

Xpress-MP Un linguaggio per la Programmazione Matematica

di **Roberto Bello** www.freeopen.org

con riferimenti tradotti da:

Getting Started - Release 2006

Last update 6 April, 2006

Published by **Dash Optimization Ltd**

<http://www.dashoptimization.com/>

libro completo: www.lulu.com/content/1970938

1. Premessa

Si presuppone che il lettore abbia già una buona conoscenza della teoria della programmazione lineare e sia già in grado, nella pratica, di realizzare dei piccoli modelli risolvendoli in modalità grafica o meglio ricorrendo alle apposite funzioni disponibili nei programmi di elaborazione delle tabelle di calcolo di Excel o di OpenOffice.org.

Questo documento ha l'obiettivo di illustrare come utilizzare il programma Xpress-MP per costruire e risolvere dei modelli di programmazione lineare anche in ambiti professionali.

Il programma è disponibile in versione **studente** al link:

www.dashoptimization.com/home/products/evaluation/student_request.html

I limiti della **versione studente (gratuita)** sono abbastanza ampi, tali da consentire la progettazione di modelli utilizzabili anche in ambito aziendale.

Essi sono:

- vincoli (righe) = 400
- variabili (colonne) = 80
- elementi matrice = 5000
- variabili binarie = 400

E' richiesta la registrazione per ottenere la password per l'utilizzo del software; il manuale di uso e molti esempi di modelli applicativi sono ugualmente scaricabili.

Dopo la registrazione si riceve una email:

Thank you for downloading Xpress-IVE (student license). Please note that this software is only for use by students in full-time or part-time education studying for a recognized qualification.

To install the software, please extract the installation kit from the zip archive you have downloaded. You will be asked for a password, which is:

studentonly

The Getting Started manual tells you how to use Xpress-MP. For a complete language reference, please refer to the Mosel Reference Manual.....

Il vantaggio principali di XPRESS sono quelli di:

- rappresentare il modello di programmazione lineare in forma matematica con ampio utilizzo di riferimenti sintetici e simbolici a matrici e a vettori
- separare la logica del modello dai dati di ingresso, rendendo il modello applicabile anche a dati di input con diversa numerosità delle variabili interessate
- integrare nel modello filtri e condizioni particolari di calcolo
- scegliere le modalità di rappresentazione dei risultati e dei formati di output.

2. Contenuti

Il documento contiene una concisa e facile introduzione alla modellistica e alla risoluzione dei diversi tipi di ottimizzazione con Xpress-MP.

Sono trattati problemi di Programmazione Lineare, di Programmazione a variabili miste e di Programmazione Quadratica, tutti risolti attraverso l'utilizzo di un programma ad interfaccia grafica di nome Xpress-IVE.

Sono disponibili due alternative:

- utilizzare un linguaggio di programmazione che utilizza una libreria di costruzioni dei modelli di nome Xpress BCL
- inserire direttamente l'input in Optimizer in forma di matrice

In questo documento sono illustrate delle variazioni di un unico problema che si riferisce alla scelta di un portafoglio titoli.

Chi fosse interessato ad altri tipi di applicazione, può consultare 'Applications of Optimization with Xpress-MP' (Dash Optimization, 2002), vedere anche: http://www.dashoptimization.com/applications_book.html

Questo documento mostra come formulare e risolvere un ampio numero di problemi di programmazione matematica con Xpress.

Altra documentazione (guide utente, manuali di riferimento, linguaggio Mosel, moduli Mosel, suggerimenti di modellistica, ecc.) è disponibile nel sito già citato.

3. Programmazione Matematica

La Programmazione Matematica è una tecnica di ottimizzazione matematica.

Molti problemi nella vita reale, ad esempio nella produzione industriale, nei trasporti, nelle telecomunicazioni, nelle finanze, nella pianificazione delle risorse, possono essere espressi in modelli di Programmazione Matematica composti da insiemi di variabili di decisione e di funzione obiettivo da massimizzare o da minimizzare.

I problemi di Programmazione Matematica sono usualmente classificati nell'ambito della Programmazione Lineare quando tutti i vincoli e la funzione obiettivo sono espressioni lineari delle variabili di decisione e quando le variabili possono assumere dei valori continui.

Altre volte le variabili continue non sono adatte a rappresentare decisioni di carattere discreto (sì / no oppure 1,2,3,...) e allora vengono in aiuto gli algoritmi Mixed Integer Programming (MIP) nei quali i vincoli e la funzione obiettivo sono lineari, come nella Programmazione Lineare, mentre le variabili possono assumere sia valori continui sia valori discreti.

Per risolvere problemi di questo tipo, le tecniche di Programmazione Lineare sono accompagnate da quelle di tipo enumerativo (conosciute come Branch-and-Bound) per la ricerca dei valori discreti ammissibili.

I metodi di tipo enumerativo possono comportare tempi esagerati di calcolo anche per problemi relativamente modesti, impedendo di fatto la soluzione ottimale dei modelli di tipo MIP.

Negli ultimi anni, il continuo miglioramento della velocità dei computer e degli ancora più significativi progressi negli algoritmi (ad esempio tecniche del *cutting plane* e *specialized branching schemes*) hanno reso possibile gestire problemi sempre più complessi e rappresentativi di problemi reali.

Altre categorie di problemi relativamente ben gestiti sono rappresentate dalla Programmazione Quadratica (QP), problemi che differiscono da quelli della Programmazione Lineare nel fatto che essi hanno termini quadratici nella funzione obiettivo, mentre i vincoli restano lineari.

Le variabili di decisione possono essere sia continue sia discrete; in quest'ultimo caso si parla di Mixed Integer Quadratic Programming (MIQP)

E' molto difficile gestire i problemi con vincoli non lineari (NLP) Non-linear Programming. Frequentemente sono impiegati metodi euristici o approssimativi per trovare *buone soluzioni* (ottimi locali). Un metodo per risolvere problemi di questo tipo è Successive Linear Programming (SLP) che è anche una tecnica di soluzione compresa nei programmi di Xpress-MP. Comunque in questo documento non ci sono approfondimenti su questa tecnica.

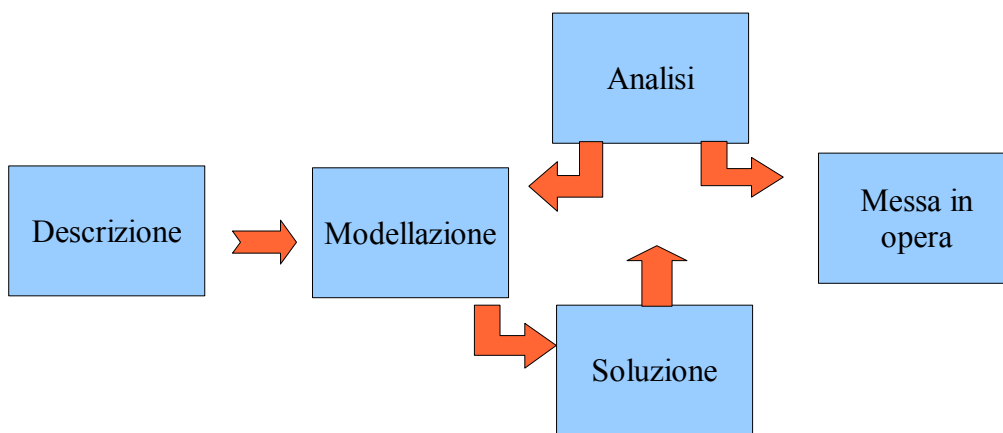
Costruire un modello, risolverlo e poi applicarlo al mondo reale, non sono

attività che si svolgono in modo lineare.

Si fanno spesso degli errori nella fase di modellazione che vengono rilevati solo dal processo di modellazione (esempio modello *unbounded* oppure *infeasible*) oppure che sono in contrasto con una nostra ferma intuizione.

Quando ciò accade siamo obbligati a riflettere sul modello, aggiornarlo, ricalcolarlo ed analizzare nuovamente i risultati ottenuti.

Nel corso di questo processo è normale aggiungere dei vincoli, togliere vincoli non opportuni, correggere dati errati, inserire nuovi dati prima ritenuti non necessari.



Questo documento accompagna il lettore nel percorso:

- dalla descrizione testuale allo sviluppo di un modello matematico da risolvere
- proposte di diversi miglioramenti, aggiunte e riformulazioni nei capitoli successivi
- esame dei mezzi disponibili per analizzare i risultati ottenuti

4. Xpress-MP Generalità

In relazione alle diverse esigenze e preferenze degli utilizzatori, esistono molti modi per utilizzare gli strumenti di modellazione e di ottimizzazione che compongono Xpress-MP.

4.1. Linguaggio di alto livello

Il linguaggio Xpress-Mosel permette all'utente di definire i suoi modelli in una

forma molto simile alla normale notazione algebrica e di risolvere il modello restando nello stesso ambiente applicativo.

L'ambiente di sviluppo integrato Xpress-IVE consente, fra le altre cose, la rappresentazione anche grafica delle soluzioni trovate.

Attraverso il concetto dei moduli l'ambiente Mosel è interamente aperto alle aggiunte; i moduli resi disponibili da Dash consentono l'accesso alle diverse tecniche di risoluzione delle categorie dei modelli (Xpress-Optimizer per LP, MIP, e QP, Xpress-SLP, Xpress-SP), funzioni per la gestione dei dati (esempio tramite ODBC) e accesso alle funzioni di sistema. Inoltre, tramite la Mosel Native Interface, gli utenti possono definire i loro personali moduli per aggiungere nuove funzionalità (ad esempio per gestire problemi specifici di trasformazione dei dati, oppure per collegarsi a programmi risolutori o algoritmi esterni).

4.2. Librerie per le inclusioni

Sono disponibili due diverse opzioni per includere dei modelli matematici in applicazioni di grandi dimensioni.

Un modello sviluppato usando il linguaggio Mosel può essere eseguito e richiamato da un altro ambiente di programmazione (esempio C, C++, Java, etc.) tramite le librerie Mosel; alcuni moduli consentono anche un accesso diretto alle loro funzioni dall'ambiente del linguaggio di programmazione.

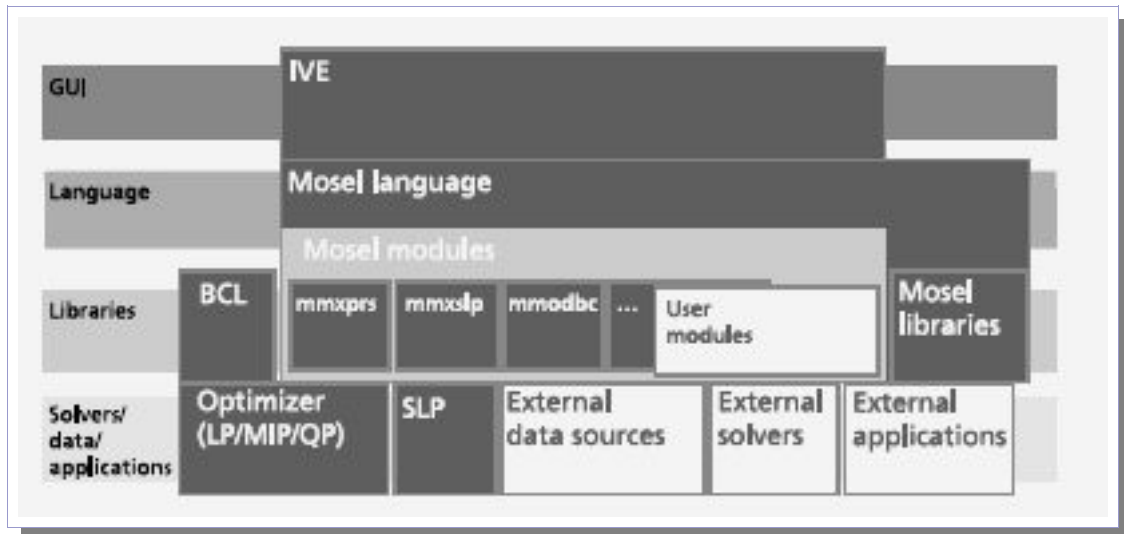
La seconda possibilità consiste nello sviluppare un modello direttamente in un linguaggio di programmazione con l'aiuto di una libreria per la costruzione dei modelli Xpress-BCL.

BCL consente all'utente di formulare i suoi modelli usando *oggetti* (variabili di decisione, vincoli, vettori, matrici) simili a quelli degli altri linguaggi di modellazione.

Tutte le librerie sono disponibili per C, C++, Java, e Visual Basic (VB).

4.3. Accesso diretto ai risolutori (solvers)

Al più basso e immediato livello è possibile lavorare direttamente con Xpress-Optimizer oppure con Xpress-SLP nella forma di libreria oppure di programma indipendente (*standalone*).



Questa funzionalità può essere utile per includere *Optimizer* nelle applicazioni che possiedono le loro procedure di generazione delle matrici.

Fra queste tre menzionate caratteristiche il linguaggio di alto livello consente certamente di rendere più facile la comprensione della Programmazione Matematica.

Nella prima e più estesa parte di questo documento si descrive come definire e risolvere i problemi utilizzando il linguaggio Xpress-Mosel e anche come includere i modelli risultanti nelle applicazioni dell'utente utilizzando le librerie di Mosel.

E' possibile lavorare con i modelli di Mosel per mezzo dell'interfaccia grafica utente di Xpress-IVE che consente anche l'esame passo a passo (debugging), l'analisi della soluzione nelle singole parti componenti.

Nella parte restante di questo documento si mostra come formulare e risolvere problemi di Programmazione Matematica direttamente nell'ambiente del linguaggio di programmazione.

Ciò può essere realizzato utilizzando il supporto di modellazione fornito da BCL oppure direttamente usando la libreria di *Xpress-Optimizer*.

Con BCL i modelli possono essere sviluppati in una forma simile a quella della normale formulazione algebrica, consentendo una facile comprensione e gestione.

Nel documento le realizzazioni in BCL fanno riferimento agli stessi esempi utilizzati con Mosel.

L'ultima parte del documento spiega come i problemi possono essere introdotti direttamente in *Optimizer*, sia in forma di matrici (eventualmente create da un altro strumento come Mosel o BCL), sia di provenienza da archivi oppure specificando i coefficienti delle matrici direttamente nel programma applicativo. La possibilità di operare direttamente con la libreria di *Optimizer* è riservata ad utenti esperti di *Xpress-MP*.

4.4. Note sulle versioni utilizzate

Tutti gli esempi inclusi in questo documento sono stati risolti usando Xpress-MP versioni (Mosel 1.6.2, IVE 1.17.2, BCL 3.0.1, *Optimizer* 17.0.2). Se le elaborazioni fossero eseguite con versioni diverse i risultati potrebbero essere diversi.

5. Creare modelli

Questo capitolo mostra in dettaglio come convertire la descrizione testuale di un problema in un modello matematico.

E' usato, a titolo di esempio, la scelta della composizione di un portafoglio di titoli, esempio che sarà usato ulteriormente nel documento.

Sebbene non sia richiesta alcuna esperienza di Programmazione Matematica per la creazione dei modelli, si presume che il lettore sia pratico con l'uso dei simboli come x oppure y per rappresentare quantità ignote da usare come variabili nelle equazioni e disequazioni, per esempio:

$$x+y \leq 6$$

che significa 'la quantità rappresentata da x con aggiunta la quantità rappresentata da y deve essere minore o uguale a sei'.

Glie deve anche essere familiare il concetto di somma di un insieme di variabili; ad esempio se $produzione_i$ è usato per rappresentare la quantità prodotta del prodotto i , allora la produzione totale di tutti gli n prodotti può essere scritta come:

$$\sum_{i=1}^n produzione_i$$

Un altro simbolo matematico comune è \forall (da leggere come *per tutti*)

Se l'insieme fosse composto dagli elementi 1, 4, 7, 9 allora l'espressione:

$$\forall i \in \text{ITEMS} : produzione_i \leq 100$$

è la forma ridotta di:

$$\begin{aligned} & produzione_1 \leq 100 \\ & produzione_4 \leq \dots \\ & produzione_7 \leq \dots \\ & produzione_9 \leq 100 \end{aligned}$$

I linguaggi di modellazione e in particolare Mosel in esame, imitano la notazione matematica che l'analista di solito utilizza per descrivere i problemi facilitando quindi la comprensione e la predisposizione all'uso del linguaggio

Mosel.

6. Esempio di Problema

Un investitore desidera investire un certo ammontare di denaro fra dieci diverse aree di investimento per le quali stima il rendimento ad un anno di distanza.

La tabella che segue fornisce, per ogni area di investimento, il paese di origine, la categoria di rischio (R: alto rischio, N: basso rischio) e il risultato atteso (return on investment ROI).

L'investitore specifica alcuni vincoli.

Per contenere il rischio egli vuole investire al massimo il 30% del capitale in una singola area.

Inoltre egli vuole investire al minimo la metà del capitale nelle aree del Nord America e al massimo un terzo del capitale nelle aree ad alto rischio.

La domanda è come il capitale debba essere ripartito fra le aree di investimento per ottenere il massimo profitto nel rispetto dei vincoli imposti.

Numero	Descrizione	Origine	Rischio	ROI %
1	treasury (finanza)	Canada	N	5
2	hardware (hardware)	USA	R	17
3	theater (teatro)	USA	R	26
4	telecom (comunicazioni)	USA	R	12
5	brewery (birra)	UK	N	8
6	highways (autostrade)	France	N	9
7	cars (automobili)	Germany	N	7
8	bank (banche)	Luxemburg	N	6
9	software (software)	India	R	31
10	electronics (elettronica)	Japan	R	21

Per costruire un modello matematico dobbiamo, per prima cosa, identificare le decisioni che devono essere prese per ottenere una soluzione: nel caso in esame si desidera conoscere l'ammontare di denaro da investire nelle aree in grado di dare il massimo rendimento. Occorre quindi definire le frazioni delle variabili di decisione che rappresenteranno la frazione del capitale investito nell'area s .

Ciò significa che le variabili di decisione assumeranno valori compresi fra 0 e 1 (dove 1 corrisponde al 100% del capitale totale).

In realtà, ogni variabile di decisione non può assumere un valore superiore al 30% del capitale totale da investire.

La condizione è espressa dal vincolo che segue:

$$\forall s \in \text{SHARES} : 0 \leq \text{frac}_s \leq 0.3$$

dove la variabile frac_s si legge per tutte s in SHARE.

Nella formulazione matematica si scrive SHARES per l'insieme dei possibili investimenti, si scrive RETs per i ritorni attesi ROI per forma di investimento, si scrive NA per il sottoinsieme degli investimenti di origine Nord America, si scrive RISK per il sottoinsieme degli investimenti in relazione al rischio.

L'investitore vuole spendere tutto il suo capitale e quindi la sommatoria degli investimenti deve essere pari al 100% del capitale.

Questa condizione è espressa dal vincolo:

$$\sum_{s \in \text{SHARES}} \text{frac}_s = 1$$

E' necessario esprimere i due vincoli specificati dall'investitore.

Al massimo un terzo del capitale può essere investito in aree di alto rischio; cioè la somma investita in questa categoria non deve eccedere 1/3 del capitale totale:

$$\sum_{s \in \text{RISK}} \text{frac}_s \leq 1/3$$

L'investitore desidera spendere almeno il 50% del capitale in titoli del Nord America:

$$\sum_{s \in \text{NA}} \text{frac}_s \geq 0.5$$

Questi due ultimi vincoli sono delle disequaglianze.

L'obiettivo dell'investitore è quello di massimizzare il rendimento dell'investimento sui titoli delle aree scelte; in altri termini è quello di massimizzare la seguente somma:

$$\sum_{s \in \text{SHARES}} \text{RET}_s * \text{frac}_s$$

Questa è la funzione obiettivo del nostro modello matematico.

Raccogliendo le diverse parti, si ottiene la seguente formulazione completa del modello matematico.

$$\text{massimizzare} \quad \sum_{s \in \text{SHARES}} RET_s * frac_s$$

$$\text{con i vincoli} \quad \sum_{s \in \text{RISK}} frac_s \leq 1/3$$

$$\sum_{s \in \text{NA}} frac_s \geq 1.0$$

$$\sum_{s \in \text{SHARES}} frac_s = 1$$

$$\forall s \in \text{SHARES} : 0 \leq frac_s \leq 0.3$$

Nel prossimo capitolo è descritto come trasformare il modello matematico in un modello Mosel che può essere risolto con *Xpress-Optimizer*. In un altro capitolo è descritto come usare BCL per questo scopo ed inoltre come introdurre direttamente questo modello in *Optimizer* senza il supporto delle funzioni di modellazione.

7. Definire e risolvere un problema di Programmazione Lineare

In questo capitolo si riprende l'esempio formulato nel capitolo precedente e si mostra come trasformarlo in un modello di Mosel che poi è risolto usando Xpress-IVE.

Più precisamente ciò comporta seguire queste fasi:

- mandare in esecuzione Xpress-IVE,
- creare e salvare l'archivio del modello in formato Mosel,
- usare il linguaggio Mosel per esaminare il modello,
- correggere gli eventuali errori e modificare il modello,
- risolvere il modello e comprendere i risultati dell'ottimizzazione ,
- visualizzare e verificare la soluzione verso gli obiettivi reali attesi

7.1. Eseguire Xpress-IVE e creare un nuovo modello

Si spiega come sviluppare ed eseguire il nostro modello Mosel con l'ambiente grafico Xpress-IVE.

Se avete seguito la procedura standard di installazione di Xpress-IVE, per mandarlo in esecuzione basta cliccare due volte sull'icona presente sul desktop oppure scegliere *Start >> Programs >> Xpress-MP >> Xpress-IVE*. Oppure potete dar inizio a Xpress-IVE digitando in una finestra DOS di window la stringa *IVE* (nome del programma eseguibile), oppure ancora cliccare due volte sul nome dell'archivio contenente il modello (archivio con estensione *.mos*).

Xpress-IVE si apre con una finestra suddivisa in diversi pannelli:

In testa esiste il menu principale e la barra degli attrezzi.

La finestra alla sinistra è la barra dei progetti, sul fondo è collocata la barra delle informazioni, sulla destra è presente la barra delle esecuzioni.

Si possono mescolare queste barre usando il menu *View* oppure con gli appositi bottoni a ciò predisposti:

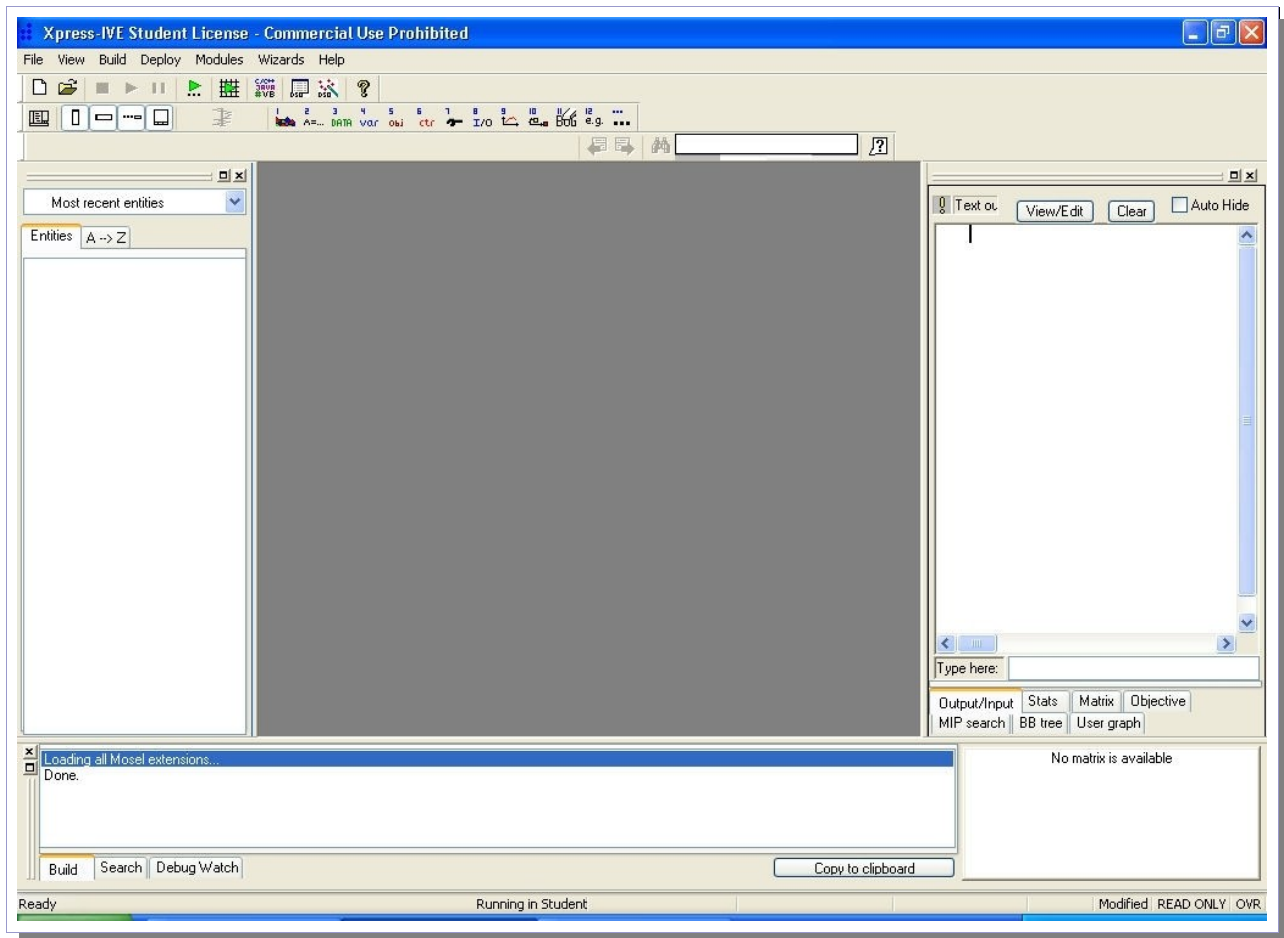


Per ritornare alla configurazione originale delle barre scegliere *View >> Repair Window Layout*.

Per creare un nuovo modello scegliere *File >> New* oppure cliccare sul primo bottone della barra degli strumenti:



Ciò consente di aprire una finestra di dialogo dove scegliere una cartella (directory) e digitare il nome del nuovo archivio. Daremo come nome **foliopl**; l'estensione **.mos** per tipo di archivio 'Mose!' è aggiunta automaticamente. Cliccare *Save* per confermare la scelta. La finestra centrale di IVE diventa di colore bianco e il cursore è posizionato sulla prima linea, pronto per inserire la definizione del modello, descrizione controllata dal tracciato reso disponibile da IVE.



7.2. Modello LP

Il modello matematico descritto al punto precedente può essere trasformato nel modello Mosel che segue utilizzando IVE:

```

model "Portfolio optimization with LP"
uses "mmxprs"           ! Use Xpress-Optimizer
declarations
  SHARES = 1..10        ! Set of shares
  RISK = {2,3,4,9,10}   ! Set of high-risk values among shares
  NA = {1,2,3,4}        ! Set of shares issued in N.-America
  RET: array(SHARES) of real ! Estimated return in investment
  frac: array(SHARES) of mpvar ! Fraction of capital used per share
end-declarations
RET:= [5,17,26,12,8,9,7,6,31,21]
! Objective: total return
Return:= sum(s in SHARES) RET(s)*frac(s)
! Limit the percentage of high-risk values
sum(s in RISK) frac(s) <= 1/3
! Minimum amount of North-American values
sum(s in NA) frac(s) >= 0.5

```

```

! Spend all the capital
sum(s in SHARES) frac(s) = 1
! Upper bounds on the investment per share
forall(s in SHARES) frac(s) <= 0.3
! Solve the problem
maximize(Return)
! Solution printing
writeln("Total return: ", getobjval)
forall(s in SHARES) writeln(s, ": ", getsol(frac(s))*100, "%")
end-model

```

Di seguito le spiegazioni di ciò che abbiamo appena scritto.

7.3. Struttura generale

Ogni programma Mosel inizia con la parola chiave *model*, seguita dal nome del modello scelto dall'utente. Il programma Mosel termina con la parola chiave *end-model*.

Tutti gli oggetti devono essere dichiarati in una sezione chiamata *declarations*, salvo che gli stessi siano già dichiarati in modo non ambiguo tramite delle assegnazioni.

Per esempio:

$$\text{Return} := \text{sum}(s \text{ in } \text{SHARES}) \text{RET}(s) * \text{frac}(s)$$

definisce *Return* come un vincolo lineare ed assegna ad esso il valore dell'espressione

$$\text{sum}(s \text{ in } \text{SHARES}) \text{RET}(s) * \text{frac}(s)$$

Possono esistere diverse sezioni di *declarations* in posti differenti del modello. In questo esempio sono definiti tre insiemi e due matrici:

- *SHARES* è un *range set*, un insieme di numeri interi consecutivi (da 1 a 10).
- *RISK* e *NA* sono dei semplici insiemi di valori interi.
- *RET* è una matrice di numeri reali indirizzati dal *set* *SHARES*, i valori sono assegnati dopo le *declarations*.
- *frac* è una matrice di variabili di decisione di tipo *mpvar*, ancora indirizzato dal *set* *SHARES*.

Nel modello sono indicate anche le variabili di decisione.

Il modello poi contiene la funzione obiettivo, due disequazioni, un'eguaglianza e definizioni del limite superiore di variabili di decisione.

Come nei modelli matematici si utilizza un ciclo *forall loop* per considerare

tutte le occorrenze dell'insieme *SHARES*.

7.4. Soluzione

Con la procedura *maximize*, chiediamo a Xpress-Optimizer di massimizzare l'espressione lineare *Return*.

Poiché Mosel non è un programma di risoluzione, si indica a Xpress-Optimizer quale tecnica di ottimizzazione usare, fra quelle adatte al modello, indicandone il nome dopo la parola chiave *uses* all'inizio del modello: nel nostro caso "mmxprs".

Invece di definire la funzione obiettivo *Return* separatamente, è possibile definirla insieme alle altre disequazioni nella forma:

$$\text{maximize}(\text{sum}(s \text{ in } \text{SHARES}) \text{RET}(s) * \text{frac}(s))$$

7.5. Stampa dei risultati

Le due ultime linee:

```
writeln("Total return: ", getobjval)
forall(s in SHARES) writeln(s, ": ", getsol(frac(s))*100, "%")
```

hanno lo scopo di stampare il valore delle variabili di decisione della soluzione ottimale.

Per stampare delle addizionali righe vuote, inserire il comando *writeln* (senza argomenti).

Per scrivere più valori in una singola riga usare *write* invece di *writeln*.

7.6. Formati (Formating)

Rientri, spazi, righe vuote possono essere aggiunti nel nostro modello per aumentarne la leggibilità.

Essi sono poi trascurati da Mosel.

E' anche possibile porre più istruzioni in una singola riga separandole con il carattere ; (punto e virgola), come ad esempio:

$$\text{RISK} = \{2,3,4,9,10\}; \text{NA} = \{1,2,3,4\}$$

poiché non è previsto alcun comando di *fine linea* oppure di caratteri di continuazione, ogni istruzione che occupa più righe, deve terminare con un operatore (+, >=, ecc.) oppure con un carattere virgola `, ' segnalante che l'istruzione non è terminata.

Come mostrato nell'esempio, la singola riga di commento ha, come primo

carattere ! (punto esclamativo).

Commenti su più righe devono avere un punto esclamativo all'inizio ed un punto esclamativo alla fine del testo.

7.7. Correzione degli errori e messa a punto

Avendo immesso in IVE il modello sopra descritto, ora possiamo cercare di trovare la soluzione ottimale vedendone i risultati.

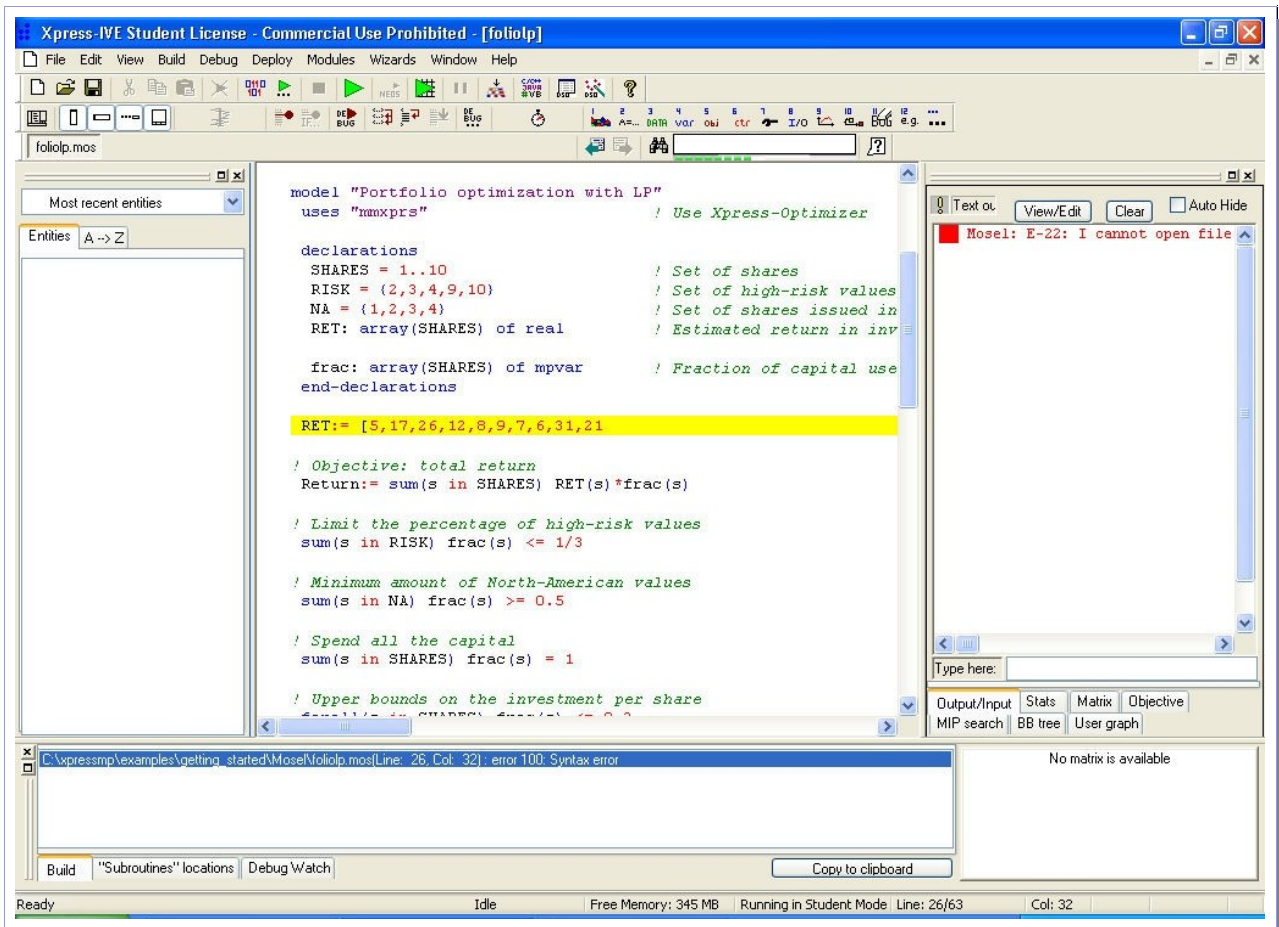
Scegliere *Build >> Run* oppure in alternativa cliccare sul bottone *run*



E' normale che la prima esecuzione del modello generi degli errori con l'emissione del messaggio: *Compilation failed. Please check for errors.*

La finestra nel fondo mostra la natura degli errori trovati da Mosel, come nell'esempio della figura sotto riportata.

Per creare l'errore di sintassi nel modello prima descritto, è stato deliberatamente introdotto un comune errore formale cancellando la parentesi quadra di chiusura dell'istruzione di definizione dei valori dei rendimenti nel vettore *RET*. Cliccando sul testo del messaggio di errore, è evidenziata in giallo l'istruzione che ha provocato l'errore trovato.



7.8. Barra informativa dei messaggi degli errori

Il messaggio:

error 100: Syntax error

si riferisce alla riga

RET:= [5,17,26,12,8,9,7,6,31,21

E' necessario aggiungere la chiusura della parentesi quadra al termine della definizione di *RET*; se la definizione dovesse continuare nella riga successiva, sarebbe necessario aggiungere una , (virgola) alla termine della prima riga per indicare che la definizione continua.

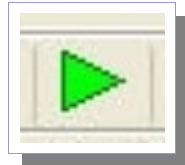
Gli eventuali messaggi di attenzione (*warning*) non compromettono l'effettuazione del calcolo, calcolo che però potrebbe produrre risultati non corretti come nel caso di uso di =. invece di := operatori entrambi validi , ma con risultati diversi.

7.9. Soluzione, visualizzazione dei risultati

Come già descritto, per eseguire il nostro modello dobbiamo scegliere:

Build >> Run

oppure cliccare sul bottone:



e la finestra cambia come sotto riportato:

The screenshot displays the Xpress-IVE Student License interface. The main window shows a model named "Portfolio optimization with LP" with the following code:

```
model "Portfolio optimization with LP"
uses "mmaxprs"           ! Use Xpress-Optimizer

declarations
  SHARES = 1..10          ! Set of shares
  RISK = {2,3,4,9,10}     ! Set of high-risk values
  NA = {1,2,3,4}         ! Set of shares issued in
  RET: array(SHARES) of real ! Estimated return in inv
end-declarations

frac: array(SHARES) of mpvar ! Fraction of capital use

RET:= [5,17,26,12,8,9,7,6,31,21]

! Objective: total return
Return:= sum(s in SHARES) RET(s)*frac(s)

! Limit the percentage of high-risk values
sum(s in RISK) frac(s) <= 1/3

! Minimum amount of North-American values
sum(s in NA) frac(s) >= 0.5

! Spend all the capital
sum(s in SHARES) frac(s) = 1

! Upper bounds on the investment per share
sum(s in SHARES) frac(s) <= 0.2
```

The right-hand pane shows the solution results:

Entity	Value
Total return	14.0667
1	30%
2	0%
3	20%
4	0%
5	6.66667%
6	30%
7	0%
8	0%
9	13.3333%
10	0%

The bottom status bar indicates the model was compiled successfully and is running in Student Mode. The status bar also shows: Ready, Idle, Free Memory: 338 MB, Running in Student Mode, Line: 26/63, Col: 33.

La parte inferiore della finestra contiene il log dell'esecuzione di Mosel, la parte destra della finestra contiene i dati della soluzione del problema, mentre la parte sinistra della finestra contiene l'elenco delle entità del modello.

Scegliere il pulsante *Output/input* presente nella parte destra della finestra per visualizzare i risultati portati in stampa dal programma.

Total return: 14.0667

1: 30%

2: 0%

3: 20%

4: 0%

5: 6.66667%

6: 30%

7: 0%

8: 0%

9: 13.3333%

10: 0%

Significa che il rendimento massimo di 14.0667 è ottenuto con un portafoglio composto da azioni delle aree 1, 3, 5, 6 e 9; il 30% dell'investimento è speso in azioni dell'area 1, il 30% in azioni dell'area 6, il 20% in azioni dell'area 3, il 13.3333% in azioni dell'area 9 e il 6.6667% in azioni dell'area 5.

Si può facilmente constatare che tutti i vincoli sono soddisfatti; infatti il 50% delle azioni sono del Nord America (1 e 3) e il 33.33% sono di alto rischio (3 e 9).

7.10. Statistiche

Premendo il pulsante *Stats* si ottengono dettagliate informazioni sul modello LP risolto.

La parte superiore della finestra contiene delle informazioni riguardo la matrice nella sua forma originale e in quella di *presoluzione* (*prerisolvere* un problema significa applicare dei metodi numerici per semplificare o trasformare il modello).

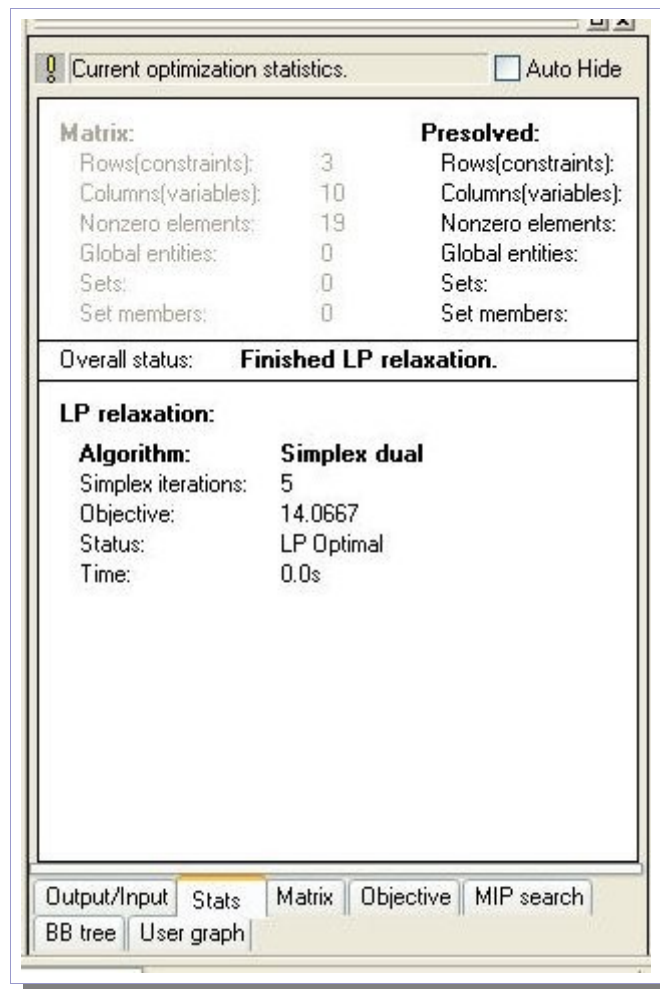
La parte centrale indica quale algoritmo LP è stato usato (Simplex), il numero di iterazioni e il tempo totale impiegato dall'algoritmo per la soluzione.

La parte inferiore indica il tempo aggiuntivo che è stato richiesto per le varie visualizzazioni in IVE.

Considerando che il problema è molto piccolo, esso è risolto quasi istantaneamente.

Per vedere altre informazioni di dettaglio sulla soluzione, si possono esaminare le *entità* nel lato sinistro della finestra espandendo le *decision variables* e i *constraints* cliccando sul segno '+'. Muovendo delicatamente il mouse sulle diverse *entità*, sono visualizzate le diverse informazioni (per le matrici, fare doppio click sull'entità per ottenere una nuova finestra con tutti i diversi valori dell'entità).

Notare che fra i *constraints* è indicata solo la funzione obiettivo poiché è il solo vincolo al quale è stato assegnato un nome.



7.11. Da indici a stringhe di caratteri

Per rendere maggiormente leggibile il modello, può essere una buona idea sostituire gli indici numerici con stringhe di caratteri.

Nel nostro modello modifichiamo:

$$SHARES = 1..10$$

$$RISK = \{2,3,4,9,10\}$$

$$NA = \{1,2,3,4\}$$

con le seguenti linee:

$$SHARES = \{"treasury", "hardware", "theater", "telecom", "brewery", "highways", "cars", "bank", "software", "electronics"\}$$

$$RISK = \{"hardware", "theater", "telecom", "software", "electronics"\}$$

$$NA = \{"treasury", "hardware", "theater", "telecom"\}$$

Per inizializzare la matrice RET è ora necessario usare questi indici:

$$RET("treasury"):=5; RET("hardware"):=17; RET("theater"):=26$$

```
RET("telecom"):=12;      RET("brewery"):=8;      RET("highways"):=9;
RET("cars"):=7
RET("bank"):=6; RET("software"):=31; RET("electronics"):=21
```

Non è necessario apportare altre modifiche al nostro modello.
Salviamo il modello modificato con nome *foliolps.mos*.
I risultati della soluzione del modello modificato sono di interpretazione più immediata:

```
Total return: 14.0667
treasury: 30%
hardware: 0%
theater: 20%
telecom: 0%
brewery: 6.66667%
highways: 30%
cars: 0%
bank: 0%
software: 13.3333%
electronics: 0%
```

Naturalmente anche le entità del modello avranno cambiato nome.

8. Gestire i dati

Si introducono alcune funzionalità di base di Mosel:

- i blocchi di istruzioni per leggere e scrivere i dati nel formato specifico di Mosel,
- i dati in output per archivi a formato libero (*free format*),
- rendere parametrici i nome degli archivi e delle costanti numeriche,
- *formattare* alcuni output .

8.1. Dati di Input provenienti da archivi

Con Mosel sono possibili diverse modalità per leggere (e scrivere) dati provenienti da (e verso) archivi esterni.

Per semplicità ci si limita ora a trattare solo archivi in formato testo.

Mosel, con moduli specifici, consente di scambiare dati con fogli elettronici (*spreadsheets*) e con *database*, per esempio usando la connessione ODBC, ma ciò va oltre l'obiettivo di questo documento.

L'archivio *folio.dat*, che stiamo per esaminare, contiene i seguenti dati:

```
! Data file for 'folio*.mos'
RET: [("treasury") 5 ("hardware") 17 ("theater") 26 ("telecom") 12
```

```

("brewery") 8 ("highways") 9 ("cars") 7 ("bank") 6
("software") 31 ("electronics") 21 ]
RISK: ["hardware" "theater" "telecom" "software" "electronics"]
NA: ["treasury" "hardware" "theater" "telecom"]

```

Come già visto, una linea di commento è preceduta dal carattere ! (punto esclamativo).

Ogni dato in input deve essere in accordo con il nome di *entità* presente nel modello.

I dati di input possono essere separati da spazi (*blank*), tabulazioni (*tab*), separatori di linea (*line breaks*) oppure virgole.

Introduciamo le seguenti modifiche al nostro modello:

```

declarations
  SHARES: set of string      ! Set of shares
  RISK: set of string       ! Set of high-risk values among
shares
  NA: set of string         ! Set of shares issued in N.-America
  RET: array(SHARES) of real ! Estimated return in investment
end-declarations
initializations from "folio.dat"
  RISK RET NA
end-initializations
declarations
  frac: array(SHARES) of mpvar ! Fraction of capital used per share
end-declarations

```

Diversamente dal precedente modello *foliolp.mos*, tutti gli insiemi e le matrici dei dati sono creati come oggetti dinamici: la loro dimensione non è conosciuta al momento della creazione ed essi sono *inizializzati* successivamente alla lettura dell'archivio *folio.dat*.

Se si vuole, dopo l'inizializzazione dall'archivio, possiamo *fissare gli insiemi* rendendoli *statici*.

Si renderebbe molto più efficiente la gestione delle matrici dichiarate successivamente e, molto più importante, si permetterebbe a Mosel di controllare la circostanza dell'errore di fuori capacità (*out of range*), errore non rilevabile nel caso di insiemi dinamici.

```

finalize(SHARES); finalize(RISK); finalize(NA)

```

Notare che non è inizializzato esplicitamente l'insieme *SHARES*, esso è costruito automaticamente quando è letta la matrice *RET*.

Notare anche che si dichiarano le variabili di decisione solo dopo l'inizializzazione dei dati e quindi quando la loro dimensione è nota.

8.2. Dati di output verso gli archivi

Come per l'input, così per l'output, Mosel consente di scrivere i dati in formato standard.

Se si vuole indirizzare il testo mostrato nella finestra di *Output* così come appare, occorre semplicemente includere le istruzioni di stampa fra le procedure *fopen* e *fclose*:

```
fopen("result.dat", F_OUTPUT)
writeln("Total return: ", getobjval)
forall(s in SHARES) writeln(s, ": ", getsol(frac(s))*100, "%")
fclose(F_OUTPUT)
```

Il primo argomento di *fopen* è il nome dell'archivio di output, il secondo (*F_OUTPUT*) indica il modo di apertura dell'archivio; con questa modalità, ogni ulteriore esecuzione comporta la riscrittura dell'archivio.

Per accodare i nuovi risultati a quelli già presenti nell'archivio (esempio *result.dat*) usare l'istruzione:

```
fopen("result.dat", F_OUTPUT+F_APPEND)
```

Per ottenere un output con aspetto più gradevole, si possono usare le seguenti istruzioni:

```
forall(s in SHARES)
    writeln(strfmt(s,-12), ": \t", strfmt(getsol(frac(s))*100,5,2),
"%")
```

La funzione *strfmt* indica lo spazio minimo riservato per la stampa di una stringa di caratteri o di un numero.

Un numero negativo come secondo argomento serve per ottenere un output con i valori allineati a sinistra.

Il terzo argomento opzionale serve per dichiarare il numero dei digit dopo il punto decimale.

Con le specifiche sopra descritte si ottiene il seguente risultato:

```
Total return: 14.0667
treasury      : 30.00%
hardware      :  0.00%
theater       : 20.00%
telecom       :  0.00%
brewery       :  6.67%
highways      : 30.00%
cars          :  0.00%
bank          :  0.00%
```

software : 13.33%
electronics : 0.00%

8.3. Parametri

E' considerato un buon stile di modellazione, codificare manualmente il minor numero di informazioni per descrivere il modello.

Infatti parametri e dati dovrebbero restare esterni e letti da archivi in modo da rendere il modello più versatile e facilmente usabile anche al variare della dimensione del problema.

Con Mosel è possibile definire, per esempio, i nomi degli archivi e le costanti numeriche in forma di parametri, il valore dei quali può essere modificato, prima di ogni esecuzione, senza che sia necessario modificare il modello descrittivo del problema.

Nel nostro esempio possiamo i nomi degli archivi di input e di output, come pure lo possiamo fare per le costanti e per i valori di soglia (valori del vettore di destra della matrice dei coefficienti).

La definizione dei parametri deve essere fatta nelle prime righe del modello, immediatamente dopo l'istruzione *uses*:

```
parameters  
  DATAFILE= "folio.dat"   ! File with problem data  
  OUTFILE= "result.dat"  ! Output file  
  MAXRISK = 1/3          ! Max. investment into high-risk values  
  MAXVAL  = 0.3          ! Max. investment per share  
  MINAM   = 0.5          ! Min. investment into N.-American values  
end-parameters
```

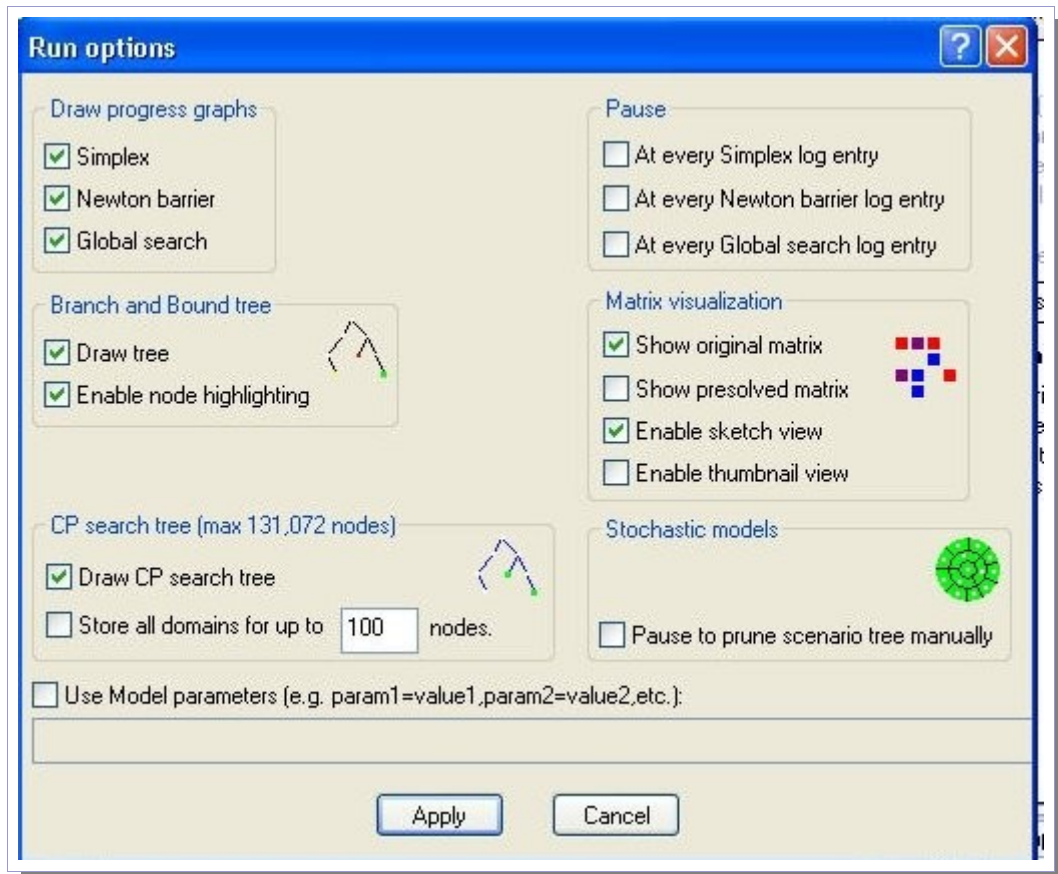
Nel resto del modello i valori dei parametri sostituiscono i corrispondenti valori precedentemente inseriti.

Per modificare i valori di questi parametri, occorre scegliere in IVE l'opzione *Build*; si apre una finestra di dialogo, scegliere *Options* oppure premere il bottone run:



Contrassegnare il campo *Use model parameters* ed inserire i nuovi valori per i parametri nelle righe successive.

Ad esempio per il nome dell'archivio dei risultati ed i valori di *MAXRISK* e di *MAXVAL*.



Si fa notare che l'utilizzo dei parametri diventa veramente importante quando il modello non è più in fase di progettazione, ma piuttosto quando è in fase di prova e di sperimentazione (modalità batch o linea di comando), oppure nella fase di utilizzo finale e operativo.

Nel caso si voglia scrivere una procedura che esegua il nostro modello *foliodata.mos* più volte con differenti valori dei parametri e poi voglia scrivere i risultati ogni volta in un differente archivio, potrebbe aggiungere le seguenti poche righe ad un archivio di procedura (*batch file*), usando poi una versione standalone di Mosel per eseguire il modello richiamando il comando *mosel*:

```
mosel -c "exec foliodata MAXRISK=0.1,OUTFILE='result1.dat'"
mosel -c "exec foliodata MAXRISK=0.2,OUTFILE='result2.dat'"
mosel -c "exec foliodata MAXRISK=0.3,OUTFILE='result3.dat'"
mosel -c "exec foliodata MAXRISK=0.4,OUTFILE='result4.dat'"
```

Un altro vantaggio di usare i parametri è quello che se i modelli sono distribuiti come archivi di tipo BIM (*portable, compiled BInary Model files*), allora essi restano parametrici, senza l'inconveniente di dover rendere visibile il modello, proteggendo così la proprietà intellettuale.

8.4. Esempio completo

L'esempio completo del modello *foliodata.mos* con incluse tutte le funzionalità prima presentate è il seguente:

```
model "Portfolio optimization with LP"
uses "mmxprs"          ! Use Xpress-Optimizer
parameters
  DATAFILE= "folio.dat"    ! File with problem data
  OUTFILE= "result.dat"    ! Output file
  MAXRISK = 1/3            ! Max. investment into high-risk values
  MAXVAL = 0.3            ! Max. investment per share
  MINAM = 0.5              ! Min. investment into N.-American
values
end-parameters
declarations
  SHARES: set of string    ! Set of shares
  RISK: set of string      ! Set of high-risk values among shares
  NA: set of string        ! Set of shares issued in N.-America
  RET: array(SHARES) of real ! Estimated return in investment
end-declarations
initializations from DATAFILE
  RISK RET NA
end-initializations
declarations
  frac: array(SHARES) of mpvar ! Fraction of capital used per share
end-declarations
! Objective: total return
Return:= sum(s in SHARES) RET(s)*frac(s)
! Limit the percentage of high-risk values
sum(s in RISK) frac(s) <= MAXRISK
! Minimum amount of North-American values
sum(s in NA) frac(s) >= MINAM
! Spend all the capital
sum(s in SHARES) frac(s) = 1
! Upper bounds on the investment per share
forall(s in SHARES) frac(s) <= MAXVAL
! Solve the problem
maximize(Return)
! Solution printing to a file
fopen(OUTFILE, F_OUTPUT)
writeln("Total return: ", getobjval)
forall(s in SHARES)
  writeln(strfmt(s,-12), ": \t", strfmt(getsol(frac(s))*100,2,3), "%")
fclose(F_OUTPUT)
```

9. Disegnare grafici

Si mostra come disegnare un grafico con IVE.

Il grafico che vogliamo mostrare è creato da differenti esecuzioni del modello usando diversi valori dei parametri.

Dapprima vedremo un esempio di scrittura di un semplice algoritmo in linguaggio Mosel che prevede:

- re-definizione di vincoli,
- ripetute nuove ottimizzazioni,
- salvataggio delle informazioni delle soluzioni,
- definizioni del grafico: punti, linee, ed etichette,
- semplici passi di esecuzione (cicli e scelte).

9.1. Descrizione estesa del problema

In aggiunta ai dati già considerati, l'investitore ha disponibili anche le stime delle deviazioni dall'atteso rendimento delle azioni.

Con queste informazioni aggiuntive, egli decide di eseguire il modello con differenti limiti sulla presenza di azioni ad alto rischio, rappresentando in forma grafica il rendimento totale verso le deviazioni quali misure del rischio.

Numero	Descrizione	Deviazione
1	treasury (finanza)	0.1
2	hardware (hardware)	19.0
3	theater (teatro)	28.0
4	telecom (telefoni)	22.0
5	brewery (birra)	4.0
6	highways (autostrade)	3.5
7	cars (automobili)	5.0
8	bank (banche)	0.5
9	software (software)	25.0
10	electronics (elettronica)	16.0

9.2. Ottimizzazioni ripetute

Si modifica il modello *foliodata.mos* già visto in precedenza in modo che esso possa essere ottimizzato in modo ripetuto usando diversi limiti di percentuale di rischio.

In dettaglio il modello sarà trasformato per realizzare il seguente algoritmo:

1. Definizione delle parti del modello che devono restare immutate anche quando cambiano i valori dei parametri.
2. Per ogni valore del parametro:
 - Re-definire il limite della percentuale dei valori di alto rischio.
 - Risolvere il problema definito.
 - Se il problema ha una soluzione ammessa: memorizzarne i valori.
3. Disegnare il grafico risultante.

Per memorizzare i valori della soluzione e le deviazioni totali stimate risultanti dopo ogni elaborazione, è necessario dichiarare le seguenti due matrici:

```
declarations
  SOLRET: array(range) of real      ! Solution values (total return)
  SOLDEV: array(range) of real      ! Solution values (average
deviation)
end-declarations
```

Il seguente frammento di codice consente di introdurre un ciclo nelle definizioni dei vincoli relativi alla presenza di azioni ad alto rischio.

Per sovrascrivere ad ogni iterazione le precedenti definizioni, occorre dare un nome a questo vincolo, *Risk*.

Se il vincolo non avesse un nome, allora ad ogni esecuzione del ciclo, sarebbe aggiunto un nuovo vincolo senza alcuna sostituzione del vincolo esistente.

```
ct:=0
forall(r in 0..20) do
  ! Limit the percentage of high-risk values
  Risk:= sum(s in RISK) frac(s) <= r/20
  maximize(Return)          ! Solve the problem
  if (getprobat = XPRS_OPT) then ! Save the optimal solution
value
  ct+=1
  SOLRET(ct):= getobjval
  SOLDEV(ct):= getsol(sum(s in SHARES) DEV(s)*frac(s))
else
  writeln("No solution for high-risk values <= ", 100*r/20, "%")
end-if
end-do
```

Sopra è stata usata la seconda forma del ciclo di *forall*, e cioè *forall/do*. Questa forma deve essere usata quando più istruzioni sono comprese nel ciclo che termina con l'istruzione *end-do*.

Un'altra novità nel codice è la presenza delle istruzioni *if/then/else/end-if*. Hanno il significato di voler considerare i valori solo se l'elaborazione ha trovato una soluzione ottima: occorre verificare che il valore restituito dalla funzione *getprobat* sia *solved to optimality* (rappresentato dalla variabile *XPRS_OPT*).

L'istruzione di selezione possiede due altre forme: *if/then/end-if* e *if/then/elif/then/else/end-if* dove *elif/then* può essere ripetuto più volte.

9.3. Disegnare un grafico

Si potrà ora riunire tutte le informazioni raccolte per disegnare un grafico. Le funzioni grafiche sono fornite dal modulo *mmive* (documentato da *Mosel Language Reference Manual*), modulo che deve essere definito all'inizio del modello aggiungendo la seguente istruzione:

```
uses "mmive"
```

Poi usare le seguenti ulteriori istruzioni per creare un grafico:

```
declarations
  plot1: integer
end-declarations

plot1 := IVEaddplot("Solution values", IVE_BLACK)
forall(r in 1..ct) IVEdrawpoint(plot1, SOLRET(r), SOLDEV(r));
forall(r in 2..ct)
  IVEdrawline(plot1, SOLRET(r-1), SOLDEV(r-1), SOLRET(r), SOLDEV(r))
```

Il grafico richiesto sarà mostrato nella finestra di destra di IVE.

Scegliere il pulsante *User graph* per portarlo in primo piano.

Si avrà il seguente output che, per le interrelazioni dei vari vincoli, non è una linea retta come uno si aspetterebbe a priori.

In aggiunta al primo grafico, si possono anche visualizzare le etichette dei punti che rappresentano i dati di input (*plot2* per le azioni a basso rischio e *plot3* per quelle ad alto rischio):

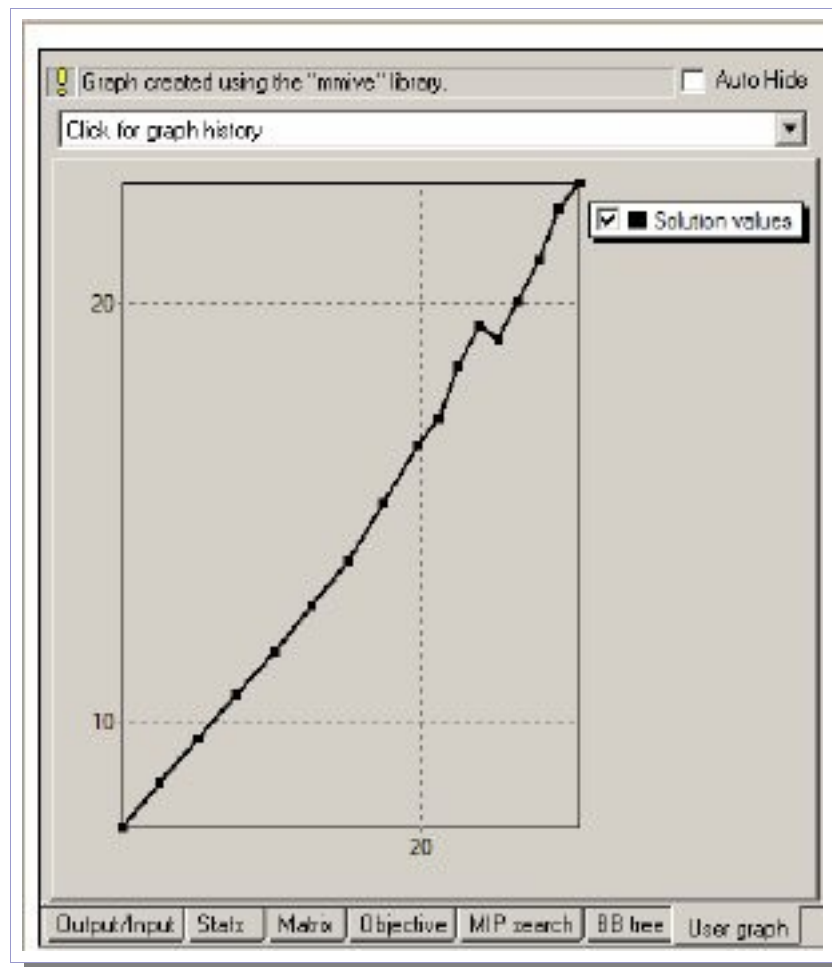
```
declarations
  plot2, plot3: integer
end-declarations
plot2 := IVEaddplot("Low risk", IVE_YELLOW)
forall (s in SHARES - RISK) do
  IVEdrawpoint(plot2, RET(s), DEV(s))
```

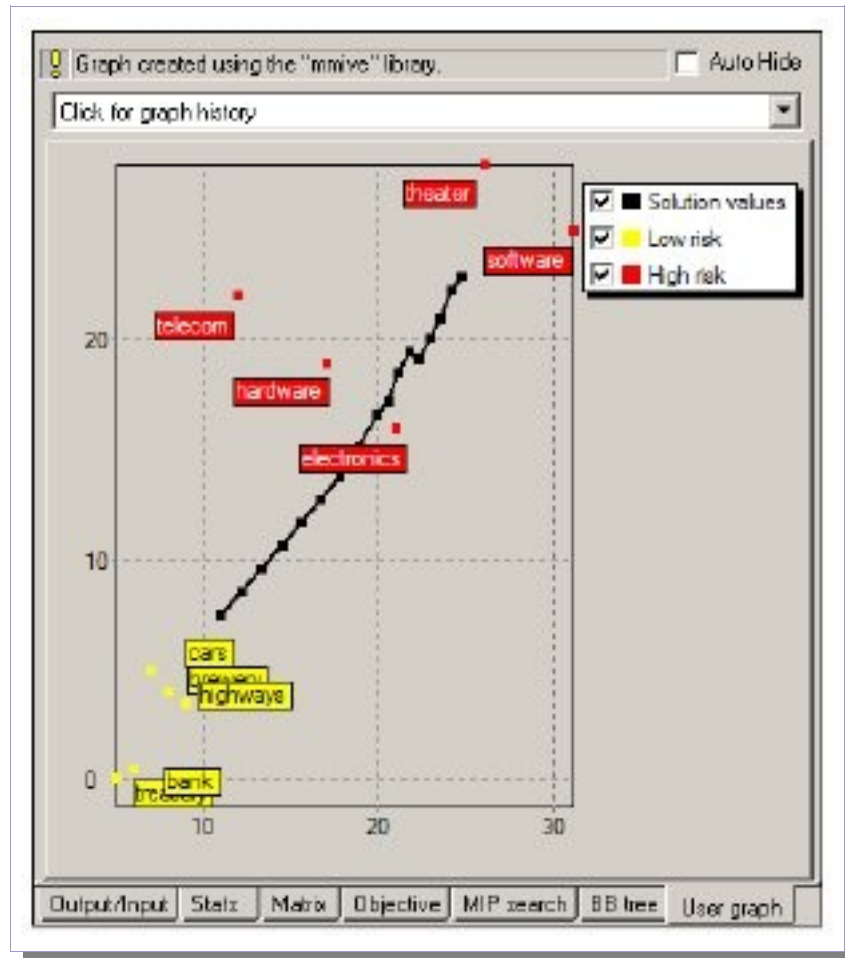
```

    IVEdrawlabel(plot2, RET(s)+3.4, 1.3*(DEV(s)-1), s)
end-do
plot3 := IVEaddplot("High risk", IVE_RED)
forall (s in RISK) do
    IVEdrawpoint(plot3, RET(s), DEV(s))
    IVEdrawlabel(plot3, RET(s)-2.5, DEV(s)-2, s)
end-do

```

Notare che la notazione: *SHARES* - *RISK* significa 'tutti gli elementi di *SHARES* che non sono contenuti in *RISK*'.





9.4. Esempio completo

L'esempio completo è in *foliograph.mos* che contiene tutte le funzionalità appena descritte.

Notare che i due moduli *mmxprs* e *mmive* possono essere definiti in una sola istruzione. I dati delle deviazioni possono essere aggiunti all'archivio originale dei dati oppure, come descritto in questo caso, essere inclusi e letti da un secondo archivio.

```

model "Portfolio optimization with LP"
uses "mmxprs", "mmive"           ! Use Xpress-Optimizer with IVE graphing
parameters
  DATAFILE= "folio.dat"         ! File with problem data
  DEVDATA= "foliodev.dat"       ! File with deviation data
  MAXVAL = 0.3                   ! Max. investment per share
  MINAM = 0.5                    ! Min. investment into N.-American values
end-parameters
declarations
  SHARES: set of string          ! Set of shares

```

```

RISK: set of string          ! Set of high-risk values among shares
NA: set of string           ! Set of shares issued in N.-America
RET: array(SHARES) of real  ! Estimated return in investment
DEV: array(SHARES) of real  ! Standard deviation
SOLRET: array(range) of real ! Solution values (total return)
SOLDEV: array(range) of real ! Solution values (average deviation)
end-declarations
initializations from DATAFILE
    RISK RET NA
end-initializations
initializations from DEVDATA
    DEV
end-initializations
declarations
    frac: array(SHARES) of mpvar    ! Fraction of capital used per share
    Return, Risk: lincpr            ! Constraint declaration (optional)
end-declarations
! Objective: total return
Return := sum(s in SHARES) RET(s)*frac(s)
! Minimum amount of North-American values
sum(s in NA) frac(s) >= MINAM
! Spend all the capital
sum(s in SHARES) frac(s) = 1
! Upper bounds on the investment per share
forall(s in SHARES) frac(s) <= MAXVAL
! Solve the problem for different limits on high-risk shares
ct:=0
forall(r in 0..20) do
    ! Limit the percentage of high-risk values
    Risk := sum(s in RISK) frac(s) <= r/20
    maximize(Return)                ! Solve the problem
    if (getprobstat = XPRS_OPT) then ! Save the optimal solution value
        ct+=1
        SOLRET(ct) := getobjval
        SOLDEV(ct) := getsol(sum(s in SHARES) DEV(s)*frac(s))
    else
        writeln("No solution for high-risk values <= ", 100*r/20, "%")
    end-if
end-do
! Drawing a graph to represent results ('plot1') and data ('plot2' & 'plot3')
declarations
    plot1, plot2, plot3: integer
end-declarations
plot1 := IVEaddplot("Solution values", IVE_BLACK)
plot2 := IVEaddplot("Low risk", IVE_YELLOW)
plot3 := IVEaddplot("High risk", IVE_RED)

```

```

forall(r in 1..ct) IVEdrawpoint(plot1, SOLRET(r), SOLDEV(r));
forall(r in 2..ct)
IVEdrawline(plot1, SOLRET(r-1), SOLDEV(r-1), SOLRET(r), SOLDEV(r))
forall (s in SHARES-RISK) do
    IVEdrawpoint(plot2, RET(s), DEV(s))
    IVEdrawlabel(plot2, RET(s)+3.4, 1.3*(DEV(s)-1), s)
end-do
forall (s in RISK) do
    IVEdrawpoint(plot3, RET(s), DEV(s))
    IVEdrawlabel(plot3, RET(s)-2.5, DEV(s)-2, s)
end-do
end-model

```

Il problema non ha soluzioni per piccoli valori limite del vincolo *Risk*. Dal grafico otteniamo quindi i seguenti messaggi testuali di avviso:

```

No solution for high-risk values <= 0%
No solution for high-risk values <= 5%
No solution for high-risk values <= 10%
No solution for high-risk values <= 15%

```

10. Programmazione Mixed Integer (a variabili miste)

Il solito esempio sarà ora modificato in un problema MIP (Mixed Integer Programming): si descrive come:

- definire differenti tipi di variabili discrete,
- comprendere ed eseguire problemi MIP con IVE.

10.1. Descrizione iniziale del problema

L'investitore potrebbe non gradire di possedere importi modesti di azioni. Ha disponibili due possibilità per esprimere i vincoli necessari:

1. Limitare il numero di azioni differenti presenti nel portafoglio
2. Il titolo può entrare nella soluzione purché il valore minimo (*MINVAL*) non sia al inferiore al 10% del budget totale.

I due successivi modelli hanno lo scopo di descrivere come ottenere quanto sopra indicato.

10.2. Modello 1 – Limitare il numero delle differenti azioni

Per contare il numero delle differenti azioni entrate nella soluzione si deve introdurre un secondo insieme di *variabili di acquisto* di tipo *binario*.

Le variabili di acquisto di tipo *binario* assumono valore 1 se l'azione entra

nella soluzione (non importa l'ammontare), mentre assume il valore 0 se l'azione non entra nella soluzione.

Si crea un nuovo vincolo (*MAXNUM*) per limitare il numero delle diverse azioni che possono entrare nella soluzione del modello.

$$\sum_{s \in \text{SHARES}} buy_s \leq MAXNUM$$

Ora occorre associare le nuove variabili *binarie* con le variabili *frac*, ovvero con le quantità di ogni azione presente nel portafoglio.

La relazione che si vuole esprimere è: *se l'azione è nel portafoglio, allora attiva a valore 1 la variabile buy.*

$$\text{if } frac_s > 0 \text{ then } buy_s = 1$$

La seguente disequazione vuole esprimere proprio questo.

$$\forall s \in \text{ITEMS} : frac_s \leq buy_s$$

Se per qualche s , $frac_s$ è maggiore di zero, allora buy_s deve essere maggiore di zero e quindi uguale a 1 essendo la variabile binaria.

A riprova se buy_s è uguale a zero, allora $frac_s$ è anch'esso uguale a zero, significando che nessuna azione di quel tipo è presente nel portafoglio.

Si fa notare che questi vincoli non impediscono la possibilità che buy_s abbia valore 1 e che $frac_s$ abbia valore zero.

Comunque questa circostanza non ha alcun effetto negativo sulla soluzione che resta comunque valida essendo il valore $frac_s$ uguale a zero.

10.3. Modello 1 – Sviluppo in Mosel

Di seguito la presentazione del modello con le nuove variabili e con i nuovi vincoli.

La circostanza che le nuove variabili siano di tipo binario (con 0 e 1 come unici valori ammessi) è espressa dalla proprietà *is_binary*.

Un altro tipo di variabile discreta è la variabile intera (*integer variable*), che è una variabile che può assumere solo valori interi nell'ambito di un minimo e di un massimo.

Questo tipo di variabile è definito in Mosel dalla proprietà *is_integer*.

Nel seguito, presentando il modello MIP modello 2, vedremo un altro esempio di variabili discrete, chiamate variabili semi-continue (*semi-continuous*

variables).

```
model "Portfolio optimization with MIP"
uses "mmaxprs"           ! Use Xpress-Optimizer
parameters
  MAXRISK = 1/3           ! Max. investment into high-risk values
  MAXVAL = 0.3           ! Max. investment per share
  MINAM = 0.5            ! Min. investment into N.-American values
  MAXNUM = 4             ! Max. number of different assets
end-parameters
declarations
  SHARES: set of string   ! Set of shares
  RISK: set of string     ! Set of high-risk values among shares
  NA: set of string       ! Set of shares issued in N.-America
  RET: array(SHARES) of real ! Estimated return in investment
end-declarations
initializations from "folio.dat"
  RISK RET NA
end-initializations
declarations
  frac: array(SHARES) of mpvar ! Fraction of capital used per share
  buy: array(SHARES) of mpvar ! 1 if asset is in portfolio, 0 otherwise
end-declarations
! Objective: total return
Return:= sum(s in SHARES) RET(s)*frac(s)
! Limit the percentage of high-risk values
sum(s in RISK) frac(s) <= MAXRISK
! Minimum amount of North-American values
sum(s in NA) frac(s) >= MINAM
! Spend all the capital
sum(s in SHARES) frac(s) = 1
! Upper bounds on the investment per share
forall(s in SHARES) frac(s) <= MAXVAL
! Limit the total number of assets
sum(s in SHARES) buy(s) <= MAXNUM
forall(s in SHARES) do
  buy(s) is_binary           ! Turn variables into binaries
  frac(s) <= buy(s)         ! Linking the variables
end-do
! Solve the problem
maximize(Return)
! Solution printing
writeln("Total return: ", getobjval)
forall(s in SHARES)
  writeln(s, ": ", getsol(frac(s))*100, "% (", getsol(buy(s)), ")")
```

end-model

Nel modello sopra riportato *foliomip1.mos* è stata usata la seconda forma dell'istruzione di ciclo *forall* ed esattamente *forall/do*, forma che deve essere usata se nel ciclo sono incluse più istruzioni.

Altrimenti avremmo dovuto scrivere:

```
forall(s in SHARES) buy(s) is_binary
forall(s in SHARES) frac(s) <= buy(s)
```

10.4. Modello 1 – Analizzare la soluzione

Se si interrompe l'esecuzione (usando *Build >> Options* o cliccando sul bottone



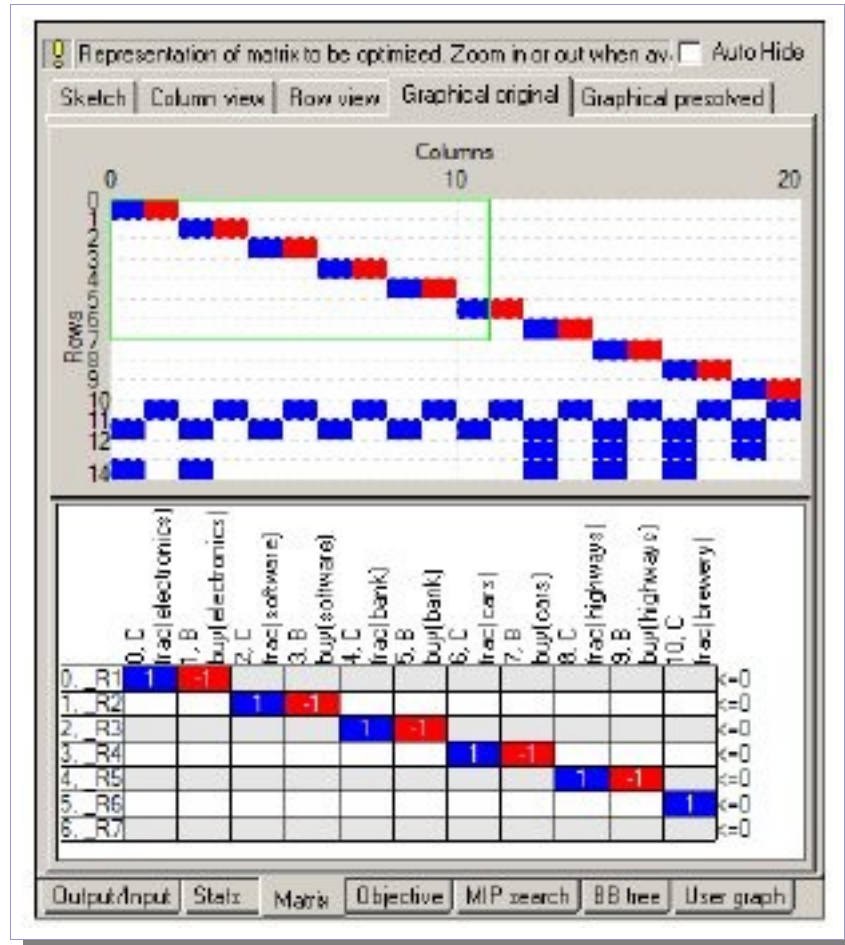
poi sul pulsante *Pause* per visualizzare la matrice) vedremo l'immagine sotto rappresentata.

Ci sono più righe (vincoli) e più colonne (variabili) rispetto alla matrice iniziale del problema LP.

Si noti che il vincolo $frac_s \leq buy_s$ è stato trasformato $frac_s - buy_s \leq 0$ poiché nella matrice tutte le variabili di decisione devono risiedere alla sinistra del segno operativo (anche Mosel memorizza i vincoli in forma normalizzata).

Come risultato dell'esecuzione del modello, (scegliere *Output/input* della finestra della soluzione) si ottiene il seguente output:

Total return:	13.1
treasury:	20% (1)
hardware:	0% (0)
theater:	30% (1)
telecom:	0% (0)
brewery:	20% (1)
highways:	30% (1)
cars:	0% (0)
bank:	0% (0)
software:	0% (0)
electronics:	0% (0)



Il rendimento massimo è ora inferiore a quello del problema originale di tipo LP a motivo dei vincoli addizionali introdotti.

Come richiesto sono presenti nel portafoglio solo 4 differenti tipi di azioni.

Vediamo, nell'immagine sotto riportata, le informazioni di dettaglio della soluzione trovata (pulsante *Stats*).

Ora appare una colonna addizionale, *Global search*, contenente le informazioni specifiche del modello MIP; questa colonna appare il posizione centrale.

Come possiamo vedere è relativamente facile trasformare un modello LP in un modello MIP aggiungendo semplicemente il vincolo del valore intero su alcune (o su tutte) le variabili.

Comunque non lo stesso avviene per l'algoritmo risolutivo: i problemi MIP sono risolti tramite una soluzione ripetitiva di problemi LP.

Inizialmente il problema è risolto senza alcun vincolo di variabili intere.

Poi, una per volta, è scelta una variabile discreta che non soddisfa ancora la condizione di variabile intera e nuovi limiti inferiore e superiore sono fissati per questa variabile per forzarla ad assumere un valore intero.

Se rappresentiamo ogni soluzione LP come un nodo e connettiamo questi nodi con un cambiamento di limiti oppure aggiungiamo dei vincoli, allora otteniamo una struttura ad albero (*Branch-and-Bound tree*).

Current optimization statistics				AutoHide	
Matrix:		Presolved:			
Rows(constraints)	14	Rows(constraints)	14		
Columns(variables)	20	Columns(variables)	20		
Nonzero elements	49	Nonzero elements	49		
Global entities	10	Global entities	10		
Sets	0	Sets	0		
Set members	0	Set members	0		
Overall status: Finished global search.					
LP relaxation:			Global search:		
Algorithm:	Simplex dual	Current node:	1		
Simplex iterations:	14	Depth:	0		
Objective:	14.0687	Active nodes:	-1		
Status:	LP Optimal	Best bound:	13.1		
Time:	0.0s	Best solution:	13.1		
			Gap:	0%	
			Status:	Solution is opt	
			Time:	0.2s	
Time overheads:					
Progress graphs:	0.0s				
Writing output:	0.0s				
Pausing:	0.0s				
Updating status:	0.0s				
<input type="button" value="Output/Input"/> <input type="button" value="Stats"/> <input type="button" value="Matrix"/> <input type="button" value="Objective"/> <input type="button" value="MIP search"/> <input type="button" value="BB tree"/> <input type="button" value="User graph"/>					

In particolare l'informazione *branching* dice quanti nodi *Branch-and-Bound* sono stati necessari per risolvere il problema: nel nostro caso uno solo, la fase di enumerazione non risulta mai partita.

Automaticamente Xpress-Optimizer consente alcuni trattamenti preliminari dei dati, fra i quali la generazione automatica dei tagli (*cuts*): cioè i vincoli aggiuntivi che *riducono lo spazio* delle soluzioni LP, ma nessuna delle soluzioni MIP.

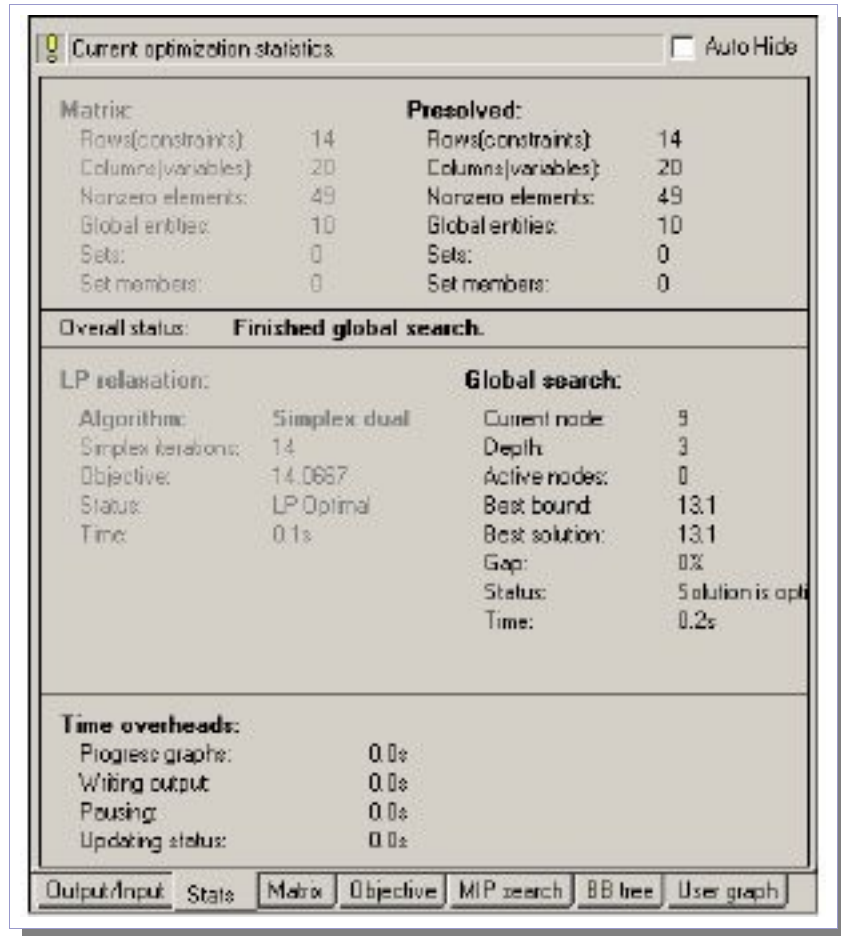
Ciò accade per problemi di modesta dimensione con tempi di calcolo molto rapidi.

Aggiungere la riga:

```
setparam("XPRS_CUTSTRATEGY",0)
```

al modello prima del comando *maximize* e poi rieseguire il modello.

Con questa modifica è stato escluso il trattamento preliminare dell'algoritmo MIP (riduzione dello spazio delle soluzioni) con la conseguenza che saranno necessari diversi nodi per giungere alla soluzione del problema.



E poi possiamo visualizzare, nell'immagine successiva, l'albero *Branch-and-Bound* premendo sul pulsante *BB tree*.

Sono usati differenti colori per rappresentare i nodi dell'albero.

Da notare che i nodi dove sono state trovate delle soluzioni con valori interi sono rappresentati con quadrati verdi.

Piccoli punti rossi identificano relazioni non applicabili (*infeasible LP relaxations*).

Muovendo il cursore sopra i nodi dell'albero, sono mostrate altre informazioni, fra le quali il numero del nodo e la scelta della variabile di *branching*.

E' possibile mettere in pausa l'algoritmo di *Branch-and-Bound* per studiare in dettaglio la costruzione dell'albero di ricerca.

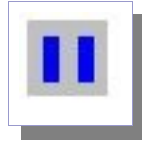
Scegliere *Build >> Options* oppure cliccare sul pulsante *run*



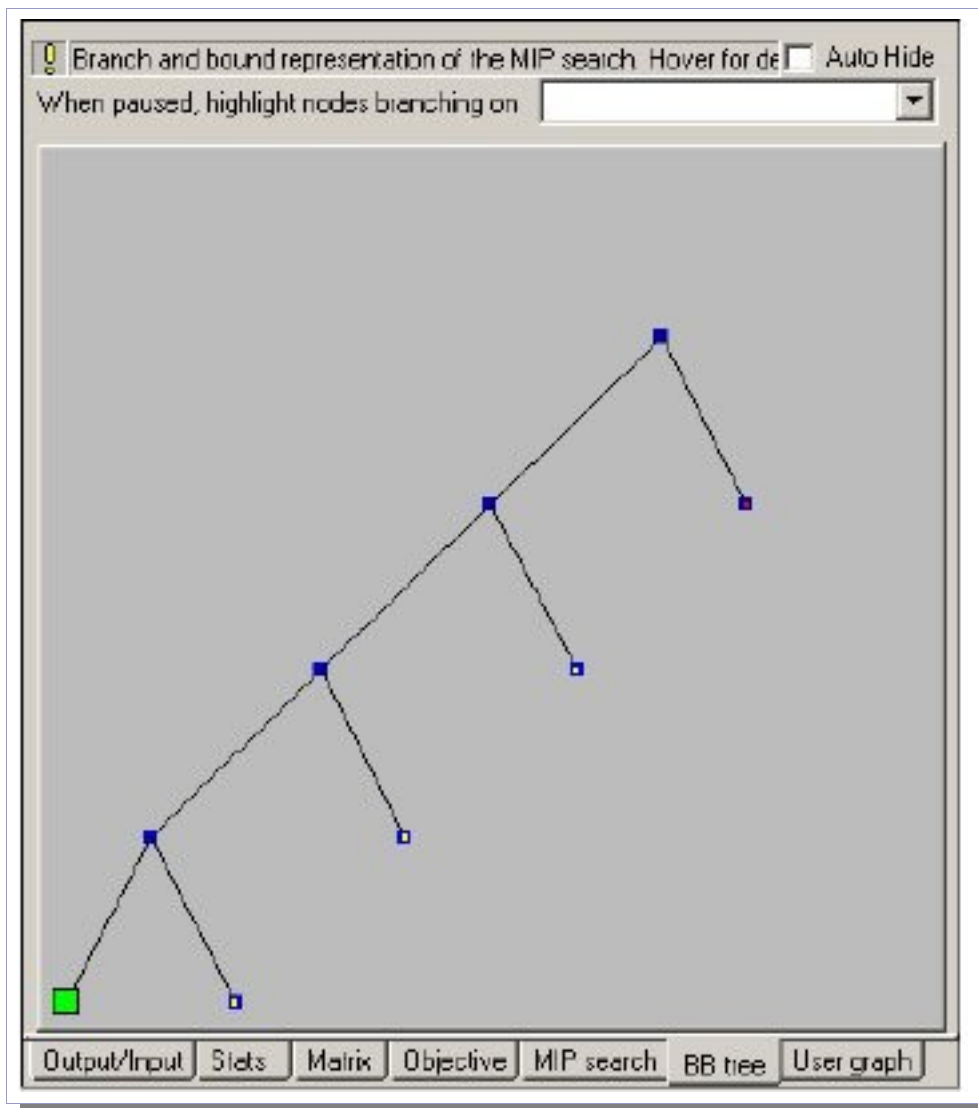
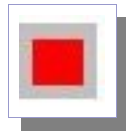
Nella finestra visualizzata, sotto la voce *Pause*, contrassegnare la voce *every global search log entry* e confermare con *Apply*.

Eseguendo nuovamente il modello, il calcolo si fermerà ad ogni nodo in

esame, lasciando la possibilità di esaminare come l'albero di ricerca è costruito; per far continuare l'esecuzione del modello, cliccare sul bottone *pause*:



per interrompere l'esecuzione usare il bottone stop:



10.5. Modello 2 – Imporre un investimento minimo per azione

Per formulare il secondo modello MIP si considera di nuovo con il problema iniziale in LP.

Il nuovo vincolo che si vuole introdurre è *l'ammontare minimo di acquisto di un'azione non può essere inferiore al 10% del budget (MINVAL = 10%)*.

Invece di imporre un vincolo ad ogni variabile $frac_s$ di assumere un valore fra 0 e MAXVAL, ora si vuole imporre il vincolo che il valore sia compreso fra MINVAL e MAXVAL oppure che il valore sia 0.

Variabili di questo tipo sono conosciute come variabili semi-continue.

In questo nuovo modello, i limiti superiori ed inferiori delle variabili $frac_s$ sono sostituiti dal seguente vincolo:

$$\forall s \in \text{SHARES} : frac_s = 0 \text{ or } MINVAL \leq frac_s \leq MAXVAL$$

10.6. Modello 2 – Sviluppo in Mosel

Il modello che segue *foliomip2.mos* sviluppa il modello 2 con algoritmo MIP, ancora una volta partendo dal modello iniziale con algoritmo LP e con i dati presenti in archivi esterni.

Le variabili semi-continue sono definite da vincoli di tipo *is_semcont*.

Un tipo simile è disponibile per le variabili intere che possono assumere il valore 0 oppure un valore intero compreso fra un limite inferiore e un limite superiore (chiamati interi semi-contigui – tipo *is_semint*).

Un terzo tipo è l'intero parziale (*partial integer*) che assume valori interi dal suo limite inferiore fino al limite superiore, restando con valori continui oltre i predetti limiti (tipo *is_partint*).

```
model "Portfolio optimization with MIP"
uses "mmxprs"                ! Use Xpress-Optimizer
parameters
  MAXRISK = 1/3                ! Max. investment into high-risk values
  MINAM = 0.5                  ! Min. investment into N.-American
values
  MAXVAL = 0.3                ! Max. investment per share
  MINVAL = 0.1                ! Min. investment per share
end-parameters
declarations
  SHARES: set of string       ! Set of shares
  RISK: set of string         ! Set of high-risk values among shares
  NA: set of string           ! Set of shares issued in N.-America
```

```

    RET: array(SHARES) of real      ! Estimated return in investment
end-declarations
initializations from "folio.dat"
    RISK RET NA
end-initializations
declarations
    frac: array(SHARES) of mpvar    ! Fraction of capital used per share
end-declarations
! Objective: total return
Return:= sum(s in SHARES) RET(s)*frac(s)
! Limit the percentage of high-risk values
sum(s in RISK) frac(s) <= MAXRISK
! Minimum amount of North-American values
sum(s in NA) frac(s) >= MINAM
! Spend all the capital
sum(s in SHARES) frac(s) = 1
! Upper and lower bounds on the investment per share
forall(s in SHARES) do
    frac(s) <= MAXVAL
    frac(s) is_semcont MINVAL
end-do
! Solve the problem
maximize(Return)
! Solution printing
writeln("Total return: ", getobjval)
forall(s in SHARES) writeln(s, ": ", getsol(frac(s))*100, "%")
end-model

```

Mandando in esecuzione questo modello e scegliendo il pulsante *Output/input* nella finestra della soluzione del problema, si ottengono i seguenti risultati:

Total return:	14.0333
treasury:	30%
hardware:	0%
theater:	20%
telecom:	0%
brewery:	10%
highways:	26.6667%
cars:	0%
bank:	0%
software:	13.3333%
electronics:	0%

Ora si scelgono altre garanzie per la composizione del portafoglio, ciascuna in grado di garantire al minimo il 10% ed al massimo il 30% dell'investimento totale.

A motivo dei vincoli aggiunti, l'ottima soluzione MIP è ancora inferiore alla soluzione originale LP.

11. Programmazione Euristica

Introducendo una semplice variabile binaria si stabilisce una soluzione euristica che comporta:

- strutturare un modello Mosel tramite la definizione di *subroutine*
- l'intervento di una soluzione euristica procedurale ed interattiva con Xpress-Optimizer tramite l'utilizzo di parametri, salvataggi e ripristini di dati e modifiche di variabili relative ai limiti inferiori e superiori.

11.1. Variabile binaria per la Programmazione Euristica

La soluzione euristica che si vuole realizzare prevede i seguenti passi:

1. Risolvere le relazioni LP per stabilire le basi di una soluzione ottimale
2. Approssimare all'euristico: Fissare tutte le variabili *buy* a 0 se il loro valore è prossimo a 0, e a 1 se il valore è relativamente ampio.
3. Risolvere il problema MIP risultante.
4. Se esiste una soluzione di tipo intero, salvare i valori come migliore soluzione.
5. Ripristinare il problema originale attribuendo a tutte le variabili i loro limiti originali e caricare le basi salvate.
6. Risolvere il problema MIP originale usando la soluzione euristica come valori di separazione (*cutoff*).

Passo 2: Poiché le variabili *frac* hanno un limite superiore di 0.3, quale valore relativamente ampio, in questo caso possiamo scegliere il valore 0.2.

In altri casi, per variabili di tipo binario sarebbe meglio scegliere valori del tipo $1 - \varepsilon$, dove ε rappresenta un valore molto piccolo come 10^{-5} .

Passo 6: Imporre un valore di separazione (*cutoff*), significa che si ricercano soluzioni migliori del valore indicato. Le riduzioni dell'approssimazione della soluzione LP fino al rientro dell'errore nei limiti specificati di tipo intero, potrebbe comportare che il nodo diventi peggiore.

11.2. Esempio completo

Per lo sviluppo dell'esempio completo (archivio *folioheur.mos*) si considera il modello sopra descritto come MIP 1.

Attraverso la definizione dell'algoritmo euristico sotto forma di *subroutine* (più precisamente una *procedure*), si introducono minimi cambiamenti allo stesso

modello:

per iniziare si dichiara la procedura usando il termine *forward* e, prima di risolvere il nostro problema invocando la funzione di massimizzazione, si esegue la nostra specifica soluzione di tipo euristico, adattando anche la funzione di stampa.

```
model "Portfolio optimization solved heuristically"
uses "mmaxprs"           ! Use Xpress-Optimizer
parameters
  MAXRISK = 1/3           ! Max. investment into high-risk values
  MAXVAL = 0.3           ! Max. investment per share
  MINAM = 0.5            ! Min. investment into N.-American
values
  MAXNUM = 4             ! Max. number of assets
end-parameters
forward procedure solve_heur ! Heuristic solution procedure
declarations
  SHARES: set of string   ! Set of shares
  RISK: set of string     ! Set of high-risk values among shares
  NA: set of string       ! Set of shares issued in N.-America
  RET: array(SHARES) of real ! Estimated return in investment
end-declarations
initializations from "folio.dat"
  RISK RET NA
end-initializations
declarations
  frac: array(SHARES) of mpvar ! Fraction of capital used per share
  buy: array(SHARES) of mpvar ! 1 if asset is in portfolio, 0 otherwise
end-declarations
! Objective: total return
Return := sum(s in SHARES) RET(s)*frac(s)
! Limit the percentage of high-risk values
sum(s in RISK) frac(s) <= MAXRISK
! Minimum amount of North-American values
sum(s in NA) frac(s) >= MINAM
! Spend all the capital
sum(s in SHARES) frac(s) = 1
! Upper bounds on the investment per share
forall(s in SHARES) frac(s) <= MAXVAL
! Limit the total number of assets
sum(s in SHARES) buy(s) <= MAXNUM
forall(s in SHARES) do
  buy(s) is_binary
  frac(s) <= buy(s)
end-do
! Solve problem heuristically
```

```

solve_heur
! Solve the problem
maximize(Return)
! Solution printing
if getprobat=XPRS_OPT then
  writeln("Exact solution: Total return: ", getobjval)
  forall(s in SHARES) writeln(s, ": ", getsol(frac(s))*100, "%")
else
  writeln("Heuristic solution is optimal.")
end-if
!-----
procedure solve_heur
declarations
  TOL: real ! Solution feasibility tolerance
  fsol: array(SHARES) of real ! Solution values for 'frac' variables
end-declarations
  setparam("XPRS_VERBOSE",true) ! Enable message printing in
mmxprs
  setparam("XPRS_CUTSTRATEGY",0) ! Disable automatic cuts
  setparam("XPRS_HEURSTRATEGY",0) ! Disable automatic MIP
heuristics
  setparam("XPRS_PRESOLVE",0) ! Switch off presolve
  TOL:=getparam("XPRS_FEASTOL") ! Get feasibility tolerance
  setparam("ZEROTOL",TOL) ! Set comparison tolerance
  maximize(XPRS_TOP,Return) ! Solve the LP problem
  savebasis(1); ! Save the current basis
  ! Fix all variables 'buy' for which 'frac' is at 0 or at a relatively
  ! large value
  forall(s in SHARES) do
    fsol(s):= getsol(frac(s)) ! Get the solution values of 'frac'
    if (fsol(s) = 0) then
      setub(buy(s), 0)
    elif (fsol(s) >= 0.2) then
      setlb(buy(s), 1)
    end-if
  end-do
  maximize(Return) ! Solve the MIP problem
  ifgsol:=false
  if getprobat=XPRS_OPT then ! If an integer feas. solution
    ! was found
    ifgsol:=true
    solval:=getobjval ! Get the value of the best solution
    writeln("Heuristic solution: Total return: ", solval)
    forall(s in SHARES) writeln(s, ": ", getsol(frac(s))*100,
"%")
  end-if

```

```

! Reset variables to their original bounds
forall(s in SHARES)
  if ((fsol(s) = 0) or (fsol(s) >= 0.2)) then
    setlb(buy(s), 0)
    setub(buy(s), 1)
  end-if
loadbasis(1)           ! Load the saved basis
if ifgsol then        ! Set cutoff to the best known solution
  setparam("XPRS_MIPABSCUTOFF", solval+TOL)
end-if
end-procedure
end-model

```

Questo modello certamente richiede delle spiegazioni aggiuntive.

11.3. Subroutine

Una *subroutine* in Mosel ha una struttura simile a quella dello stesso modello: una *procedura* inizia con il termine *procedure*, seguito dal suo nome e terminante con il termine *end-procedure*. In modo analogo una funzione inizia con il termine *function*, seguito dal suo nome e poi termina con la parola *end-function*.

Entrambi i tipi prevedono una lista di argomenti; inoltre le funzioni devono prevedere una variabile di *return*, ad esempio:

```
function myfunc(myint: integer, myarray: array(range) of string): real
```

per una funzione che restituisce un valore reale e prevede, come argomenti di input, e una matrice di *stringhe* di caratteri.

Come mostrato nell'esempio, una *subroutine* può contenere uno o più blocchi di dichiarazioni.

Gli oggetti definiti in una *subroutine* hanno validità solo locale e sono cancellati al termine dell'utilizzo della *subroutine*.

11.4. Parametri di ottimizzazione e funzioni

Parametri: La soluzione euristica inizia con la configurazione dei parametri necessari per Xpress-Optimizer.

Ricorda che con IVE tutti i parametri e le altre funzionalità possono essere visualizzati con il *module browser*: scegliere *Modules >> List available modules* oppure in alternativa cliccare su



Tutti i parametri sono accessibili tramite le subroutine Mosel *setparam* e *getparam*.

Nell'esempio dapprima si abilita la stampa tramite il modulo *mmxprs*.

Come risultato si ottiene un maggior numero di risultati nella finestra *Output/input*.

I risultati provenienti da *Optimizer* sono contraddistinti da barre colorate (blu per la parte LP, arancio per la parte MIP):

The screenshot displays the Xpress-MP software interface. The main window shows a Mosel model named 'falloheur.mos'. The code defines a procedure 'solve_heur' that maximizes 'XPRS_TOP' and sets various parameters like 'XPRS_VERBOSE', 'XPRS_CUTSTRATEGY', 'XPRS_HEURSTRATEGY', and 'XPRS_PRESOLVE'. The model is solved using the 'optimizer' module. The results are displayed in the 'Text output from Model/Optimizer' window, which shows the following output:

```
*** Search completed ***
Node 1: BestSoln 13.100000, BestBound 14.066667, Sols Active 1 0
Number of integer feasible solutions found is 1
Best integer solution found is 13.100000
Heuristic solution: Total return: 13.1
treasury: 20%
hardware: 0%
theater: 30%
telecom: 0%
brewery: 20%
highways: 30%
cars: 0%
bank: 0%
software: 0%
electronics: 0%
```

The output also includes a table with columns 'Ita', 'Obj Value', 'S', 'Minf', and 'Nlog', showing an optimal solution found at iteration 0 with an objective value of 14.066667. The status is 'Optimal solution found'.

E' opzionale disattivare la generazione automatica dei *cut* (parametro *XPRS_CUTSTRATEGY*) e dei parametri euristici MIP (parametro *XPRS_HEURSTRATEGY*), da fare nel caso fosse necessario disabilitare il meccanismo di pre-risoluzione (opzione di trasformazione della matrice che tenta di ridurre la dimensione e migliorarne le proprietà numeriche con il parametro *XPRS_PRESOLVE*), poiché si è in interreagisce con il problema in *Optimizer* durante i tentativi di soluzione e ciò è possibile solo se la matrice

non è stata modificata da Optimizer.

In aggiunta alla configurazione dei parametri, si può visualizzare il valore della tolleranza di fattibilità (*feasibility tolerance*) usata da Xpress-Optimizer. Optimizer opera con valori di tolleranza per la *integer feasibility* e per la *solution feasibility* che sono tipicamente predefiniti al valore di 10^{-6} .

Nel valutare una soluzione, per esempio nel fare dei confronti, è molto importante tener conto di queste tolleranze.

Istruzioni di ottimizzazione: Si usa una nuova versione della procedura di massimizzazione con un argomento aggiuntivo, *XPRS_TOP*, indicante che si vuole risolvere il nodo iniziale delle approssimazioni (e non l'intero problema MIP).

Salvare e caricare le basi: Per accelerare il processo di ricerca delle soluzioni, si salva in memoria la base corrente dell'algoritmo del simpleso dopo aver risolto l'iniziale approssimazione LP e prima di introdurre modifiche al problema.

Questa base è caricata ancora alla fine, quando è ricaricato il problema originale.

In questo modo l'algoritmo risolutivo MIP non deve ricalcolare il problema LP partendo dall'inizio, potendo riprendere la situazione al punto della sua interruzione da parte del cambiamento euristico dei limiti.

Cambiamento dei limiti: Quando un problema sia già stato caricato in Optimizer (cioè dopo l'esecuzione dell'istruzione di ottimizzazione oppure dopo una chiamata esplicita a *loadprob*), i cambiamenti dei limiti, tramite *setlb* e *setub*, sono trasferiti direttamente a Optimizer.

Qualsiasi altro cambiamento (aggiunta oppure cancellazione di vincoli o variabili) comporterebbe un completo ricalcolo del problema.

11.5. Tolleranze di confronto

Dopo aver visualizzato le tolleranze di Optimizer, si possono configurare le tolleranze di confronto di Mosel (*ZEROTOL*) al valore di *TOL*.

Questo significa che la condizione $fsol(s) = 0$ è vera se $fsol(s)$ è un valore compreso fra $-TOL$ e TOL , ed anche la condizione $fsol(s) \geq 0.2$ è vera per valori minimi di $fsol(s)$ pari a $0.2-TOL$.

I confronti in Mosel fanno sempre uso di tolleranze aventi valori predefiniti molto piccoli.

12. Altri esempi

Sono presentati tre semplici modelli di programmazione lineare:

- ricetta del biscotto
- programmazione della programmazione
- pianificazione di una campagna pubblicitaria di banner sul web

Per ogni esempio sono forniti una breve descrizione, il sorgente completo del modello e i risultati del calcolo.

12.1. Ricetta del biscotto

Nei lontani anni '70 del secolo scorso i computer non erano ancora *portatili*; occupavano ampi locali climatizzati, si chiamavano elaboratori centrali, avevano la dimensione di grossi armadi *quattro stagioni* e i dati di input che li alimentavano di informazioni erano scritti su schede di cartoncino con tanti buchi rettangolari.

Negli anni '70 del secolo scorso prosperava un'azienda italiana che produceva alimenti dietetici per l'infanzia: biscotti, omogenizzati, pastine, pappe, succhi di frutta ed altri prodotti simili.

A capo dei Sistemi Informativi c'era un dirigente ventottenne che amava la statistica e la matematica applicata.

Un giorno gli venne l'idea di provare se fosse stato possibile ridurre il costo del biscotto, che era il prodotto principale dell'azienda, con il vincolo di non compromettere le sue qualità nutrizionali.

A quei tempi il biscotto, come gli altri prodotti dietetici, oltre a rispettare i vincoli di composizione chimica indicati in etichetta, doveva rispettare la lista delle materie prime utilizzate indicate in ordine decrescente di presenza: lista depositata presso il Ministero della Sanità di allora.

In etichetta erano indicate le percentuali minime o massime di proteine, carboidrati, lipidi, zuccheri, minerali, calorie, residuo secco, ecc.

Nella lista delle materie prime impiegate si indicava che nel biscotto si utilizzava della farina di grano senza alcuna ulteriore specificazione se la farina di grano fosse di tipo zero o di tipo uno.

La farina di tipo uno è più ricca di proteine e di minerali rispetto alla farina di tipo zero.

La farina di tipo uno costava molto meno della farina di tipo zero.

A quei tempi le aziende dietetiche facevano a gara a chi avesse il biscotto maggiormente proteico per rendere i bambini sani e forti.

La presenza del 14 % di proteine nel biscotto era ottenuta dall'apporto del latte in polvere e da un integratore alimentare molto costoso e costituito da un super concentrato di proteine di origine animale.

Il dirigente ventottenne aveva già tradotto in linguaggio Fortran l'algoritmo del simplesso della programmazione lineare e poi aveva caricato il programma compilato nelle memorie a dischi dell'elaboratore centrale.

Dapprima il nostro dirigente ventottenne volle verificare quale fosse la ricetta ottimale utilizzando soltanto la farina di tipo zero nel rispetto dei vincoli di

composizione chimica e di sequenza nella lista degli ingredienti ottenendo il costo finale della ricetta.

Il modello, rappresentato su schede perforate, conteneva la matrice dei coefficienti, il vettore dei vincoli e la funzione obiettivo da minimizzare.

Occorrevano diversi minuti perché uscisse dalla stampante la soluzione; ora il tempo di elaborazione sarebbe limitato a meno di un secondo utilizzando un normalissimo PC.

Poi il nostro dirigente ventottenne provò ad introdurre anche la farina di tipo uno con risultati davvero importanti: il costo finale della ricetta risultava del 10 % inferiore alla ricetta che utilizzava solo la farina di tipo zero.

L'apporto proteico della farina di tipo uno aveva reso possibile un minor utilizzo del latte in polvere.

Una riduzione del 10 % nel costo delle materie prime del biscotto applicata ai volumi correnti della produzione annuale era equivalente al 150 % del budget totale dei Sistemi Informativi dell'azienda: canoni di locazione dell'elaboratore centrale e delle periferiche, stipendi dei dipendenti, materiali di consumo, consulenze, spese di manutenzione.

Ma sembrava troppo bello per essere vero, occorrevo delle verifiche.

Il Direttore Centrale Marketing e l'Amministratore Delegato vollero verificare che la nuova ricetta fosse ugualmente apprezzata dalle mamme.

Così si organizzarono dei panel di assaggio nei quali si posero a confronto i biscotti di nuova formulazione verso i biscotti di formulazione standard, naturalmente senza rendere evidente alle mamme assaggiatrici quali fossero i biscotti nuovi e quali quelli tradizionali.

L'esito, sempre confermato da panel di assaggio sempre più estesi, era sempre lo stesso: il biscotto di nuova formulazione era migliore perché di gusto più naturale e meno farmaceutico.

Questo episodio, lontano nel tempo, dimostra quanto la matematica applicata possa essere utile alle aziende a dispetto dei preconcetti e delle diffidenze di molti manager.

Matrice dei coefficienti e vettore dei vincoli

descrizione	farina1	farina0	zucch.	latte	superlatte	oliococco	olioliva	burroc	vincolo	valore
lipidi	1	1		0,5	0,01	99,9	99,9	99,8	<=	7,8
proteine	12	6		25	60				>=	14
carboidr	52,6	65	99,89	50,5	10,3				>=	60,1
zucch.			1						>=	5
latte				1	1				>=	5
olii						1	1		>=	2,5
miner.	3	0,5	0,01	7,9	1,3				<=	2,1
calorie	331	331	400	348	10,5	90	90	89,8	>=	330
calcio	17	17		1323	11			1,8	>=	350
sodio	0,3	0,3		55	20				>=	260
gr.saturi						86,8	16,16	59	<=	30,5
peso tot.	1	1	1	1	1	1	1	1	=	100
farina	1	1							>=	57

Alcuni costi unitari della funzione obiettivo da minimizzare (lire)

Ingrediente	Lire
farina di tipo uno	510
farina di tipo zero	590
zucchero	1.505
latte magro	5.340
<i>superlatte</i>	20.720
olio di cocco	1.570
olio di oliva	6.500
burro concentrato	9.400
altro: acqua, aromi, etc.	0

12.1.1. Modello in Mosel

Attenzione: Ogni vincolo termina solo con la riga bianca successiva, tenerne conto nell'importare il testo in IVE.

Ad esempio il vincolo *LIPIDI* inizia con LIPIDI e termina con 7.8.

!-----

model biscotto

uses "mmxprs"; !gain access to the Xpress-Optimizer solver

!sample declarations section

declarations

FARINA0: mpvar

FARINA1: mpvar

ZUCCHERO: mpvar

LATTEMAGRO: mpvar

LATTESUPER: mpvar

OLIOCOCCO: mpvar

OLIOOLIVA: mpvar

BURROCONC: mpvar

ALTRO: mpvar

end-declarations

PROFIT := FARINA1 * 510 + FARINA0 * 590 + ZUCCHERO * 1505 +
LATTEMAGRO * 5340 + LATTESUPER * 20720 + OLIOCOCCO * 1570 +
OLIOOLIVA * 6500 + BURROCONC * 9400

LIPIDI := FARINA1 * .01 + FARINA0 * .01 + LATTEMAGRO * .005 +
LATTESUPER * .001 + OLIOCOCCO * .999 + OLIOOLIVA * .999 +
BURROCONC * .998 <= 7.8

PROTEINE := FARINA1 * .12 + FARINA0 * .06 + LATTEMAGRO * .25 +
LATTESUPER * .60 >= 14

CARBOIDR := FARINA1 * .526 + FARINA0 * .65 + ZUCCHERO * .9989 +
LATTEMAGRO * .505 + LATTESUPER * .103 >= 60.10

ZUCCHINF := ZUCCHERO >= 5

LATTE:= LATTEMAGRO + LATTESUPER >= 5

OLII := OLIOCOCCO + OLIOOLIVA >= 2.5

MINERALI := FARINA1 * .03 + FARINA0 * .005 + ZUCCHERO * .0001 +
LATTEMAGRO * .079 + LATTESUPER * .013 <= 2.1

CALORIE := FARINA1 * 33.1 + FARINA0 * 33.1 + ZUCCHERO * 40.0 +
LATTEMAGRO * 34.8 + LATTESUPER * 10.05 + OLIOCOCCO * 90 +
OLIOOLIVA * 90 + BURROCONC * 89.8 >= 0

CALCIO := FARINA1 * 1.7 + FARINA0 * 1.7 + LATTEMAGRO * 132.3 +
LATTESUPER * 11.0 + BURROCONC * 1.8 >= 350

SODIO:= FARINA1 * .3 + FARINA0 * .3 + LATTEMAGRO * 55.0 +
LATTESUPER * 20.0 >= 260

```

GRASSAT:= OLIOCOCCO * .868 + OLIOOLIVA * .1616 + BURROCONC * .59
<= 30.5
PESOTOT := FARINA1 + FARINA0 + ZUCCHERO + LATTEMAGRO +
LATTESUPER + OLIOCOCCO + OLIOOLIVA + BURROCONC + ALTRO = 100
FARINATOT := FARINA1 + FARINA0 >= 57
minimize (PROFIT)
writeln("inizio elaborazione")
writeln(" ")
writeln("farina 0 ", getsol(FARINA0), " % ")
writeln("farina 1 ", getsol(FARINA1), " % ")
writeln("latteS  ", getsol(LATTESUPER), " % ")
writeln("latteN  ", getsol(LATTEMAGRO), " % ")
writeln("zucchero ", getsol(ZUCCHERO), " % ")
writeln("olio oli ", getsol(OLIOOLIVA), " % ")
writeln("olio coc ", getsol(OLIOCOCCO), " % ")
writeln("altro   ", getsol(ALTRO), " % ")
writeln(" ")
writeln("proteine ", getact(PROTEINE), " % ")
writeln("lipidi   ", getact(LIPIDI), " % ")
writeln("carboidr  ", getact(CARBOIDR), " % ")
writeln("calorie  ", getact(CALORIE), " % ")
writeln(" ")
writeln("Profitto ", getsol(PROFIT), " euro ")
writeln(" ")
writeln("fine elaborazione")
end-model

```

12.1.2. Risultati

Soluzione e confronti

Costo Totale ed Ingredienti	Lire / %	Prima	Dopo
costo totale kg 100	lire	381.727	346.204
farina tipo uno	%	0,00	43,78
farina tipo zero	%	57,00	13,22
zucchero	%	15,09	26,75
latte magro	%	13,33	0,84
superlatte	%	12,08	12,91
olio di cocco	%	2,50	2,50

The screenshot displays the Xpress-MP software interface. The main window shows the model code for 'biscotto.mos'. The code includes declarations for decision variables (FARINAO, FARINA1, ZUCCHERO, LATTEMAGRO, LATTESUPER, OLIOCOCCO, OLIOOLIVA, BURROCONC, ALTRO) and objective functions for PROFIT, LIPIDI, PROTEINE, CARBOIDR, ZUCCHINF, LATTE, OLII, MINERALI, CALORIE, CALCIO, SODIO, GRASSAT, and PESOTOT. The output window shows the results of the optimization, including the total cost (381.727) and the composition of the ingredients (e.g., farina 0 13.2212 %, farina 1 43.7788 %, latteS 12.9054 %, etc.).

```

model biscotto
uses "mmaxprs"; !gain access to the Xpress-Optimizer solver

!sample declarations section
declarations
FARINAO: mpvar
FARINA1: mpvar
ZUCCHERO: mpvar
LATTEMAGRO: mpvar
LATTESUPER: mpvar
OLIOCOCCO: mpvar
OLIOOLIVA: mpvar
BURROCONC: mpvar
ALTRO: mpvar
end-declarations
PROFIT := FARINA1 * 510 + FARINAO * 590 + ZUCCHERO * 1505 +
LIPIDI := FARINA1 * .01 + FARINAO * .01 + LATTEMAGRO * .005
PROTEINE := FARINA1 * .12 + FARINAO * .06 + LATTEMAGRO * .25
CARBOIDR := FARINA1 * .526 + FARINAO * .65 + ZUCCHERO * .998
ZUCCHINF := ZUCCHERO >= 5
LATTE := LATTEMAGRO + LATTESUPER >= 5
OLII := OLIOCOCCO + OLIOOLIVA >= 2.5
MINERALI := FARINA1 * .03 + FARINAO * .005 + ZUCCHERO * .001
CALORIE := FARINA1 * 33.1 + FARINAO * 33.1 + ZUCCHERO * 40.0
CALCIO := FARINA1 * 1.7 + FARINAO * 1.7 + LATTEMAGRO * 132.3
SODIO := FARINA1 * .3 + FARINAO * .3 + LATTEMAGRO * 55.0 + LA
GRASSAT := OLIOCOCCO * .868 + OLIOOLIVA * .1616 + BURROCONC
PESOTOT := FARINA1 + FARINAO + ZUCCHERO + LATTEMAGRO + LATTE
  
```

Output results:

```

inizio elaborazione
farina 0 13.2212 %
farina 1 43.7788 %
latteS 12.9054 %
latteN 0.840063 %
zucchero 26.7545 %
olio oli 0 %
olio coc 2.5 %
altro 0 %

proteine 14 %
lipidi 3.08461 %
carboidr 60.1 %
calorie 3340.81 %

Profitto 346205 euro

fine elaborazione
  
```

12.2. Programmazione della produzione

Fra i modelli di Programmazione Matematica, un ruolo importante assumono quelli che si riferiscono alla programmazione della produzione di più prodotti per più periodi temporali consecutivi, considerando diversi elementi di costo che globalmente devono essere minimizzati.

Obiettivi comuni e fra di loro contrastanti sono ad esempio:

- soddisfare le previsioni delle vendite
- mantenere un basso livello delle scorte producendo il più tardi possibile
- occupare le capacità produttive disponibili.

Produrre il più tardi possibile può essere in contrasto con delle capacità produttive non idonee a soddisfare alti picchi di vendite stagionali, mentre produrre in anticipo per soddisfare le previsioni delle vendite future stagionali comporta creare (e finanziare) le scorte da mettere a magazzino in attesa delle future vendite.

La funzione obiettivo deve quindi considerare tutti i costi coinvolti e trovare la soluzione globalmente migliore.

Un modello che interessi più periodi contigui deve essere formulato in modo di legare ogni periodo a quello precedente e a quello successivo.

In via generale la scorta di un prodotto in un generico periodo può essere espressa dalla seguente uguaglianza:

$$\text{scorta iniziale} + \text{produzione} - \text{vendite} = \text{scorta finale}$$

che può anche essere scritta:

$$\text{scorta iniziale} + \text{produzione} - \text{scorta finale} = \text{vendite}.$$

Quest'ultima forma consente di avere alla destra del segno = il valore delle previsioni di vendita (vettore dei vincoli).

Il modello deve prevedere come input:

- scorte iniziali per prodotto
- previsioni delle vendite per prodotto / periodo
- capacità produttiva per periodo

come output:

- produzioni per prodotto / periodo

- scorte finali per prodotto / periodo
- capacità non usate per periodo

come funzione obiettivo la somma dei costi di mantenimento delle scorte e i costi del non utilizzo degli impianti, cioè:

- la sommatoria dei prodotti delle scorte finali di periodo per il costo finanziario unitario
- la sommatoria delle capacità produttive non utilizzate per il costo unitario di non utilizzo.

12.2.1. Modello in Mosel

```

model prog_prod
uses "mxxprs"; !gain access to the Xpress-Optimizer solver
!sample declarations section
declarations
NMesi = 1..5
ScorteIniA: real           ! Scorte Iniziali prod. A
ScorteIniB: real           ! Scorte Iniziali prod. B
ScorteFinA: array(NMesi) of mpvar ! Scorte finali prod A
ScorteFinB: array(NMesi) of mpvar ! Scorte finali prod B
ProduzionA: array(NMesi) of mpvar ! Produzioni prod A
ProduzionB: array(NMesi) of mpvar ! Produzioni prod B
NonProduzM: array(NMesi) of mpvar ! Capacità produttiva non usata da A e
B

PrevisioA: array(NMesi) of real      ! Previsioni mensili Prod. A
PrevisioB: array(NMesi) of real      ! Previsioni mensili Prod. B

CostoScorte: real           ! Costo finanziario scorte
CostoInatt: real           ! Costo non saturazione impianto
CapacProd: real           ! Capacità produttiva
end-declarations
PrevisioA := [15, 10, 40, 20, 10]
PrevisioB := [20, 20, 20, 30, 40]
CostoScorte := 30
CostoInatt := 5
ScorteIniA := 7
ScorteIniB := 4
CapacProd := 40

ValMin := SUM(i in 1..5) ScorteFinA(i)*CostoScorte + SUM(i in 1..5)
ScorteFinB(i)*CostoScorte + SUM(i in 1..5) NonProduzM(i)*CostoInatt

IniScortaA := ScorteIniA + ProduzionA(1)- ScorteFinA(1)= PrevisioA(1)
IniScortaB := ScorteIniB + ProduzionB(1)- ScorteFinB(1)= PrevisioB(1)

```

```
forall(i in 2..5) AltreScorteA(i) :=  
  (ScorteFinA(i) = ScorteFinA(i-1) + ProduzionA(i) - PrevisioA(i))
```

```
forall(i in 2..5) AltreScorteB(i) :=  
  (ScorteFinB(i) = ScorteFinB(i-1) + ProduzionB(i) - PrevisioB(i))
```

```
forall(i in 1..5) ProdNonUsa(i) := ProduzionA(i) + ProduzionB(i) +  
NonProduzM(i) = CapacProd
```

```
minimize (ValMin)
```

```
writeln("inizio elaborazione")  
writeln(" ")  
writeln("Profitto ", getsol(ValMin), " euro ")
```

```
writeln("Produzione / Scorte:")  
writeln(" ")  
writeln("Prodotto A Mese 1 ")  
write("prod ", strfmt(getsol(ProduzionA(1)),6,1), " scorta ",  
      strfmt(getsol(ScorteFinA(1)),5,1))  
writeln(" ")  
writeln("Prodotto A Mese 2 ")  
write("prod ", strfmt(getsol(ProduzionA(2)),6,1), " scorta ",  
      strfmt(getsol(ScorteFinA(2)),5,1))  
writeln(" ")  
writeln("Prodotto A Mese 3 ")  
write("prod ", strfmt(getsol(ProduzionA(3)),6,1), " scorta ",  
      strfmt(getsol(ScorteFinA(3)),5,1))  
writeln(" ")  
writeln("Prodotto A Mese 4 ")  
write("prod ", strfmt(getsol(ProduzionA(4)),6,1), " scorta ",  
      strfmt(getsol(ScorteFinA(4)),5,1))  
writeln(" ")  
writeln("Prodotto A Mese 5 ")  
write("prod ", strfmt(getsol(ProduzionA(5)),6,1), " scorta ",  
      strfmt(getsol(ScorteFinA(5)),5,1))
```

```
writeln(" ")
```

```
writeln("Prodotto B Mese 1 ")  
write("prod ", strfmt(getsol(ProduzionB(1)),6,1), " scorta ",  
      strfmt(getsol(ScorteFinB(1)),5,1))  
writeln(" ")  
writeln("Prodotto B Mese 2 ")  
write("prod ", strfmt(getsol(ProduzionB(2)),6,1), " scorta ",
```

```

    strfmt(getsol(ScorteFinB(2)),5,1))
writeln(" ")
writeln("Prodotto B Mese 3 ")
write("prod ", strfmt(getsol(ProduzionB(3)),6,1), " scorta ",
    strfmt(getsol(ScorteFinB(3)),5,1))
writeln(" ")
writeln("Prodotto B Mese 4 ")
write("prod ", strfmt(getsol(ProduzionB(4)),6,1), " scorta ",
    strfmt(getsol(ScorteFinB(4)),5,1))
writeln(" ")
writeln("Prodotto B Mese 5 ")
write("prod ", strfmt(getsol(ProduzionB(5)),6,1), " scorta ",
    strfmt(getsol(ScorteFinB(5)),5,1))
writeln(" ")
writeln("fine elaborazione")

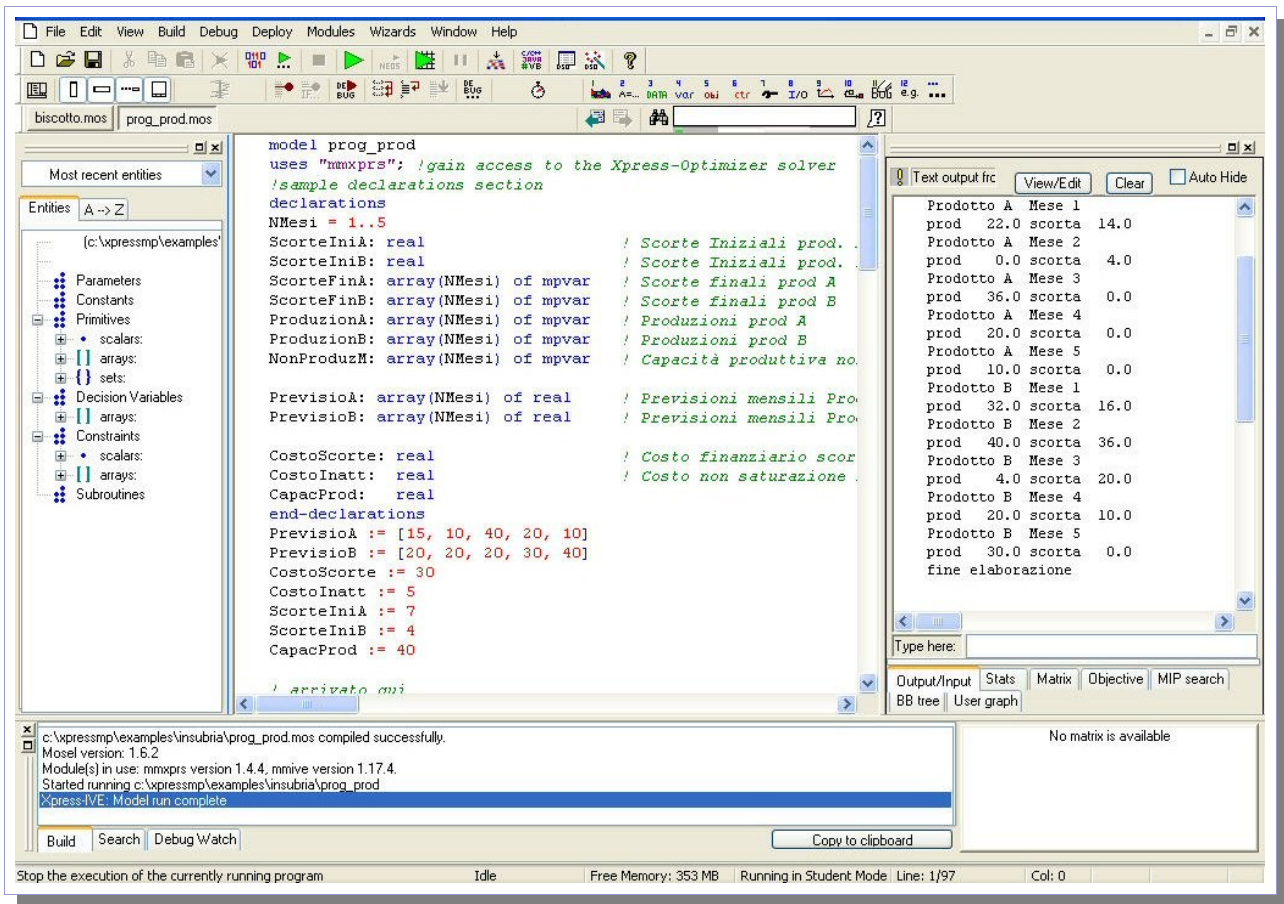
end-model

```

12.2.2. Risultati

Valore della funzione obiettivo: 2.300

Mese	Prodotto	Produzione	Scorte Finali	Capacità non utilizzata
1		A	10	30
1		B	10	
2		A	10	10
2		B	30	
3		A	40	
3		B	10	
4		A	20	
4		B	30	
5		A	10	
5		B	40	



12.3. Pianificazione di una campagna pubblicitaria

Il modello riguarda la pianificazione di investimenti pubblicitari in banner sulla rete Internet al minimo costo con vincoli relativi a:

- numero minimo di viste del banner
- numero minimo di accessi al proprio sito
- non investire su più di 3 siti.

Questo modello ha la particolarità di prevedere delle variabili di decisione di tipo binario; infatti i diversi tipi di banner possono essere scelti (valore 1) oppure non scelti (valore 0); vedere l'istruzione *forall(i in 1..8) WEB(i) is_binary*.

12.3.1. Modello in Mosel

model banner ! Pianificazione acquisto banner in Internet

*uses "mmxprs"; !gain access to the Xpress-Optimizer solver
declarations*

```

NWEB = 1..8           ! Numero dei siti WEB
MinView = 120000      ! Numero minimo delle viste
MinAxes = 800        ! Numero minimo degli accessi
WEB: array(NWEB) of mpvar
Viste: array(NWEB) of real   ! Viste previste del banner nei WEB
Accessi: array(NWEB) of real ! Accessi previsti al banner nei WEB
Costi: array(NWEB) of real   ! Costi di acquisto del banner del WEB
end-declarations

Viste := [15000,10000,40000,20000,10000,30000,22000,70000]
Accessi := [350,400,510,180,120,600,240,750]
Costi := [22,10,50,45,15,25,44,120]

CostoMin:= SUM(i in 1..8) Costi(i)*WEB(i)   ! Funzione obiettivo
MinViste:= SUM(i in 1..8) Viste(i)*WEB(i) >= MinView ! Viste minime
MinAcces:= SUM(i in 1..8) Accessi(i)*WEB(i) >= MinAxes ! Accessi minimi
MaxNMWEB:= SUM(i in 1..8) WEB(i) <= 5      ! Numero massimo dei siti
MinNMWEB:= SUM(i in 1..8) WEB(i) >= 2      ! Numero minimo dei siti

forall(i in 1..8) WEB(i) is_binary          ! Variabili binarie

minimize (CostoMin)

writeln("inizio elaborazione")
writeln(" ")
writeln("Costo Totale ", getsol(CostoMin), " euro ")
writeln(" ")
writeln("web 1 ", strfmt(getsol(WEB(1)),1,0))
writeln("web 2 ", strfmt(getsol(WEB(2)),1,0))
writeln("web 3 ", strfmt(getsol(WEB(3)),1,0))
writeln("web 4 ", strfmt(getsol(WEB(4)),1,0))
writeln("web 5 ", strfmt(getsol(WEB(5)),1,0))
writeln("web 6 ", strfmt(getsol(WEB(6)),1,0))
writeln("web 7 ", strfmt(getsol(WEB(7)),1,0))
writeln("web 8 ", strfmt(getsol(WEB(8)),1,0))
writeln(" ")
writeln("fine elaborazione")
end-model

```

12.3.2. Risultati

Costo minimo 170,00 Banner dai siti 2,5,6,8
 Viste 120.000 Accessi 1.870

The screenshot displays the Xpress-Mosel IDE interface. The main window shows the Mosel model code for 'Pianificazione acquisto banner in Internet'. The code defines variables for the number of websites (NWEB), minimum views (MinView), minimum accesses (MinAxes), and arrays for views, accesses, and costs. The objective is to minimize the total cost (CostoMin) while satisfying constraints on views and accesses. The model is solved using the Xpress-Optimizer.

The output window shows the following results:

```

inizio elaborazione
Costo Totale 170 euro

web 1 0
web 2 1
web 3 0
web 4 0
web 5 1
web 6 1
web 7 0
web 8 1

fine elaborazione
  
```

The status bar at the bottom indicates: "Submit for solving to NEOS Optimization server", "Idle", "Free Memory: 370 MB", "Running in Student Mode", "Line: 1/44", "Col: 0".

Indice

1. Premessa.....	1
2. Contenuti.....	2
3. Programmazione Matematica.....	3
4. Xpress-MP Generalità	4
4.1. Linguaggio di alto livello.....	4
4.2. Librerie per le inclusioni.....	5
4.3. Accesso diretto ai risolutori (solvers).....	5
4.4. Note sulle versioni utilizzate.....	7
5. Creare modelli.....	7
6. Esempio di Problema.....	8
7. Definire e risolvere un problema di Programmazione Lineare.....	10
7.1. Eseguire Xpress-IVE e creare un nuovo modello.....	10
7.2. Modello LP	12
7.3. Struttura generale.....	13
7.4. Soluzione.....	14
7.5. Stampa dei risultati.....	14
7.6. Formati (Formating).....	14
7.7. Correzione degli errori e messa a punto.....	15
7.8. Barra informativa dei messaggi degli errori.....	16
7.9. Soluzione, visualizzazione dei risultati	17
7.10. Statistiche.....	18
7.11. Da indici a stringhe di caratteri.....	19
8. Gestire i dati.....	20
8.1. Dati di Input provenienti da archivi.....	20
8.2. Dati di output verso gli archivi.....	22
8.3. Parametri.....	23
8.4. Esempio completo.....	25
9. Disegnare grafici	26
9.1. Descrizione estesa del problema.....	26
9.2. Ottimizzazioni ripetute.....	27
9.3. Disegnare un grafico.....	28
9.4. Esempio completo.....	30
10. Programmazione Mixed Integer (a variabili miste).....	32
10.1. Descrizione iniziale del problema.....	32
10.2. Modello 1 – Limitare il numero delle differenti azioni.....	32
10.3. Modello 1 – Sviluppo in Mosel.....	33
10.4. Modello 1 – Analizzare la soluzione.....	35
10.5. Modello 2 – Imporre un investimento minimo per azione.....	40
10.6. Modello 2 – Sviluppo in Mosel.....	40
11. Programmazione Euristica.....	42
11.1. Variabile binaria per la Programmazione Euristica.....	42
11.2. Esempio completo.....	42
11.3. Subroutine.....	45
11.4. Parametri di ottimizzazione e funzioni.....	45
11.5. Tolleranze di confronto.....	47

12. Altri esempi.....	47
12.1. Ricetta del biscotto.....	48
12.1.1. Modello in Mosel.....	50
12.1.2. Risultati.....	53
12.2. Programmazione della produzione	54
12.2.1. Modello in Mosel.....	55
12.2.2. Risultati.....	57
12.3. Pianificazione di una campagna pubblicitaria.....	58
12.3.1. Modello in Mosel.....	58
12.3.2. Risultati.....	60