto non sufficientemente collaudato. Test successivi hanno comunque rivelato come l'incidenza di tali costi sul risultato totale non sarebbe stata particolarmente importante.

Assolutamente rilevante, rispetto ai valori misurati, è stata invece un'anomalia in alcuni dei *file* utilizzati: `KMSETUP.txt` e `KMPIAN1.txt` (si veda, per dettagli sulla struttura e sulla funzione di questi, il punto 8.3.2). Il file utilizzato per determinare lo stato iniziale di tutti i filatoi era stato ricavato dalla base di dati aziendale a distanza di qualche tempo rispetto al file degli ordini da programmare. Il setup delle macchine risultante dal file `KMSETUP.txt` era relativamente recente rispetto al file degli ordini ancora da programmare: in pratica, molti degli ordini presentavano una data di consegna anteriore al momento in cui i filatoi sarebbero risultati liberi e pronti per la produzione.

Tale anomala situazione ha generato sensibili costi di ritardo, tra l'altro inevitabili, vista la impossibilità di produrre in tempo ordini "già scaduti" al momento della programmazione.

Una parte piuttosto cospicua dei costi totali risultanti era rappresentata proprio da tali ineliminabili costi da ritardo; tenendo conto dell'incidenza di questi la situazione si avvicina molto a quella degli ultimi esperimenti, in cui le performance del dispositore e dell'AG sono sensibilmente migliori della media delle programmazioni *random*.

9.3 Alcune considerazioni sul confronto tra dispositore umano ed AG

Dalle prove effettuate è emerso che le programmazioni ottimizzate tramite AG generano minori costi totali rispetto alle programmazioni effettuate dal dispositore umano che lavora presso la Filatura. I risultati presentati sono piuttosto significativi: l'AG arriva a dimezzare il valore del parametro che misura il costo di ogni programmazione.

Anche considerando che i parametri impiegati per la valutazione sono stati fissati secondo criteri talvolta soggettivi (ma più spesso hanno un preciso fondamento oggettivo), e le "regole del gioco" usate nella simulazione potrebbero non rispecchiare fedelmente la complessa realtà aziendale, la differenza pare abissale. Per questo motivo si è passato molto tempo a riflettere sul vero significato dei risultati ottenuti, sospettando un qualche vizio di fondo nel modo in cui il modello è stato concepito.

La ragione di tanta differenza nelle performance è invece apparsa evidente dopo avere compreso a fondo, tramite numerosi sopralluoghi e colloqui, le dinamiche che regolano la programmazione della produzione all'interno dell'azienda.

Il dispositore lavora a stretto contatto con gli operai del reparto filatura, i quali, organizzati in diverse squadre, si occupano tra le altre cose delle operazioni di riattrezzaggio. Al tempo stesso, il dispositore non ha tecnicamente modo di interagire con i clienti in attesa di consegna.

Esaminando le singole voci di costo relative alle programmazioni messe a confronto si nota una differenza sistematica tra il caso AG ed il caso del dispositore: le componenti che presentano le differenze più sensibili sono i ritardi ed i riattrezzaggi. Nelle programmazioni effettuate dal dispositore i costi da ritardo sono molto più elevati, mentre i costi da riattrezzaggio risultano moderatamente inferiori.

Se ne può dedurre immediatamente che il dispositore presta una particolare attenzione all'aspetto "riattrezzaggi", tanto che il desiderio di ridurli ad un minimo va a discapito dei tempi di consegna, i quali slittano sensibilmente rispetto a quanto avviene per le programmazioni ottimizzate da AG.

Le pressioni più o meno esplicite che il dispositore umano riceve da parte di coloro che devono materialmente effettuare il lavoro di riattrezzaggio (ritenuto particolarmente sgradevole perchè estremamente ripetitivo) hanno delle chiare ripercussioni sulle sue decisioni operative, tanto più che tali richieste non possono essere compensate da pressioni simili da parte dei clienti, con i quali il dispositore non entra mai in contatto e che finiscono per essere penalizzati.

10 L’interfaccia utente del programma

Nel corso dell’esposizione di questa tesi si trovano molti riferimenti all’interfaccia utente del software sviluppato: nel seguito sono rappresentati alcuni dei pannelli di controllo e di inserimento dati del programma.

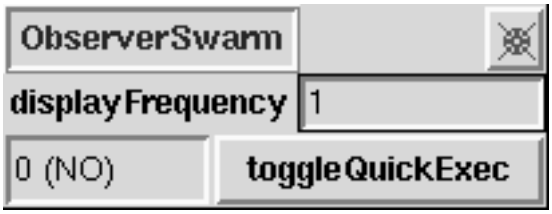

Figura 8: L'interfaccia utente di `ObserverSwarm`

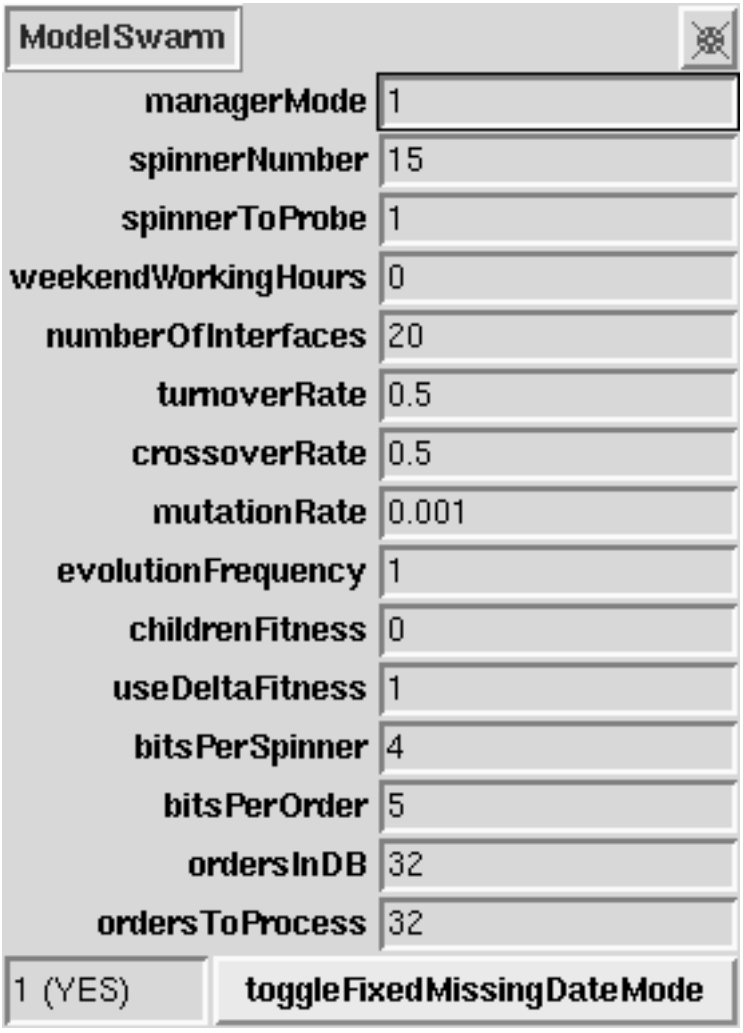

Figura 9: L'interfaccia utente di `ModelSwarm`, che comprende i parametri iniziali dell'AG

Figura 10: I pulsanti del pannello di controllo `probeGenerator`

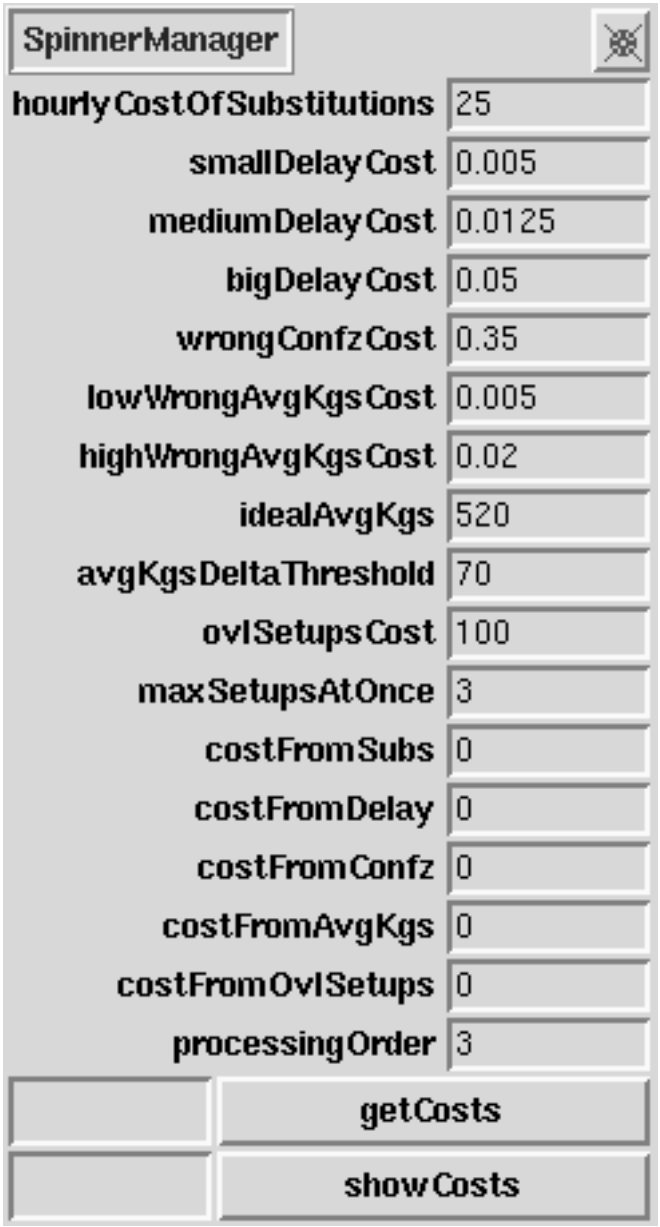

Figura 11: I parametri dell'agente `SpinnerManager`

Figura 12: Il pannello di controllo di `DBInterface`

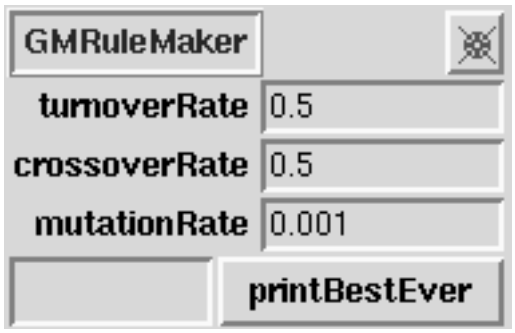

Figura 13: Il pannello per il controllo *real-time* di `GMRuleMaker`

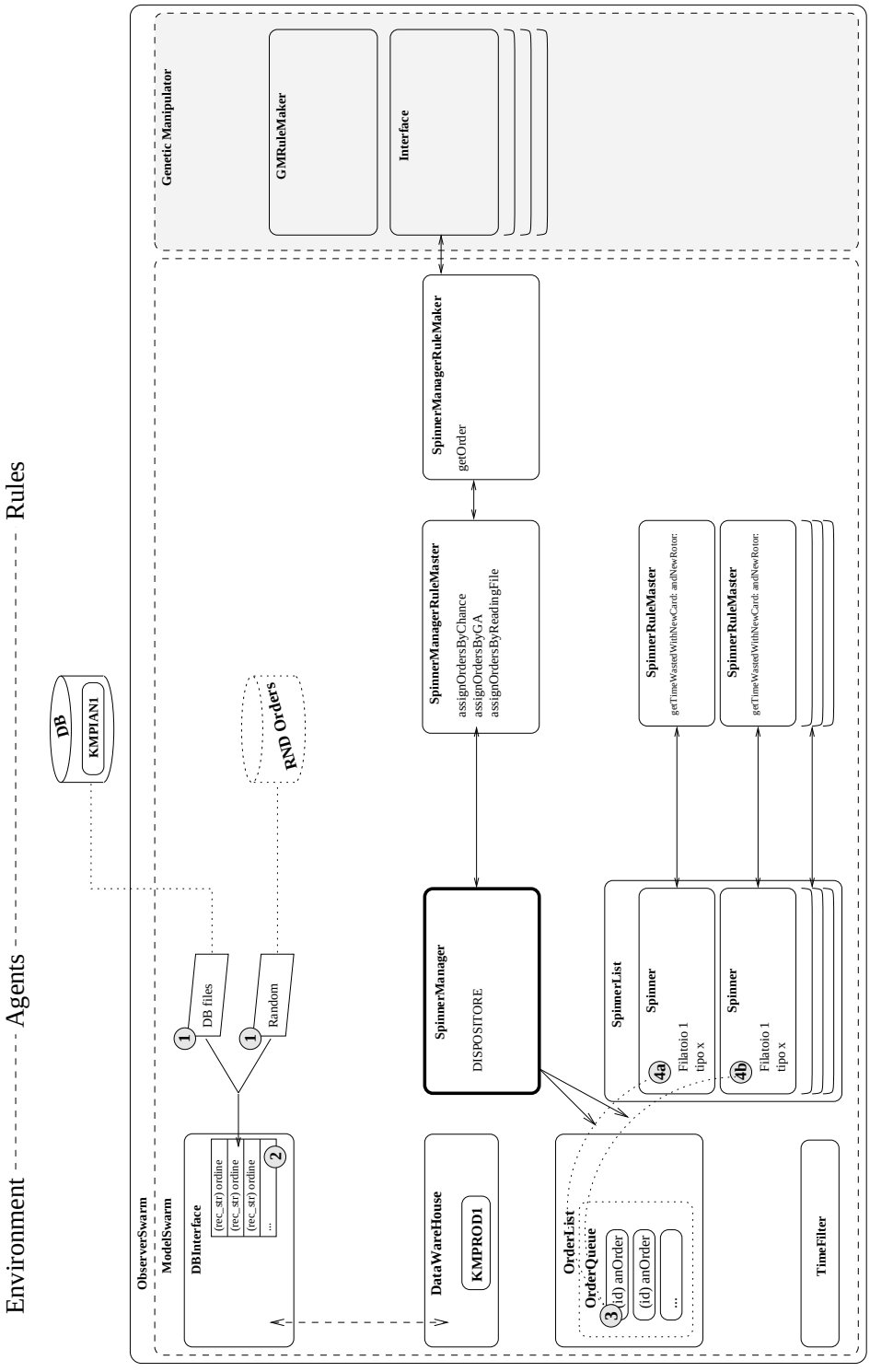

Figura 14: La gerarchia ERA degli oggetti ed il flusso degli ordini all'interno del modello della Filatura ($1 \rightarrow 2 \rightarrow 3 \rightarrow 4x$)

A debugged program is one for which

you have not yet found the conditions that make it fail.

– Jerry Ogdin

11 Estratti di codice Objective-C (Appendice)

11.1 `Macro.h`

```
// Macro.h

// _LOG1  Observer
//#define _LOG1

// _LOG2  Model
#define _LOG2

// _LOG3  DataWarehouse
//#define _LOG3

// _LOG4  DBInterface
//#define _LOG4

// _LOG5  OrderList
//#define _LOG5

// _LOG6  Order
#define _LOG6

// _LOG7  Spinner
//#define _LOG7

// _LOG8  SpinnerSetupper
#define _LOG8

// _LOG9  SpinnerRuleMaster
//#define _LOG9

// _LOG10 SpinnerManager
//#define _LOG10

// _LOG11 SpinnerManagerRuleMaster
//#define _LOG11

// _LOG12 SpinnerManagerRuleMaker
//#define _LOG12

// _LOG13 TimeFilter
//#define _LOG13

// _LOG14 GMRuleMaker
//#define _LOG14

// _LOG15 Interface
//#define _LOG15

// User-configurable defines end here ---

#ifdef _LOG1
#define L1(A) A
#else
#define L1(A) //
#endif

#ifdef _LOG2
#define L2(A) A
```

```
#else
#define L2(A) //
#endif

#ifdef _LOG3
#define L3(A) A
#else
#define L3(A) //
#endif

[...]
```

11.2 TimeFilter.h

```
// TimeFilter.h

#import <objectbase/SwarmObject.h>
#import <stdlib.h>
#import <time.h>

@interface TimeFilter: SwarmObject
{


}

- (time_t) getAbsoluteTime: (char *) str;

- createEnd;

@end
```

11.3 TimeFilter.m

```
// TimeFilter.m

#import "TimeFilter.h"
#import "Macro.h"

@implementation TimeFilter

- (time_t) getAbsoluteTime: (char *) str
{
  time_t t;
  struct tm brokenDownTime;
  //int  year,     mon,     mday,     hour,     min,     sec;
  char year_s[5],mon_s[3],mday_s[3],hour_s[3],min_s[3];

  sscanf(str,"%4s%2s%2s%2s%2s",
 year_s,mon_s,mday_s,hour_s,min_s);

  brokenDownTime.tm_sec = 0;
  brokenDownTime.tm_min = strtol(min_s,0,10);
  brokenDownTime.tm_hour = strtol(hour_s,0,10);
  brokenDownTime.tm_mday = strtol(mday_s,0,10);
  brokenDownTime.tm_mon = strtol(mon_s,0,10)-1;
  brokenDownTime.tm_year = strtol(year_s,0,10)-1900;
  brokenDownTime.tm_isdst = -1;

  t=mktime(&brokenDownTime); //Also sets DST info
  L13(printf("[TFL] Transforming %s[TFL] into UTC %i (DST: %i)\n",
     ctime(&t), (int) t, brokenDownTime.tm_isdst);)
  return (time_t) t; //WILL OVERFLOW CET 2038 Jan 19 around 4:14 am on 32-bit machines
}

- createEnd
{
   [super createEnd];
   return self;
}

@end
```

11.4 Order.h

```
// Order.h

#import <objectbase/SwarmObject.h>
#import <simtools.h>  // necessary to invoke OutFile protocol
#import <time.h>
#import "DBInterface.h"

@interface Order: SwarmObject
{
  int sernum; // each order has a unique serial number;
  struct rec_str data; //encapsulated data (= 1 order)
}

- setSerNum: (int) sn;
- (int) getSerNum;

- printData;
- setData: (struct rec_str) ord;
- (struct rec_str) getData;

- (time_t) getEndTime;
- (int) compare: other;
- createEnd;

@end
```

11.5 Order.m

```
// Order.m

#import "Order.h"
#import "Macro.h"

@implementation Order

- createEnd
{
  [super createEnd];
  return self;
}

- setSerNum: (int) sn
{
  sernum=sn;
  return self;
}

- (int) getSerNum
{
  return sernum;
}

- printData
{
  // char variables only defined if logging;
  // avoids "unused variables" compiler warnings

#ifdef _LOG6
const char  *cardCode[3] = {"AA","BB","CC"}; //brutto definirli qui,
const char *rotorCode[5] = {"T1","T2","S1","S2","S3"}; //ma serve solo al log
const char *confzCode[3] = {"T","M","S"};
#endif

  int i,j,k;

  i = data.card;
  j = data.rotor;
  k = data.confz;

  L6(printf("[ORD] /-------------\n");)
  L6(printf("[ORD] # %i CODE<%s> (Kgs %7.2f)\n",
      sernum, data.code, data.kgs);)
  L6(printf("[ORD] ghh %7.2f [%s|%s|%s]  Expected %i\n",
      data.ghh,cardCode[i], rotorCode[j], confzCode[k], (int) data.dlvry);)
  L6(printf("[ORD]                              -------------/\n");)

  return self;
}

- setData: (struct rec_str) ord
{
  data = ord;
  return self;
}

- (struct rec_str) getData
{
   return data;
}
```

```
- (time_t) getEndTime
{
  return data.dlvry;
}

- (int) compare: other
{
  if (self == other) {
    return 0;
  }
  if ([self getEndTime] > [other getEndTime]) {
    return 1;
  }

  if ([self getEndTime] == [other getEndTime]) {
    return 0;
  }

  return -1;
}

@end
```

11.6 DBInterface.h

```
// DBInterface.h

#import <objectbase/SwarmObject.h>
#import <simtools.h>  // necessary to invoke In/OutFile protocol
#import <time.h>
#import "DataWareHouse.h"
#import "TimeFilter.h"

@interface DBInterface: SwarmObject
{
  id theObserverSwarm; //impossible to typize, recursive import would occur
  DataWareHouse * myDataWareHouse;
  TimeFilter * myTimeFilter;

  id <InFile> ordersFile;

  char *lastOrdersFileName;      //displayed in my probe
  char *conventionallyGivenDate; //if an order is missing a delivery date use this
  int daysLater;

  struct rec_str {  // typical database entry made available for other agents
    char  code[16]; // product code "xxxx/yyyy/zzzzn"
    time_t dlvry;   // expected delivery date (UTC)
    float kgs;      // order entity, in kilograms
    float ghh;      // grams/head/hour;
    int card;       // type 1,2,3=(AA,BB,CC)
    int rotor;      // type 1,2,3,4,5=(S1,S2,S3,T1,T2)
    int confz;      // type 1,2,3=(T,M,S)
  } rec;

  struct cod_kgs { // data read from KMPIAN before matching with KMPROD occurred;
    char code[16]; // used only internally by -ReadOrder and passed to getOrder
    time_t dlvry;
    float kgs;
    int confz;
  } orderFromFile;

  BOOL returnFakeOrders;
  BOOL fixedMissingDateMode;
}

+ createBegin: aZone;
- createEnd;

- setObserverSwarm: os;
- setDataWareHouse: dwh;
- setTimeFilter: theTimeFilter;

- readOrdersFrom: (char *) of;
- (struct cod_kgs) readOrder; //data read from file
- (struct rec_str) getOrder;  //data completed by the matching done by DWH

- (BOOL) toggleFakeOrdersGenerator;
- setFixedMissingDateMode: (BOOL) mdm;
@end
```

Riferimenti bibliografici

[AE94] R. L. Axtell and J. M. Epstein. Agent-based modelling: Understanding our creations. *Bulletin of the Santa Fe Institute*, 9(2), 1994.

[App99] Apple Computer, Inc. *Objected-oriented Programming and the Objective-C Language*, developer's library edition, 1999.

[Axt99] R. Axtell. *The Emergence of Firms in a Population of Agents*. Brookings Institution, Washington, 1999.

[Axt00] R. Axtell. *Why Agents? On the Varied Motivations for Agent Computing in the Social Sciences*. Center on Social and Economic Dynamics, November 2000. Working Paper No. 17.

[Bil97] T. Bilgiç. *Research in Enterprise Modelling and Integration*. Bogaziçi University, Istanbul, 1997. `http://www.ie.doun.edu.tr/faculty/taner/taner.html`.

[BMT96] A. Beltratti, S. Margarita, and P. Terna. *Neural Networks for Economic and Financial Modelling*. Thomson Computer Press, 1996.

[Cho59] N. Chomsky. On certain formal properties of grammars. *Information and Control*, 2(2):137–167, June 1959.

[Dew84] A. K. Dewdney. (ri)creazioni al calcolatore. *Le Scienze*, 193:124–131, Settembre 1984.

[DG88] J.H. Holland D.E. Goldberg. Genetic algorithms and machine learning. *Machine Learning*, 3:95–104, 1988.

[Dol00] A. Dolan. Ga playground. `http://www.aridolan.com/ga/gaa/gaa.html` and `http://www.aridolan.com/ga/gaa/Gaplayground.zip`, November 2000.

[Eps96] J. M. Epstein. *Growing Artificial Societies*. Brookings Institution Press, Washington, D. C., 1996.

[Eps99a] J. M. Epstein. Agent-based computational models and generative social science. *Complexity*, 4(5):41–60, 1999.

[Eps99b] J. M. Epstein. *Learning To Be Thoughtless: Social Norms and Individual Computation*. Center on Social and Economic Dynamics, Settembre 1999. Working Paper No. 6.

[FD96] G. Fine and J. Deegan. Three principles of serendip: Insight, chance, and discovery in qualitative research. *Qualitative Studies in Education*, 9(4), 1996.

[Fel97] R. Feldmann. Computer chess: Algorithms and heuristics for a deep look into the future. In F. Plasil and K.G. Jeffrey, editors, *SOFSEM '97: Theory and Practice of Informatics*, pages 1–18. Springer, 1997. Lecture Notes in Computer Science.

[Fer00] G. Ferraris. Algoritmi genetici per l'economia. Tesi di Laurea, Facoltà di Economia, Università degli Studi di Torino, Giugno 2000.

[Fer01] G. Ferraris. *GAMES: Algoritmi Genetici per l'Economia*. Number 51 in Quaderni del Dipartimento di Scienze Economiche e Finanziarie "G. Prato". Università degli studi di Torino, Facoltà di Economia, March 2001.

[FM98] R. Feldmann and Burkhard Monien. Selective game tree search on a Cray T3E. Technical report, University of Paderborn, 1998.

[GC95] N. Gilbert and R. Conte, editors. *Artificial Societies: The Computer Simulation of Social Life*. UCL Press, London, 1995.

[Ghe01] Massimo Ghelfi. Analisi delle cause che influenzano la formazione del pilling nei filati open–end. Monografia, Corso di diploma in Ingegneria Chimica, 2001.

[GT99] N. Gilbert and K. G. Troitzsch. *Simulation for the Social Scientist.* Open University Press, Buckingham, May 1999.

[GT00] N. Gilbert and P. Terna. How to build and use agent-based models in social science. *Mind & society*, 1(1), 2000.

[Haj97] S. Hajek. Intelligenza e adattamento - intervista a J. Holland. *Forum Teorema*, 1997. `http://www.forumteorema.it/Interview.htm`.

[Hoe85] Ulrico Hoepli, editor. *Nuovo Colombo – Manuale dell'Ingegnere*, volume 2. Hoepli, Milano, 81st edition, 1985.

[Hol75] J. H. Holland. *Adaptation in Natural and Artificial Systems*. MIT Press, Cambridge, MA, 1975.

[Hol98] B. Holmstrom. The firm as a subeconomy. In *Bureaucracy: Issues and Apparatus*, Ottobre 1998.

[HR99] M. Harris and A. Raviv. *Organization Design*. University of Chicago, Luglio 1999.

[HS00] A. Heifetz and Y. Spiegel. *On the Evolutionary Emergence of Optimism*. Tel Aviv University, Ottobre 2000. `http://www.tau.ac.il/~spiegel/papers/optimism53.pdf`.

[JLS99] P. Johnson, A. Lancaster, and B. Stefansson. *Swarm User Guide*. Swarm Development Group, Novembre 1999.

[Kuh62] T. S. Kuhn. *The structure of scientific revolutions*. University of Chicago Press, Chicago, 1962.

[LeB00] B. LeBaron. Agent-based computational finance: Suggested readings and early research. *Journal of Economic Dynamics & Control*, 24:679–702, 2000.

[LS00a] Francesco Luna and Benedikt Stefansson, editors. *Economic Simulations in Swarm: Agent-Based Modelling and Object Oriented Programming*. Kluwer Academic Publishers, 2000.

[LS00b] Francesco Luna and Benedikt Stefansson, editors. *Economic Simulations in Swarm: Agent-Based Modelling and Object Oriented Programming*, chapter 9. Kluwer Academic Publishers, 2000. F.-R. Lin, T. J. Strader, M. J. Shaw, Using Swarm for Simulation the Order Fulfillment Pricess in Divergent Assembly Supply Chains.

[LS00c] Francesco Luna and Benedikt Stefansson, editors. *Economic Simulations in Swarm: Agent-Based Modelling and Object Oriented Programming*, chapter 10. Kluwer Academic Publishers, 2000. C. Schlueter-Langdon, P. Bruhn, M. J. Shaw, Online Supply Chain Modelling and Simulation.

[LS00d] Francesco Luna and Benedikt Stefansson, editors. *Economic Simulations in Swarm: Agent-Based Modelling and Object Oriented Programming*, chapter 3. Kluwer Academic Publishers, 2000. P. Terna, Economic Experiments with Swarm: a Neural Network Approach to the Self-Development of Consistency in Agents' Behavior.

[LTS96] F.-R. Lin, G. W. Tan, and M. J. Shaw. *Multi-Agent Enterprise Modelling*. University of Illinois at Urbana-Champaign, Ottobre 1996. Office of Research Working Paper 96-0134.

[Mar92] S. Margarita. Verso un "robot oeconomicus" algoritmi genetici ed economia. *Sistemi Intelligenti*, 4(3):421–459, Dicembre 1992.

[MBLA96] N. Minar, R. Burkhart, C. Langton, and M. Askenazi. *The Swarm Simulation System: A Toolkit for Building Multi-agent Simulations*. Santa Fe Institute, Giugno 1996. `http://www.swarm.org/`.

[McFng] D. McFadden. Rationality for economists? *Journal of Risk and Uncertainty*, Special Issue on Preference Elicitation, Forthcoming.

[MT00a] J. P. Marney and H. F. E. Tarbert. Why do simulation? toward a working epistemology for practitioners of the dark arts. *Journal of Artificial Societies and Social Simulation*, 3(4), Ottobre 2000.

[MT00b] S. Mullainathan and R. Thaler. *Behavioral Economics*. Massachusetts Institute of Technology, Settembre 2000. Working Paper 00-27.

[Par99] S. Parinov. *Toward a Theory and Agent-Based Model of the Networked Economy*. Institute of Economics & IE SB RAS, Luglio 1999. `http://www.ieie.nsc.ru/~parinov`.

[PCG99] M. J. Prietula, K. M. Carley, and L. Gasser, editors. *Simulating Organizations, Computational Models of Institutions and Groups*. AAAI Press – The MIT Press, 1999.

[Pro00] Linux Documentation Project. *Linux Programmer's Manual*. Red Hat, Inc. `http://bugzilla.redhat.com/bugzilla`, man-pages 1.28-6 edition, Marzo 2000.

[RN98] S. J. Russell and P. Norvig. *Intelligenza artificiale*. Prentice Hall International – Utet Libreria, 1998.

[SLS96] T. J. Strader, F.-R. Lin, and M. J. Shaw. *Information infrastructure for electronic virtual organization management*. University of Illinois at Urbana-Champaign, Ottobre 1996. Office of Research Working Paper 96-0135.

[SLS98] T. J. Strader, F.-R. Lin, and M. J. Shaw. Simulation of order fulfillment in divergent assembly supply chains. *Journal of Artificial Societies and Social Simulation*, 1(2), Marzo 1998.

[Ter94] P. Terna. XXXV riunione scientifica annuale della società italiana degli economisti, Milano. In *Reti neurali artificiali e modelli con agenti adattivi*. Dipartimento di Scienze economiche e finanziarie G. Prato, Facoltà di Economia, Università degli studi di Torino, Ottobre 1994.

[Ter98] P. Terna. Simulation tools for social scientists: Building agent based models with swarm. *Journal of Artificial Societies and Social Simulation*, 1(2), 1998. `http://www.soc.surrey.ac.uk/JASSS/1/2/4.html`.

[Ter00] P. Terna. Ct scheme and era scheme. Unpublished, August 2000.

[Ter01] P. Terna. Creating artificial worlds: A note on *Sugarscape* and two comments. *Journal of Artificial Societies and Social Simulation*, 4(2), 2001. `http://www.soc.surrey.ac.uk/JASSS/4/2/9.html`.

[Vri95] N. J. Vriend. Self organization of markets: An example of a computational approach. *Journal of Computational Economics*, 8(3):205–231, 1995.

[Vri98] N. J. Vriend. *An Illustration of the Essential Difference between Individual and Social Learning, and its Consequences for Computational Analyses*. Queen Mary and Westfield College, University of London, Dept. of Economics, Luglio 1998.