UNIVERSITÀ DEGLI STUDI DI PADOVA

Dipartimento di Tecnica e Gestione dei Sistemi Industriali

GESTIONE INTELLIGENTE DI SCORTE CON LOGICA FUZZY

Laureando: Eris Chinellato
Relatore: Ch.mo Prof. Silverio Bolognani

Anno Accademico 1997-1998

PRESENTAZIONE

Il presente lavoro è stato concepito come un nuovo approccio al problema della gestione di scorte. Impiegando la logica fuzzy, si è progettato e realizzato un sistema software di gestione che funziona in modo sostanzialmente diverso da altri metodi automatici impiegati dalle aziende.

Il sistema creato riesce ad emulare la perizia decisionale di un operatore umano esperto, e per questa ragione è stato definito intelligente. Esso infatti sa prendere in considerazione quei fattori empirici e poco analitici che gli usuali sistemi automatici trascurano, e che rendono invece così efficiente il ragionamento umano.

L'attitudine ad applicare conoscenze di questo genere, unita alla capacità di elaborare grosse quantità di dati tipica di un sistema informatico, costituisce la forza innovativa dello strumento di gestione realizzato.

Le prove finali di funzionamento sono state eseguite con dati reali. Tali prove, oltre che a valutare le prestazioni del sistema, aiutano a comprendere gli eventuali sviluppi cui potrebbe condurre l'approfondimento del filone di applicazioni gestionali indicato da questo progetto.

INDICE

INTRODUZIONE

Le aziende moderne tendono sempre più a perseguire livelli di massima qualità sotto ogni aspetto di gestione, al fine di ottenere i risultati migliori con il minimo impiego di risorse. Nonostante questa crescente attenzione ad ogni possibile causa d'inefficienza, si constata che alcune funzioni aziendali offrono ancora ampi margini di miglioramento. Ad esempio, le prestazioni concernenti la gestione dei materiali, ed in particolare delle scorte di magazzino, sono spesso lontane dall'essere ottimizzate.

Tra le aziende dotate di sistemi per la gestione di scorte automatizzati, alcune utilizzano procedure molto complesse, basate su strumenti statistici sofisticati. Tali procedure offrono in genere risultati soddisfacenti, ma difficilmente sono esportabili ad altre realtà aziendali, essendo sviluppate in base alle esigenze specifiche della realtà d'interesse. Più spesso la gestione è affidata a procedure basate su modelli classici, fondati su presupposti raramente riscontrabili nella realtà. In questi casi può nascere l'esigenza di correggere manualmente i risultati poco soddisfacenti offerti dalle procedure automatiche di calcolo dei fabbisogni.

Affidandosi alle capacità decisionali di personale specializzato, ci si accorge di quanto possano rivelarsi importanti per la gestione una serie di fattori difficili da quantificare e formalizzare. Mentre un operatore esperto sa tenere conto di tali fattori, confidando nella propria competenza e nella profonda conoscenza che ha del problema, non è per niente agevole inserirli in un modello matematico.

Ciò che manca agli attuali sistemi automatici di gestione scorte è proprio la capacità umana di considerare gli aspetti "sfumati" della realtà. D'altro canto, sistemi che utilizzano un elaboratore elettronico possono prendere decisioni relative a migliaia di articoli in pochi secondi.

L'obiettivo di questo lavoro è stato quello di progettare e sviluppare un sistema di gestione scorte automatico ma intelligente, capace di riprodurre su larga scala l'abilità decisionale di un operatore umano.

Per ottenere tale obiettivo, serviva uno strumento di progettazione che potesse andare oltre le capacità degli usuali metodi matematici e statistici, tenendo conto di tutti quei fattori poco analitici che possono avere peso notevole in problemi di questo tipo. Si è ritenuto che la logica fuzzy

presentasse le potenzialità adatte a supportare lo sviluppo di un sistema con le caratteristiche desiderate.

La *fuzzy logic*, o logica fuzzy, è uno strumento teorico avente sviluppi applicativi notevoli. Molti sistemi di controllo funzionano secondo i principi della logica fuzzy, ed è in forte aumento il numero di soluzioni fuzzy nel campo dei sistemi decisionali. L'approccio non dicotomico, ma *sfumato*, approssimato, e l'utilizzo di variabili e regole linguistiche al posto di costrutti matematici sono le caratteristiche che avvicinano i sistemi fuzzy al modo di pensare umano. Le stesse peculiarità fanno sì che i sistemi fuzzy riescano ad affrontare con successo proprio quei problemi che risultano più complessi da risolvere analiticamente.

La logica fuzzy non ha la precisione caratteristica degli strumenti analitici, e non risulta perciò conveniente se il modello matematico di un processo è noto con precisione e sicurezza. Quando i fattori che generano incertezza aumentano, amplificando la complessità del problema, i modelli analitici iniziano a mostrare i loro limiti, e i sistemi fuzzy si rivelano molto spesso più adatti. In questi casi, le capacità di adattamento e di semplificazione dei problemi, tipiche della logica fuzzy, divengono predominanti. La gestione di scorte rientra senza dubbio in questa categoria di problemi, avendo carattere fortemente aleatorio e poco analitico, ed essendo difficile da formalizzare e rappresentare tramite modelli matematici di validità generale.

Per quanto riguarda il progetto da sviluppare, una volta stabilito che la gestione "intelligente" si sarebbe avvalsa della logica fuzzy, si sono definiti gli obiettivi più specifici da perseguire.

Si è scelto di creare un sistema software di gestione che fosse utilizzabile da un utente finale, ossia da qualcuno che si occupi della gestione di un magazzino. Le caratteristiche della *fuzzy logic* hanno facilitato il compito di produrre un'interfaccia utente che non richieda particolari abilità matematiche o informatiche, ma piuttosto conoscenze relative al problema specifico da affrontare.

L'analisi di funzionamento del sistema è stata fatta su dati di gestione reali, relativi a prodotti finiti. Nei test, non si è cercato di ottenere la soluzione ottimale allo specifico caso di gestione a disposizione. Lo scopo primario della verifica era, invece, valutare l'effettiva applicabilità del nuovo approccio fuzzy al problema della gestione delle scorte di magazzino, e comprenderne le potenzialità in vista di sviluppi ulteriori.

Il sistema di gestione è stato sviluppato con l'appoggio della ditta **SEAP** di Milano, interessata a nuovi sviluppi nel campo dei sistemi di gestione, e quindi propensa a verificare la validità di questo tipo di approccio.
I dati utilizzati nella fase di test sono stati gentilmente forniti da **Porsche Italia S.p.A.**

Contenuto dei capitoli

Nel **primo capitolo** sono presentate le basi teoriche della logica fuzzy, con particolare attenzione al processo di progettazione di un sistema fuzzy.

Le caratteristiche della gestione di scorte in un'azienda manifatturiera sono l'argomento del **secondo capitolo**, che si conclude con un'analisi più approfondita delle ragioni che hanno spinto ad applicare la logica fuzzy alla gestione di scorte.

L'analisi del problema è svolta nel **terzo capitolo**, nel quale si considerano in modo dettagliato tutte le variabili che rivestono interesse per la gestione. Alla fine del capitolo è presentata una descrizione del funzionamento generale del sistema.

Il **quarto capitolo** contiene la descrizione dettagliata delle caratteristiche progettuali del sistema fuzzy e dei singoli blocchi di calcolo, con la motivazione delle scelte che hanno portato alla configurazione finale.

Nel **quinto capitolo** è approfondita la spiegazione del funzionamento del sistema. Ci si sofferma in particolare sulla struttura del software progettato (l'intero sistema è suddiviso in due sezioni, una di calcolo e una d'interfaccia), e sul modo in cui sono impiegate le informazioni di gestione.

La fase di messa a punto e i risultati conseguiti nelle prove sui dati reali sono gli argomenti trattati nel **sesto capitolo**, nel corso del quale si espongono e commentano le diverse modalità d'impiego del sistema.

Infine, nel **settimo capitolo** sono presentate dapprima alcune applicazioni software finalizzate alla progettazione di sistemi fuzzy, e quindi, in modo più dettagliato, l'applicazione scelta per realizzare il sistema.

In appendice sono riportati i listati dei due programmi che compongono il sistema di gestione. L'**appendice A** contiene il codice sorgente relativo al programma di calcolo, scritto in *C++*. L'**appendice B** riguarda invece il programma d'interfaccia, sviluppato in *Visual Basic*.

1

LA LOGICA FUZZY

1.1 INTRODUZIONE

La maggior parte dei concetti con cui le persone hanno a che fare ogni giorno sono soggettivi, difficili da quantificare e da classificare con sicurezza. Ad esempio, è possibile stabilire con certezza se una persona è alta? Chiunque affermerebbe che una persona della statura di due metri appartiene alla categoria degli *alti*, ma una persona di 178 cm è alta? E una di 175? Secondo la logica matematica tradizionale, si dovrebbe definire un limite preciso al di sopra del quale le persone si possono considerare alte: le persone che misurano almeno 178 cm sono alte, le altre non lo sono. Si comprende come una definizione di questo tipo sia poco rappresentativa del modo di pensare umano. È molto più naturale pensare all'insieme delle persone alte come ad un insieme che degrada in modo più o meno regolare, a partire dalle persone che sono inequivocabilmente alte per arrivare a quelle che certamente non lo sono. In questo caso chiunque stia tra i due estremi è alto, ma solo parzialmente: qualcuno lo è di più, qualcun altro di meno.

Risulta perciò evidente che l'appartenenza di una persona all'insieme degli *alti* non segue i canoni della logica tradizionale, non è esprimibile facilmente con un *sì* o un *no*. Tale appartenenza è invece descritta molto meglio definendo per ogni persona un certo *grado di appartenenza*, che esprime "quanto" la persona appartiene all'insieme degli *alti*. Lo stesso ragionamento si potrebbe ripetere per concetti come velocità *elevata*, prezzo *economico*, clima *freddo* e così via. La **teoria della logica fuzzy** si basa sulla definizione di insiemi "sfumati" di questo tipo, al fine di ottenere una rappresentazione più realistica di grandezze e concetti che sono per loro natura graduali, non dicotomici. Le variabili fuzzy non sono numeriche, ma linguistiche, ed assumono proprio valori come *alto*, *basso*, *freddo*, *caldo*.

La seconda caratteristica che avvicina la logica fuzzy al modo di pensare umano è il suo modo di rappresentare i ragionamenti. Di solito i controlli impiegano formule matematiche e metodi numerici per stabilire le corrispon-

La **teoria della logica fuzzy** è stata introdotta negli anni sessanta dall'ingegnere statunitense di origine iraniana Lotfi Zadeh. Prima di lui, diversi matematici si erano dedicati allo sviluppo di logiche multivalore, non dicotomiche. È però nella sua pubblicazione del 1965 "*Information and control n.8*" che prende forma per la prima volta una teoria unitaria capace di formalizzare alcuni aspetti *sfumati* del ragionamento umano.

denze tra le variabili d'ingresso e quelle d'uscita. Il ragionamento umano è invece caratterizzato dall'utilizzo di regole empiriche, a volte approssimative, dovute al buon senso o all'esperienza, ma difficilmente traducibili in termini analitici. Anche in questo caso la teoria fuzzy si rifà ai criteri decisionali umani, utilizzando regole linguistiche e non matematiche per definire il modo in cui le variabili si influenzano tra loro.

Nel guidare un'automobile eseguiamo continuamente azioni basate su ragionamenti del tipo: se la velocità è *elevata* e l'ostacolo *vicino*, premi *forte* sul pedale del freno; se la velocità è *moderata* e l'ostacolo si trova *a media distanza*, premi *leggermente* sul pedale del freno. Qualunque guidatore esegue spontaneamente e istantaneamente ragionamenti del genere, mentre risulta molto più difficile quantificare in modo preciso la forza in newton da applicare al pedale del freno in corrispondenza di una certa velocità in chilometri orari e di una certa distanza dall'ostacolo in metri. Le regole linguistiche fuzzy sono analoghe alle regole descrittive empiriche qui espresse, e non richiedono l'utilizzo di formule o di complessi modelli analitici.

Grazie a questo modo di "ragionare" i sistemi fuzzy si comportano in modo soddisfacente proprio in quelle situazioni che una persona saprebbe gestire con facilità, ma che risultano le più difficili da affrontare con metodi analitici, come l'esempio della frenata appena descritto.

I sistemi basati sulla logica fuzzy sono particolarmente adatti a lavorare in condizioni di incertezza e di disturbi nell'acquisizione dei dati. Si adattano bene a processi variabili nel tempo o fortemente non lineari, e quindi difficili da rappresentare con modelli matematici. Caratteristica della fuzzy logic è la notevole facilità di utilizzo e di comprensione, dovuta alla sua affinità con il ragionamento umano.

1.2 IMPIEGO DELLA LOGICA FUZZY

I sistemi gestiti con logica fuzzy sono in rapida espansione in molti campi. Le grandi aree di utilizzo sono prevalentemente due, i sistemi di controllo e i sistemi esperti o di supporto decisionale. Esempi di applicazioni del primo tipo sono la regolazione di umidificatori e condizionatori, l'eliminazione delle vibrazioni e la messa a fuoco per macchine fotografiche e telecamere, la gestione di sistemi di sicurezza nei trasporti (come ABS, sospensioni intelligenti, mantenimento automatico della distanza di sicurezza), la definizione delle strategie di lavaggio per

lavabiancheria in funzione delle caratteristiche del carico. Tra le applicazioni decisionali si possono citare sistemi di compravendita di azioni e di valutazione del rischio, sistemi per le previsioni meteorologiche e geofisiche, riconoscitori di caratteri e di immagini. L'applicazione presentata in questa tesi appartiene alla seconda categoria.

La **lavabiancheria fuzzy** richiede all'utente solo di caricare i panni e di premere il pulsante d'avvio. È il suo microprocessore fuzzy che decide, in ogni momento del processo di lavaggio, quanto detersivo immettere, quanta acqua, a che velocità deve girare l'agitatore e quante volte ripetere il risciacquo. Tali decisioni vengono prese in base alle informazioni comunicate da appositi sensori che valutano le dimensioni del carico, la sporcizia in base all'opacità dell'acqua e l'assorbimento d'acqua da parte del tessuto.

In molti sistemi fuzzy, le variabili d'ingresso sono espresse con valori numerici (p. es. la temperatura letta da un sensore, o il costo di una particolare decisione), ed è richiesto un valore numerico anche per le risposte che il sistema deve fornire (la potenza da erogare a un condizionatore, l'entità di un investimento). In tali situazioni, si presenta la necessità di creare un'interfaccia tra il ragionamento fuzzy e il mondo dei numeri. A questo scopo si utilizzano le operazioni di *fuzzificazione* e *defuzzificazione*, che trasformano un valore numerico in uno fuzzy e viceversa. Tra queste due fasi si inserisce il processo d'inferenza fuzzy, che fa corrispondere agli ingressi le uscite appropriate.

Tornando ai sistemi fuzzy di supporto decisionale, essi offrono un vantaggio concettuale rispetto ai sistemi decisionali basati sulla ricerca operativa o su altri metodi analitici. Quando si deve operare una scelta basandosi sull'utilizzo di metodi analitici, ci si trova di fronte ad uno spazio decisionale, finito o infinito, contenente le alternative possibili. Si cerca allora di trovare l'alternativa che massimizza una certa funzione obiettivo, rispettando nel contempo una serie di vincoli. La funzione obiettivo permette di ordinare le alternative secondo un grado di preferibilità, mentre i vincoli limitano lo spazio delle alternative. La scelta della funzione obiettivo, che deve essere formulata analiticamente, e la definizione dei vincoli risultano perciò determinanti sull'esito del processo. Nei casi in cui si vogliano conseguire più obiettivi, specie se contrastanti, ci si trova vincolati dai limiti di questa impostazione.

Per maggiori informazioni sull'utilizzo della logica fuzzy come strumento decisionale, vedere Zimmermann [12].

Al contrario, nella filosofia decisionale fuzzy, obiettivi e vincoli sono gestiti allo stesso modo. Entrambi sono espressi tramite funzioni particolari dette *di appartenenza*, mentre l'importanza e il ruolo che assumono nel sistema vengono stabiliti da regole linguistiche. In questo modo è molto più agevole far convivere obiettivi concorrenti, e fornire delle indicazioni al sistema senza dover necessariamente decidere se vanno usate come vincoli o come obiettivi. Vedremo in seguito come si possono sfruttare queste opportunità.

1.3 DEFINIZIONI

Nella teoria classica degli insiemi, fissato l'universo del discorso *X*, un elemento *x* di *X* può appartenere o no ad un certo sottoinsieme *A* di *X*. Si può definire una funzione di appartenenza $\mu_A(x)$ che stabilisce il legame tra gli elementi *x* e l'insieme *A*, e che può assumere due soli valori, zero o uno:

$$m_A(x) = \begin{cases} 1 & se\ x \in A \\ 0 & se\ x \notin A \end{cases} \tag{1.1}$$

La teoria degli insiemi fuzzy estende la teoria classica, introducendo il concetto di **grado di appartenenza** all'insieme (*membership*). La teoria prevede che un elemento possa appartenere parzialmente ad un insieme, secondo una **funzione di appartenenza** a valori reali nell'intervallo [0,1]:

$$m_A: X \rightarrow [0,1] \tag{1.2}$$

Un **fuzzy set** (insieme fuzzy) *A* può quindi essere definito come l'insieme di coppie ordinate costituite dagli elementi di *X* e dal corrispondente valore della funzione di appartenenza:

$$A = \{(x, m_A(x)) \mid x \in X\} \tag{1.3}$$

Se l'insieme universo *X* è continuo si può rappresentare il fuzzy set A con la notazione:

$$A = \int_X m_A(x)/x \tag{1.4}$$

Viceversa, se *X* è discreto, si può usare la notazione:

$$A = \sum_i m_A(x_i)/x_i \tag{1.5}$$

In queste scritture i simboli di integrale e sommatoria indicano un'unione, mentre il simbolo "/" non rappresenta una frazione, ma il legame tra un valore di appartenenza e l'elemento cui si riferisce.

Nella terminologia fuzzy, un insieme di tipo classico con funzione di appartenenza booleana viene anche detto *crisp set*. Esistono delle operazioni che permettono di convertire insiemi fuzzy in corrispondenti insiemi crisp.

Si definisce **supporto** del fuzzy set *A* l'insieme crisp

$S(A)$ costituito da tutti gli elementi di X aventi grado di appartenenza in A non nullo:

$$S(A) = \{x \in X \mid m_A(x) > 0\} \tag{1.6}$$

Analogamente, viene detto **supporto-**a (α-cut) di A l'insieme crisp $S(A)_a$ (o A_a) costituito dagli elementi di X aventi grado di appartenenza in A maggiore di α:

$$A_a = \{x \in X \mid m_A(x) > a\} \tag{1.7}$$

Un fuzzy set viene detto **singleton** se il suo supporto è costituito da un solo elemento di X.

Si definisce **nucleo** di un fuzzy set A l'insieme crisp $K(A)$ costituito da tutti e soli gli elementi di X aventi grado di appartenenza 1 in A:

$$K(A) = \{x \in X \mid m_A(x) = 1\} \tag{1.8}$$

Riunendo le definizioni di **nucleo** e **normalità** in un unico enunciato, si può dire che un fuzzy set A è **normale** se almeno uno dei suoi elementi ha grado di appartenenza 1 in A.

Un fuzzy set si dice **normale** se il suo nucleo contiene almeno un elemento di X.

Viene detto **convesso** un fuzzy set A che soddisfi la seguente condizione:

$$\begin{aligned} &\forall x, y \in X, \forall l \in [0,1] \Rightarrow \\ &m_A(lx + (1-l)y) \geq \min(m_A(x), m_A(y)) \end{aligned} \tag{1.9}$$

Un fuzzy set A in X normale e convesso viene denominato **numero fuzzy**.

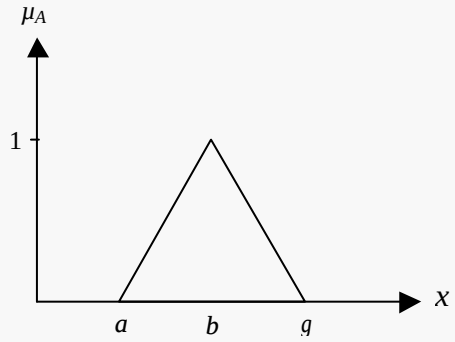
1.4 LE FUNZIONI DI APPARTENENZA

Nel caso di insiemi discreti e limitati la funzione di appartenenza può essere espressa numericamente da coppie di valori. Altrimenti, si deve definire una funzione che permetta di calcolare la membership di un elemento tramite un'espressione analitica.

A seconda del tipo di applicazione si possono definire funzioni di appartenenza anche molto diverse. Le seguenti sono quelle più frequentemente usate.

Funzione di appartenenza triangolare (fig.1.1)

È definita con tre parametri: gli estremi a e g e il punto di massimo b.

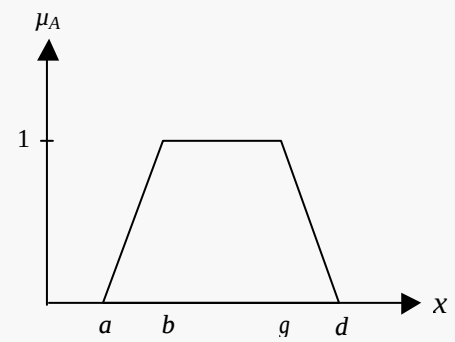


$$m_A(x,a,b,g)=\begin{cases}0 & \text{per } x<a\\ \dfrac{x-a}{b-a} & \text{per } a\le x\le b\\ \dfrac{g-x}{g-b} & \text{per } b<x\le g\\ 0 & \text{per } x>g\end{cases}$$

Fig. 1.1 Funzione di appartenenza triangolare con parametri di definizione.

Funzione di appartenenza trapezoidale (fig.1.2)

Presenta quattro parametri: gli estremi a e d e i valori inferiore e superiore dell'intervallo di massimo b e g.



$$m_A(x,a,b,g,d)=\begin{cases}0 & \text{per } x<a\\ \dfrac{x-a}{b-a} & \text{per } a\le x\le b\\ 1 & \text{per } b<x\le g\\ \dfrac{d-x}{d-g} & \text{per } g<x\le d\\ 0 & \text{per } x>d\end{cases}$$

Fig. 1.2 Funzione di appartenenza trapezoidale con parametri di definizione.

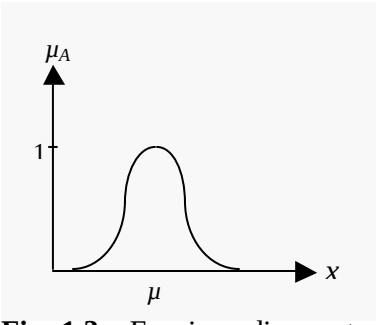


Fig. 1.3 Funzione di appartenenza a campana.

Funzione di appartenenza a campana (fig.1.3)

Si può ottenere con i parametri della triangolare usando archi di parabola al posto di segmenti retti, oppure con una gaussiana fissando i parametri m e s della distribuzione.

Per una variabile fuzzy si devono definire diverse funzioni di appartenenza, corrispondenti ai diversi valori linguistici che la variabile può assumere (ad esempio velocità *moderata, media, elevata*).

La scelta delle funzioni di appartenenza è un passo fondamentale nella messa a punto di un sistema fuzzy, visto che determina le caratteristiche dei processi di fuzzificazione degli ingressi e defuzzificazione delle uscite. La fuzzificazione permette di calcolare il grado di appartenenza di ogni valore numerico assunto da una variabile d'ingresso ad ogni fuzzy set definito per essa. Viceversa, la defuzzificazione calcola, a partire dal risultato

fuzzy ottenuto nel processo di inferenza, un valore reale per la variabile in uscita.

Per le funzioni di appartenenza degli insiemi fuzzy di una stessa variabile linguistica, un criterio di progettazione di validità generale è fare in modo che non ci siano parti dell'universo del discorso della variabile che rimangano scoperte. Questo si può evitare sovrapponendo parzialmente le funzioni di appartenenza della variabile.

I due fuzzy set situati agli estremi dell'insieme di definizione della variabile sono spesso descritti da normali funzioni di appartenenza, limitate però alla loro parte destra (limite inferiore) o sinistra (limite superiore), in modo che i valori estremi presentino su di esse grado di appartenenza unitario.

1.5 OPERAZIONI SUGLI INSIEMI FUZZY

Molte sono le operazioni definibili sugli insiemi fuzzy, alcune derivate dalle corrispondenti della teoria classica, altre peculiari della fuzzy logic.

Siano A e B due fuzzy set di uno stesso universo X, aventi funzioni di appartenenza μ_A e μ_B rispettivamente. Su di essi sono definibili le seguenti operazioni, descritte per mezzo delle loro funzioni di appartenenza.

Uguaglianza. Due fuzzy set A e B sono uguali se e solo se le loro funzioni di appartenenza sono uguali in tutto X:

$$A = B \Leftrightarrow m_A(x) = m_B(x) \quad \forall x \in X \tag{1.10}$$

Inclusione. Il fuzzy set A è contenuto nel fuzzy set B se e solo se la sua funzione di appartenenza è minore rispetto a quella di B in tutto X:

$$A \subseteq B \Leftrightarrow m_A(x) \leq m_B(x) \quad \forall x \in X \tag{1.11}$$

Unione (OR). L'insieme unione di A e B definiti in X è ancora un insieme di X, avente funzione di appartenenza (fig.1.4):

$$m_{A \cup B}(x) = \max\left(m_A(x), m_B(x)\right) \quad \forall x \in X \tag{1.12}$$

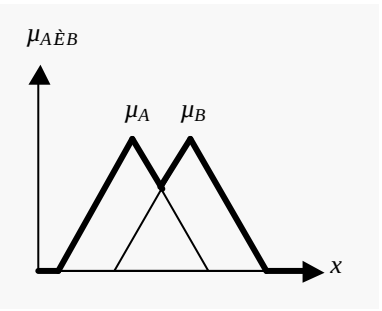


Fig. 1.4 Operazione di OR (unione) tra insiemi fuzzy.

Intersezione (AND). L'insieme intersezione di A e B definiti in X è ancora un insieme di X, avente funzione di appartenenza (fig. 1.5):

$$m_{A \cap B}(x) = \min\left(m_A(x), m_B(x)\right) \quad \forall x \in X \tag{1.13}$$

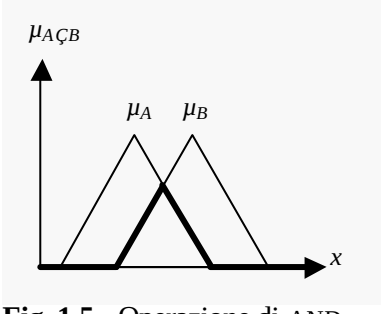


Fig. 1.5 Operazione di AND (intersezione) tra insiemi fuzzy.

Complemento (NOT). L'insieme complemento dell'insieme fuzzy A definito in X è ancora un insieme di X, avente funzione di appartenenza:

$$m_{\bar{A}}(x) = 1 - m_A(x) \quad \forall x \in X \tag{1.14}$$

È interessante notare che, a differenza di quanto accade per gli insiemi crisp, l'intersezione di A e $\bar{A}$ non dà necessariamente l'insieme vuoto, così come la loro unione non dà necessariamente l'insieme universo. Si può verificare che ci si avvicina alle situazioni tipiche della teoria classica quanto più il fuzzy set A si "avvicina" ad un insieme classico (vedi a lato il concetto di ***fuzziness***).

Fuzziness di un insieme. La "vicinanza" di un insieme fuzzy A ad un insieme classico fornisce in qualche modo una misura di quanto l'insieme sia fuzzy. È intuitivo capire che non tutti gli insiemi sono fuzzy allo stesso modo, si pensi al caso di un insieme che sia crisp per tutti i suoi elementi tranne uno. Più difficile è quantificare il contenuto di fuzziness di un insieme, o dire con certezza che un insieme è più fuzzy di un altro. A tal fine sono state proposte diverse misure. Una delle più semplici e più usate è l'*indice di fuzziness*, secondo cui un insieme è tanto più fuzzy quanto più si discosta (in termini di distanze metriche) dall'insieme crisp ad esso più vicino. Evidentemente, un insieme crisp deve avere fuzziness nulla, mentre un livello di fuzziness massimo si può ottenere solo con un insieme A avente $\mu_A(x)=0{,}5$ per ogni elemento dell'universo X.

Normalizzazione. Questa operazione permette di rendere normale un insieme fuzzy A, portando a uno il massimo della sua funzione di appartenenza. Per fare ciò, è sufficiente determinare il massimo attuale della funzione di appartenenza di A e dividere tutta la funzione per questo valore:

$$m_{NORM(A)}(x) = m_A(x) / \max_X (m_A(x)) \quad \forall x \in X \tag{1.15}$$

Si può notare come l'operazione non apporti nessuna modifica ad un fuzzy set che sia già normale.

Concentrazione. Un fuzzy set A si può "concentrare" modificando la sua funzione di appartenenza in modo tale da aumentare il divario, in termini di grado di appartenenza, tra gli elementi aventi membership più elevata e gli altri. Una concentrazione di A si può ottenere ad esempio facendo il quadrato della funzione di appartenenza originale, in modo che le μ_A più alte si riducano meno:

$$m_{CON(A)}(x) = (m_A(x))^2 \quad \forall x \in X \tag{1.16}$$

Diluizione. È l'operazione opposta alla concentrazione, e permette di ridurre la concentrazione della funzione di appartenenza aumentando in modo più consistente i valori di membership degli elementi con grado di appartenenza minore:

$$m_{DIL(A)}(x) = \sqrt{m_A(x)} \quad \forall x \in X \tag{1.17}$$

Esiste anche l'operatore di intensificazione, combinazione dei precedenti, che concentra i valori di membership di quegli elementi aventi grado di appartenenza inferiore a

0,5 e diluisce quelli degli elementi aventi membership maggiore di 0,5.

Gli ultimi operatori descritti corrispondono a dei modificatori linguistici da applicare ai valori delle variabili. Come si vedrà in seguito, essi permettono di aumentare la versatilità d'uso dei termini linguistici che costituiscono le variabili stesse.

Somma algebrica. La somma algebrica di due fuzzy set A e B definiti in X è ancora un fuzzy set di X, con funzione di appartenenza:

Nella **somma algebrica**, la sottrazione del prodotto dalla somma fa in modo che la funzione di appartenenza risultante non superi mai l'unità.

$$m_{A+B}(x) = m_A(x) + m_B(x) - m_A(x)m_B(x) \quad \forall x \in X \tag{1.18}$$

Prodotto algebrico. Il prodotto algebrico di due fuzzy set A e B definiti in X è ancora un fuzzy set di X, avente funzione di appartenenza:

$$m_{A \cdot B}(x) = m_A(x) \cdot m_B(x) \quad \forall x \in X \tag{1.19}$$

Somma limitata. La somma limitata di A e B definiti in X è un fuzzy set di X avente funzione di appartenenza:

Anche **somma limitata** e **prodotto limitato** sono definiti in modo da rispettare le prerogative degli insiemi fuzzy, "tagliando" i valori superiori ad uno o inferiori a zero.

$$m_{A \oplus B}(x) = \min(1, m_A(x) + m_B(x)) \quad \forall x \in X \tag{1.20}$$

Prodotto limitato. Il prodotto limitato di A e B definiti in X è ancora un insieme di X, con funzione di appartenenza:

$$m_{A \otimes B}(x) = \max(0, m_A(x) + m_B(x) - 1) \quad \forall x \in X \tag{1.21}$$

Le operazioni viste si possono estendere al caso in cui gli insiemi A e B non sono definiti nello stesso universo del discorso. Per fare ciò, si devono prima definire i concetti di norma e conorma triangolare, che permetteranno anche di introdurre le relazioni fuzzy.

Una **norma triangolare** è una funzione T avente dominio $[0,1]\times[0,1]$ e codominio $[0,1]$, tale da soddisfare le seguenti proprietà:

- commutativa: $T(a,b) = T(b,a)$
- associativa: $T(a,T(b,c)) = T(T(a,b),c)$
- monotonia: $T(a,b) > T(c,d)$ se $a > c$ e $b > d$
- identità: $T(a,1) = a$

Fra le possibili T ci sono alcune operazioni definite in precedenza:
- intersezione fuzzy: $a \wedge b = \min(a,b)$
- prodotto algebrico: $a * b = ab$
- prodotto limitato: $a \otimes b = \max(0, a+b-1)$

Una **conorma triangolare** è una funzione T' avente dominio $[0,1]\times[0,1]$ e codominio $[0,1]$, tale da soddisfare le stesse proprietà valevoli per la norma triangolare. L'unica variazione è la sostituzione dell'elemento neutro 1 con l'elemento neutro 0, e la conseguente modifica della proprietà d'identità:
- identità: $T'(a,0) = a$

Anche tra le T' ritroviamo operazioni già viste per fuzzy set dello stesso universo:
- unione fuzzy: $a \vee b = \max(a,b)$
- somma algebrica: $a \bullet b = a+b-ab$
- somma limitata: $a \oplus b = \min(1, a+b)$

1.6 RELAZIONI FUZZY

In senso classico, una relazione n-aria è un sottoinsieme del prodotto cartesiano $X_1 \times ... \times X_n$, ossia un insieme di n-uple ordinate $x_1,...,x_n$ con $x_i \hat{I} X_i$. In analogia alla definizione data per i fuzzy set, si possono definire il prodotto cartesiano fuzzy e la relazione fuzzy come estensioni dei corrispondenti concetti crisp.

Se $A_1,...,A_n$ sono fuzzy set definiti rispettivamente in $X_1,...,X_n$, il loro **prodotto cartesiano** è un fuzzy set definito in $X_1 \times ... \times X_n$, e descritto da una funzione di appartenenza ottenuta applicando una norma triangolare; in genere si utilizzano le due norme più comuni, ossia l'intersezione fuzzy e il prodotto algebrico:

$$m_{A_1,...A_n}(x_1,...,x_n) = \min\left(m_{A_1}(x_1),...,m_{A_n}(x_n)\right) \qquad (1.22)$$

$$m_{A_1,...A_n}(x_1,...,x_n) = m_{A_1}(x_1) \cdot ... \cdot m_{A_n}(x_n) \qquad (1.23)$$

Una **relazione fuzzy** n-aria R è un fuzzy set in $X_1 \times ... \times X_n$ definito da una funzione di appartenenza μ_R: $X_1 \times ... \times X_n \rightarrow [0,1]$. Si può quindi scrivere:

$$R = \left\{(x_1,...,x_n), m_R(x_1,...,x_n) \mid (x_1,...,x_n) \in X_1 \times ... \times X_n\right\} \qquad (1.24)$$

Per costruire relazioni fuzzy binarie, si possono applicare a due fuzzy set *A* e *B*, definiti negli universi *X* e *Y*, degli operatori di norma triangolare (*T*) o conorma triangolare (*T'*), ottenendo delle relazioni dette rispettivamente congiunzione e disgiunzione fuzzy:

congiunzione fuzzy

$$m_{AandB}(x,y) = (m_A(x))T(m_B(y)) \tag{1.25}$$

disgiunzione fuzzy

$$m_{AorB}(x,y) = (m_A(x))T'(m_B(y)) \tag{1.26}$$

Grazie agli stessi operatori si possono definire inoltre delle operazioni sulle relazioni fuzzy. Considerando due relazioni binarie *R* e *S* definite in $X \times Y$, si può ottenere la loro **unione** tramite l'uso di una qualsiasi conorma triangolare; applicando ad esempio l'operatore di unione usato per gli insiemi fuzzy, si ottiene:

$$m_{R \cup S}(x,y) = \max(m_R(x,y), m_S(x,y)) \quad \forall (x,y) \in X \times Y \tag{1.27}$$

Allo stesso modo la loro **intersezione** si effettua applicando una norma triangolare, in questo caso l'operatore di minimo (intersezione):

$$m_{R \cap S}(x,y) = \min(m_R(x,y), m_S(x,y)) \quad \forall (x,y) \in X \times Y \tag{1.28}$$

È possibile definire anche l'operatore di **composizione** di due relazioni fuzzy definite su universi diversi. Siano *R* e *S* due relazioni fuzzy definite rispettivamente in $X \times Y$ e in $Y \times Z$, la loro composizione è un insieme fuzzy $R \circ S$ definito nell'universo $X \times Z$, avente funzione di appartenenza:

$$m_{R \circ S}(x,z) = \max_Y((m_R(x,y))T(m_S(y,z))) \quad \forall (x,y,z) \in X \times Y \times Z \tag{1.29}$$

Nel definire le relazioni fuzzy, si può scegliere tra diversi operatori di norma triangolare *T*. In genere sono preferiti l'operatore di minimo o il prodotto algebrico, che danno origine ai due tipi di composizione più usati:

composizione max-min

$$m_{R\circ S}(x,z)=\max_Y(\min(m_R(x,y),m_S(y,z))) \quad \forall(x,y,z)\in X\times Y\times Z \tag{1.30}$$

composizione max-product (o max-dot)

$$m_{R\circ S}(x,z)=\max_Y(m_R(x,y)\cdot m_S(y,z)) \quad \forall(x,y,z)\in X\times Y\times Z \tag{1.31}$$

Spesso l'operazione di composizione riguarda un insieme ed una relazione. Sia A un fuzzy set definito in X e R una relazione fuzzy definita in $X\times Y$. La composizione di A con R risulta in un fuzzy set B definito in Y ed esprimibile nei modi seguenti (usando ancora i due tipi di composizione visti sopra):

$$m_B(y)=\max_X(\min(m_A(x),m_R(x,y))) \quad \forall(x,y)\in X\times Y \tag{1.33}$$

$$m_B(y)=\max_X(m_A(x)\cdot m_R(x,y)) \quad \forall(x,y)\in X\times Y \tag{1.34}$$

1.7 REGOLE E IMPLICAZIONI FUZZY

Le conoscenze umane sono spesso empiriche, dovute all'esperienza, difficili da quantificare e codificare. La logica fuzzy è in grado di tradurre conoscenze di questo tipo in costrutti formali, direttamente elaborabili da un calcolatore.

La base della conoscenza di un sistema fuzzy è costituita da due componenti fondamentali: le funzioni di appartenenza delle variabili e l'insieme delle **regole d'inferenza fuzzy**. Le regole fuzzy rappresentano il punto di passaggio tra le conoscenze di tipo empirico descritte sopra e la loro elaborazione numerica. Tali regole sono qualitative, espresse con linguaggio naturale, ma costituiscono al tempo stesso una descrizione formale del sistema. Infatti, una volta messe in relazione con le funzioni di appartenenza, esse forniscono un modello del sistema puramente numerico, su cui può lavorare anche un calcolatore.

Una **regola d'inferenza** permette di associare a determinati valori delle variabili linguistiche d'ingresso (**antecedenti**) i corrispondenti valori delle variabili linguistiche d'uscita (**conseguenti**).

Una regola fuzzy è solitamente espressa con un costrutto del tipo `if-then`, e può presentare uno o più antecedenti e uno o più conseguenti. Una regola con un antecedente ed un conseguente assume quindi la seguente forma:

`if` x is A `then` y is B

Una regola di questo tipo è equivalente all'implicazione fuzzy $A \rightarrow B$. Un'implicazione fuzzy non rappresenta un'implicazione logica usuale con la corrispondente tabella di verità, ma piuttosto una relazione fuzzy sugli insiemi A e B. Un'**implicazione fuzzy** è, in effetti, una relazione vera e propria, e possiamo quindi scrivere:

$$\mu_{A \rightarrow B}(x, y) = \mu_A(x) \Im \mu_B(y) \tag{1.35}$$

con $\Im$ operatore d'implicazione. È importante disporre di diverse forme di implicazioni fuzzy, per poter scegliere quella che più si adatta al sistema su cui si sta lavorando. Le due funzioni d'implicazione più usate, ossia le implicazioni fuzzy *min* e *product*, sono anche le più semplici, e utilizzano come operatore d'implicazione rispettivamente la norma triangolare di minimo e quella di prodotto algebrico:

implicazione fuzzy min (di Mamdani)

$$\mu_{A \rightarrow B}(x, y) = \min\left(\mu_A(x), \mu_B(y)\right) \tag{1.36}$$

implicazione fuzzy product (di Larsen)

$$\mu_{A \rightarrow B}(x, y) = \mu_A(x) \cdot \mu_B(y) \tag{1.37}$$

Una regola fuzzy corrisponde quindi ad una relazione fuzzy, ed è proprio sulla relazione che si lavora quando si prende in considerazione la regola che la esprime.

Generalmente le regole fuzzy sono del tipo a più ingressi ed una uscita (MISO – multiple input, single output) o a più ingressi e più uscite (MIMO – multiple input, multiple output). Si presenta quindi la necessità di introdurre delle operazioni di collegamento tra i diversi antecedenti e tra i diversi conseguenti. Per poter considerare congiuntamente i diversi ingressi, si usano i connettivi `and` e `or`, mentre si utilizza il connettivo `also` per indicare che una regola presenta più uscite. La forma generale di una regola (ad esempio la k-esima nella base di conoscenza, avente n ingressi e m uscite) sarà quindi la seguente:

`if` x_1 is A_{k1} `and` ... `and` x_i is A_{ki} `or` ... `or` x_n is A_{kn}
`then` y_1 is B_{k1} `also` ... `also` y_m is B_{km}

Va detto che l'operazione di aggregazione `or` è raramente utilizzata nelle applicazioni pratiche, e verrà trascurata nella trattazione che segue. Se ne possono comunque ricavare le caratteristiche d'impiego dall'analogia con l'aggregatore `and`.

Osservando che il connettivo `and` e l'operatore d'implicazione `then` sono entrambi tradotti matematicamente da norme triangolari, si comprende come dal loro uso congiunto si possano ottenere diversi tipi di relazioni. Considerando solo le due norme più usate (minimo e prodotto algebrico), si hanno quattro combinazioni, ovvero quattro differenti forme per la relazione risultante. Le due forme più usuali sono quelle che utilizzano la stessa norma per intersezione e implicazione.

Prima di analizzare gli aspetti della teoria fuzzy più vicini al ragionamento umano, ne completiamo la descrizione formale, illustrando come avviene l'interazione tra le due componenti della base della conoscenza.

1.8 PROCESSO D'INFERENZA

Il processo d'inferenza permette di ricavare le grandezze in uscita applicando le regole d'inferenza ai valori noti degli ingressi. Ricordando che una regola fuzzy non è altro che una relazione tra antecedente e conseguente, si può comprendere come l'operazione più adatta ad agire da congiunzione tra regole e funzioni d'ingresso sia la loro composizione. Infatti, abbiamo già visto come, dalla composizione di una relazione R di $X \times Y$ con un fuzzy set A' di X, si possa inferire un fuzzy set B' di Y:

La scrittura $A' \circ R$ indica l'operazione di composizione tra la relazione R e il fuzzy set A.

$$B' = A' \circ R \qquad \text{con} \qquad R: A \rightarrow B$$

Chiamando A' l'antecedente e R la regola, si può ricavare il conseguente B' con una delle composizioni del paragrafo 1.6:

composizione max-min (1.33)

$$m_{B'}(y) = \max_X \left(\min \left(m_{A'}(x), m_R(x,y) \right) \right) \quad \forall (x,y) \in X \times Y$$

composizione max-product (1.34)

$$m_{B'}(y) = \max_X \left(m_{A'}(x) \cdot m_R(x,y) \right) \quad \forall (x,y) \in X \times Y$$

Servendoci di esempi grafici, analizziamo in dettaglio due leggi d'inferenza usate molto frequentemente.

La prima si ottiene dalla composizione *max-min* utilizzando per la regola *R* un'implicazione di minimo e sfruttando la proprietà associativa della norma triangolare (v. fig.1.6):

$$m_{B'}(y) = \max_X \left(\min\left(m_{A'}(x), m_A(x), m_B(y)\right)\right) \quad \forall (x,y) \in X \times Y \tag{1.38}$$

La seconda si può ricavare dalla prima sostituendo l'operatore di minimo con il prodotto algebrico, ossia abbinando una composizione *max-product* con una implicazione *fuzzy product* (fig.1.7):

$$m_{B'}(y) = \max_X \left(m_{A'}(x) \cdot m_A(x) \cdot m_B(y)\right) \quad \forall (x,y) \in X \times Y \tag{1.39}$$

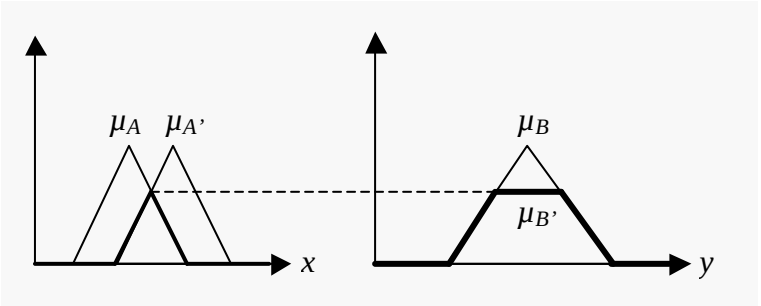


Fig. 1.6 Legge d'inferenza con composizione *max-min* ed implicazione *fuzzy min*.

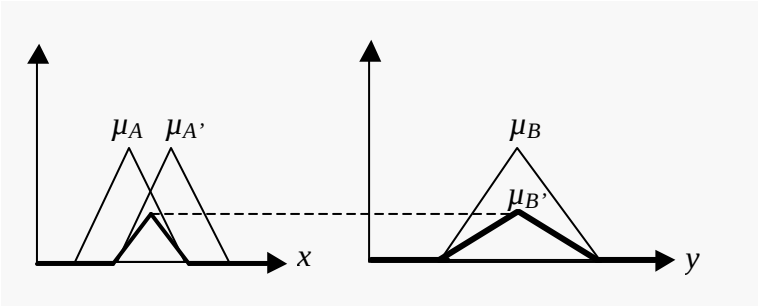


Fig. 1.7 Legge d'inferenza con composizione *max-product* ed implicazione *fuzzy product.*

In entrambi i casi il processo logico è il seguente. Per iniziare si interseca l'insieme d'ingresso *A'* con l'antecedente *A* dell'implicazione, tenendo conto che il termine intersezione può assumere significati diversi a seconda della norma prescelta. Si valuta poi il massimo rispetto a *x* della funzione ottenuta, ricavando così il valore di verità dell'ingresso rispetto alla regola, detto anche grado di attivazione della regola. Si interseca quindi que-

Nel processo d'inferenza, la scelta dell'operazione di composizione determina il modo in cui viene valutato il grado di verità dell'antecedente (intersezione tra l'insieme d'ingresso *A'* e l'antecedente *A* della regola). L'operatore prescelto per l'implicazione influisce invece sulla modalità con cui viene modificato l'insieme conseguente *B* della regola al fine ottenere l'insieme d'uscita *B'*.

sto valore numerico con il conseguente *B* dell'implicazione, assegnando in questo modo a *B* un'importanza data dal grado di verità dell'ingresso *A'*. La funzione di appartenenza finale, definita in *Y*, è quella dell'insieme d'uscita *B'*. Impiegando una implicazione *fuzzy product*, l'insieme *B* viene moltiplicato per il grado di verità dell'antecedente (fig. 1.7), mentre l'utilizzo di una implicazione *fuzzy min* comporta il troncamento dell'insieme, che diviene un trapezio avente come valore massimo il grado di verità dell'antecedente (fig.1.6).

Molto frequentemente si impiegano come ingressi dei fuzzy singleton normalizzati, cioè insiemi fuzzy nei quali un solo elemento dell'universo ha membership uno e tutti gli altri zero. Quando i parametri d'ingresso sono puntuali l'introduzione di ingressi singleton non toglie significato al processo, e permette di semplificare molto i calcoli. Nel caso si utilizzino fuzzy singleton risulta inoltre indifferente la scelta tra i due tipi di composizione, come si può vedere dall'esempio di fig. 1.8, in cui l'implicazione utilizzata è la *fuzzy min*.

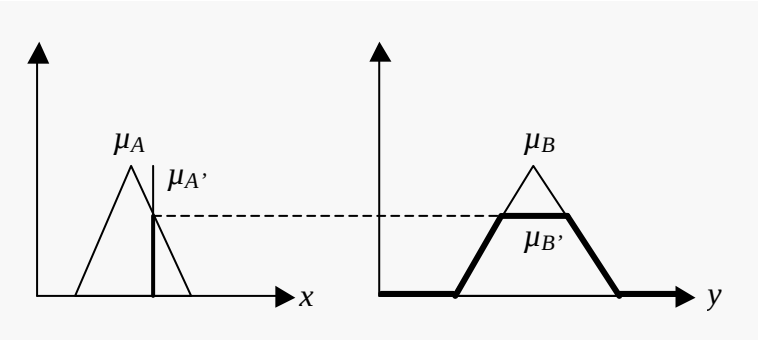


Fig. 1.8 Inferenza con fuzzy singleton.

Utilizzando fuzzy singleton, oltre all'operatore di implicazione, si deve comunque scegliere anche l'operatore di intersezione per l'`and` logico. Analizziamo una regola con due ingressi A_1' e A_2' nella quale vengono impiegati per l'`and` prima l'operatore di minimo, poi il prodotto algebrico, ricordando che si possono usare anche altre norme triangolari. L'implicazione scelta è in entrambi i casi la *fuzzy-product*.

Si può notare, dal confronto delle fig. 1.9 e 1.10, che l'impiego del prodotto algebrico come operatore d'intersezione produce un insieme d'uscita più "schiacciato", in cui il peso relativo dei valori massimi è minore.

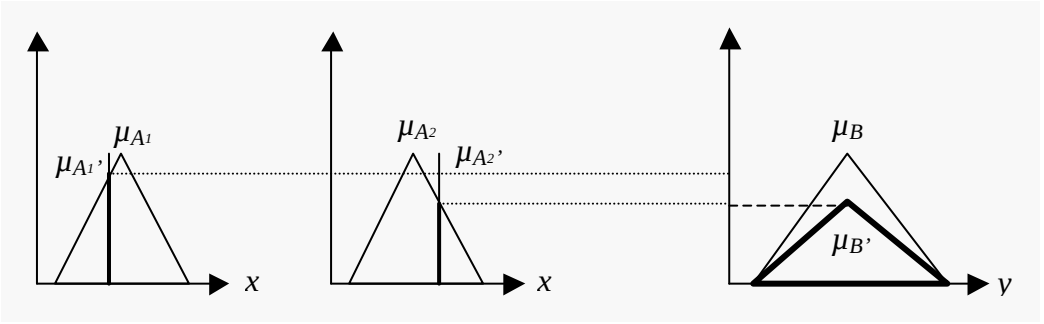


Fig. 1.9 and eseguito con operatore di minimo.

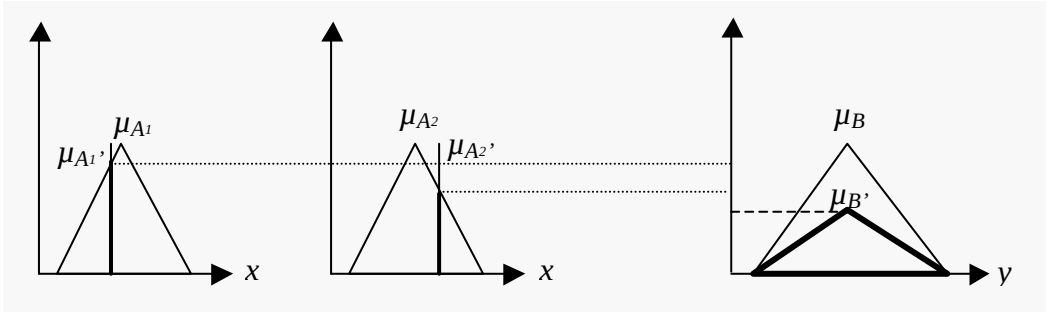


Fig. 1.10 and eseguito con prodotto algebrico.

1.9 VARIABILI LINGUISTICHE

Una variabile linguistica è una variabile *fuzzy*, che non assume valori numerici, ma valori espressi in linguaggio naturale. I valori che una variabile linguistica può assumere sono detti termini linguistici. Ad esempio, alla variabile linguistica "febbre" potremmo assegnare i valori: *assente, leggera, media, alta, altissima*. L'insieme dei termini relativi ad una certa variabile è detto *term set*. Ogni termine è associato alla variabile linguistica cui si riferisce tramite la sua funzione di appartenenza. L'universo del discorso della variabile è coperto da tutte le funzioni di appartenenza dei termini ad essa associati.

Nell'esempio relativo alla variabile linguistica febbre (fig. 1.11) si può notare che le funzioni di appartenenza non hanno necessariamente la stessa forma e che i termini linguistici estremi coprono anche le aree di più improbabile utilizzo nell'universo del discorso. I *term set* sono costruiti in modo da tenere in debito conto anche le condizioni di lavoro più insolite. Molto importante è anche la sovrapposizione dei termini linguistici, necessaria affinché il sistema si comporti in maniera *fuzzy*, attivando più regole contemporaneamente.

Interessante è la possibilità di modificare le funzioni di appartenenza degli insiemi con l'impiego di **modificatori**

linguistici. I più diffusi sono *molto*, *poco* e *non*, che corrispondono rispettivamente alle operazioni di concentrazione, diluizione e complemento definite nel paragrafo 1.5. Ad esempio, passare dall'insieme che descrive il valore *alta* a quello che rappresenta il valore *molto alta* equivale ad applicare l'operazione di concentrazione all'insieme di partenza.

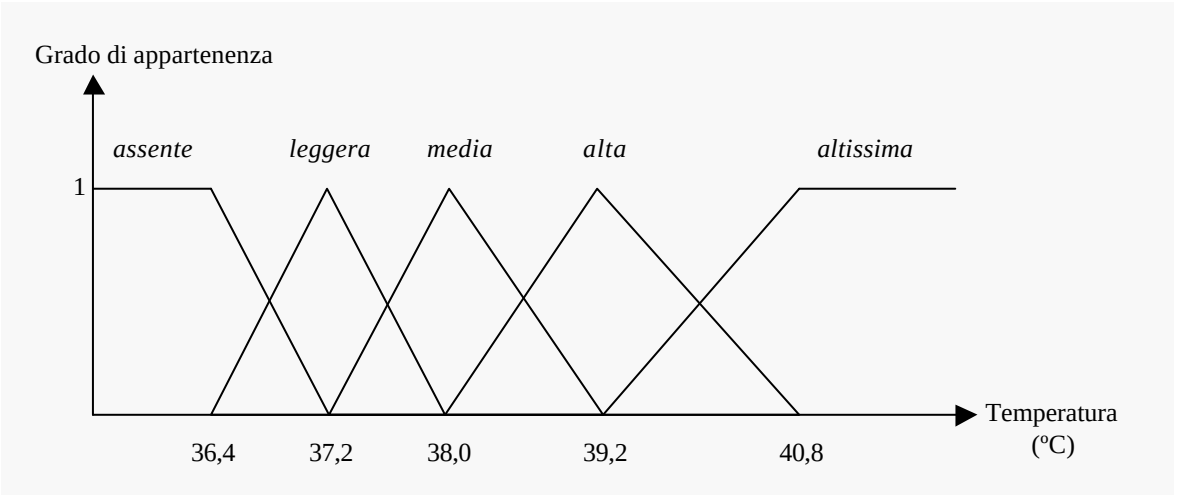


Fig. 1.11 *Term set* della variabile "febbre", con funzioni di appartenenza.

1.10 STRUTTURA DI UN SISTEMA FUZZY

Un sistema fuzzy è costituito da quattro unità fondamentali (fig.1.12), ossia la base di conoscenza e le 3 unità di calcolo: *fuzzificazione*, inferenza e *defuzzificazione*.

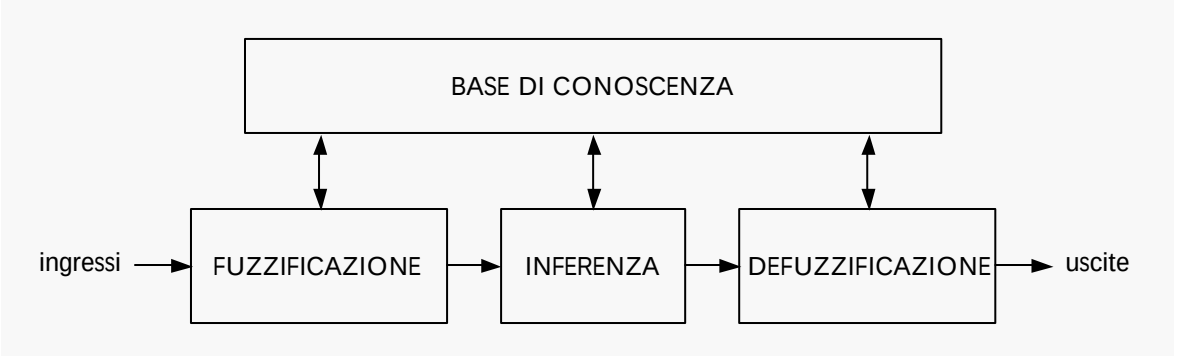


Fig. 1.12 Struttura generale di un sistema fuzzy con le interazioni tra le unità costitutive.

La **base di conoscenza** contiene tutte le informazioni sul sistema, che permettono alle altre unità di elaborare i dati d'ingresso per ottenere le uscite. Tali informazioni

possono essere suddivise nei due blocchi *data base* e *rule base*. Il primo comprende la descrizione di tutte le variabili con le loro funzioni di appartenenza, il secondo le regole linguistiche d'inferenza.

Dato che quasi sempre i dati in ingresso sono in forma crisp e che il sistema fuzzy lavora su insiemi "sfumati" è necessario operare una conversione, per tradurre un dato numerico standard in un dato fuzzy. L'operazione che attua questa conversione si chiama **fuzzificazione**. La fuzzificazione si effettua sfruttando le funzioni di appartenenza dei fuzzy set relativi alla variabile da trattare. Per un valore *crisp* d'ingresso vengono stabiliti dei gradi di appartenenza relativi ognuno ad un termine linguistico della variabile.

Il **motore d'inferenza** è il cuore del sistema fuzzy. Utilizzando le informazioni contenute nella base di conoscenza, esso determina lo stato delle uscite corrispondente ad una determinata configurazione degli ingressi. Si sono già visti diversi tipi di processi d'inferenza, che operano in ogni caso su ingressi fuzzy, restituendo valori fuzzy in uscita.

L'operazione che converte i valori fuzzy d'uscita in valori numerici utilizzabili è detta **defuzzificazione**. In questa fase, partendo da un particolare insieme fuzzy ottenuto con l'inferenza (spesso di forma irregolare, dovuta all'unione dei risultati di regole diverse), si vuole determinare un singolo valore *crisp* che rappresenti nel miglior modo possibile tale insieme. Il valore ottenuto rappresenta l'uscita finale del sistema, e sarà usato come azione di controllo o come parametro decisionale.

Sono state elaborate varie strategie di defuzzificazione, nessuna delle quali si può considerare conveniente in assoluto: sta al progettista stabilire qual è la metodologia che meglio si adatta alle esigenze poste dal problema. Tra le strategie più usate si possono citare i metodi di massimo ed il metodo del baricentro.

Nei **metodi di massimo** viene scelto il punto in cui la funzione di appartenenza dell'insieme d'uscita raggiunge il suo massimo. Se i punti di massimo sono molteplici, è possibile decidere se scegliere il primo (*metodo del primo massimo*, PM nella fig. 1.13), l'ultimo (*metodo dell'ultimo massimo*, UM) o la loro media (*metodo della media dei massimi*, MM). Tali metodi si utilizzano nel caso si voglia scegliere, tra diverse possibilità, il risultato più plausibile, trascurando eventuali contributi di minor peso.

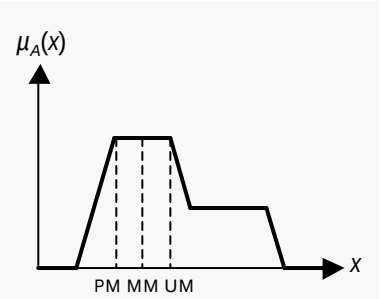


Fig. 1.13 Defuzzificazione con i metodi di massimo.

Il **metodo del baricentro** è anche detto *metodo del centroide* o *COG* (*center of gravity*). Il valore calcolato è l'ascissa del baricentro della figura formata dalla funzione

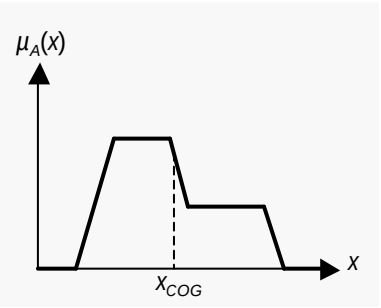


Fig. 1.14 Defuzzificazione con il metodo del baricentro

di appartenenza (fig. 1.14). Essendo l'insieme inferito, supposto continuo, descritto da $\mu_A(x)$, l'uscita x_{COG} sarà:

$$x_{COG} = \frac{\int_X m_A(x) x dx}{\int_X m_A(x) dx} \tag{1.40}$$

Il metodo del baricentro è preferibile quando si cerca la miglior soluzione di compromesso, poiché tiene conto anche dei contributi dati dalle regole meno influenti. È in genere il più usato nei controlli, ma risulta oneroso dal punto di vista computazionale. Una variante che permette calcoli più rapidi è il *metodo COM* (*center of maximum*), ossia il calcolo della media ponderata dei diversi massimi relativi della $\mu_A(x)$.

1.11 PROGETTAZIONE DI UN SISTEMA FUZZY

Per approfondimenti sui criteri di progettazione di un sistema fuzzy, si può fare riferimento a Driankov [4].

Le decisioni da prendere nel corso della progettazione di un sistema fuzzy sono molte, e spesso correlate. Viste le continue interazioni tra le varie fasi del sistema, la seguente descrizione rappresenta una sequenza decisionale che, pur essendo logica, non può essere considerata strettamente cronologica.

1.11.1 Analisi del problema

Molto spesso, un sistema fuzzy è composto da diversi blocchi d'inferenza interagenti tra loro. Si avranno quindi delle variabili d'ingresso, delle variabili d'uscita e delle variabili intermedie di collegamento tra i blocchi. In un sistema avente dei blocchi di calcolo posti in sequenza le uscite dei blocchi a monte vengono impiegate come ingressi dei blocchi a valle. La prima fase della progettazione consiste nella definizione delle variabili e del loro ruolo all'interno del sistema, precisando ingressi e uscite di ogni blocco e legami tra i blocchi.

È opportuno ricordare che spesso, per problemi complessi, le soluzioni migliori si ottengono con sistemi ibridi, contenenti cioè blocchi fuzzy, blocchi lineari ed eventualmente anche moduli di apprendimento (algoritmi genetici o reti neurali). Completata la fase di analisi si dovrebbe poter disporre di uno schema generale del sistema da implementare, eventualmente rivedibile in base a *feedback* forniti dalle fasi successive.

Elementi da definire nella fase di analisi:

- variabili (ingressi, uscite e variabili intermedie);
- struttura generale del sistema e dei suoi blocchi d'inferenza;

Le fasi seguenti della progettazione devono essere attuate per ogni blocco fuzzy costituente il sistema, e sono: la definizione delle caratteristiche degli ingressi, la defini-

zione delle caratteristiche delle uscite, la progettazione dettagliata del motore d'inferenza.

1.11.2 Definizione delle caratteristiche degli ingressi

Si tratta dapprima di definire l'universo del discorso di ogni variabile, ossia l'intervallo in cui si presume siano contenuti tutti i valori che la variabile può assumere. Se si utilizzano degli intervalli normalizzati, utili per impiegare un unico sistema fuzzy in condizioni diverse, è necessaria la fase di *pre-elaborazione*, in cui si riconducono gli ingressi effettivi agli intervalli di definizione impostati. Per eseguire correttamente la pre-elaborazione è perciò necessario definire accuratamente i fattori di scala relativi a ciascuna variabile.

Si devono poi stabilire il numero di termini di ogni variabile e la forma delle funzioni di appartenenza. In linea di massima, utilizzare meno termini del necessario porta a sistemi pigri, poco sensibili alle variazioni. Viceversa, se i termini sono troppi, si rischia di ottenere un sistema frenetico, instabile. I numeri tipici di funzioni di membership usate sono tre, cinque, e sette (dispari, perché è spesso preferibile disporre di un valore centrale).

Per quanto riguarda le funzioni di appartenenza, si è verificato che funzioni a campana o gaussiane, con derivata prima continua, imitano molto bene la percezione umana, e forniscono variazioni delle uscite più graduali. Tuttavia, il costo computazionale ad esse associato può essere notevole, e nei sistemi in cui il tempo di risposta è importante vengono generalmente utilizzate forme triangolari e trapezoidali. Anche nei sistemi con piccoli microprocessori fuzzy dedicati vengono privilegiate forme più semplici, per questioni di memoria.

Molto importante è il grado di sovrapposizione tra le funzioni di appartenenza. La sovrapposizione fa in modo che vengano attivate contemporaneamente più regole, e ciò è indispensabile affinché l'uscita sia fuzzy e non reale. La sovrapposizione tipica è quella che fa corrispondere il punto di massimo di un termine con i limiti dei supporti dei termini adiacenti.

Un ultima decisione da prendere riguardo agli ingressi è la scelta dell'operatore di fuzzificazione. Per ingressi puntuali, abbiamo visto come l'utilizzo di fuzzy singleton semplifichi i calcoli senza ridurre, almeno dal punto di vista teorico, il contenuto d'informazione disponibile. Operatori diversi, rappresentati dalle usuali categorie di funzioni d'appartenenza (triangolo, trapezio, campana), ga-

Elementi da definire in relazione agli ingressi:

- Intervalli di esistenza della variabili e fattori di scala per la pre-elaborazione;
- numero dei termini di ogni variabile e loro funzioni di appartenenza;
- operatore di fuzzificazione;

rantiscono tuttavia un migliore comportamento fuzzy da parte del sistema, con l'attivazione contemporanea di più regole, ma richiedono un costo computazionale spesso molto maggiore.

1.11.3 Definizione delle caratteristiche delle uscite

Anche nella definizione delle variabili d'uscita il primo passo è quello di stabilire gli insiemi di esistenza, ed eventualmente i fattori di scala necessari alla *post-elaborazione* (questa volta si passa dai valori normalizzati a quelli effettivi).

Per quanto riguarda il numero delle funzioni di appartenenza valgono le considerazioni fatte sopra, mentre la forma scelta per esse è quasi sempre quella triangolare. Si è infatti osservato che l'utilizzo di forme più complesse e dispendiose per l'elaborazione non conduce a variazioni degne di nota.

Molto più importante è invece la scelta del metodo di defuzzificazione, spesso successiva alla definizione delle regole. Gli orientamenti principali sono due. A volte è preferibile scegliere il valore che meglio rappresenta l'intero insieme risultante, tenendo conto di tutti i contributi, seppur minimi. Altre volte si rende necessaria una scelta più radicale, in cui una soluzione di compromesso non avrebbe senso (ad esempio quando, nell'evitare un ostacolo, si deve decidere se girare a sinistra o a destra). Nei casi del primo tipo si impiegano metodi come quello del baricentro, mentre i metodi di massimo sono più adatti al secondo genere di problemi. Tuttavia, si riscontrano spesso vincoli alla libertà di scelta, posti dalla limitata capacità di calcolo o dal breve tempo di risposta necessario. Metodi affini al *COM* (centro dei massimi) possono rivelarsi buone soluzioni di compromesso in questi casi.

Elementi da definire in relazione alle uscite:

- intervalli di definizione delle variabili e fattori di scala per la post-elaborazione;
- numero dei termini di ogni variabile e loro funzioni di appartenenza;
- operatore di defuzzificazione;

Per quanto riguarda le variabili intermedie, spesso non c'è l'esigenza di pre-elaborazione e post-elaborazione, visto che escono normalizzate da un blocco e possono rientrare ancora normalizzate nel blocco successivo. A volte, si può utilizzare direttamente come ingresso del secondo blocco l'insieme inferito dal primo, senza effettuare le operazioni di defuzzificazione e rifuzzificazione del risultato ottenuto.

1.11.4 Progettazione del motore d'inferenza

Il modo in cui le variabili si influenzano tra loro risulta fissato, almeno a grandi linee, già dalla fase di analisi. Una volta definiti i *term set* di ogni variabile, si possono precisare nel dettaglio i diversi meccanismi d'interazione, defi-

nendo le regole del motore d'inferenza.

La prima stesura della *rule base* si fa di solito in base all'analisi di dati storici, di osservazioni empiriche e di interviste ad esperti. Questa stesura viene in genere ampiamente rivista e completata sulla scorta dei risultati che si ottengono nella successiva fase di test.

Il modo di definire le regole è molto soggettivo, anche se è frequente un tipo d'impostazione che privilegia la codifica delle regole più ovvie e affidabili, lasciando per un secondo momento il completamento del motore d'inferenza. Spesso, grazie ad una consistente sovrapposizione tra le funzioni di appartenenza definite, è anche possibile lasciare incompleta la base di regole. Nei sistemi in cui le regole da scrivere sarebbero molte questa risulta essere effettivamente la prassi più diffusa.

Un blocco fuzzy avente 3 variabili linguistiche d'ingresso, ognuna composta di 5 fuzzy set, può presentare un totale di 5^3=125 regole. In realtà, un numero di regole compreso tra 20 e 40 risulta in genere sufficiente a costruire una base di conoscenza del tutto esauriente.

Alcuni ambienti di progettazione permettono di assegnare dei pesi alle regole, in modo da decidere preventivamente quali devono essere le regole di maggiore influenza. Quando possibile, questa operazione viene di frequente effettuata da algoritmi di apprendimento (il loro impiego è descritto nel capitolo 7).

Oltre alla *rule base*, per completare la base di conoscenza è necessario scegliere gli operatori da impiegare nell'aggregazione degli ingressi, nella composizione e nell'implicazione.

Le norme disponibili sono molte, e talvolta i progettisti ne introducono di nuove se ritengono che possano adattarsi meglio di quelle esistenti alle esigenze del problema. Nondimeno, nella grande maggioranza delle applicazioni reali si utilizzano le soluzioni più semplici, come gli operatori di minimo e di prodotto. Soprattutto per quanto riguarda la composizione e l'implicazione, sembra che in genere non sia giustificata l'adozione di procedure più complesse.

Fanno eccezione gli operatori utilizzati per le operazioni di aggregazione. La più usuale tra esse è l'and logico tra gli antecedenti, che stabilisce il modo in cui i diversi ingressi interagiscono tra loro per fornire il grado di verità della regola. L'impiego in questo senso dell'operatore di minimo equivale a prendere in considerazione solo l'antecedente con il grado di verità inferiore. Non si tiene così conto che gli altri ingressi possono assumere valori molto diversi, senza per questo modificare minimamente il risultato. D'altro canto, l'utilizzo dell'operatore di prodotto, che moltiplica i valori di verità di ogni antecedente, fornisce spesso valori di verità finali molto bassi che tendono ad appiattire l'insieme d'uscita, e questo inconveniente si accentua rapidamente al crescere

Elementi da definire in relazione al motore d'inferenza:

- regole (antecedenti, conseguenti, eventuali pesi);
- operatori di aggregazione, composizione ed implicazione;

del numero degli ingressi. Una soluzione di compromesso è l'aggiunta di una *compensazione* al risultato ottenuto con l'operatore di minimo. La **compensazione** si attua sovrapponendo gli effetti di diversi operatori, ad esempio con una media pesata calcolata tra i risultati parziali forniti da essi.

Sono stati definiti diversi **operatori di compensazione**. In genere richiedono l'impostazione di un parametro, il quale determina il peso relativo delle componenti dell'operatore. Assegnando al parametro i suoi valori estremi (di solito zero e uno) si ottengono degli operatori "puri".

Durante tutta la progettazione di dettaglio non si deve mai perdere di vista l'interdipendenza tra i vari blocchi, ma è nella prossima fase di messa a punto che il sistema torna ad essere effettivamente considerato come un tutt'uno.

1.11.5 Messa a punto del sistema

La messa a punto del sistema è una fase fondamentale del lavoro. Essa ha lo scopo di analizzare il comportamento del sistema rispetto alle diverse possibili situazioni che potrebbero verificarsi durante il suo funzionamento. In alcuni casi può essere un'operazione onerosa poiché richiede l'inserimento di dati, la loro analisi e soprattutto la valutazione dei risultati.

Non di rado, le informazioni ricavate nella fase di verifica vengono impiegate per modificare e riadattare il sistema, nel tentativo di renderlo più adeguato al compito che deve svolgere.

La gamma di valori da impiegare per gli ingressi nella fase di messa a punto deve essere più ampia possibile, e deve comprendere anche i casi marginali, più improbabili o comunque posti in vicinanza dei limiti degli intervalli di definizione. È importante definire in anticipo le risposte che si vorrebbero ottenere dal sistema, per poter così giudicare oggettivamente l'effettiva validità del suo comportamento. Dal confronto tra i risultati attesi e quelli realmente forniti dall'elaborazione è possibile comprendere quali sono i punti deboli del sistema, ed analizzando i risultati parziali forniti da ogni blocco si può circoscrivere la zona del sistema, o addirittura il singolo blocco, che fornisce i risultati non previsti.

A questo punto, identificate le parti da modificare, si deve trovare un'alternativa che fornisca risposte più corrette. A volte, questo può essere fatto solo per tentativi, anche se una conoscenza dettagliata del funzionamento del sistema permette di ridurre la gamma di alternative plausibili. In certi casi, può essere utile in questa fase l'impiego di algoritmi di apprendimento, che richiede però di prestabilire almeno in parte i risultati numerici che si vogliono ottenere.

Avendo a che fare con sistemi esperti, il cui compito è quello di prendere delle decisioni autonome in riferimento

a determinate situazioni d'incertezza, può essere difficile o addirittura privo di significato stabilire in partenza i risultati attesi. Più agevole è valutare a posteriori la ragionevolezza delle decisioni prese dal sistema. Dapprima questo può essere fatto in modo soggettivo dal progettista sulla base di una mole ridotta di dati, ma si dovranno in seguito stabilire degli indici di valutazione che permettano di quantificare più precisamente le prestazioni fornite dall'elaborazione. La prima verifica soggettiva permette di identificare e correggere le anomalie più macroscopiche, in modo che la fase di controllo quantitativa possa concentrarsi sull'ottimizzazione delle prestazioni, lavorando su un sistema che funziona già in modo sostanzialmente corretto.

Terminata la fase di messa a punto, se il progetto dimostra di avere un interesse applicativo, basta disporre le interfacce necessarie all'utilizzo da parte dell'utente finale, ed il sistema è pronto per essere messo in funzione. In questo caso, il funzionamento sul campo può essere la migliore verifica del comportamento del sistema, che dovrà presentare caratteristiche di versatilità tali da permettere ulteriori modifiche e miglioramenti.

2

LA GESTIONE DELLE SCORTE DI MAGAZZINO

2.1 INTRODUZIONE

Le tematiche aziendali connesse ai problemi di organizzazione e gestione del magazzino, nonché del finanziamento delle scorte di materie prime, semilavorati e prodotti finiti, costituiscono uno dei maggiori e più interessanti filoni di analisi nel campo economico-aziendale. Tale interesse è in particolar modo suscitato dall'elevato tasso di innovazione in questa specifica area aziendale, che ha subito diversi cambiamenti nel corso del tempo, assumendo anche la forma di laboratorio per possibili forme di intervento specifico in altre aree.

La visione dei compiti del magazzino era dapprima di tipo esclusivamente pratico. Le principali operazioni consistevano infatti nelle procedure operative e gestionali di rinnovo e scarico. È successivo il raggiungimento della consapevolezza che le scorte comportano autentici **costi opportunità** visto che, aumentando le giacenze a magazzino, si deve rinunciare ad altri investimenti redditizi. Si è passati quindi ad una configurazione sempre più dinamica e strutturata nell'ambito aziendale, in quanto il magazzino veniva ad assumere la veste di autentico centro di costo, la cui ottimizzazione era resa necessaria a fronte del problema più ampio della redditività d'impresa. Il processo che è seguito si è sviluppato nella generazione di modelli sempre più sofisticati, il cui obiettivo principale consisteva non solo nell'ottimizzare la liquidità, ma anche nell'introdurre tecniche di gestione innovative. Tali tecniche hanno portato ad una sempre maggiore automazione delle operazioni di magazzino e delle attività operative a supporto.

Sono detti **costi opportunità** gli introiti cui si rinuncia decidendo di investire dei capitali in una certa attività a scapito di un'altra, la cui redditività risulta nota con un buon margine di sicurezza (per esempio i depositi bancari).

L'introduzione di metodologie innovative nell'area della produzione (come il ***just in time***) e di modelli di attenzione alla qualità (***lean production***) ha generato una visione dinamica del magazzino aziendale. Secondo questa nuova visione si sono introdotti modelli organizzativi orientati ad una maggiore qualità nella gestione delle scorte, riducendo quei costi operativi diretti e indiretti che

Just in time e ***lean production*** (produzione snella) sono due filosofie di gestione, nate nell'industria giapponese, che puntano ad ottenere i massimi risultati con il minimo impiego di risorse. Questo obiettivo viene perseguito tramite la massima integrazione verticale ed orizzontale, la ricerca incessante della qualità a tutti i livelli e la tendenza all'apprendimento continuo.

spesso rendevano onerosa ed inefficiente la gestione globale degli approvvigionamenti.

Oggi assistiamo, in particolare nelle attività di assemblaggio e nella componentistica, a forme di magazzino pressoché *leggere* ed accentrate, che cercano l'economicità nella gestione associata al mantenimento di standard qualitativi nel servizio al cliente.

È in questa prospettiva che si inserisce il progetto di gestione qui proposto, che prevede l'inserimento di una filosofia di controllo innovativa, com'è la logica fuzzy, in un campo che ha visto spesso convivere criteri analitici piuttosto rigidi con soluzioni empiriche dovute al buon senso di personale specializzato.

2.2 CARATTERISTICHE DELLA GESTIONE SCORTE

Come detto sopra, la gestione ideale è quella che minimizza i costi garantendo il desiderato grado di servizio al cliente. Tuttavia, gli ingredienti di questa ricetta sono molteplici e vari, e non sembra possibile fissare in modo univoco gli obiettivi specifici da perseguire per ottenere il modello di gestione ideale. Le modalità di produzione, di approvvigionamento e di vendita sono molteplici e di importanza basilare nello stabilire le metodologie di gestione delle scorte.

Anche la posizione di un magazzino all'interno del ciclo produttivo è determinante per stabilire i requisiti da soddisfare con il suo utilizzo. Esistono magazzini di materie prime e di semilavorati acquisiti dall'esterno, che riflettono gli sfasamenti temporali e quantitativi tra l'acquisizione degli input da parte dei produttori e la loro immissione nel processo; si utilizzano poi scorte anche per quei prodotti semilavorati che, a causa dei tempi tecnici di produzione, si trovano in una fase intermedia del processo di trasformazione; infine, le scorte di prodotti finiti vengono tenute da produttori o commercianti in vista della vendita agli utenti finali.

In alcuni casi, l'organizzazione del processo produttivo punta esplicitamente alla massima riduzione delle scorte. Ad esempio, nella produzione su commessa si lavora sul venduto, e si elimina quindi ogni problema di gestione del magazzino prodotti finiti. D'altra parte, anche le materie prime impegnate sono di frequente fornite in conto-lavori dal cliente. Tutto ciò consente di contenere in modo notevole anche il magazzino materie prime. È invece inevitabile per chi produce su commessa il mantenimento di un forte immobilizzo in semilavorati.

Di segno opposto sono gli impegni di chi produce in continuo: qui la filosofia di progettazione degli impianti ruota attorno al concetto di prodotto standard realizzato in tempi estremamente brevi, e ciò significa la quasi inesistenza di semilavorati a scorta. Anche gli *stock* di materie prime possono essere contenuti, perché i fabbisogni sono noti e generalmente non subiscono oscillazioni notevoli. Le difficoltà maggiori riguardano invece i prodotti finiti, dove le scorte possono accumularsi pericolosamente o viceversa non essere sufficienti a soddisfare un incremento delle richieste.

La maggior parte dei processi produttivi si pone in una posizione intermedia alle due situazioni estreme descritte, rendendo necessaria la massima attenzione per ogni punto del processo in cui vi sia accumulo di materiali.

La missione aziendale ed il posizionamento strategico dell'azienda sono parimenti decisivi nell'influenzare la filosofia di gestione delle scorte e nel definire gli obiettivi specifici da prefiggersi. Se si punta ad un soddisfacimento pressoché totale delle richieste dei clienti, è necessario il mantenimento di un magazzino prodotti finiti molto consistente. L'onere di gestione può essere notevole, ma il ritorno in termini d'immagine e l'instaurarsi di un rapporto di fedeltà da parte del cliente stesso, per altro difficilmente monetizzabili, potrebbero ripagare ampiamente la maggiorazione dei costi sostenuti. Viceversa, ricercare la massima economia di gestione, con annessa riduzione delle giacenze, può generare un vantaggio competitivo se si riesce comunque a mantenere una consistente quota di mercato. È evidente che in questi casi, e in tutte le situazioni intermedie, le decisioni riguardanti le scorte sono parte integrante del meccanismo di gestione.

Di seguito analizziamo più in dettaglio le caratteristiche della gestione scorte, illustrando la loro utilità, i costi associati, le decisioni richieste dalla loro conduzione, la valutazione delle prestazioni di gestione.

Nei testi di Marini e Presutti [15] e Schmenner [16] si possono trovare analisi più dettagliate delle problematiche associate alla gestione di scorte

2.2.1 Utilità delle scorte

Sono molteplici le motivazioni che inducono a tenere delle scorte. La filosofia di gestione dovrà basarsi sulla diversa importanza che si assegna ad ognuna di queste motivazioni. Elenchiamo di seguito le principali.

- Soddisfare le richieste facendo fronte a irregolarità e andamenti ciclici della domanda.
- Permettere l'acquisto in quantità tali da consentire riduzioni di prezzo.

- Agevolare la distribuzione e i trasporti (per esempio formando lotti economici).
- Costituire una riserva di sicurezza a fronte di eventi imprevedibili.
- Svincolare le fasi produttive. Nell'approvvigionamento si vuole evitare che la produzione sia troppo condizionata dai tempi di consegna e da eventuali ritardi. Nella produzione si vuole fare in modo che le diverse stazioni di lavoro non si blocchino, il che può avvenire sia per attendere materiale dalle stazioni a monte, sia per aspettare che si liberino le stazioni a valle. Inoltre, si cerca di fare in modo che la domanda finale non richieda variazioni troppo frequenti della quantità e del tipo di articoli prodotti, che farebbero lievitare i costi di messa a punto delle attrezzature.

2.2.2 Costi associati alle scorte

Fig. 2.1 Andamento dei costi di gestione delle scorte.

Utilizzare dei magazzini per il deposito dei materiali, cercando di farli funzionare al meglio, comporta una serie di oneri finanziari. Tali oneri si suddividono usualmente in tre categorie, rappresentate anche in fig. 2.1.

Costi del tenere. Sono tutti quei costi che crescono con l'aumentare delle giacenze, potendosi considerare proporzionali alla quantità delle giacenze, al loro valore e al tempo di giacenza. Come indice di costo in un periodo si utilizza spesso il valore medio della giacenza nel periodo. Vediamo le principali voci di costo che compongono i costi del tenere, ordinate indicativamente in ordine decrescente d'importanza.

- Costo opportunità del capitale immobilizzato nelle giacenze. È la potenziale rendita finanziaria cui si rinuncia decidendo di mantenere un certo livello di scorte. Si calcola stimando quanto si sarebbe potuto ottenere investendo in altro modo il capitale impegnato nelle scorte. Come rendita minima si può considerare l'interesse di un deposito bancario, ma gli investimenti aziendali forniscono in genere rendite maggiori, e risulta quindi più realistica (ma non completamente affidabile) una stima basata sul valore del **MARR**.
- Costi sostenuti per assicurarsi contro eventuali rischi d'incendio, furto o altro. Sono anch'essi proporzionali al valore unitario del prodotto.
- Costi operativi di immagazzinamento e movimentazione. Possono essere di moltissimi tipi e dipendono soprattutto dalle caratteristiche fisiche del prodotto: costo dello spazio impegnato, costo di luce e calore nel magazzino, costo del lavoro di movimentazione, ecc.

Il **MARR** (*Rate of Return*) è il saggio minimo di rendimento conveniente, ovvero il rendimento minimo di un investimento considerato accettabile. È spesso impiegato per valutare i costi opportunità, poiché, impiegando risorse in una proposta d'investimento, si rinuncia ad un rendimento pari al MARR, che l'azienda ritiene di poter conseguire facilmente.

- Costi dovuti ad obsolescenza dei prodotti o a cicli di vita limitati. È probabilmente la voce di costo più volubile, potendo essere insignificante per alcuni prodotti e determinante per altri. Gli articoli per cui può assumere grande importanza sono quelli deperibili o soggetti a scadenza, quelli legati alla moda e in generale tutti quei prodotti che perdono rapidamente valore con il passare del tempo. In tutti questi casi è determinante garantire un elevata rotazione delle scorte.

Costi delle ordinazioni. Risultano in genere crescenti al diminuire delle giacenze. Per mantenere basso il livello delle scorte, servono infatti ordini frequenti di quantità ridotte. Alcuni costi, dovuti alle emissioni degli ordini, al trasporto o alla ricezione, sono legati strettamente al numero di ordini, e non dipendono dalla loro entità. Tali costi risultano, a quantità totale da ordinare costante, crescenti rispetto al numero di ordini in cui essa è suddivisa, e perciò crescenti al diminuire della giacenza media.

Costi delle rotture di scorta. Sono i costi o le perdite dovuti al mancato adempimento delle richieste dei clienti quando non si riesce a soddisfarne la domanda. Possono consistere nella perdita della transazione di vendita, ma anche nella perdita del cliente e nella compromissione dell'immagine aziendale. Meno drasticamente, possono coincidere con l'applicazione di sconti a causa di ritardi nelle consegne. Tali costi risultano crescenti al diminuire delle scorte, poiché è più probabile soddisfare picchi di domanda se la scorta di sicurezza è elevata. Sono indubbiamente i costi più difficili da quantificare e da prevedere, e per questa ragione non vengono inclusi nella maggior parte dei modelli analitici di gestione, che prendono in considerazione solo le prime due categorie di costo.

2.2.3 Decisioni da prendere nella gestione scorte

Per gestire in modo funzionale le scorte è anzitutto necessario mantenere un controllo regolare delle giacenze. In genere la verifica costituita dal conteggio fisico del numero di articoli presenti a magazzino è attuata periodicamente, ma può essere conveniente l'utilizzo contemporaneo di un monitoraggio continuo che, tenendo conto delle movimentazioni, permetta di conoscere in ogni momento l'entità delle giacenze.

Altro passo basilare è l'effettuare, nel modo più affidabile possibile, una previsione sui fabbisogni futuri, la quale presenta la massima incertezza nel caso dei prodotti finiti, mentre risulta più agevole nel caso delle materie prime e dei semilavorati. Le quantità necessarie per ogni

componente si possono infatti determinare anche in modo puramente algebrico dalla distinta base, ma è importante ricordare che errori di stima nei fabbisogni finali si ripercuotono su tutta la catena produttiva, e che quindi un certo grado di incertezza si osserva lungo tutto il processo industriale.

I dati precedenti devono essere integrati con il tempo di consegna (*lead time*), cioè il tempo che intercorre tra il momento dell'ordinazione e il momento del ricevimento del materiale ordinato.

Oltre a queste informazioni essenziali, molti sono i fattori che possono influenzare la gestione anche in modo sostanziale, e saranno più avanti analizzati in dettaglio. Supponendo infine di possedere tutte le informazioni utili, le decisioni da prendere nella gestione delle scorte di un articolo sono la scelta del momento in cui ripristinare la scorta e la dimensione del lotto di ripristino. In breve, si tratta di stabilire *quando* e *quanto* ordinare, supposto di possedere comunque dei criteri di base che permettano di stabilire *quando* e *quanto* vendere.

Il processo di gestione continua poi con l'analisi delle prestazioni ottenute. Questa prevede un resoconto preciso dei costi sostenuti, che può comprendere anche una stima delle perdite dovute all'immobilizzazione e alle rotture di stock. In caso contrario, se non si riesce a quantificare in modo affidabile tali perdite, come in genere accade, è utile impiegare degli indici di prestazione, assegnando loro gradi diversi d'importanza in base alle strategie aziendali adottate. La gestione futura può così tenere conto di una serie di informazioni aggiuntive, utilizzando come *feedback* i dati ottenuti dalla valutazione dei risultati passati. Si instaura in questo modo un processo di apprendimento teso al miglioramento continuo.

2.2.4 Misura delle prestazioni

La valutazione delle prestazioni ottenute con la gestione è di importanza fondamentale. Nel processo di misurazione delle prestazioni, prima di procedere al calcolo vero e proprio, è indispensabile stabilire quali sono, per le diverse funzioni aziendali e per l'azienda in generale, gli indici che meglio si adattano a valutare le prestazioni ottenute, in base agli scopi che si volevano conseguire con la gestione. Evidentemente, un'azienda che punta alla massima riduzione dei costi di gestione, deve prevedere metodi di valutazione diversi da una che mira al massimo soddisfacimento dei clienti. Quantomeno, tali aziende as-

segneranno un peso diverso agli indici di prestazione calcolati.

I costi sostenuti, suddivisi in base alle causali, sono il più ovvio indice di prestazione, ma spesso non sono sufficienti per rappresentare fedelmente i risultati del processo di gestione. Molto spesso inoltre si incontrano problemi di assegnazione dei costi, che rendono questo metodo, oltre che complesso, quasi inapplicabile. In aggiunta o in sostituzione ad esso si utilizzano perciò diversi altri indici, che forniscono indicazioni più precise su alcuni aspetti della gestione.

Una misura comunemente usata è l'*indice di rotazione delle scorte*, che si calcola dividendo il valore totale del venduto (relativo ad un determinato periodo, più spesso annuo) per il valore medio delle giacenze nei magazzini. Idealmente, si ottiene un valore che indica quante volte i magazzini sono stati svuotati e riempiti nel periodo d'interesse. L'indice di rotazione è comunemente usato per confrontare le prestazioni di aziende diverse di una stessa industria, o per valutare gli effetti di un cambiamento strategico nelle decisioni relative alle scorte. Un alto valore dell'indice di rotazione può corrispondere ad un elevato tasso di rendita dei capitali investiti nelle scorte, ma non è comunque sufficiente a dimostrare l'utilità di mantenere tali scorte.

Per stabilire se l'utilizzo di magazzini reca effettivamente dei benefici, diversi indici tendono a misurare il livello del servizio che si è reso al destinatario della merce. Ad esempio si può calcolare la percentuale di richieste dei clienti che si è stati in grado di soddisfare immediatamente, o il numero di volte che si è interrotta la produzione a causa di una carenza d'inventario. Indici specifici di questo tipo devono comunque essere scelti molto accuratamente caso per caso, essendo strettamente correlati alle decisioni strategiche di gestione.

Nel complesso, al momento di giudicare la bontà di una politica di gestione, si devono considerare e soppesare attentamente costi e benefici, anche se ciò può rivelarsi proibitivo al momento di quantificare le grandezze in gioco per renderle confrontabili.

2.3 METODI CLASSICI DI GESTIONE

Come si è detto, una volta considerate note tutte le informazioni necessarie, e stabilite le mete da raggiungere, gestire le scorte significa decidere quando fare le ordinazioni e che quantitativi ordinare. Sia la frequenza delle ordinazioni sia l'entità degli ordini possono essere mantenuti

costanti o fatti variare a seconda delle esigenze: si ottengono quattro possibili tipi di modelli di gestione, illustrati nella tab. 2.1 (tratta da Berry, Vollmann, Whybark [14]).

Tab. 2.1 Tipi di modelli decisionali di gestione

		Quantità d'ordine	
		Costante (Q)	Variabile (L)
Frequenza d'ordine	Costante (T)	1. (T,Q)	3. (T,L)
	Variabile (P)	2. (P,Q)	4. (P,L)

Frequenza d'ordine: T = ordinare ad intervalli fissi lunghi T periodi
P = ordinare quando la scorta scende fino al punto di riordino P
Quantità d'ordine: Q = ordinare la quantità fissa Q
L = ordinare la quantità che ripristini la scorta al livello di riordino L

Modello *EOQ*

Le variabili:
Q [pezzi/periodo], consumo complessivo
C_C [lire/ordine], costo di una consegna
C_G [lire/pezzo] costo unitario di giacenza nel periodo
L [pezzi/lotto] dimensioni lotto di consegna

Il problema: si tratta di determinare L (suddividendo perciò il fabbisogno totale in D/L ordini) in modo tale da minimizzare i costi totali, dati dalla somma dei costi delle consegne $C_C \cdot D/L$ con i costi delle giacenze $C_G \cdot L/2$ calcolati sulla giacenza media $L/2$.
Il lotto economico $L_{(EOQ)}$ è quel valore di L che annulla la derivata prima dei costi totali:

$$L_{(EOQ)} = \sqrt{\frac{2 \cdot C_C \cdot Q}{C_G}}$$

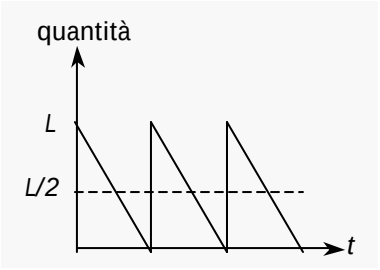


Fig. 2.2 Andamento delle giacenze secondo il modello *EOQ*.

Modelli appartenenti al tipo 1., in cui le variabili si possono determinare una volta per tutte e rimangono costanti nel tempo, raramente si riscontrano tali e quali nei sistemi reali, ma possono essere utili come punti di partenza per la progettazione di sistemi più completi e versatili. Un modello di tipo 1. molto semplice e noto, che serve a determinare le quantità ottimali di T e Q, è il modello del Lotto Economico d'ordinazione, LE, detto anche modello di Wilson o *EOQ* (*Economic Order Quantity*). Le condizioni d'uso richieste da modelli di questo genere li rendono poco utilizzabili, se non con opportune correzioni in fase di applicazione. Le ipotesi restrittive previste sono infatti che i consumi siano noti e costanti nel tempo, che la quantità ordinata venga consegnata dopo un tempo noto e costante, che i costi di riordino e di mantenimento siano noti e costanti, che non ci siano vincoli dimensionali sul lotto di consegna.

Di solito le condizioni d'utilizzo non sono ideali, e le informazioni che servono per il processo decisionale sono incomplete o solo stimate. Per esempio, la domanda non è nota a priori, se non in termini probabilistici. Si deve perciò tenere conto della componente aleatoria del processo, che viene presa in considerazione, seppure in modo indiretto, nei modelli del secondo e terzo tipo. Il tipo 2 (a punto di riordino) prevede un controllo continuo della quantità giacente, che viene confrontata con una prefissata quantità P detta punto di riordino: quando la giacenza raggiunge o scende sotto al valore di P si emette un ordine di entità costante Q. Il tipo 3 (a livello di riordino), non richiede invece un monitoraggio continuo: l'ordine viene emesso ad intervalli fissi di tempo T, mentre la quantità da ordinare si calcola sottraendo la giacenza attuale da un predefinito livello di riordino L.

In entrambi i casi, la variabile da determinare in base alle condizioni del momento è quella che conferisce elasticità al modello di gestione. In genere, modelli del secondo tipo forniscono migliori prestazioni in termini di riduzione delle giacenze medie, ma gli ordini emessi a tempi regolari, caratteristici del terzo tipo, possono risultare molto convenienti se permettono la consegna congiunta di articoli diversi provenienti dallo stesso fornitore o dalla stessa zona geografica.

Per far fronte alla complessità delle situazioni reali si utilizzano frequentemente modelli del quarto tipo, che cercano di far convivere i vantaggi dei modelli 2 e 3. Garantendosi la massima libertà decisionale rispetto alle variabili in gioco, questi modelli risultano senz'altro i più potenti e versatili, ma richiedono anche una maggior complessità di progettazione e una profonda conoscenza del problema che si vuole affrontare. Ciò nonostante, è indubbio che, intendendo sviluppare un sistema di gestione innovativo di interesse generale, ci si debba per forza orientare verso modelli di questo tipo.

A completamento dei modelli visti, si utilizzano procedure che consentono di tenere conto di molte incertezze proprie delle situazioni reali, con l'introduzione e il calcolo di scorte di sicurezza finalizzate a coprire variabilità temporali e quantitative. Nei sistemi più completi, per determinare i parametri di gestione, si impiegano stime di quelle grandezze che sono di importanza basilare nell'economia complessiva del sistema di gestione, ma che risultano difficili da quantificare. Ad esempio, si possono stimare ed inserire nei calcoli la gravità di una rottura di stock, il costo opportunità di un immobilizzo, l'importanza di migliorare di un punto percentuale il livello di servizio. Fondamentale da questo punto di vista è riuscire a considerare dovutamente gli indici di prestazione.

Attualmente, modelli di gestione molto sofisticati si fondano su metodi probabilistici. Le diverse variabili di gestione sono calcolate sulla base di stime più o meno soggettive delle distribuzioni di probabilità dei parametri in gioco.

2.4 IL FUTURO DELLA GESTIONE SCORTE

L'evoluzione dei metodi di governo dei magazzini si inserisce in un processo più ampio di miglioramento globale delle aziende manifatturiere, tendente a promuovere forme di organizzazione orientate ad eccellere nella qualità dei servizi offerti. La propensione delle aziende al perfezionamento continuo ha origine con l'introduzione

La filosofia di gestione **TQM** (*Total Quality Management*) comprende diversi aspetti di gestione, tutti basati sul presupposto che cercare il massimo della qualità risulta alla fine sempre conveniente. Le metodologie ad essa collegate, tra cui si possono citare *Just in Time* e *Lean Production*, stanno influenzando in modo sempre crescente anche le industrie occidentali.

della filosofia della qualità totale (**TQM**), avvenuta in Giappone alcuni decenni fa. Nell'ottica della qualità totale, non è possibile affidare al caso o a soluzioni estemporanee le decisioni riguardanti un settore dell'azienda, quale la gestione delle scorte, che influisce in modo non trascurabile sul bilancio aziendale e sul servizio che l'azienda fornisce ai clienti. La gestione dei magazzini deve quindi essere sempre più integrata con il processo produttivo, con le scelte di marketing e strategiche dell'azienda.

L'analisi del presente lavoro non vuole spingersi ad influenzare decisioni riguardanti aspetti gestionali che non siano strettamente inclusi nell'argomento di base. Tuttavia, si cercherà di valutare come la gestione scorte possa tenere in debito conto ciò che concerne altri aspetti aziendali collegati in qualche modo ad essa, ritenendo ciò molto importante in un ottica di crescita globale a livello aziendale.

2.4.1 Margini di miglioramento

Oramai, per ottenere i massimi risultati con un sistema di gestione scorte, non è più sufficiente l'utilizzo ottimale dei dati numerici in proprio possesso. Questo aspetto è assai importante, ma si considera assodato che i modelli attuali non presentino carenze sostanziali al riguardo. Come risulta dall'analisi fatta, si rivela invece fonte di possibili ulteriori miglioramenti l'inserimento sistematico e metodico nel sistema di gestione delle variabili più soggettive, difficili da quantificare e formalizzare. Questo inserimento è attuato ora prevalentemente da personale specializzato dotato di competenza e capacità di giudizio, che corregge i risultati ottenuti analiticamente in base ad intuizioni personali fondate sull'esperienza e sul buon senso. Raramente si tenta di integrare tali intuizioni direttamente nell'elaborazione, essendo l'operazione senza dubbio ardua. In questo modo esse rimangono vincolate alla disponibilità di lavoratori con esperienza e conoscenza del problema, che in ogni caso non potranno mai riprodurre i loro ragionamenti alla velocità di una elaborazione computerizzata. Se non si riesce perciò ad aumentare l'intelligenza dei sistemi di gestione, fornendo loro ulteriori capacità decisionali simili a quelle di un esperto, difficilmente sarà possibile utilizzare capacità di questo tipo in modo sistematico su grosse moli di dati. In questa direzione si muovono i progettisti di *sistemi esperti* e i promotori dell'*intelligenza artificiale*, che puntano a fornire ai me-

todi di elaborazione automatica alcune competenze peculiari del ragionamento umano.

2.4.2 Perché la *fuzzy logic*

Anche la logica fuzzy, applicata a sistemi decisionali, è orientata verso un integrazione tra uomo e macchina del tipo sopra accennato: l'esperto deve "spiegare" ai sistemi di elaborazione come riesce ad operare certe scelte, in modo che i ragionamenti fatti vengano poi imitati ed applicati ripetutamente su grosse quantità di dati. L'abilità dei sistemi fuzzy di riprodurre su grande scala la capacità di giudizio di un esperto è una delle ragioni che fa ritenere la logica fuzzy una delle vie possibili verso l'integrazione sopra auspicata.

L'esame degli obiettivi della gestione dei magazzini rafforza l'idea che una soluzione ottenuta con la logica fuzzy possa ben adattarsi alla natura del problema. È possibile identificare alcune caratteristiche peculiari di tali obiettivi, ottenute analizzando le particolarità del problema e il diverso contributo che le funzioni aziendali possono fornire nell'affrontarlo.
Secondo l'analisi fatta gli obiettivi perseguibili da un sistema di gestione scorte sono:

- **Contrastanti;** si osservano obiettivi molto variegati, derivanti da esigenze diverse. Differenti funzioni aziendali con differenti compiti e mentalità identificano diversi centri di costo; gli obiettivi che ne conseguono vengono formulati in modo indipendente, risultando spesso contrastanti. Si rende necessario coordinare questi obiettivi, passando da una filosofia ottimizzante al perseguimento di trade-off tra esigenze opposte. Ad esempio, non si può sperare di ottenere contemporaneamente giacenza media nulla e livello di servizio del 100%.
- **Dinamici;** gli obiettivi sono spesso in evoluzione continua, in adattamento rispetto ai cambiamenti strategici, alle modalità di produzione, alle iniziative di marketing. Esempi di ciò possono essere l'aumento delle giacenze dovuto una previsione di forte crescita della domanda, oppure il tentativo di allungare il ciclo di vita di un prodotto riducendo i tempi di attesa nelle consegne ed aumentandone quindi le scorte.
- **Non analitici;** è difficile stabilire misurazioni univoche o creare modelli di comportamento analitici affidabili, non legati a casi molto particolari. Abbiamo visto quanto sia importante quantificare e misurare le

prestazioni, ma anche quanto ciò sia difficile da fare con le metodologie classiche.

Le caratteristiche citate rendono problematico l'approcciarsi alla gestione con i metodi classici, per le ragioni già spiegate. Tali caratteristiche sono invece tipiche dei problemi gestiti con successo da sistemi fuzzy.

Un'altra caratteristica dei sistemi decisionali fuzzy che può risultare utile in problemi di questo tipo è già stata presentata nel primo capitolo, ed è la possibilità di far convivere più obiettivi, senza dovere inoltre decidere se una variabile deve essere considerata un vincolo o un obiettivo.

3

GESTIONE DI SCORTE CON LOGICA FUZZY

3.1 IL PROGETTO DI GESTIONE

L'obiettivo di questo progetto è implementare un sistema software, basato su un'elaborazione di tipo fuzzy, che riesca a riprodurre su grande scala la capacità decisionale di un operatore umano nella gestione di un magazzino scorte.

Il progetto è stato sviluppato in collaborazione con la ditta **SEAP** di Milano. Tra le altre attività, la SEAP progetta e realizza sistemi di gestione per altre aziende, e nello specifico elabora soluzioni per la gestione delle scorte di magazzino. La ricerca di soluzioni innovative, alla base della filosofia aziendale SEAP, ha portato in questo caso ad interessarsi ad un progetto che propone una soluzione tecnologica nuova, come la logica fuzzy, per un problema tipico nella conduzione di un azienda manifatturiera, quale la gestione scorte.

La ditta **SEAP**, costituita nel 1975, si occupa di consulenze e soluzioni di gestione relative a diversi settori aziendali. Si possono citare in particolare le competenze in materia di getione del personale, gestione dei materiali, progettazione di software dedicato e di reti informatiche e certificazioni di qualità.
La missione aziendale è tesa al massimo soddisfacimento delle esigenze dei clienti. La SEAP è orientata verso un investimento continuo nella conoscenza, ai fini di seguire l'evoluzione tecnologica dei vari settori e, quando possibile, anticiparla con soluzioni innovative.

Il sistema da implementare si compone di due parti interagenti, come si può vedere dalla fig.3.1: la registrazione della situazione delle giacenze e il processo decisionale atto a stabilire, in base a tale situazione, la sequenza degli ordini da emettere.

L'ossatura di partenza su cui costruire il sistema è perciò un classico procedimento per l'inventariazione di scorte in un'azienda manifatturiera. Il ruolo decisionale è invece svolto da una elaborazione fuzzy, che sostituisce i modelli tradizionali per il calcolo dei fabbisogni, cercando di imitare le modalità di scelta di un operatore umano e la sua capacità di giudizio.

Nella prima parte di questo capitolo è illustrato il funzionamento di un tipico inventario di un magazzino scorte, il quale deve prendere nota di tutte le operazioni per valutare la loro influenza sulla situazione delle giacenze attuale e futura. Per adattare il procedimento descritto al progetto saranno introdotte alcune modifiche strutturali e metodologiche. In particolare, sarà creato appositamente un nuovo archivio per ogni articolo trattato, che servirà nella

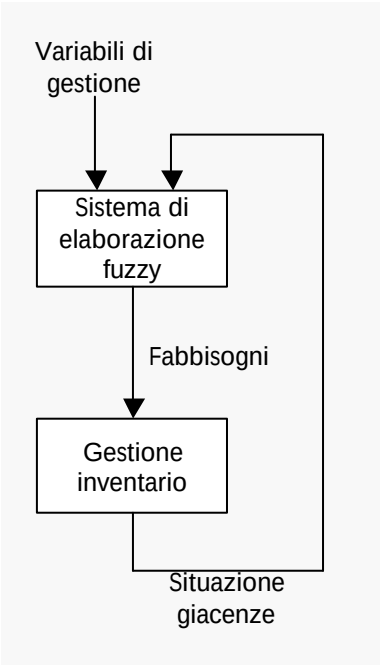


Fig. 3.1 Schema generale del sistema. Sono evidenziate le due sezioni che lo compongono e i flussi d'informazioni.

fase di elaborazione. Le decisioni prese devono poi essere riportate sugli archivi originali.

Nel seguito del capitolo si descrive la fase di analisi del sistema decisionale fuzzy, che costituisce il primo stadio della progettazione. Sarà fornita una descrizione del funzionamento generale, prima di passare allo studio dettagliato delle variabili concernenti la gestione di scorte, tra le quali verranno scelte quelle più importanti da impiegare nel sistema di elaborazione fuzzy.

3.2 LA GESTIONE DELL'INVENTARIO

Al fine di illustrare il meccanismo di registrazione della situazione delle giacenze di un magazzino, si presentano due estratti dei tracciati record di un ideale database per la gestione di scorte. Gli archivi che interessano sono il *progressivo degli articoli*, che registra lo stato delle giacenze degli articoli trattati e il *cronologico dei movimenti*, contenente la successione di tutte le operazioni che modificano il *progressivo*. Viene presentata quindi la tabella delle causali dei movimenti che definiscono le modalità di variazione dei record.

Infine, sarà introdotto e spiegato il nuovo archivio *sequenziale*, che costituisce la soluzione adottata per abbinare l'aggiornamento dell'inventario alla procedura di calcolo fuzzy.

3.2.1 L'archivio *progressivo* degli articoli

Partendo dalla situazione delle giacenze di ogni articolo ad inizio anno, l'**archivio *progressivo*** (di cui a fianco si mostra un possibile tracciato record semplificato) registra tutte le modifiche apportate alle giacenze fino al momento attuale. Nella gestione del magazzino si tengono sotto controllo congiuntamente sia le operazioni fisiche che quelle logiche. La **situazione fisica** si riferisce ai movimenti reali di merce e alle giacenze che effettivamente occupano spazio nel magazzino. La **situazione logica** del magazzino riguarda invece trasferimenti e giacenze previste, potenziali, ed è in base a questa che si possono prevedere le reali disponibilità future, e decidere quindi come dovrà essere governato il magazzino in seguito (non ha senso emettere un ordine senza tenere conto che altri ordini possono già essere in arrivo, o garantire diverse consegne basandosi sulla stessa giacenza di magazzino).

Si esaminano ora più nel dettaglio i vari campi di un generico record dell'archivio in questione. L'*esistenza iniziale* rappresenta lo stato effettivo delle giacenze

Tracciato record dell'archivio ***progressivo* degli articoli**

Codice articolo	
[Campi fisici]	
Esistenza iniziale	*a*
Progressivo carico *PC*	*b*
Progressivo scarico *PS*	*c*
Esistenza attuale *EA*	*d=a+b-c*
[Campi logici]	
Impegni con clienti *IC*	*e*
Disponibilità reale *DR*	*f=d-e*
Ordini fornitori *OF*	*g*
Disponibilità prevista *DP*	*h=f+g*

NOTA: il campo sottolineato è la chiave primaria dell'archivio.

all'inizio del periodo di archiviazione (spesso è l'inizio dell'anno). L'*esistenza attuale* (detta anche *esistenza dinamica*), indica invece la quantità di articoli fisicamente presenti nel magazzino al momento presente. I campi *carico* e *scarico* rappresentano i movimenti cumulati di merci che rispettivamente aumentano e diminuiscono le giacenze effettive nel magazzino nei confronti di quelle iniziali.

Gli *impegni con clienti* rappresentano scarichi futuri, e sono importanti soprattutto per conoscere qual è la *disponibilità reale* di merce, data dalle giacenze registrate nell'*esistenza attuale* diminuite delle consegne previste (gli *impegni con clienti*). Gli *ordini ai fornitori* registrano invece i carichi previsti, che aumentano la *disponibilità prevista,* ovvero il valore che dovrebbe tradursi in *esistenza attuale* una volta passato il periodo di attesa delle consegne. I campi logici sono infatti calcolati anche per i periodi futuri, per poter prendere in anticipo i provvedimenti necessari ad evitare eventuali situazioni indesiderate. Ad esempio, si vorrà evitare che si verifichino carenze nei magazzini e rotture di scorta, rappresentate da valori negativi rispettivamente nei campi delle *disponibilità previste* e *reali.*

3.2.2 L'archivio cronologico dei movimenti

L'**archivio dei movimenti** contiene lo svolgimento cronologico di tutte le operazioni, siano esse fisiche o logiche, che modificano il *progressivo* degli articoli. L'effetto di ogni movimento sui campi dell'archivio precedente dipende dall'intervallo di tempo che intercorre tra la data di consegna e quella attuale, dalla quantità interessata e dalla causale del movimento, che indica il tipo di operazione eseguita. Qui a fianco si illustra un esempio semplificato di tracciato record.

Tracciato record dell'archivio dei **movimenti**

Codice articolo
Codice cliente/fornitore
Data
Quantità
Prezzo/costo
Data consegna
Codice Causale

I valori contenuti nei campi dell'archivio *progressivo* sono collegati secondo le regole indicate dalla **tabella delle causali dei movimenti** (tab.3.1). In essa sono elencati i diversi tipi di operazioni possibili, indicando se i campi del *progressivo* vengono modificati e in che modo. Le caselle contrassegnate con '+' indicano che il campo corrispondente viene aumentato della quantità contenuta nel campo *quantità* del file movimentazioni, il segno '-' rappresenta invece una sottrazione, ricordando che i campi logici possono contenere anche valori negativi. I campi *disponibilità reale* e *prevista* non compaiono nella tabella, poiché si calcolano direttamente dal valore degli altri

campi con le formule riportate nella rappresentazione dell'archivio *progressivo.*

Tab. 3.1 Causali dei movimenti ed effetti sul file *Progressivo.*

Cod.	Causale dei movimenti	*PC*	*PS*	*EA*	*IC*	*OF*
001	Carico ordine cliente				+	
002	Storno ordine cliente				-	
003	Carico ordine fornitore					+
004	Storno ordine fornitore					-
011	Carico a magazzino da ordine fornitore	+		+		-
012	Scarico da magazzino per ordine cliente		+	-	-	
013	Vendita al banco		+	-		
014	Acquisto immediato (senza ordine)	+		+		
015	Perdita per scadenza/deperibilità		+	-		

Risulta evidente che il meccanismo di manutenzione dell'inventario illustrato non necessita di una elaborazione fuzzy, essendo puramente algebrico e non presentando margini d'incertezza.

Al fine di adattare la gestione dell'inventario al processo di elaborazione fuzzy è tuttavia necessario introdurre delle aggiunte al metodo di gestione descritto. Si è ritenuto opportuno non modificare direttamente contenuto e funzionamento degli archivi fondamentali del sistema, e si è quindi introdotto un nuovo archivio che funga da interfaccia tra la sezione di calcolo e quella puramente gestionale. Quest'ultima rimane inalterata e non farà quindi parte delle prossime fasi di analisi.

3.2.3 L'archivio *sequenziale* delle giacenze

In un database viene detta **istanza** di un archivio l'insieme dei valori dei campi dell'archivio in un determinato momento.

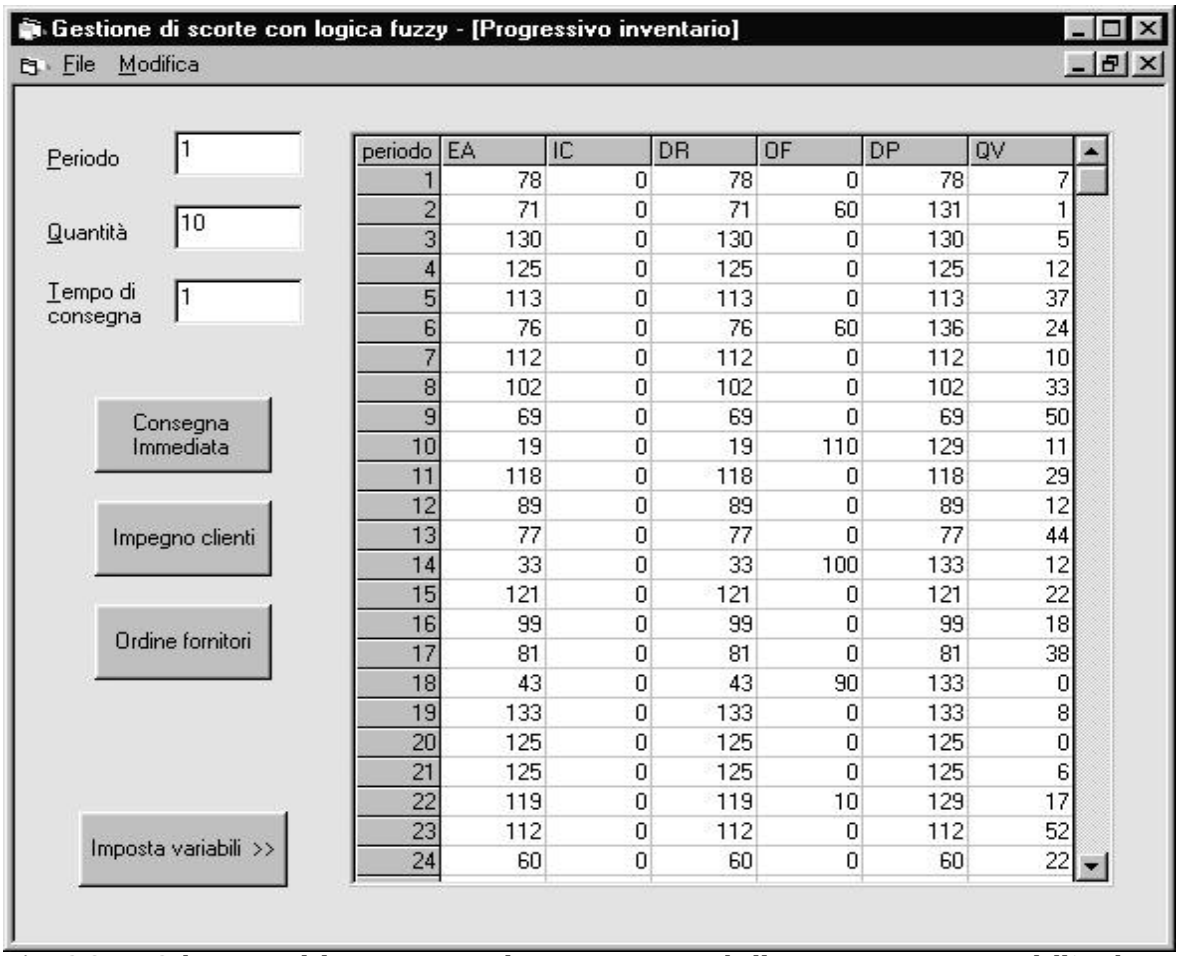
Nel nostro caso, l'archivio *progressivo* viene scomposto in base ai diversi articoli che considera. Per ogni articolo si crea un archivio contenente la sequenza delle **istanze** del *progressivo*, riferite a quell'articolo, prese ad intervalli di tempo regolari.

Il nuovo archivio introdotto, denominato ***sequenziale* delle giacenze** (v. fig. 3.2), contiene alcuni dei campi dell'archivio progressivo, ossia quelli che descrivono in ogni momento la situazione attuale: *esistenza attuale* (*EA*), *impegni con clienti* (*IC*), *disponibilità reale* (*DR*), *ordini a fornitori* (*OF*), *disponibilità prevista* (*DP*). A questi è stato aggiunto un campo che, in riferimento all'intervallo di tempo considerato, contiene l'entità delle vendite effettuate senza preavviso, ossia senza che ci sia la possibilità di contemplarle negli *impegni con clienti* dei periodi precedenti. Il nuovo campo si chiama *quantità venduta* (*QV*).

La serie di record rappresenta ora una sequenza temporale delle diverse situazioni delle giacenze di quell'articolo in istanti successivi. Sia il *periodo* l'unità di tempo impiegata come intervallo di campionamento dei dati. Lo stesso termine si utilizza per distinguere i record, identificandoli con l'istante di campionamento cui si riferiscono. Il record che descrive la situazione attuale è il numero uno.

I record che interessano la gestione partono da quello attuale, per arrivare fino all'ultimo che risulta in qualche modo influenzato da ordini in arrivo o in partenza.

I campi sono comunque aggiornati con la stessa logica che serve a governare il *progressivo*: gli impegni presi con i clienti diminuiscono le disponibilità reali, quelle previste e l'esistenza attuale dal momento in cui la consegna stessa avviene. Gli ordini ai fornitori agiscono in modo analogo, aumentando il valore di *DP* dal momento dell'emissione e quelli di *EA* e *DR* da quel momento in poi. Nella fig. 3.2 è mostrato un esempio di come l'ordine ai fornitori modifica l'archivio.



periodo	EA	IC	DR	OF	DP	QV
1	78	0	78	0	78	7
2	71	0	71	60	131	1
3	130	0	130	0	130	5
4	125	0	125	0	125	12
5	113	0	113	0	113	37
6	76	0	76	60	136	24
7	112	0	112	0	112	10
8	102	0	102	0	102	33
9	69	0	69	0	69	50
10	19	0	19	110	129	11
11	118	0	118	0	118	29
12	89	0	89	0	89	12
13	77	0	77	0	77	44
14	33	0	33	100	133	12
15	121	0	121	0	121	22
16	99	0	99	0	99	18
17	81	0	81	0	81	38
18	43	0	43	90	133	0
19	133	0	133	0	133	8
20	125	0	125	0	125	0
21	125	0	125	0	125	6
22	119	0	119	10	129	17
23	112	0	112	0	112	52
24	60	0	60	0	60	22

Fig. 3.2 Schermata del programma di gestione. La tabella contiene i campi dell'archivio *sequenziale* di uno degli articoli trattati.

Le quantità vendute all'istante possono essere considerate un caso particolare di impegni con clienti, aventi periodo di consegna nullo, e riducono quindi immediatamente anche il valore di *EA*.

Il modo in cui l'archivio *sequenziale* è stato costruito serve a poter calcolare frequentemente i fabbisogni, e a controllare l'interdipendenza tra le operazioni che si svolgono in periodi di tempo vicini.

D'ora in poi, nella descrizione del funzionamento del sistema, sarà preso in considerazione un solo articolo alla volta, con il corrispondente archivio *sequenziale*, che rappresenta in pratica la sezione del sistema incaricata di gestire l'inventario delle giacenze.

Il prossimo passo da compiere nella progettazione rappresenta il cuore del lavoro, e riguarda l'implementazione della procedura decisionale fuzzy, che richiede un'apposita fase di analisi.

3.3 IL SISTEMA DECISIONALE FUZZY

Osservando il funzionamento dell'inventario delle scorte, si verifica l'ipotesi fatta nel secondo capitolo riguardo alle decisioni da prendere nella gestione di un magazzino. Nota la situazione delle giacenze di un articolo nel corso del tempo, tali decisioni sono sostanzialmente di due tipi: quando e quanto comprare, quando e quanto vendere.

Come ipotesi di partenza, si può assumere che le decisioni di vendita seguano fondamentalmente lo stato delle disponibilità, e che quindi la decisione di accettare o meno una richiesta d'acquisto si basi esclusivamente sulla possibilità di soddisfarla. Emerge in modo evidente che la criticità di tutto il processo sta nel determinare in anticipo lo stato delle giacenze più opportuno per soddisfare nel migliore dei modi le richieste.

Il sistema che si vuole sviluppare dovrà perciò essere in grado di decidere, sulla base della situazione delle giacenze e di altre variabili di gestione che verranno introdotte, se emettere delle ordinazioni, e di che entità. In sostanza, esso dovrà stabilire il valore da assegnare al campo *ordini fornitori* per ogni record dell'archivio *sequenziale*. La procedura di gestione dell'inventario apporterà poi le conseguenti modifiche al resto dell'archivio.

La filosofia risolutiva impiegata prevede controlli periodici dello stato di tutte le variabili, che fungeranno da ingressi del sistema di elaborazione. Il blocco di calcolo è incaricato di determinare ogni volta, in base al valore degli ingressi, le due uscite finali del sistema, ovvero le variabili

quantità necessaria ed ***emissione ordine***. Con la prima si calcola l'entità dell'ordine necessario, con la seconda si decide se è effettivamente opportuno emettere tale ordine (lo schema relativo è mostrato in fig. 3.3).

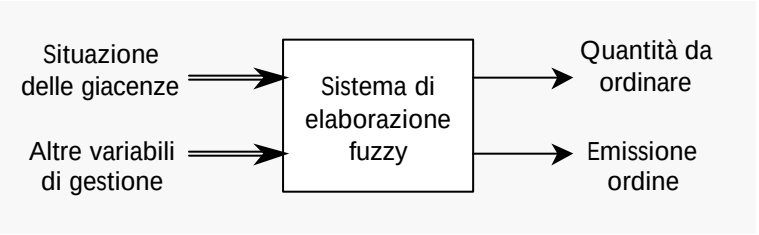


Fig. 3.3 Schema di base del sistema della sezione di calcolo

Volendo classificare il metodo proposto in base ai criteri della tabella 2.1, che distingue i sistemi di gestione a seconda della loro flessibilità nei confronti della quantità e della frequenza di riordino, esso si inserisce fra i metodi del quarto tipo, caratterizzati dalla massima libertà decisionale nei confronti di entrambe le variabili.

Se la decisione riguardante le variabili d'uscita del sistema risulta pertanto quasi ovvia, non lo è altrettanto la scelta delle variabili d'ingresso, ossia di tutti quei fattori che in qualche modo possono influenzare il processo decisionale.

3.4 VARIABILI DELLA GESTIONE SCORTE

Per costruire il sistema di elaborazione fuzzy, è necessario prendere in considerazione, nel modo più esaustivo possibile, le variabili che in qualche modo possono influenzare la gestione di scorte.

Le variabili influenzanti si dovranno dapprima raggruppare e quindi sfoltire, per ottenere un sistema di gestione di base che tenga conto delle determinanti principali del problema. In seguito, si possono aggiungere gradualmente al sistema già formato nuove variabili, o variabili in precedenza scartate, per far in modo che esso si adatti a condizioni di lavoro particolari e caratteristiche. L'intero processo è illustrato nella fig. 3.4.

È importante ricordare che, nella progettazione di un sistema fuzzy, vincoli ed obiettivi non assumono ruoli diversi. In quest'ottica ogni variabile ha, per così dire, pari dignità delle altre, salvo l'attribuzione di pesi d'influenza diversi al momento di progettare le regole d'inferenza fuzzy. È per questa ragione che, nell'analisi seguente, non si porranno distinzioni di ruolo tra le variabili.

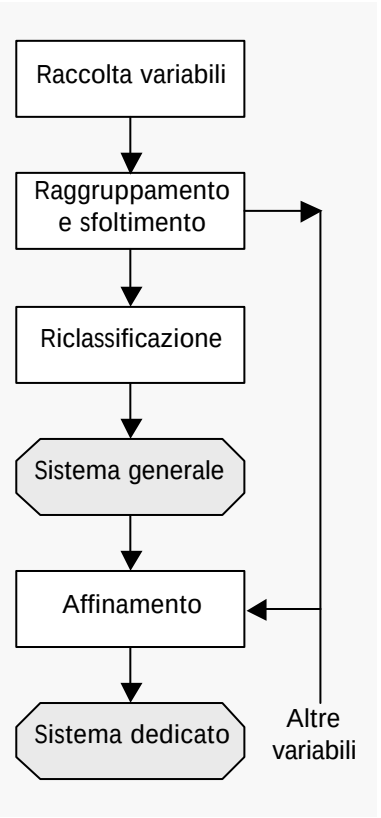


Fig. 3.4 Procedimento logico di selezione ed impiego delle variabili d'interesse.

3.4.1 Le scorte nell'azienda manifatturiera

Prima di analizzare nel dettaglio le variabili d'interesse di un sistema di gestione di scorte, si presenta in fig. 3.5 uno schema generale semplificato di come può strutturarsi un'azienda manifatturiera. In esso sono evidenziate la collocazione logica dei magazzini e delle fasi produttive a cui fanno capo.

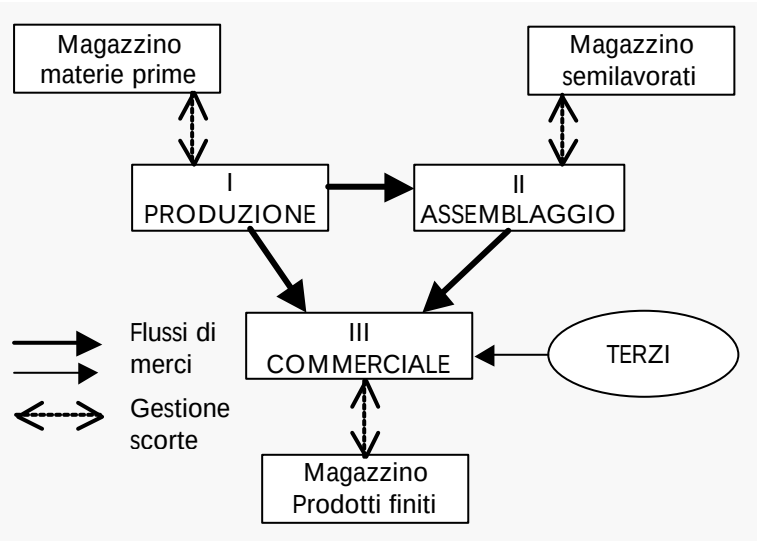


Fig. 3.5 Schema generale di azienda manifatturiera. Sono evidenziate le funzioni principali e i legami con i magazzini di scorte.

Considerando la struttura di un'azienda manifatturiera tipo, tre funzioni sono principalmente interessate al flusso fisico di materiali. Tali funzioni, corrispondenti ad altrettante fasi del processo produttivo/distributivo, sono la produzione, l'assemblaggio e la funzione commerciale. In ogni fase si gestiscono uno o più magazzini di scorte, e le diverse funzioni hanno la responsabilità dei magazzini che competono loro. Per quanto riguarda la gestione dei flussi di merci in entrata e uscita, ogni funzione considera fornitore la fase immediatamente a monte e cliente la fase a valle.

Il fine che ci si pone con il presente lavoro è quello di progettare un sistema di gestione di carattere generale, che possa essere impiegato, grazie all'opportuna modifica di alcune variabili, nelle diverse fasi produttive.

3.4.2 Raccolta delle variabili

Come si è visto nel secondo capitolo, molti sono i fattori che possono influire sui meccanismi di gestione delle scorte. Riferendosi allo schema di fig. 3.5 e ai diversi tipi di magazzini di un'azienda manifatturiera, si verificherà nell'analisi seguente che le variabili influenzanti la ge-

stione non si differenziano in modo sostanziale tra un magazzino e l'altro. Ciò che contraddistingue i processi di gestione relativi alle varie funzioni sono piuttosto i valori d'importanza attribuiti ad ogni variabile, che possono variare notevolmente in base alle caratteristiche della funzione considerata.

Nella seguente tabella 3.2 si presenta una lista di variabili influenzanti la gestione, che potrebbero costituire ingressi per il sistema fuzzy di elaborazione. Ogni variabile è indicata con un numero progressivo ed è inserita in una categoria indicata con una lettera minuscola. Per ogni funzione si sono indicati i fattori caratterizzanti che possono determinare il significato e il valore della variabile. Per alcune variabili è riportata una gamma esemplificativa di valori che possono assumere a seconda dell'ambito aziendale cui si riferiscono.

Tab. 3.2 Variabili che influiscono sulla gestione di scorte

Variabili \ Funzioni	PRODUZIONE (mag. materie prime)	ASSEMBLAGGIO (mag. semilavorati)	COMMERCIALE (mag. prodotti finiti)
Domanda	NOTA: la domanda finale e l'incertezza relativa si ripercuotono sulle fasi precedenti; l'incertezza risulta massima all'inizio del processo		
1. Andamento	Stima del possibile andamento della domanda; alcuni casi fondamentali di domanda finale sono: domanda regolare (beni di largo consumo – es. alimentari) domanda ciclica (beni stagionali – es. ombrelli) vita breve (beni di rapida obsolescenza – es. high tech) vita brevissima (beni soggetti a moda)		
2. Grado di incertezza	Dipende dalla conoscenza dei clienti, del mercato e del prodotto; distinguendo tre soli livelli il grado d'incertezza può essere: minimo (es. bene primario con poca concorrenza) medio (es. mercato in evoluzione) massimo (es. bene di nuova introduzione)		
Costi di gestione	NOTA: In alcuni casi i lotti economici possono diventare lotti minimi indivisibili, sia per acquisto, sia per produzione		
3. Costi accessori d'acquisto	Lotti economici di consegna, trasporto; riduzione prezzi per quantità crescenti	Lotti convenienti di trasferimento interno, da considerare assieme ai lotti del punto 5.	
4. Costi accessori di vendita	Lotti convenienti di trasferimento interno, da considerare assieme ai lotti del punto 5.		Lotti economici di consegna, trasporto
5. Cicli di produzione	Si distinguono tre categorie fondamentali; la seconda introduce la variabile "lotto economico": produzione a flusso continuo (non esistono lotti) produzione a lotti (sono importanti costi e tempi di set-up) produzione singola (è importante la flessibilità a tutti i livelli)		
	Lotti economici di produzione (set-up delle attrezzature, minimizzazione tempi di attesa e inattività)	Lotti economici di assemblaggio (set-up delle attrezzature, minimizzazione tempi di attesa e inattività)	

6. Costo di mantenimento fisico scorte	Proporzionale alla quantità singola o a lotti di materiale (p. es. al numero di pallet, indipendentemente da quanto sono carichi); dipende dallo spazio occupato e dall'uso alternativo che se ne potrebbe fare		
7. Rimanenze	perdite totali immediate (prodotti deperibili o non più vendibili) perdite parziali (prodotti vendibili in quantità o a prezzi ridotti) perdite solo finanziarie (prodotti che non perdono valore)		
	Spreco di materiale		Spreco di materiale; vendite a prezzi ridotti
Costi opportunità			
8. Mancate vendite per rottura di stock	Costi da attribuire alla fase in cui si è verificata la rottura di stock; carenza più grave se si è mancato un impegno di consegna;		
9. Costi finanziari di immobilizzazione	Proporzionali al valore della merce giacente; stimabili in base alla rendita che si potrebbe ottenere investendo in altro modo		
Variabili temporali			
10. Tempi di arrivo (causati dalla fase precedente)	Tempi di consegna	Tempi di produzione	Tempi di assemblaggio
11. Tempi di partenza (causati dalla fase stessa)	Tempi di produzione	Tempi di assemblaggio	Tempi di consegna
12. Tempi massimi di permanenza	Alcune merci, per deperibilità o obsolescenza, presentano tempi massimi ammissibili di permanenza o rapide perdite di valore		
13. Tempi di controllo	Metodologia di controllo delle giacenze; di solito il controllo può avvenire in modo continuo o ad intervalli di tempo più o meno regolari		
Movimentazione logica			
14. Impegni con clienti (consegne previste)	La quantità di scorte di un articolo effettivamente presente nel magazzino è detta *esistenza attuale*. Sottraendo da essa le quantità in partenza (impegni con clienti) si ottiene la *disponibilità reale*; la *disponibilità prevista* si ottiene sommando a quella reale le quantità in arrivo (ordini a fornitori); i tempi descritti in 10. e 11. informano su quando le movimentazioni logiche (impegni) potranno tradursi in movimentazioni fisiche (consegne)		
15. Ordini da fornitori (arrivi previsti)			
Interdipendenze tra merci diverse	Possono essere di moltissimi tipi, e vanno considerate per ogni caso particolare; ne vediamo solo alcune, distinguendole in due classi		
16. Interdipendenze di trasporto e giacenza	Convenienza di ordini congiunti (p. es. dallo stesso fornitore);	Convenienza di giacenze non contemporanee (p. es. ridotto spazio disponibile)	Convenienza di consegne congiunte (p. es. verso la stessa zona)
17. Interdipendenze di processo	Necessità di avere disponibilità contemporanee per merci da produrre assieme	Necessità di disponibilità contemporanee per assemblaggio	Disponibilità non contemporanee (p. es. merci la cui produzione è alternativa)

3.4.3 Analisi dettagliata delle variabili

Come secondo passo dell'analisi, considerando nel dettaglio le diverse variabili considerate, si può tentare di snellire lo schema aggregandone alcune e facendo delle assunzioni vincolanti per altre.

Si possono porre anzitutto due vincoli che semplificano lo schema, riguardanti lo spazio in magazzino e la cadenza dei controlli delle giacenze. La validità delle assunzioni fatte è riscontrabile facilmente nella maggior parte delle aziende manifatturiere attuali.

Per quanto attiene lo spazio assegnato alle scorte e il suo utilizzo (variabile 6) si assume che esso sia fissato in anticipo, e rimanga vincolato a questo utilizzo. Non si prenderanno quindi in considerazione, a meno di grandi

cambiamenti da affrontare come casi speciali, problemi per mancanza di spazio, né costi imputabili allo spazio occupato.

La cadenza dei controlli di giacenza (variabile 13) si può ritenere continua. Infatti, nei sistemi moderni la gestione delle giacenze prevede un monitoraggio ininterrotto, secondo cui movimenti ed ordini sono registrati in tempo reale, e i valori attuali aggiornati automaticamente.

Riferendosi alla variabile 14 (impegni di consegna) è importante considerare la possibilità che ci siano dei ***backorder***. Di questo tipo di ordini si dovrà tenere conto al momento di calcolare la disponibilità reale e quindi le quantità necessarie da ordinare.

Un ***backorder*** si verifica quando un impegno di consegna non è stato soddisfatto nei tempi prestabiliti, ma può ancora essere esaudito entro un certo limite di tempo.

La variabile 5 sui cicli di produzione può riferirsi ad almeno tre categorie diverse di sistemi produttivi: produzione a flusso continuo, a lotti e singola. Nel primo caso non sono previste variazioni nella configurazione del sistema produttivo, e non è quindi introdotta complessità nella gestione. Sistemi del terzo tipo, per i quali ogni singolo articolo prodotto è unico e irripetibile, prevedono una grande flessibilità a tutti i livelli del processo, dalla progettazione al collaudo. I sistemi di gestione di tali processi di produzione sono generalmente realizzati secondo le esigenze specifiche di ogni caso, e risultano quindi difficilmente classificabili. Sono i sistemi di produzione del secondo tipo, a lotti, che offrono il grado di applicabilità maggiore rispetto al tipo di gestione prevista, e che richiedono quindi un'analisi più accurata. Da notare che, nel caso di prodotti finiti, il materiale prodotto con sistemi a flusso può effettivamente essere gestito con la filosofia dei sistemi a lotti, utilizzando unità di misura continue (ad esempio m^2 o kg) al posto delle quantità fisiche degli articoli.

Nei **sistemi di produzione a lotti**, diversi articoli si alternano nella stessa linea o nello stesso centro di produzione o assemblaggio. Nei casi in cui il passaggio da un articolo all'altro nella lavorazione non prevede cambiamenti nella configurazione del sistema produttivo, si può decidere di lavorare indifferentemente su qualsiasi articolo, assimilando così il sistema ad una produzione a flusso. Quando invece per cambiare l'articolo su cui lavorare sono previste delle modifiche al sistema produttivo, ed è il caso più frequente, possono essere necessarie interruzioni del processo, che comportano un allungamento dei tempi e un aumento dei costi. Tali periodi di inattività, detti **tempi di *attrezzaggio*** o di *set-up*, vanno quindi minimizzati, tenendo conto che la produzione deve comunque rispettare il mix di prodotti richiesto. È in quest'ottica

Nelle industrie automobilistiche, i **tempi di attrezzaggio** delle presse per lo stampaggio delle carrozzerie erano uno dei principali fattori che limitavano la possibilità di porre in lavorazione frequentemente modelli diversi di veicoli. Attualmente, con tempi di attrezzaggio di pochi minuti, si producono modelli quasi personalizzati.

che si definiscono i *lotti economici* di produzione e di assemblaggio, ossia le quantità ottimali di articoli uguali da produrre consecutivamente, calcolati trovando il compromesso migliore tra le esigenze di disponibilità dei prodotti e le esigenze di ottimizzazione della produzione. Da ciò risulta che i lotti convenienti variano con la domanda dei beni e con le caratteristiche delle forniture. Le decisioni sui lotti di produzione vengono in definitiva prese congiuntamente alle decisioni di ordine e di consegna, e la variabile 5 si può quindi inglobare nella gestione delle variabili 3 e 4 relative ai costi accessori d'acquisto e vendita, che a sua volta va fatta considerando l'interdipendenza tra tali costi.

Anche le variabili dalla 10 in poi (esclusa la 13 già analizzata) influenzano il processo di gestione degli ordini e delle consegne ora citato, e si possono quindi far rientrare in questa categoria.

Per quanto riguarda le variabili 16 e 17, la grande varietà di situazioni che si possono riscontrare non permette una classificazione agevole. Si può comunque stabilire che si tenga conto delle modalità di produzione e assemblaggio (17) con le distinte base, inserendo perciò anche questo aspetto tra le variabili inerenti la gestione. Lo stesso si può fare per la variabile (16) *evasioni totali*, che rappresenta un vincolo di consegna, indicante la necessità di una consegna congiunta di prodotti diversi. Nel caso di una evasione totale la giacenza di un articolo è vincolata alla disponibilità di altri articoli, e si cercherà perciò di ottenere disponibilità il più possibile ravvicinate.

Alla luce di quanto detto, si può presentare una nuova classificazione più sintetica e organica, che permetta di mettere in evidenza i fattori di maggiore influenza.

3.5 RICLASSIFICAZIONE

Il nuovo approccio prevede la suddivisione delle variabili da inserire nel sistema fuzzy in tre gruppi. Il primo è inerente la *gestione di ordini e consegne*. In questa categoria vengono incluse anche variabili molto diverse tra loro, ma che rispondono ad un'unica logica di gestione e che devono perciò essere prese in considerazione in modo congiunto. Ci sono poi le variabili della domanda, che possono essere classificate come *previsionali*, visto che non comportano movimenti attuali, né fisici né logici, ma che possono influenzare notevolmente la gestione futura. La terza classe di variabili contiene quei parametri che riguardano più da vicino gli obiettivi da perseguire, e che tentano di misurare le *prestazioni* del sistema, determi-

nando la filosofia di gestione. In questa classe sono inserite le variabili 7, 8 e 9, nel tentativo di minimizzare l'incidenza dei costi loro associati sui costi globali di gestione.

Di seguito si analizzano più in dettaglio i tre gruppi di variabili, ricordando che le variabili scelte come ingressi del sistema verranno solo definite e spiegate brevemente. Il loro significato più completo e il modo di impiegarle costituiranno l'argomento del prossimo capitolo.

Alla fine del paragrafo sono illustrate inoltre le variabili utilizzate nella parte algebrica del sistema di gestione, che non rientrano nella procedura di calcolo fuzzy e a cui sono assegnati quindi solo dei valori numerici .

I tre **gruppi di variabili** dopo la riclassificazione:
- variabili di gestione delle movimentazioni (ordini e consegne)
- variabili previsionali
- variabili legate alle prestazioni

3.5.1 Variabili concernenti la gestione di ordini e consegne

All'interno di questo gruppo si possono distinguere due ulteriori categorie. Si osserva infatti un insieme di variabili, definibili **deterministiche** che risultano note quasi con certezza. Tali variabili, collegate tra loro da operazioni algebriche, sono quelle che provengono direttamente dalla procedura di aggiornamento delle giacenze, e concernono i movimenti, fisici e logici, riguardanti il magazzino delle scorte. L'incertezza ad esse collegata è minima, e relativa soprattutto alla tempistica delle transazioni. Un secondo gruppo di variabili, dette **aleatorie**, riguarda elementi più incerti e caratteristici della particolare situazione che si vuole affrontare. Alcune di queste variabili possiedono una valenza generale ed entrano nel modello base, altre, più specifiche, possono essere aggiunte in momenti successivi in base alle caratteristiche peculiari della gestione che si intende implementare.

Variabili a carattere deterministico

Sono le variabili che nella fig. 3.1 vengono indicate con la dicitura *situazione delle giacenze*. Tra le variabili considerate dalla gestione dell'inventario, si impiegano come ingressi del sistema fuzzy la **disponibilità reale** e gli **ordini fornitori**. La prima è la più rappresentativa della situazione effettiva delle giacenze, perché sulla base di essa si decide se sono necessarie ulteriori ordinazioni. È però necessario tenere conto che consegne in arrivo potrebbero modificare nettamente la situazione delle disponibilità future. A questo proposito, gli ordini fornitori permettono di introdurre una correzione ai fabbisogni calcolati in base alle disponibilità reali.

Nel caso in cui, decidendo sull'opportunità di emettere un ordine, non si tenga conto di eventuali ordini già aperti, si può verificare una situazione di ordini contemporanei. Tali ordini possono sembrare tutti necessari al momento dell'emissione, ma si traducono poi in una crescita incontrollata delle giacenze all'arrivo delle merci.

Variabili a carattere aleatorio

Tra le variabili di gestione del secondo tipo, quelle più incerte e soggettive, si possono includere già in una prima analisi la variabilità nei tempi di consegna e l'onere associato ad ogni consegna.

Tempi di consegna non noti o non rispettati possono causare un notevole grado d'incertezza. Perciò, il grado di **variabilità nei tempi di consegna** è introdotto tra le variabili di base del sistema.

Il secondo fattore considerato è l'onere della consegna. In molti casi le consegne si distinguono in *normali*, emesse ad intervalli prefissati (per es. una volta alla settimana, o una volta al mese, sempre lo stesso giorno), ed *extra*, poste cioè nel lasso di tempo che intercorre tra una consegna regolare e la successiva. Generalmente le consegne *extra* risultano in qualche modo più onerose di quelle *normali*, potendo comportare prezzi maggiorati nei trasporti e nel costo della merce. Talvolta possono implicare più lunghi tempi di consegna, e in certi casi risultano del tutto non effettuabili. Tutto ciò, assieme all'opportunità di emettere ordini congiunti, è contemplato nella quarta e ultima variabile di questo gruppo inserita nel sistema, il **genere dell'ordine**.

Variabili di gestione deterministiche

- Disponibilità reale
- Ordini fornitori

Variabili di gestione aleatorie

- Variabilità tempi di consegna
- Genere dell'ordine

3.5.2 Variabili previsionali

Affinché la gestione risulti efficiente, è importante stimare la quantità che si prevede venga richiesta nell'intervallo di tempo tra una consegna in arrivo dai fornitori e la successiva. Al netto di qualsiasi fonte d'incertezza, ogni ordine dovrebbe coincidere con tale quantità. In realtà, non ci si può attendere né di ridurre esattamente a zero la disponibilità reale al momento del riordino, né che venga richiesta proprio la quantità pronosticata.

La prima fonte di incertezza è proprio l'aleatorietà della previsione, che induce a tutelarsi dal rischio di sottostimare le richieste. La **domanda attesa**, o prevista, è una stima dell'effettiva richiesta che si potrà verificare.

La stima non presenterà lo stesso livello di difficoltà in tutti i casi, ed è importante saper giudicare il grado di incertezza ad essa collegato. Tale incertezza viene quantificata nella seconda variabile di questo gruppo, la **variabilità della domanda**, che valuta appunto l'entità degli scostamenti che si prevede possano verificarsi tra la domanda reale e quella attesa.

Variabili previsionali

- Domanda attesa
- Variabilità della domanda

3.5.3 Variabili di prestazione

Una gestione ottimale dovrebbe fornire il livello di servizio desiderato garantendo il minimo valore possibile per i costi di gestione. Si tratta allora di stabilire quale sia il compromesso migliore tra l'esigenza di soddisfare i clienti (siano essi esterni o interni) e la necessità di minimizzare le risorse impiegate.

Anzitutto, ponendosi nell'ottica di assegnare, per quanto possibile, un valore quantitativo alle prestazioni di un sistema di gestione scorte, è importante attribuire un significato preciso e caratterizzante alle diciture generali *rottura di stock* e *livello di servizio*.

In Anderson [13] si può approfondire lo studio sui diversi modi di valutare una prestazione di gestione.

A seconda delle esigenze dei clienti, delle caratteristiche della merce e del sistema produttivo, si possono distinguere almeno tre significati diversi. Tali significati portano a modalità di misurazione, obiettivi e filosofie di gestione diversi.

Tab. 3.3 Significati attribuiti alle rotture di stock e misurazione del livello di servizio

Tipi di rotture di stock	Casi di riferimento	Misura livello di servizio
Costo indipendente dalla quantità mancante e dalla durata della mancanza	La mancanza di un materiale interrompe un processo costoso da riavviare	Numero di rotture di stock in un arco di tempo prestabilito
Costo dipendente dalla quantità, indipendente dalla durata	La mancata consegna nei tempi prestabiliti implica la perdita della vendita (cui possono aggiungersi la perdita del cliente e la rimanenza di merce invenduta)	Percentuale di domanda soddisfatta (in quantità o in valore)
Costo dipendente dalla quantità e dalla durata della mancanza	Le vendite perse sono proporzionali al tempo di ritardo nella consegna	Tempi medi di ritardo nelle consegne

Scelto il significato che meglio rappresenta la propria realtà, è importante definire, in base alla strategia di gestione assunta, quanto sia importante evitare rotture di stock. Il giudizio dato è rappresentato nella variabile **livello di servizio desiderato**.

I costi del mantenimento delle scorte sono contemplati invece nella variabile **onerosità di giacenze e rimanenze**. Le due variabili 7 e 9 sono infatti considerate congiuntamente, pur avendo un significato sostanzialmente differente. Si è optato per questa scelta osservando che l'entità del loro contributo deriva dalla stessa situazione di ordini elevati e poco frequenti. Tenendo conto che difficilmente si può modificare la tempistica delle ordinazioni regolari, i due aspetti sono legati in modo molto simile alla strategia

Variabili di prestazione

- Livello di servizio desiderato
- Onerosità di giacenze/rimanenze

di gestione. Indicativamente, la variabile presentata aumenta il suo peso sia in condizioni di costi elevati per le giacenze, sia nei casi in cui è importante evitare rimanenze.

3.5.4 Variabili di gestione utilizzate in modo algebrico

Per mettere in atto il processo di gestione si devono introdurre alcuni parametri aggiuntivi. Tali parametri sono di tipo algebrico, ed assumono quindi valori numerici precisi. Essi non sono perciò utilizzati come ingressi del processo di elaborazione fuzzy, ma risultano necessari ai fini del corretto funzionamento del sistema nel suo complesso.

Il primo parametro di cui tenere conto riguarda i tempi di consegna degli ordini, che sono misurati in *periodi* ed espressi nella variabile ***lead time***. Tali tempi spesso non sono invariabili, ed il grado di incertezza loro associato è considerato nella variabile già introdotta *variabilità dei tempi di consegna*. Il *lead time* è utilizzato nella procedura di aggiornamento dell'inventario per registrare in modo accurato l'effetto degli ordini sui *periodi* futuri. Se il *lead time* è più lungo di un periodo, i limiti dell'intervallo di esistenza di alcune variabili fuzzy sono ampliati, come si vedrà nel prossimo capitolo.

La variabile *genere ordine* serve a valutare l'onerosità associata all'emissione di un ordine, e ha caratteristiche indubbiamente fuzzy, ma la tempistica degli ordini normali è di solito stabilita in maniera precisa dagli accordi con i fornitori. Senza dover precisare quindi per ogni periodo se l'ordine associato è *normale* o *extra*, può essere utile definire in partenza la frequenza degli ordini *normali*, quantificandola con il numero di *periodi* che intercorrono tra un ordine *normale* ed il successivo. L'informazione viene inserita nella variabile ***frequenza ordini normali***, alla quale si deve aggiungere, per poter determinare in ogni *periodo* il tipo di ordine associato, la variabile ***primo ordine normale***, che contiene l'indicazione del primo *periodo* d'interesse caratterizzato da un ordine normale. Grazie all'utilizzo di queste variabili aggiuntive, è sufficiente inserire in *genere ordine* l'indicazione dell'onerosità degli ordini *extra*, e la caratterizzazione di ogni ordine risulta completa.

Variabili aggiuntive a carattere algebrico

- *Lead time*
- Frequenza ordini normali e primo ordine normale
- Confezione d'acquisto

Un ultimo dato, temporaneamente trascurato al momento di sfoltire le variabili, concerne la possibilità di avere dei vincoli sui lotti di consegna. Si è ritenuto opportuno considerare l'eventualità, peraltro frequente, che le confezioni d'acquisto non siano singole, ma contengano

un numero prefissato di articoli. La variabile ***confezione d'acquisto*** indica perciò il numero di unità da considerare lotto minimo e indivisibile per un determinato articolo (uno se l'articolo può essere acquistato singolarmente).

3.6 FUNZIONAMENTO DEL SISTEMA DI GESTIONE

Riassumendo, si può scomporre concettualmente il sistema in due sezioni differenti interagenti tra loro: la gestione dei movimenti e il calcolo dei fabbisogni.

La **gestione dei movimenti** controlla i flussi logici e fisici di merce modificando di conseguenza il *sequenziale* delle giacenze. Il suo funzionamento, basato sulle relazioni riportate nel paragrafo 3.1.3 e sulle variabili aggiuntive introdotte nel paragrafo 3.4.4, è semplice e di tipo classico.

I record del *sequenziale* sono riferiti ad istanti di tempo successivi, ed è importante stabilire l'intervallo di tempo, che abbiamo chiamato ***periodo***, da inserire tra un record ed il seguente. Adottando una prospettiva di controllo continuo delle giacenze, le modifiche apportate all'archivio da ogni movimento vengono registrate immediatamente, ma non è necessario che il *sequenziale* contenga tutte le modifiche a prescindere da quanto sono ravvicinate. Infatti, tenendo conto che un ordine dovrà inevitabilmente lasciare un certo lasso di tempo tra sé e il precedente, proprio questo **intervallo minimo tra due ordini** può essere impiegato come distanza temporale tra i record.

È opportuno che il *periodo* così definito sia sostanzialmente inferiore a quello tra due ordini normali. Se, ad esempio, gli ordini normali vengono emessi una volta al mese, l'intervallo minimo potrebbe essere di una settimana, o ancora meglio di un giorno, sempre che questo abbia un senso in relazione agli accordi stipulati con i fornitori.

Il **calcolo dei fabbisogni**, effettuato in base alla situazione delle giacenze e alle altre variabili d'interesse, costituisce la seconda sezione del sistema, ed è eseguito con un processo di elaborazione fuzzy. Per ogni *periodo* d'interesse, corrispondente ad un record dell'archivio *sequenziale*, si valuta la quantità che si dovrebbe ordinare, e l'opportunità di emettere l'ordine in questione. Il calcolo viene eseguito ogni volta che una movimentazione modifica i valori dell'archivio, oppure allo scadere dei periodi prefissati. Chiamando 1 il *periodo* attuale e t il numero di periodi di *lead time* previsto, il calcolo interessa tutti i record a partire dal $(t+1)$-esimo, fino al primo record in

Descrizione matematica del sistema

Siano:

- P_i record *i*-esimo del *sequenziale*
- V_i variabili d'ingresso riferite al *periodo i*
- P_i' record *i*-esimo dopo l'elaborazione fuzzy
- t *lead time*
- n ultimo *periodo* di calcolo
- f calcolo fuzzy dei fabbisogni
- g calcolo algebrico di gestione dei movimenti

L'elaborazione fuzzy si effettua per tutti i P_i con $t \le i \le n$:

$$P_i' = f(V_i)$$

V_i comprende i campi *DR* e *OF* di P_i.
La modifica di P_i influisce sui *t-1 periodi* precedenti (oltre che su tutti i successivi):

$$P_{i-t}'', \ldots, P_{i-1}'' = g(P_i')$$

l'unico campo non influenzato dal calcolo è *IC*.

cui *esistenza attuale*, *disponibilità reale* e *prevista* coincidono (questo accade quando non ci sono ordini aperti). Se in un qualsiasi *periodo p* si decide di lanciare l'ordine suggerito, si deve modificare di conseguenza il *sequenziale*, facendo in modo che l'ordine al fornitore parta nel periodo *p-t* e arrivi così al periodo che ha richiesto l'ordine. Anche questa modifica causa immediatamente un nuovo calcolo che tenga conto delle modifiche apportate. In seguito ad un ciclo di calcolo completo, ad ogni periodo d'interesse sarà associata una quantità da ordinare (in molti casi 0), che può essere tradotta in un ordine vero e proprio nel momento in cui il record cui si riferisce diventa quello attuale. Da questo istante, escludendo il caso in cui si riesca ad ottenere un *lead time* inferiore a *t*, la disponibilità relativa al record *t*-esimo non può più essere aumentata.

I dettagli di funzionamento del sistema saranno illustrati al momento di descrivere il software sviluppato.

4

PROGETTAZIONE E STRUTTURA DEL SISTEMA FUZZY

4.1 FASE DI ANALISI

L'analisi del problema descritta nel capitolo precedente ha permesso di delineare la struttura generale del sistema di gestione, che risulta essere composto da due sezioni interagenti, una algebrica di aggiornamento dell'inventario ed una fuzzy di calcolo dei fabbisogni.

L'aggiornamento della situazione delle giacenze è eseguito in maniera deterministica, visto che il contenuto d'incertezza del problema è gestito interamente dalla sezione fuzzy. La prima sezione del progetto non presenta quindi caratteristiche distintive particolari. Con essa si attua in sostanza l'aggiornamento continuo dell'archivio delle giacenze, che nel paragrafo 3.2.3 è stato chiamato *sequenziale*.

La sezione di calcolo dei fabbisogni è invece un vero e proprio sistema fuzzy, composto di più blocchi d'inferenza. La progettazione di tale sistema ha richiesto un'analisi accurata, ed è stata eseguita seguendo i passi descritti nel paragrafo 1.11. In questo capitolo sono illustrate le caratteristiche del sistema fuzzy concepito, presentate rispettando la sequenza logica delle fasi di progettazione. La descrizione della fase finale di messa a punto verrà fatta invece nel capitolo sei, dopo aver spiegato più nel dettaglio il funzionamento dell'intero sistema.

Nel corso dell'analisi svolta nel capitolo tre sono state definite per il sistema otto variabili d'ingresso e due variabili d'uscita. Le variabili scelte non sono le sole degne d'interesse, ma si ritiene che possano garantire il giusto compromesso tra le due esigenze opposte di completezza e semplicità. È infatti molto importante che non siano trascurate variabili significative per il problema da affrontare, ma è altrettanto importante che il sistema non risulti appesantito, sia in progettazione che in esecuzione, da variabili o blocchi di calcolo di peso marginale.

Fissate le variabili, si possono ora definire i vari blocchi d'inferenza, con relativi ingressi e uscite. La struttura e la disposizione dei blocchi deve rappresentare il modo in

cui gli ingressi interagiscono, fino a determinare il valore delle uscite finali. Visto il numero di variabili d'ingresso coinvolte, è necessario introdurre alcune variabili intermedie.

Nello stabilire la quantità e le caratteristiche dei blocchi, bisogna ricordare che una riduzione nel numero di blocchi totali, e quindi di variabili intermedie, si paga in termini di complessità dei blocchi stessi, che richiedono molte più regole e risultano più difficili da progettare e da gestire. Per questa ragione, si è deciso di utilizzare solo blocchi a due ingressi e una uscita, che permettono inoltre una più agevole comprensione dei risultati forniti nella fase di messa a punto.

Soluzioni progettuali alternative
Un sistema fuzzy con 4 ingressi, composti ognuno di 5 variabili linguistiche, e una uscita può essere costruito con un solo blocco fuzzy. In tal caso si possono definire fino a 5^4=625 regole. Si può decidere di implementare il sistema con tre blocchi fuzzy e due variabili intermedie, che fungono da uscite dei primi due blocchi e da ingressi del terzo. Con questa soluzione il numero massimo di regole è $3 \cdot 5^2$=75.

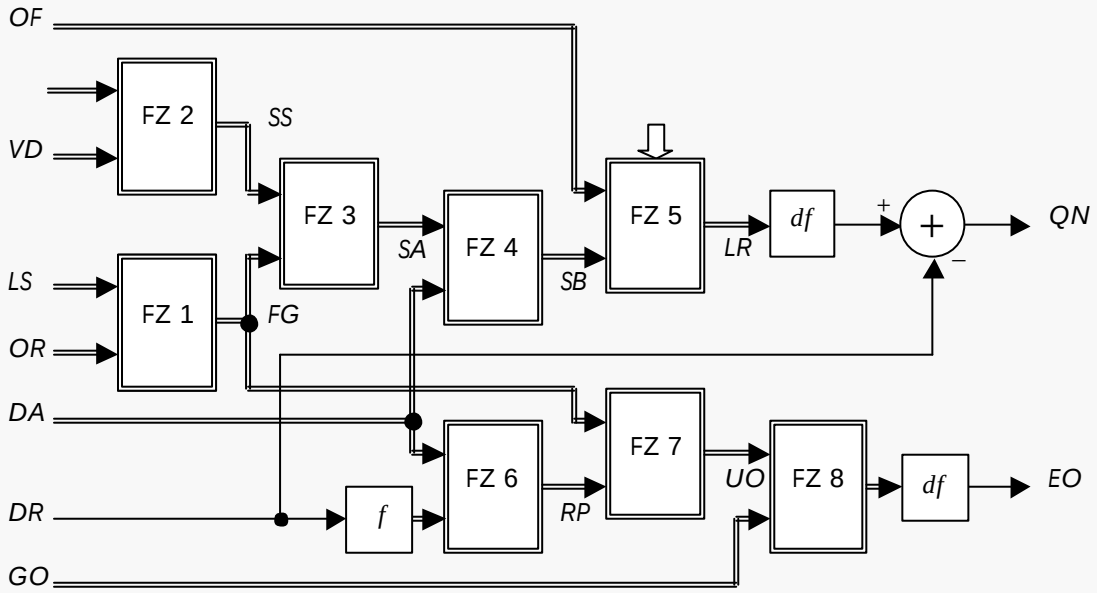
4.1.1 Struttura del sistema

Le due uscite del sistema, pur essendo collegate, sono influenzate diversamente dalle variabili d'ingresso, e si possono perciò determinare secondo due percorsi di calcolo paralleli. La variabile d'uscita *quantità necessaria* è più legata a concetti affini al fabbisogno quantitativo e all'incertezza previsionale, mentre la variabile d'uscita *emissione ordine* attiene soprattutto all'urgenza della consegna e all'onere che questa può comportare. In realtà, ci saranno delle variabili d'ingresso comuni ai due percorsi e delle interazioni tra di essi, ma la distinzione permette di separare parzialmente l'analisi dei due progetti minori, che vengono poi riuniti nella struttura finale.

Al fine di stabilire le variabili d'ingresso e d'uscita di ciascun blocco e definire le variabili intermedie si è cercato di rispettare le affinità più evidenti tra le variabili. Per fare ciò si sono prodotte due strutture ad albero, una per ognuna delle due uscite. I due percorsi di calcolo sono paralleli, ma interagenti, e raggruppano un numero sempre maggiore di fattori man mano che ci si avvicina alle uscite finali. Si è anche studiato il funzionamento di alcuni sistemi di gestione moderni piuttosto complessi, che tengono conto dei margini d'incertezza presenti in ogni decisione, basandosi soprattutto su distribuzioni statistiche e calcoli probabilistici.

Per approfondimenti su sistemi di gestione basati su calcoli probabilistici vedere Berry, Vollmann, Whybark [14].

Importante è stata soprattutto la possibilità di collaborare con esperti della ditta SEAP, aventi notevole esperienza riguardo alle tematiche di gestione scorte. Il contributo dato dalla consultazione di queste persone ha permesso di rendere l'intero progetto più attinente alle problematiche di gestione che si fronteggiano realmente in un'azienda.



Variabili d'ingresso

LS	*Livello di Servizio desiderato*
OR	*Onerosità giacenze e Rimanenze*
VT	*Variabilità Tempo di consegna*
VD	*Variabilità Domanda*
DA	*Domanda Attesa*
OF	*Ordini Fornitori*
DR	*Disponibilità Reale*
GO	*Genere Ordine*

Variabili intermedie

FG	*Filosofia di Gestione*
SS	*Scorta di Sicurezza*
SA	*Scorta Aggiuntiva*
SB	*Scorta Base*
LR	*Livello di Riordino*
RP	*Rotture Previste*
UO	*Urgenza Ordine*

Variabili d'uscita

QN	*Quantità Necessaria*
EO	*Emissione Ordine*

Fig. 4.1 Schema generale del blocco di elaborazione fuzzy con legenda delle variabili.

4.1.2 Descrizione generale

Lo schema complessivo risultante dalla fase di analisi è rappresentato nella fig. 4.1. Le linee doppie rappresentano variabili e blocchi fuzzy, quelle semplici variabili ed operatori algebrici. Le variabili d'ingresso devono sempre essere sottoposte all'operazione di fuzzificazione, che è sottintesa per tutte le variabili tranne che per la *disponibilità reale* (*DR*), essendo essa utilizzata sia come variabile fuzzy sia con il suo valore numerico effettivo.

Si possono osservare i due percorsi di calcolo che portano alle due uscite: il primo percorso determina la quantità necessaria al ripristino delle scorte (*QN*), il secondo decide se è opportuna l'emissione dell'ordine (*EO*). Prima di illustrare le caratteristiche specifiche di ogni blocco

forniamo una descrizione generale delle due sezioni del sistema.

Determinazione della quantità necessaria

Questa parte del sistema coinvolge ben sette delle otto variabili d'ingresso. Il fabbisogno è deciso in base alla situazione effettiva delle giacenze, ai consumi previsti e a una serie di variabili tendenti a influenzare opportunamente la quantità di scorte ritenuta più adatta nelle diverse situazioni. La sequenza logica delle operazioni svolte risulta più evidente se la spiegazione del funzionamento viene fatta a ritroso, partendo dall'uscita finale per giungere ai primi blocchi di calcolo e alle variabili d'ingresso.

La variabile d'uscita ***quantità necessaria*** (*QN*) è calcolata algebricamente come differenza tra la variabile intermedia ***livello di riordino*** (*LR*) e le ***disponibilità reali*** (*DR*), che in questa parte del sistema rappresentano una variabile numerica non fuzzy. L'uscita fuzzy della prima parte del sistema è quindi proprio la variabile *LR*, che rappresenta la quantità a cui si ritiene debbano essere ripristinate le giacenze. Su tale variabile si deve eseguire l'operazione di defuzzificazione, rappresentata nella fig. 4.1 con il simbolo *df*, al fine di ottenere il valore numerico da cui sottrarre l'entità delle *disponibilità reali.*

Nel determinare il *livello di riordino* si deve tenere conto tra le altre cose di eventuali **ordini in corso**. Trascurare l'eventuale esistenza di lotti di merci in arrivo porta infatti a sovrastimare i reali fabbisogni. Nel sistema la questione è affrontata nel **blocco 5**, che stabilisce il *livello di riordino* a partire da eventuali ***ordini dai fornitori*** (*OF*) attualmente in corso e dal valore della **scorta base** (*SB*).

La *scorta base*, definibile come la quantità di ripristino al netto degli ordini in corso, viene determinata nel **blocco 4**, tenendo conto del contributo della ***domanda attesa*** (*DA*) e della ***scorta aggiuntiva*** (*SA*). La *domanda attesa* è una stima dei consumi che si avranno nel corso del *lead time*, la *scorta aggiuntiva* serve ad introdurre nella gestione le fonti d'incertezza e gli orientamenti strategici dell'azienda. Essa viene calcolata nel **blocco 3**, a partire dalla variabile ***filosofia di gestione*** (*FG*) e dal valore della ***scorta di sicurezza*** (*SS*), uscite dei primi due blocchi fuzzy.

Nel **blocco 1** gli ingressi sono costituiti dalle due variabili ***livello di servizio desiderato*** (*LS*) e ***onerosità di giacenze e rimanenze*** (*OR*), entrambe direttamente legate alla strategia aziendale. Esse influenzano la *filosofia di gestione*, variabile che serve sia per determinare la *scorta aggiuntiva* sia per stabilire l'urgenza di una ordinazione.

Nel **blocco 2** viene stabilita la variabile *scorta di sicurezza*, introdotta per assicurarsi un margine di errore nelle stime. Gli ingressi del blocco sono la ***variabilità della domanda*** (*VD*) e la ***variabilità dei tempi di consegna*** (*VT*), che influenzano in modo simile la gestione, valutando il livello d'incertezza insito nel processo decisionale.

Emissione dell'ordine

Nella parte inferiore dello schema di fig. 4.1 è illustrata la sezione di calcolo incaricata di stabilire se la proposta d'ordine espressa nella variabile *quantità necessaria* deve effettivamente tradursi nell'emissione dell'ordine stesso.

Anche in questo caso si procede a ritroso, partendo dal **blocco 8**. Esso è incaricato di stabilire se l'ordine deve essere emesso, e il valore assegnato alla variabile d'uscita ***emissione ordine*** (*EO*) costituisce la risposta finale. La decisione viene presa in base al ***genere dell'ordine*** (*GO*) in questione e all'***urgenza dell'ordine*** stesso (*UO*).

Il *genere dell'ordine* può essere *normale* o *extra*, e in quest'ultimo caso si deve valutare l'onerosità aggiuntiva che comporterebbe la sua emissione. L'*urgenza dell'ordine* è invece stabilita nel **blocco 7**, a seconda della *filosofia di gestione* prescelta (uscita del blocco 1) e dell'entità delle ***rotture di scorta previste*** (*RP*) nel caso l'ordine non fosse emesso.

Il valore della variabile *RP* è a sua volta determinato nel precedente **blocco 6**, avente come ingressi due variabili già impiegate nella determinazione del fabbisogno: la *domanda attesa* e la *disponibilità reale*, questa volta utilizzata come variabile fuzzy (nella fig. 4.1 è evidenziata in un blocco specifico l'operazione di fuzzificazione applicata a *DR*).

4.2 SCELTE DI PROGETTAZIONE DI VALIDITÀ GENERALE

La progettazione di dettaglio, successiva alla fase di analisi, comprende la scelta delle caratteristiche degli ingressi, dell'uscita e del motore d'inferenza per ogni blocco fuzzy progettato.

Alcune decisioni prese derivano da considerazioni generali valide per il sistema nel suo complesso, altre sono peculiari dei diversi blocchi d'inferenza. Si presentano dapprima le caratteristiche comuni a tutti i blocchi, per poi entrare nel dettaglio delle particolarità progettuali di ogni blocco. Vale la pena di fare però una premessa sui tempi di calcolo.

4.2.1 Tempi di calcolo

Sono dette **operazioni elementari** le operazioni che un elaboratore considera naturali e che può quindi svolgere in un solo passaggio (somme, confronti e, nei processori moderni, anche prodotti). Ogni operazione complessa è equivalente, per quanto riguarda il suo tempo di esecuzione, ad un certo numero di operazioni elementari.

I tempi di calcolo di un processo sono proporzionali al numero di **operazioni elementari** che il calcolatore deve eseguire nel corso dell'elaborazione. Il numero di operazioni richieste per l'esecuzione di un processo è detto **costo computazionale** del processo.

Il numero di operazioni elementari richieste da un blocco d'inferenza fuzzy può essere in alcuni casi anche molto elevato, e varia considerevolmente in base agli operatori scelti per l'esecuzione delle varie fasi del processo di elaborazione. Il costo computazionale di un processo di calcolo fuzzy è influenzato in modo determinante dalla scelta delle funzioni di appartenenza e delle operazioni di fuzzificazione e defuzzificazione.

Nei sistemi di controllo fuzzy, impiegati ad esempio nell'elettronica, il costo computazionale è un fattore fondamentale per valutare l'applicabilità di una certa soluzione, data l'importanza che assumono in questi casi i tempi di risposta. Nel caso dei sistemi decisionali come il nostro, è più improbabile che i tempi di risposta rappresentino un aspetto critico del problema. Per questa ragione, nel progetto sono state adottate anche soluzioni complesse, pur se costose dal punto di vista computazionale, nei casi in cui esse garantivano una maggior precisione e una maggiore rispondenza ai requisiti del problema. In ogni caso, i tempi di esecuzione sono risultati dell'ordine di pochi secondi, quantità di tempo improponibile per processi automatici, ma del tutto accettabile per un processo decisionale.

Si indicano di seguito alcune scelte operate anche sulla scorta del ragionamento esposto, relative a diverse fasi della progettazione.

4.2.2 Fuzzificazione

L'applicazione *UNFUZZY*, scelta per la progettazione del sistema fuzzy, mette a disposizione dell'utente cinque tipi di insiemi tra i quali scegliere per assegnare la forma voluta all'insieme d'ingresso: singleton, a triangolo, a trapezio, a campana, a campana trapezoidale. Con tale applicazione è inoltre possibile specificare per ogni insieme d'ingresso, ad esclusione dei singleton, su quanti punti dovrà essere calcolata l'intersezione con le funzioni di appartenenza al fine di determinare il grado di attivazione delle regole.

Per testare la validità delle varie scelte possibili, si sono provate diverse soluzioni su uno dei blocchi del sistema e si è analizzato graficamente l'andamento della risposta.

L'obiettivo era quello di ottenere una risposta che variasse in modo graduale, senza repentine variazioni di pendenza o gradini evidenti, che possono verificarsi a volte in corrispondenza dei valori di passaggio tra le regole. Pendenze di maggiore o minore entità si possono ottenere agendo opportunamente sulle regole, ma in questo caso sono desiderate, e non dovute ad imperfezioni progettuali. Osservando la fig. 4.2 è possibile fare un raffronto immediato di alcune delle alternative provate.

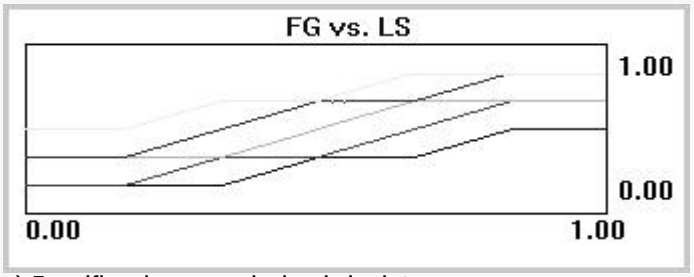
In considerazione della risposta molto spigolosa che ha fornito, la fuzzificazione con insiemi singleton (fig.4.2.a) è stata scartata in partenza. La sua validità è legata infatti unicamente ai bassissimi tempi di esecuzione che richiede.

Passando ad insiemi di altro tipo, si è verificato che, più che la forma degli insiemi stessi, è fortemente influente sul risultato il numero di punti di valutazione impiegati. Insiemi con cinque punti di valutazione forniscono risposte considerevolmente più regolari rispetto a quelli con tre punti di valutazione, come si può verificare confrontando i grafici 4.2.c 4.2.d riferiti entrambi all'insieme trapezoidale. Non si osservano invece ulteriori miglioramenti dall'utilizzo di sette punti di calcolo, che richiedono inoltre tempi decisamente più lunghi.

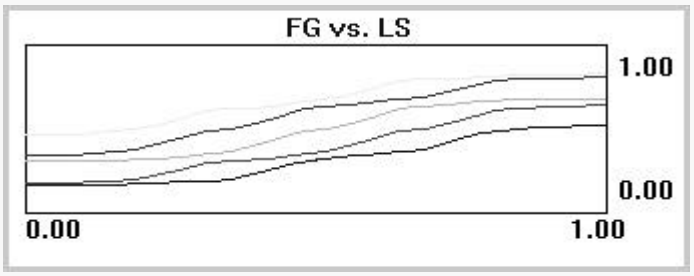
Pur non essendoci vincoli temporali da rispettare, la questione dei tempi d'esecuzione si è rivelata nel nostro contesto non trascurabile, poiché l'aumento delle operazioni necessarie al calcolo è più che proporzionale al numero di punti impiegati. La successiva scelta di un operatore di defuzzificazione veloce ha comunque permesso di utilizzare la fuzzificazione con cinque punti di valutazione, che altrimenti si sarebbe rivelata piuttosto onerosa da questo punto di vista.

Per quanto riguarda la scelta dell'insieme di fuzzificazione, le varie alternative sono state sperimentate, tramite prove incrociate, assieme alle diverse forme disponibili per le funzioni d'appartenenza. La risposta più regolare è stata fornita dall'insieme di fuzzificazione trapezoidale, decisamente preferibile rispetto a quello triangolare (fig. 4.2.d 4.2.b rispettivamente). Risultati simili, ma non migliori, sono stati ottenuti con insiemi di fuzzificazione a campana trapezoidale applicati a variabili con funzioni d'appartenenza a campana (fig. 4.2.e).

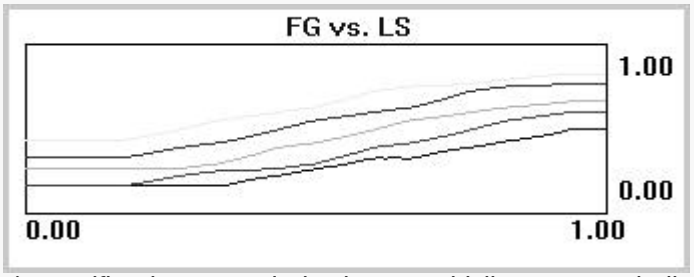
La scelta finale è ricaduta quindi su insiemi d'ingresso trapezoidali con cinque punti di valutazione, che offrono le loro migliori prestazioni su *term set* formati da insiemi fuzzy triangolari. L'alternativa preferita è illustrata, oltre che nella fig. 4.2.d, anche nella fig.4.5, in cui viene mostrata l'intera finestra di calcolo.



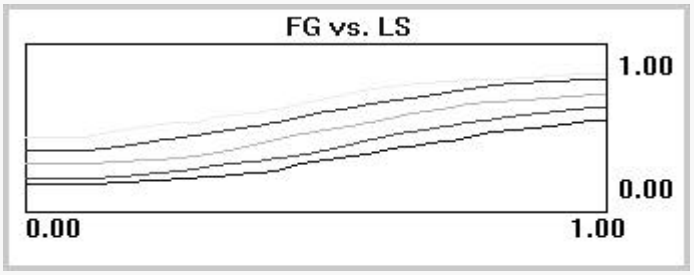
a) Fuzzificazione con insiemi singleton



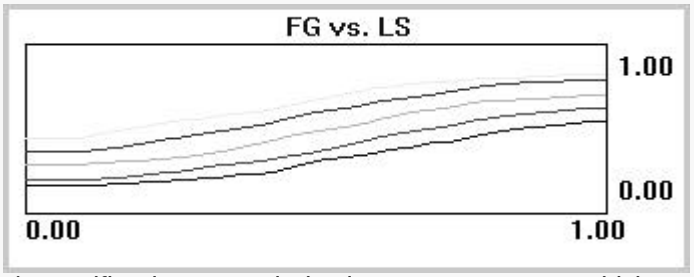
b) Fuzzificazione con insiemi triangolari a cinque punti di valutazione



c) Fuzzificazione con insiemi trapezoidali a tre punti di valutazione



d) Fuzzificazione con insiemi trapezoidali a cinque punti di valutazione



e) Fuzzificazione con insiemi a campana trapezoidale a cinque punti di valutazione

Fig. 4.2 Comparazione di diversi tipi di insiemi di fuzzificazione applicati al blocco d'inferenza fuzzy 1. È rappresentata in ordinata l'uscita *FG* in funzione dell'ingresso *LS*, con la variabile *OR* impiegata come parametro. Il tipo di fuzzificazione prescelto, insiemi trapezoidali a cinque punti di valutazione, è illustrato nella fig. 4.2.d.

Caratteristica comune a tutte le variabili è l'intervallo di definizione normalizzato [0,1], impiegato sia per le variabili in ingresso sia per quelle in uscita dai blocchi.

La forma stabilita per gli **insiemi d'appartenenza** è quella triangolare, che si adatta molto bene al tipo di fuzzificazione scelta.

Per quanto riguarda infine il numero di termini linguistici di ogni variabile è sembrato che cinque valori potessero essere sufficienti. Con blocchi a due ingressi, non è giustificata la riduzione a tre valori per variabile, mentre il tentativo fatto di portare a sette i termini di alcune variabili non ha procurato miglioramenti sensibili.

Alcune considerazioni più specifiche, concernenti gli **insiemi d'appartenenza** ai margini degli intervalli di definizione delle variabili, saranno esposte nella descrizione dettagliata dei blocchi.

4.2.3 Defuzzificazione

Per la scelta dell'operatore di defuzzificazione si hanno cinque alternative: i tre operatori di massimo (primo massimo, ultimo massimo e media dei massimi), il metodo del baricentro e l'operatore denominato *Altura*, simile all'operatore *COM*, che calcola una media ponderata tra i valori centrali degli insiemi inferiti da ogni regola, con pesi dati dai gradi di verità delle regole stesse.

Nel caso della defuzzificazione, i metodi di massimo e quelli di baricentro sono rappresentativi di due filosofie risolutive divergenti, orientate la prima ad una scelta netta, che privilegia le regole con grado di attivazione massimo su tutte le altre, la seconda ad una soluzione più complessa, che tiene conto di tutti i contributi, anche di quelli con influenza minima.

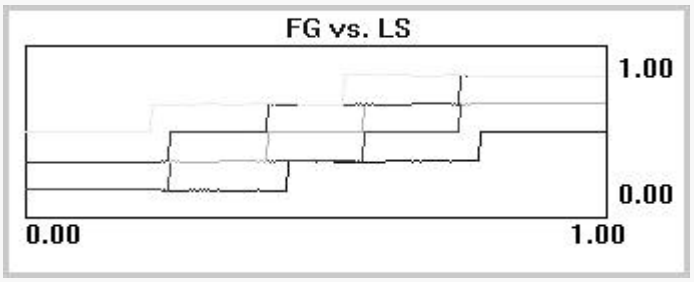
Nella fig. 4.3 si possono confrontare le due filosofie di defuzzificazione. Si noti come, applicando un metodo di massimo (in questo caso la *media dei massimi*, fig. 4.3.a), il passaggio da un valore ad un altro della variabile linguistica sia netto, rendendo quasi impossibile l'assunzione di valori intermedi ai cinque prestabiliti. Viceversa, i metodi di baricentro, come il *COG* che ha originato il grafico di fig. 4.3.b, determinano variazioni molto più graduali

Nel nostro caso, il metodo della *media dei massimi* è senza dubbio più adatto ad essere utilizzato nel blocco che deve stabilire se emettere l'ordine oppure no, per il quale una risposta di compromesso non avrebbe alcun senso. Per gli altri blocchi, di tipo numerico, si sono preferiti i metodi di baricentro, che permettono di non perdere informazioni durante il processo di elaborazione.

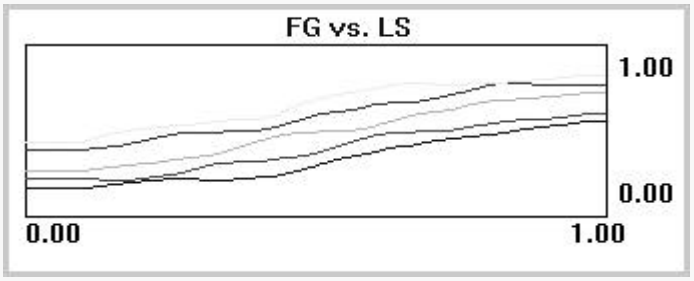
Una serie di prove eseguita con le stesse modalità impiegate per gli operatori di fuzzificazione ha fatto rilevare che l'operatore *Altura*, oltre a permettere calcoli molto più rapidi rispetto all'operatore centro di gravità (*COG*), offre

anche una risposta dall'andamento più regolare. Ciò è dovuto al fatto che, nei casi in cui molte regole sono attivate contemporaneamente, alcune di esse possono generare insiemi d'uscita molto sovrapposti. In tali casi, mentre l'operatore *Altura* tiene separatamente conto di tutte le regole, il *COG* valuta solo la forma dell'insieme finale, trascurando il fatto che la stessa forma può in realtà provenire da contributi diversi. Le differenze sono evidenti se si paragona il grafico di fig. 4.3.b, che illustra i risultati forniti dall'operatore *COG*, con quello della fig. 4.5, che mostra la stessa elaborazione effettuata utilizzando l'operatore *Altura*. Un altro vantaggio che si può ottenere impiegando questo operatore è la possibilità di raggiungere i limiti estremi degli insiemi di esistenza per le variabili d'uscita (impossibile per il metodo *COG*).

Optare per l'operatore più veloce ha permesso inoltre di utilizzare il fuzzificatore a trapezio con cinque punti di valutazione, che comporta dei miglioramenti notevoli all'andamento della risposta.



a) Defuzzificazione con il metodo della *media dei massimi*



b) Defuzzificazione con il metodo *COG* (centro di gravità)

Fig. 4.3 Comparazione delle due principali strategie di defuzzificazione.

Come le variabili d'ingresso, anche le uscite, a parte *EO* che può assumere solo i valori *sì*/*no*, presentano cinque valori possibili, descritti da funzioni triangolari, e sono definite nell'intervallo [0,1]. Si rimanda alla descrizione dettagliata dei blocchi per caratteristiche più specifiche e per la post-elaborazione.

4.2.4 Motore d'inferenza

Le regole d'inferenza, da inserire nella base di conoscenza di un blocco fuzzy, devono chiaramente essere elaborate in maniera specifica all'interno di ogni blocco, una volta stabiliti in modo accurato ruolo e caratteristiche delle variabili. Tuttavia, alcune decisioni concernenti le procedure di calcolo possono considerarsi valide per tutti i blocchi, mentre la trattazione dei casi particolari è rimandata alla descrizione dettagliata dei blocchi stessi. Ad esempio, la scelta degli operatori d'implicazione e composizione può essere legata più alle caratteristiche dell'intero sistema che a quelle del singolo blocco fuzzy.

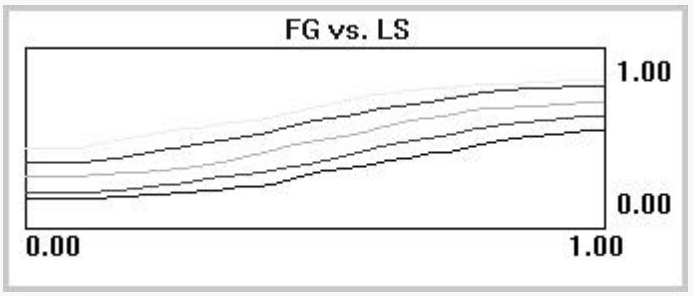
Per saggiare la validità delle alternative possibili, sono state eseguite prove sui diversi operatori forniti, come per i processi di fuzzificazione e defuzzificazione. Anche in questo caso, tenendo sempre conto dei tempi di esecuzione, le scelte sono state fatte in base all'andamento delle uscite in risposta ad una variazione continua delle variabili d'ingresso. Sono stati privilegiati i metodi di calcolo che hanno fornito curve di risposta meno ripide (indice di variazioni più graduali) e senza bruschi cambiamenti di pendenza.

Confrontando la figura 4.4, che illustra i risultati conseguiti con alcune delle soluzioni disponibili, e la fig.4.5, che mostra l'esito della prova eseguita con la combinazione di alternative scelta, si possono verificare le effettive variazioni che comporta la sostituzione dei diversi operatori.

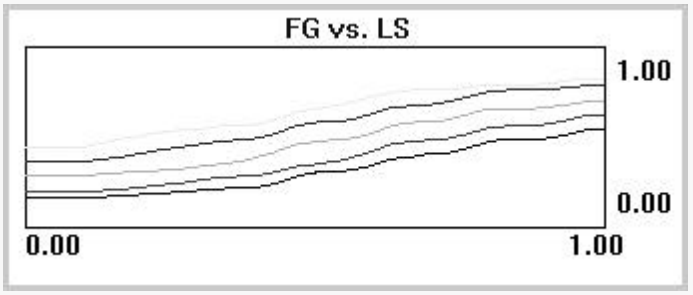
Le prove hanno evidenziato che la scelta dell'operatore d'implicazione (fig. 4.4.a) ha un'influenza marginale sull'andamento della risposta (le differenze sono quasi impercettibili). Si è quindi preferito l'*operatore di minimo*, che offre tempi di calcolo sostanzialmente più contenuti rispetto al *prodotto algebrico* fuzzy.

Più complessa si è rivelata la decisione riguardante l'operatore di composizione (fig. 4.4.b). Una serie di prove ha comunque dimostrato che la composizione *max-product* comporta variazioni di pendenza meno repentine sia rispetto alla *max-min*, sia rispetto ad operatori più complessi.

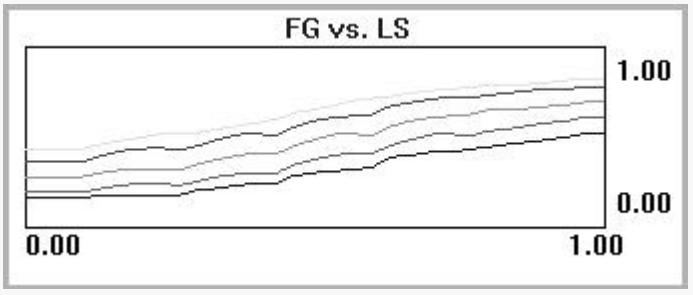
Anche l'opzione riguardante l'intersezione degli antecedenti ha richiesto diverse prove (fig. 4.4c). È stato preferito agli altri l'operatore di *prodotto algebrico*, ed anche se in alcune prove le curve di risposta sono praticamente le stesse fornite da altri operatori, in certi casi, come quello rappresentato in figura, i miglioramenti sono evidenti.



a) Implicazione con operatore prodotto algebrico al posto dell'operatore di minimo



b) Composizione con operatore di minimo al posto del prodotto algebrico



c) Intersezione and con operatore di minimo al posto del prodotto algebrico

Fig. 4.4 Comparazione di diversi operatori impiegati nel processo d'inferenza del blocco fuzzy 1. In ogni elaborazione si è cambiato, rispetto alla combinazione di operatori scelta, solo l'opzione indicata nella didascalia.

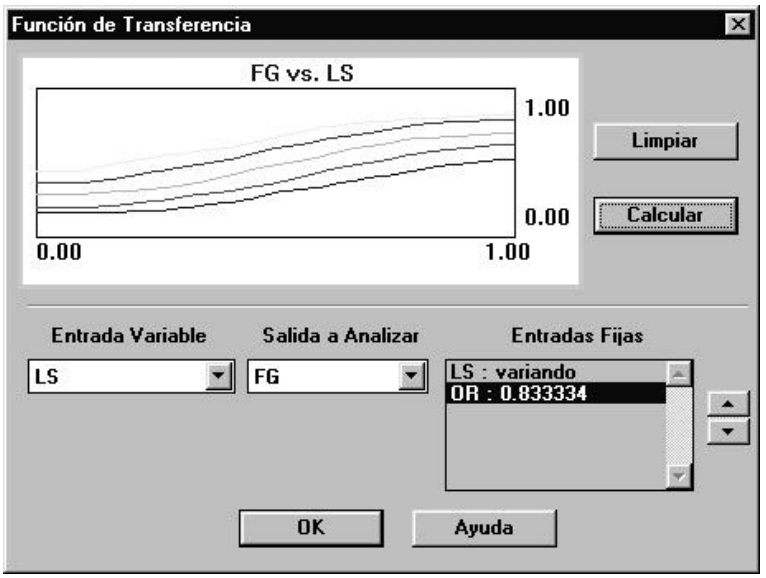


Fig. 4.5 Elaborazione eseguita con la combinazione di operatori scelta: prodotto algebrico per composizione e intersezione e minimo per l'implicazione. La defuzzificazione è eseguita con l'operatore *Altura*.

4.3 DETTAGLI PROGETTUALI DEI BLOCCHI DI ELABORAZIONE

Nelle prossime pagine sono descritti in modo particolareggiato gli otto blocchi fuzzy che costituiscono il sistema di calcolo.

Per ogni blocco è presentata prima di tutto una tabella riassuntiva con le caratteristiche principali delle variabili e dei loro valori linguistici. A questo proposito, si può anticipare che i valori linguistici scelti per la progettazione saranno modificati nell'ambito dell'interfaccia grafica utente. Questo perché nell'ambito progettuale la maggior significatività dei termini è preferita all'omogeneità degli stessi, che invece serve per rendere più chiara e semplice la fase di esecuzione per l'utente finale.

Segue una descrizione del significato delle variabili e del blocco stesso, con particolare attenzione al modo in cui le variabili d'ingresso influenzano l'uscita.

In basso si nota invece il grafico della funzione di trasferimento, che mostra l'andamento dell'uscita in funzione dei valori degli ingressi. La rappresentazione tridimensionale permette di coprire integralmente i campi d'esistenza di entrambi gli ingressi, mostrando tutti i possibili casi di funzionamento del blocco.

Nella facciata a fronte, sono rappresentati graficamente i *term set* delle tre variabili, con a fianco i valori numerici precisi dei punti di definizione delle funzioni di appartenenza. Le forme triangolari si possono descrivere con le due ascisse dei vertici alla base del triangolo (prima e terza colonna) e con l'ascissa del vertice avente ordinata uno (seconda colonna). Per le forme agli estremi degli universi di definizione, quando non sono triangolari, sono sufficienti due valori. Uno è l'ascissa dell'unico vertice di ordinata zero (terza o prima colonna a seconda se l'insieme descrive il primo o l'ultimo dei valori linguistici), l'altro l'ascissa del vertice superiore adiacente ad esso (seconda colonna). Quest'ultimo, nel caso di forme a triangolo rettangolo, coincide con il terzo vertice, il quale non è indicato, avendo sempre per ordinata uno e per ascissa un estremo dell'universo di esistenza della variabile.

Più in basso, si mostra l'insieme delle regole linguistiche in formato tabulare. Infine, si offrono alcune precisazioni sulle particolarità progettuali dell'intero blocco, con attenzione soprattutto alla struttura delle regole e alle forme delle funzioni d'appartenenza.

4.3.1 Blocco fuzzy 1

Variabili d'ingresso				Variabile d'uscita	
Livello di Servizio	*LS*	*Onerosità Rimanenze*	*OR*	*Filosofia di Gestione*	*FG*
Minimo	MM	Zero	Z	Rimanenze Minime	RM
Basso	B	Bassa	B	Rimanenze Basse	RB
Medio	M	Media	M	Neutra	N
Alto	A	Alta	A	Buon Servizio	BS
Molto Alto	MA	Molto Alta	MA	Ottimo Servizio	OS

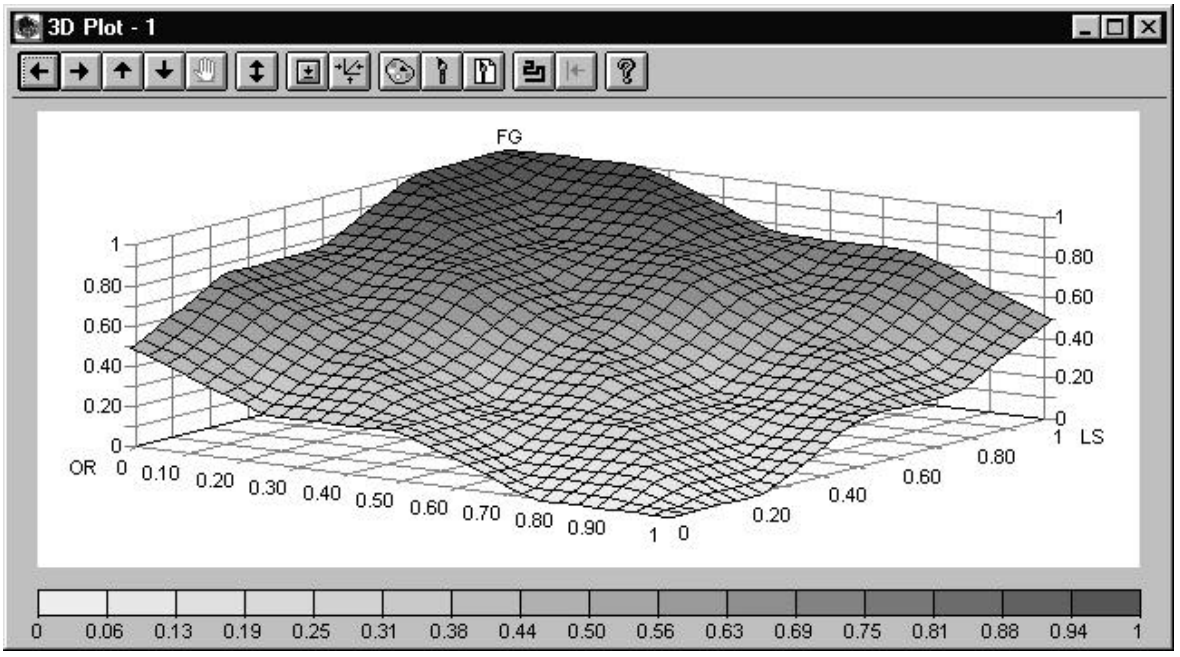
Fuzzificazione		Defuzzificazione
Insieme *Pi* (trapezoidale) con 5 punti di valutazione	Insieme *Pi* (trapezoidale) con 5 punti di valutazione	Operatore *Altura* (media pesata dei risultati delle regole)

La *Filosofia di Gestione*, variabile d'uscita del blocco, esprime gli obiettivi strategici che si perseguono con la gestione di scorte. Essa può mutare gradualmente tra gli estremi di massimo servizio reso (OS) e di massima economia di gestione (RM), relativi a due visioni diametralmente opposte della strategia aziendale. Al primo estremo ci si avvicina al crescere del livello di servizio desiderato, mentre l'aumento dei costi associati alle scorte tende invece a portare alla diminuzione del livello delle giacenze.

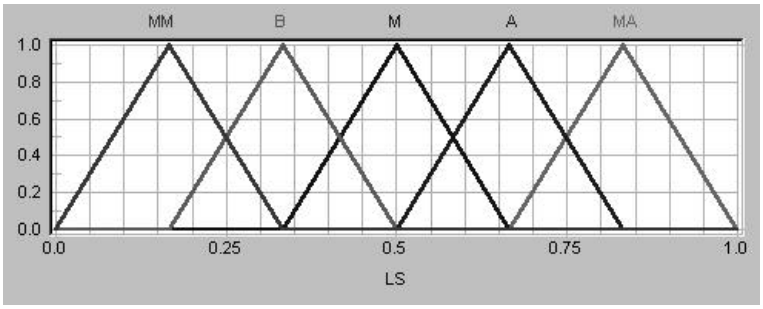
La variabile d'ingresso *Livello di Servizio* serve a valutare quanto sia importante evitare che si verifichino rotture di scorta. Ciò dipende dalle conseguenze che un eventuale mancanza di giacenze disponibili potrebbe causare (si veda al proposito il paragrafo 3.3.5 e in particolare la tabella 3.3).

Con la variabile *Onerosità delle Rimanenze* si tiene conto in primo luogo dell'entità dei costi associati al mantenimento delle giacenze, considerati in genere proporzionali al valore delle giacenze stesse. La stessa variabile è impiegata anche per dar peso alle situazioni di deperibilità della merce o di rapida perdita di valore. In tali casi aumenta l'importanza di evitare che gli articoli rimangano a lungo giacenti in magazzino.

Funzione di trasferimento

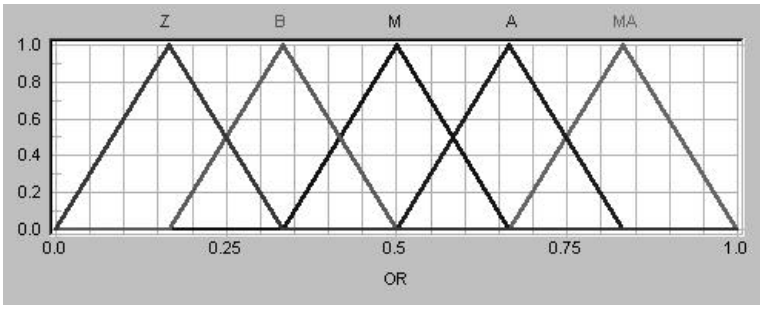


Variabile d'ingresso *LS*



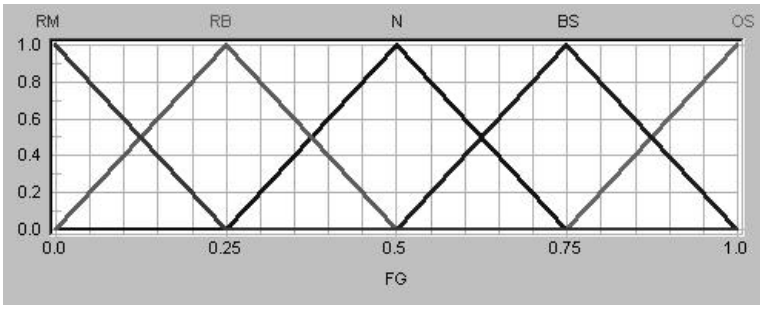
Insiemi	Punti di definizione		
MM		0,167	0,333
B	0,167	0,333	0,500
M	0,333	0,500	0,667
A	0,500	0,667	0,833
MA	0,667	0,833	

Variabile d'ingresso *OR*



Insiemi	Punti di definizione		
Z		0,167	0,333
B	0,167	0,333	0,500
M	0,333	0,500	0,667
A	0,500	0,667	0,833
MA	0,667	0,833	

Variabile d'uscita *FG*



Insiemi	Punti di definizione		
RM		0,000	0,250
RB	0,000	0,250	0,500
N	0,250	0,500	0,750
BS	0,500	0,750	1,000
OS	0,750	1,000	

Motore d'inferenza

Uscita: *FG*		*OR*				
		Z	B	M	A	MA
LS	MM	N	RB	RB	RM	RM
	B	BS	N	RB	RB	RM
	M	BS	BS	N	RB	RB
	A	OS	BS	BS	N	RB
	MA	OS	OS	BS	BS	N

La caratteristica peculiare di questo blocco è la forma delle funzioni di appartenenza della variabile d'uscita. Gli insiemi agli estremi dell'universo di esistenza della variabile sono infatti a forma di triangolo rettangolo, invece che di trapezio rettangolo, come per gli ingressi, o di triangolo, come accade spesso per le uscite. In questo modo, se si impiega un operatore di fuzzificazione appropriato, come l'operatore *Altura*, è possibile far assumere all'uscita tutti i valori del suo insieme di definizione, cosa altrimenti irrealizzabile.

4.3.2 Blocco fuzzy 2

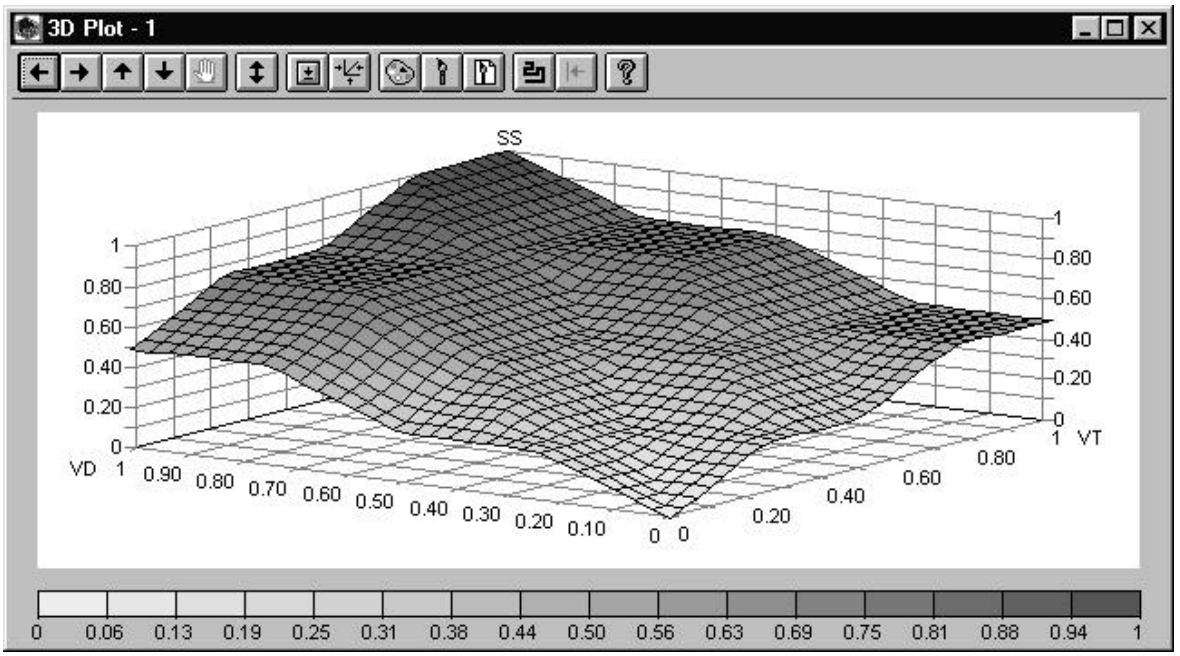
Variabili d'ingresso				Variabile d'uscita	
Variabilità Tempi	*VT*	*Variabilità Domanda*	*VD*	*Scorta di Sicurezza*	*SS*
Zero	Z	Zero	Z	Zero	Z
Bassa	B	Bassa	B	Bassa	B
Media	M	Media	M	Media	M
Alta	A	Alta	A	Alta	A
Molto Alta	MA	Molto Alta	MA	Molto Alta	MA
Fuzzificazione				Defuzzificazione	
Insieme *Pi* (trapezoidale) con 5 punti di valutazione		Insieme *Pi* (trapezoidale) con 5 punti di valutazione		Operatore *Altura* (media pesata dei risultati delle regole)	

Entrambe le variabili d'ingresso del blocco servono a quantificare l'incertezza associata a previsioni degli eventi futuri. La *Variabilità dei Tempi di consegna* valuta l'affidabilità dei tempi indicati dai fornitori (o previsti sulla base di esperienze passate e di altri fattori) ed inseriti nella variabile algebrica *lead time.* La *Variabilità della Domanda* assegna invece un grado di affidabilità al pronostico fatto sulle richieste di consegna future ed espresso con la variabile fuzzy *Domanda Attesa*.

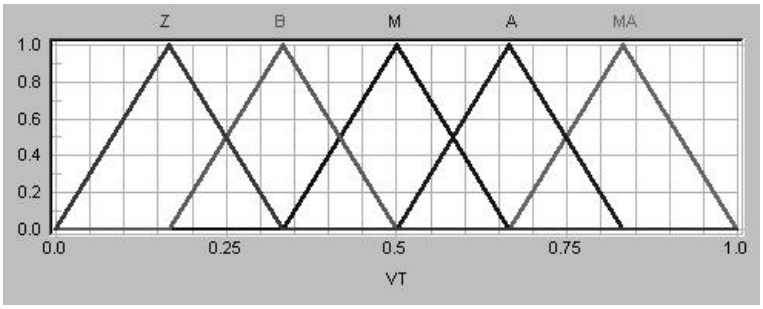
La *Scorta di Sicurezza* serve quindi a coprire le oscillazioni delle grandezze impiegate come variabili, che inevitabilmente si verificano in un processo aleatorio come la gestione di scorte. Essa cresce quanto più si ritiene poco prevedibile il comportamento futuro delle diverse grandezze in gioco, e quindi all'aumentare sia della *Variabilità dei Tempi* sia della *Variabilità della Domanda*.

Il peso relativo delle due variabili è lo stesso, come si può osservare dalla simmetria della funzione di trasferimento, ma è più spesso la *Domanda Attesa* a presentare i più alti valori di imprevedibilità.

Funzione di trasferimento

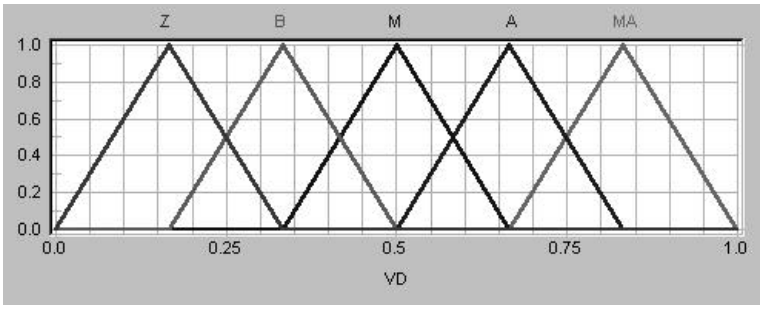


Variabile d'ingresso *VT*



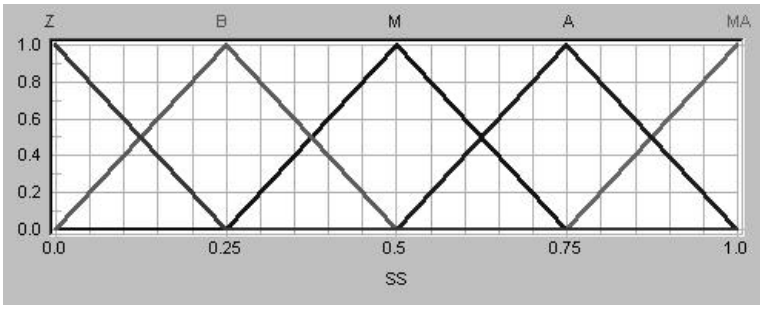
Insiemi	Punti di definizione		
Z		0,167	0,333
B	0,167	0,333	0,500
M	0,333	0,500	0,667
A	0,500	0,667	0,833
MA	0,667	0,833	

Variabile d'ingresso *VD*



Insiemi	Punti di definizione		
Z		0,167	0,333
B	0,167	0,333	0,500
M	0,333	0,500	0,667
A	0,500	0,667	0,833
MA	0,667	0,833	

Variabile d'uscita *SS*



Insiemi	Punti di definizione		
Z		0,000	0,250
B	0,000	0,250	0,500
M	0,250	0,500	0,750
A	0,500	0,750	1,000
MA	0,750	1,000	

Motore d'inferenza

Uscita: *SS*		*VD*				
		Z	B	M	A	MA
VT	Z	Z	B	B	M	M
	B	B	B	M	A	A
	M	B	M	M	A	A
	A	M	M	A	A	MA
	MA	M	M	A	A	MA

Le considerazione fatte a proposito della forma degli insiemi di appartenenza della variabile d'uscita nel primo blocco valgono anche in questo caso. Soprattutto è importante che la *Scorta di Sicurezza* possa assumere il valore zero numerico. Non ha infatti senso tenere scorte aggiuntive se si ritiene che non ci siano elementi di incertezza nella gestione, come nel caso in cui si lavori solo su ordini (*Domanda Attesa* sempre nulla) e con consegne completamente affidabili.

4.3.3 Blocco fuzzy 3

Variabili d'ingresso — Variabile d'uscita

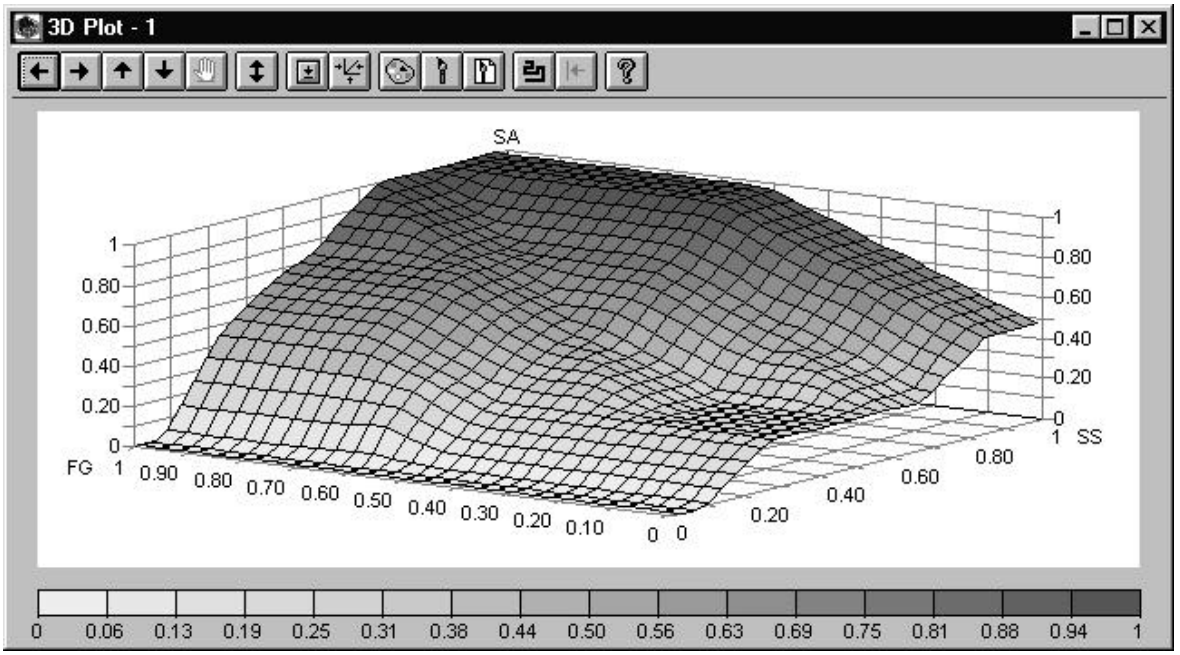
Scorta di Sicurezza	*SS*	*Filosofia di Gestione*	*FG*	*Scorta Aggiuntiva*	*SA*
Zero	Z	Rimanenze Minime	RM	Zero	Z
Bassa	B	Rimanenze Basse	RB	Bassa	B
Media	M	Neutra	N	Media	M
Alta	A	Buon Servizio	BS	Alta	A
Molto Alta	MA	Ottimo Servizio	OS	Molto Alta	MA

Fuzzificazione		Defuzzificazione
Insieme *Pi* (trapezoidale) con 5 punti di valutazione	Insieme *Pi* (trapezoidale) con 5 punti di valutazione	Operatore *Altura* (media pesata dei risultati delle regole)

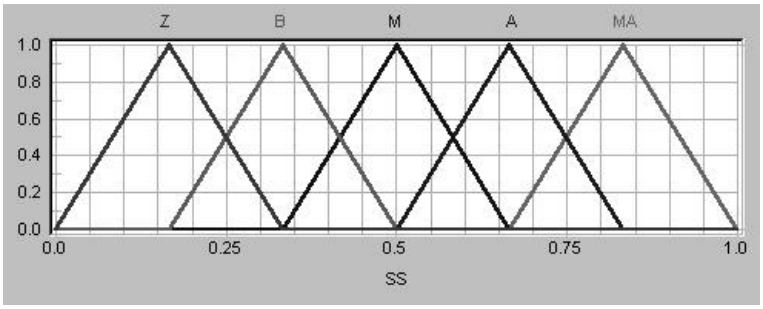
Stabilito nel blocco 1 l'orientamento strategico della gestione e nel blocco 2 il livello d'incertezza ad essa associato, si tratta ora di riunire l'influenza dei due fattori, per valutare l'entità delle giacenze che andranno ad aggiungersi a quelle strettamente necessarie per far fronte alla *Domanda Attesa*, variabile che non contiene in se stessa l'indicazione dell'aleatorietà ad essa associata .

La variabile d'uscita di questo blocco, i cui ingressi sono le uscite dei blocchi 1 e 2, è la *Scorta Aggiuntiva.* Essa esprime la quantità di giacenze che servono a creare un margine di sicurezza rispetto alle semplici esigenze stimate, margine valutato in base alle condizioni di incertezza e a come vengono affrontate a seconda degli obiettivi prefissati. Più è alta la *Scorta di Sicurezza* dovuta alla difficoltà delle previsioni, e più si vuole perseguire una *Filosofia di Gestione* che renda il massimo servizio ai clienti, più sarà ingente la Scorta Aggiuntiva necessaria per l'ottenimento di una gestione soddisfacente.

Funzione di trasferimento

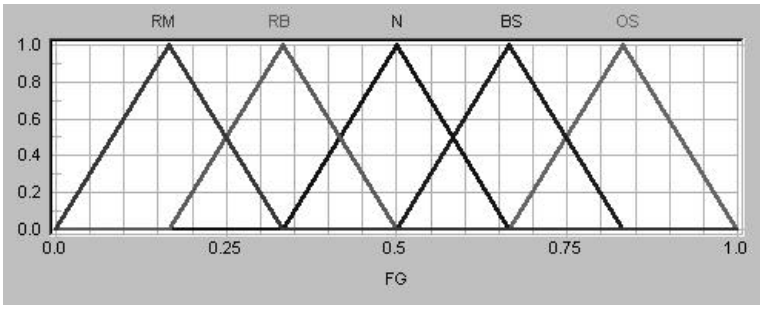


Variabile d'ingresso *SS*



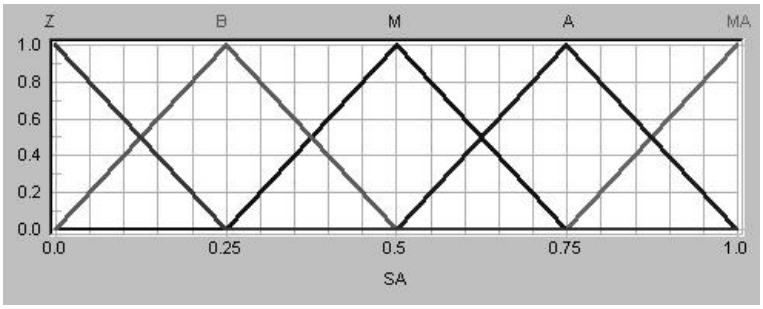
Insiemi	Punti di definizione		
Z		0,167	0,333
B	0,167	0,333	0,500
M	0,333	0,500	0,667
A	0,500	0,667	0,833
MA	0,667	0,833	

Variabile d'ingresso *FG*



Insiemi	Punti di definizione		
RM		0,167	0,333
RB	0,167	0,333	0,500
N	0,333	0,500	0,667
BS	0,500	0,667	0,833
OS	0,667	0,833	

Variabile d'uscita *SA*



Insiemi	Punti di definizione		
Z		0,000	0,250
B	0,000	0,250	0,500
M	0,250	0,500	0,750
A	0,500	0,750	1,000
MA	0,750	1,000	

Motore d'inferenza

Uscita: *SA*		*FG*				
		RM	RB	N	BS	OS
SS	Z	Z	Z	Z	Z	Z
	B	B	B	B	M	M
	M	B	B	M	A	A
	A	B	M	A	A	MA
	MA	M	A	MA	MA	MA

La particolarità di questo blocco, che ripropone per le funzioni di appartenenza degli insiemi linguistici le stesse forme usate nei due precedenti, è la capacità d'influenza maggiore assegnata alla variabile *SS* rispetto ad *FG*. Le variazioni della prima fanno cambiare infatti in modo più deciso il valore dell'uscita. In particolare, se l'incertezza è nulla la gestione diviene deterministica, e la *Filosofia di Gestione* non ha alcuna influenza sull'entità della *Scorta Aggiuntiva,* che sarà comunque zero.

4.3.4 Blocco fuzzy 4

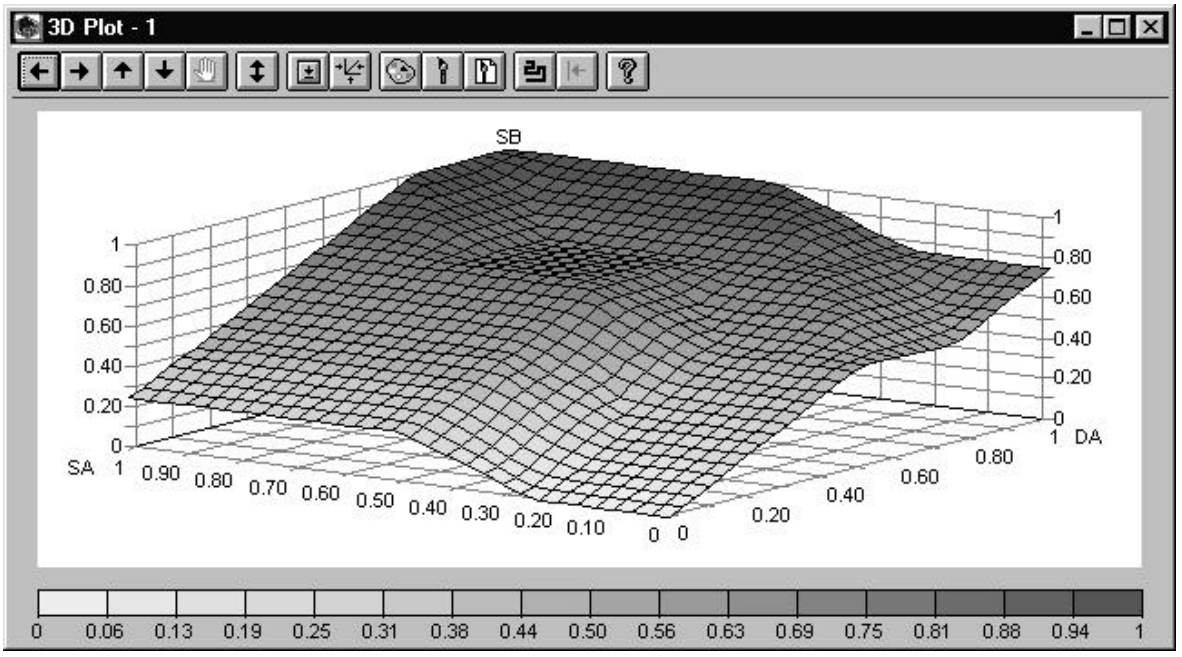
Variabili d'ingresso				Variabile d'uscita	
Domanda Attesa	*DA*	*Scorta Aggiuntiva*	*SA*	*Scorta Base*	*SB*
Molto bassa	MB	Zero	Z	Minima	MM
Bassa	B	Bassa	B	Bassa	B
Media	M	Media	M	Media	M
Alta	A	Alta	A	Alta	A
Molto Alta	MA	Molto Alta	MA	Molto Alta	MA

Fuzzificazione		Defuzzificazione
Insieme *Pi* (trapezoidale) con 5 punti di valutazione	Insieme *Pi* (trapezoidale) con 5 punti di valutazione	Operatore *Altura* (media pesata dei risultati delle regole)

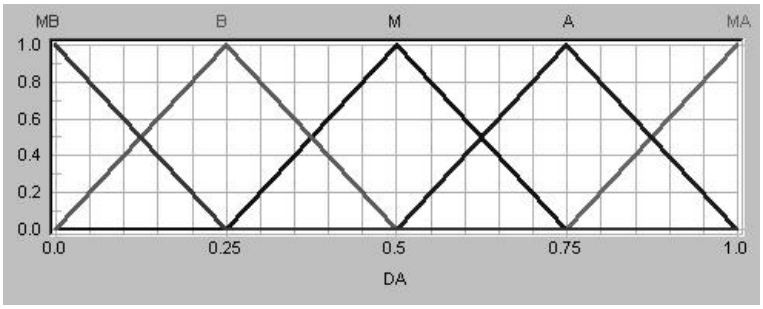
La *Domanda Attesa*, ossia la quantità che si prevede venga richiesta durante il periodo che intercorre tra due ordini successivi, ha, come si è detto, carattere aleatorio. Se essa fosse nota con precisione a priori, sarebbe sufficiente ordinare in anticipo l'esatta quantità necessaria a soddisfare le richieste, e non ci sarebbe bisogno della *Scorta Aggiuntiva.* In realtà, si è visto analizzando i blocchi precedenti che questa grandezza è fondamentale, e che può variare notevolmente in base al contesto di gestione in cui si agisce.

La somma dei contributi di queste due variabili forma la *Scorta Base*, che è la quantità a cui si deve ripristinare la scorta ad ogni successiva emissione d'ordine, a meno delle ordinazioni già in arrivo dai fornitori. Tale quantità dovrebbe coprire più esattamente possibile il fabbisogno che si verifica nell'intervallo di tempo tra due ordini consecutivi, senza che si incorra in mancanze né che si mantenga un livello residuo di giacenze eccessivo. L'avvicinamento all'uno o all'altro dei due rischi dipende dalla Filosofia di Gestione prescelta. Entrambe le variabili d'ingresso crescendo fanno aumentare la *Scorta Base*, dovendo essa tenere conto sia delle richieste attese sia dell'entità aggiuntiva di giacenze necessaria a coprire le incertezze.

Funzione di trasferimento

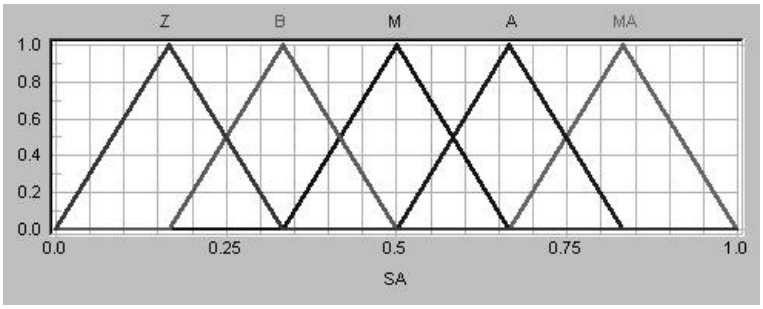


Variabile d'ingresso *DA*



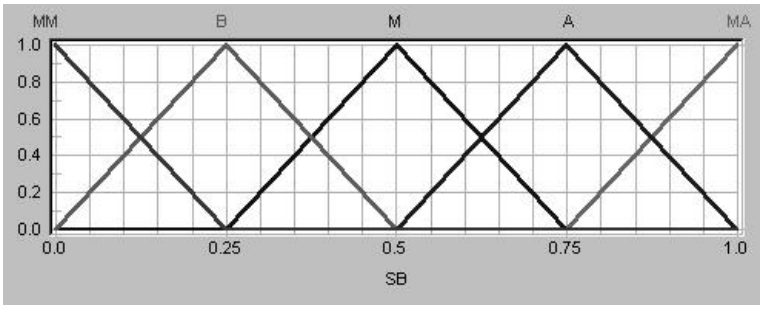
Insiemi	Punti di definizione		
MB		0,000	0,250
B	0,000	0,250	0,500
M	0,250	0,500	0,750
A	0,500	0,750	1,000
MA	0,750	1,000	

Variabile d'ingresso *SA*



Insiemi	Punti di definizione		
Z		0,167	0,333
B	0,167	0,333	0,500
M	0,333	0,500	0,667
A	0,500	0,667	0,833
MA	0,667	0,833	

Variabile d'uscita *SB*



Insiemi	Punti di definizione		
MM		0,000	0,250
B	0,000	0,250	0,500
M	0,250	0,500	0,750
A	0,500	0,750	1,000
MA	0,750	1,000	

Motore d'inferenza

Uscita: *SB*		*SA*				
		Z	B	M	A	MA
DA	MB	MM	MM	B	B	B
	B	B	B	M	M	M
	M	M	M	A	A	A
	A	M	A	A	A	MA
	MA	A	A	MA	MA	MA

Rispetto ai blocchi precedenti, cambia la forma degli insiemi per la variabile d'ingresso *Domanda Attesa*, che viene assimilata da questo punto di vista alle variabili d'uscita, al fine di distinguere più nettamente gli effetti dei valori vicini ai limiti del campo di esistenza. Da notare che il valore dell'uscita è molto più sensibile alle variazioni di *DA* che non a quelle di *SA*. In sostanza, la variabile concernente la domanda stabilisce un intervallo abbastanza ristretto di valori plausibili, all'interno del quale ci si sposta modificando *SA*.

4.3.5 Blocco fuzzy 5

Variabili d'ingresso				Variabile d'uscita	
Scorta Base	*SB*	*Ordini Fornitori*	*OF*	*Livello di Riordino*	*LR*
Minima	MM	Zero	Z	Minimo	MM
Bassa	B	Bassi	B	Basso	B
Media	M	Medi	M	Medio	M
Alta	A	Alti	A	Alto	A
Molto Alta	MA	Molto Alti	MA	Massimo	MA

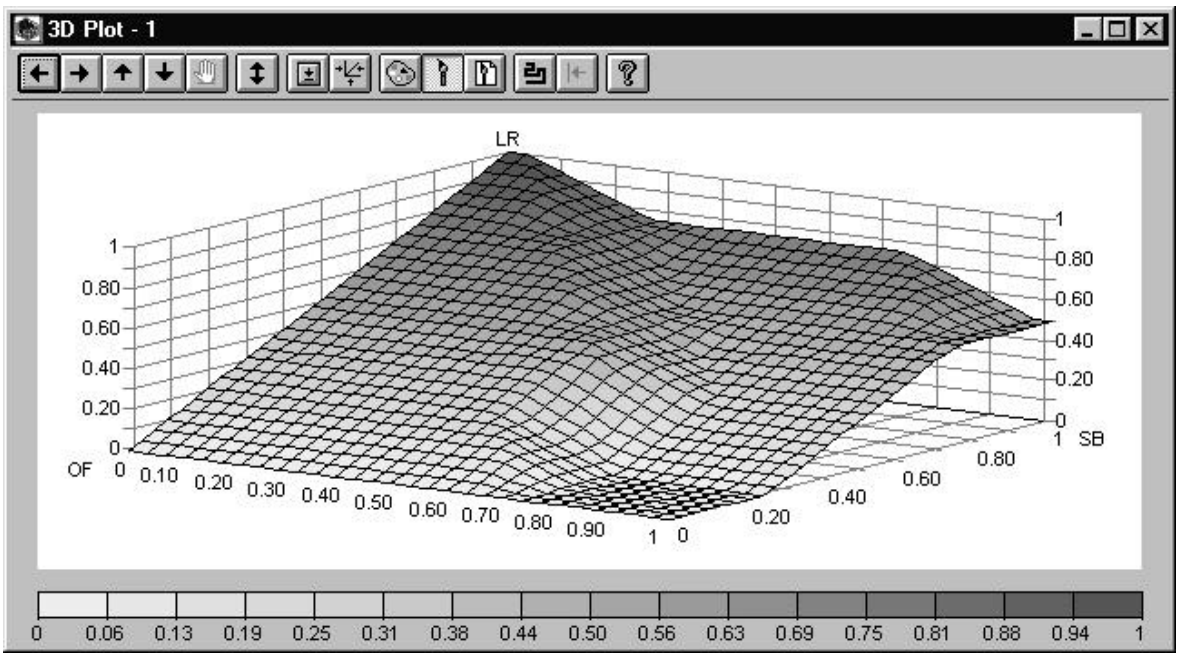
Fuzzificazione		Defuzzificazione
Insieme *Pi* (trapezoidale) con 5 punti di valutazione	Insieme *Pi* (trapezoidale) con 5 punti di valutazione	Operatore *Altura* (media pesata dei risultati delle regole)

Il *Livello di Riordino* è una correzione per difetto della scorta base, necessaria se la disponibilità prevista (comprensiva degli ordini in arrivo) è maggiore di quella reale. Non tenere conto di eventuali ordini in corso, che aumenterebbero comunque la disponibilità, potrebbe infatti causare la richiesta ai fornitori di quantità eccessive, dovute ad un errata stima del fabbisogno reale.

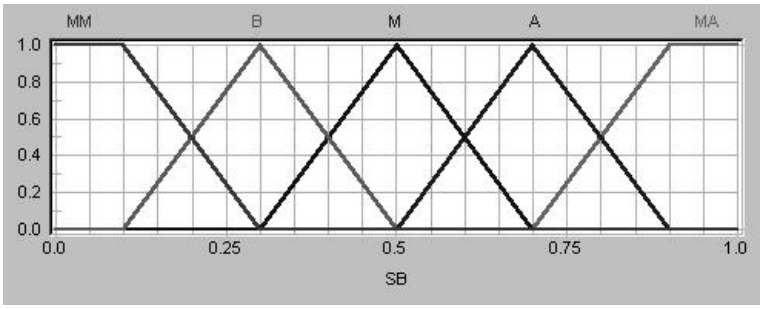
La variabile *Ordini Fornitori* è la quantità già ordinata, attesa per i periodi futuri, e serve a modificare in questo senso la *Scorta Base*. La quantità di ripristino finale *LR*, decresce quindi all'aumentare di *OF*, e cresce all'aumentare di *SB*. In particolare, se non ci sono ordini in sospeso, la quantità di ripristino iniziale e quella modificata coincidono.

È importante ricordare che la variabile *LR*, una volta defuzzificata, non viene impiegata come nuovo ingresso di un blocco fuzzy, essendo questo l'ultimo blocco della prima parte del sistema, deputata a calcolare la *Quantità Necessaria* da ordinare. Dal valore numerico di *LR* ottenuto si sottraggono algebricamente le *Disponibilità Reali*, per calcolare così il valore di *QN*. Si dovrà poi decidere se emettere effettivamente tale ordine.

Funzione di trasferimento

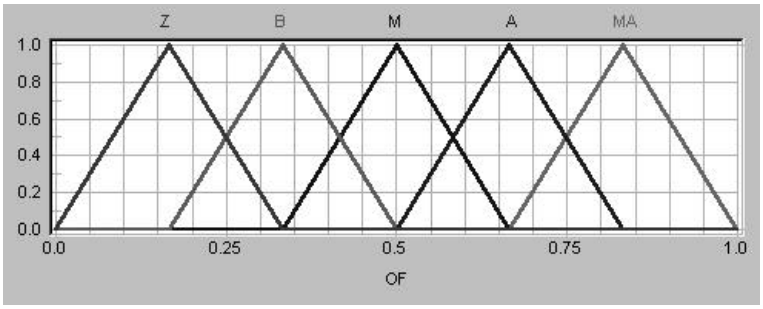


Variabile d'ingresso *SB*



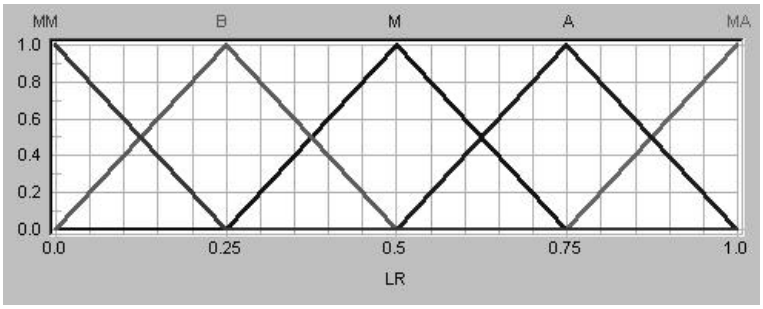
Insiemi	Punti di definizione		
MM		0,100	0,300
B	0,100	0,300	0,500
M	0,300	0,500	0,700
A	0,500	0,700	0,900
MA	0,700	0,900	

Variabile d'ingresso *OF*



Insiemi	Punti di definizione		
Z		0,167	0,333
B	0,167	0,333	0,500
M	0,333	0,500	0,667
A	0,500	0,667	0,833
MA	0,667	0,833	

Variabile d'uscita *LR*



Insiemi	Punti di definizione		
MM		0,000	0,250
B	0,000	0,250	0,500
M	0,250	0,500	0,750
A	0,500	0,750	1,000
MA	0,750	1,000	

Motore d'inferenza

Uscita: *LR*		*OF*				
		Z	B	M	A	MA
SB	MM	MM	MM	MM	MM	MM
	B	B	B	B	MM	MM
	M	M	M	M	B	B
	A	A	A	M	M	M
	MA	MA	A	A	A	A

In questo blocco risulta notevole la differenza di peso tra le due variabili d'ingresso. Le variazioni provocate dal mutare di *OF* sono infatti lievi, e comportano in sostanza delle correzioni alla quantità determinata dall'ingresso *SB*. Al fine di rendere *LR* il più possibile simile a *SB* nel caso in cui il contributo di *OF* sia nullo o molto basso, si sono opportunamente dimensionati gli insiemi di *SB*, e per la sua fuzzificazione si è impiegato un insieme trapezoidale più stretto, che aiuta ad ottenere una risposta più lineare.

4.3.6 Blocco fuzzy 6

Variabili d'ingresso				Variabile d'uscita	
Disponibilità Reali	*DR*	*Domanda Attesa*	*DA*	*Rotture Previste*	*RP*
Negative	N	Molto bassa	MB	Minime	MM
Zero	Z	Bassa	B	Basse	B
Basse	B	Media	M	Medie	M
Medie	M	Alta	A	Alte	A
Alte	A	Molto Alta	MA	Molto Alte	MA

Fuzzificazione		Defuzzificazione
Insieme *Pi* (trapezoidale) con 5 punti di valutazione	Insieme *Pi* (trapezoidale) con 5 punti di valutazione	Operatore *Altura* (media pesata dei risultati delle regole)

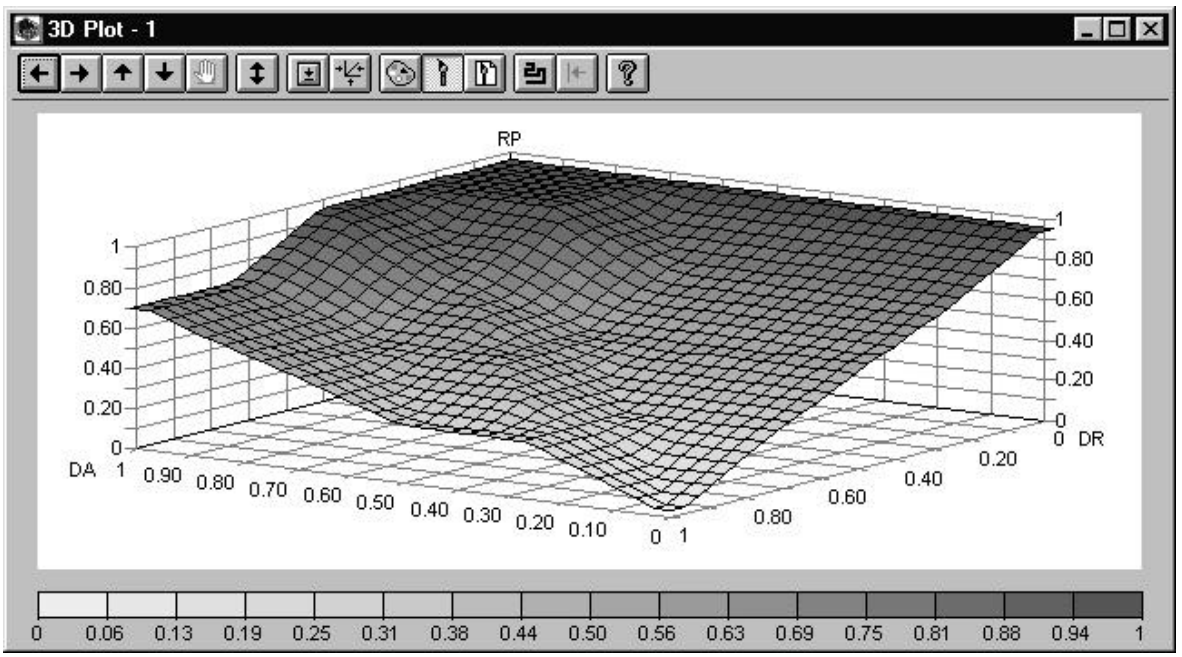
Il blocco 6 è il primo blocco della seconda parte del sistema fuzzy, quella che deve decidere o meno se emettere l'ordine la cui entità è determinata dalla variabile *QN*.

La *Domanda Attesa*, impiegata con il suo valore numerico nella sottrazione che determina dal *Livello di Riordino* la *Quantità Necessaria*, è in questo blocco usata in forma fuzzy, ai fini di stabilire le conseguenze cui potrebbe portare il non emettere un ordine nella situazione in cui ci si trova.

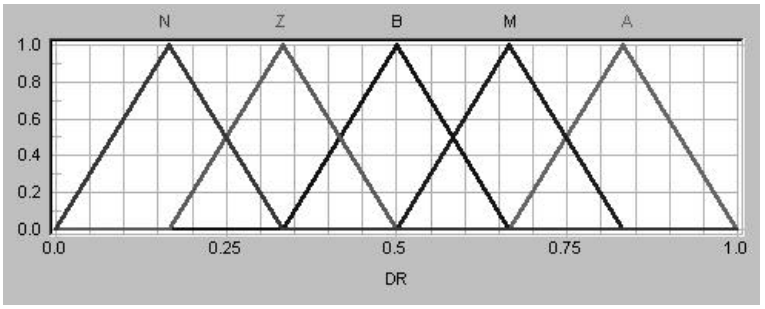
Tali conseguenze sono stimate dalla variabile *Rotture Previste*, uscita di questo blocco che, confrontando appunto la quantità di richieste previste con la *Disponibilità Reale* attuale, cerca di valutare l'entità delle rotture di scorta cui si andrebbe incontro nel caso si decidesse di non ordinare in questo periodo.

Avere un'elevata disponibilità riduce, dal punto di vista probabilistico, il numero di mancanze che si verificheranno, mentre se la *Domanda Attesa* è alta, ci si può aspettare che tali mancanze aumentino. Questi legami tra le variabili d'ingresso e d'uscita sono rappresentati nelle regole scelte per il blocco.

Funzione di trasferimento

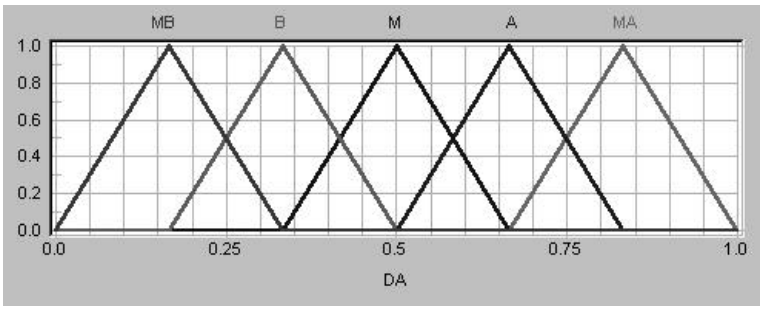


Variabile d'ingresso *DR*



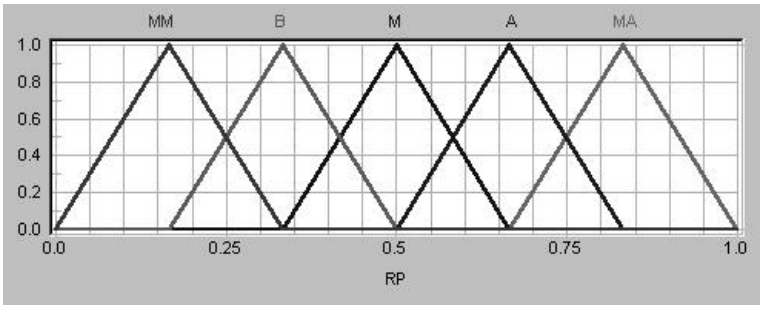
Insiemi	Punti di definizione		
N		0,167	0,333
Z	0,167	0,333	0,500
B	0,333	0,500	0,667
M	0,500	0,667	0,833
A	0,667	0,833	

Variabile d'ingresso *DA*



Insiemi	Punti di definizione		
MB		0,167	0,333
B	0,167	0,333	0,500
M	0,333	0,500	0,667
A	0,500	0,667	0,833
MA	0,667	0,833	

Variabile d'uscita *RP*



Insiemi	Punti di definizione		
MM		0,167	0,333
B	0,167	0,333	0,500
M	0,333	0,500	0,667
A	0,500	0,667	0,833
MA	0,667	0,833	

Motore d'inferenza

Uscita: *RP*		*DA*				
		MB	B	M	A	MA
DR	N	MA	MA	MA	MA	MA
	Z	A	A	A	MA	MA
	B	M	M	M	A	MA
	M	B	B	M	M	A
	A	MM	B	B	M	A

La particolarità progettuale più evidente di questo blocco sta nelle regole linguistiche. In tutti i casi che presentano *Disponibilità Reale* negativa le regole assegnano il massimo valore a *Rotture Previste*, indipendentemente dalla *Domanda Attesa*. Questa scelta è stata fatta per non correre il rischio di lasciare scoperti degli impegni già presi con i clienti. Infatti, se *DR* è negativa, significa che è stata promessa una quantità attualmente non disponibile. In tal caso, è assolutamente necessario procurarsi in tempo questa quantità.

4.3.7 Blocco fuzzy 7

Variabili d'ingresso — Variabile d'uscita

Rotture Previste	*RP*	*Filosofia di Gestione*	*FG*	*Urgenza Ordine*	*UO*
Minime	MM	Rimanenze Minime	RM	Nessuna	N
Basse	B	Rimanenze Basse	RB	Bassa	B
Medie	M	Neutra	N	Media	M
Alte	A	Buon Servizio	BS	Alta	A
Molto Alte	MA	Ottimo Servizio	OS	Massima	MS

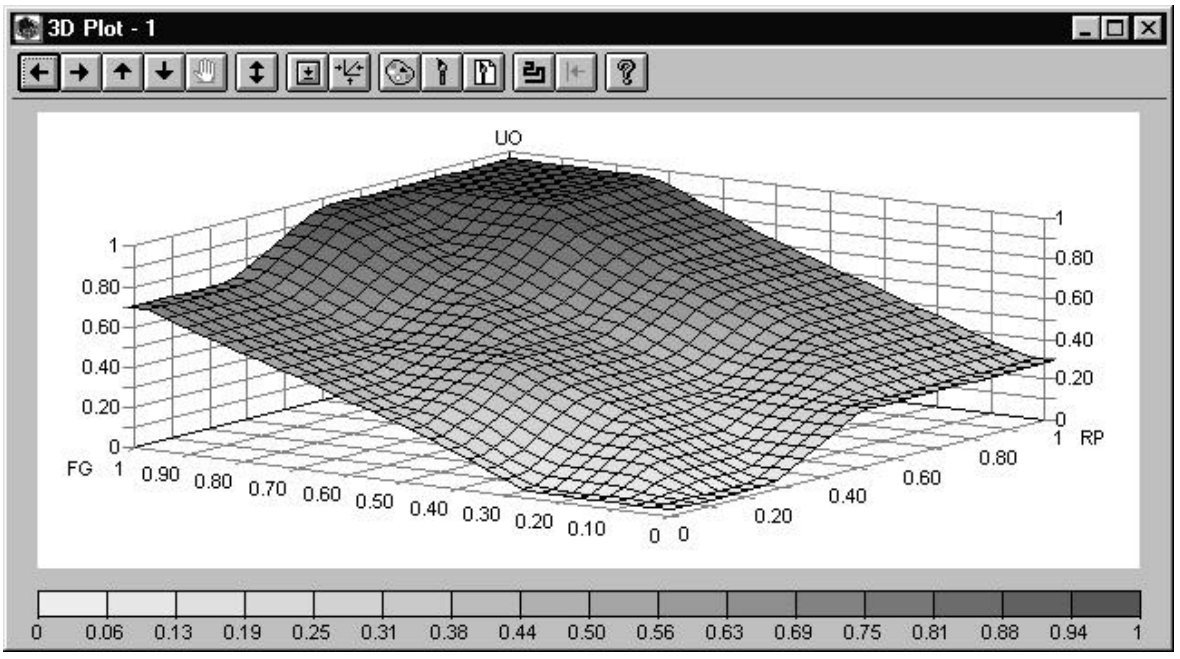
Fuzzificazione		Defuzzificazione
Insieme *Pi* (trapezoidale) con 5 punti di valutazione	Insieme *Pi* (trapezoidale) con 5 punti di valutazione	Operatore *Altura* (media pesata dei risultati delle regole)

Per decidere se è opportuno emettere un ordine, è importante quantificare nel modo migliore possibile la sua urgenza. Il presente blocco ha proprio questo compito, ed il risultato che si determina viene inserito nella variabile *Urgenza Ordine*.

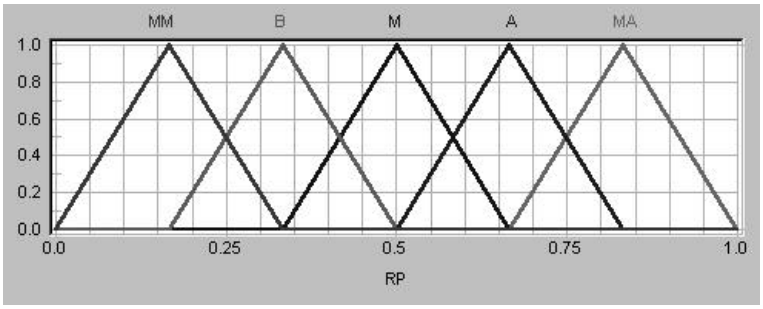
La variabile d'ingresso *RP* valuta la probabilità che si verifichino rotture di stock, mentre *FG* stabilisce in questo caso la gravità attribuita a tali eventi, visto che lo stesso numero di carenze può avere significati molto diversi a seconda degli obiettivi gestionali che si perseguono.

Un ordine si può considerare tanto più urgente quanto più sono alte le *Rotture di scorta Previste* nel caso non venga emesso, e la stessa urgenza aumenta con il desiderio di orientare la propria *Filosofia di Gestione* verso un alto livello di servizio.

Funzione di trasferimento

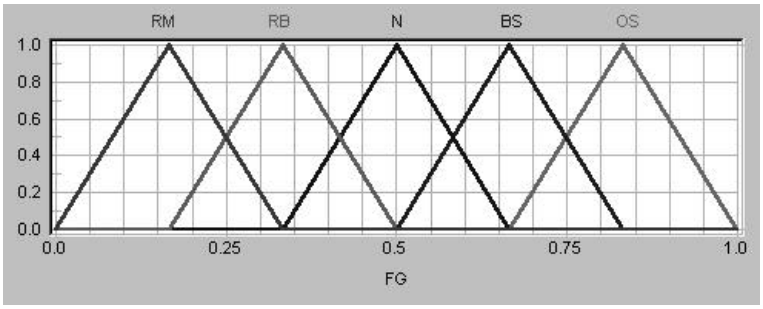


Variabile d'ingresso *RP*



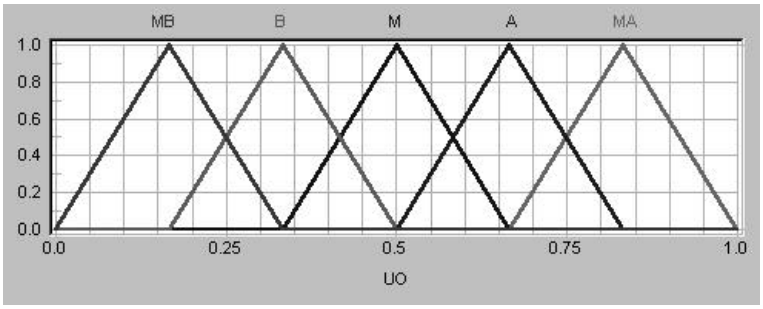
Insiemi	Punti di definizione		
MM		0,167	0,333
B	0,167	0,333	0,500
M	0,333	0,500	0,667
A	0,500	0,667	0,833
MA	0,667	0,833	

Variabile d'ingresso *FG*



Insiemi	Punti di definizione		
RM		0,167	0,333
RB	0,167	0,333	0,500
N	0,333	0,500	0,667
BS	0,500	0,667	0,833
OS	0,667	0,833	

Variabile d'uscita *UO*



Insiemi	Punti di definizione		
N		0,167	0,333
B	0,167	0,333	0,500
M	0,333	0,500	0,667
A	0,500	0,667	0,833
MS	0,667	0,833	

Motore d'inferenza

Uscita: *UO*		*FG*				
		RM	RB	N	BS	OS
RP	MM	N	N	B	B	B
	B	N	B	B	M	M
	M	B	M	M	A	A
	A	M	M	A	MS	MS
	MA	A	A	MS	MS	MS

In questo blocco e nel precedente le variabili d'uscita non hanno un significato numerico vero e proprio, che deve spaziare quindi su tutto il campo di esistenza, ma esprimono un giudizio sulla condizione contingente di gestione. Ciò rende le forme scelte per gli insiemi meno importanti ai fini di determinare i risultati corretti. Per questa ragione nel blocco presente, come del resto nel blocco 6, le funzioni di appartenenza impiegate hanno tutte la stessa semplice forma triangolare.

4.3.8 Blocco fuzzy 8

Variabili d'ingresso — Variabile d'uscita

Urgenza Ordine	*UO*	*Genere Ordine*	*GO*	*Emissione Ordine*	*EO*
Nessuna	N	Normale	N	Sì	S
Bassa	B	Poco Oneroso	PO	No	N
Media	M	Oneroso	O		
Alta	A	Molto Oneroso	MO		
Massima	MS	Impossibile	I		

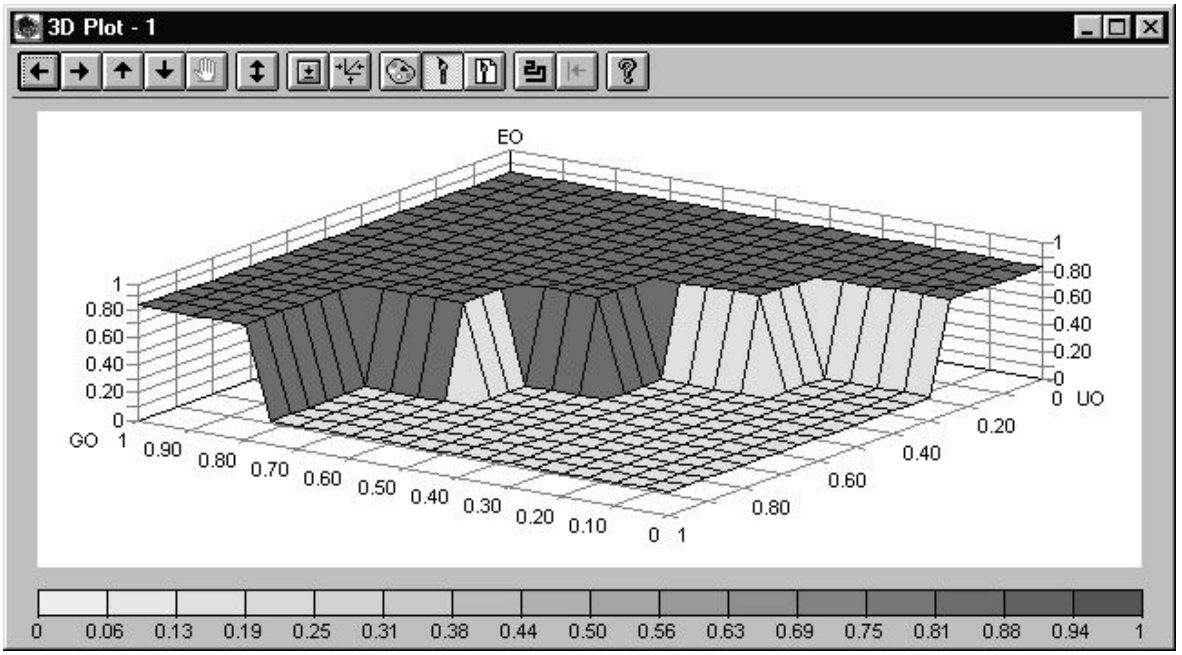
Fuzzificazione		Defuzzificazione
Insieme *Pi* (trapezoidale) con 5 punti di valutazione	Insieme *Pi* (trapezoidale) con 5 punti di valutazione	Metodo *Media dei Massimi*

Con questo blocco si decide se ordinare effettivamente la quantità *QN*. L'uscita Emissione Ordine è binaria, e per determinarla si considerano due fattori: il primo è l'Urgenza dell'Ordine calcolata nel blocco precedente, il secondo è l'onere di consegna associato all'emissione dell'ordine nel caso venga effettuata nel *periodo* attuale.

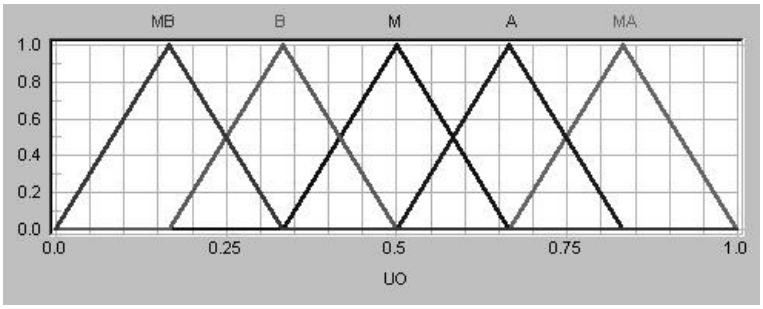
L'onere di consegna, indicato nella variabile *Genere Ordine*, dipende prima di tutto dalla distinzione tra *ordini normali* e *ordini extra*, che viene operata grazie alle variabili algebriche *primo ordine normale* e *frequenza ordini normali*. Se un ordine rientra nella programmazione regolare delle ordinazioni viene ad esso assegnata un'onerosità minima. Dall'altro lato si pongono invece gli ordini detti *impossibili*, che non possono in nessun caso essere emessi, ad esempio perché nessun fornitore è in grado di rendere disponibile al di fuori degli ordini normali la quantità richiesta.

Per quanto riguarda *EO*, è tanto più facile che l'ordine venga emesso quanto più esso è considerato urgente, e quanto meno è considerato oneroso.

Funzione di trasferimento

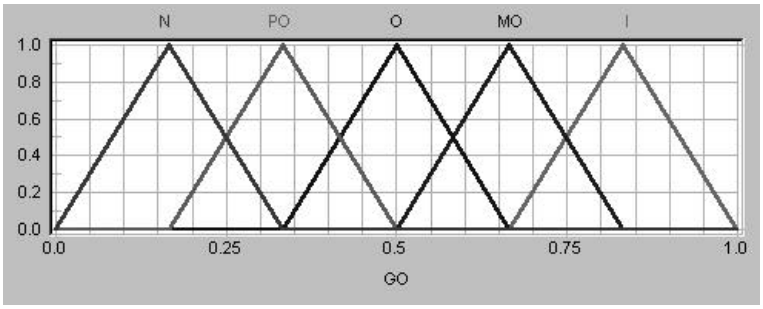


Variabile d'ingresso *UO*



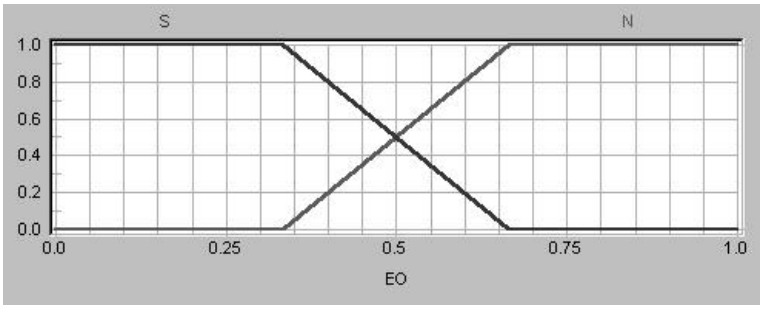
Insiemi	Punti di definizione		
N		0,167	0,333
B	0,167	0,333	0,500
M	0,333	0,500	0,667
A	0,500	0,667	0,833
MS	0,667	0,833	

Variabile d'ingresso *GO*



Insiemi	Punti di definizione		
N		0,167	0,333
PO	0,167	0,333	0,500
O	0,333	0,500	0,667
MO	0,500	0,667	0,833
I	0,667	0,833	

Variabile d'uscita *EO*



Insiemi	Punti di definizione		
S		0,333	0,667
N	0,333	0,667	

Motore d'inferenza

Uscita: *EO*		*GO*				
		N	PO	O	MO	I
UO	N	N	N	N	N	N
	B	S	N	N	N	N
	M	S	S	N	N	N
	A	S	S	S	N	N
	MS	S	S	S	S	N

L'ultimo blocco è quello che più si distingue da tutti gli altri. In esso infatti la variabile d'uscita *EO* assume solo due valori, tra i quali si sceglie in seguito ad una defuzzificazione fatta con la *media dei massimi.* In questo caso, non avrebbe infatti senso trovare una soluzione di compromesso fornita da un operatore come *Altura*. Si può verificare nel grafico della funzione di trasferimento come la defuzzificazione usata riduca al massimo le situazioni di incertezza tra il *Sì* e il *No*, grazie alla forte pendenza delle rampe di salita.

5

FUNZIONAMENTO DEL SOFTWARE E SUO UTILIZZO

5.1 STRUTTURA E FUNZIONAMENTO DEL SISTEMA SOFTWARE REALIZZATO

Il sistema di gestione scorte messo a punto è totalmente di tipo software. Ciò significa che per la realizzazione del sistema di calcolo fuzzy non sono stati impiegati microprocessori dedicati. Non essendoci esigenze di leggerezza o trasportabilità, si è scelto di implementare il tutto in modo che possa funzionare su un normale *personal computer*. L'unico requisito necessario è il funzionamento a **trentadue bit**, caratteristico dei moderni calcolatori che utilizzano il sistema operativo *Windows '95* o successivi.

I sistemi a **32 bit** sono chiamati in questo modo in contrapposizione ai precedenti sistemi a 16 bit. I valori 16 e 32 sono riferiti al numero di unità elementari d'informazione utilizzabili per trasmettere un dato o identificare un indirizzo di memoria.

5.1.1 Struttura generale

L'architettura software del sistema ha la caratteristica di essere sostanzialmente scomponibile in due parti interagenti, ma strutturalmente diverse.

I blocchi fuzzy necessari per il calcolo dei fabbisogni sono stati progettati con l'applicazione *UNFUZZY*, la quale genera il codice sorgente relativo nel linguaggio di programmazione *C++*, e in particolare con la sintassi caratteristica del *C++ Borland*. La scelta dell'applicazione con cui sviluppare l'intera sezione di calcolo è stata perciò obbligata: la versione 5.0 del *Borland C++*.

La parte prettamente gestionale del sistema, che deve essere usata dagli utenti finali, necessita di un'interfaccia grafica immediata e di semplice utilizzo. L'ambiente di programmazione che fornisce le maggiori potenzialità da questo punto di vista è senza dubbio *Visual Basic*, il quale è caratterizzato da una **programmazione *ad eventi*** molto funzionale, e garantisce la possibilità di utilizzare agevolmente delle strutture *database*.

La filosofia di **programmazione** è detta ***ad eventi***, poiché l'applicazione è costituita da procedure relativamente indipendenti, che vengono attivate in qualsiasi momento per mezzo di azioni da parte dell'utente o del programma stesso (gli *eventi* appunto). Nella classica programmazione *procedurale*, l'interazione con l'utente comporta invece uno stato di attesa da parte dell'applicazione.

Nel complesso, il sistema è perciò costituito da due programmi software separati, ma interagenti. Paragonando l'intero sistema ad un'automobile, la sezione *C++* ne costituisce il motore, che l'utente non vede, a meno che non voglia modificarne il funzionamento. Egli può agire in-

vece liberamente sui controlli d'interfaccia che gli sono messi a disposizione dalla sezione *Visual Basic*, i quali corrispondono ai comandi su cui agisce il guidatore. Il modo in cui avviene l'interazione tra guidatore e comandi determina la reazione del motore, ossia il comportamento della sezione di calcolo. Gli esiti di tale reazione si possono di nuovo osservare dal posto di guida, visto che l'applicazione d'interfaccia è pronta a mostrare all'utente i risultati finali, forniti dalla sezione di calcolo.

La comunicazione tra le due applicazioni, sia all'andata sia al ritorno, avviene tramite ***file sequenziali***, visto che i *file dati* richiedono una formattazione sostanzialmente diversa nei due ambienti di programmazione. Per ogni articolo da gestire è creato un apposito *file* di scambio.

I ***file sequenziali***, detti anche *file text*, sono archivi in cui non è prevista una particolare formattazione dei dati. Al contrario, nei cosiddetti *file dati*, le informazioni sono organizzate secondo criteri precisi.
L'impiego di *file text* permette all'utente di definire a piacere l'organizzazione da dare ai dati, mentre all'interno dei *file dati* le applicazioni stesse inseriscono tabulazioni, punti e virgola ed altre strutture di classificazione.

Si descrivono ora più in particolare le caratteristiche delle due sezioni, ed il modo in cui interagiscono.

5.1.2 L'interfaccia utente

L'applicazione che gestisce questa parte del sistema è stata sviluppata con *Visual Basic*, versione 5.0, ed è costruita per funzionare in un ambiente a finestre a trentadue bit.

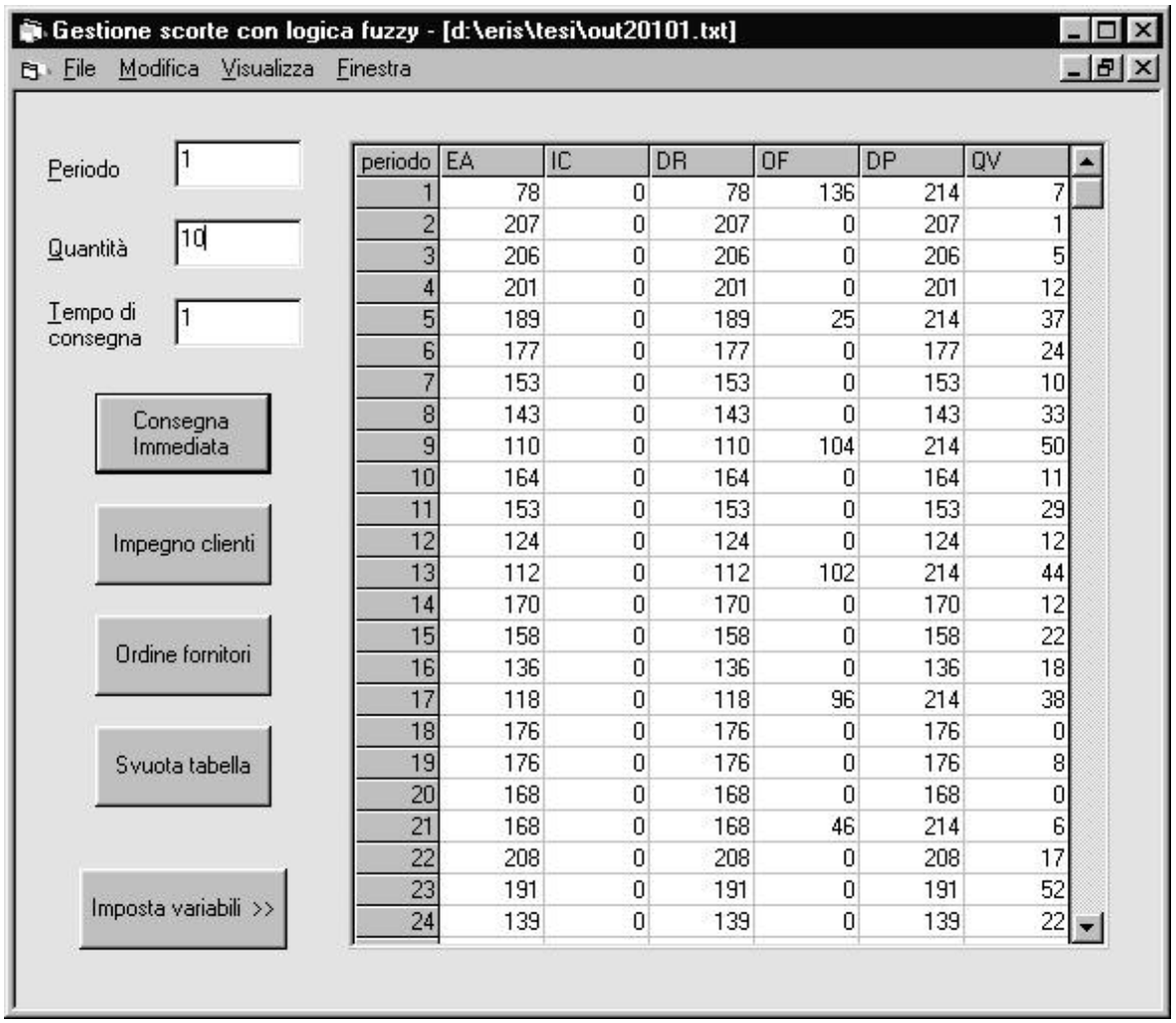
L'applicazione presenta due finestre principali. La prima è impiegata per l'osservazione e la modifica dello stato delle giacenze. La seconda per impostare i valori delle variabili di calcolo. Entrambe sono utilizzabili per la gestione di un solo articolo alla volta. Scelto l'articolo su cui lavorare, si carica l'archivio corrispondente, contenente sia le indicazioni delle giacenze, sia i valori delle variabili stabiliti per quell'articolo.

La prima finestra mostra lo stato dell'archivio, denominato *sequenziale*, relativo ad un articolo scelto, e permette di apportare delle modifiche al suo contenuto. Prima di avviare la procedura di calcolo, è importante assicurarsi che la tabella che illustra tale archivio sia aggiornata. Gli eventi che permettono di modificare lo stato dell'archivio sono gli impegni di consegna presi con i clienti, le vendite immediate e gli eventuali ordini ai fornitori. Questi ultimi nel funzionamento normale sono stabiliti autonomamente dal sistema decisionale fuzzy.

La finestra in questione, illustrata nella fig. 5.1, è visualizzata all'avvio del programma. Si esaminano di seguito i controlli di cui dispone, descrivendo nel contempo i meccanismi di funzionamento soggiacenti.

La casella di testo *Periodo* contiene l'indicazione sul periodo attuale o, a preferenza dell'utente, l'indice del periodo futuro interessato da un eventuale impegno di ven-

dita o acquisto. Il suo valore è incrementato automaticamente dopo aver eseguito un'operazione d'inserimento, in modo che sia possibile introdurre agevolmente una serie cronologica di modifiche consecutive. La casella *Quantità* serve all'utente per indicare l'entità di un movimento. L'unità di misura impiegata è il numero di articoli. La casella *Tempo di consegna* può contenere invece l'indicazione del numero di periodi necessari all'evasione di un ordine, sia d'acquisto sia di vendita.



periodo	EA	IC	DR	OF	DP	QV
1	78	0	78	136	214	7
2	207	0	207	0	207	1
3	206	0	206	0	206	5
4	201	0	201	0	201	12
5	189	0	189	25	214	37
6	177	0	177	0	177	24
7	153	0	153	0	153	10
8	143	0	143	0	143	33
9	110	0	110	104	214	50
10	164	0	164	0	164	11
11	153	0	153	0	153	29
12	124	0	124	0	124	12
13	112	0	112	102	214	44
14	170	0	170	0	170	12
15	158	0	158	0	158	22
16	136	0	136	0	136	18
17	118	0	118	96	214	38
18	176	0	176	0	176	0
19	176	0	176	0	176	8
20	168	0	168	0	168	0
21	168	0	168	46	214	6
22	208	0	208	0	208	17
23	191	0	191	0	191	52
24	139	0	139	0	139	22

Fig. 5.1 Prima finestra dell'applicazione sviluppata in *Visual Basic*, che visualizza e permette di modificare lo stato dell'archivio *sequenziale* delle giacenze di un articolo.

I pulsanti sottostanti alle caselle richiamano delle procedure. L'esito delle operazioni svolte da tali procedure è determinato dalle informazioni contenute nelle caselle. Per poter registrare un impegno preso con i clienti o un'ordinazione richiesta ai fornitori, entrambe le caselle *Quantità* e *Tempo di consegna* devono contenere dei valori numerici positivi. I tempi possono essere tutt'al più

uguali a zero, con significato di attesa nulla. In caso contrario i pulsanti *Impegno clienti* e *Ordine fornitori* risultano disabilitati. Le procedure avviate da questi due pulsanti influenzano tutti i periodi a partire da quello indicato nell'apposita casella.

L'***Impegno con clienti*** aumenta del valore indicato da *Quantità* il contenuto delle celle della colonna *IC* (*impegni clienti*) e diminuisce quelli delle colonne *EA* (*esistenza attuale*), *DR* (*disponibilità reale*) e *DP* (*disponibilità prevista*). Tuttavia, la colonna *IC* è influenzata da tale operazione solo nel periodo *p + Tempo di consegna*, con *p* istante attuale, per identificare in modo univoco il periodo in cui si dovrà effettuare la consegna. Le altre colonne sono modificate invece da quel periodo in poi, per evidenziare le ripercussioni future dell'operazione. Analogamente, la selezione del pulsante ***Ordine fornitori*** aumenta di *Quantità* i valori delle colonne *OF* (*ordini fornitori*), *EA*, *DR* e *DP*. La variabile *OF* cambia solo nei periodi di attesa dell'ordine, mentre *EA* e *DR* sono incrementate a partire dal periodo che si prevede coincidente con l'arrivo della merce.

Il pulsante ***Consegna immediata*** attiva una procedura la cui filosofia di modifica delle giacenze è diversa dalle precedenti. Tale opzione permette infatti di inserire nel periodo desiderato l'indicazione di una consegna effettuata senza preavviso, come effettivamente avviene nelle vendite cosiddette "al banco". La procedura di calcolo fuzzy dei fabbisogni agisce cronologicamente su tutte le righe dell'archivio *sequenziale*. È importante fare in modo che, al momento di calcolare il fabbisogno relativo ad un certo periodo, non siano note le richieste d'acquisto che si riceveranno nel corso di quel periodo. Per tale ragione, si è riservata, per consegne di questo tipo, una colonna apposita, denominata *QV* (*quantità venduta*). Essa contiene le indicazioni delle quantità di quell'articolo richieste nel corso dei diversi periodi, quantità che andranno poi a diminuire a consuntivo le disponibilità effettive di ogni fine periodo. Per abilitare il pulsante delle *Consegne immediate* è quindi sufficiente che sia maggiore di zero la casella *Quantità*, essendo ininfluente il tempo di consegna.

Il pulsante *Svuota* ha solo un'utilità pratica. Esso azzera tutte le caselle della tabella, ed è utile nel caso in cui si voglia effettuare sullo stesso articolo un calcolo completamente diverso.

L'ultimo pulsante di questa finestra, ossia *Imposta variabili >>*, permette all'utente di spostarsi nell'altra schermata dell'applicazione, dedicata alla visione e alla

Effetti delle operazioni di modifica del *sequenziale*

Siano p il periodo attuale, q la quantità prescelta e t il tempo di consegna; descriviamo le modifiche che le diverse operazioni introducono sulla situazione delle giacenze, indicando i diversi periodi con l'indice i.

Impegno clienti

$IC(i)=IC(i)+q$	per $p+t = i$
$EA(i)=EA(i)-q$	per $p+t \geq i$
$DR(i)=DR(i)-q$	per $p+t \geq i$
$DP(i)=DP(i)-q$	per $p+t \geq i$

Ordine fornitori

$OF(i)=OF(i)+q$	per $p \leq i < p+t$
$EA(i)=EA(i)+q$	per $p+t \leq i$
$DR(i)=DR(i)+q$	per $p+t \leq i$
$DP(i)=DP(i)+q$	per $p \leq i$

Consegna immediata

$QV(i)=q$	per $p = i$

modifica delle variabili di gestione relative all'articolo in questione.

In questa seconda finestra (fig.5.2) si possono determinare i valori di tutte le variabili di gestione impiegate nel sistema di calcolo, ad esclusione di quelle strettamente legate alla situazione temporale delle giacenze, ossia le *disponibilità reali*, gli *ordini a fornitori* e le *quantità vendute*, ricavabili dalla tabella appena descritta.

L'illustrazione 5.2 evidenzia la distinzione tra le variabili a carattere fuzzy, nella parte alta dello schermo, e quelle che abbiamo denominato "algebriche", nella parte in basso a sinistra. L'impiego di queste ultime, i cui valori sono definibili tramite le relative caselle di testo, è già stato descritto nel paragrafo 3.4.4. Vale la pena di far notare però la presenza della variabile *Lead time*, che assume un significato diverso dal *Tempo di consegna* della prima finestra. Quest'ultimo serviva esclusivamente a modificare nel modo desiderato la tabella *sequenziale* delle giacenze. La variabile *Lead time*, il cui valore è contenuto nella casella omonima, riguarda solo gli ordini a fornitori. Costituendo una caratteristica peculiare dell'articolo, essa sarà utilizzata come variabile algebrica ausiliaria nella procedura di calcolo.

Per la visualizzazione delle variabili che costituiscono gli ingressi del sistema fuzzy, è importante impiegare una rappresentazione che permetta di controllare e modificare il loro stato, evitando possibilmente di dover inserire valori numerici. In questo contesto i numeri rappresentano infatti, in modo convenzionale, il contenuto d'informazione di tipo approssimato e soggettivo presente nelle suddette variabili, i cui valori sono più correttamente di tipo linguistico. *Visual Basic* offre due valide alternative per la realizzazione di un'interfaccia grafica che sottolinei e valorizzi la caratteristica fuzzy di queste variabili: le caselle opzionali e le barre di scorrimento.

L'impiego di *caselle opzionali*, che presentano come alternative di scelta i diversi valori linguistici che una variabile può assumere, ha il privilegio della grande semplicità d'uso. È però compromessa la possibilità di selezionare valori intermedi, e risulta così ridotta la flessibilità del sistema per un utente che voglia affinare l'analisi aumentando la precisione delle impostazioni.

L'utilizzo delle *barre di scorrimento* è un po' meno immediato, ma senza dubbio più versatile, visto che permette di scegliere un qualsiasi valore appartenente alla gamma di variazione delle variabili. Si è optato quindi per questa seconda soluzione. La corrispondenza con i valori linguistici rimane evidente grazie alla legenda sovrappo-

sta, ed è ora possibile selezionare dei valori intermedi a quelli già predisposti, senza essere costretti a scegliere tra essi.

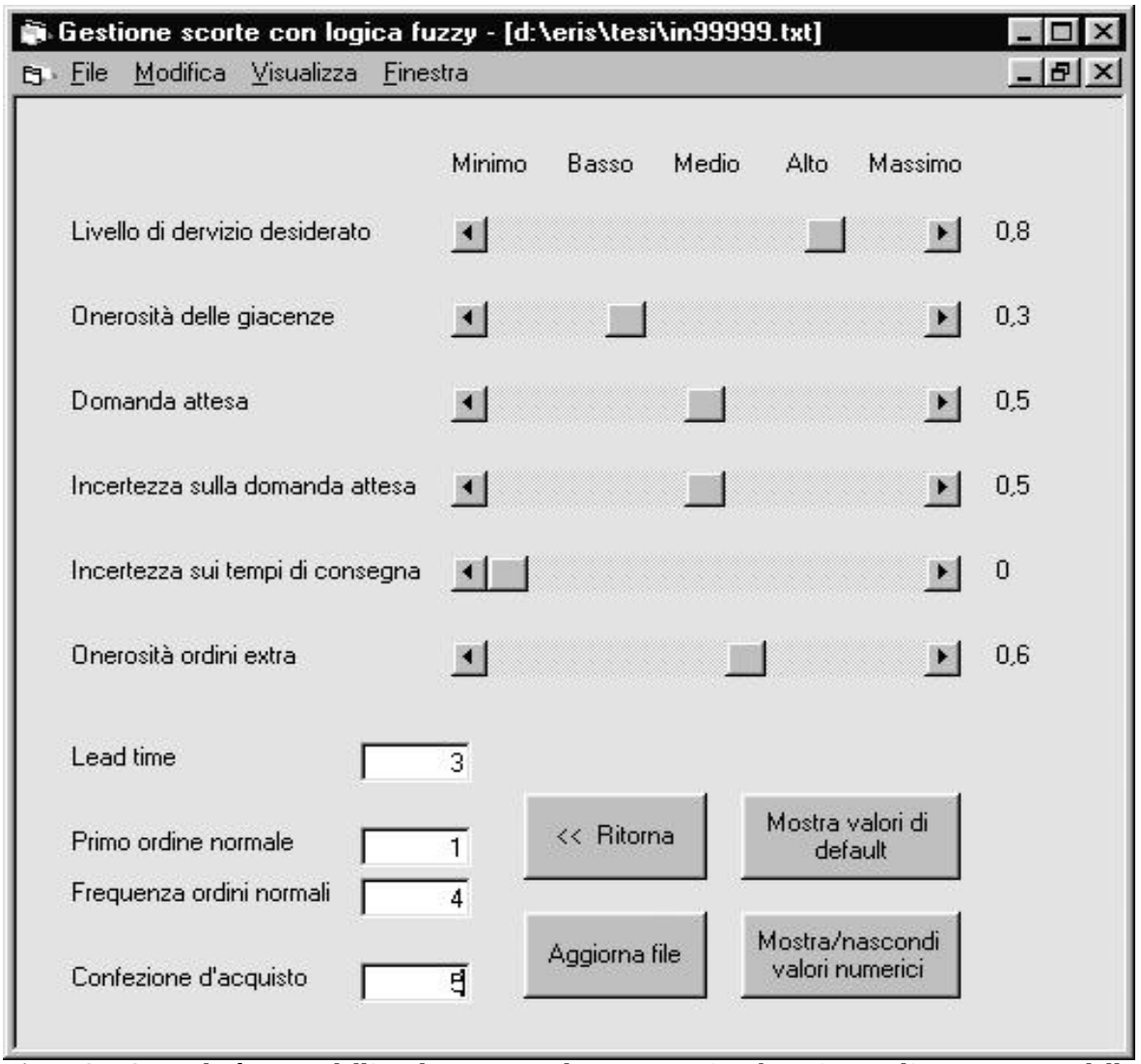


Fig. 5.2 Seconda finestra dell'applicazione sviluppata in *Visual Basic*, per l'impostazione delle variabili di gestione fuzzy e algebriche di un articolo.

Per gli utenti più esperti alla ricerca delle soluzioni ottimali, si offre l'opportunità di verificare in modo numerico il posizionamento preciso della barra di ogni variabile, indicandolo a lato con un valore compreso tra zero (*Minimo*) e uno (*Massimo*). Questa opzione non si considera tuttavia indispensabile, e potrebbe distogliere dal reale significato delle variabili impiegate, che è fuzzy e non numerico. È perciò possibile attivarla e disattivarla agendo sull'apposito pulsante in basso a destra (*Mostra/nascondi valori numerici*).

Gli altri pulsanti di questa finestra sono: << *Ritorna*, che permette di tornare alla schermata iniziale; *Aggiorna file*, che salva sull'archivio relativo all'articolo in elaborazione le modifiche apportate alle impostazioni; e *Imposta valori di default*. Per le considerazioni gestionali che hanno portato alla sua creazione, quest'ultimo pulsante merita un discorso a parte.

Molto frequentemente gli articoli gestiti in un magazzino sono raggruppati in famiglie in base a determinate caratteristiche che li accomunano. Spesso, nelle aziende che trattano grandi varietà di merci, le famiglie sono a loro volta riunite in famiglie più ampie, e così via fino a quattro o più livelli di raggruppamento. Nello sviluppo del sistema di gestione, si è voluto tenere conto di questa realtà, fermandosi comunque al primo livello di raggruppamento, visto che livelli ulteriori si possono implementare con facilità aumentando le informazioni contenute nella banca dati dell'applicazione.

Considerando che gli articoli su cui il sistema è stato impiegato appartengono tutti ad un unico gruppo, sono stati fissati dei valori che si ritiene rappresentino in modo soddisfacente le caratteristiche gestionali dell'intera famiglia di prodotti. Tali valori saranno utilizzati come **valori di *default***, attribuiti inizialmente a tutti gli articoli, ed impiegati per il calcolo nel caso non siano apportate modifiche alle variabili di gestione corrispondenti. Con il pulsante *Imposta valori di default* si dà all'utente la facoltà di ripristinare tutte le variabili ai valori proposti inizialmente per la famiglia cui appartengono.

Nel linguaggio informatico, sono chiamati **valori di *default*** quei valori che vengono attribuiti inizialmente alle variabili senza che l'utente abbia agito su di esse. Sono impiegati per evitare che alcune variabili rimangano senza valore, o per definire dei valori plausibili che vengono proposti all'utente, il quale rimane libero di modificarli o di accettarli come sono.

La validità dei valori di *default* dipende dal grado di affinità esistente tra gli articoli di una stessa famiglia. In alcuni casi potrebbe essere necessario rivedere completamente le impostazioni proposte per ogni singolo articolo. In altri casi, se le variabili da modificare, per ogni articolo della famiglia, sono solo una o due, può diventare molto vantaggioso decidere con cura i valori di default, e ridurre così notevolmente il lavoro d'impostazione dei singoli articoli.

Entrambe le finestre descritte presentano anche un *menu a tendina*, che contiene i comandi corrispondenti ai diversi pulsanti illustrati e il sottomenu *File*, per mezzo del quale si può accedere alle opzioni di gestione degli archivi relativi agli articoli. Selezionando il comando *Apri archivio* viene visualizzata la terza finestra dell'applicazione (fig.5.3), che permette di caricare in memoria e visualizzare gli archivi relativi ad ogni articolo. La voce *Apri database* provvede invece al caricamento di *file database di Access* (`*.mdb`), e permette di

visualizzare e modificare istanze d'inventario salvate in questo formato. I file di questo tipo vengono tradotti dall'applicazione in *file sequenziali* adatti ad essere utilizzati dal programma di calcolo.

Tutti i file modificati possono essere salvati, sia in formato testo che in formato *database*, e al momento della chiusura l'applicazione avverte se si sta uscendo senza salvare un archivio al quale sono state apportate delle modifiche.

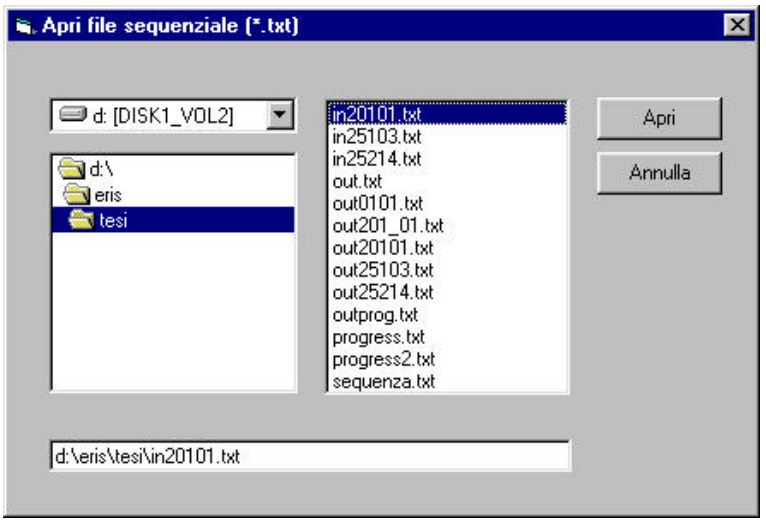


Fig. 5.3 Terza finestra dell'applicazione, che provvede al caricamento degli archivi relativi agli articoli (*sequenziali* o *database*).

Una volta salvato l'archivio contenente la sequenza delle giacenze e lo stato delle variabili per un certo articolo, si può passare alla fase di elaborazione, che è svolta dall'applicazione, complementare a quella appena descritta, sviluppata in *C++*.

Dopo aver eseguito i calcoli, è possibile caricare il file modificato, per verificare le decisioni prese dal sistema fuzzy.

5.1.3 La sezione di calcolo

Questa sezione comprende sia il calcolo fuzzy dei fabbisogni, con il conseguente processo decisionale automatico, sia il relativo aggiornamento degli archivi, nei quali viene modificato lo stato delle giacenze in base alle decisioni prese dal sistema.

La **programmazione *ad oggetti*** permette al progettista di definire le proprie strutture dati e le relative funzioni che agiscono su di esse. Ciò rende i programmi più chiari e flessibili, e permette di condividere agevolmente con altri progetti le strutture sviluppate.

In *C++* s'impiega la **programmazione *ad oggetti***, che migliora la leggibilità e la versatilità delle applicazioni. Nel nostro caso, ciò ha permesso di costruire liberamente il programma partendo dalla struttura dati dell'applicazione *UNFUZZY*, inserita nei due *file* sorgente *C++* chiamati `fuzzy.cpp` e `fuzzy.hpp`, forniti con l'applicazione stessa. La struttura è completata dal codice

generato appositamente per ogni blocco di calcolo, che può essere inserito direttamente nella propria applicazione.

L'interazione tra i diversi *file* sarà descritta in maniera più specifica al momento d'illustrare le caratteristiche dell'applicazione *UNFUZZY*. Ciò che interessa in questa sede è il fatto che, sfruttando le funzioni messe a disposizione con il codice sorgente fornito, si è realizzata una procedura di calcolo capace di eseguire un intero ciclo di elaborazione fuzzy. Essa riceve come parametri i valori delle otto variabili in ingresso, esegue fuzzificazione, inferenza e defuzzificazione dei diversi blocchi e restituisce i valori delle due variabili d'uscita.

L'applicazione sviluppata in *C++* non doveva particolari requisiti per quanto riguarda l'interfaccia, dovendo essere impiegata come motore dell'intera struttura. Si è quindi deciso di realizzare, essendo indispensabile nella fase di progettazione, una semplice interfaccia testuale *MS-DOS*, utile per visualizzare in modo immediato i risultati ottenuti con le elaborazioni.

Questa scelta ha permesso di implementare un programma principale di tipo *procedurale*, tipico di un linguaggio di programmazione strutturato, che richiama in maniera sequenziale le diverse funzioni di calcolo e di modifica degli archivi.

La prima operazione che il programma esegue è la lettura dell'archivio relativo all'articolo selezionato, cui segue la prima impostazione delle variabili d'ingresso. Alcune di esse sono uguali per tutti i periodi di calcolo, altre cambiano in ogni periodo, come le *disponibilità reali* e il *genere dell'ordine*, calcolato in base al valore delle variabili *primo ordine normale* e *frequenza ordini normali*. Per le variabili che provengono direttamente dalla tabella delle giacenze è necessaria inoltre la pre-elaborazione. Con questa operazione si rapportano i valori effettivi ai valori massimi prefissati per quell'articolo, ottenendo così delle variabili normalizzate, aventi come insieme di esistenza l'intervallo [0,1].

Fatto questo, si passa ad eseguire il calcolo fuzzy dei fabbisogni, per decidere in quali periodi emettere gli ordini e di che entità. Il calcolo è eseguito in modo sequenziale su ogni periodo della tabella delle giacenze, a partire dal primo che risulta modificabile in base al *lead time* impostato. Nell'elaborazione sono comprese fuzzificazione, inferenza e defuzzificazione, seguite da una procedura di post-elaborazione delle due variabili finali.

Nel caso il sistema fuzzy decida per l'emissione di un ordine, si deve modificare di conseguenza lo stato dell'archivio *sequenziale* delle giacenze. Ogni ordine emesso

modifica infatti lo stato delle righe precedenti e successive, secondo i meccanismi illustrati nel paragrafo 5.1.2 (v. effetti del pulsante *Ordini fornitori*). È necessario, in questi casi, provvedere, oltre che alla modifica dell'archivio, ad una nuova impostazione delle variabili concernenti le giacenze, in modo che le variabili riferite ai periodi futuri risultino aggiornate al momento del calcolo.

Alla fine del processo di calcolo, si dispone di una versione modificata dell'archivio iniziale dell'articolo, contenente la sequenza degli ordini ritenuti opportuni per far fronte alle esigenze gestionali del magazzino. L'ultima operazione è perciò il salvataggio delle modifiche apportate all'archivio, il quale può a questo punto essere caricato dall'applicazione d'interfaccia per essere mostrato all'utente.

5.2 INFORMAZIONI DI GESTIONE COME CONOSCENZA PER IL SISTEMA

Tutte le variabili impiegate nel sistema, sia in modo algebrico sia come grandezze fuzzy, derivano da una serie d'informazioni e conoscenze relative al processo di gestione che si vuole automatizzare. È quindi importante stabilire in base a quali criteri tali variabili vadano impostate, e come l'esperienza acquisita debba tradursi in competenza nel sistema di elaborazione.

La maggior parte della conoscenza relativa alla gestione di scorte, se non altro nella sua forma più generale, andava inserita nel progetto di base sviluppato in questo lavoro. Tuttavia, il modo in cui l'utente interagisce con il sistema è senza dubbio determinante sul tenore dei risultati che esso può fornire. In sostanza, compito fondamentale per il progettista è stato quello di rendere il sistema flessibile ed adattabile, oltre che di semplice utilizzo. Spetta invece all'utente trasferire al sistema la sua conoscenza dello specifico problema di gestione e la sua esperienza al riguardo, in modo che il comportamento del processo automatico si adatti alle caratteristiche specifiche del caso.

Il significato gestionale delle variabili impiegate è stato ampiamente trattato nei capitoli precedenti. In questa sede interessa stabilire in che modo utilizzare al meglio i dati già disponibili. È importante soprattutto distinguere le informazioni trasferibili al sistema già durante la progettazione da quelle strettamente legate alla fase di esecuzione, ossia all'impiego che l'utente fa del sistema stesso. Per tornare alla metafora dell'automobile, le informazioni del primo tipo corrispondono alla messa a punto del motore, quelle del secondo tipo alla guida del veicolo.

5.2.1 Informazioni trasferite con la progettazione

Nella fase di progettazione si può definire il funzionamento generale di tutto il sistema, comprensivo della sezione di elaborazione fuzzy e della sezione di aggiornamento dell'inventario. Tuttavia, anche la struttura di base non può prescindere dal caso gestionale che si vuole affrontare.

Le operazioni di pre-elaborazione e post-elaborazione richiedono la conoscenza del campo di variazione effettivo delle variabili cui sono applicate. In riferimento ad uno specifico articolo, non è ragionevole richiedere ad ogni esecuzione l'inserimento di questa informazione, che ha senso modificare solo nel lungo termine, in seguito ad una variazione consolidata della realtà di gestione. È invece opportuno disporre di una banca dati in cui inserire le informazioni di lungo termine, come i limiti degli universi di esistenza delle variabili.

Le variabili fuzzy interessate dalle procedure di pre-elaborazione e post-elaborazione sono quelle concernenti lo stato delle giacenze, ossia gli ingressi *disponibilità reale* e *ordini fornitori* e l'uscita *livello di riordino*. Gli insiemi di definizione di queste variabili sono strettamente legati tra loro, e si è perciò ritenuto opportuno registrare tutte le informazioni in una sola variabile aggiuntiva. Da essa si ricavano, per mezzo di calcoli molto semplici, i valori minimi e massimi di tali insiemi. Questa variabile, che d'ora in poi sarà chiamata k, è stata calcolata in base alla storia recente della gestione di ogni articolo. Al fine di migliorare il più possibile le prestazioni di gestione, si è però pensato di fare riferimento non tanto alla storia delle giacenze, che deriva da una metodologia di conduzione che si vuole sostituire, quanto alla storia dei fabbisogni, ossia delle richieste avute nel passato.

Si deve comunque ricordare che il metodo sviluppato non è statistico, e che l'accuratezza del dato scelto è un fattore marginale per le prestazioni del sistema. Importante è soprattutto garantire allo stesso la massima flessibilità, visto che la precisione si dovrà ottenere nella fase di esecuzione agendo sulle variabili fuzzy. Nella scelta del dato più rappresentativo si è quindi privilegiata la semplicità di calcolo, e l'ampiezza del raggio d'azione che questo forniva al sistema. La media μ e la deviazione standard s delle vendite registrate nell'ultimo anno sono gli indici più facili e immediati da determinare. Il calcolo è stato fatto prendendo come tempo base l'intervallo tra un ordine *normale* e il successivo. Sulla base di diverse prove sperimentali, si è scelto di porre $k = 3 \cdot (\mu + s)$. Tale valore è impiegato direttamente come limite superiore per la varia-

Definizione degli insiemi di esistenza

$K = 3 \cdot (\mu + s)$
$LR \in [0,k]$
$OF \in [0,k/2]$
$DR \in [-k/4,k/2]$

con μ e s calcolate sulle vendite tra due ordini *normali* successivi nell'ultimo anno.

bile *livello di riordino* (*LR*), il cui limite inferiore è zero. La metà di k è invece il limite superiore di *disponibilità reale* (*DR*) e di *ordini fornitori* (*OF*), aventi rispettivamente come limiti inferiori $-k/4$ e zero (solo *DR* può essere negativa). Ricordiamo ancora che tutti questi valori sono stati scelti in seguito a prove empiriche, e che il comportamento del sistema non è determinato in modo sostanziale da essi. Semplicemente agendo sulla variabile *domanda attesa* si può infatti modificare drasticamente il reale campo di variazione di ogni variabile durante la fase di esecuzione. Ciò che più conta è che i valori massimi siano sufficientemente alti, tali che il sistema sappia adattarsi a crescite anche notevoli delle quantità movimentate. I valori scelti danno, da questo punto di vista, un ampio margine di sicurezza. Nel complesso, questo metodo comporta la memorizzazione nella banca dati di un solo valore numerico per ogni articolo trattato.

Il secondo tipo d'informazioni da immagazzinare deriva dal fatto che gli articoli sono spesso divisi in famiglie, e che si può identificare, in riferimento ad ogni famiglia di articoli trattata, un valore di default per ognuna delle variabili di gestione considerate. In questo caso si tratta di immagazzinare per ogni gruppo di articoli (e non più per ogni singolo articolo) i valori delle sei variabili fuzzy e delle quattro variabili algebriche considerati validi, almeno in prima approssimazione, per le variabili di quel tipo. Per stabilire tali valori è opportuno che il progettista sia coadiuvato dall'utente, o comunque da qualcuno che sia meglio a conoscenza delle problematiche di gestione peculiari della realtà in cui sarà fatto funzionare il sistema.

5.2.2 La fase di esecuzione: criteri d'impostazione delle variabili e loro validità temporale

I dati che l'utente deve fornire al sistema nella fase di esecuzione sono fondamentalmente di due tipi. La prima serie d'informazioni riguarda la sequenza dello stato delle giacenze nella successione dei periodi d'interesse. Il secondo blocco d'informazioni concerne i valori attribuiti alle diverse variabili di gestione riferite all'articolo da trattare.

Come si è visto nel paragrafo 5.1, queste informazioni sono stabilite dall'utente, e comunicate al sistema mediante le due finestre dell'applicazione sviluppata con *Visual Basic*. In particolare, essendoci per le variabili di gestione i suddetti valori di default, l'utente è libero di modificarli o no, scegliendo al limite di non cambiare per nulla la configurazione proposta.

Le variabili hanno tuttavia caratteristiche diverse per quanto riguarda la loro variabilità temporale, e si ritiene opportuno distinguerle in base ai loro tempi indicativi di validità, per capire quali di esse sarà il caso di tenere sott'occhio più frequentemente. Nella trattazione che segue, oltre a stabilire indicativamente i tempi di modifica più adatti ad ogni variabile, ne forniamo anche i criteri generali d'impostazione.

Variabili fuzzy

Definiamo anzitutto il significato dei termini che useremo per indicare la validità temporale delle variabili. Una variabile è di *breve periodo* se può facilmente cambiare da un periodo di calcolo al successivo, di *medio periodo* se ci si può aspettare che cambi solo dopo un numero di ordini abbastanza elevato (per ordini mensili potrebbe essere un intervallo di revisione di circa sei mesi), di *lungo periodo* se cambia solo in casi particolari (ad esempio in seguito ad una modifica sostanziale del piano di produzione). Si è scelto di rappresentare la validità temporale delle variabili su una scala di tipo fuzzy (fig. 5.4), in modo da non dover forzare le stesse a rientrare solo in una o nell'altra categoria.

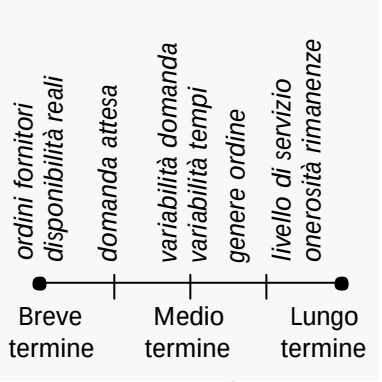


Fig. 5.4 Validità temporale delle variabili fuzzy.

Prendendo in considerazione dapprima lo stato delle giacenze, esso si modifica continuamente da un periodo all'altro. Le variabili associate, ossia *ordini fornitori* e *disponibilità reali*, si possono considerare senza dubbio di breve termine, dovendo essere sempre aggiornate alla situazione attuale. In ogni caso, l'aggiornamento non spetta all'utente, ma al processo automatico di registrazione dei movimenti d'inventario. Più articolato è il discorso per tutti gli altri fattori.

La variabile *livello di servizio* permette di scegliere la visione ritenuta più adatta nei confronti delle rotture di *stock*. Il giudizio sull'atteggiamento da tenere nella situazione attuale è soggettivo, ed è importante fare delle prove preliminari per definire cosa significa per il sistema un servizio medio, buono oppure ottimo. Nel complesso, questa variabile non dovrebbe cambiare sostanzialmente di peso in breve tempo, visto che la strategia di gestione è di solito un fattore stabile per un'azienda. Il *livello di servizio* si può perciò classificare come variabile a medio-lungo termine (più vicina al lungo che al medio), da rivedere nel caso in cui si decida di modificare i propri obiettivi.

Lo stesso discorso si può fare per l'*onerosità delle rimanenze*. Oltre alle considerazioni strategiche, c'è da tenere conto del fatto che il tipo di merce stabilisce in modo vincolante il tempo ammissibile di giacenza. Con merci deperibili, difficilmente si può tollerare che rimangano

spesso delle quantità invendute. Anche in questo caso il valore specifico è attribuito con un giudizio qualitativo di medio-lungo termine, di validità probabilmente anche più lunga rispetto alla variabile precedente.

Senza dubbio più frequenti saranno le modifiche richieste dalla variabile *domanda attesa*. Esse possono essere dovute a variazioni nella tendenza delle richieste, ma anche a fluttuazioni periodiche che non sono da trascurare. Tale variabile è quindi classificabile a medio-breve termine (per ordini mensili, potrebbe essere utile rivederla anche ad ogni ordine), e per impostare il suo valore si può integrare un'analisi dei dati storici di vendita con previsioni personali riguardanti la situazione futura.

La *variabilità dei tempi di consegna* e la *variabilità della domanda* possono essere entrambe considerate variabili di medio termine. Dipendendo rispettivamente dal comportamento dei fornitori e dalla dinamica delle interazioni di mercato, si ritiene possano rivelarsi più stabili della *domanda attesa*, ma andranno comunque riviste più spesso delle variabili strategiche. Il giudizio su di esse non può essere solo soggettivo: delle semplici statistiche sulla storia recente della gestione daranno delle indicazioni utili su possibili tendenze di queste variabili. L'impostazione può avvenire estrapolando tali tendenze, senza però trascurare eventuali percezioni e conoscenze che un esperto può avere sulla questione (ad esempio, la previsione di un periodo di difficoltà per un certo fornitore).

Il *genere dell'ordine* è impostato in base alla distinzione tra ordini *normali* ed *extra* e all'onerosità attribuita a questo secondo tipo di ordini. La classificazione dell'ordine in questione è senz'altro una decisione di breve termine, ma è eseguita automaticamente grazie all'impiego delle variabili *primo ordine normale* e *frequenza ordini normali*. Ciò che qui è chiamato *genere ordine* è quindi l'*onerosità degli ordini extra* impostata nella finestra delle variabili del programma di gestione. Questa variabile si può posizionare un po' oltre il medio termine, essendo legata a questioni di movimentazioni fisiche e di accordi con i fornitori che difficilmente possono cambiare molto spesso.

Variabili algebriche

Abbiamo già citato le variabili *primo ordine normale* e *frequenza ordini normali.* La prima si può considerare come variabile ausiliaria, che aiuta il programma di gestione, ma senza avere in realtà un vero e proprio significato gestionale. In una versione più automatizzata del progetto essa potrebbe essere tranquillamente eliminata. La seconda fa invece parte della strategia di gestione, ed è in-

fluenzata in modo determinante dalla disponibilità dei fornitori. Si può perciò classificare come variabile di medio-lungo termine.

Lo stesso discorso vale anche per la variabile *lead time,* da ritoccare solo in seguito a mutate condizioni di consegna. Si potrebbero però verificare situazioni con richieste di consegna particolari in certi periodi, e il medio termine potrebbe quindi essere più adeguato a questa variabile.

Ultima da considerare è la variabile *confezione*, che indica la quantità minima indivisibile di consegna. Pure in questo caso, si possono apportare delle modifiche al suo valore solo previo accordo con i fornitori. Anche *confezione* sarà quindi una variabile di medio termine, un po' più spostata verso il lungo rispetto alla precedente *lead time.*

Una rappresentazione su scala fuzzy della validità temporale delle variabili algebriche, del tutto analoga a quella impiegata per le variabili di gestione fuzzy, è esposta nella fig. 5.5.

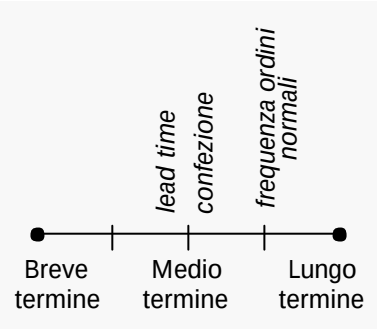


Fig. 5.5 Validità temporale delle variabili algebriche

6

MESSA A PUNTO DEL SISTEMA E RISULTATI OTTENUTI

6.1 FASE DI MESSA A PUNTO

Il processo di messa a punto, o di revisione, riveste per un sistema fuzzy un'importanza basilare. Se i blocchi di calcolo sono molti, come nel nostro caso, è difficile prevedere con precisione in anticipo quale potrà essere il comportamento del sistema nelle diverse situazioni. Per questa ragione, è fondamentale che il sistema sia provato nel modo più esaustivo possibile, in maniera da ridurre la probabilità che, in seguito, si trovi ad affrontare un caso particolare che non sa gestire.

Per ottenere il perfezionamento del progetto, è stata necessaria una lunga sequenza di prove e modifiche, che si è considerata conclusa solo quando i risultati sono apparsi soddisfacenti.

Tutto il processo di revisione si può dividere in due fasi successive, aventi caratteristiche diverse, che saranno chiamate fasi di *messa a punto qualitativa* e *quantitativa.*

6.1.1 Messa a punto qualitativa

Per eseguire al meglio questa fase, si devono avere bene in mente gli scopi primari della gestione e le caratteristiche principali delle variabili e del loro modo di interagire. Le operazioni di rettifica del sistema sono possibili solo dopo aver confrontato i risultati ottenuti dalle diverse prove con quelli attesi, e compreso le motivazioni che hanno condotto a tali risultati.

Per comprendere a fondo le ragioni del comportamento del sistema nei diversi casi, è utile osservare il contributo specifico dato dai diversi blocchi di calcolo in ogni esecuzione. Ciò permette di circoscrivere le cause delle anomalie nel funzionamento, ed intervenire con le dovute correzioni in modo mirato e preciso. A tal fine, si è sviluppata una versione modificata del programma di calcolo specifica per la fase di test. In essa si eseguono le singole elaborazioni con valori diversi delle variabili di gestione, e per ogni elaborazione, oltre ai risultati finali, si possono con-

trollare le risposte parziali di tutti i blocchi fuzzy. In questo modo, si può registrare su un solo archivio l'esito di molte prove con caratteristiche differenti, ed identificare il blocco o l'insieme di blocchi responsabili di eventuali anomalie.

Si descrivonoo ora sinteticamente il tipo di prove eseguite, e le modifiche più importanti che ne sono derivate.

Per prima cosa, si è verificata la condotta del sistema nei confronti delle singole variabili. Mantenendo costanti tutti gli altri fattori, le variabili sono state fatte mutare, una alla volta o tutt'al più a coppie, all'interno dell'intero campo di esistenza, per verificare l'andamento della risposta del sistema rispetto a tali variazioni. Il processo è stato ripetuto per parecchi valori diversi delle variabili usate come parametri, al fine di aumentare la significatività dei risultati ottenuti. Il procedimento usato ha permesso di scovare e correggere le prime imperfezioni progettuali. L'esito più significativo di questo stadio di revisione è stata la modifica di diverse regole linguistiche.

Si è poi verificato che le uscite raggiungessero, in corrispondenza a determinati valori degli ingressi, i limiti estremi previsti per esse. Un problema in particolare era importante risolvere. Se la *domanda attesa* e la *variabilità* ad essa associata sono nulle, il sistema deve emettere ordini solo nel caso sia necessario soddisfare impegni di consegna presi con clienti. In tutti gli altri casi il valore della variabile d'uscita *quantità necessaria* deve essere zero. Ciò significa che la variabile *livello di riordino* in queste situazioni deve assumere il suo valore minimo, cioè zero. Partendo da insiemi linguistici di forma e disposizione standard e dal metodo di defuzzificazione di *baricentro*, le modifiche necessarie per ottenere l'azzeramento del *livello di riordino* sono state diverse.

L'operatore di defuzzificazione a *baricentro* è stato sostituito con l'**operatore *Altura***. Tale operatore, se associato alle giuste funzioni d'appartenenza, permette di raggiungere i valori estremi delle variabili d'uscita.

L'**operatore *Altura*** esegue la media pesata dei risultati forniti dalle varie regole d'inferenza attivate. In alcuni casi produce risultati molto simili all'operatore di baricentro, ma in certe condizioni il differente metodo di calcolo genera effetti sostanzialmente diversi.

La seconda modifica importante ha riguardato la forma delle funzioni d'appartenenza dei *term set* relativi alle uscite. Si è deciso di alterare gli insiemi ai margini dell'intervallo di esistenza delle variabili d'uscita dei blocchi. A tali insiemi è stata assegnata una forma a triangolo rettangolo, invece che a triangolo isoscele. In tal modo, si sono traslati verso gli estremi i centri di gravità degli insiemi minimi e massimi. Nella fig. 6.1 è possibile osservare graficamente l'effetto di tale modifica. La combinazione delle due modifiche ha permesso di risolvere il

problema dell'annullamento del *livello di riordino* in assenza di vendite non prestabilite.

Il metodo trovato per ampliare gli effettivi intervalli di variazione delle variabili si è poi rivelato utile in diversi casi. È stato infatti impiegato in diversi blocchi fuzzy, a volte con modifiche ulteriori nella forma delle funzioni d'appartenenza.

Nell'ambito della fase di messa a punto qualitativa si è verificato anche il corretto funzionamento delle variabili algebriche. Di minor peso sono state le modifiche conseguenti a questo tipo di controllo, anche se valori della variabile *lead time* superiori ad un periodo richiedono degli aggiustamenti agli insiemi di definizione. Questi ultimi devono essere leggermente ampliati all'aumentare dei tempi di consegna (operazione peraltro resa automatica nel programma di calcolo).

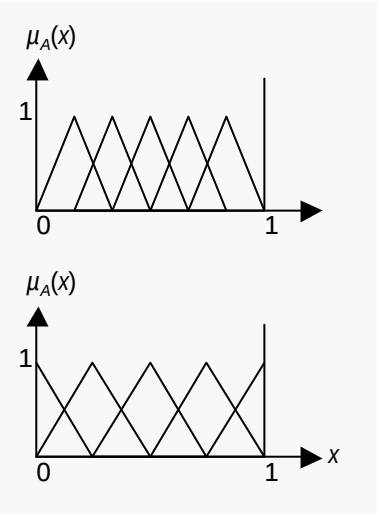


Fig. 6.1 *Term set* delle variabili d'uscita prima (in alto) e dopo le modifiche apportate nella fase di revisione qualitativa.

6.1.2 Messa a punto quantitativa

Questo stadio della progettazione ha richiesto l'utilizzo di dati di gestione reali, sui quali verificare la bontà del comportamento del sistema.

La conoscenza dei dati relativi ai fabbisogni di un magazzino effettivamente funzionante permette di affinare lo studio dell'influenza che le diverse variabili hanno sul comportamento del sistema. I dati sugli ordini emessi dal sistema di gestione attualmente applicato offrono l'opportunità di fare un raffronto sulle prestazioni dei due metodi, e di dare quindi un giudizio oggettivo sull'applicabilità effettiva del sistema progettato. Il raffronto delle prestazioni ottenute, e delle diverse tendenze di gestione che si ottengono cambiando le variabili del sistema, sono descritti nei prossimi paragrafi.

Le modifiche apportate al sistema in questa fase hanno riguardato ancora le regole linguistiche di alcuni blocchi e i *term set* delle variabili *domanda attesa* e la *scorta base*. Considerata la loro importanza, si è fatto in modo che queste variabili assumessero un ruolo preminente nei blocchi per i quali fungono da ingressi. Una serie di prove, durante le quali sono stati cambiati i valori di tutte le altre variabili, hanno permesso di determinare le forme migliori da attribuire alle funzioni d'appartenenza delle variabili suddette. Per la *scorta base* si è resa necessaria anche una modifica dell'insieme di fuzzificazione, al quale è stata attribuita una forma più compatta che garantisce una risposta più lineare.

È da segnalare anche una modifica importante provata in questa fase, ma abbandonata perché non ha dato i risul-

tati voluti. Tale modifica riguardava la sostituzione del blocco lineare, che determina *QN* a partire da *LR* e *DR*, con un blocco fuzzy. Con questa sostituzione si voleva ridurre da tre a due il numero d'intervalli di esistenza da definire. Non sarebbe più stata post-processata *LR*, ma *QN*, cui si sarebbe potuto assegnare verosimilmente lo stesso campo di variazione di *OF*. In questo modo, un solo campo di variazione sarebbe stato sufficiente per entrambe le variabili. Tuttavia, la risposta del blocco fuzzy non è risultata pienamente lineare, e venivano introdotte delle piccole oscillazioni che si è ritenuto opportuno evitare, tornando alla soluzione di partenza.

L'utilizzo di sequenze di dati complete e reali ha richiesto anche l'aggiornamento del programma d'interfaccia e di quello di calcolo. Il primo è stato reso più flessibile e più attinente ad una gestione reale. Il secondo è stato adattato alle più specifiche esigenze di verifica da parte del progettista, ed è stato completato con l'aggiunta della banca dati relativa alla famiglia di articoli trattata. Per entrambi è stato infine perfezionato il meccanismo di comunicazione: ogni articolo ha ora il proprio archivio, leggibile da entrambe le applicazioni.

L'archivio di scambio definitivo è costituito da due sezioni. Nella prima sezione ogni riga contiene il valore di una variabile. La prima riga contiene l'indicazione del numero di righe da elaborare (una per ogni periodo d'interesse). Questo dato serve anche per comunicare alle applicazioni quali sono le dimensioni del *file* che stanno leggendo. Le quattro righe successive contengono i valori delle variabili algebriche, nell'ordine: *lead time*, *primo ordine regolare*, *frequenza ordini regolari*, *confezione*. Le ultime sei righe della prima sezione contengono i valori delle variabili fuzzy per quell'articolo: *livello di servizio*, *onerosità giacenze*, *variabilità tempi*, *variabilità domanda*, *domanda attesa* e *genere ordine*. La seconda sezione contiene, in ognuna delle sue righe, lo stato delle giacenze nel periodo corrispondente, rappresentato dalle variabili *EA*, *IC*, *DR*, *OF*, *DP* e *QV*, che possono assumere valori diversi per ognuno dei periodi di calcolo.

6.2 PROVENIENZA E SELEZIONE DEI DATI

Per saggiare la validità del sistema, si è deciso di impiegare dati inerenti la gestione di prodotti finiti. Quasi sempre, la complessità di gestione associata ad essi è maggiore rispetto a quella relativa ai semilavorati e alle materie prime. Nel caso dei prodotti finiti è più alto il livello d'imprevedibilità, e testare il sistema nel caso di

massima severità garantisce una più agevole estensione verso le situazioni meno problematiche.

I dati reali impiegati per convalidare il sistema di gestione progettato provengono da **Porsche Italia**, importatore unico ufficiale per il nostro Paese della casa automobilistica tedesca **Porsche AG**. In particolare, le informazioni utilizzate riguardano la gestione di una famiglia di ricambi Porsche, relativi al motore di alcuni modelli di autovetture.

La **Porsche Italia** ha sede e magazzino centrale a Padova. È stata fondata nel 1985, e gestisce attualmente più di 80000 articoli diversi. La rete italiana di concessionari Porsche autorizzati (26) costituisce l'insieme dei clienti cui sono fatte le forniture. I concessionari sono collegati a loro volta con le 15 officine autorizzate.
La casa automobilistica **Porsche AG** ha sede a Stoccarda (D). Ha compiuto nel 1998 i 50 anni di attività. Nel 1997 ha contato sull'apporto di 7900 collaboratori, ed ha visto uscire dai suoi stabilimenti 32390 autovetture, tutte di tipo sportivo.

Si è potuto disporre d'informazioni dettagliate riguardanti i movimenti di magazzino della suddetta famiglia di articoli, i progressivi d'inventario e l'**anagrafica degli articoli**. Questo archivio contiene una serie d'informazioni specifiche sugli articoli, sul loro utilizzo e sulle modalità di gestione ad essi associate. Ad esempio, dall'anagrafica degli articoli sono state ricavate le informazioni riguardanti le confezioni d'acquisto, i costi e le descrizioni dei vari articoli.

Dai progressivi di magazzino della famiglia di articoli (298 codici in tutto) sono stati estratti i dati delle vendite mensili negli ultimi dodici mesi. In base a tali informazioni si sono scelti gli articoli più richiesti, sui quali effettuare le prove di funzionamento del sistema. Lo svolgimento di tali prove richiede la conoscenza delle vendite e degli ordini giornalieri. Dall'archivio dei movimenti sono state ricavate quindi, per tali articoli, le informazioni riguardanti tutte le variazioni delle giacenze verificatesi negli ultimi sei mesi (i dati disponibili cominciavano da agosto).

A questo punto, si doveva fissare il periodo di revisione delle giacenze, e quindi di calcolo dei fabbisogni. Gli ordini *normali* degli articoli trattati hanno frequenza mensile, e la settimana è sembrata perciò il lasso di tempo più adatto, visto che anche il *lead time* difficilmente è inferiore ai sette giorni.

I dati giornalieri sono stati perciò raggruppati in settimane. Ciò che interessava erano le *quantità vendute*, che sono servite come dati d'ingresso per le elaborazioni, e gli *ordini fornitori*, impiegati come termine di confronto, per valutare la bontà della sequenza di ordinazioni decise dal sistema.

La lettura dei *file*, la selezione dei campi utili (gli archivi impiegati possiedono tra i quaranta e i settanta campi), la sistemazione dei dati in modo da poterli elaborare e la successiva analisi dei risultati sono state eseguite con l'applicazione *Excel*.

6.3 ANALISI DEI RISULTATI

Il sistema fornisce, come risultato finale di un'elaborazione, la sequenza degli ordini che ha deciso di emettere nel lasso di tempo previsto per la sperimentazione. Nel nostro caso la verifica è stata eseguita su dati

consuntivi, cosa che per l'analisi dei risultati non costituisce un problema. Basta infatti avere l'accortezza di utilizzare correttamente i dati di cui si dispone, in modo che l'elaborazione non sia influenzata da informazioni cronologicamente posteriori all'istante di calcolo, che rappresenta virtualmente il momento attuale.

Diamo di seguito alcune indicazioni per la corretta interpretazione dei risultati.

Per quanto riguarda la registrazione delle giacenze si è scelta la convenzione d'inizio periodo. I dati relativi ad un periodo si riferiscono all'inizio dello stesso, eccetto quelli relativi alle vendite, che sono distribuite lungo tutto l'arco della settimana. Le giacenze di fine periodo sono date quindi dalle giacenze d'inizio periodo ridotte delle vendite settimanali.

Per quanto riguarda i calcoli, è come se fossero eseguiti durante il fine settimana. Con essi si determinano gli ordini fornitori all'inizio del periodo entrante. Anche le quantità ordinate arrivano ad inizio periodo, dopo un intervallo di periodi dato dal *lead time*. Le giacenze iniziali di un periodo equivalgono perciò alle giacenze finali del periodo precedente, aumentate della quantità in arrivo all'inizio di quel periodo. Se ci sono delle consegne da fare dovute ad impegni presi in precedenza con i clienti, esse diminuiscono le giacenze d'inizio periodo.

Calcolo delle giacenze del periodo *i*-esimo

Giacenze iniziali EA_i

$EA_i(i)=EA_f(i\text{-}1)+OF(i\text{-}leadtime)\text{-}IC(i)$

Giacenze finali EA_f

$EA_f(i)=EA_i(i)\ \text{-}QV(i)$

Nel caso più semplice di *lead time* uguale ad uno e impegni con clienti nulli, per ottenere l'*esistenza finale* di un periodo è sufficiente sommare all'*esistenza finale* del precedente gli *ordini fornitori* dello stesso, e sottrarre le *quantità vendute*.

Le modalità di vendita gestite dal sistema sono due: gli impegni di consegna e le consegne immediate. I primi sono stabiliti in base ad accordi con i clienti, e per essi la quantità da fornire è nota con un certo anticipo (almeno un periodo). Le consegne immediate si verificano invece nel corso di ogni periodo, senza che si possa conoscere preventivamente la loro entità. Queste due tipologie di vendita devono essere trattate diversamente dal sistema di gestione. Nell'applicare i dati a disposizione al processo d'elaborazione si sono perciò seguiti due filoni di test specifici, per provare la risposta separatamente nei due casi. Nelle prove conclusive, le due modalità di vendita sono riunite, in modo che le elaborazioni tengano conto contemporaneamente di entrambe. Ciò serve a convalidare in modo completo la procedura di gestione, che deve saper adattare il proprio funzionamento alla più grande varietà di situazioni possibile.

6.3.1 Prove eseguite con consegne immediate nulle

Nel caso in cui le vendite siano associate esclusivamente ad impegni di consegna presi con i clienti, le quantità vendute sotto forma di consegne immediate devono essere poste a zero. In questo modo, si può verificare se il sistema riesce, senza mantenere giacenze superflue, a soddisfare i clienti che hanno esplicitamente richiesto determinate quantità degli articoli desiderati.

Per un'elaborazione algebrica, conoscendo in anticipo l'entità dei fabbisogni, questo non è un compito particolarmente impegnativo. Si è però preferito gestire anche le eccezioni all'interno dell'elaborazione fuzzy, per rendere il sistema più omogeneo e robusto. In particolare, due problemi di funzionamento legati a questa modalità di vendita hanno reso necessario apportare alcune modifiche alla base di conoscenza del sistema. Tali modifiche si sono poi rese utili anche in condizioni di funzionamento diverse.

Il primo problema da affrontare era il già citato annullamento degli ordini nel caso di fabbisogno nullo. Esso è stato risolto ponendo a zero la *domanda attesa* e le variabili di stima dell'incertezza, e ritoccando i blocchi fuzzy, facendo in modo che gli ordini riguardassero esclusivamente le quantità richieste in anticipo dai clienti.

Il secondo problema derivava dal fatto che la sezione deputata a decidere l'*emissione dell'ordine* non distingue le due modalità di vendita. Essa stabilisce se ordinare o no solo in base all'urgenza di un ordine e alla sua onerosità. È sorto quindi il problema di far risaltare l'urgenza di un ordine necessario a soddisfare un impegno preso. Non volendo forzare il sistema fuzzy con l'aggiunta di eccezioni gestite in modo deterministico, la soluzione migliore è sembrata quella di modificare opportunamente le regole del blocco fuzzy 6. In esso si stabilisce, in base alla *disponibilità reale* (*DR*) e alla *domanda attesa* (*DA*), il numero di *rotture di scorta previste* (*RP*). Si è fatto in modo che, nel caso *DR* sia negativa, *RP* assuma il suo valore massimo indipendentemente dal valore di *DA*. L'*urgenza dell'ordine*, decisa nel successivo blocco 7, risulta così essere molto alta, cosa che rende quasi certa l'emissione dell'ordine stesso.

6.3.2 Prove eseguite con impegni di consegna nulli

Nella sezione precedente di prove, le verifiche di funzionamento e le eventuali correzioni erano soprattutto di carattere qualitativo. Procedendo con l'elaborazione dei dati concernenti le consegne immediate, si devono ora anticipare i fabbisogni, non più seguirli. L'introduzione del

nuovo elemento d'aleatorietà rende le prove di questa fase più difficili da giudicare. Visto che i problemi sono più quantitativi che qualitativi, lo si è fatto comparando le medie delle giacenze e controllando il verificarsi di eventuali rotture di scorte.

Le prove di questa sezione sono le più severe per il sistema, poiché in questa fase si verificano effettivamente l'importanza e l'utilità che possono avere le variabili. Nella trattazione che segue si cercherà di offrire una gamma rappresentativa di tali prove, anche se i casi di gestione, e le conseguenti configurazioni di funzionamento, possono essere infiniti.

Comparazione tra dati reali e risultati ottenuti

Non si espongono qui i risultati di tutte le elaborazioni fatte, essendo lo scopo della sperimentazione quello di verificare la validità del progetto di gestione sviluppato, non quello di risolvere un problema gestionale di una specifica realtà aziendale. Presentiamo quindi le informazioni relative alle elaborazioni delle giacenze di tre articoli, che saranno chiamati A, B e C, tutti appartenenti alla famiglia di ricambi 207. Tra gli altri, si è scelto di mostrare le elaborazioni fatte su di essi soprattutto perché presentano movimenti più frequenti rispetto ad altri. Ciò consente di aumentare la significatività delle prove. I componenti A e B sono di poco valore, ma presentano un grande movimento e quindi alte giacenze medie. L'articolo C ha smercio inferiore, ma un valore decisamente più alto.

I dati specifici, estratti dall'archivio *anagrafica di magazzino*, si possono osservare in tab. 6.1. Il *Costo* indicato nella quarta colonna è il prezzo pagato da Porsche Italia per avere a disposizione il componente, ed è perciò in base ad esso che è calcolato il valore totale delle giacenze. La quinta colonna è calcolata su dati estratti dall'archivio dei movimenti, e mostra la media μ_G delle giacenze (in numero di pezzi) nel corso degli ultimi sei mesi. La sesta colonna mostra il valore medio delle giacenze in questo periodo (V_G), calcolato moltiplicando i valori delle due colonne precedenti. In questo caso e nei successivi, le giacenze medie sono calcolate considerando le vendite regolarmente distribuite durante la settimana.

Tab.6.1 Dati degli articoli sulle cui giacenze si sono eseguite le elaborazioni fuzzy.

Art.	Codice	Descrizione	Costo (Lit)	μ_G (pz)	V_G (Lit)
A	99320720101	Filtro	12.540	364,0	4.564.560
B	99320725802	Vite	11.435	70,6	807.310
C	96420725214	Tubazione olio	252.564	14,2	3.586.410

Nelle fig. 6.2, 6.3, 6.4 si possono osservare i risultati di tre elaborazioni, una per ogni articolo descritto. In ogni figura, la situazione delle giacenze che si sarebbe ottenuta impiegando quella particolare elaborazione (grafico più scuro) è comparata alla situazione realmente verificatasi nel magazzino di cui si possiedono i dati.

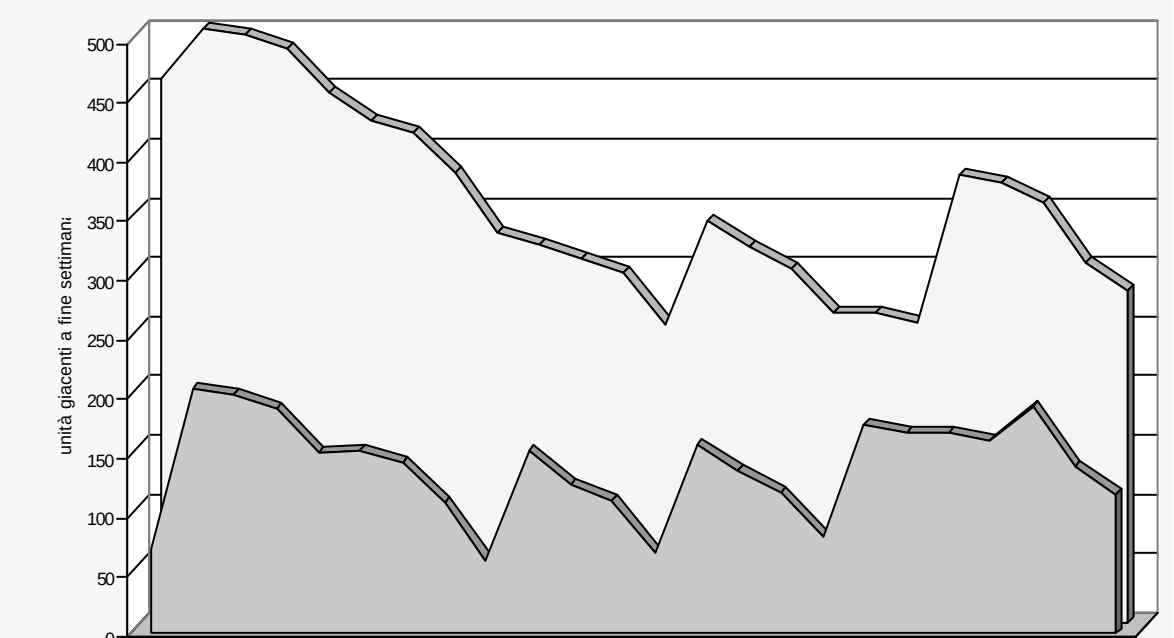
Le elaborazioni sono state eseguite impostando le variabili fuzzy ai valori di *default*, ritenuti adatti per questa famiglia di articoli. I parametri usati per i tre articoli sono quindi gli stessi. L'unica eccezione è l'*onerosità delle giacenze* dell'articolo C, nettamente più costoso degli altri. Per default l'*onerosità delle giacenze* è *medio bassa* (valore numerico 0,4), nel caso dell'articolo C si è impostata un'*onerosità alta* (valore numerico 0,8).

Utilizzando nelle tre prove gli stessi valori per le variabili fuzzy, non si ottengono probabilmente i risultati ottimali. Questi sono raggiungibili modificando attentamente per ogni articolo alcune variabili di gestione. Tuttavia, condurre le prove con questo metodo permette di verificare l'autonomia decisionale del sistema. Infatti, in questo modo è richiesta, salvo alcune eccezioni, una sola impostazione iniziale per tutta la famiglia di articoli, e l'impegno richiesto all'utente è perciò molto limitato.

Parametri di *default* scelti per la famiglia 207

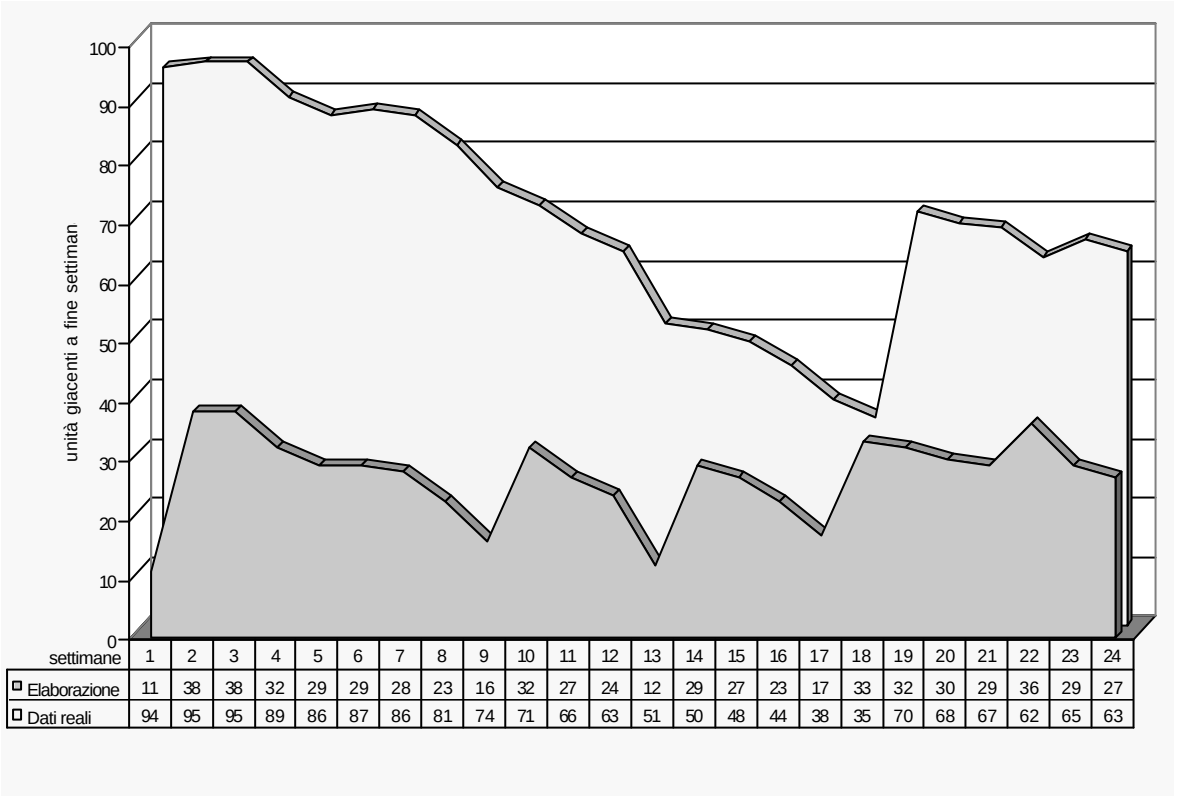
Livello di servizio desiderato:
alto (valore numerico 0,8)
Onerosità giacenze:
medio bassa (0,4)
Domanda attesa:
medio alta (0,6)
Incertezza domanda attesa:
medio bassa (0,4)
Incertezza tempi consegna:
bassa (0,3)
Onerosità ordini extra:
alta (0,7)
Lead time = 1
Primo ordine normale = 1
Frequenza ordini normali = 4
Confezione = 1

I valori mostrati sono quelli impiegati per le elaborazioni, con eccezione dell'*onerosità giacenze* dell'articolo C (0,8).



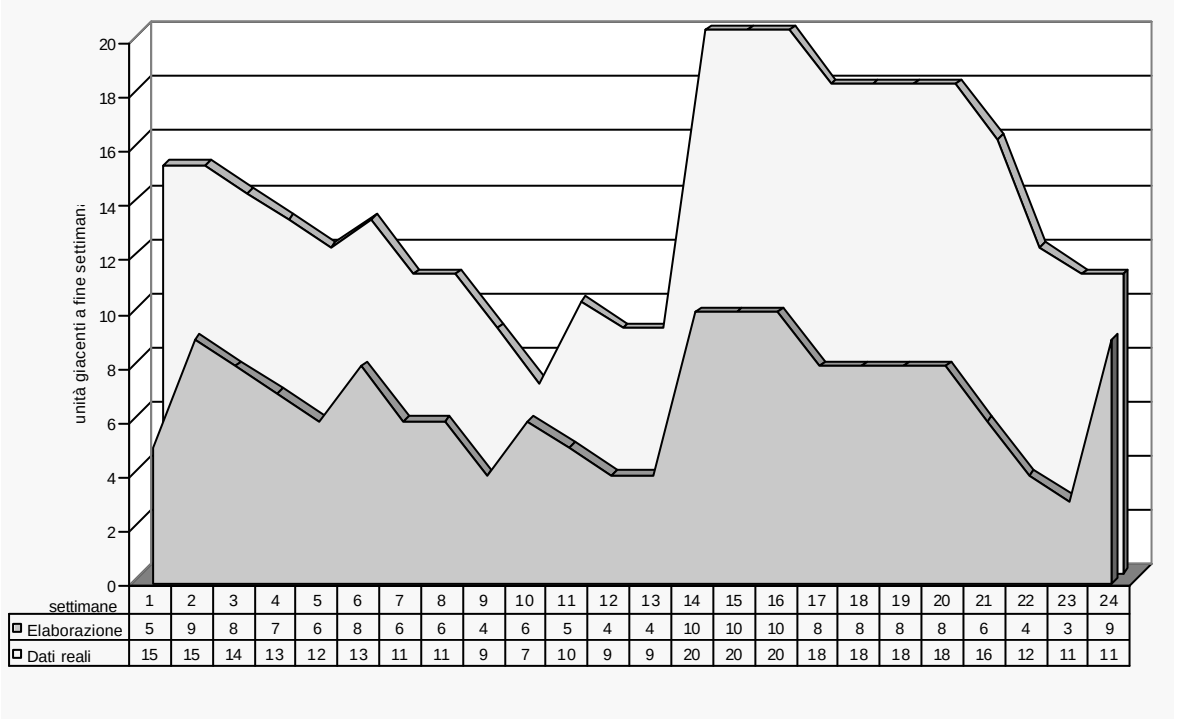
settimane	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24
Elaborazione	71	207	202	190	153	154	144	111	61	154	125	113	69	159	137	119	81	177	169	169	163	192	140	118
Dati reali	458	500	495	483	446	422	412	379	329	318	306	294	250	338	316	298	260	260	252	377	371	354	302	280

Fig. 6.2 Articolo A - Giacenze di fine settimana nelle ultime 24 settimane, confronto tra i dati reali e quelli ottenuti con l'elaborazione fuzzy (grafico più scuro).



settimane	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24
Elaborazione	11	38	38	32	29	29	28	23	16	32	27	24	12	29	27	23	17	33	32	30	29	36	29	27
Dati reali	94	95	95	89	86	87	86	81	74	71	66	63	51	50	48	44	38	35	70	68	67	62	65	63

Fig. 6.3 Articolo B - Giacenze di fine settimana nelle ultime 24 settimane, confronto tra i dati reali e quelli ottenuti con l'elaborazione fuzzy (grafico più scuro).



settimane	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24
Elaborazione	5	9	8	7	6	8	6	6	4	6	5	4	4	10	10	10	8	8	8	8	6	4	3	9
Dati reali	15	15	14	13	12	13	11	11	9	7	10	9	9	20	20	20	18	18	18	18	16	12	11	11

Fig. 6.4 Articolo C - Giacenze di fine settimana nelle ultime 24 settimane, confronto tra i dati reali e quelli ottenuti con l'elaborazione fuzzy (grafico più scuro).

Nonostante siano stati utilizzati i valori di *default*, i miglioramenti ottenuti applicando ai dati disponibili il sistema di gestione realizzato sono notevoli, ed una prima stima del risparmio associato fornisce risultati incoraggianti. Nella tabella 6.2 si possono confrontare i dati della gestione reale *Gr* con quelli dell'elaborazione fatta *Ge*. Per tutti e tre gli articoli trattati, nelle ultime tre colonne sono illustrate le differenze tra i due metodi. Esse sono espresse in termini di giacenze assolute, di valore delle giacenze e, nell'ultima colonna, di percentuale di risparmio che si sarebbe ottenuta applicando su questi articoli il metodo di gestione sviluppato.

Tab.6.2 Comparazione tra i dati di gestione reali e quelli ottenuti con le elaborazioni fuzzy. Giacenze medie in quantità e valore e percentuale di risparmio.

Art.	μ_{Gr} **(pz)**	μ_{Ge} **(pz)**	$D\mu_G$ **(pz)**	V_{Gr} **(Lit)**	V_{Ge} **(Lit)**	DV **(Lit)**	DV **%**
A	364,0	150,5	213,5	4.564.560	1.887.270	2.677.290	58,7
B	70,6	29,1	41,5	807.310	332.760	474.550	58,8
C	14,2	7,2	7,0	3.586.410	1.818.460	1.767.950	49,3

Le percentuali di risparmio che si otterrebbero sono molto alte. Un po' inferiore alle altre è quella relativa all'articolo C, che probabilmente, avendo il valore più alto, è già gestito con maggiore attenzione. È da segnalare però il fatto che il risparmio maggiore in termini assoluti si otterrebbe con l'articolo A, che ha un valore molto basso.
La tabella 6.3 mostra un confronto diverso tra le prestazioni ottenute nei due casi, basato sulla rotazione delle scorte.

Tab.6.3 Comparazione tra i dati di gestione reali e quelli ottenuti con le elaborazioni fuzzy. Settimane di permanenza media degli articoli a magazzino.

Art.	D **(pz)**	μ_{Gr} **(pz)**	μ_{Ge} **(pz)**	P_{Gr} **(sett.)**	P_{Ge} **(sett.)**
A	470	364,0	150,5	18,6	7,7
B	93	70,6	29,1	18,2	7,5
C	22	14,2	7,2	15,5	7,9

Essendo i dati relativi a ventiquattro settimane, si è scelto di non calcolare un indice di rotazione annuale, che richiederebbe un'estrapolazione eccessiva. Rimanendo invece all'interno del periodo osservato, si è calcolato un indice di rotazione che risulta significativo in un intervallo temporale più ristretto. In realtà, tale indice (P) è reciproco alla rotazione, e valuta il tempo medio di giacenza a magazzino di un articolo in tale periodo. Tempi minori di permanenza media sono chiaramente preferibili. P è espresso in settimane, e si calcola dividendo le giacenze

medie per la quantità di venduto nelle 24 settimane considerate (*D*), e moltiplicando per 24. Anche osservando questa misura le differenze sono notevoli. Si passa da un tempo medio di permanenza originale che varia, a seconda dell'articolo, tra le 15 e le 19 settimane (P_{Gr}) ad uno di 7-8 settimane, ottenuto con le elaborazioni fuzzy (P_{Ge}).

Essendo il *livello di servizio desiderato alto*, i risultati ottenuti sono validi solo nel caso in cui ci sia quasi la certezza di evitare rotture di scorta. Soprattutto per ricambi di poco valore, eventuali disservizi potrebbero nuocere fortemente all'immagine aziendale, a fronte di una riduzione minima nell'onere di gestione. Dai grafici si può notare che le giacenze di fine periodo sono sempre molto alte. Esse sono tali che una carenza nel servizio reso ai clienti si sarebbe potuta verificare solo in seguito ad una quantità richiesta più che doppia rispetto a quella massima osservata nel corso del periodo studiato. Curando in modo più specifico per ogni articolo le variabili di gestione, si possono ottenere risparmi ancora più alti, come si può verificare dall'analisi delle prove illustrate nel prossimo paragrafo.

Verso l'ottimizzazione dei risultati

Si consideri ora solo l'articolo B. Per esso è stata eseguita un'altra serie di elaborazioni, allo scopo di migliorare ulteriormente le prestazioni di gestione ottenute. In un processo reale, piccole modifiche tendenti ad ottimizzare i risultati sono da fare soprattutto nel corso del periodo iniziale di funzionamento. All'inizio è infatti possibile tarare con precisione il comportamento del sistema, secondo una sequenza di controlli e rettifiche.

La fig. 6.5 mostra l'esito delle prime due modifiche provate. Il grafico centrale rappresenta la prima elaborazione, già mostrata nella fig. 6.3. Il grafico più scuro illustra il risultato che si ottiene portando al *massimo* il *livello di servizio desiderato* e al *minimo* l'*onerosità delle giacenze*. Essendo già in partenza orientati verso un servizio molto buono, la variazione è abbastanza contenuta. Il grafico più chiaro in primo piano si ottiene invece con le impostazioni opposte: *livello di servizio minimo* e *onerosità delle giacenze massima*. In questo caso la traslazione verso il basso è notevole, e nel corso del periodo 13 si verifica anche una rottura di *stock*. Al fine di evidenziarne l'entità (3 unità richieste non disponibili), si mostra una giacenza negativa, ricordando però che, dal punto di vista del magazzino, valori inferiori allo zero non hanno senso. Per effettuare le prove successive, si è deciso di riportare i

valori delle due variabili modificate alle impostazioni iniziali di *default*.

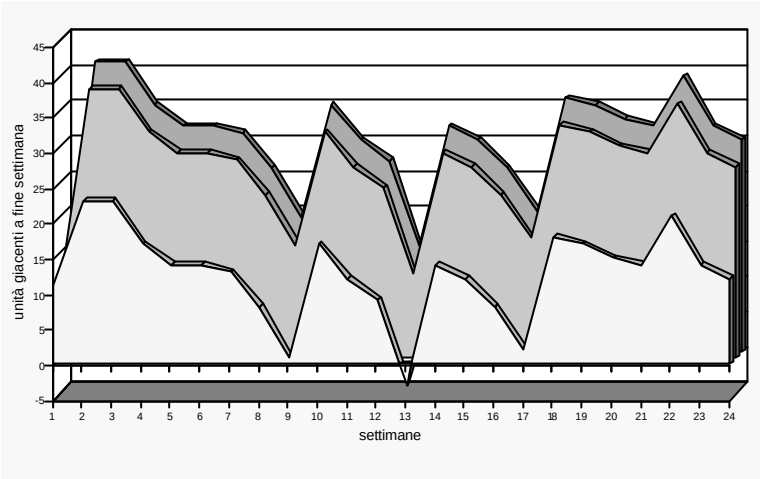


Fig. 6.5 Elaborazioni dell'articolo B. Il grafico al centro mostra l'elaborazione originale. Gli altri si sono ottenuti ponendo al massimo prima il *livello di servizio* (grafico più scuro), poi l'*onerosità delle giacenze* (grafico più chiaro).

Ripartendo dall'elaborazione iniziale, rappresentata nella fig. 6.6 dal grafico più scuro, si è eseguita una nuova prova, abbassando di un valore numerico 0,1 la *domanda attesa*, che si può ora classificare *medio alta.* Il risultato ottenuto è evidenziato nel grafico centrale della stessa figura. Volendo ridurre ulteriormente le giacenze medie, si sono diminuite dello stesso valore 0,1 anche le due variabili relative all'incertezza. Nel grafico in primo piano, l'*incertezza sulla domanda attesa* è *bassa* e l'*incertezza sui tempi di consegna* è *poco più che minima*. Alla fine del periodo 13, il più critico per la gestione, le giacenze raggiungono esattamente il valore zero.

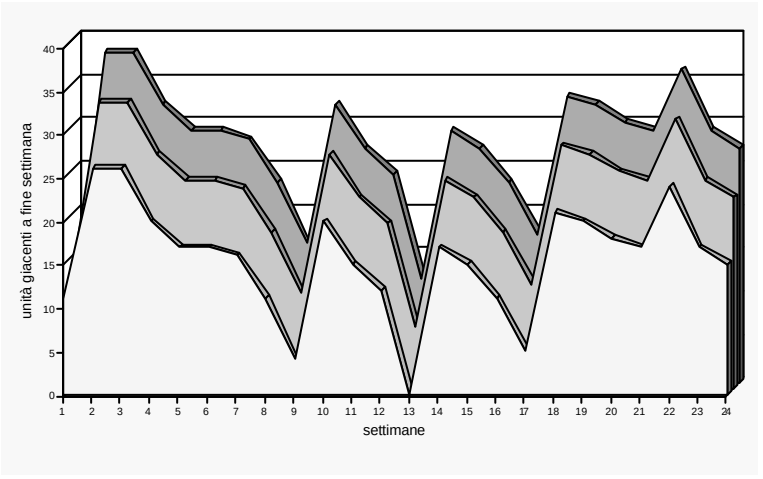


Fig. 6.6 Elaborazioni dell'articolo B. Sullo sfondo l'elaborazione originale. Il grafico centrale deriva dall'abbassamento della *domanda attesa*, quello in primo piano dalla riduzione aggiuntiva delle due variabili sull'incertezza di gestione.

La situazione raggiunta con l'ultima elaborazione potrebbe sembrare ideale, ma lo è solo a consuntivo. Può essere infatti molto rischioso l'impiego di una modalità di gestione così vicina al disservizio. Per evitare che l'even-

tuale rottura di scorte si verifichi, un metodo possibile è quello di concedere al sistema la libertà di emettere degli ordini extra. Tale libertà non è incondizionata, ma deriva dal confronto tra l'urgenza dell'ordine e la sua onerosità. Nel caso specifico che si sta analizzando, è sufficiente abbassare leggermente l'onerosità degli ordini extra, ed il sistema adatta il suo comportamento, emettendo degli ordini aggiuntivi nelle due situazioni più rischiose per la gestione. Questa operazione si può verificare nella fig. 6.7.

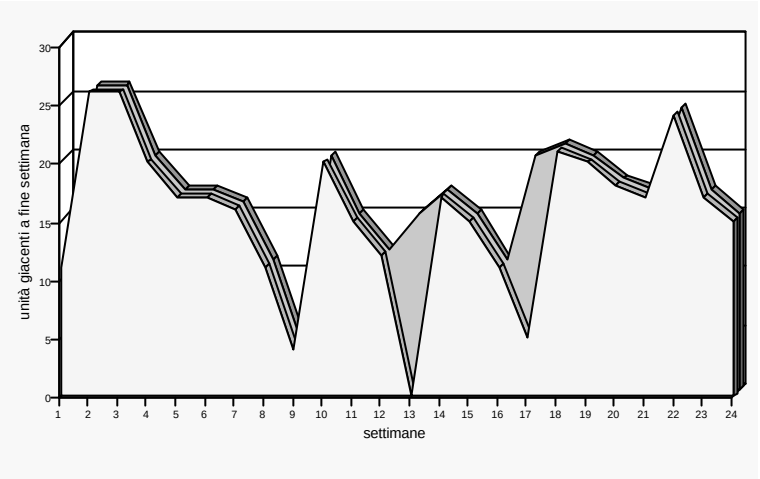


Fig. 6.7 Elaborazioni dell'articolo B. In primo piano l'ultima prova di fig. 6.6, in secondo piano la correzione ottenuta riducendo l'*onerosità degli ordini extra.*

L'ultimo risultato ottenuto con le modifiche precedenti, mostrato in primo piano, è ritoccato portando da *alta* a *medio alta* l'*onerosità degli ordini extra* (riduzione dello 0,1 numerico). Ciò permette di evitare i rischi più ingenti, grazie all'emissione di due ordini anticipati nei periodi 12 e 16, osservabili nel grafico più scuro in secondo piano.

Alla fine, si è ottenuta una prestazione di gestione che presenta una giacenza media molto bassa, garantendo contemporaneamente un buon margine di protezione dall'eventualità che si verifichino rotture di scorta.

Nella tabella 6.4 sono riassunti i risultati conseguiti. Si possono confrontare le prestazioni della gestione reale (prima riga) con quelle ottenute nel calcolo fatto con i valori di default (elaborazione comune) e con quelle relative all'ultima elaborazione, specifica per l'articolo B, mostrata nel grafico scuro della fig. 6.7. I simboli impiegati sono gli stessi delle tabelle precedenti. Il risparmio percentuale è calcolato per le due elaborazioni fuzzy (seconda e terza riga) rispetto al valore reale delle giacenze (prima riga). L'ulteriore miglioramento ottenuto, evidenziato nell'ultima riga, non è trascurabile. È però incoraggiante osservare che la prima elaborazione, la quale richiede una sola impostazione iniziale dei valori delle variabili, fornisce il salto di qualità più grande. Ciò

significa che, a prescindere dalle impostazioni specifiche, è l'intero sistema di gestione che risulta essere più efficiente.

Tab.6.4 Riassunto delle prove relative alle giacenze dell'articolo B.

Articolo B	μ **(pz)**	***V* (Lit)**	***P* (sett.)**	*DV%*
Gestione reale	70,6	807.310	18,2	
Elaborazione comune	29,1	332.760	7,5	58,8
Elaborazione specifica	18,8	214.980	4,9	73,4

Come si è visto, progressi ulteriori si possono ottenere con ritocchi dei valori delle variabili quando il sistema è già in funzione. Spetta all'utente stabilire, in base alla propria realtà gestionale, quale può essere il livello d'impegno da dedicare all'ottimizzazione dei risultati. L'importante è che il maggior impiego di risorse necessario sia bilanciato dal risparmio aggiuntivo che si può conseguire.

Quanta cura dedicare alla fase di perfezionamento dipende soprattutto dalle caratteristiche degli articoli da gestire. Per articoli molto simili dal punto di vista della gestione sarà in genere conveniente non dedicare troppe energie a questa mansione, per articoli aventi attributi peculiari sarà invece opportuno affinare maggiormente il processo di gestione.

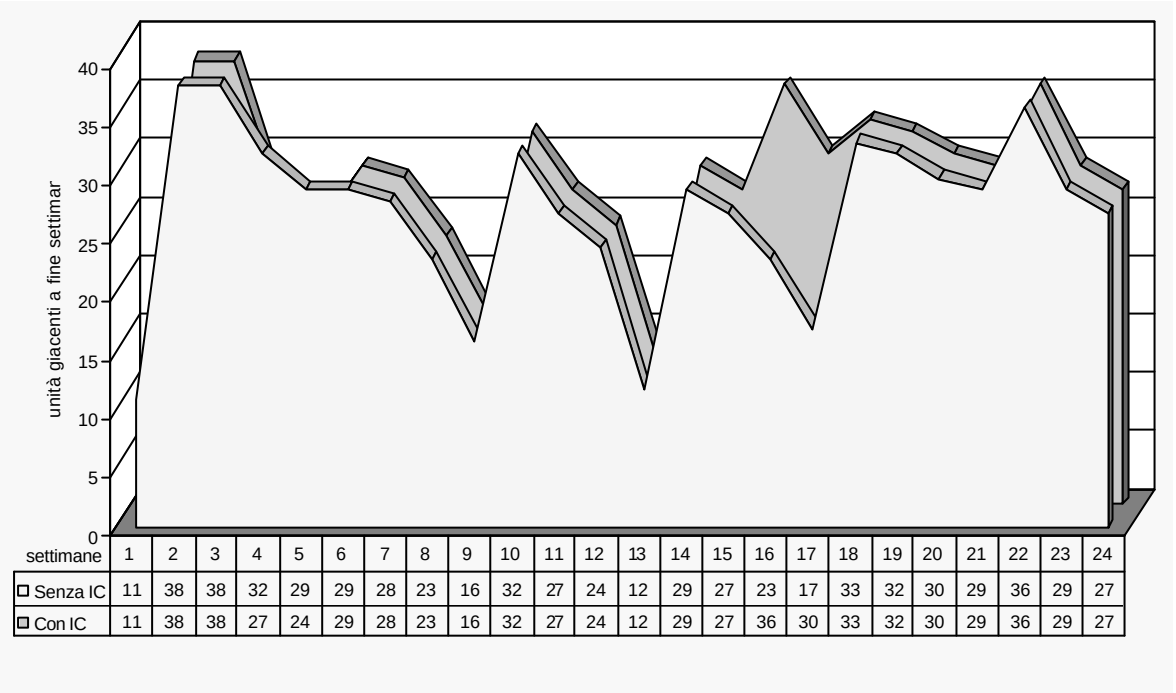
6.3.3 Prove del funzionamento globale del sistema

Vista la diversa natura delle modalità di vendita e del modo di affrontarle, il fatto che il sistema riesca a gestirle entrambe separatamente dovrebbe già garantire che possa anche gestirle congiuntamente senza grossi problemi ulteriori. In effetti, succede semplicemente che le quantità necessarie a soddisfare gli impegni presi preventivamente si aggiungono alle quantità che, in previsione, dovrebbero garantire l'evasione delle richieste che insorgono nell'intervallo di tempo tra un ordine e l'altro.

Le fig. 6.8 e 6.9 evidenziano i risultati di un'elaborazione di questo tipo, rispettivamente in forma grafica e in forma tabulare. Nella fig. 6.8, il grafico in primo piano mostra l'elaborazione comune, fatta con i valori di *default*, il grafico più scuro in secondo piano illustra la stessa elaborazione, con l'aggiunta di tre impegni presi con i clienti.

Nella fig. 6.9 è illustrata l'elaborazione comprensiva degli impegni con i clienti. Le colonne *EA* e *DR* mostrano le giacenze d'inizio settimana, e per ottenere i valori della

tabella relativa al grafico è sufficiente sottrarre le quantità vendute *QV*. Va detto anche che *EA* e *DR* coincidono, poiché i loro valori sono calcolati dopo la consegna delle quantità promesse ai clienti (altrimenti risulterebbe *EA* = *DR* + *IC*). Analizziamo ora le scelte operate dal sistema in questa situazione.

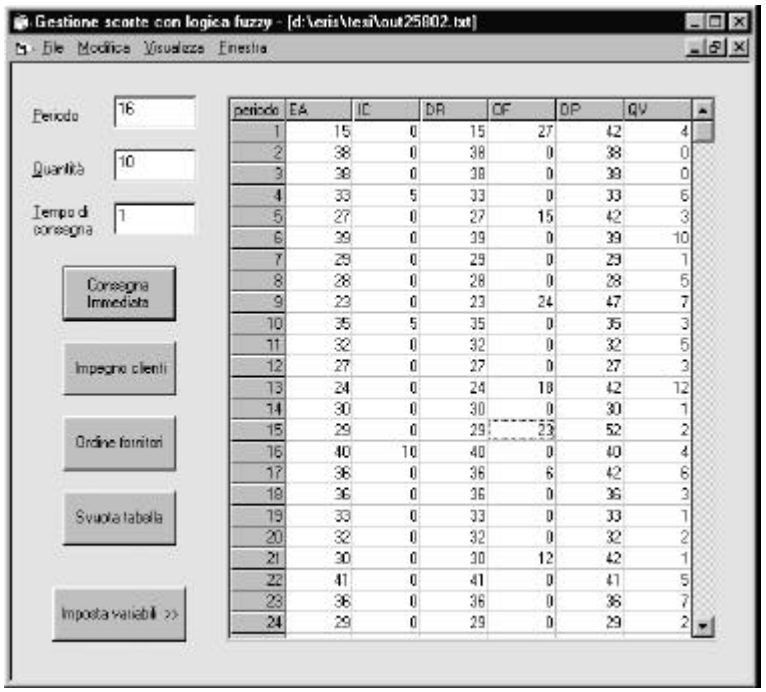


settimane	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24
Senza IC	11	38	38	32	29	29	28	23	16	32	27	24	12	29	27	23	17	33	32	30	29	36	29	27
Con IC	11	38	38	27	24	29	28	23	16	32	27	24	12	29	27	36	30	33	32	30	29	36	29	27

Fig. 6.8 Comparazione di un'elaborazione eseguita senza impegni con clienti (grafico più chiaro in primo piano) con una che comprende 3 impegni di consegna (grafico più scuro).

Il primo impegno è la consegna di cinque unità all'inizio del periodo 4. Essendo le giacenze già sufficientemente alte, il sistema non ha ritenuto opportuno emettere in previsione di ciò un ordine supplementare. L'ordine *normale* successivo sarà però maggiore di cinque unità rispetto all'originale. Il secondo impegno di consegna ha la stessa entità del primo, ed è riferito al periodo 10. Questo impegno non comporta cambiamenti delle giacenze e non si può notare in figura, poiché coincide con l'arrivo di un ordine normale, anche questa volta aumentato di cinque unità. L'ultimo impegno preso è quello di consegnare 10 unità all'inizio del periodo 16. In questo caso il sistema ritiene opportuno emettere un ordine *extra* all'inizio del periodo 15, per non lasciare che il livello delle giacenze dopo la consegna si abbassi troppo (rimarrebbero solo 13 unità all'inizio del periodo17). Si ritengono gli eventuali costi aggiuntivi da addebitare in questo caso all'emissione dell'ordine *extra*, indipendentemente dalla sua entità. Per questa ragione, si ordina per il periodo 16 più dello stretto necessario a soddisfare *IC*, e il succes-

sivo ordine *normale* in arrivo al periodo 18 risulterà perciò di minore entità rispetto all'elaborazione originale.



periodo	EA	IC	DR	OF	OP	QV
1	15	0	15	27	42	4
2	38	0	38	0	38	0
3	39	0	39	0	39	0
4	33	5	33	0	33	6
5	27	0	27	15	42	3
6	39	0	39	0	39	10
7	29	0	29	0	29	1
8	28	0	28	0	28	5
9	23	0	23	24	47	7
10	35	5	35	0	35	3
11	32	0	32	0	32	5
12	27	0	27	0	27	3
13	24	0	24	18	42	12
14	30	0	30	0	30	1
15	29	0	29	23	52	2
16	40	10	40	0	40	4
17	36	0	36	6	42	6
18	36	0	36	0	36	3
19	33	0	33	0	33	1
20	32	0	32	0	32	2
21	30	0	30	12	42	1
22	41	0	41	0	41	5
23	36	0	36	0	36	7
24	29	0	29	0	29	2

Fig. 6.9 Stato delle giacenze ottenuto aggiungendo all'elaborazione originale dell'articolo B degli impegni di consegna (periodi 4, 10, 16). Si può notare l'ordine *extra* di 23 unità emesso nel periodo 15.

È interessante notare che le giacenze medie passano da 29,1 a 29,7 unità, presentando un aumento molto contenuto. Contemporaneamente, a causa dell'aumento della quantità venduta totale, le settimane di permanenza media a magazzino degli articoli scendono notevolmente, da 7,5 a 6,3.

L'introduzione degli impegni con i clienti, a fronte di un piccolo incremento delle giacenze totali dovuto alla crescita dei movimenti, garantisce perciò prestazioni migliori. Ciò è dovuto al fatto che risulta percentualmente ridotto il grado di aleatorietà della gestione. Lavorando solo su ordinazioni si potrebbe idealmente raggiungere un livello di scorte sempre nullo.

7

RASSEGNA DI APPLICAZIONI PER LO SVILUPPO DI SISTEMI FUZZY

7.1 CRITERI DI ANALISI E DI SCELTA

Per l'implementazione software della sezione fuzzy del sistema progettato si è ritenuto opportuno affidarsi ad un'applicazione dotata d'interfaccia grafica, che rendesse in grado di produrre e modificare agevolmente sistemi di elaborazione fuzzy. Durante la ricerca dell'applicazione più adatta, sono stati provati diversi prodotti, soprattutto ***freeware*** e versioni dimostrative di prodotti commerciali. Vista la varietà dell'offerta, si è deciso di presentare un quadro il più possibile rappresentativo del software attualmente disponibile concernente la logica fuzzy.

Viene definito ***freeware*** un programma distribuito gratuitamente, per l'utilizzo del quale gli autori non richiedono alcun compenso.

Si ritiene che i programmi qui presentati, provenienti perlopiù dai siti Internet delle aziende e dei centri di ricerca che li hanno sviluppati, costituiscano una rappresentanza significativa di tutte le applicazioni esistenti in materia.

Due interessanti siti Internet contenenti riferimenti a programmi fuzzy sono:

De Montfort University (UK)
http://www.cms.dmu.ac.uk/People/rij/fuzzy.html

University of Southampton (UK)
Department of Electronics and Computer Science
http://www.isis.ecs.soton.ac.uk/resources/nfinfo/fzsware.shtml

Al fine di stabilire l'adeguatezza di ogni applicazione nei confronti del sistema da sviluppare, sono stati impiegati soprattutto due criteri: la versatilità dell'interfaccia grafica e l'esportabilità dei progetti.

Il poter disporre di un'interfaccia grafica funzionale permette di sviluppare rapidamente i progetti, e soprattutto di modificarli in base ai risultati conseguiti nel corso della sperimentazione.

Nel caso di un sistema non puramente fuzzy come quello progettato, è inoltre indispensabile che la sezione fuzzy risulti interfacciabile con un linguaggio di programmazione classico. A tal fine, il metodo utilizzato più frequentemente dalle applicazioni provate è la generazione di codice, ossia la traduzione del sistema fuzzy progettato in un programma scritto con un linguaggio di programmazione, ad esempio *C* o *Fortran*. Una volta generato il codice, non è più possibile modificare il sistema graficamente, ma si possono utilizzare delle procedure di calcolo fuzzy all'interno di proprie applicazioni. La soluzione ideale è poter impiegare procedure

fuzzy all'interno di linguaggi che, da un lato, siano capaci di produrre agevolmente interfacce per gli utenti di semplice utilizzo, e dall'altro rendano facile la gestione di basi di dati. Si possono citare a questo proposito gli ambienti *Visual Basic* e *Visual C++*.

Una terza caratteristica, meno irrinunciabile rispetto alle prime due, ma comunque degna di attenzione, è la possibilità di scelta offerta dalle applicazioni di sviluppo nei confronti dei principali elementi della base di conoscenza. Importanti sono, da questo punto di vista, la quantità e il tipo di norme disponibili per le operazioni di congiunzione, implicazione e composizione, la disponibilità di differenti tipi di operatori di defuzzificazione, la possibilità di assegnare dei pesi alle regole. Le alternative offerte sono spesso molte e di vario genere, ma a volte per un sistema può risultare preferibile su tutte le altre una specifica operazione o una forma particolare da assegnare ad un insieme, che non tutti i programmi mettono a disposizione.

Un'ulteriore possibilità è quella fornita da certi programmi, che presentano delle procedure di autoapprendimento capaci di generare e modificare autonomamente alcune caratteristiche dei sistemi. Tale opzione è interessante, ma non indispensabile, essendo molto utile soprattutto durante procedure di ottimizzazione di sistemi già ben funzionanti. L'apprendimento viene in genere sviluppato grazie ad algoritmi genetici o, più spesso, a reti neurali, che migliorano le prestazioni dei sistemi se si forniscono loro dei calcoli di esempio da cui imparare.

7.2 APPLICAZIONI ESAMINATE

7.2.1 *Autogen, Fuzzy Relational Model Generator*

© Bruce Postlethwaite, Process Cybernetics Group
University of Strathclyde (UK)

Un **modello fuzzy relazionale** è un modello di un processo reale la cui base di conoscenza è costituita da una rete di relazioni di tipo fuzzy tra le grandezze che descrivono il sistema reale.

È un'applicazione che genera **modelli fuzzy relazionali** di processi dinamici reali. Un modello è costruito direttamente in base all'analisi di una sequenza di dati d'ingresso e uscita. I dati per la generazione del modello si ottengono dall'osservazione del comportamento del processo, variando una alla volta le variabili d'ingresso e registrando l'effetto di queste variazioni sulle altre variabili del sistema. Per elaborare il modello s'impiegano degli algoritmi particolari che collegano variabili in ingresso e in uscita tramite relazioni fuzzy. *Autogen*

utilizza i metodi di riconoscimento e previsione elaborati da Ridley, Shaw e Kruger.

Il programma permette di creare sistemi fuzzy ad una uscita e più ingressi, a partire da un *file* dati di riferimento da cui il sistema ricava il modello che meglio rappresenta il processo reale. La sequenza dei dati forniti deve essere temporale: l'uscita relativa ad un istante di tempo è calcolata in base ai valori degli ingressi e dell'uscita stessa negli istanti precedenti. Non è possibile modificare direttamente il motore d'inferenza associato al modello, ma solo le caratteristiche delle variabili: ruolo, funzioni di membership e significatività temporale (numero di periodi passati da prendere in considerazione per ogni variabile). Una volta costruito il modello, si può visualizzare una comparazione tra i dati forniti per la costruzione del sistema e i risultati elaborati dal sistema stesso (fig. 7.1). Non è prevista la possibilità di interfacciare il programma con altre applicazioni, né la generazione di codice in un qualsiasi linguaggio di programmazione.

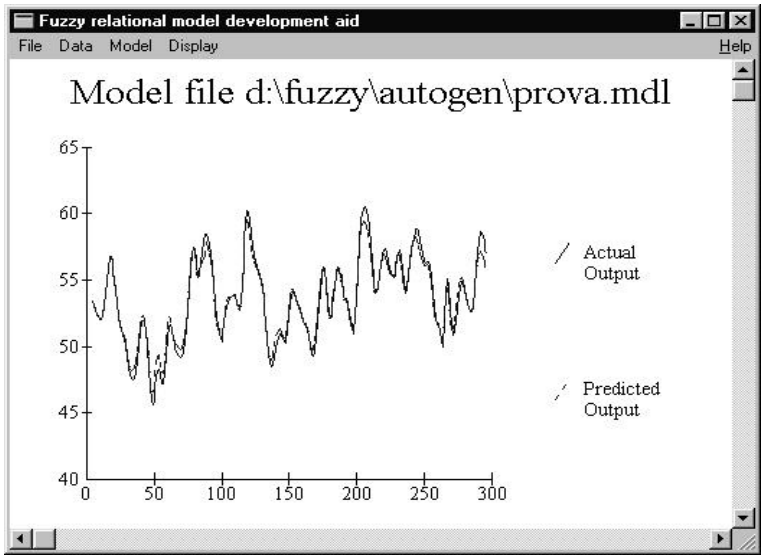


Fig. 7.1 *Autogen* – Grafico per la comparazione delle uscite calcolate dal sistema definito nel progetto *prova.mdl* con quelle previste, utilizzate in precedenza come input per il processo di apprendimento.

7.2.2 *CubiCalc* 2.0

© HyperLogic Corporation (USA)

È un ambiente di progettazione di sistemi fuzzy piuttosto versatile, che fornisce una buona libertà di scelta soprattutto per quanto riguarda la progettazione dei fuzzy set relativi alle variabili. L'interfaccia grafica permette infatti di definire agevolmente nuove funzioni di membership personalizzate. Non è però prevista la selezione grafica delle regole o la loro

auto-generazione sulla base di poche indicazioni iniziali: il motore d'inferenza richiede una scrittura per esteso.

Caratteristica distintiva di *CubiCalc* è la possibilità di scrivere moduli di pre-elaborazione e post-elaborazione. Anche questi ultimi non dispongono di supporto grafico, ma possono ugualmente essere molto utili, permettendo di progettare un sistema che non ha bisogno di elaborazioni aggiuntive provenienti da altri programmi. È accettato input da tastiera o da file, mentre l'output può essere registrato su *file* o visualizzato sullo schermo, sia in forma di tabulato sia graficamente. È possibile la costruzione di grafici personalizzati. Non sono previsti l'interfacciamento con altro software e la generazione di codice. La versione disponibile su Internet (mostrata nella fig. 7.2) è dimostrativa, e non permette il salvataggio dei progetti. Esistono più edizioni commerciali che si differenziano per il numero di variabili e d'insiemi di regole che permettono di utilizzare.

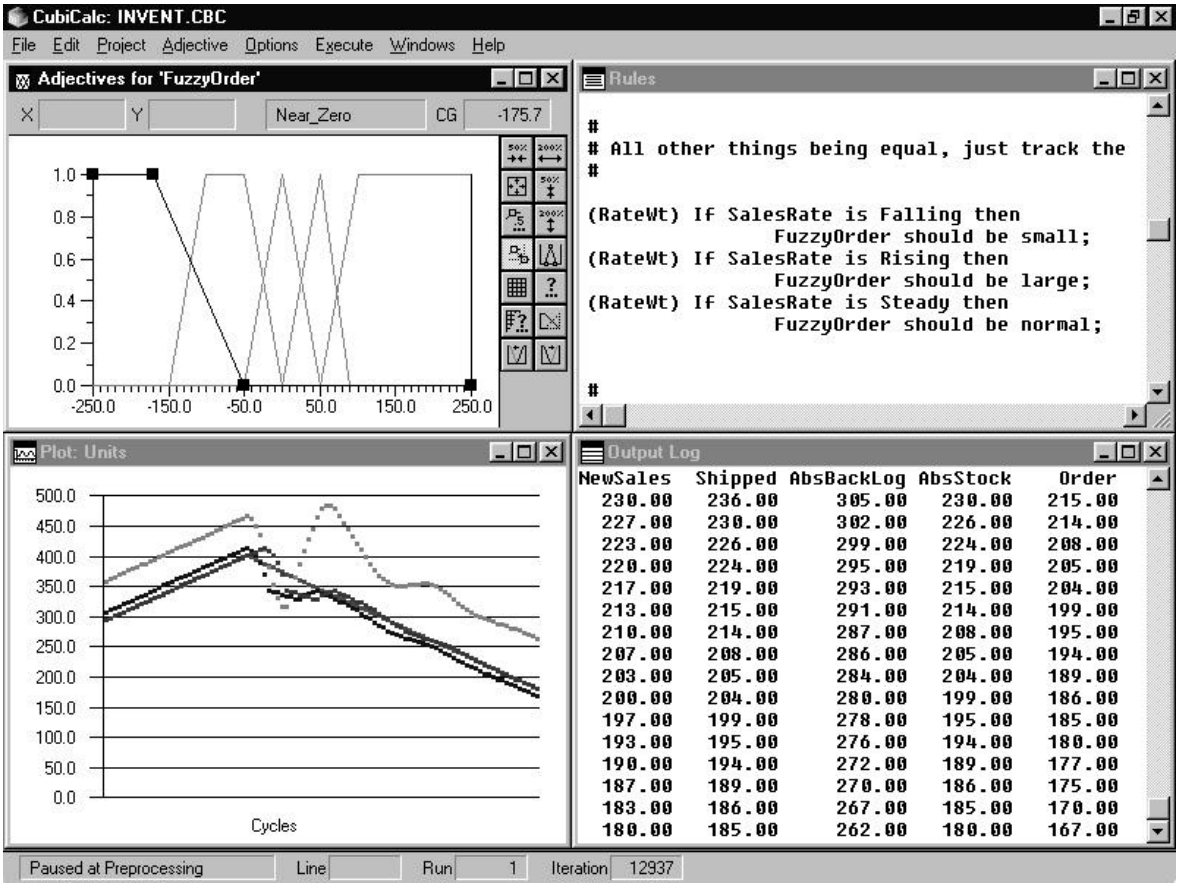


Fig. 7.2 *CubiCalc* – Schermata rappresentativa dei diversi elementi del programma. In alto due finestre di progettazione: a sinistra una variabile linguistica, a destra alcune regole d'inferenza. In basso due finestre visualizzate in fase d'esecuzione, che rappresentano le uscite fornite dal sistema in modo grafico (a sinistra) e su un tabulato (a destra).

7.2.3 *EDIP* Knowledge Manager 1.3

Alexandr A. Savinov
Institute of Mathematics, Kishinev (MOL)

EDIP è un ambiente di sviluppo di sistemi di conoscenza finalizzati alla risoluzione di problemi reali. È basato sulla teoria della *fuzzy propositional logic*, recentemente sviluppata nel laboratorio di Sistemi ad Intelligenza Artificiale dell'Istituto di Matematica della facoltà di Kishinev, in Moldavia. Il concetto fondamentale su cui l'applicazione si basa è quello di *modello fuzzy per attributi*. Tale modello è utilizzato sia per la rappresentazione della conoscenza, sia per il processo d'inferenza logica. I fondamenti teorici della logica fuzzy rimangono, ma sono utilizzati in maniera differente dal solito. La versione resa disponibile è sperimentale: per la versione definitiva da commercializzare è prevista l'aggiunta di un nuovo principio di risoluzione fuzzy attualmente in fase di sviluppo.

7.2.4 *Fuzzy Calculator* 1.0

©Savinov A., Lobyntsev Y.
Institute of Mathematics, Kishinev (MOL)

Assieme all'ambiente di progettazione *EDIP* è fornita una piccola applicazione d'interesse soprattutto didattico (v. fig. 7.3). È una macchinetta calcolatrice, del tutto simile a quelle che si usano ogni giorno, ma i risultati che restituisce sono fuzzy, non algebrici. Le operazioni impiegate sono scelte tra le norme più semplici presentate nel primo capitolo, ma c'è anche la possibilità di verificare l'effetto della compensazione, agendo sul parametro *s* che modifica gradualmente gli operatori utilizzati. Ad esempio, per quanto riguarda la moltiplicazione, si passa in modo continuo dall'operatore di minimo (*s*=0) ad un prodotto algebrico puro (*s*=1).

Fig. 7.3 *Fuzzy Calculator*

7.2.5 *fSC-Net*, Fuzzy Symbolic Connectionist Network 1.0

©Universal Problem Solvers Inc. (USA)

È un programma per la progettazione di sistemi a reti neurali ibridi, che impiega contemporaneamente la **rappresentazione simbolica** e quella **per connessioni**, e sfrutta le potenzialità della fuzzy logic per quanto riguarda la gestione dell'incertezza. Lo scopo principale dell'applicazione è quello di agire come strumento di acquisizione di conoscenza, da utilizzare nello

La **rappresentazione simbolica** utilizza strutture vicine al ragionamento umano, come variabili, comparatori, regole, che agevolano la progettazione, la modifica e la comprensione del funzionamento dei sistemi.
La **rappresentazione per connessioni**, tipica delle reti neurali, aumenta la versatilità di calcolo, la tolleranza a rumori e non linearità, la possibilità di far convivere in un unica rete conoscenze differenti.

sviluppo di sistemi esperti. I sistemi generati sono capaci di apprendere dai dati forniti, modificando le caratteristiche delle connessioni, eliminando le connessioni superflue e aggiungendone di nuove.

Il programma prevede sia l'inserimento di connessioni, regole e insiemi fuzzy da parte dell'utente, sia la loro autogenerazione basata sull'apprendimento. *fSC-Net* non è di utilizzo immediato: il numero elevato di menu e la necessità di seguire una sequenza logica di progettazione predeterminata richiedono una discreta conoscenza dei metodi impiegati, peraltro acquisibile dalla lettura della *Guida In Linea*. In essa sono anche esposte, seppure sinteticamente, le basi teoriche su cui si fonda il funzionamento del programma. Una schermata d'esempio dell'applicazione è mostrata in fig. 7.4.

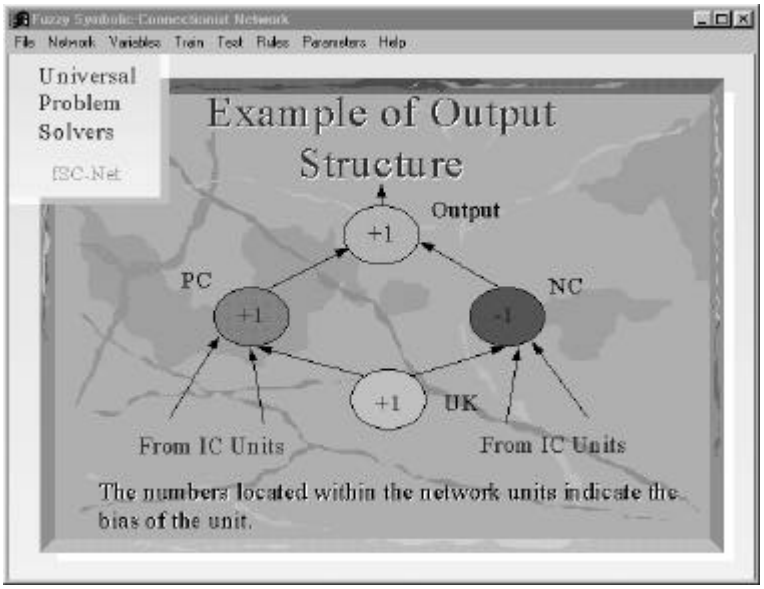


Fig. 7.4 *fSC-Net* – Esempio di struttura di conoscenza a rete nella rappresentazione per connessioni.

7.2.6 *Fuzzle* 3.0

©Modico Inc. (USA)

È un tipico ambiente di sviluppo di sistemi fuzzy con interfaccia utente grafica. Non ha la capacità di autodefinire regole o insiemi, che risultano comunque progettabili con facilità. L'applicazione genera il codice di programmazione relativo ai sistemi progettati, ma solo nei linguaggi *C* e *Fortran*. E' fornita l'interessante opportunità di creare i propri programmi fuzzy eseguibili. La versione disponibile su Internet è solo una rassegna dimostrativa delle caratteristiche del programma commerciale, e non permette di creare un sistema di esempio.

7.2.7 *O'INCA Design Framework*

©Intelligent Machines Inc. (USA)

O'INCA Design Framework è una piattaforma di progettazione software per la costruzione di sistemi esperti. Con essa è possibile creare non solo sistemi basati sulla logica fuzzy, ma anche sistemi di **reti neurali** e sistemi ibridi neuro-fuzzy.

L'interfaccia grafica di progettazione è abbastanza versatile, anche se forse non altrettanto immediata di altre qui presentate. L'output può essere visualizzato sullo schermo o registrato su un file, ma non c'è la possibilità di rappresentarlo graficamente. Per quanto riguarda la generazione di codice di programmazione, l'unico linguaggio supportato è il *C*.

Purtroppo, la versione dimostrativa provata non permette di eseguire realmente i sistemi progettati (è fornito un output casuale). Una schermata che mostra alcune finestre utilizzabili durante la fase di progettazione è illustrata nella fig. 7.5.

Le **reti neurali** costituiscono uno dei tentativi, probabilmente il meglio riuscito, di imitare in una struttura artificiale l'anatomia e il funzionamento del cervello. Le reti neurali sono costituite da collegati tra loro, e l'informazione è contenuta proprio nei collegamenti, come avviene per le sinapsi nel cervello. Caratteristica primaria delle reti neurali è la loro capacità di apprendimento, che permette di migliorare le prestazioni di un sistema di conoscenza per mezzo di fasi di addestramento.

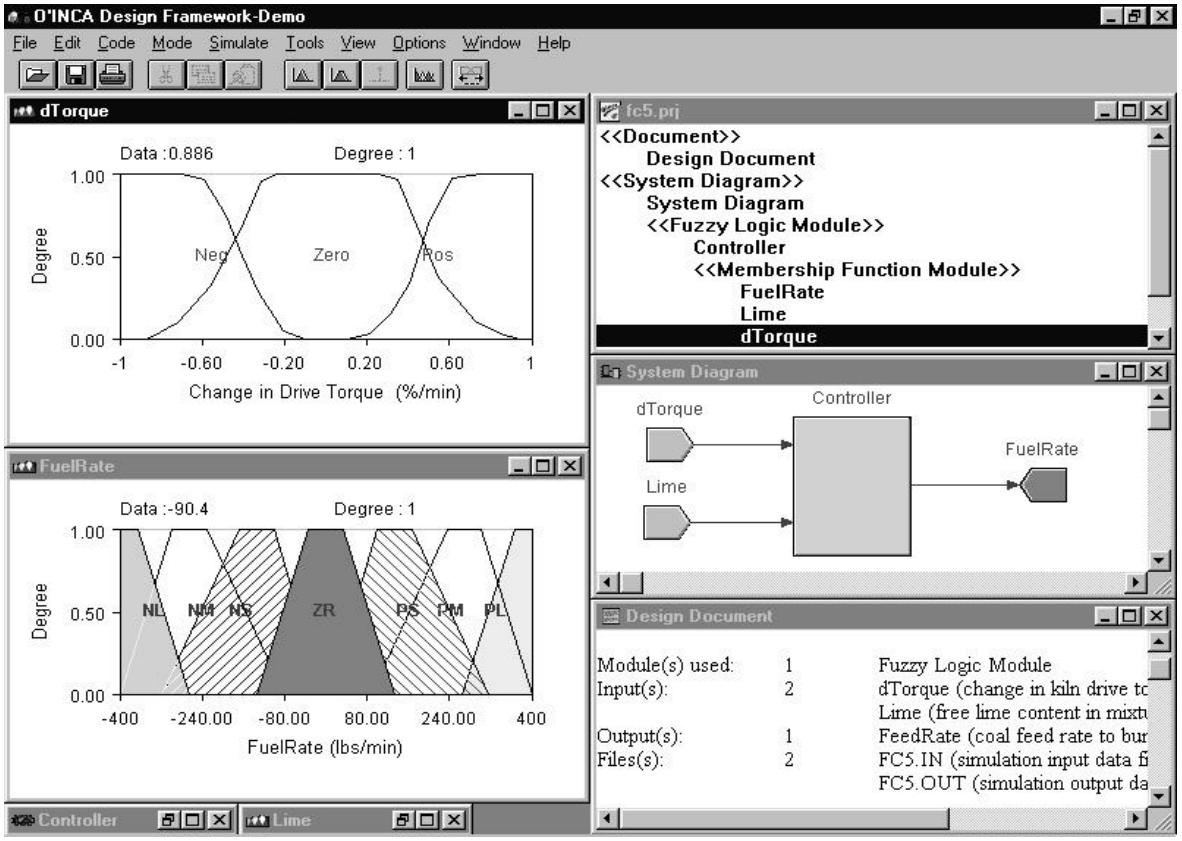


Fig. 7.5 *O'INCA Design Framework* – Schermata d'esempio contenente alcune finestre di progettazione. A sinistra le rappresentazioni di due variabili linguistiche. A destra, dall'alto: lo schema costruttivo del progetto, il diagramma a blocchi del sistema fuzzy e la descrizione delle relative caratteristiche progettuali.

Senza dubbio, la caratteristica di maggior pregio dell'applicazione è la possibilità di utilizzo simultaneo di reti neurali e fuzzy logic. Almeno in via teorica, i sistemi ibridi neuro-fuzzy dovrebbero infatti presentare contemporaneamente le caratteristiche migliori dei due metodi: la facilità di progettazione e modifica da parte dell'utente fornita dalla logica fuzzy e l'ottima flessibilità e capacità di apprendimento peculiari delle reti neurali.

7.2.8 *CLIPS* e *JFS*

CLIPS	*JFS*
©COSMIC (USA)	©Mortensen J.E. (DK)

Sono due linguaggi di programmazione specifici dedicati allo sviluppo di sistemi fuzzy. *CLIPS* è più completo e versatile, e presenta una gamma maggiore d'istruzioni e comandi, ma *JFS* risulta senz'altro di più immediato apprendimento e utilizzo. Non sono provvisti d'interfaccia grafica, né offrono la possibilità di generare codice in altri linguaggi di programmazione di carattere generale. Sono entrambi dotati di appositi compilatori, che permettono di rendere eseguibile il codice scritto.

7.2.9 fuzzyTECH for Business

©Inform GmbH (D)

È probabilmente l'applicazione più completa e versatile, almeno tra quelle analizzate. La progettazione è molto veloce: bastano poche indicazioni da parte dell'utente e il programma genera variabili linguistiche e blocchi di regole, peraltro facilmente modificabili in seguito per mezzo d'interfacce grafiche. Una schermata con diverse finestre di *fuzzyTECH* utilizzabili nella fase di progettazione è illustrata nella fig. 7.6.

Con *fuzzyTECH* è possibile far convivere più blocchi d'inferenza fuzzy all'interno dello stesso sistema, e migliorare le prestazioni dei blocchi creati con una procedura di addestramento basata sull'utilizzo di reti neurali. L'output può essere visualizzato graficamente e si può controllare in esecuzione lo stato di attivazione delle regole. Tenendo costanti eventuali ingressi aggiuntivi, si può rappresentare su un grafico tridimensionale la funzione a due variabili che lega l'uscita a due degli ingressi di un blocco fuzzy. La struttura del progetto si può modificare anche in fase di esecuzione, ed è possibile controllare immediatamente l'effetto delle modifiche

sul funzionamento del sistema. La fig. 7.7 mostra alcune finestre di un progetto durante la fase di esecuzione.

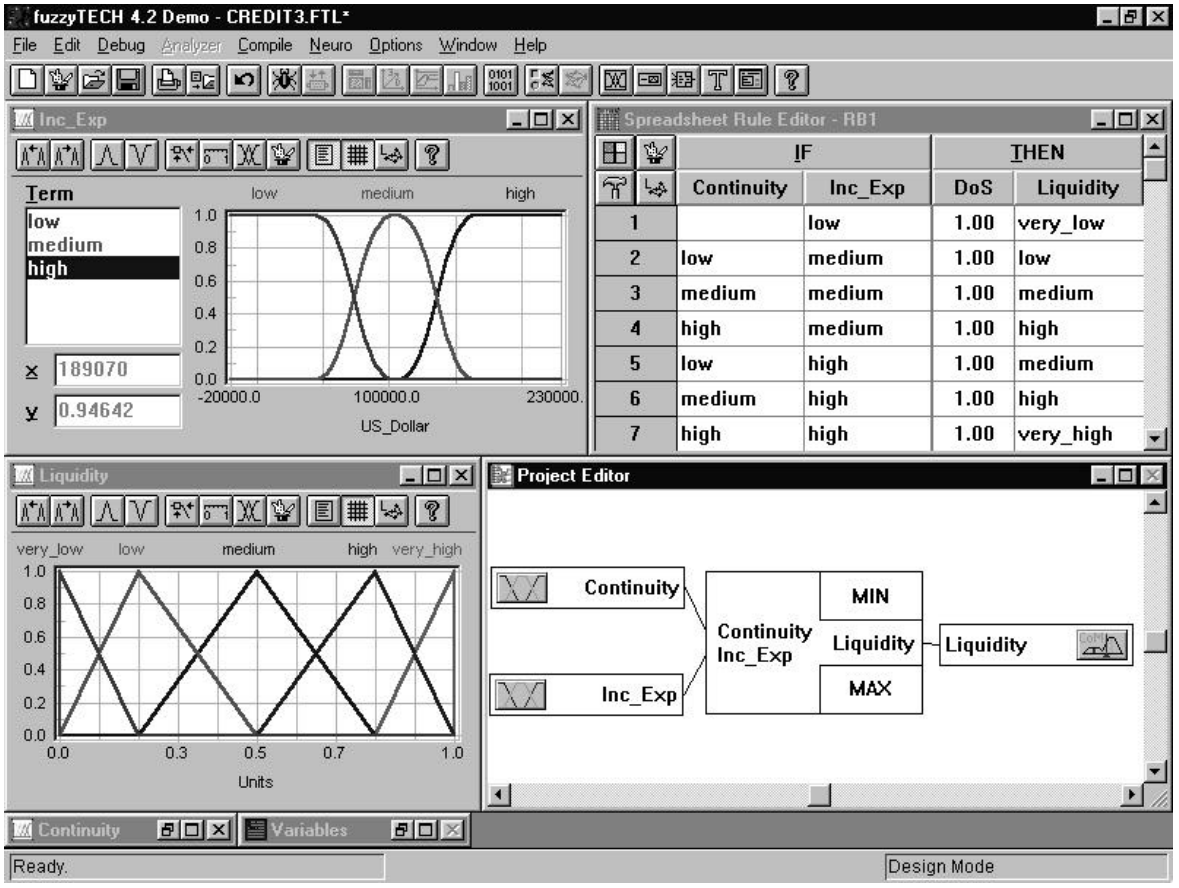


Fig. 7.6 *fuzzyTECH for Business* – Schermata della fase di progettazione. Sulla sinistra sono rappresentati due *term set* di variabili linguistiche; sulla destra si osservano un insieme di regole d'inferenza in alto e lo schema a blocchi del progetto in basso.

È supportata la generazione di codice in diversi linguaggi, tra cui vari tipi di *C*, ma soprattutto è fornita la possibilità di interfacciare i progetti fuzzy con le applicazioni *Access*, *Excel* e *Visual Basic*. Da *Visual Basic* è possibile eseguire, grazie a specifiche **DLL**, un sistema fuzzy precedentemente salvato con *fuzzyTECH*, fornendo gli ingressi come parametri ed ottenendo le uscite corrispondenti.

La versione utilizzata è dimostrativa e non permette di salvare i progetti, né di generare codice di programmazione, inficiando quindi la possibilità di interfacciare i progetti con altre applicazioni.

I grafici descrittivi dei blocchi d'inferenza presentati nel paragrafo 4.3 sono stati creati con questa applicazione, che fornisce rispetto alle altre rappresentazioni più chiare ed esplicative.

Le **DLL** (*Dynamic Link Library*) sono librerie, compilate come eseguibili, che vengono collegate in fase di esecuzione ad applicazioni o ad altre DLL. In questo modo mettono a disposizione le funzioni e le altre risorse che possono contenere. Quando una DLL è stata caricata in memoria dal sistema operativo, può essere condivisa da diverse applicazioni, senza la necessità di caricarne una seconda copia.
Le DLL sono frequentemente impiegate dai linguaggi di programmazione per scambiarsi procedure e funzioni.

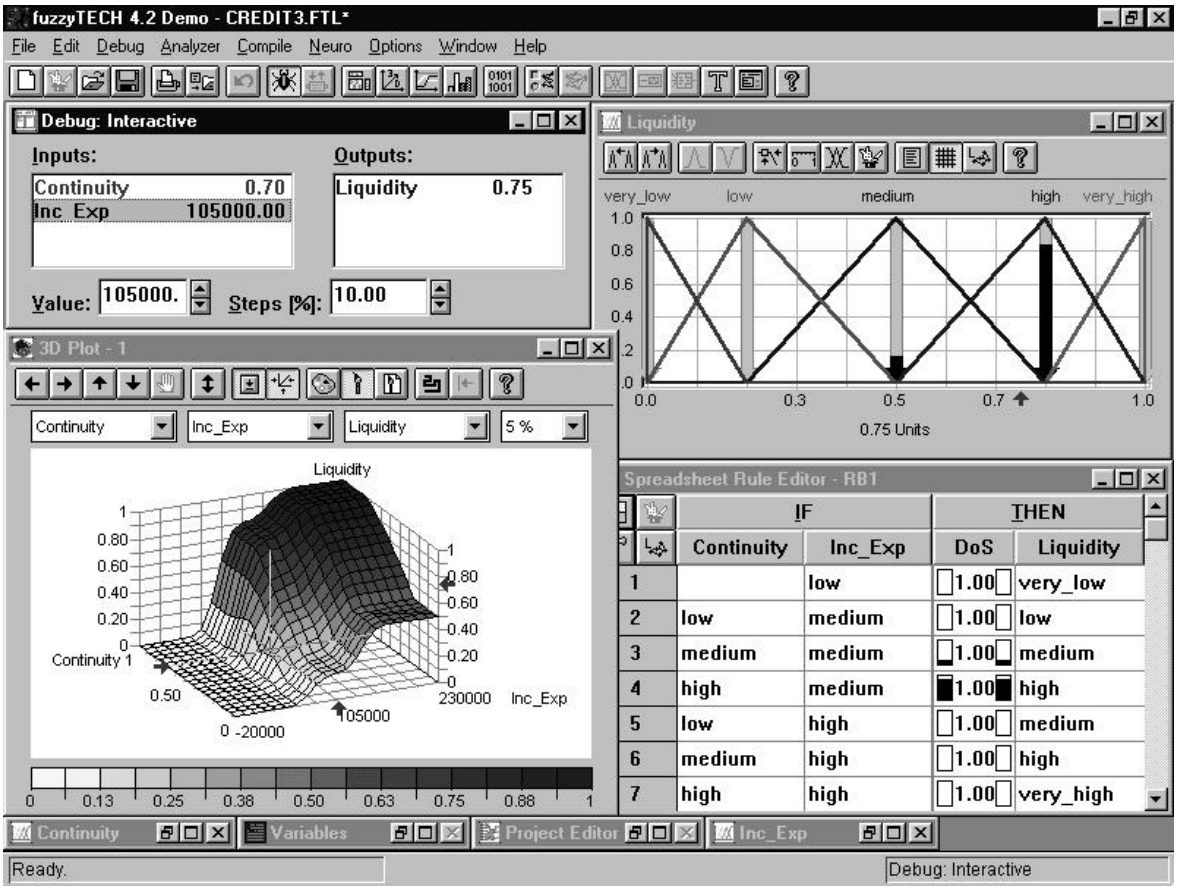


Fig. 7.7 *fuzzyTECH for Business* – Finestra della fase di esecuzione. A sinistra, in alto il *display* con i valori delle variabili in ingresso e in uscita, in basso il grafico tridimensionale della funzione di trasferimento fuzzy con l'uscita, rappresentata sull'asse verticale. A destra, in alto è evidenziato lo stato di attivazione dei termini linguistici dell'uscita e in basso lo stato di attivazione delle regole, in corrispondenza degli ingressi forniti.

7.3 L'APPLICAZIONE SCELTA: *UNFUZZY* 1.1

Fig. 7.8 *UNFUZZY* - Logo dell'applicazione.

UNFUZZY 1.1
Duarte O.G.
Universidad Nacional de Colombia (COL)

È probabilmente il miglior programma *freeware* tra quelli esaminati, evidenziando requisiti superiori alla media delle applicazioni commerciali. Per le sue caratteristiche, questa applicazione è stata preferita alle altre come base di progettazione dei blocchi fuzzy costituenti il sistema, e sarà quindi descritta in modo più dettagliato. Nella fig. 7.8 è rappresentato il logo dell'applicazione *UNFUZZY* con le generalità del progettista.

7.3.1 Caratteristiche generali

Diverse sono le peculiarità che fanno di *UNFUZZY* un ambiente di programmazione fuzzy molto interessante.

La progettazione è molto agevole, ed è offerta la possibilità di generare automaticamente sia le variabili con le caratteristiche preferite, sia gli insiemi di regole.

La gamma di opzioni disponibili è senza dubbio la più ampia tra tutte le applicazioni provate. Sono disponibili tutte le principali funzioni di appartenenza, cinque tipi d'insiemi d'ingresso e cinque operatori di defuzzificazione. È possibile assegnare dei pesi alle regole e variare con dei modificatori le caratteristiche degli antecedenti. È fornita anche la potenzialità di modificare automaticamente l'insieme di regole. Ciò è fatto tramite l'utilizzo di un algoritmo genetico, che impara da un insieme di risultati d'esempio.

Si hanno a disposizione ben nove tipi di norme diverse per le operazioni di congiunzione tra gli antecedenti e di composizione, quattro per la congiunzione dei conseguenti e nove per l'operazione d'implicazione. Alcune di queste norme sono inoltre operatori di compensazione, provvisti di un parametro aggiuntivo che ne aumenta la versatilità.

E' fornita la possibilità di analizzare nel dettaglio lo stato di attivazione delle regole in corrispondenza ai valori d'ingresso immessi. Si può inoltre rappresentare graficamente il variare di un'uscita in relazione alla variazione di un ingresso lungo tutto il suo insieme di definizione.

Il limite maggiore dell'applicazione è purtroppo la mancanza della possibilità di riunire in un unico sistema più blocchi d'inferenza, che devono per forza essere progettati e salvati separatamente.

Oltre che alle peculiarità descritte, la scelta di questa applicazione è dovuta alla sua capacità di generazione di codice nel linguaggio *C++*, imprescindibile ai fini della realizzazione del progetto.

La schermata principale dell'applicazione è mostrata nella fig. 7.9. In essa si notano in alto il menu e la barra degli strumenti e al centro la struttura di uno dei progetti su cui si sta lavorando (è possibile caricare in memoria contemporaneamente più di un progetto). I comandi e le finestre di progettazione dettagliata possono essere attivati perciò in tre modi: con il menu, con i pulsanti della barra degli strumenti e con le icone della rappresentazione grafica del sistema fuzzy.

Analizziamo ora più nel dettaglio gli strumenti di lavoro messi a disposizione dall'applicazione.

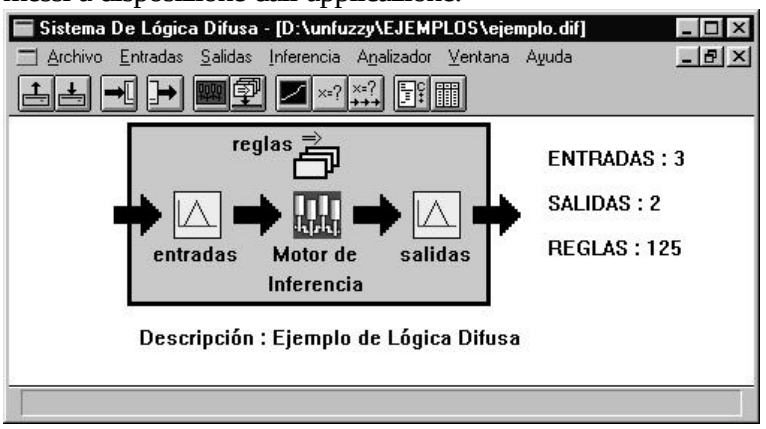


Fig. 7.9 *UNFUZZY* – Schermata principale dell'applicazione, con gli strumenti e lo schema del progetto.

7.3.2 La progettazione di sistemi fuzzy con *UNFUZZY*

Nella fig. 7.10 è mostrata la finestra di progettazione degli ingressi. Con le opzioni della parte superiore della finestra si possono creare le variabili d'ingresso e modificarne nome, insieme di definizione e insieme d'ingresso associato. Soprattutto quest'ultima possibilità si è rivelata alquanto utile, e se ne parlerà più estesamente in seguito.

Con le caselle di dialogo ed i pulsanti della parte bassa è possibile definire la forma specifica delle funzioni di appartenenza relative ad ogni termine linguistico. Con il pulsante in basso a destra (*Autodefinir*) si costruisce automaticamente l'intero *term set*, scegliendo se si vuole descrivere la variabile con insiemi retti oppure a campana.

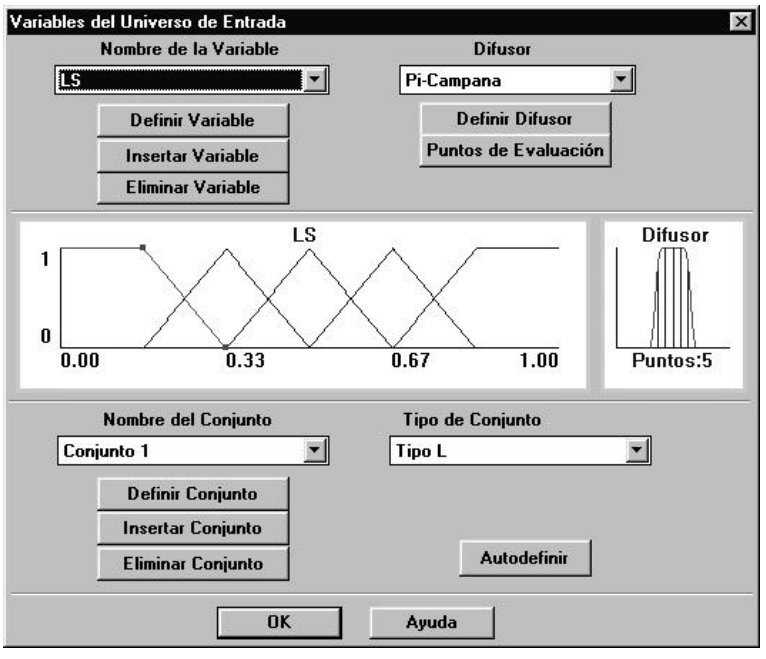


Fig. 7.10 *UNFUZZY* – Finestra per la definizione degli ingressi. In alto si definiscono le caratteristiche della variabile linguistica e dell'operatore di fuzzificazione, in basso quelle dei singoli termini linguistici.

La finestra per la definizione delle uscite è analoga, ma al posto dell'opzione di fuzzificazione si ha quella di defuzzificazione, nella quale è possibile scegliere, tra i tre operatori di massimo e i due di baricentro, quello più adatto alle esigenze del sistema.

La fig. 7.11 illustra la finestra in cui è possibile stabilire le caratteristiche dell'insieme impiegato per la fuzzificazione della variabile selezionata. Oltre a decidere il tipo d'insieme e il numero di punti di valutazione associati (come si è visto nel capitolo quattro, tale scelta è fondamentale per la qualità della risposta), è possibile definirne la forma in modo più specifico, stabilendo quanto l'insieme debba essere "stretto" o "schiacciato", nonché se debba apparire simmetrico oppure no. L'insieme rappresentato in figura è quello trapezoidale (*Pi* nella denominazione del programma), prescelto dopo diverse prove sulle prestazioni fornite. Una finestra analoga è impiegata per modificare in modo specifico le caratteristiche degli insiemi di appartenenza della variabile, sia per gli ingressi sia per le uscite.

Fig. 7.11 *UNFUZZY* – Finestra per la progettazione dell'insieme di fuzzificazione, con i diversi parametri che ne definiscono la forma.

Nelle tre figure successive 7.12, 7.13 e 7.14 si possono definire le regole e scegliere tra le opzioni previste per la progettazione del motore d'inferenza.

La fig. 7.12 mostra la finestra di autogenerazione dell'insieme di regole del sistema. Si possono costruire automaticamente tutte le regole necessarie al progetto, decidendo se si vuole già stabilire l'andamento generale dell'uscita in funzione degli ingressi.

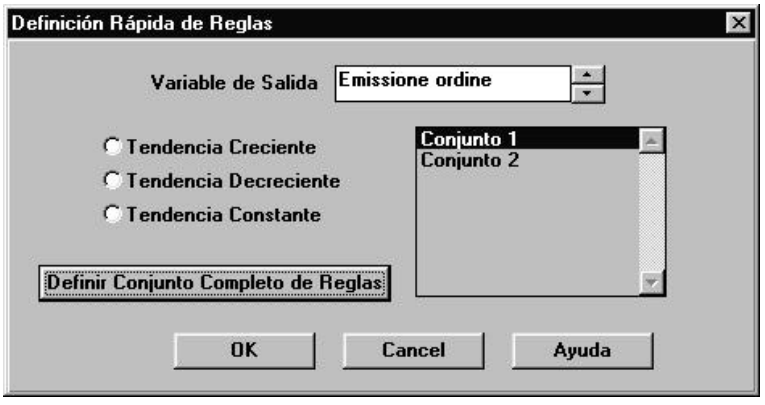


Fig. 7.12 *UNFUZZY* – Finestra per l'autogenerazione delle regole, con possibilità di definire l'uscita direttamente o inversamente proporzionale agli ingressi

Per modificare le regole già definite, cancellarle o costruirne di nuove, s'impiega la finestra di fig. 7.13. In essa si possono specificare inoltre dei pesi per le regole (in alto a destra) e dei modificatori linguistici per gli ingressi (in basso a sinistra).

L'ultima finestra impiegata nella progettazione del motore d'inferenza è illustrata nella fig. 7.14, e permette di definire tutte le caratteristiche del processo di calcolo. In senso orario, partendo dalla sinistra in alto, si possono scegliere: l'operatore per l'implicazione, quello per la composizione (del quale sono evidenziate alcune opzioni disponibili), quello per l'aggregazione delle uscite di ogni regola, quello per l'aggregazione degli ingressi di una stessa regola.

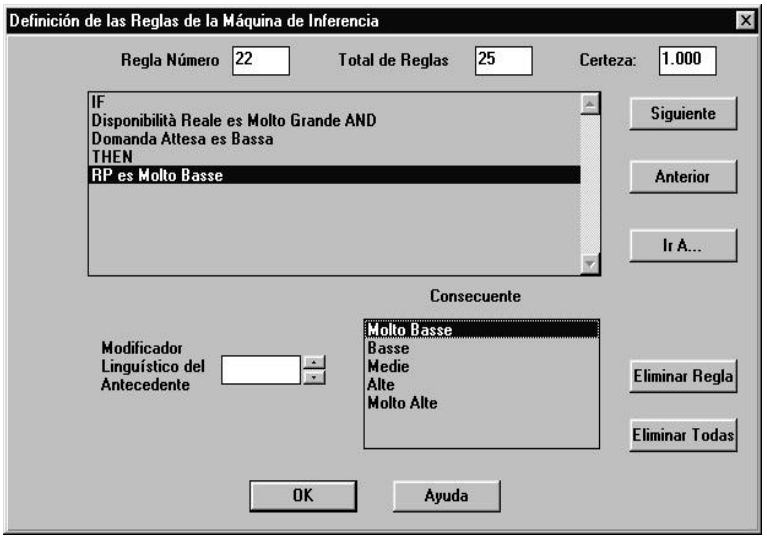


Fig. 7.13 *UNFUZZY* – Finestra per la definizione e la modifica delle regole del motore d'inferenza.

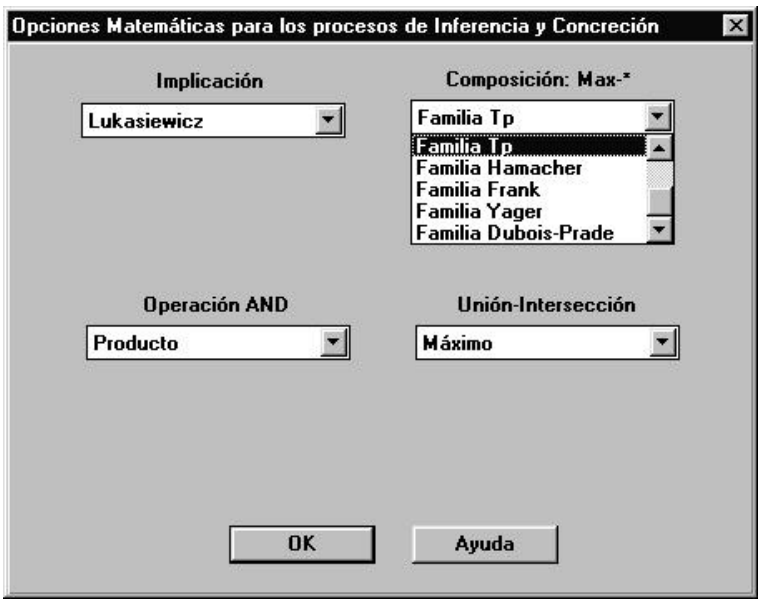


Fig. 7.14 *UNFUZZY* – Finestra per la scelta degli operatori da utilizzare nel processo d'inferenza. Dall'alto a sinistra, in senso orario: operatore d'implicazione, operatore di composizione, operatore di unione tra gli insiemi inferiti dalle diverse regole, operatore di intersezione tra gli antecedenti di una stessa regola.

7.3.3 L'analisi di funzionamento dei sistemi

Nello svolgimento della fase di messa a punto, le potenzialità che *UNFUZZY* mette a disposizione dell'utente

rendono agevole lo studio del comportamento del sistema progettato.

La finestra di fig. 7.15 illustra l'interfaccia grafica di analisi della risposta del sistema progettato. Oltre alla normale esecuzione con tutti gli ingressi inseriti dall'utente, è infatti possibile lasciare un grado di libertà al sistema. In questo modo si può tracciare un grafico che rappresenta l'andamento dell'uscita rispetto alla variazione di una variabile lungo tutto il suo campo di esistenza. Variando ad ogni esecuzione in modo regolare i valori delle altre variabili (nel nostro caso una) si possono ottenere delle famiglie di curve, che illustrano in modo piuttosto chiaro il comportamento del blocco di calcolo fuzzy.

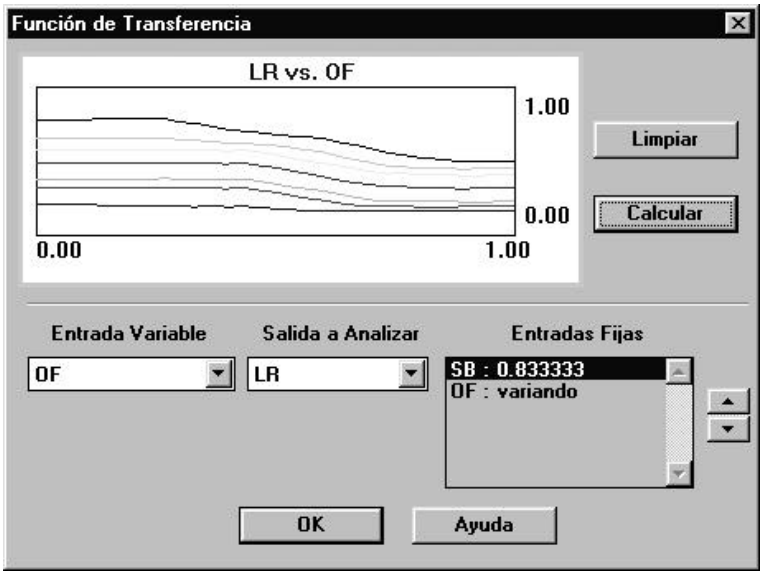


Fig. 7.15 *UNFUZZY* – Finestra per la rappresentazione grafica dei valori d'uscita in funzione di un solo ingresso che varia lungo tutto il suo insieme di definizione, mentre gli altri ingressi rimangono costanti in ogni ciclo di calcolo.

Un'altra metodologia di analisi si concentra invece su unico ciclo di calcolo fatto su ingressi prefissati, ma del quale si studia l'intero procedimento di elaborazione fuzzy. Nella fig. 7.16 è riprodotta la finestra del calcolo passo per passo. Ogni operazione che il sistema esegue, al fine di restituire il valore in uscita corrispondente ai valori forniti in ingresso, è controllabile in modo specifico utilizzando questo strumento. Vale la pena di vedere più nel dettaglio questo strumento, analizzando la sequenza delle operazioni mostrate.

I passi dell'esecuzione sono cinque, e si possono eseguire uno per volta o tutti assieme, analizzandone in seguito le caratteristiche. Il **primo passo** mostra, sul riquadro grafico in alto a destra, come gli insiemi d'ingresso (nel nostro caso trapezoidali) si sovrappongano ai *term set* delle variabili nella procedura di fuzzificazione. Con il **secondo passo** si evidenziano tutte le regole attivate nel processo d'inferenza.

Per ogni variabile, gli insiemi di appartenenza che "toccano" l'insieme d'ingresso corrispondono ad un antecedente con grado di verità non nullo. Le regole attivate sono quelle con entrambi gli antecedenti attivati, la cui intersezione fornisce appunto un grado di verità finale della regola maggiore di zero. Visto che nell'esempio gli insiemi d'ingresso intersecano tre valori linguistici per ognuna delle due variabili, le regole attivate sono in questo caso nove.

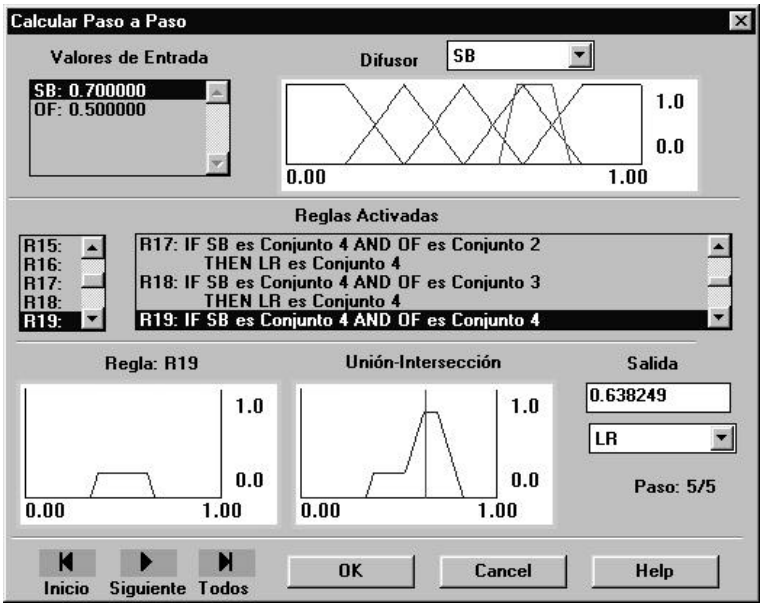


Fig. 7.16 *UNFUZZY* – Finestra per l'esecuzione passo passo del processo d'inferenza.

Nel **terzo passo** sono determinati gli insiemi risultanti dall'applicazione di ogni regola, rappresentati nel riquadro in basso a sinistra. Essi derivano dalla composizione dei gradi di attivazione delle regole con i conseguenti delle regole stesse. Il **quarto passo** riunisce gli insiemi inferiti dalle diverse regole che formano assieme l'insieme d'uscita, dalla cui defuzzificazione, eseguita nel **quinto passo**, si ottiene il valore puntuale dell'uscita. Le due ultime fasi si possono osservare in basso a destra nella fig. 7.16 (il risultato finale è indicato dalla dicitura *Salida*).

Un ultimo metodo per analizzare il funzionamento di un sistema è illustrato nella fig. 7.17, che mostra un tabulato generato dall'applicazione *UNFUZZY*. Nelle prime colonne sono visualizzati dei valori da assegnare alle variabili d'ingresso, generati automaticamente dall'applicazione dividendo gli insiemi di definizione in intervalli regolari. In ogni riga dell'ultima colonna sono visualizzati i valori della variabile d'uscita corrispondenti ai valori d'ingresso della stessa riga. L'utente deve solo specificare in quanti **intervalli di**

Un numero di **intervalli di valutazione** pari a n corrisponde per una variabile a n+1 valori di calcolo visto che i valori estremi sono entrambi presi in considerazione. Nella fig. 6.17, per entrambe le variabili sono stati fissati 30 intervalli di valutazione. Le righe elaborate sono perciò 31^2=961.

valutazione intende dividere gli ingressi, il programma visualizza sullo schermo l'intero tabulato, e fornisce la possibilità di salvare il tutto su un archivio sequenziale.

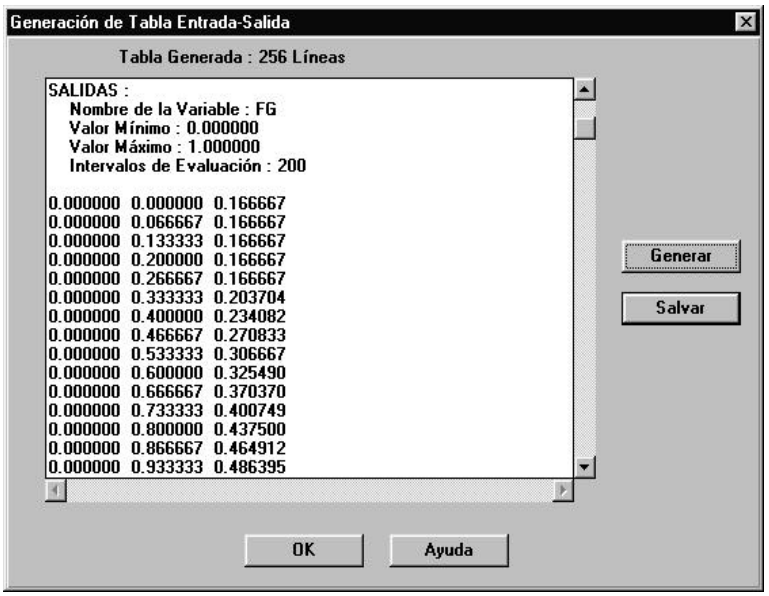


Fig. 7.17 *UNFUZZY* – Finestra per la generazione e il salvataggio su file del tabulato di calcolo contenente i diversi valori degli ingressi e le uscite corrispondenti.

7.3.4 La programmazione *C++* con il codice generato da *UNFUZZY*

Come si è detto, la generazione di codice di programmazione nel linguaggio *C++* è stata una delle determinanti principali che hanno fatto propendere per la scelta di *UNFUZZY* in qualità di ambiente di progettazione dei blocchi fuzzy.

La fig. 7.18 illustra la finestra di generazione di codice sorgente. I linguaggi supportati sono *C* e *C++* (che permette di utilizzare una più moderna e versatile **programmazione ad oggetti**), con l'opportunità aggiuntiva di generare una descrizione delle caratteristiche progettuali del sistema in forma di testo (opzione *Texto*).

Partendo da un sistema fuzzy composto di un solo blocco d'inferenza, avente tutte le caratteristiche desiderate, si può generare il codice di programmazione corrispondente. Nel file di tipo `.cpp` che se ne ricava, il blocco fuzzy progettato, che nel nostro esempio prende il nome di *BloccoFuzzy1*, è implementato come una *classe* derivata dalla *classe* generale *SistemaLogicaDifusa*. Da essa eredita, secondo i meccanismi della progettazione ad oggetti, le funzioni che permettono di mettere in atto tutto il processo di calcolo. La classe *SistemaLogicaDifusa* e le funzioni associate sono contenute

nei *file* `fuzzy.cpp` e `fuzzy.hpp` forniti assieme all'applicazione.

Per utilizzare il codice generato, è necessario costruire un progetto in un ambiente di programmazione *C++*. Esso deve comprendere un file principale nel quale inserire il codice eseguibile del programma e le *classi* generate relative ad ogni blocco (che si possono comunque salvare in *file* distinti). Nel progetto si deve inserire anche il file `fuzzy.cpp`, mentre il *file* `fuzzy.hpp`, come gli altri dello stesso tipo già forniti dai compilatori *C++*, sono richiamati da quelli `.cpp` al momento della compilazione.

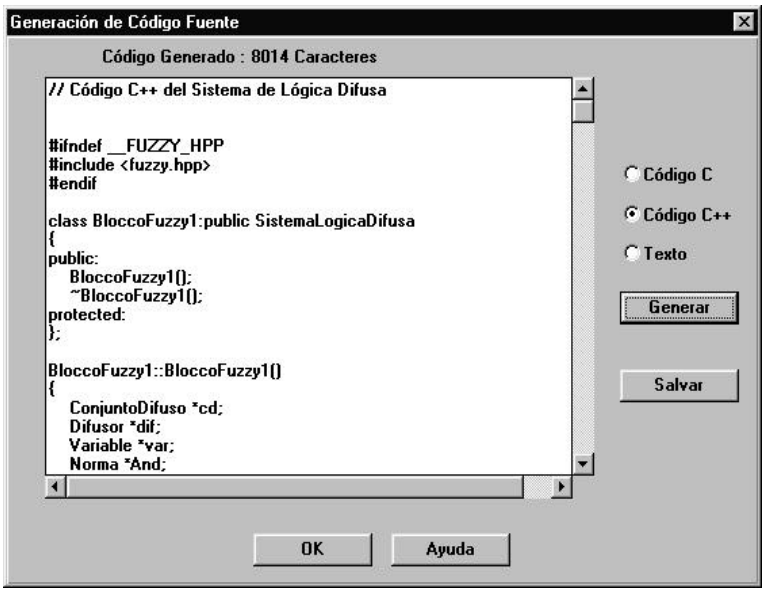


Fig. 7.18 *UNFUZZY* – Finestra per la generazione di codice sorgente nei linguaggi di programmazione *C* e *C++* o, in alternativa, di una descrizione testuale delle caratteristiche del progetto.

Va detto che il codice prodotto da *UNFUZZY* segue gli standard del *C++* distribuito dalla casa *Borland*, il quale presenta molte differenze nella sintassi se confrontato con il *Visual C++ Microsoft*. Per questa ragione si è impiegato come ambiente di progettazione l'edizione 5.0 di *Borland C++*. *Visual C++* sarebbe risultato più facilmente interfacciabile con *Visual Basic*, prodotto dalla stessa casa, per mezzo del quale è stata implementata l'interfaccia grafica del progetto globale.

7.3.5 Documentazione

L'utilizzo di UNFUZZY è indubbiamente semplice e rapido, e i meccanismi di progettazione e di analisi si comprendono in tempi molto brevi. Tuttavia, l'elevato numero di opzioni e di scelte cui l'utente si trova di fronte, unito all'impiego della lingua spagnola anche per termini tecnici che

si è abituati a leggere in inglese, determinano in alcuni casi il bisogno di spiegazioni più dettagliate ed esaustive. In questi casi, viene incontro all'utente una *Guida In Linea* (pulsante *Ayuda* di ogni finestra) sintetica ma completa, che spiega come sfruttare al meglio le potenzialità offerte dall'applicazione, aggiungendo anche delle brevi considerazioni teoriche sulla logica fuzzy, che permettono di capire meglio gli effetti delle scelte che si operano.

La *Guida In Linea*, la cui schermata iniziale è illustrata nella fig. 7.19, è riprodotta interamente in un file documento fornito assieme all'applicazione. In questa versione, la guida è organizzata non più come ipertesto, ma in forma di vero e proprio manuale utente, da stampare ed usare su carta.

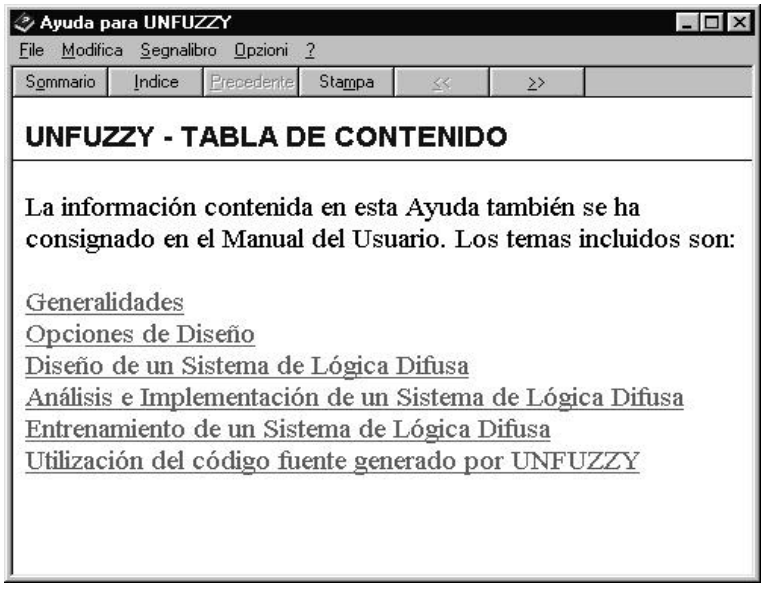


Fig. 7.19 *UNFUZZY* – Schermata iniziale della *Guida In Linea*.

CONCLUSIONI

Con il lavoro svolto si è perseguito l'obiettivo di sviluppare un sistema di gestione scorte di nuova concezione, che potesse costituire una valida alternativa alle metodologie attualmente in uso.

Il sistema di gestione, basato sulla logica fuzzy, è stato progettato ed implementato in un'applicazione software utilizzabile da utenti finali, ossia da qualcuno che si occupi della gestione di un magazzino.

La fase di messa a punto e l'analisi finale delle prestazioni sono state eseguite utilizzando dati di gestione reali per gli ingressi del sistema. Indicazioni utili si sono ricavate dapprima comparando le prestazioni ottenute con quelle del sistema di gestione originale, poi confrontando le diverse modalità di utilizzo del sistema sviluppato.

Il primo tipo di test ha mostrato che, pur mantenendo un margine di sicurezza notevole nei confronti di eventuali rotture di *stock*, l'applicazione del sistema fuzzy alla realtà analizzata consentirebbe un risparmio di gestione che si aggira intorno al 50%. Questo dato è particolarmente significativo, se si considera che è stato ottenuto utilizzando per le variabili di gestione i valori di *default*, stabiliti per un'intera famiglia di articoli. In questo modo l'impegno di gestione è minimo; basta impostare i valori delle variabili validi per tutta la famiglia e il sistema non avrà bisogno di istruzioni aggiuntive. Solo in certi casi è necessario correggere, per alcuni articoli aventi caratteristiche particolari, le impostazioni di base. Rispetto al metodo di gestione originale il vantaggio è duplice: oltre a non esserci il bisogno di rettificare gli ordini decisi dal sistema, la fase iniziale d'impostazione risulta molto più rapida.

Curando con maggiore attenzione i valori delle variabili per ogni singolo articolo, si può ottenere un risparmio aggiuntivo stimato vicino al 20%, sempre rimanendo lontani da possibili disservizi. Questa impostazione è una possibile integrazione al solo utilizzo dei valori di *default*, poiché, a fronte di un accresciuto impegno nella fase d'impostazione, fa conseguire una ulteriore riduzione nei costi di mantenimento delle scorte. Indicativamente, se gli articoli sono molti e con caratteristiche simili, conviene una gestione di tipo standardizzato. Quando, invece, si ha a che fare con articoli costosi, o difficili da riunire in gruppi, può essere vantaggioso dedicarsi con più attenzione ad ogni singolo codice.

Spetta comunque all'utente decidere l'utilizzo del sistema che più si addice alle sue esigenze. Per quanto riguarda la progettazione, l'obiettivo è stato quello di fornire uno strumento flessibile, capace di lavorare con un buon grado d'autonomia, ma anche di soddisfare quegli utenti che avessero l'esigenza di ottimizzare le prestazioni di gestione.

L'inserimento di fattori sfumati nel processo di gestione, utilizzati secondo la logica decisionale di un operatore umano, si è perciò rivelato attuabile. Il sistema realizzato ha fornito risultati incoraggianti, non solo per le prestazioni raggiunte, ma anche per quanto riguarda possibili sviluppi futuri.

Nella fase di verifica, l'impiego di dati inerenti la gestione di prodotti finiti ha reso più significativa l'analisi delle prestazioni, data la maggiore aleatorietà associata in genere a questo tipo di gestione. Tuttavia, il sistema fuzzy implementato ha carattere generale, e non si esclude che l'applicazione a determinate realtà gestionali possa richiedere il suo affinamento, ad esempio con l'aggiunta di alcune variabili più specifiche.

Una prima direzione di sviluppo del sistema potrebbe essere quindi l'aumento del numero di variabili contemplate. Le nuove variabili sarebbero comunque da considerarsi facoltative, per non appesantire il nucleo del sistema, che si ritiene comprenda già i fattori basilari di una gestione di scorte.

Un altro perfezionamento possibile è l'accrescimento dell'autonomia decisionale del sistema, al fine di ridurre ulteriormente la necessità di interventi umani. Per ottenerlo, si dovrebbe rendere automatica l'impostazione dei valori linguistici da attribuire alle variabili di gestione fuzzy. Almeno per alcune di esse, ciò si potrebbe realizzare ricavando le informazioni necessarie dai dati storici recenti della gestione. In particolare, potrebbe rivelarsi molto utile l'impiego di variabili di *feedback*, che permettano la modifica automatica delle impostazioni in base al confronto tra le prestazioni ottenute e quelle desiderate. Interessante sarebbe sperimentare a questo proposito l'utilizzo congiunto di logica fuzzy e reti neurali, per ottenere sistemi con migliori capacità di apprendimento.

APPENDICE A

IL PROGRAMMA DI CALCOLO – CODICE *C++*

```
#ifndef __FUZZY_HPP          // header file fornito con l'applicazione UNFUZZY, contenente
#include <fuzzy.hpp>         // la dichiarazione della classe SistemaLogicaDifusa;
#endif                       // l'altro file fornito (fuzzy.cpp) deve essere compilato
                             // assieme al codice trascritto in questa appendice

#ifndef __CONIO_H            // header file del C++, che permette di utilizzare alcune
#include <conio.h>           // funzioni di I/O
#endif


// blocco fz1
// IN1: LS
// IN2: OR
// OUT: FG

// DICHIARAZIONE DELLA CLASSE fz1 (CHE RAPPRESENTA IL BLOCCO FUZZY 1)
// la classe è dichiarata a partire dalla classe genitore
// SistemaLogicaDifusa (che descrive un blocco fuzzy), di cui eredita tipi e funzioni

class fz1:public SistemaLogicaDifusa
{
public:
    fz1();
    ~fz1();
protected:
};

// COSTRUTTORE DELLA CLASSE fz1

fz1::fz1()
{

// DICHIARAZIONI DELLE VARIABILI INTERNE USATE NELLA CLASSE
    ConjuntoDifuso *cd;
    Difusor *dif;
    Variable *var;
    Norma *And;
    Norma *Composicion;
    Norma *Conjuncion;
    Implicacion *Implica;
    Concresor *conc;

    entradas=new Universo(2);
    salidas=new Universo(1);

// IMPOSTAZIONE DELLA PRIMA VARIABILE D'INGRESSO DEL BLOCCO

    var=new Variable(5);            // creazione variabile con cinque valori
    var->rangoMinimo(0.000000);     // limite inferiore campo di esistenza
    var->rangoMaximo(1.000000);     // limite superiore campo di esistenza

// si aggiungono cinque insiemi linguistici alla variabile,
// descrivendo il tipo e i punti di definizione di ognuno
    cd=new ConjuntoL(0.000000,0.166667,0.333333);
    var->adicionarConjuntos(cd);
    cd=new ConjuntoTriangulo(0.166667,0.333333,0.500000);
    var->adicionarConjuntos(cd);
    cd=new ConjuntoTriangulo(0.333333,0.500000,0.666667);
    var->adicionarConjuntos(cd);
    cd=new ConjuntoTriangulo(0.500000,0.666667,0.833333);
    var->adicionarConjuntos(cd);
    cd=new ConjuntoGamma(0.666667,0.833333,1.000000);
    var->adicionarConjuntos(cd);

// si definisce l'insieme di fuzzificazione: forma, punti di definizione
// e numero di punti di valutazione
    dif=new DifusorPi(0.500000,0.200000,0.100000,0.100000,0.200000);
```

```
    dif->numeroPuntos(5);
    var->difusorEntrada(dif);

    var->nombreVariable("LS");              // nome della variabile
    var->numeroIntervalos(30);              // intervalli di calcolo per i tabulati
    entradas->adicionarVariable(var);       // specifica che la variabile è d'ingresso

// IMPOSTAZIONE DELLA SECONDA VARIABILE D'INGRESSO DEL BLOCCO
// (stesse modalità della prima variabile)

    var=new Variable(5);
    var->rangoMinimo(0.000000);
    var->rangoMaximo(1.000000);

    cd=new ConjuntoL(0.000000,0.166667,0.333333);
    var->adicionarConjuntos(cd);
    cd=new ConjuntoTriangulo(0.166667,0.333333,0.500000);
    var->adicionarConjuntos(cd);
    cd=new ConjuntoTriangulo(0.333333,0.500000,0.666667);
    var->adicionarConjuntos(cd);
    cd=new ConjuntoTriangulo(0.500000,0.666667,0.833333);
    var->adicionarConjuntos(cd);
    cd=new ConjuntoGamma(0.666667,0.833333,1.000000);
    var->adicionarConjuntos(cd);

    dif=new DifusorPi(0.500000,0.200000,0.100000,0.100000,0.200000);
    dif->numeroPuntos(5);
    var->difusorEntrada(dif);

    var->nombreVariable("OR");
    var->numeroIntervalos(30);
    entradas->adicionarVariable(var);

// IMPOSTAZIONE DELLA VARIABILE D'USCITA DEL BLOCCO
// (stesse modalità delle variabili d'ingresso)

    var=new Variable(5);
    var->rangoMinimo(0.000000);
    var->rangoMaximo(1.000000);

    cd=new ConjuntoL(0.000000,0.000000,0.250000);
    var->adicionarConjuntos(cd);
    cd=new ConjuntoTriangulo(0.000000,0.250000,0.500000);
    var->adicionarConjuntos(cd);
    cd=new ConjuntoTriangulo(0.250000,0.500000,0.750000);
    var->adicionarConjuntos(cd);
    cd=new ConjuntoTriangulo(0.500000,0.750000,1.000000);
    var->adicionarConjuntos(cd);
    cd=new ConjuntoGamma(0.750000,1.000000,1.000000);
    var->adicionarConjuntos(cd);

    var->nombreVariable("FG");
    var->numeroIntervalos(292);     // intervalli di suddivisione dell'insieme di
                                    // definizione; determina la precisione di calcolo
    salidas->adicionarVariable(var);        // specifica che la variabile è d'uscita

// IMPOSTAZIONE DEL MOTORE D'INFERENZA

    motor=new MaquinaInferencia(entradas,salidas,25);
                                    // crea un motore d'inferenza contenente 25 regole

// scelta delle norme da utilizzare nel processo d'inferenza:
    And=new Producto();                     // scelta dell'operazione d'intersezione
    Composicion=new Producto();             // scelta dell'operazione di composizione
    Implica=new ImplicacionMinimo();        // scelta dell'operazione d'implicazione
    motor->and(And);
    motor->composicion(Composicion);
    motor->implicacion(Implica);

// impostazione delle regole linguistiche; il primo parametro è il numero
// della regola, la seconda il numero della variabile, la terza il valore
// linguistico della variabile per quella regola
    motor->conjuntoEntrada(0,0,0);
    motor->conjuntoEntrada(0,1,0);    motor->conjuntoSalida(0,0,2);
```

```
    motor->conjuntoEntrada(1,0,0);
    motor->conjuntoEntrada(1,1,1);    motor->conjuntoSalida(1,0,1);
    motor->conjuntoEntrada(2,0,0);
    motor->conjuntoEntrada(2,1,2);    motor->conjuntoSalida(2,0,1);
    motor->conjuntoEntrada(3,0,0);
    motor->conjuntoEntrada(3,1,3);    motor->conjuntoSalida(3,0,0);
    motor->conjuntoEntrada(4,0,0);
    motor->conjuntoEntrada(4,1,4);    motor->conjuntoSalida(4,0,0);
    motor->conjuntoEntrada(5,0,1);
    motor->conjuntoEntrada(5,1,0);    motor->conjuntoSalida(5,0,3);
    motor->conjuntoEntrada(6,0,1);
    motor->conjuntoEntrada(6,1,1);    motor->conjuntoSalida(6,0,2);
    motor->conjuntoEntrada(7,0,1);
    motor->conjuntoEntrada(7,1,2);    motor->conjuntoSalida(7,0,1);
    motor->conjuntoEntrada(8,0,1);
    motor->conjuntoEntrada(8,1,3);    motor->conjuntoSalida(8,0,1);
    motor->conjuntoEntrada(9,0,1);
    motor->conjuntoEntrada(9,1,4);    motor->conjuntoSalida(9,0,0);
    motor->conjuntoEntrada(10,0,2);
    motor->conjuntoEntrada(10,1,0);    motor->conjuntoSalida(10,0,3);
    motor->conjuntoEntrada(11,0,2);
    motor->conjuntoEntrada(11,1,1);    motor->conjuntoSalida(11,0,3);
    motor->conjuntoEntrada(12,0,2);
    motor->conjuntoEntrada(12,1,2);    motor->conjuntoSalida(12,0,2);
    motor->conjuntoEntrada(13,0,2);
    motor->conjuntoEntrada(13,1,3);    motor->conjuntoSalida(13,0,1);
    motor->conjuntoEntrada(14,0,2);
    motor->conjuntoEntrada(14,1,4);    motor->conjuntoSalida(14,0,1);
    motor->conjuntoEntrada(15,0,3);
    motor->conjuntoEntrada(15,1,0);    motor->conjuntoSalida(15,0,4);
    motor->conjuntoEntrada(16,0,3);
    motor->conjuntoEntrada(16,1,1);    motor->conjuntoSalida(16,0,3);
    motor->conjuntoEntrada(17,0,3);
    motor->conjuntoEntrada(17,1,2);    motor->conjuntoSalida(17,0,3);
    motor->conjuntoEntrada(18,0,3);
    motor->conjuntoEntrada(18,1,3);    motor->conjuntoSalida(18,0,2);
    motor->conjuntoEntrada(19,0,3);
    motor->conjuntoEntrada(19,1,4);    motor->conjuntoSalida(19,0,1);
    motor->conjuntoEntrada(20,0,4);
    motor->conjuntoEntrada(20,1,0);    motor->conjuntoSalida(20,0,4);
    motor->conjuntoEntrada(21,0,4);
    motor->conjuntoEntrada(21,1,1);    motor->conjuntoSalida(21,0,4);
    motor->conjuntoEntrada(22,0,4);
    motor->conjuntoEntrada(22,1,2);    motor->conjuntoSalida(22,0,3);
    motor->conjuntoEntrada(23,0,4);
    motor->conjuntoEntrada(23,1,3);    motor->conjuntoSalida(23,0,3);
    motor->conjuntoEntrada(24,0,4);
    motor->conjuntoEntrada(24,1,4);    motor->conjuntoSalida(24,0,2);

// impostazione dei modificatori linguistici delle variabili d'ingresso
// il primo parametro è la regola, il secondo la variabile, il terzo il
// valore del modificatore (1.0 se la variabile non è modificata)
    motor->modificador(0,0,1.000000);    motor->modificador(0,1,1.000000);
    motor->modificador(1,0,1.000000);    motor->modificador(1,1,1.000000);
    motor->modificador(2,0,1.000000);    motor->modificador(2,1,1.000000);
    motor->modificador(3,0,1.000000);    motor->modificador(3,1,1.000000);
    motor->modificador(4,0,1.000000);    motor->modificador(4,1,1.000000);
    motor->modificador(5,0,1.000000);    motor->modificador(5,1,1.000000);
    motor->modificador(6,0,1.000000);    motor->modificador(6,1,1.000000);
    motor->modificador(7,0,1.000000);    motor->modificador(7,1,1.000000);
    motor->modificador(8,0,1.000000);    motor->modificador(8,1,1.000000);
    motor->modificador(9,0,1.000000);    motor->modificador(9,1,1.000000);
    motor->modificador(10,0,1.000000);    motor->modificador(10,1,1.000000);
    motor->modificador(11,0,1.000000);    motor->modificador(11,1,1.000000);
    motor->modificador(12,0,1.000000);    motor->modificador(12,1,1.000000);
    motor->modificador(13,0,1.000000);    motor->modificador(13,1,1.000000);
    motor->modificador(14,0,1.000000);    motor->modificador(14,1,1.000000);
    motor->modificador(15,0,1.000000);    motor->modificador(15,1,1.000000);
    motor->modificador(16,0,1.000000);    motor->modificador(16,1,1.000000);
    motor->modificador(17,0,1.000000);    motor->modificador(17,1,1.000000);
    motor->modificador(18,0,1.000000);    motor->modificador(18,1,1.000000);
    motor->modificador(19,0,1.000000);    motor->modificador(19,1,1.000000);
    motor->modificador(20,0,1.000000);    motor->modificador(20,1,1.000000);
    motor->modificador(21,0,1.000000);    motor->modificador(21,1,1.000000);
```

```
    motor->modificador(22,0,1.000000);    motor->modificador(22,1,1.000000);
    motor->modificador(23,0,1.000000);    motor->modificador(23,1,1.000000);
    motor->modificador(24,0,1.000000);    motor->modificador(24,1,1.000000);

// impostazione degli ultimi parametri del sistema
    concreto=new BloqueConcrecion(motor);
    Conjuncion=new Maximo();         // scelta dell'operatore per la congiunzione
                                     // degli insiemi inferiti dalle regole
    conc=new Altura(motor,0,Conjuncion);    // scelta dell'operatore di defuzzificazione
    concreto->adicionarConcresor(conc);
    concreto->motor(motor);
    concreto->conjuncion(Conjuncion);
}

// DISTRUTTORE DELLA CLASSE; si impiega per disallocare le celle di memoria

fz1::~fz1()
{
    delete concreto;
    delete motor;
    delete entradas;
    delete salidas;
}


//blocco fz2
//IN1: VT
//IN2: VD
//OUT: SS

// DICHIARAZIONE DELLA CLASSE fz2 (CHE RAPPRESENTA IL BLOCCO FUZZY 2)
// le modalità usate per la classe fz1 sono valide per tutte le 8 classi
// relative agli 8 blocchi fuzzy che costituiscono il sistema

class fz2:public SistemaLogicaDifusa
{
public:
    fz2();
    ~fz2();
protected:
};

fz2::fz2()
{
    ConjuntoDifuso *cd;
    Difusor *dif;
    Variable *var;
    Norma *And;
    Norma *Composicion;
    Norma *Conjuncion;
    Implicacion *Implica;
    Concresor *conc;

    entradas=new Universo(2);
    salidas=new Universo(1);

    var=new Variable(5);
    var->rangoMinimo(0.000000);
    var->rangoMaximo(1.000000);
    cd=new ConjuntoL(0.000000,0.166667,0.333333);
    var->adicionarConjuntos(cd);
    cd=new ConjuntoTriangulo(0.166667,0.333333,0.500000);
    var->adicionarConjuntos(cd);
    cd=new ConjuntoTriangulo(0.333333,0.500000,0.666667);
    var->adicionarConjuntos(cd);
    cd=new ConjuntoTriangulo(0.500000,0.666667,0.833333);
    var->adicionarConjuntos(cd);
    cd=new ConjuntoGamma(0.666667,0.833333,1.000000);
    var->adicionarConjuntos(cd);
    dif=new DifusorPi(0.500000,0.200000,0.100000,0.100000,0.200000);
    dif->numeroPuntos(5);
    var->difusorEntrada(dif);
    var->nombreVariable("Variable 1");
    var->numeroIntervalos(30);
```

```
entradas->adicionarVariable(var);
var=new Variable(5);
var->rangoMinimo(0.000000);
var->rangoMaximo(1.000000);
cd=new ConjuntoL(0.000000,0.166667,0.333333);
var->adicionarConjuntos(cd);
cd=new ConjuntoTriangulo(0.166667,0.333333,0.500000);
var->adicionarConjuntos(cd);
cd=new ConjuntoTriangulo(0.333333,0.500000,0.666667);
var->adicionarConjuntos(cd);
cd=new ConjuntoTriangulo(0.500000,0.666667,0.833333);
var->adicionarConjuntos(cd);
cd=new ConjuntoGamma(0.666667,0.833333,1.000000);
var->adicionarConjuntos(cd);
dif=new DifusorPi(0.500000,0.200000,0.100000,0.100000,0.200000);
dif->numeroPuntos(5);
var->difusorEntrada(dif);
var->nombreVariable("Variable 2");
var->numeroIntervalos(30);
entradas->adicionarVariable(var);
var=new Variable(5);
var->rangoMinimo(0.000000);
var->rangoMaximo(1.000000);
cd=new ConjuntoL(0.000000,0.000000,0.250000);
var->adicionarConjuntos(cd);
cd=new ConjuntoTriangulo(0.000000,0.250000,0.500000);
var->adicionarConjuntos(cd);
cd=new ConjuntoTriangulo(0.250000,0.500000,0.750000);
var->adicionarConjuntos(cd);
cd=new ConjuntoTriangulo(0.500000,0.750000,1.000000);
var->adicionarConjuntos(cd);
cd=new ConjuntoGamma(0.750000,1.000000,1.000000);
var->adicionarConjuntos(cd);
var->nombreVariable("Uscita");
var->numeroIntervalos(292);
salidas->adicionarVariable(var);
motor=new MaquinaInferencia(entradas,salidas,25);
And=new Producto();
Composicion=new Producto();
Implica=new ImplicacionMinimo();
motor->and(And);
motor->composicion(Composicion);
motor->implicacion(Implica);

motor->conjuntoEntrada(0,0,0);
motor->conjuntoEntrada(0,1,0);    motor->conjuntoSalida(0,0,0);
motor->conjuntoEntrada(1,0,0);
motor->conjuntoEntrada(1,1,1);    motor->conjuntoSalida(1,0,1);
motor->conjuntoEntrada(2,0,0);
motor->conjuntoEntrada(2,1,2);    motor->conjuntoSalida(2,0,1);
motor->conjuntoEntrada(3,0,0);
motor->conjuntoEntrada(3,1,3);    motor->conjuntoSalida(3,0,2);
motor->conjuntoEntrada(4,0,0);
motor->conjuntoEntrada(4,1,4);    motor->conjuntoSalida(4,0,2);
motor->conjuntoEntrada(5,0,1);
motor->conjuntoEntrada(5,1,0);    motor->conjuntoSalida(5,0,1);
motor->conjuntoEntrada(6,0,1);
motor->conjuntoEntrada(6,1,1);    motor->conjuntoSalida(6,0,1);
motor->conjuntoEntrada(7,0,1);
motor->conjuntoEntrada(7,1,2);    motor->conjuntoSalida(7,0,2);
motor->conjuntoEntrada(8,0,1);
motor->conjuntoEntrada(8,1,3);    motor->conjuntoSalida(8,0,2);
motor->conjuntoEntrada(9,0,1);
motor->conjuntoEntrada(9,1,4);    motor->conjuntoSalida(9,0,3);
motor->conjuntoEntrada(10,0,2);
motor->conjuntoEntrada(10,1,0);    motor->conjuntoSalida(10,0,1);
motor->conjuntoEntrada(11,0,2);
motor->conjuntoEntrada(11,1,1);    motor->conjuntoSalida(11,0,2);
motor->conjuntoEntrada(12,0,2);
motor->conjuntoEntrada(12,1,2);    motor->conjuntoSalida(12,0,2);
motor->conjuntoEntrada(13,0,2);
motor->conjuntoEntrada(13,1,3);    motor->conjuntoSalida(13,0,3);
motor->conjuntoEntrada(14,0,2);
motor->conjuntoEntrada(14,1,4);    motor->conjuntoSalida(14,0,3);
```

```
    motor->conjuntoEntrada(15,0,3);
    motor->conjuntoEntrada(15,1,0);    motor->conjuntoSalida(15,0,2);
    motor->conjuntoEntrada(16,0,3);
    motor->conjuntoEntrada(16,1,1);    motor->conjuntoSalida(16,0,2);
    motor->conjuntoEntrada(17,0,3);
    motor->conjuntoEntrada(17,1,2);    motor->conjuntoSalida(17,0,3);
    motor->conjuntoEntrada(18,0,3);
    motor->conjuntoEntrada(18,1,3);    motor->conjuntoSalida(18,0,3);
    motor->conjuntoEntrada(19,0,3);
    motor->conjuntoEntrada(19,1,4);    motor->conjuntoSalida(19,0,3);
    motor->conjuntoEntrada(20,0,4);
    motor->conjuntoEntrada(20,1,0);    motor->conjuntoSalida(20,0,2);
    motor->conjuntoEntrada(21,0,4);
    motor->conjuntoEntrada(21,1,1);    motor->conjuntoSalida(21,0,3);
    motor->conjuntoEntrada(22,0,4);
    motor->conjuntoEntrada(22,1,2);    motor->conjuntoSalida(22,0,3);
    motor->conjuntoEntrada(23,0,4);
    motor->conjuntoEntrada(23,1,3);    motor->conjuntoSalida(23,0,3);
    motor->conjuntoEntrada(24,0,4);
    motor->conjuntoEntrada(24,1,4);    motor->conjuntoSalida(24,0,4);

    motor->modificador(0,0,1.000000);    motor->modificador(0,1,1.000000);
    motor->modificador(1,0,1.000000);    motor->modificador(1,1,1.000000);
    motor->modificador(2,0,1.000000);    motor->modificador(2,1,1.000000);
    motor->modificador(3,0,1.000000);    motor->modificador(3,1,1.000000);
    motor->modificador(4,0,1.000000);    motor->modificador(4,1,1.000000);
    motor->modificador(5,0,1.000000);    motor->modificador(5,1,1.000000);
    motor->modificador(6,0,1.000000);    motor->modificador(6,1,1.000000);
    motor->modificador(7,0,1.000000);    motor->modificador(7,1,1.000000);
    motor->modificador(8,0,1.000000);    motor->modificador(8,1,1.000000);
    motor->modificador(9,0,1.000000);    motor->modificador(9,1,1.000000);
    motor->modificador(10,0,1.000000);    motor->modificador(10,1,1.000000);
    motor->modificador(11,0,1.000000);    motor->modificador(11,1,1.000000);
    motor->modificador(12,0,1.000000);    motor->modificador(12,1,1.000000);
    motor->modificador(13,0,1.000000);    motor->modificador(13,1,1.000000);
    motor->modificador(14,0,1.000000);    motor->modificador(14,1,1.000000);
    motor->modificador(15,0,1.000000);    motor->modificador(15,1,1.000000);
    motor->modificador(16,0,1.000000);    motor->modificador(16,1,1.000000);
    motor->modificador(17,0,1.000000);    motor->modificador(17,1,1.000000);
    motor->modificador(18,0,1.000000);    motor->modificador(18,1,1.000000);
    motor->modificador(19,0,1.000000);    motor->modificador(19,1,1.000000);
    motor->modificador(20,0,1.000000);    motor->modificador(20,1,1.000000);
    motor->modificador(21,0,1.000000);    motor->modificador(21,1,1.000000);
    motor->modificador(22,0,1.000000);    motor->modificador(22,1,1.000000);
    motor->modificador(23,0,1.000000);    motor->modificador(23,1,1.000000);
    motor->modificador(24,0,1.000000);    motor->modificador(24,1,1.000000);

    concreto=new BloqueConcrecion(motor);
    Conjuncion=new Maximo();
    conc=new Altura(motor,0,Conjuncion);
    concreto->adicionarConcresor(conc);
    concreto->motor(motor);
    concreto->conjuncion(Conjuncion);
}

fz2::~fz2()
{
    delete concreto;
    delete motor;
    delete entradas;
    delete salidas;
}


//blocco fz3
//IN1: SS
//IN2: FG
//OUT: SA

class fz3:public SistemaLogicaDifusa
{
public:
    fz3();
    ~fz3();
```

```
protected:
};

fz3::fz3()
{
    ConjuntoDifuso *cd;
    Difusor *dif;
    Variable *var;
    Norma *And;
    Norma *Composicion;
    Norma *Conjuncion;
    Implicacion *Implica;
    Concresor *conc;

    entradas=new Universo(2);
    salidas=new Universo(1);

    var=new Variable(5);
    var->rangoMinimo(0.000000);
    var->rangoMaximo(1.000000);
    cd=new ConjuntoL(0.000000,0.166667,0.333333);
    var->adicionarConjuntos(cd);
    cd=new ConjuntoTriangulo(0.166667,0.333333,0.500000);
    var->adicionarConjuntos(cd);
    cd=new ConjuntoTriangulo(0.333333,0.500000,0.666667);
    var->adicionarConjuntos(cd);
    cd=new ConjuntoTriangulo(0.500000,0.666667,0.833333);
    var->adicionarConjuntos(cd);
    cd=new ConjuntoGamma(0.666667,0.833333,1.000000);
    var->adicionarConjuntos(cd);
    dif=new DifusorPi(0.500000,0.200000,0.100000,0.100000,0.200000);
    dif->numeroPuntos(5);
    var->difusorEntrada(dif);
    var->nombreVariable("Variable 1");
    var->numeroIntervalos(30);
    entradas->adicionarVariable(var);
    var=new Variable(5);
    var->rangoMinimo(0.000000);
    var->rangoMaximo(1.000000);
    cd=new ConjuntoL(0.000000,0.166667,0.333333);
    var->adicionarConjuntos(cd);
    cd=new ConjuntoTriangulo(0.166667,0.333333,0.500000);
    var->adicionarConjuntos(cd);
    cd=new ConjuntoTriangulo(0.333333,0.500000,0.666667);
    var->adicionarConjuntos(cd);
    cd=new ConjuntoTriangulo(0.500000,0.666667,0.833333);
    var->adicionarConjuntos(cd);
    cd=new ConjuntoGamma(0.666667,0.833333,1.000000);
    var->adicionarConjuntos(cd);
    dif=new DifusorPi(0.500000,0.200000,0.100000,0.100000,0.200000);
    dif->numeroPuntos(5);
    var->difusorEntrada(dif);
    var->nombreVariable("Variable 2");
    var->numeroIntervalos(30);
    entradas->adicionarVariable(var);
    var=new Variable(5);
    var->rangoMinimo(0.000000);
    var->rangoMaximo(1.000000);
    cd=new ConjuntoL(0.000000,0.000000,0.250000);
    var->adicionarConjuntos(cd);
    cd=new ConjuntoTriangulo(0.000000,0.250000,0.500000);
    var->adicionarConjuntos(cd);
    cd=new ConjuntoTriangulo(0.250000,0.500000,0.750000);
    var->adicionarConjuntos(cd);
    cd=new ConjuntoTriangulo(0.500000,0.750000,1.000000);
    var->adicionarConjuntos(cd);
    cd=new ConjuntoGamma(0.750000,1.000000,1.000000);
    var->adicionarConjuntos(cd);
    var->nombreVariable("Uscita");
    var->numeroIntervalos(292);
    salidas->adicionarVariable(var);
    motor=new MaquinaInferencia(entradas,salidas,25);
    And=new Producto();
    Composicion=new Producto();
```

```
Implica=new ImplicacionMinimo();
motor->and(And);
motor->composicion(Composicion);
motor->implicacion(Implica);

motor->conjuntoEntrada(0,0,0);
motor->conjuntoEntrada(0,1,0);    motor->conjuntoSalida(0,0,0);
motor->conjuntoEntrada(1,0,0);
motor->conjuntoEntrada(1,1,1);    motor->conjuntoSalida(1,0,0);
motor->conjuntoEntrada(2,0,0);
motor->conjuntoEntrada(2,1,2);    motor->conjuntoSalida(2,0,0);
motor->conjuntoEntrada(3,0,0);
motor->conjuntoEntrada(3,1,3);    motor->conjuntoSalida(3,0,0);
motor->conjuntoEntrada(4,0,0);
motor->conjuntoEntrada(4,1,4);    motor->conjuntoSalida(4,0,0);
motor->conjuntoEntrada(5,0,1);
motor->conjuntoEntrada(5,1,0);    motor->conjuntoSalida(5,0,1);
motor->conjuntoEntrada(6,0,1);
motor->conjuntoEntrada(6,1,1);    motor->conjuntoSalida(6,0,1);
motor->conjuntoEntrada(7,0,1);
motor->conjuntoEntrada(7,1,2);    motor->conjuntoSalida(7,0,1);
motor->conjuntoEntrada(8,0,1);
motor->conjuntoEntrada(8,1,3);    motor->conjuntoSalida(8,0,2);
motor->conjuntoEntrada(9,0,1);
motor->conjuntoEntrada(9,1,4);    motor->conjuntoSalida(9,0,2);
motor->conjuntoEntrada(10,0,2);
motor->conjuntoEntrada(10,1,0);    motor->conjuntoSalida(10,0,1);
motor->conjuntoEntrada(11,0,2);
motor->conjuntoEntrada(11,1,1);    motor->conjuntoSalida(11,0,1);
motor->conjuntoEntrada(12,0,2);
motor->conjuntoEntrada(12,1,2);    motor->conjuntoSalida(12,0,2);
motor->conjuntoEntrada(13,0,2);
motor->conjuntoEntrada(13,1,3);    motor->conjuntoSalida(13,0,3);
motor->conjuntoEntrada(14,0,2);
motor->conjuntoEntrada(14,1,4);    motor->conjuntoSalida(14,0,3);
motor->conjuntoEntrada(15,0,3);
motor->conjuntoEntrada(15,1,0);    motor->conjuntoSalida(15,0,1);
motor->conjuntoEntrada(16,0,3);
motor->conjuntoEntrada(16,1,1);    motor->conjuntoSalida(16,0,2);
motor->conjuntoEntrada(17,0,3);
motor->conjuntoEntrada(17,1,2);    motor->conjuntoSalida(17,0,3);
motor->conjuntoEntrada(18,0,3);
motor->conjuntoEntrada(18,1,3);    motor->conjuntoSalida(18,0,3);
motor->conjuntoEntrada(19,0,3);
motor->conjuntoEntrada(19,1,4);    motor->conjuntoSalida(19,0,4);
motor->conjuntoEntrada(20,0,4);
motor->conjuntoEntrada(20,1,0);    motor->conjuntoSalida(20,0,2);
motor->conjuntoEntrada(21,0,4);
motor->conjuntoEntrada(21,1,1);    motor->conjuntoSalida(21,0,3);
motor->conjuntoEntrada(22,0,4);
motor->conjuntoEntrada(22,1,2);    motor->conjuntoSalida(22,0,4);
motor->conjuntoEntrada(23,0,4);
motor->conjuntoEntrada(23,1,3);    motor->conjuntoSalida(23,0,4);
motor->conjuntoEntrada(24,0,4);
motor->conjuntoEntrada(24,1,4);    motor->conjuntoSalida(24,0,4);

motor->modificador(0,0,1.000000);    motor->modificador(0,1,1.000000);
motor->modificador(1,0,1.000000);    motor->modificador(1,1,1.000000);
motor->modificador(2,0,1.000000);    motor->modificador(2,1,1.000000);
motor->modificador(3,0,1.000000);    motor->modificador(3,1,1.000000);
motor->modificador(4,0,1.000000);    motor->modificador(4,1,1.000000);
motor->modificador(5,0,1.000000);    motor->modificador(5,1,1.000000);
motor->modificador(6,0,1.000000);    motor->modificador(6,1,1.000000);
motor->modificador(7,0,1.000000);    motor->modificador(7,1,1.000000);
motor->modificador(8,0,1.000000);    motor->modificador(8,1,1.000000);
motor->modificador(9,0,1.000000);    motor->modificador(9,1,1.000000);
motor->modificador(10,0,1.000000);    motor->modificador(10,1,1.000000);
motor->modificador(11,0,1.000000);    motor->modificador(11,1,1.000000);
motor->modificador(12,0,1.000000);    motor->modificador(12,1,1.000000);
motor->modificador(13,0,1.000000);    motor->modificador(13,1,1.000000);
motor->modificador(14,0,1.000000);    motor->modificador(14,1,1.000000);
motor->modificador(15,0,1.000000);    motor->modificador(15,1,1.000000);
motor->modificador(16,0,1.000000);    motor->modificador(16,1,1.000000);
motor->modificador(17,0,1.000000);    motor->modificador(17,1,1.000000);
```

```
    motor->modificador(18,0,1.000000);    motor->modificador(18,1,1.000000);
    motor->modificador(19,0,1.000000);    motor->modificador(19,1,1.000000);
    motor->modificador(20,0,1.000000);    motor->modificador(20,1,1.000000);
    motor->modificador(21,0,1.000000);    motor->modificador(21,1,1.000000);
    motor->modificador(22,0,1.000000);    motor->modificador(22,1,1.000000);
    motor->modificador(23,0,1.000000);    motor->modificador(23,1,1.000000);
    motor->modificador(24,0,1.000000);    motor->modificador(24,1,1.000000);

    concreto=new BloqueConcrecion(motor);
    Conjuncion=new Maximo();
    conc=new Altura(motor,0,Conjuncion);
    concreto->adicionarConcresor(conc);
    concreto->motor(motor);
    concreto->conjuncion(Conjuncion);
}

fz3::~fz3()
{
    delete concreto;
    delete motor;
    delete entradas;
    delete salidas;
}


//blocco fz4
//IN1: DA
//IN2: SA
//OUT: SB

class fz4:public SistemaLogicaDifusa
{
public:
    fz4();
    ~fz4();
protected:
};

fz4::fz4()
{
    ConjuntoDifuso *cd;
    Difusor *dif;
    Variable *var;
    Norma *And;
    Norma *Composicion;
    Norma *Conjuncion;
    Implicacion *Implica;
    Concresor *conc;

    entradas=new Universo(2);
    salidas=new Universo(1);

    var=new Variable(5);
    var->rangoMinimo(0.000000);
    var->rangoMaximo(1.000000);
    cd=new ConjuntoL(0.000000,0.000000,0.250000);
    var->adicionarConjuntos(cd);
    cd=new ConjuntoTriangulo(0.000000,0.250000,0.500000);
    var->adicionarConjuntos(cd);
    cd=new ConjuntoTriangulo(0.250000,0.500000,0.750000);
    var->adicionarConjuntos(cd);
    cd=new ConjuntoTriangulo(0.500000,0.750000,1.000000);
    var->adicionarConjuntos(cd);
    cd=new ConjuntoGamma(0.750000,1.000000,1.000000);
    var->adicionarConjuntos(cd);
    dif=new DifusorPi(0.500000,0.200000,0.100000,0.100000,0.200000);
    dif->numeroPuntos(5);
    var->difusorEntrada(dif);
    var->nombreVariable("Variable 1");
    var->numeroIntervalos(30);
    entradas->adicionarVariable(var);
    var=new Variable(5);
    var->rangoMinimo(0.000000);
    var->rangoMaximo(1.000000);
```

```
cd=new ConjuntoL(0.000000,0.166667,0.333333);
var->adicionarConjuntos(cd);
cd=new ConjuntoTriangulo(0.166667,0.333333,0.500000);
var->adicionarConjuntos(cd);
cd=new ConjuntoTriangulo(0.333333,0.500000,0.666667);
var->adicionarConjuntos(cd);
cd=new ConjuntoTriangulo(0.500000,0.666667,0.833333);
var->adicionarConjuntos(cd);
cd=new ConjuntoGamma(0.666667,0.833333,1.000000);
var->adicionarConjuntos(cd);
dif=new DifusorPi(0.500000,0.200000,0.100000,0.100000,0.200000);
dif->numeroPuntos(5);
var->difusorEntrada(dif);
var->nombreVariable("Variable 2");
var->numeroIntervalos(30);
entradas->adicionarVariable(var);
var=new Variable(5);
var->rangoMinimo(0.000000);
var->rangoMaximo(1.000000);
cd=new ConjuntoL(0.000000,0.000000,0.250000);
var->adicionarConjuntos(cd);
cd=new ConjuntoTriangulo(0.000000,0.250000,0.500000);
var->adicionarConjuntos(cd);
cd=new ConjuntoTriangulo(0.250000,0.500000,0.750000);
var->adicionarConjuntos(cd);
cd=new ConjuntoTriangulo(0.500000,0.750000,1.000000);
var->adicionarConjuntos(cd);
cd=new ConjuntoGamma(0.750000,1.000000,1.000000);
var->adicionarConjuntos(cd);
var->nombreVariable("Uscita");
var->numeroIntervalos(292);
salidas->adicionarVariable(var);
motor=new MaquinaInferencia(entradas,salidas,25);
And=new Producto();
Composicion=new Producto();
Implica=new ImplicacionMinimo();
motor->and(And);
motor->composicion(Composicion);
motor->implicacion(Implica);

motor->conjuntoEntrada(0,0,0);
motor->conjuntoEntrada(0,1,0);    motor->conjuntoSalida(0,0,0);
motor->conjuntoEntrada(1,0,0);
motor->conjuntoEntrada(1,1,1);    motor->conjuntoSalida(1,0,0);
motor->conjuntoEntrada(2,0,0);
motor->conjuntoEntrada(2,1,2);    motor->conjuntoSalida(2,0,1);
motor->conjuntoEntrada(3,0,0);
motor->conjuntoEntrada(3,1,3);    motor->conjuntoSalida(3,0,1);
motor->conjuntoEntrada(4,0,0);
motor->conjuntoEntrada(4,1,4);    motor->conjuntoSalida(4,0,1);
motor->conjuntoEntrada(5,0,1);
motor->conjuntoEntrada(5,1,0);    motor->conjuntoSalida(5,0,1);
motor->conjuntoEntrada(6,0,1);
motor->conjuntoEntrada(6,1,1);    motor->conjuntoSalida(6,0,1);
motor->conjuntoEntrada(7,0,1);
motor->conjuntoEntrada(7,1,2);    motor->conjuntoSalida(7,0,2);
motor->conjuntoEntrada(8,0,1);
motor->conjuntoEntrada(8,1,3);    motor->conjuntoSalida(8,0,2);
motor->conjuntoEntrada(9,0,1);
motor->conjuntoEntrada(9,1,4);    motor->conjuntoSalida(9,0,2);
motor->conjuntoEntrada(10,0,2);
motor->conjuntoEntrada(10,1,0);    motor->conjuntoSalida(10,0,2);
motor->conjuntoEntrada(11,0,2);
motor->conjuntoEntrada(11,1,1);    motor->conjuntoSalida(11,0,2);
motor->conjuntoEntrada(12,0,2);
motor->conjuntoEntrada(12,1,2);    motor->conjuntoSalida(12,0,3);
motor->conjuntoEntrada(13,0,2);
motor->conjuntoEntrada(13,1,3);    motor->conjuntoSalida(13,0,3);
motor->conjuntoEntrada(14,0,3);
motor->conjuntoEntrada(14,1,4);    motor->conjuntoSalida(14,0,3);
motor->conjuntoEntrada(15,0,3);
motor->conjuntoEntrada(15,1,0);    motor->conjuntoSalida(15,0,2);
motor->conjuntoEntrada(16,0,3);
motor->conjuntoEntrada(16,1,1);    motor->conjuntoSalida(16,0,3);
```

```
    motor->conjuntoEntrada(17,0,3);
    motor->conjuntoEntrada(17,1,2);    motor->conjuntoSalida(17,0,3);
    motor->conjuntoEntrada(18,0,3);
    motor->conjuntoEntrada(18,1,3);    motor->conjuntoSalida(18,0,3);
    motor->conjuntoEntrada(19,0,3);
    motor->conjuntoEntrada(19,1,4);    motor->conjuntoSalida(19,0,4);
    motor->conjuntoEntrada(20,0,4);
    motor->conjuntoEntrada(20,1,0);    motor->conjuntoSalida(20,0,3);
    motor->conjuntoEntrada(21,0,4);
    motor->conjuntoEntrada(21,1,1);    motor->conjuntoSalida(21,0,3);
    motor->conjuntoEntrada(22,0,4);
    motor->conjuntoEntrada(22,1,2);    motor->conjuntoSalida(22,0,4);
    motor->conjuntoEntrada(23,0,4);
    motor->conjuntoEntrada(23,1,3);    motor->conjuntoSalida(23,0,4);
    motor->conjuntoEntrada(24,0,4);
    motor->conjuntoEntrada(24,1,4);    motor->conjuntoSalida(24,0,4);

    motor->modificador(0,0,1.000000);    motor->modificador(0,1,1.000000);
    motor->modificador(1,0,1.000000);    motor->modificador(1,1,1.000000);
    motor->modificador(2,0,1.000000);    motor->modificador(2,1,1.000000);
    motor->modificador(3,0,1.000000);    motor->modificador(3,1,1.000000);
    motor->modificador(4,0,1.000000);    motor->modificador(4,1,1.000000);
    motor->modificador(5,0,1.000000);    motor->modificador(5,1,1.000000);
    motor->modificador(6,0,1.000000);    motor->modificador(6,1,1.000000);
    motor->modificador(7,0,1.000000);    motor->modificador(7,1,1.000000);
    motor->modificador(8,0,1.000000);    motor->modificador(8,1,1.000000);
    motor->modificador(9,0,1.000000);    motor->modificador(9,1,1.000000);
    motor->modificador(10,0,1.000000);    motor->modificador(10,1,1.000000);
    motor->modificador(11,0,1.000000);    motor->modificador(11,1,1.000000);
    motor->modificador(12,0,1.000000);    motor->modificador(12,1,1.000000);
    motor->modificador(13,0,1.000000);    motor->modificador(13,1,1.000000);
    motor->modificador(14,0,1.000000);    motor->modificador(14,1,1.000000);
    motor->modificador(15,0,1.000000);    motor->modificador(15,1,1.000000);
    motor->modificador(16,0,1.000000);    motor->modificador(16,1,1.000000);
    motor->modificador(17,0,1.000000);    motor->modificador(17,1,1.000000);
    motor->modificador(18,0,1.000000);    motor->modificador(18,1,1.000000);
    motor->modificador(19,0,1.000000);    motor->modificador(19,1,1.000000);
    motor->modificador(20,0,1.000000);    motor->modificador(20,1,1.000000);
    motor->modificador(21,0,1.000000);    motor->modificador(21,1,1.000000);
    motor->modificador(22,0,1.000000);    motor->modificador(22,1,1.000000);
    motor->modificador(23,0,1.000000);    motor->modificador(23,1,1.000000);
    motor->modificador(24,0,1.000000);    motor->modificador(24,1,1.000000);

    concreto=new BloqueConcrecion(motor);
    Conjuncion=new Maximo();
    conc=new Altura(motor,0,Conjuncion);
    concreto->adicionarConcresor(conc);
    concreto->motor(motor);
    concreto->conjuncion(Conjuncion);
}

fz4::~fz4()
{
    delete concreto;
    delete motor;
    delete entradas;
    delete salidas;
}


//blocco fz5
//IN1: SB
//IN2: OF
//OUT: LR

class fz5:public SistemaLogicaDifusa
{
public:
    fz5();
    ~fz5();
protected:
};

fz5::fz5()
```

```
{
    ConjuntoDifuso *cd;
    Difusor *dif;
    Variable *var;
    Norma *And;
    Norma *Composicion;
    Norma *Conjuncion;
    Implicacion *Implica;
    Concresor *conc;

    entradas=new Universo(2);
    salidas=new Universo(1);

    var=new Variable(5);
    var->rangoMinimo(0.000000);
    var->rangoMaximo(1.000000);
    cd=new ConjuntoL(0.000000,0.100000,0.300000);
    var->adicionarConjuntos(cd);
    cd=new ConjuntoTriangulo(0.100000,0.300000,0.500000);
    var->adicionarConjuntos(cd);
    cd=new ConjuntoTriangulo(0.300000,0.500000,0.700000);
    var->adicionarConjuntos(cd);
    cd=new ConjuntoTriangulo(0.500000,0.700000,0.900000);
    var->adicionarConjuntos(cd);
    cd=new ConjuntoGamma(0.700000,0.900000,1.000000);
    var->adicionarConjuntos(cd);
    dif=new DifusorPi(0.500000,0.100000,0.050000,0.050000,0.100000);
    dif->numeroPuntos(5);
    var->difusorEntrada(dif);
    var->nombreVariable("Variable 1");
    var->numeroIntervalos(30);
    entradas->adicionarVariable(var);
    var=new Variable(5);
    var->rangoMinimo(0.000000);
    var->rangoMaximo(1.000000);
    cd=new ConjuntoL(0.000000,0.166667,0.333333);
    var->adicionarConjuntos(cd);
    cd=new ConjuntoTriangulo(0.166667,0.333333,0.500000);
    var->adicionarConjuntos(cd);
    cd=new ConjuntoTriangulo(0.333333,0.500000,0.666667);
    var->adicionarConjuntos(cd);
    cd=new ConjuntoTriangulo(0.500000,0.666667,0.833333);
    var->adicionarConjuntos(cd);
    cd=new ConjuntoGamma(0.666667,0.833333,1.000000);
    var->adicionarConjuntos(cd);
    dif=new DifusorPi(0.500000,0.200000,0.100000,0.100000,0.200000);
    dif->numeroPuntos(5);
    var->difusorEntrada(dif);
    var->nombreVariable("Variable 2");
    var->numeroIntervalos(30);
    entradas->adicionarVariable(var);
    var=new Variable(5);
    var->rangoMinimo(0.000000);
    var->rangoMaximo(1.000000);
    cd=new ConjuntoL(0.000000,0.000000,0.250000);
    var->adicionarConjuntos(cd);
    cd=new ConjuntoTriangulo(0.000000,0.250000,0.500000);
    var->adicionarConjuntos(cd);
    cd=new ConjuntoTriangulo(0.250000,0.500000,0.750000);
    var->adicionarConjuntos(cd);
    cd=new ConjuntoTriangulo(0.500000,0.750000,1.000000);
    var->adicionarConjuntos(cd);
    cd=new ConjuntoGamma(0.750000,1.000000,1.000000);
    var->adicionarConjuntos(cd);
    var->nombreVariable("Uscita");
    var->numeroIntervalos(292);
    salidas->adicionarVariable(var);
    motor=new MaquinaInferencia(entradas,salidas,25);
    And=new Producto();
    Composicion=new Producto();
    Implica=new ImplicacionMinimo();
    motor->and(And);
    motor->composicion(Composicion);
    motor->implicacion(Implica);
```

```
motor->conjuntoEntrada(0,0,0);
motor->conjuntoEntrada(0,1,0);    motor->conjuntoSalida(0,0,0);
motor->conjuntoEntrada(1,0,0);
motor->conjuntoEntrada(1,1,1);    motor->conjuntoSalida(1,0,0);
motor->conjuntoEntrada(2,0,0);
motor->conjuntoEntrada(2,1,2);    motor->conjuntoSalida(2,0,0);
motor->conjuntoEntrada(3,0,0);
motor->conjuntoEntrada(3,1,3);    motor->conjuntoSalida(3,0,0);
motor->conjuntoEntrada(4,0,0);
motor->conjuntoEntrada(4,1,4);    motor->conjuntoSalida(4,0,0);
motor->conjuntoEntrada(5,0,1);
motor->conjuntoEntrada(5,1,0);    motor->conjuntoSalida(5,0,1);
motor->conjuntoEntrada(6,0,1);
motor->conjuntoEntrada(6,1,1);    motor->conjuntoSalida(6,0,1);
motor->conjuntoEntrada(7,0,1);
motor->conjuntoEntrada(7,1,2);    motor->conjuntoSalida(7,0,1);
motor->conjuntoEntrada(8,0,1);
motor->conjuntoEntrada(8,1,3);    motor->conjuntoSalida(8,0,0);
motor->conjuntoEntrada(9,0,1);
motor->conjuntoEntrada(9,1,4);    motor->conjuntoSalida(9,0,0);
motor->conjuntoEntrada(10,0,2);
motor->conjuntoEntrada(10,1,0);    motor->conjuntoSalida(10,0,2);
motor->conjuntoEntrada(11,0,2);
motor->conjuntoEntrada(11,1,1);    motor->conjuntoSalida(11,0,2);
motor->conjuntoEntrada(12,0,2);
motor->conjuntoEntrada(12,1,2);    motor->conjuntoSalida(12,0,2);
motor->conjuntoEntrada(13,0,2);
motor->conjuntoEntrada(13,1,3);    motor->conjuntoSalida(13,0,1);
motor->conjuntoEntrada(14,0,2);
motor->conjuntoEntrada(14,1,4);    motor->conjuntoSalida(14,0,1);
motor->conjuntoEntrada(15,0,3);
motor->conjuntoEntrada(15,1,0);    motor->conjuntoSalida(15,0,3);
motor->conjuntoEntrada(16,0,3);
motor->conjuntoEntrada(16,1,1);    motor->conjuntoSalida(16,0,3);
motor->conjuntoEntrada(17,0,3);
motor->conjuntoEntrada(17,1,2);    motor->conjuntoSalida(17,0,3);
motor->conjuntoEntrada(18,0,3);
motor->conjuntoEntrada(18,1,3);    motor->conjuntoSalida(18,0,2);
motor->conjuntoEntrada(19,0,3);
motor->conjuntoEntrada(19,1,4);    motor->conjuntoSalida(19,0,2);
motor->conjuntoEntrada(20,0,4);
motor->conjuntoEntrada(20,1,0);    motor->conjuntoSalida(20,0,4);
motor->conjuntoEntrada(21,0,4);
motor->conjuntoEntrada(21,1,1);    motor->conjuntoSalida(21,0,4);
motor->conjuntoEntrada(22,0,4);
motor->conjuntoEntrada(22,1,2);    motor->conjuntoSalida(22,0,3);
motor->conjuntoEntrada(23,0,4);
motor->conjuntoEntrada(23,1,3);    motor->conjuntoSalida(23,0,3);
motor->conjuntoEntrada(24,0,4);
motor->conjuntoEntrada(24,1,4);    motor->conjuntoSalida(24,0,2);

motor->modificador(0,0,1.000000);    motor->modificador(0,1,1.000000);
motor->modificador(1,0,1.000000);    motor->modificador(1,1,1.000000);
motor->modificador(2,0,1.000000);    motor->modificador(2,1,1.000000);
motor->modificador(3,0,1.000000);    motor->modificador(3,1,1.000000);
motor->modificador(4,0,1.000000);    motor->modificador(4,1,1.000000);
motor->modificador(5,0,1.000000);    motor->modificador(5,1,1.000000);
motor->modificador(6,0,1.000000);    motor->modificador(6,1,1.000000);
motor->modificador(7,0,1.000000);    motor->modificador(7,1,1.000000);
motor->modificador(8,0,1.000000);    motor->modificador(8,1,1.000000);
motor->modificador(9,0,1.000000);    motor->modificador(9,1,1.000000);
motor->modificador(10,0,1.000000);    motor->modificador(10,1,1.000000);
motor->modificador(11,0,1.000000);    motor->modificador(11,1,1.000000);
motor->modificador(12,0,1.000000);    motor->modificador(12,1,1.000000);
motor->modificador(13,0,1.000000);    motor->modificador(13,1,1.000000);
motor->modificador(14,0,1.000000);    motor->modificador(14,1,1.000000);
motor->modificador(15,0,1.000000);    motor->modificador(15,1,1.000000);
motor->modificador(16,0,1.000000);    motor->modificador(16,1,1.000000);
motor->modificador(17,0,1.000000);    motor->modificador(17,1,1.000000);
motor->modificador(18,0,1.000000);    motor->modificador(18,1,1.000000);
motor->modificador(19,0,1.000000);    motor->modificador(19,1,1.000000);
motor->modificador(20,0,1.000000);    motor->modificador(20,1,1.000000);
motor->modificador(21,0,1.000000);    motor->modificador(21,1,1.000000);
```

```
    motor->modificador(22,0,1.000000);    motor->modificador(22,1,1.000000);
    motor->modificador(23,0,1.000000);    motor->modificador(23,1,1.000000);
    motor->modificador(24,0,1.000000);    motor->modificador(24,1,1.000000);

    concreto=new BloqueConcrecion(motor);
    Conjuncion=new Maximo();
    conc=new Altura(motor,0,Conjuncion);
    concreto->adicionarConcresor(conc);
    concreto->motor(motor);
    concreto->conjuncion(Conjuncion);
}

fz5::~fz5()
{
    delete concreto;
    delete motor;
    delete entradas;
    delete salidas;
}


//blocco fz6
//IN1: DR
//IN2: DA
//OUT: RP

class fz6:public SistemaLogicaDifusa
{
public:
    fz6();
    ~fz6();
protected:
};

fz6::fz6()
{
    ConjuntoDifuso *cd;
    Difusor *dif;
    Variable *var;
    Norma *And;
    Norma *Composicion;
    Norma *Conjuncion;
    Implicacion *Implica;
    Concresor *conc;

    entradas=new Universo(2);
    salidas=new Universo(1);

    var=new Variable(5);
    var->rangoMinimo(0.000000);
    var->rangoMaximo(1.000000);
    cd=new ConjuntoL(0.000000,0.166667,0.333333);
    var->adicionarConjuntos(cd);
    cd=new ConjuntoTriangulo(0.166667,0.333333,0.500000);
    var->adicionarConjuntos(cd);
    cd=new ConjuntoTriangulo(0.333333,0.500000,0.666667);
    var->adicionarConjuntos(cd);
    cd=new ConjuntoTriangulo(0.500000,0.666667,0.833333);
    var->adicionarConjuntos(cd);
    cd=new ConjuntoGamma(0.666667,0.833333,1.000000);
    var->adicionarConjuntos(cd);
    dif=new DifusorPi(0.500000,0.200000,0.100000,0.100000,0.200000);
    dif->numeroPuntos(5);
    var->difusorEntrada(dif);
    var->nombreVariable("DR");
    var->numeroIntervalos(30);
    entradas->adicionarVariable(var);
    var=new Variable(5);
    var->rangoMinimo(0.000000);
    var->rangoMaximo(1.000000);
    cd=new ConjuntoL(0.000000,0.166667,0.333333);
    var->adicionarConjuntos(cd);
    cd=new ConjuntoTriangulo(0.166667,0.333333,0.500000);
    var->adicionarConjuntos(cd);
```

```
cd=new ConjuntoTriangulo(0.333333,0.500000,0.666667);
var->adicionarConjuntos(cd);
cd=new ConjuntoTriangulo(0.500000,0.666667,0.833333);
var->adicionarConjuntos(cd);
cd=new ConjuntoGamma(0.666667,0.833333,1.000000);
var->adicionarConjuntos(cd);
dif=new DifusorPi(0.500000,0.200000,0.100000,0.100000,0.200000);
dif->numeroPuntos(5);
var->difusorEntrada(dif);
var->nombreVariable("DA");
var->numeroIntervalos(30);
entradas->adicionarVariable(var);
var=new Variable(5);
var->rangoMinimo(0.000000);
var->rangoMaximo(1.000000);
cd=new ConjuntoL(0.000000,0.166667,0.333333);
var->adicionarConjuntos(cd);
cd=new ConjuntoTriangulo(0.166667,0.333333,0.500000);
var->adicionarConjuntos(cd);
cd=new ConjuntoTriangulo(0.333333,0.500000,0.666667);
var->adicionarConjuntos(cd);
cd=new ConjuntoTriangulo(0.500000,0.666667,0.833333);
var->adicionarConjuntos(cd);
cd=new ConjuntoGamma(0.666667,0.833333,1.000000);
var->adicionarConjuntos(cd);
var->nombreVariable("RP");
var->numeroIntervalos(292);
salidas->adicionarVariable(var);
motor=new MaquinaInferencia(entradas,salidas,25);
And=new Producto();
Composicion=new Producto();
Implica=new ImplicacionMinimo();
motor->and(And);
motor->composicion(Composicion);
motor->implicacion(Implica);

motor->conjuntoEntrada(0,0,0);
motor->conjuntoEntrada(0,1,0);    motor->conjuntoSalida(0,0,4);
motor->conjuntoEntrada(1,0,0);
motor->conjuntoEntrada(1,1,1);    motor->conjuntoSalida(1,0,4);
motor->conjuntoEntrada(2,0,0);
motor->conjuntoEntrada(2,1,2);    motor->conjuntoSalida(2,0,4);
motor->conjuntoEntrada(3,0,0);
motor->conjuntoEntrada(3,1,3);    motor->conjuntoSalida(3,0,4);
motor->conjuntoEntrada(4,0,0);
motor->conjuntoEntrada(4,1,4);    motor->conjuntoSalida(4,0,4);
motor->conjuntoEntrada(5,0,1);
motor->conjuntoEntrada(5,1,0);    motor->conjuntoSalida(5,0,3);
motor->conjuntoEntrada(6,0,1);
motor->conjuntoEntrada(6,1,1);    motor->conjuntoSalida(6,0,3);
motor->conjuntoEntrada(7,0,1);
motor->conjuntoEntrada(7,1,2);    motor->conjuntoSalida(7,0,3);
motor->conjuntoEntrada(8,0,1);
motor->conjuntoEntrada(8,1,3);    motor->conjuntoSalida(8,0,4);
motor->conjuntoEntrada(9,0,1);
motor->conjuntoEntrada(9,1,4);    motor->conjuntoSalida(9,0,4);
motor->conjuntoEntrada(10,0,2);
motor->conjuntoEntrada(10,1,0);    motor->conjuntoSalida(10,0,2);
motor->conjuntoEntrada(11,0,2);
motor->conjuntoEntrada(11,1,1);    motor->conjuntoSalida(11,0,2);
motor->conjuntoEntrada(12,0,2);
motor->conjuntoEntrada(12,1,2);    motor->conjuntoSalida(12,0,2);
motor->conjuntoEntrada(13,0,2);
motor->conjuntoEntrada(13,1,3);    motor->conjuntoSalida(13,0,3);
motor->conjuntoEntrada(14,0,2);
motor->conjuntoEntrada(14,1,4);    motor->conjuntoSalida(14,0,4);
motor->conjuntoEntrada(15,0,3);
motor->conjuntoEntrada(15,1,0);    motor->conjuntoSalida(15,0,1);
motor->conjuntoEntrada(16,0,3);
motor->conjuntoEntrada(16,1,1);    motor->conjuntoSalida(16,0,1);
motor->conjuntoEntrada(17,0,3);
motor->conjuntoEntrada(17,1,2);    motor->conjuntoSalida(17,0,2);
motor->conjuntoEntrada(18,0,3);
motor->conjuntoEntrada(18,1,3);    motor->conjuntoSalida(18,0,2);
```

```
    motor->conjuntoEntrada(19,0,3);
    motor->conjuntoEntrada(19,1,4);    motor->conjuntoSalida(19,0,3);
    motor->conjuntoEntrada(20,0,4);
    motor->conjuntoEntrada(20,1,0);    motor->conjuntoSalida(20,0,0);
    motor->conjuntoEntrada(21,0,4);
    motor->conjuntoEntrada(21,1,1);    motor->conjuntoSalida(21,0,1);
    motor->conjuntoEntrada(22,0,4);
    motor->conjuntoEntrada(22,1,2);    motor->conjuntoSalida(22,0,1);
    motor->conjuntoEntrada(23,0,4);
    motor->conjuntoEntrada(23,1,3);    motor->conjuntoSalida(23,0,2);
    motor->conjuntoEntrada(24,0,4);
    motor->conjuntoEntrada(24,1,4);    motor->conjuntoSalida(24,0,3);

    motor->modificador(0,0,1.000000);    motor->modificador(0,1,1.000000);
    motor->modificador(1,0,1.000000);    motor->modificador(1,1,1.000000);
    motor->modificador(2,0,1.000000);    motor->modificador(2,1,1.000000);
    motor->modificador(3,0,1.000000);    motor->modificador(3,1,1.000000);
    motor->modificador(4,0,1.000000);    motor->modificador(4,1,1.000000);
    motor->modificador(5,0,1.000000);    motor->modificador(5,1,1.000000);
    motor->modificador(6,0,1.000000);    motor->modificador(6,1,1.000000);
    motor->modificador(7,0,1.000000);    motor->modificador(7,1,1.000000);
    motor->modificador(8,0,1.000000);    motor->modificador(8,1,1.000000);
    motor->modificador(9,0,1.000000);    motor->modificador(9,1,1.000000);
    motor->modificador(10,0,1.000000);    motor->modificador(10,1,1.000000);
    motor->modificador(11,0,1.000000);    motor->modificador(11,1,1.000000);
    motor->modificador(12,0,1.000000);    motor->modificador(12,1,1.000000);
    motor->modificador(13,0,1.000000);    motor->modificador(13,1,1.000000);
    motor->modificador(14,0,1.000000);    motor->modificador(14,1,1.000000);
    motor->modificador(15,0,1.000000);    motor->modificador(15,1,1.000000);
    motor->modificador(16,0,1.000000);    motor->modificador(16,1,1.000000);
    motor->modificador(17,0,1.000000);    motor->modificador(17,1,1.000000);
    motor->modificador(18,0,1.000000);    motor->modificador(18,1,1.000000);
    motor->modificador(19,0,1.000000);    motor->modificador(19,1,1.000000);
    motor->modificador(20,0,1.000000);    motor->modificador(20,1,1.000000);
    motor->modificador(21,0,1.000000);    motor->modificador(21,1,1.000000);
    motor->modificador(22,0,1.000000);    motor->modificador(22,1,1.000000);
    motor->modificador(23,0,1.000000);    motor->modificador(23,1,1.000000);
    motor->modificador(24,0,1.000000);    motor->modificador(24,1,1.000000);

    concreto=new BloqueConcrecion(motor);
    Conjuncion=new Maximo();
    conc=new Altura(motor,0,Conjuncion);
    concreto->adicionarConcresor(conc);
    concreto->motor(motor);
    concreto->conjuncion(Conjuncion);
}

fz6::~fz6()
{
    delete concreto;
    delete motor;
    delete entradas;
    delete salidas;
}


//blocco fz7
//IN1: RP
//IN2: FG
//OUT: UO

class fz7:public SistemaLogicaDifusa
{
public:
    fz7();
    ~fz7();
protected:
};

fz7::fz7()
{
    ConjuntoDifuso *cd;
    Difusor *dif;
    Variable *var;
```

```
Norma *And;
Norma *Composicion;
Norma *Conjuncion;
Implicacion *Implica;
Concresor *conc;

entradas=new Universo(2);
salidas=new Universo(1);

var=new Variable(5);
var->rangoMinimo(0.000000);
var->rangoMaximo(1.000000);
cd=new ConjuntoL(0.000000,0.166667,0.333333);
var->adicionarConjuntos(cd);
cd=new ConjuntoTriangulo(0.166667,0.333333,0.500000);
var->adicionarConjuntos(cd);
cd=new ConjuntoTriangulo(0.333333,0.500000,0.666667);
var->adicionarConjuntos(cd);
cd=new ConjuntoTriangulo(0.500000,0.666667,0.833333);
var->adicionarConjuntos(cd);
cd=new ConjuntoGamma(0.666667,0.833333,1.000000);
var->adicionarConjuntos(cd);
dif=new DifusorPi(0.500000,0.200000,0.100000,0.100000,0.200000);
dif->numeroPuntos(5);
var->difusorEntrada(dif);
var->nombreVariable("Variable 1");
var->numeroIntervalos(30);
entradas->adicionarVariable(var);
var=new Variable(5);
var->rangoMinimo(0.000000);
var->rangoMaximo(1.000000);
cd=new ConjuntoL(0.000000,0.166667,0.333333);
var->adicionarConjuntos(cd);
cd=new ConjuntoTriangulo(0.166667,0.333333,0.500000);
var->adicionarConjuntos(cd);
cd=new ConjuntoTriangulo(0.333333,0.500000,0.666667);
var->adicionarConjuntos(cd);
cd=new ConjuntoTriangulo(0.500000,0.666667,0.833333);
var->adicionarConjuntos(cd);
cd=new ConjuntoGamma(0.666667,0.833333,1.000000);
var->adicionarConjuntos(cd);
dif=new DifusorPi(0.500000,0.200000,0.100000,0.100000,0.200000);
dif->numeroPuntos(5);
var->difusorEntrada(dif);
var->nombreVariable("Variable 2");
var->numeroIntervalos(30);
entradas->adicionarVariable(var);
var=new Variable(5);
var->rangoMinimo(0.000000);
var->rangoMaximo(1.000000);
cd=new ConjuntoTriangulo(0.000000,0.166667,0.333333);
var->adicionarConjuntos(cd);
cd=new ConjuntoTriangulo(0.166667,0.333333,0.500000);
var->adicionarConjuntos(cd);
cd=new ConjuntoTriangulo(0.333333,0.500000,0.666667);
var->adicionarConjuntos(cd);
cd=new ConjuntoTriangulo(0.500000,0.666667,0.833333);
var->adicionarConjuntos(cd);
cd=new ConjuntoTriangulo(0.666667,0.833333,1.000000);
var->adicionarConjuntos(cd);
var->nombreVariable("Uscita");
var->numeroIntervalos(292);
salidas->adicionarVariable(var);
motor=new MaquinaInferencia(entradas,salidas,25);
And=new Producto();
Composicion=new Producto();
Implica=new ImplicacionMinimo();
motor->and(And);
motor->composicion(Composicion);
motor->implicacion(Implica);

motor->conjuntoEntrada(0,0,0);
motor->conjuntoEntrada(0,1,0);    motor->conjuntoSalida(0,0,0);
motor->conjuntoEntrada(1,0,0);
```

```
motor->conjuntoEntrada(1,1,1);    motor->conjuntoSalida(1,0,0);
motor->conjuntoEntrada(2,0,0);
motor->conjuntoEntrada(2,1,2);    motor->conjuntoSalida(2,0,1);
motor->conjuntoEntrada(3,0,0);
motor->conjuntoEntrada(3,1,3);    motor->conjuntoSalida(3,0,1);
motor->conjuntoEntrada(4,0,0);
motor->conjuntoEntrada(4,1,4);    motor->conjuntoSalida(4,0,1);
motor->conjuntoEntrada(5,0,1);
motor->conjuntoEntrada(5,1,0);    motor->conjuntoSalida(5,0,0);
motor->conjuntoEntrada(6,0,1);
motor->conjuntoEntrada(6,1,1);    motor->conjuntoSalida(6,0,1);
motor->conjuntoEntrada(7,0,1);
motor->conjuntoEntrada(7,1,2);    motor->conjuntoSalida(7,0,1);
motor->conjuntoEntrada(8,0,1);
motor->conjuntoEntrada(8,1,3);    motor->conjuntoSalida(8,0,2);
motor->conjuntoEntrada(9,0,1);
motor->conjuntoEntrada(9,1,4);    motor->conjuntoSalida(9,0,2);
motor->conjuntoEntrada(10,0,2);
motor->conjuntoEntrada(10,1,0);    motor->conjuntoSalida(10,0,1);
motor->conjuntoEntrada(11,0,2);
motor->conjuntoEntrada(11,1,1);    motor->conjuntoSalida(11,0,2);
motor->conjuntoEntrada(12,0,2);
motor->conjuntoEntrada(12,1,2);    motor->conjuntoSalida(12,0,2);
motor->conjuntoEntrada(13,0,2);
motor->conjuntoEntrada(13,1,3);    motor->conjuntoSalida(13,0,3);
motor->conjuntoEntrada(14,0,2);
motor->conjuntoEntrada(14,1,4);    motor->conjuntoSalida(14,0,3);
motor->conjuntoEntrada(15,0,3);
motor->conjuntoEntrada(15,1,0);    motor->conjuntoSalida(15,0,2);
motor->conjuntoEntrada(16,0,3);
motor->conjuntoEntrada(16,1,1);    motor->conjuntoSalida(16,0,2);
motor->conjuntoEntrada(17,0,3);
motor->conjuntoEntrada(17,1,2);    motor->conjuntoSalida(17,0,3);
motor->conjuntoEntrada(18,0,3);
motor->conjuntoEntrada(18,1,3);    motor->conjuntoSalida(18,0,4);
motor->conjuntoEntrada(19,0,3);
motor->conjuntoEntrada(19,1,4);    motor->conjuntoSalida(19,0,4);
motor->conjuntoEntrada(20,0,4);
motor->conjuntoEntrada(20,1,0);    motor->conjuntoSalida(20,0,3);
motor->conjuntoEntrada(21,0,4);
motor->conjuntoEntrada(21,1,1);    motor->conjuntoSalida(21,0,3);
motor->conjuntoEntrada(22,0,4);
motor->conjuntoEntrada(22,1,2);    motor->conjuntoSalida(22,0,4);
motor->conjuntoEntrada(23,0,4);
motor->conjuntoEntrada(23,1,3);    motor->conjuntoSalida(23,0,4);
motor->conjuntoEntrada(24,0,4);
motor->conjuntoEntrada(24,1,4);    motor->conjuntoSalida(24,0,4);

motor->modificador(0,0,1.000000);    motor->modificador(0,1,1.000000);
motor->modificador(1,0,1.000000);    motor->modificador(1,1,1.000000);
motor->modificador(2,0,1.000000);    motor->modificador(2,1,1.000000);
motor->modificador(3,0,1.000000);    motor->modificador(3,1,1.000000);
motor->modificador(4,0,1.000000);    motor->modificador(4,1,1.000000);
motor->modificador(5,0,1.000000);    motor->modificador(5,1,1.000000);
motor->modificador(6,0,1.000000);    motor->modificador(6,1,1.000000);
motor->modificador(7,0,1.000000);    motor->modificador(7,1,1.000000);
motor->modificador(8,0,1.000000);    motor->modificador(8,1,1.000000);
motor->modificador(9,0,1.000000);    motor->modificador(9,1,1.000000);
motor->modificador(10,0,1.000000);    motor->modificador(10,1,1.000000);
motor->modificador(11,0,1.000000);    motor->modificador(11,1,1.000000);
motor->modificador(12,0,1.000000);    motor->modificador(12,1,1.000000);
motor->modificador(13,0,1.000000);    motor->modificador(13,1,1.000000);
motor->modificador(14,0,1.000000);    motor->modificador(14,1,1.000000);
motor->modificador(15,0,1.000000);    motor->modificador(15,1,1.000000);
motor->modificador(16,0,1.000000);    motor->modificador(16,1,1.000000);
motor->modificador(17,0,1.000000);    motor->modificador(17,1,1.000000);
motor->modificador(18,0,1.000000);    motor->modificador(18,1,1.000000);
motor->modificador(19,0,1.000000);    motor->modificador(19,1,1.000000);
motor->modificador(20,0,1.000000);    motor->modificador(20,1,1.000000);
motor->modificador(21,0,1.000000);    motor->modificador(21,1,1.000000);
motor->modificador(22,0,1.000000);    motor->modificador(22,1,1.000000);
motor->modificador(23,0,1.000000);    motor->modificador(23,1,1.000000);
motor->modificador(24,0,1.000000);    motor->modificador(24,1,1.000000);
```

```
    concreto=new BloqueConcrecion(motor);
    Conjuncion=new Maximo();
    conc=new Altura(motor,0,Conjuncion);
    concreto->adicionarConcresor(conc);
    concreto->motor(motor);
    concreto->conjuncion(Conjuncion);
}

fz7::~fz7()
{
    delete concreto;
    delete motor;
    delete entradas;
    delete salidas;
}


//blocco fz8
//IN1: UO
//IN2: GO
//OUT: EO

class fz8:public SistemaLogicaDifusa
{
public:
    fz8();
    ~fz8();
protected:
};

fz8::fz8()
{
    ConjuntoDifuso *cd;
    Difusor *dif;
    Variable *var;
    Norma *And;
    Norma *Composicion;
    Norma *Conjuncion;
    Implicacion *Implica;
    Concresor *conc;

    entradas=new Universo(2);
    salidas=new Universo(1);

    var=new Variable(5);
    var->rangoMinimo(0.000000);
    var->rangoMaximo(1.000000);
    cd=new ConjuntoL(0.000000,0.166667,0.333333);
    var->adicionarConjuntos(cd);
    cd=new ConjuntoTriangulo(0.166667,0.333333,0.500000);
    var->adicionarConjuntos(cd);
    cd=new ConjuntoTriangulo(0.333333,0.500000,0.666667);
    var->adicionarConjuntos(cd);
    cd=new ConjuntoTriangulo(0.500000,0.666667,0.833333);
    var->adicionarConjuntos(cd);
    cd=new ConjuntoGamma(0.666667,0.833333,1.000000);
    var->adicionarConjuntos(cd);
    dif=new DifusorPi(0.500000,0.200000,0.100000,0.100000,0.200000);
    dif->numeroPuntos(5);
    var->difusorEntrada(dif);
    var->nombreVariable("Variable 1");
    var->numeroIntervalos(30);
    entradas->adicionarVariable(var);
    var=new Variable(5);
    var->rangoMinimo(0.000000);
    var->rangoMaximo(1.000000);
    cd=new ConjuntoL(0.000000,0.166667,0.333333);
    var->adicionarConjuntos(cd);
    cd=new ConjuntoTriangulo(0.166667,0.333333,0.500000);
    var->adicionarConjuntos(cd);
    cd=new ConjuntoTriangulo(0.333333,0.500000,0.666667);
    var->adicionarConjuntos(cd);
    cd=new ConjuntoTriangulo(0.500000,0.666667,0.833333);
    var->adicionarConjuntos(cd);
```

```
cd=new ConjuntoGamma(0.666667,0.833333,1.000000);
var->adicionarConjuntos(cd);
dif=new DifusorPi(0.500000,0.200000,0.100000,0.100000,0.200000);
dif->numeroPuntos(5);
var->difusorEntrada(dif);
var->nombreVariable("Variable 2");
var->numeroIntervalos(30);
entradas->adicionarVariable(var);
var=new Variable(2);
var->rangoMinimo(0.000000);
var->rangoMaximo(1.000000);
cd=new ConjuntoL(0.000000,0.333333,0.666667);
var->adicionarConjuntos(cd);
cd=new ConjuntoGamma(0.333333,0.666667,1.000000);
var->adicionarConjuntos(cd);
var->nombreVariable("Uscita");
var->numeroIntervalos(292);
salidas->adicionarVariable(var);
motor=new MaquinaInferencia(entradas,salidas,25);
And=new Producto();
Composicion=new Producto();
Implica=new ImplicacionMinimo();
motor->and(And);
motor->composicion(Composicion);
motor->implicacion(Implica);

motor->conjuntoEntrada(0,0,0);
motor->conjuntoEntrada(0,1,0);    motor->conjuntoSalida(0,0,0);
motor->conjuntoEntrada(1,0,0);
motor->conjuntoEntrada(1,1,1);    motor->conjuntoSalida(1,0,0);
motor->conjuntoEntrada(2,0,0);
motor->conjuntoEntrada(2,1,2);    motor->conjuntoSalida(2,0,0);
motor->conjuntoEntrada(3,0,0);
motor->conjuntoEntrada(3,1,3);    motor->conjuntoSalida(3,0,0);
motor->conjuntoEntrada(4,0,0);
motor->conjuntoEntrada(4,1,4);    motor->conjuntoSalida(4,0,0);
motor->conjuntoEntrada(5,0,1);
motor->conjuntoEntrada(5,1,0);    motor->conjuntoSalida(5,0,1);
motor->conjuntoEntrada(6,0,1);
motor->conjuntoEntrada(6,1,1);    motor->conjuntoSalida(6,0,0);
motor->conjuntoEntrada(7,0,1);
motor->conjuntoEntrada(7,1,2);    motor->conjuntoSalida(7,0,0);
motor->conjuntoEntrada(8,0,1);
motor->conjuntoEntrada(8,1,3);    motor->conjuntoSalida(8,0,0);
motor->conjuntoEntrada(9,0,1);
motor->conjuntoEntrada(9,1,4);    motor->conjuntoSalida(9,0,0);
motor->conjuntoEntrada(10,0,2);
motor->conjuntoEntrada(10,1,0);    motor->conjuntoSalida(10,0,1);
motor->conjuntoEntrada(11,0,2);
motor->conjuntoEntrada(11,1,1);    motor->conjuntoSalida(11,0,1);
motor->conjuntoEntrada(12,0,2);
motor->conjuntoEntrada(12,1,2);    motor->conjuntoSalida(12,0,0);
motor->conjuntoEntrada(13,0,2);
motor->conjuntoEntrada(13,1,3);    motor->conjuntoSalida(13,0,0);
motor->conjuntoEntrada(14,0,2);
motor->conjuntoEntrada(14,1,4);    motor->conjuntoSalida(14,0,0);
motor->conjuntoEntrada(15,0,3);
motor->conjuntoEntrada(15,1,0);    motor->conjuntoSalida(15,0,1);
motor->conjuntoEntrada(16,0,3);
motor->conjuntoEntrada(16,1,1);    motor->conjuntoSalida(16,0,1);
motor->conjuntoEntrada(17,0,3);
motor->conjuntoEntrada(17,1,2);    motor->conjuntoSalida(17,0,1);
motor->conjuntoEntrada(18,0,3);
motor->conjuntoEntrada(18,1,3);    motor->conjuntoSalida(18,0,0);
motor->conjuntoEntrada(19,0,3);
motor->conjuntoEntrada(19,1,4);    motor->conjuntoSalida(19,0,0);
motor->conjuntoEntrada(20,0,4);
motor->conjuntoEntrada(20,1,0);    motor->conjuntoSalida(20,0,1);
motor->conjuntoEntrada(21,0,4);
motor->conjuntoEntrada(21,1,1);    motor->conjuntoSalida(21,0,1);
motor->conjuntoEntrada(22,0,4);
motor->conjuntoEntrada(22,1,2);    motor->conjuntoSalida(22,0,1);
motor->conjuntoEntrada(23,0,4);
motor->conjuntoEntrada(23,1,3);    motor->conjuntoSalida(23,0,1);
```

```
    motor->conjuntoEntrada(24,0,4);
    motor->conjuntoEntrada(24,1,4);    motor->conjuntoSalida(24,0,0);

    motor->modificador(0,0,1.000000);    motor->modificador(0,1,1.000000);
    motor->modificador(1,0,1.000000);    motor->modificador(1,1,1.000000);
    motor->modificador(2,0,1.000000);    motor->modificador(2,1,1.000000);
    motor->modificador(3,0,1.000000);    motor->modificador(3,1,1.000000);
    motor->modificador(4,0,1.000000);    motor->modificador(4,1,1.000000);
    motor->modificador(5,0,1.000000);    motor->modificador(5,1,1.000000);
    motor->modificador(6,0,1.000000);    motor->modificador(6,1,1.000000);
    motor->modificador(7,0,1.000000);    motor->modificador(7,1,1.000000);
    motor->modificador(8,0,1.000000);    motor->modificador(8,1,1.000000);
    motor->modificador(9,0,1.000000);    motor->modificador(9,1,1.000000);
    motor->modificador(10,0,1.000000);    motor->modificador(10,1,1.000000);
    motor->modificador(11,0,1.000000);    motor->modificador(11,1,1.000000);
    motor->modificador(12,0,1.000000);    motor->modificador(12,1,1.000000);
    motor->modificador(13,0,1.000000);    motor->modificador(13,1,1.000000);
    motor->modificador(14,0,1.000000);    motor->modificador(14,1,1.000000);
    motor->modificador(15,0,1.000000);    motor->modificador(15,1,1.000000);
    motor->modificador(16,0,1.000000);    motor->modificador(16,1,1.000000);
    motor->modificador(17,0,1.000000);    motor->modificador(17,1,1.000000);
    motor->modificador(18,0,1.000000);    motor->modificador(18,1,1.000000);
    motor->modificador(19,0,1.000000);    motor->modificador(19,1,1.000000);
    motor->modificador(20,0,1.000000);    motor->modificador(20,1,1.000000);
    motor->modificador(21,0,1.000000);    motor->modificador(21,1,1.000000);
    motor->modificador(22,0,1.000000);    motor->modificador(22,1,1.000000);
    motor->modificador(23,0,1.000000);    motor->modificador(23,1,1.000000);
    motor->modificador(24,0,1.000000);    motor->modificador(24,1,1.000000);

    concreto=new BloqueConcrecion(motor);
    Conjuncion=new Maximo();
    conc=new Altura(motor,0,Conjuncion);
    concreto->adicionarConcresor(conc);
    concreto->motor(motor);
    concreto->conjuncion(Conjuncion);
}

fz8::~fz8()
{
    delete concreto;
    delete motor;
    delete entradas;
    delete salidas;
}


// DICHIARAZIONI DELLE VARIABILI GLOBALI

   fz1 *fuzzy1;                  // dichiarazioni degli otto blocchi fuzzy
   fz2 *fuzzy2;                  // sottoforma di oggetti C++
   fz3 *fuzzy3;
   fz4 *fuzzy4;
   fz5 *fuzzy5;
   fz6 *fuzzy6;
   fz7 *fuzzy7;
   fz8 *fuzzy8;

   int progress[8][100];     // contiene la tabella delle giacenze

   float variab[17][100];    // contiene tutte le variabili di calcolo:
                             // ingressi, variabili intermedie ed uscite

   int medie[2][22];         // costituisce la banca dati: contiene per ogni articolo il
                             // parametro caratteristico che ne stima i movimenti

   int articolo;             // è l'articolo da elaborare

   int lead_time, primo_or, int_or,conf, righe;    // variabili aggiuntive di tipo
                                    // algebrico e numero di righe da elaborare

   float lsd, ogr, vt, vd, da, go;   // variabili fuzzy del sistema

   int min_dr, max_dr, amp_dr;       // intervallo di definizione di DR
   int min_of, max_of, amp_of;       // intervallo di definizione di OF
```

```
   int min_lr, max_lr, amp_lr;       // intervallo di definizione di LR


// FUNZIONI GLOBALI

// definizioni degli otto oggetti che rappresentano i blocchi fuzzy
void apri_blocchi()
{
   fuzzy1=new fz1;           // per ogni blocco si alloca la memoria necessaria:
   fuzzy2=new fz2;           // ogni istruzione invoca il costruttore della
   fuzzy3=new fz3;           // classe associata (da fz1 a fz8)
   fuzzy4=new fz4;
   fuzzy5=new fz5;
   fuzzy6=new fz6;
   fuzzy7=new fz7;
   fuzzy8=new fz8;
}

// disallocazione dello spazio in memoria occupato dai blocchi fuzzy
void chiudi_blocchi()
{
   delete fuzzy1;            // alla fine dell'esecuzione si libera la memoria,
   delete fuzzy2;            // invocando il distruttore di ogni classe
   delete fuzzy3;
   delete fuzzy4;
   delete fuzzy5;
   delete fuzzy6;
   delete fuzzy7;
   delete fuzzy8;
}

// funzione di calcolo del primo blocco fuzzy; i parametri sono i valori dei
// due ingressi, la funzione restituisce il valore calcolato per l'uscita
float out1(float in1, float in2)
{
   float *entrate;
   float *uscita, risultato;
   entrate=new float[2];
   entrate[0]=in1;
   entrate[1]=in2;
   uscita=new float[1];
   fuzzy1->calcular(entrate,uscita); // richiama la procedura di calcolo del blocco
                                     // fuzzy 1, che richiede in ingresso l'array
                                     // "entrate" e restituisce l'array "uscite"
   risultato=uscita[0];
   delete[] entrate;
   delete[] uscita;
   return(risultato);
}

// funzione di calcolo del secondo blocco fuzzy; il procedimento valido per
// il primo blocco si ripete per gli altri sette in modo analogo
float out2(float in1, float in2)
{
   float *entrate;
   float *uscita, risultato;
   entrate=new float[2];
   entrate[0]=in1;
   entrate[1]=in2;
   uscita=new float[1];
   fuzzy2->calcular(entrate,uscita);
   risultato=uscita[0];
   delete[] entrate;
   delete[] uscita;
   return(risultato);
}

float out3(float in1, float in2)
{
   float *entrate;
   float *uscita, risultato;
   entrate=new float[2];
   entrate[0]=in1;
   entrate[1]=in2;
```

```
   uscita=new float[1];
   fuzzy3->calcular(entrate,uscita);
   risultato=uscita[0];
   delete[] entrate;
   delete[] uscita;
   return(risultato);
}

float out4(float in1, float in2)
{
   float *entrate;
   float *uscita, risultato;
   entrate=new float[2];
   entrate[0]=in1;
   entrate[1]=in2;
   uscita=new float[1];
   fuzzy4->calcular(entrate,uscita);
   risultato=uscita[0];
   delete[] entrate;
   delete[] uscita;
   return(risultato);
}

float out5(float in1, float in2)
{
   float *entrate;
   float *uscita, risultato;
   entrate=new float[2];
   entrate[0]=in1;
   entrate[1]=in2;
   uscita=new float[1];
   fuzzy5->calcular(entrate,uscita);
   risultato=uscita[0];
   delete[] entrate;
   delete[] uscita;
   return(risultato);
}

float out6(float in1, float in2)
{
   float *entrate;
   float *uscita, risultato;
   entrate=new float[2];
   entrate[0]=in1;
   entrate[1]=in2;
   uscita=new float[1];
   fuzzy6->calcular(entrate,uscita);
   risultato=uscita[0];
   delete[] entrate;
   delete[] uscita;
   return(risultato);
}

float out7(float in1, float in2)
{
   float *entrate;
   float *uscita, risultato;
   entrate=new float[2];
   entrate[0]=in1;
   entrate[1]=in2;
   uscita=new float[1];
   fuzzy7->calcular(entrate,uscita);
   risultato=uscita[0];
   delete[] entrate;
   delete[] uscita;
   return(risultato);
}

float out8(float in1, float in2)
{
   float *entrate;
   float *uscita, risultato;
   entrate=new float[2];
   entrate[0]=in1;
```

```
   entrate[1]=in2;
   uscita=new float[1];
   fuzzy8->calcular(entrate,uscita);
   risultato=uscita[0];
   delete[] entrate;
   delete[] uscita;
   return(risultato);
}

// impostazione della banca dati relativa agli articoli e calcolo degli estremi
// degli intervalli di pre-elaborazione e post-elaborazione;
// medie[0][i] contiene i codici degli articoli
// medie[1][i] contiene il valore che stima i movimenti di ogni articolo
void imposta_estremi_dr_of_lr()
{
   medie[0][0]=4708;       medie[1][0]=8;
   medie[0][1]=5700;       medie[1][1]=9;
   medie[0][2]=11100;      medie[1][2]=6;
   medie[0][3]=11300;      medie[1][3]=5;
   medie[0][4]=20101;      medie[1][4]=337;
   medie[0][5]=22002;      medie[1][5]=21;
   medie[0][6]=22301;      medie[1][6]=17;
   medie[0][7]=24106;      medie[1][7]=13;
   medie[0][8]=24702;      medie[1][8]=13;
   medie[0][9]=25102;      medie[1][9]=8;
   medie[0][10]=25103;     medie[1][10]=15;
   medie[0][11]=25201;     medie[1][11]=4;
   medie[0][12]=25214;     medie[1][12]=19;
   medie[0][13]=25300;     medie[1][13]=4;
   medie[0][14]=25500;     medie[1][14]=2;
   medie[0][15]=30300;     medie[1][15]=9;
   medie[0][16]=30501;     medie[1][16]=5;
   medie[0][17]=34902;     medie[1][17]=11;
   medie[0][18]=41301;     medie[1][18]=24;
   medie[0][19]=13002;     medie[1][19]=18;
   medie[0][20]=25802;     medie[1][20]=58;
   medie[0][21]=99999;     medie[1][21]=100;

   int j=0;
   do                        // identifica l'articolo da elaborare
      j++;
   while(medie[0][j]!=articolo);

   int limite=(sqrt(sqrt(lead_time)))*medie[1][j];
   int mezzo=limite/2;       // correzione del limite per tenere conto
                             // di un eventuale lead time maggiore di 1

// impostazione degli estremi di DR, OF, LR
   min_dr=0-mezzo/2;
   max_dr=mezzo;
   min_of=0;
   max_of=mezzo;
   min_lr=0;
   max_lr=limite;
   amp_dr=mezzo*3/2;
   amp_of=mezzo;
   amp_lr=limite;
}

// funzione che esegue un ciclo di calcolo del sistema fuzzy con le variabili
// relative al periodo i, richiamando le esecuzioni successive delle
// funzioni di calcolo di tutti i blocchi del sistema
void calcolo_fuzzy(int i)
{
   variab[8][i]=out1(variab[0][i],variab[1][i]);   // richiama la funzione di calcolo del
               // blocco fuzzy 1, con ingressi le variabili 0 (LS) e 1 (OR) e uscita la
               // variabile 8 (FG); la stessa cosa è fatta per gli altri sette blocchi
   variab[9][i]=out2(variab[2][i],variab[3][i]);
   variab[10][i]=out3(variab[9][i],variab[8][i]);
   variab[11][i]=out4(variab[4][i],variab[10][i]);
   variab[12][i]=out5(variab[11][i],variab[7][i]);
   variab[13][i]=out6(variab[6][i],variab[4][i]);
   variab[14][i]=out7(variab[13][i],variab[8][i]);
   variab[15][i]=out8(variab[14][i],variab[5][i]);
```

```
}

// pre-elaborazione delle variabili d'ingresso DR e OF
void pre_elaborazione(int i)
{
   variab[6][i]=float(progress[2][i]-min_dr)/amp_dr;
   variab[7][i]=float(progress[3][i]-min_of)/amp_of;
}

// post-elaborazione della variabile d'uscita LR
void post_elaborazione(int i)
{
   progress[6][i]=min_lr+variab[12][i]*amp_lr;
}

// funzione che imposta, prima dell'elaborazione, le variabili d'ingresso fuzzy
// di ogni periodo di calcolo, sfruttando le variabili algebriche ausiliarie
// primo ordine regolare, intervallo ordini regolari e lead time
void imposta_variabili()
{
   int regolare=primo_or;
   for (int i=lead_time+1; i<=righe; i++)
   {  if (i-lead_time>regolare) regolare+=int_or;
      variab[0][i]=lsd;
      variab[1][i]=ogr;
      variab[2][i]=vt;
      variab[3][i]=vd;
      variab[4][i]=da;
      if (i-lead_time==regolare) variab[5][i]=0;
      else
      {  variab[5][i]=go;           // imposta il genere dell'ordine
      }
   }
}

// funzione che salva i risultati dopo l'elaborazione nell'archivio nome_file
// il file salvato risulta leggibile dall'applicazione d'interfaccia creata con VB
void scrittura_file(char *nome_file)
{
   FILE *stream;
   char buf[99];
   int c=6;

   stream = fopen(nome_file, "w");
   fseek(stream, SEEK_SET, 0);

   itoa(righe,buf,10);
   fputs(buf,stream);
   fputs("\n",stream);

   itoa(lead_time,buf,10);
   fputs(buf,stream);
   fputs("\n",stream);

   itoa(primo_or,buf,10);
   fputs(buf,stream);
   fputs("\n",stream);

   itoa(int_or,buf,10);
   fputs(buf,stream);
   fputs("\n",stream);

   itoa(conf,buf,10);
   fputs(buf,stream);
   fputs("\n",stream);

   gcvt(lsd,4,buf);
   fputs(buf,stream);
   fputs("\n",stream);

   gcvt(ogr,4,buf);
   fputs(buf,stream);
   fputs("\n",stream);
```

```
   gcvt(vt,4,buf);
   fputs(buf,stream);
   fputs("\n",stream);

   gcvt(vd,4,buf);
   fputs(buf,stream);
   fputs("\n",stream);

   gcvt(da,4,buf);
   fputs(buf,stream);
   fputs("\n",stream);

   gcvt(go,4,buf);
   fputs(buf,stream);
   fputs("\n",stream);

   printf("\n");
   for (int i=0; i<20; i++) buf[i]=' ';
   for (int i=1; i<=righe; i++)
   {  for (int j=0; j<c-1; j++)
      {  itoa(progress[j][i],buf,10);
         fwrite(buf,14,1,stream);
         for (int k=0; k<10; k++) buf[k]=' ';
      }
      itoa(progress[c-1][i],buf,10);
      fwrite(buf,4,1,stream);
      fputs("\n",stream);
   }
   fclose(stream);
}

// funzione che legge l'archivio contenente i dati iniziali da elaborare
// il file proviene dall'applicazione d'interfaccia creata con VB
void lettura_file(char *nome_file)
{
   FILE *stream;
   char buf[99];
   int c=6;

   stream = fopen(nome_file, "r");
   fseek(stream, SEEK_SET, 0);

   fgets(buf,14,stream);
   righe=atoi(buf);


   fgets(buf,14,stream);
   lead_time=atoi(buf);

   fgets(buf,14,stream);
   primo_or=atoi(buf);

   fgets(buf,14,stream);
   int_or=atoi(buf);

   fgets(buf,14,stream);
   conf=atoi(buf);

   fgets(buf,14,stream);
   lsd=atof(buf);

   fgets(buf,14,stream);
   ogr=atof(buf);

   fgets(buf,14,stream);
   vt=atof(buf);

   fgets(buf,14,stream);
   vd=atof(buf);

   fgets(buf,14,stream);
   da=atof(buf);

   fgets(buf,14,stream);
```

```
   go=atof(buf);

   for (int i=1; i<=righe; i++)
   {  for (int j=0; j<c-1; j++)
      {  fread(buf,14,1,stream);
       progress[j][i]=atoi(buf);
      }
      fgets(buf,14,stream);
      progress[c-1][i]=atoi(buf);
   }
   fclose(stream);
}


// PROGRAMMA PRINCIPALE

void main()
{

// definizioni variabili
   int q='c';
   int ordine;
   int colonne_progr=9;
   char str_art[10];
   char copia[10];
   char nomefile[20];
   double media;

// allocazione della memoria necessaria agli otto blocchi fuzzy
   apri_blocchi();

// inizio del ciclo di esecuzione
do
{
   clrscr();
   if (q=='c')
   {   printf(" Codice articolo : ");       // lettura numero articolo da tastiera:
       cin>>str_art;                        // il codice numerico è elaborato per
       articolo=atoi(str_art);              // formare la stringa con il nome del
       strcat(str_art,".txt");              // file da caricare in memoria
   }
   strcpy(nomefile,"d:\\eris\\tesi\\in");
   strcpy(copia,str_art);
   strcat(nomefile,copia);
   lettura_file(nomefile);          // caricamento del file relativo all'articolo scelto

   imposta_estremi_dr_of_lr();      // impostazione degli estremi dei DR, OF e LR

   for(int k=1; k<=righe; k++)      // azzeramento delle celle della matrice
   {  progress[7][k]=0;             // relative alle variabili d'uscita
      progress[8][k]=0;
   }

   imposta_variabili();             // impostazione delle variabili di calcolo

// esecuzione di un ciclo di calcolo per ogni periodo d'interesse; righe è il numero di
// periodi considerati; il primo periodo modificabile dipende dal lead time
   for (int i=lead_time+1; i<=righe; i++)
   {
      printf("%3i",i);

// esecuzione completa di una elaborazione fuzzy: pre_elab., calcolo e post-elab.
// si ripete per ogni periodo i compreso nell'intervallo voluto
      pre_elaborazione(i);
      calcolo_fuzzy(i);
      post_elaborazione(i);

// blocco lineare posteriore al blocco fuzzy 5 di calcolo dei fabbisogni: calcola
// l'uscita Quantità Necessaria sottraendo DR dal valore di LR appena calcolato
      progress[7][i]=progress[6][i]-progress[2][i];

// calcola EO trasformando in 0 o 1 il valore d'uscita ottenuto dal blocco fuzzy 8
      if ((variab[15][i]<.5)||(progress[7][i]<=0)) progress[8][i]=0;
      else progress[8][i]=1;
```

```
// correzione della quantità da ordinare in base alla confezione minima
      if ((conf!=1)&&((progress[7][i]%conf)!=0))
       ordine=(int(progress[7][i]/conf)+1)*conf;
      else ordine=progress[7][i];

// aggiornamento delle giacenze nel caso si decida di emettere l'ordine
      if (progress[8][i]==1)
      {   for (int ii=i-lead_time; ii<=righe; ii++)
                  {  progress[4][ii]+=ordine;
                              if (ii<i) progress[3][ii]+=ordine;
                     else
               {      progress[0][ii]+=ordine;
                                 progress[2][ii]+=ordine;
               }
                  }
      }

// aggiornamento delle giacenze con le quantità vendute durante la settimana; è successivo
// al calcolo dei fabbisogni, che non deve tenere conto prima del dovuto di tali vendite
      int i_lt=i-lead_time;
      if (progress[2][i_lt]>=progress[5][i_lt])
      {for (int ii=i_lt+1; ii<=righe; ii++)
         {    progress[2][ii]-=progress[5][i_lt];
              progress[0][ii]-=progress[5][i_lt];
              progress[4][ii]-=progress[5][i_lt];
         }
      }
      else
      {  for (int ii=i_lt+1; ii<=righe; ii++)
         {    progress[2][ii]-=progress[2][i_lt];
                progress[0][ii]-=progress[2][i_lt];
             progress[4][ii]-=progress[2][i_lt];
         }
      }
// fine dell'i-esimo ciclo di calcolo
   }
// fine di tutti i cicli di calcolo

// visulazzazione dei risultati sullo schermo e calcolo delle giacenze medie
   cout<<"\nper    EA    IC    DR    OF    DP    QV    LR    QN    EO\n";
   media=0;
   for (int k=1; k<=righe; k++)
   {  printf("%3i",k,"   ");
              for (int j=0; j<colonne_progr; j++) printf("%6i",progress[j][k]);
       if (progress[5][k]-progress[2][k]>0) printf("%6i",progress[5][k]-progress[2][k]);
      media+=progress[2][k];
      if (k%21==0) getch();
       printf("\n");
   }
   media/=righe;              // la media è calcolata sulle giacenze d'inizio settimana
   printf("           media = ");
   printf("%4.1f",media);
   printf("\n");

// salvataggio del file con i risultati; la stringa contenente il nome del file è
// composta partendo dal codice dell'articolo
   strcpy(nomefile,"d:\\eris\\tesi\\out");
   strcpy(copia,str_art);
   strcat(nomefile,copia);
   scrittura_file(nomefile);

// si lascia all'utente la scelta di ripetere il calcolo, cambiare articolo o terminare
// l'esecuzione; se contemporaneamente si sta eseguendo anche il programma VB, si possono
// cambiare i valori delle variabili, e ripetere l'esecuzione con parametri diversi
   printf(" <r>ipeti il calcolo, <c>ambia articolo, <e>sci) : ");
   do q=getche(); while ((q!='r')&&(q!='c')&&(q!='e'));
}
while (q!='e');

// disallocazione delle variabili dinamiche con i distruttori delle classi
chiudi_blocchi();

}
```

APPENDICE B

IL PROGRAMMA D'INTERFACCIA – CODICE *VISUAL BASIC*

Il programma è suddiviso in un modulo (Modulo1), contenente le dichiarazioni delle variabili globali, e quattro form. Uno di essi è usato come contenitore (Main). Degli altri, due sono "figli", ossia sono contenuti nel form principale, l'ultimo (Form3) è a scelta obbligata, e serve a caricare i file.

Modulo1 – Dichiarazioni globali

```
' Le variabili dichiarate in questo modulo sono condivise da tutti i Form del programma

Const m = 100                       ' numero massimo di righe della tabella delle giacenze
Public ora As Integer               ' periodo attuale
Public n As Integer                 ' numero di record (periodi) effettivamente utilizzati

Public lead_time As Integer         ' variabili algebriche
Public primo_or As Integer
Public int_or As Integer
Public confezione As Integer

Public per(1 To m) As Integer       ' in questi sette array sono inseriti i valori
Public ea(1 To m) As Integer        '  visualizzati nella tabella, in attesa di ricopiarli
Public ic(1 To m) As Integer        '  negli appositi archivi
Public dr(1 To m) As Integer
Public of(1 To m) As Integer
Public dp(1 To m) As Integer
Public qv(1 To m) As Integer

Public dbprog As Database           ' variabili che servono a caricare un database
Public rec As Recordset

Public modificato As Boolean        ' indica se il file caricato è stato modificato
                                    '  serve a segnalare il mancato salvataggio
Public file_db As Boolean    ' distingue il tipo di file da caricare: true=mdb false=txt

Public lsd As Double                ' variabili fuzzy
Public ogr As Double
Public vt As Double
Public vd As Double
Public da As Double
Public go As Double

Public nomefiletxt As String        ' nome del file caricato

Public buf As String                ' variabili ausiliarie
Public ricorda As String

Function nuovo_valore(s As String) As String       ' funzione ausiliaria per la lettura
   pos = InStr(s, " ")                             ' dei dati dagli archivi
   s = Right(s, Len(s) - pos)
   s = LTrim(s)
   nuovo_valore = s
End Function
```

Main – Form principale

```
' È un form di tipo MDI, che costituisce lo sfondo dell'applicazione e il contenitore dei
' form "figli" Form1 e Form2; in esso è costruito il menu, che risulta così valido negli
' altri form; quasi tutte le subroutine presentate di seguito corrispondono a voci del menu

Private Sub apri_file_text_Click()   ' apre la finestra di caricamento di un file .txt
   file_db = False
   apri_file.Show 1
End Sub

Private Sub consegna_Click()         ' richiama la procedura del Form1 che riproduce
   Form1.consegna_immediata          '  l'effetto di una consegna immediata
```

```
End Sub

Private Sub default_Click()          ' richiama la procedura del Form2 che imposta le
   Form2.imposta_default             '  variabili ai loro valori di default
End Sub

Private Sub MDIForm_Load()           ' non corrisponde ad una voce di menu: è la prima
   Load Form1                        '  subroutine eseguita all'avvio del programma, che
   Load Form2                        '  carica i Form e visualizza il Form1
   Form1.SetFocus
End Sub

Private Sub menu_f1_Click()          ' visualizza il Form1
   Form1.SetFocus
End Sub

Private Sub menu_f2_Click()          ' visualizza il Form2
   Form2.SetFocus
End Sub

Private Sub numeri_Click()           ' richiama la procedura del Form2 che mostra e
   Form2.val_num                     '  nasconde i valori numerici delle variabili fuzzy
End Sub

Private Sub nuovo_Click()            ' richiama la procedura del Form1 che permette di
   Form1.nuovo_file                  '  creare un nuovo archivio (si usa per gestire un
End Sub                              '  nuovo articolo)

Private Sub apri_Click()             ' apre la finestra di caricamento di un file .mdb
   file_db = True
   apri_file.Show 1
End Sub

Private Sub salva_Click()            ' richiama la procedura del Form1 che salva un
   Form1.scrivi_su_file              '  archivio database
End Sub

Private Sub salva_file_text_Click()         ' richiama la procedura del Form1 che salva
   Form1.scrivi_file_text (nomefiletxt)     '  un archivio sequenziale
End Sub

' uscita dal programma: se un archivio è stato modificato, ma non salvato, è
'  visualizzato un Message Box, che richiede se si vuole salvare il file prima di uscire

Private Sub esci_Click()
If modificato Then
      uscita = MsgBox("Il file modificato non è stato salvato" + Chr(13)
                      + Chr(10) + "Salvare il file prima di uscire?",
                      35, "Uscita da Gestione scorte con logica fuzzy")
      If uscita = vbYes Then
         Form1.scrivi_file_text (nomefiletxt)
         End
      Else:
         If uscita = vbNo Then End
      End If
   Else: End
   End If
End Sub

Private Sub svuota_Click()           ' richiama la procedura del Form1 che azzera la
   Form1.azzera                      '  tabella delle giacenze
End Sub

Private Sub menu_ic_Click()                 ' richiama la procedura del Form1 che
   Form1.scarico Val(quantita.Text)         '  riproduce l'effetto sulle giacenze
   adesso.Text = Val(adesso.Text) + 1       '  di un impegno con i clienti
End Sub

Private Sub menu_of_Click()                 ' come sopra, per un ordine ai fornitori
   Form1.carico Val(quantita.Text), Val(tempo.Text)
   adesso.Text = Val(adesso.Text) + 1
End Sub
```

Form1 – Osservazione e modifica dello stato delle giacenze

```
' questo form contiene la tabella con lo stato delle giacenze, nonché i comandi e le
'  caselle di testo necessari a modificarla nel modo desiderato; in esso sono contenute
'  anche le subroutine di gestione dei file

' subroutine definite appositamente (devono essere richiamate da altre subroutine)

Sub leggi_file_text(nomefile)        ' procedura che carica in memoria un archivio
   Form1.Caption = nomefile          '  sequenziale, ed imposta di conseguenza le
   Form2.Caption = nomefile          '  variabili e la tabella delle giacenze
   azzera
   Open nomefile For Input As #1     ' apertura del file per l'input

   Line Input #1, buf                ' numero di righe significative per le giacenze
   n = Val(buf)

   Line Input #1, buf                ' lettura variabili algebriche
   lead_time = Val(buf)
   Form2.lt_input.Text = lead_time

   Line Input #1, buf
   primo_or = Val(buf)
   Form2.primo_ord_reg.Text = primo_or

   Line Input #1, buf
   int_or = Val(buf)
   Form2.int_ord_reg.Text = int_or

   Line Input #1, buf
   confezione = Val(buf)
   Form2.conf.Text = confezione

   Line Input #1, buf                ' lettura variabili fuzzy
   lsd = Val(buf)
   Form2.barra_lsd.Value = lsd * 1000

   Line Input #1, buf
   ogr = Val(buf)
   Form2.barra_ogr.Value = ogr * 1000

   Line Input #1, buf
   vt = Val(buf)
   Form2.barra_vt.Value = vt * 1000

   Line Input #1, buf
   vd = Val(buf)
   Form2.barra_vd.Value = vd * 1000

   Line Input #1, buf
   da = Val(buf)
   Form2.barra_da.Value = da * 1000

   Line Input #1, buf
   go = Val(buf)
   Form2.barra_go.Value = go * 1000

   For i = 1 To n                    ' imposta gli array delle giacenze
      Line Input #1, buf
      buf = LTrim(buf)
      ea(i) = Val(Left(buf, 8))
      ic(i) = Val(Left(nuovo_valore(buf), 8))
      dr(i) = Val(Left(nuovo_valore(buf), 8))
      of(i) = Val(Left(nuovo_valore(buf), 8))
      dp(i) = Val(Left(nuovo_valore(buf), 8))
      qv(i) = Val(Left(nuovo_valore(buf), 8))
   Next
   Close #1
   modificato = False
   scrivi                            ' visualizza la tabella aggiornata
End Sub

Sub scrivi_file_text(nomefile)       ' procedura che salva un archivio modificato
```

```
   Open nomefile For Output As #1    ' apertura del file per l'output
   Print #1, Str(n)                  ' scrittura delle diverse variabili
   Print #1, Str(lead_time)
   Print #1, Str(primo_or)
   Print #1, Str(int_or)
   Print #1, Str(confezione)
   Print #1, Str(lsd)
   Print #1, Str(ogr)
   Print #1, Str(vt)
   Print #1, Str(vd)
   Print #1, Str(da)
   Print #1, Str(go)
   For i = 1 To n
      Print #1, Str(ea(i)), Str(ic(i)), Str(dr(i)), Str(of(i)), Str(dp(i)), Str(qv(i))
   Next
   Close #1
   modificato = False
End Sub

Sub scrivi()             ' procedura che visualizza la tabella delle giacenze aggiornata
   disp.Redraw = False
   For i = 1 To n
      disp.Row = i
      disp.Col = 0
      disp.Text = i
      disp.Col = 1
      disp.Text = ea(i)
      disp.Col = 2
      disp.Text = ic(i)
      disp.Col = 3
      disp.Text = dr(i)
      disp.Col = 4
      disp.Text = of(i)
      disp.Col = 5
      disp.Text = dp(i)
      disp.Col = 6
      disp.Text = qv(i)
   Next
   For i = n + 1 To disp.Rows - 1
      disp.Row = i
      disp.Col = 0
      disp.Text = ""
      disp.Col = 1
      disp.Text = ""
      disp.Col = 2
      disp.Text = ""
      disp.Col = 3
      disp.Text = ""
      disp.Col = 4
      disp.Text = ""
      disp.Col = 5
      disp.Text = ""
      disp.Col = 6
      disp.Text = ""
   Next
   disp.Redraw = True
End Sub

Sub caricamento(nomefile)           ' procedura per il caricamento di un filedatabase
   Set dbprog = OpenDatabase(nomefile)
   Set rec = dbprog.OpenRecordset("progressivo")
   If Not rec.EOF Then rec.MoveFirst
   i = 1
   Do Until rec.EOF
      per(i) = rec.Fields(0)
      ea(i) = rec.Fields(1)
      ic(i) = rec.Fields(2)
      dr(i) = rec.Fields(3)
      of(i) = rec.Fields(4)
      dp(i) = rec.Fields(5)
      qv(i) = rec.Fields(6)
      rec.MoveNext
      i = i + 1
```

```
   Loop
   n = i - 1
   scrivi
End Sub

Sub scrivi_su_file()                ' procedura per il salvataggio di un file database
   If rec.RecordCount <> 0 Then rec.MoveFirst
   For i = 1 To n
      If rec.EOF Then
         rec.AddNew
         rec.Fields(0) = i
         rec.Update
         rec.Bookmark = rec.LastModified
      End If
      rec.Edit
      rec.Fields(1) = ea(i)
      rec.Fields(2) = ic(i)
      rec.Fields(3) = dr(i)
      rec.Fields(4) = of(i)
      rec.Fields(5) = dp(i)
      rec.Fields(6) = qv(i)
      rec.Update
      rec.MoveNext
   Next
   rec.Close
   dbprog.Close
End Sub

Sub azzera()                        ' procedura che svuota la tabella delle giacenze
   n = 1
   per(1) = 1
   ea(1) = 0
   ic(1) = 0
   dr(1) = 0
   of(1) = 0
   dp(1) = 0
   qv(1) = 0
   modificato = True
   scrivi
End Sub

Sub carico(q, t As Integer)         ' procedura che modifica le giacenze riproducendo gli
   Do Until n >= ora + t            '  effetti di un ordine ai fornitori
      n = n + 1
      per(n) = n
      ea(n) = ea(n - 1)
      ic(n) = 0
      dr(n) = dr(n - 1)
      of(n) = 0
      dp(n) = dp(n - 1)
      qv(n) = 0
   Loop
   For i = ora To ora + t - 1
      of(i) = of(i) + q
      dp(i) = dr(i) + of(i)
   Next
   For i = ora + t To n
      ea(i) = ea(i) + q
      dr(i) = dr(i) + q
      dp(i) = dr(i) + of(i)
   Next
   modificato = True
   scrivi
End Sub

Sub scarico(q As Integer)              ' procedura che modifica le giacenze riproducendo
   adesso.Text = ora + Val(tempo.Text)  '  gli effetti di un ordine dei clienti
   Do Until n >= ora
      n = n + 1
      per(n) = n
      ea(n) = ea(n - 1)
      ic(n) = 0
      dr(n) = dr(n - 1)
      of(n) = 0
```

```
      dp(n) = dp(n - 1)
      qv(n) = 0
   Loop
   ic(ora) = ic(ora) + q
   For i = ora To n
      ea(i) = ea(i) - q
      dr(i) = dr(i) - q
      dp(i) = dr(i) + of(i)
   Next
   modificato = True
   scrivi
   adesso.Text = ora - Val(tempo.Text)
End Sub

Sub consegna_immediata()            ' procedura che modifica le giacenze riproducendo
   Do While (n < ora)               '  gli effetti di una consegna immediata
      n = n + 1
      per(n) = n
      ea(n) = ea(n - 1)
      ic(n) = 0
      dr(n) = dr(n - 1)
      of(n) = 0
      dp(n) = dp(n - 1)
      qv(n) = 0
   Loop
   qv(ora) = quantita.Text
   scrivi
   adesso.Text = ora + 1
   modificato = True
End Sub

Sub nuovo_file()                    ' procedura per la creazione di un nuovo archivio,
                                    '  con richiesta del codice dell'articolo da inserire
   nuovo = InputBox("Digita il codice del nuovo articolo da inserire", "Nuovo articolo")
   If nuovo <> "" Then
      Form1.SetFocus
      nomefile = "d:\eris\tesi\in" + nuovo + ".txt"
      azzera
      Form1.Caption = nomefile
      Form2.Caption = nomefile
      scrivi_file_text (nomefile)
   End If
End Sub

' subroutine corrispondenti ad eventi VB (soprattutto modifiche dei controlli grafici)

Private Sub adesso_Change()         ' cambiamento della casella con il periodo attuale
   ora = Val(adesso.Text)           ' il valore del periodo attuale è aggiornato
End Sub

Private Sub but_ic_Click()          ' pulsante che richiama la procedura di IC
   scarico Val(quantita.Text)
   adesso.Text = Val(adesso.Text) + 1
End Sub

Private Sub but_of_Click()          ' pulsante che richiama la procedura di OF
   carico Val(quantita.Text), Val(tempo.Text)
   adesso.Text = Val(adesso.Text) + 1
End Sub

Private Sub Form_Load()             ' caricamento del Form1
   disp.ColWidth(-1) = 800
   disp.ColWidth(0) = 640
   nomefiletxt = "d:\eris\tesi\in99999.txt"     ' caricamento, per default, dell'archivio
   leggi_file_text (nomefiletxt)                '  dell'articolo d'esempio 99999
   ora = 1
   adesso.Text = ora
   disp.Row = 0
   disp.Col = 0
   disp.Text = "periodo"
   disp.Col = 1
   disp.Text = "EA"
   disp.Col = 2
   disp.Text = "IC"
```

```
   disp.Col = 3
   disp.Text = "DR"
   disp.Col = 4
   disp.Text = "OF"
   disp.Col = 5
   disp.Text = "DP"
   disp.Col = 6
   disp.Text = "QV"
End Sub

Private Sub imp_var_Click()          ' pulsante per passare dal Form1 al Form2; cambiano i
   Form2.SetFocus                    '  pulsanti e le opzioni del menu da disabilitare
   ricorda = quantita.Text
   quantita.Text = ""
   Main.default.Enabled = True
   Main.numeri.Enabled = True
   Main.svuota.Enabled = False
End Sub

Private Sub quantita_Change()        ' cambiamento della casella con la quantità
   If quantita.Text <> "" Then       '  a seconda del suo contenuto, sono abilitati e
      but_ic.Enabled = True          '  disabilitati pulsanti e voci di menu collegati
      Main.menu_ic.Enabled = True
      vendita.Enabled = True
      Main.consegna.Enabled = True
      If tempo.Text <> "" Then
         but_of.Enabled = True
         Main.menu_of.Enabled = True
      Else
         but_of.Enabled = False
         Main.menu_of.Enabled = False
      End If
   Else
      but_of.Enabled = False
      but_ic.Enabled = False
      vendita.Enabled = False
      Main.consegna.Enabled = False
      Main.menu_of.Enabled = False
      Main.menu_ic.Enabled = False
   End If
End Sub

Private Sub svuota_but_Click()       ' pulsante che richiama la procedura di azzeramento
   azzera
End Sub

Private Sub tempo_Change()           ' cambiamento della casella con il tempo di consegna
   If quantita.Text <> "" Then       '  a seconda del suo contenuto, sono abilitati e
      but_ic.Enabled = True          '  disabilitati pulsanti e voci di menu collegati
      Main.menu_ic.Enabled = True
      vendita.Enabled = True
      Main.consegna.Enabled = True
      If tempo.Text <> "" Then
         but_of.Enabled = True
         Main.menu_of.Enabled = True
      Else
         but_of.Enabled = False
         Main.menu_of.Enabled = False
      End If
   Else
      but_of.Enabled = False
      but_ic.Enabled = False
      vendita.Enabled = False
      Main.consegna.Enabled = False
      Main.menu_of.Enabled = False
      Main.menu_ic.Enabled = False
   End If
End Sub

Private Sub vendita_Click()          ' pulsante che richiama la procedura di vendita
   consegna_immediata                '  immediata
End Sub
```

Form 2 – Impostazione delle variabili di gestione

```
' il form contiene le interfaccie di visualizzazione e modifica delle sei variabili di
' gestione fuzzy e delle quattro algebriche

Dim def_lt As Integer                ' valori di default delle dieci variabili
Dim def_por As Integer
Dim def_ior As Integer
Dim def_con As Integer
Dim def_lsd As Double
Dim def_ogr As Double
Dim def_da As Double
Dim def_vd As Double
Dim def_vt As Double
Dim def_go As Double

' subroutine definite appositamente (devono essere richiamate da altre subroutine)

Sub imposta_default()                ' procedura che assegna alle variabili i loro
   lt_input.Text = def_lt            '  valori di default
   primo_ord_reg.Text = def_por
   int_ord_reg.Text = def_ior
   conf.Text = def_con
   barra_lsd.Value = def_lsd * 1000
   barra_ogr.Value = def_ogr * 1000
   barra_da.Value = def_da * 1000
   barra_vd.Value = def_vd * 1000
   barra_vt.Value = def_vt * 1000
   barra_go.Value = def_go * 1000
   modificato = True
End Sub

Sub val_num()                                    ' procedura che mostra e nasconde i
   Lab_lsd.Visible = Not (Lab_lsd.Visible)       '  valori numerici delle variabili
   Lab_ogr.Visible = Not (Lab_ogr.Visible)
   Lab_da.Visible = Not (Lab_da.Visible)
   Lab_vd.Visible = Not (Lab_vd.Visible)
   Lab_vt.Visible = Not (Lab_vt.Visible)
   Lab_go.Visible = Not (Lab_go.Visible)
End Sub

' subroutine corrispondenti ad eventi VB

Private Sub Form_initialize()            ' subroutine richiamata al caricamento del form
   def_lt = Val(lt_input.Text)           '  salva su variabili ausiliarie i valori di
   def_por = Val(primo_ord_reg.Text)     '  default, impostati in fase di progettazione
   def_ior = Val(int_ord_reg.Text)       '  nei rispettivi controlli
   def_con = Val(conf.Text)
   def_lsd = barra_lsd.Value / 1000
   Lab_lsd.Caption = def_lsd
   def_ogr = barra_ogr.Value / 1000
   Lab_ogr.Caption = def_ogr
   def_da = barra_da.Value / 1000
   Lab_da.Caption = def_da
   def_vd = barra_vd.Value / 1000
   Lab_vd.Caption = def_vd
   def_vt = barra_vt.Value / 1000
   Lab_vt.Caption = def_vt
   def_go = barra_go.Value / 1000
   Lab_go.Caption = def_go
End Sub

Private Sub mostra_val_Click()           ' pulsante che richiama la procedura di
   val_num                               '  visualizzazione dei valori numerici
End Sub

Private Sub aggiorna_Click()             ' pulsante che richiama la procedura del Form1
   Form1.scrivi_file_text (nomefiletxt)  '  per il salvataggio dei file sequenziali
End Sub

Private Sub def_Click()                  ' pulsante che richiama la procedura che
   imposta_default                       '  imposta le variabili ai valori di default
End Sub
```

```
Private Sub Elabora_Click()          ' pulsante che lancia l'esecuzione del
                                     '  programma di calcolo in C++
   e = Shell("d:\aprog\sistema_int\sf_int.exe", 1)
End Sub

Sub Command1_Click()                 ' pulsante per passare dal Form1 al Form2; cambiano i
   Form1.SetFocus                    '  pulsanti e le opzioni del menu da disabilitare
   Form1.quantita.Text = ricorda
   Main.default.Enabled = False
   Main.bruza.Recia = True
   Main.numeri.Enabled = False
   Main.svuota.Enabled = True
End Sub

Private Sub barra_lsd_Change()       ' modifica della barra di scorrimento relativa alla
   lsd = barra_lsd.Value / 1000      '  variabile livello di servizio desiderato
   Lab_lsd.Caption = lsd             '  e aggiornamento della variabile stessa;
   modificato = True                 '  lo stesso vale per le altre variabili fuzzy
End Sub

Private Sub barra_ogr_Change()       ' variabile onerosità giacenze
   ogr = barra_ogr.Value / 1000
   Lab_ogr.Caption = ogr
   modificato = True
End Sub

Private Sub barra_da_Change()        ' variabile domanda attesa
   da = barra_da.Value / 1000
   Lab_da.Caption = da
   modificato = True
End Sub

Private Sub barra_vd_Change()        ' variabile variabilità domanda attesa
   vd = barra_vd.Value / 1000
   Lab_vd.Caption = vd
   modificato = True
End Sub

Private Sub barra_vt_Change()        ' variabile variabilità tempi di consegna
   vt = barra_vt.Value / 1000
   Lab_vt.Caption = vt
   modificato = True
End Sub

Private Sub barra_go_Change()        ' variabile genere ordine
   go = barra_go.Value / 1000
   Lab_go.Caption = go
   modificato = True
End Sub

Private Sub conf_Change()            ' modifica della casella di testo relativa alla
   confezione = Val(conf.Text)       '  variabile confezione
   modificato = True                 '  e aggiornamento della variabile stessa;
End Sub                              '  lo stesso vale per le altre variabili algebriche

Private Sub lt_input_Change()        ' variabile lead time
   lead_time = Val(lt_input.Text)
   modificato = True
End Sub

Private Sub primo_ord_reg_Change()   ' variabile primo ordine normale
   primo_or = Val(primo_ord_reg.Text)
   modificato = True
End Sub

Private Sub int_ord_reg_Change()     ' variabile frequenza ordini normali
   int_or = Val(int_ord_reg.Text)
   modificato = True
End Sub
```

Form 3 – Caricamento degli archivi

```
' è un form a scelta obbligata; non si può passare ad un altro form prima di
' aver chiuso questo tramite uno dei pulsanti disponibili (OK o Annulla)

Private Sub annulla_Click()          ' pulsante Annulla;
apri_file.Hide                       '  chiude il form senza effettuare alcuna operazione
End Sub

Private Sub apri_Click()             ' pulsante OK; richiama una delle due procedure del
   apri_file.Hide                    '  Form1 per il caricamento dei file, distinguendo
   Form1.SetFocus                    '  tra file text e database
   If file_db Then
      Form1.caricamento (percorso.Text)
   Else:
      nomefiletxt = percorso.Text
      Form1.leggi_file_text (nomefiletxt)
   End If
End Sub

' i controlli Drive1, Dir1 e File1 sono collegati, in modo che la modifica dell'uno sia
'  tenuta in debito conto dagli altri e dalla casella di testo percorso, che contiene il
'  percorso completo con il nome del file da caricare

Private Sub Drive1_Change()          ' cambiamento del drive selezionato
   Dir1.Path = Drive1.Drive
   File1.Path = Dir1.Path
   percorso.Text = File1.Path & "\" & File1.filename
End Sub

Private Sub Dir1_Change()            ' cambiamento della directory di lavoro
   File1.Path = Dir1.Path
   percorso.Text = File1.Path & "\" & File1.filename
 End Sub

Private Sub File1_Click()            ' selezione del file da caricare
   percorso.Text = File1.Path & "\" & File1.filename
End Sub

Private Sub percorso_Click()         ' scrittura del percorso direttamente come testo
   File1.filename = percorso.Text
End Sub

Private Sub File1_DblClick()                          ' doppio click con il tasto sinistro
   percorso.Text = File1.Path & "\" & File1.filename  '  del mouse sul controllo File1:
   apri_file.Hide                                     '  avvia il caricamento del file
   Form1.SetFocus                                     '  come per il pulsante OK
   If file_db Then
      Form1.caricamento (percorso.Text)
   Else:
      nomefiletxt = percorso.Text
      Form1.leggi_file_text (nomefiletxt)
   End If
End Sub

Private Sub Form_Activate()          ' subroutine lanciata al caricamento del form
   If file_db Then                   '  imposta il percorso di default, sempre
      File1.Pattern = "*.mdb"        '  distinguendo tra file text e database
      apri_file.Caption = "Apri database (*.mdb)"
      Drive1.Drive = "d:"
      Dir1.Path = "d:\eris\tesi"
'      File1.Path = "d:\eris\tesi"
'      File1.filename = "progressivo.mdb"
   Else:
      File1.Pattern = "*.txt"
      apri_file.Caption = "Apri file sequenziale (*.txt)"
      Drive1.Drive = "d:"
      Dir1.Path = "d:\eris\tesi"
'      File1.Path = "d:\eris\tesi"
'      File1.filename = "in1.txt"
   End If
   percorso.Text = File1.Path & "\" & File1.filename
End Sub
```

BIBLIOGRAFIA

Fuzzy Logic

[1] Braghini S., Mazzucchi F., *Teoria dei fuzzy set e controllo di processo*, da Internet, 1998

[2] Cammarata S., *Sistemi a logica fuzzy – Come rendere intelligenti le macchine – seconda edizione rinnovata e aggiornata*, Etas Libri, 1997

[3] Crivellari M., *Implementazione di un controllo fuzzy per azionamento in corrente continua*, Tesi di laurea, Padova, anno acc. 1991-92

[4] Driankov D., Hellendoorn H., Reinfrank M., *An Introduction to Fuzzy Control*, Springer-Verlag, 1993

[5] Iacopetti A., *Logica fuzzy ed algoritmi genetici*, da Internet, 1998

[6] Kosko B., *Il fuzzy-pensiero – Teoria e applicazioni della logica fuzzy*, Baldini e Castoldi, 1997

[7] Marks R.J. II (a cura di), *Fuzzy Logic Technology and Applications*, IEEE Technology Update Series, 1994

[8] Pedrycz W., *Fuzzy control and fuzzy systems – second, extended, edition*, Research Studies Press LTD, 1995

[9] Power J., Ryan M., Yan J., *Using fuzzy logic*, Prentice Hall International, 1994

[10] Von Altrock C, *Fuzzy Logic & Neurofuzzy Applications in Business & Finance*, Prentice Hall PTR, 1997

[11] Zecchel A., *Controllori Fuzzy: fondamenti ed applicazioni agli azionamenti in corrente continua*, Tesi di laurea, Padova, anno acc. 1991-92

[12] Zimmermann H.-J., *Fuzzy Set Theory – and Its Applications, Second, Revised Edition*, Kluwer Academic Publishers, 1991

Gestione scorte

[13] Anderson E.J., *The Management of Manufacturing – Models and Analysis*, Addison-Wesley, 1994

[14] Berry W.L., Vollmann T.E., Whybark D.C., *Manufacturing Planning and Control Systems*, Business One Irwin, Homewood, 1992

[15] Marini G., Presutti A., "La distribuzione fisica del prodotto", Boario M., De Martini M., Di Meo E., Gros-Pietro G.M. (a cura di), *Manuale di Logistica*, UTET, Torino, 1992

[16] Schmenner R., *Produzione*, Edizioni del Sole 24 Ore, Milano, 1987

[17] Tronconi A., "La gestione strategica delle scorte", *Meccanica e macchine di qualità - rivista della sicurezza e della certificazione*, n. 5, 1998

Altro

[18] AA. VV., *Enciclopedia dell'economia*, Garzanti, 1992

[19] Duarte O.G., *Unfuzzy – Manual del usuario*, da Internet, 1998

[20] Fabricky W.J., Thuesen G.J., *Economia per ingegneri*, il Mulino/Prentice Hall International, 1994

[21] Frantz G., *Programmare con Visual Basic*, Apogeo, Milano, 1992