SAPIENZA
UNIVERSITÀ DI ROMA

Facoltà di Ingegneria dell'Informazione,
Informatica e Statistica

Corso di Laurea in Ingegneria Gestionale

Appunti dalle lezioni del corso di

Laboratorio

di

Ricerca Operativa

Massimo Roma

Dipartimento di Ingegneria Informatica, Automatica e Gestionale
"A. Ruberti"

roma@dis.uniroma1.it
http://www.dis.uniroma1.it/∼roma

anno accademico 2015-16

Prefazione

Queste note sono redatte in forma di appunti delle lezioni, ad uso degli studenti del corso di *"Laboratorio di Ricerca Operativa"* del Corso di *Laurea in Ingegneria Gestionale* della Facoltà di *Ingegneria dell'Informazione, Informatica e Statistica* della SAPIENZA – Università di Roma.

Le lezioni tratteranno la formulazione algebrica di modelli di Programmazione Matematica (Programmazione Lineare, Programmazione Lineare Intera e Programmazione Non Lineare) seguita dall'implementazione dei modelli analitici attraverso il linguaggio di modellazione algebrica AMPL:

"*A Modeling Language For Mathematical Programming.*"

Il sito web di AMPL è `http://www.ampl.com`. Gli studenti possono scaricare dal sito una "demo version" del software e altro materiale.

Il testo di riferimento è [Fourer et al., 2003]:

> Robert Fourer, David M. Gay, Brian W. Kernighan. AMPL *A Modeling Language For Mathematical Programming*, Second Edition, Duxbury Thomson, 2003

disponibile gratuitamente sul sito AMPL al link

`http://www.ampl.com/resources/the-ampl-book/`.

Ulteriore materiale bibliografico è disponibile al link `http://www.ampl.com/REFS`.

L'obiettivo di questo corso è duplice: da un lato si vuole approfondire l'approccio modellistico tipico della Ricerca Operativa, dall'altro si vogliono fornire competenze specifiche sull'uso di un linguaggio di modellazione algebrica che permette di implementare in modo semplice un modello analitico, risolvendo poi il problema formulato attraverso l'utilizzo di opportuni solutori.

Naturali prerequisiti per questo corso sono gli argomenti di base della Ricerca Operativa già noti agli studenti dagli altri corsi curriculari del settore.

Queste note riporteranno i vari esempi di modelli trattati durante le lezioni contestualmente alla loro implementazione in AMPL. Non verrano inclusi in queste note sunti del manuale del linguaggio per il quale si fa riferimento al *Reference Manual* riportato nell'Appendice A del testo [Fourer et al., 2003], né si devono intendere queste note come una guida all'uso di AMPL.

1

Modelli di Programmazione Matematica

La Ricerca Operativa è una disciplina che tratta dello sviluppo e dell'applicazione di metodi scientifici per la soluzione di problemi di decisione che si presentano in molteplici e diversi settori della vita reale. Si tratta di scegliere quali decisioni prendere per gestire nel modo più efficiente un sistema reale utilizzando strumenti matematici; quindi lo scopo della Ricerca Operativa è quello di fornire una base scientifica per cercare di analizzare e comprendere situazioni anche con strutture molto complesse e quindi utilizzare queste informazioni per predire il comportamento di un sistema e per migliorare le prestazioni del sistema stesso.
La necessità di un approccio quantitativo ai problemi di decisione è largamente riconosciuto in moltissimi settori della vita reale ed in particolare nei problemi di decisione che si presentano nella gestione dei sistemi di produzione e nella gestione d'impresa. Il semplice "buon senso", cioè l'impiego di una persona competente del settore che sulla base dell'esperienza acquisita nel corso degli anni gestisca il sistema non è piú sufficiente a far fronte alla sempre piú crescente complessità organizzativa, e quindi anche decisionale, della gran parte dei sistemi di produzione e servizio. In questo settore, come in molti altri, soprattutto negli ultimi anni, si è acquisita la consapevolezza della necessità di tecniche quantitative basate su sofisticati strumenti matematici e avanzati mezzi informatici che permettano di prendere delle decisioni operative sulla base delle informazioni disponibili.

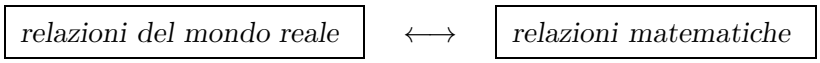
1.1 L'APPROCCIO MODELLISTICO

All'approccio semplicistico basato sull'esperienza, la Ricerca Operativa contrappone un approccio assai diverso: si tratta del cosiddetto *approccio modellistico*.

Esso organizza l'analisi di un problema reale in due fasi:

- la rappresentazione del problema attraverso un *modello matematico* che ne astragga gli aspetti essenziali e che schematizzi le interrelazioni esistenti tra i diversi aspetti del fenomeno che si sta studiando;
- lo sviluppo di *metodi matematici efficienti* (algoritmi di soluzione) per determinare una soluzione ottima del problema o una sua buona approssimazione.

Naturalmente per costruire correttamente un modello matematico che rappresenti un particolare fenomeno, si devono distinguere i parametri di controllo significativi da quelli non essenziali, identificando un criterio per la valutazione della qualità della soluzione. Una volta determinato il modello corretto, la Ricerca Operativa si occupa di fornire una procedura esplicita per determinare una soluzione di un problema; tale procedura può essere rappresentata da metodi matematici analitici o, come più spesso accade, da metodi numerici che determinano la soluzione del problema mediante specifici algoritmi di calcolo.
L'approccio modellistico necessita quindi come primo passo della costruzione di un adeguato modello matematico. Infatti, solo un modello costruito tenendo presente tutte le caratteristiche essenziali del fenomeno che si sta studiando permette di comprendere gli aspetti più importanti e di esercitare un intervento pratico efficace.
Nella fase di costruzione del modello matematico si deve fornire una descrizione formalizzata del problema di decisione facendo uso del linguaggio formale della matematica. Si dovrà cercare, quindi, una corrispondenza tra relazioni del mondo reale (relazioni tecnologiche, leggi fisiche, vincoli di mercato, etc.) e relazioni matematiche (equazioni, disequazioni, dipendenze logiche, etc.).



La costruzione di un modello richiede, quindi, scelte e valutazioni in modo da evidenziare gli aspetti più significativi del problema reale e che meglio sono suscettibili di una formalizzazione matematica. Tale procedimento di scelta spesso non è riconducibile ad un procedimento sistematico e quindi è necessario che chi costruisce il modello abbia da un lato una conoscenza approfondita del settore applicativo per evitare che le risposte ottenute dal modello abbiano scarsa rilevanza pratica; dall'altro deve avere una notevole conoscenza dei metodi matematici disponibili per la ricerca della soluzione per evitare che la formulazione matematica porti ad un problema per il quale non esistono algoritmi risolutivi utilizzabili.
È importante ribadire che un modello è definito per mezzo delle relazioni che lo costituiscono ed è quindi necessario che tali relazioni siano il più possibile

indipendenti dai dati introdotti nel modello; questo perché uno stesso modello deve poter essere usato in differenti occasioni con dati (cioè costi, disponibilità di risorse, limiti tecnologici, etc.) diversi. Lo studio di questo aspetto rientra nella fase di analisi del modello sotto il nome di analisi della stabilità del modello rispetto ai dati introdotti.
Le motivazioni che rendono molto utile la costruzione di un modello matematico sono molteplici; si riassumono di seguito le principali.

- *Possibilità di risolvere matematicamente il problema.*

  Grazie al modello è possibile analizzare matematicamente il problema ed ottenere così una soluzione che, soprattutto in riferimento a scopi di pianificazione, permette di adottare strategie che da una sola analisi strutturale del problema non apparirebbero evidenti o che a volte potrebbero essere perfino controintuitive.

- *Maggiore comprensione del problema.*

  Il modello è una rappresentazione semplificata del problema e spesso la sua costruzione consente di individuare proprietà strutturali del problema che altrimenti non sarebbero affatto evidenti.

- *Deduzione analitica di importanti proprietà.*

  Nella fase di analisi del modello è possibile dedurre per via analitica alcune importanti proprietà del problema sulla base dei risultati disponibili per la classe di problemi a cui si fa riferimento.

- *Possibilità di simulazioni.*

  Con un modello è possibile effettuare esperimenti che spesso non è possibile effettuare direttamente nella realtà; ad esempio, l'uso di un modello consente di studiare gli effetti dell'adozione di una particolare misura economica in un paese senza la necessità di sperimentarla direttamente.

Le principali critiche all'approccio modellistico e, quindi, alla costruzione di modelli per la soluzione di problemi di decisione possono essere sintetizzate nei seguenti due punti:

- Impossibilità di quantificare soddisfacentemente con opportuni valori numerici alcuni dati richiesti dal modello; questo accade, ad esempio, nel tentativo di quantificare con un costo o con un profitto alcuni valori sociali soprattutto in relazione a scopi di pianificazione.

- La qualità delle risposte che un modello produce potrebbero dipendere profondamente dall'accuratezza dei dati introdotti.

Il primo punto riguarda la possibilità di dover trattare concetti non facilmente quantificabili, ma ogni approccio scientifico può difficilmente evitare tale difficoltà; il modo migliore per superare tale problema consiste nell'incorporare tale quantificazione nel modello stesso.
La seconda critica riguarda la possibile mancanza di precisione di alcuni dei dati immessi nel modello; tale critica è meno rilevante della precedente, in quanto anche se alcuni dati introdotti sono poco accurati, è ancora possibile che la struttura del modello sia tale da garantire che la soluzione sia sufficientemente accurata.
All'estremo opposto di queste critiche si può collocare un atteggiamento di totale fiducia del modello che induca ad accettare la prima risposta prodotta dal modello senza ulteriori analisi. Tale atteggiamento, in realtà molto raro, è assai pericoloso in quanto tale risposta potrebbe rappresentare un piano operativo non accettabile nella realtà; in tal caso i motivi della non accettabilità devono essere evidenziati e incorporati in un nuovo modello modificato: si tratta, in realtà, della già citata fase di validazione del modello che quindi non può essere trascurata e che costituisce un valido mezzo per costruire modelli sempre più completi e significativi.
In conclusione, come spesso accade, l'atteggiamento corretto si colloca tra le due situazioni estreme precedentemente citate e consiste nel considerare la costruzione del modello un mezzo assai utile per affrontare un problema di decisione: rimane il fatto che la qualità delle risposte che un modello produce dipende dall'accuratezza della sua struttura e quindi non è trascurabile la fase di validazione che consente di interpretare la soluzione numerica ottenuta ed eventualmente permette di completare il modello introducendo elementi trascurati in una prima fase, in assenza dei quali la soluzione risulta non accettabile oppure di scarso rilievo dal punto di vista applicativo.

1.2 FASI DELL'APPROCCIO MODELLISTICO

Più nel dettaglio, uno studio basato sull'approccio modellistico viene di solito realizzato attraverso le seguenti fasi:

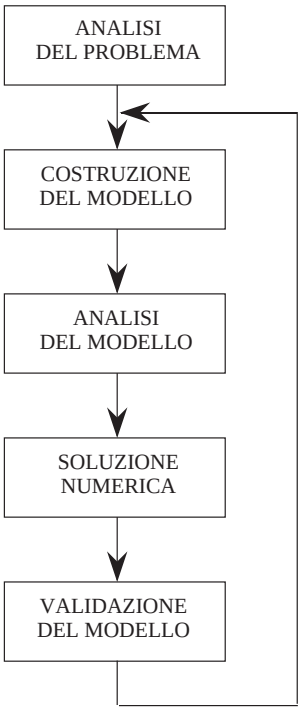
- Analisi del problema
- Costruzione del modello
- Analisi del modello
- Soluzione numerica
- Validazione del modello

La prima fase consiste nell'*analisi della struttura del problema* per individuare i legami logico-funzionali e gli obiettivi.

Nella succesiva fase di *costruzione del modello*, chiamata anche *formulazione*, si descrivono in termini matematici le caratteristiche principali del problema; questa fase di costruzione verrà descritta in dettaglio nel seguito.
Segue l'*analisi del modello* che prevede la deduzione per via analitica, in riferimento a determinate classi di problemi, di alcune importanti proprietà; le principali sono:

- *esistenza* ed *unicità* della soluzione ottima;
- *condizioni di ottimalità*, cioè una caratterizzazione analitica della soluzione ottima;
- *stabilità* delle soluzioni al variare dei dati o di eventuali parametri presenti.

La successiva fase di *soluzione* avviene mediante opportuni algoritmi di calcolo e la soluzione numerica così ottenuta deve poi essere interpretata dal punto di vista applicativo in modo da evitare che abbia scarso rilievo pratico; in questo caso le eventuali cause di inaccettabilità devono essere inglobate nel modello stesso costruendo così un nuovo modello più completo del precedente. Tale "validazione" del modello può avvenire attraverso una *verifica sperimentale* oppure con metodi di *simulazione*. La definizione di un modello si configura quindi come un processo di raffinamento iterativo, che può essere così schematizzato:



L'interesse per la modellistica matematica è notevolmente cresciuto negli anni più recenti e ai giorni nostri è sempre più viva la convinzione che ricorrendo a

modelli matematici sia possibile analizzare i molteplici aspetti del mondo reale e studiare l'influenza che l'uomo può esercitare su di essi. Ciò ha portato ad un enorme sviluppo delle applicazioni della modellistica matematica anche al di fuori delle tradizionali applicazioni alle scienze fisiche. Si è così avuta di fatto una vasta utilizzazione di modelli matematici in settori lontani dagli ambiti più tradizionali come, ad esempio, le scienze sociali, la biologia, le scienze ambientali, la psicologia. Come esempi concreti, si pensi agli studi sulla dinamica della popolazione, sulla diffusione delle epidemie, sul risanamento ambientale. Questa notevole diffusione della modellistica matematica è anche dovuta al fatto che l'evoluzione di un modello matematico può essere rapidamente studiata grazie all'uso di moderni calcolatori elettronici.
È evidente come in molti casi le situazioni rappresentate da un modello sono molto complesse e alcune volte influenzate da fenomeni di natura aleatoria; per questa ragione, sono state definite diverse classi di modelli matematici: *modelli stocastici* che considerano grandezze che possono essere influenzate da fenomeni aleatori e *modelli deterministici* che considerano grandezze esatte; inoltre a seconda che le interazioni tra le grandezze sono immediate o distribuite nel tempo, si parla di *modelli statici* e di *modelli dinamici*.
Nel seguito verranno analizzati i modelli deterministici che sono di fatto quelli più comunemente usati; in particolare si farà riferimento ai *modelli di programmazione matematica* nei quali è esplicitamente definito un obiettivo da minimizzare o massimizzare ed in cui le variabili sono vincolate ad appartenere ad un insieme prefissato. Si osservi che in questo contesto il termine *"programmazione"*, derivando dall'inglese *"programming"*, è inteso nel senso di *"pianificazione"* e non di scrittura di programmi per il calcolatore.

1.3 MODELLI DI PROGRAMMAZIONE MATEMATICA

All'interno della Ricerca Operativa, un ruolo di fondamentale importanza è svolto dalla *Programmazione Matematica* che è la disciplina che ha per oggetto lo studio dei problemi in cui si vuole minimizzare o massimizzare una funzione reale definita su $\mathbb{R}^n$ (lo spazio delle $n-$uple reali) le cui variabili sono vincolate ad appartenere ad una insieme prefissato. Si tratta quindi di *problemi di Ottimizzazione* cioè problemi nei quali si desidera minimizzare o massimizzare una quantità che è espressa attraverso una funzione.

1.3.1 Problemi di Ottimizzazione

In termini generali, data una funzione $f: \mathbb{R}^n \longrightarrow \mathbb{R}$, ed $S \subseteq \mathbb{R}^n$, un *problema di Ottimizzazione* può essere formulato nella forma

$$\begin{cases} \min f(x) \\ x \in S. \end{cases} \qquad (PO)$$

Quindi un problema di Ottimizzazione consiste nel determinare, se esiste, un punto di minimo della funzione f tra i punti dell'insieme S.
Si parlerà indifferentemente di problemi di massimo o di minimo in quanto vale $\min_{x \in S} f(x) = -\max_{x \in S} (-f(x))$.
La funzione f viene chiamata *funzione obiettivo* e l'insieme S *insieme ammissibile* cioè l'insieme delle possibili soluzioni del problema. Un punto $x \in S$ si chiama *soluzione ammissibile.*

L'insieme ammissibile S è un sottoinsieme di $\mathbb{R}^n$ e quindi $x = (x_1, x_2, \ldots, x_n)^T$ è una variabile vettoriale n-dimensionale e la funzione obiettivo f è una funzione di n variabili reali $f(x_1, x_2, \ldots, x_n)$.

Si dice che un problema di ottimizzazione (PO)

- è *inammissibile* se $S = \emptyset$, cioè se non esistono soluzioni ammissibili.

- è *illimitato (inferiormente)* se comunque scelto un valore $M > 0$ esiste un punto $x \in S$ tale che $f(x) < -M$

- *ammette soluzione ottima (finita)* se esiste un $x^\star \in S$ tale che risulti $f(x^\star) \leq f(x)$ per ogni $x \in S$. Il punto $x^\star$ è detto *soluzione ottima* o *minimo globale* e il corrispondente valore $f(x^\star)$ di dice *valore ottimo.*

Queste definizioni solo immediatamente estendibili al caso in cui un problema di Ottimizzazione è scritto in forma di massimizzazione.

All'interno dei problemi di Ottimizzazione, in base alla *struttura dell'insieme ammissibile* S, ovvero in base alla natura delle variabli, si possono distinguere le seguenti importanti classi di problemi:

- **Problemi di Ottimizzazione Continua.**

  Le variabili possono assumere tutti i valori reali ($x \in \mathbb{R}^n$);
  si parla di problemi di ottimizzazione continua

  - *vincolata* se $S \subset \mathbb{R}^n$
  - *non vincolata* se $S = \mathbb{R}^n$.

- **Problemi di Ottimizzazione Discreta.**

  Le variabili sono vincolate ad essere numeri interi ($x \in \mathbf{Z}^n$); si possono distinguere all'interno di questa classe di problemi altre due classi:

  - *programmazione a numeri interi* se $S \subseteq \mathbf{Z}^n$
  - *ottimizzazione booleana* se $S \subseteq \{0,1\}^n$.

- **Problemi misti.**

  Solo alcune delle variabili sono vincolate ad essere intere.

1.3.2 Problemi di Programmazione Matematica

Di solito l'insieme ammissibile S viene descritto da una numero finito di disuguaglianze del tipo $g(x) \geq b$, dove g è una funzione definita su $\mathbb{R}^n$ a valori reali e $b \in \mathbb{R}$. Cioè, formalmente, date m funzioni $g_i : \mathbb{R}^n \to \mathbb{R}$, $i = 1, \ldots, m$ ed m scalari $b_i \in \mathbb{R}$, $i = 1, \ldots, m$ si esprime S nella forma

$$S = \{x \in \mathbb{R}^n \;\mid\; g_1(x) \geq b_1, \;\; g_2(x) \geq b_2, \;\; \ldots \;\; , \;\; g_m(x) \geq b_m\}\,.$$

Ogni disuguaglianza $g_i(x) \geq b_i$ prende nome di *vincolo* e l'insieme ammissible è quindi formato da tutti quei punti $x \in \mathbb{R}^n$ che sono soluzione del sistema di disuguaglianze

$$\begin{cases} g_1(x) & \geq & b_1 \\ g_2(x) & \geq & b_2 \\ g_3(x) & \geq & b_3 \\ & \vdots & \\ g_m(x) & \geq & b_m. \end{cases}$$

In questa formulazione dell'insieme S si sono utilizzati vincoli di disuguaglianza nella forma di maggiore o uguale, ma è chiaro che questa notazione include i casi in cui i vincoli sono espressi con vincoli di disuguaglianza nella forma di minore o uguale e vincoli di uguaglianza; infatti si può sempre trasformare un vincolo di

minore o uguale del tipo $g(x) \leq b$ in un vincolo di maggiore o uguale semplicemente riscrivendolo nella forma $-g(x) \geq -b$. Inoltre un vincolo di uguaglianza $g(x) = b$ può essere riscritto nella forma equivalente delle due diseguaglianze $g(x) \geq b$ e $-g(x) \geq -b$.

Quindi, senza perdere di generalità, si può riscrivere il problema di ottimizzazione (PO) nella forma

$$\begin{cases} \min f(x) \\ g_i(x) \geq b_i, \quad i = 1, \ldots, m. \end{cases} \tag{1.3.1}$$

Un problema di questo tipo viene chiamato *problema di Programmazione Matematica*. I punti dell'insieme ammissibile di questo tipo di problemi sono quelli per i quali tutti i vincoli sono soddisfatti cioè tutti quei punti x tali che tutte le disuguaglianze $g_i(x) \geq b_i$, $i = 1, \ldots, m$ sono verificate.

I problemi di Programmazione Matematica si possono classificare in base alla *struttura delle funzioni che li definiscono*; in particolare si ha la seguente classificazione:

- **Problemi di Programmazione Lineare (PL)**
  La funzione obiettivo $f(x)$ e tutte le funzioni che definiscono i vincoli $g_i(x)$, $i = 1, \ldots m$ sono *lineari*, cioè esprimibili nella forma
  $$c_1 x_1 + c_2 x_2 + \cdots + c_n x_n.$$
- **Problemi di Programmazione Non Lineare (PNL)**
  Almeno una delle funzioni che definiscono un problema di Programmazione Matematica *non è lineare*.

1.3.3 Modelli di Programmazione Matematica

I modelli standard più comunemente usati nella Ricerca Operativa sono i *modelli di Programmazione Matematica*, cioè modelli che possono essere rappresentati per mezzo di un problema di Programmazione Matematica. I settori applicativi all'interno dei quali sorgono problemi di questo tipo sono moltissimi: come esempi si possono citare problemi inerenti la pianificazione industriale, problemi di progettazione ottima, problemi di gestione di reti, problemi di economia e moltissimi altri.
Tuttavia, ogni lista di classi di modelli non può essere esaustiva: possono sempre presentarsi situazioni pratiche che non possono essere modellate in modo standard oppure che possono essere modellate in più di un modo standard.

La costruzione formale di un modello di Programmazione Matematica si effettua a partire da una descrizione logica e qualitativa di un problema di decisione e richiede di:

1. Associare opportune *variabili di decisione* alle grandezze reali. Tali variabili costituiscono le incognite del problema.
2. Esprimere formalmente l'*obiettivo* che si intende minimizzare o massimizzare.
3. Esprimere quantitativamente i *legami* esistenti tra le variabili e le *limitazioni* derivanti da considerazioni di carattere fisico, economico, etc. Tali legami e limitazioni definiscono i *vincoli*. L'insieme dei valori delle variabili per cui i vincoli sono soddisfatti costituisce l'*insieme ammissibile*.

Tabella 1.3.1 Schema per la costruzione di un modello di Programmazione Matematica

A seconda della classe di problemi di Ottimizzazione entro la quale la formulazione del modello si colloca si parlerà di *modelli continui, modelli discreti, modelli misti*.

2

Introduzione ad AMPL

Come ampiamente discusso nel capitolo precedente, l'*approccio modellistico* rappresenta un potente "strumento" per la soluzione di un problema di decisione. In particolare, rappresentare un problema attraverso un modello di Programmazione Matematica permette di utilizzare i numerosi solutori disponibili che sono in grado di risolvere efficientemente problemi anche di dimensioni molto elevate. Tuttavia pur disponendo di efficienti algoritmi di soluzione, sarà necessario trasferire al solutore il problema rappresentato dal modello di Programmazione Matematica. Ovvero è necessario implementare il modello in una qualche forma riconoscibile dai vari solutori. Gli strumenti che svolgono questa funzione vengono chiamati *generatori algebrici di modelli* e, ormai da diversi anni, ne sono stati messi a punto alcuni che hanno avuto una vasta diffusione. Uno di questi è AMPL (acronimo di *A Mathematical Programming Language*) che è, ad oggi, uno dei più largamente utilizzati per la formulazione di problemi di Programmazione Lineare, Programmazione Lineare Intera, Programmazione Non Lineare e mista. Esso ha la funzione di interfaccia tra il modello algebrico ed il solutore numerico e rappresenta un potente linguaggio di modellazione algebrica, ovvero un linguaggio che contiene diverse primitive per esprimere le notazioni normalmente utilizzate per formulare un problema di Programmazione Matematica: sommatorie, funzioni matematiche elementari, operazioni tra insiemi, etc. Utilizza un'architettura molto avanzata, fornendo un'ottima flessibilità d'uso; infatti, la naturale notazione algebrica utilizzata permette di implementare modelli anche complessi e di dimensioni molto elevate in una forma assai concisa e facilmente comprensibile. È di fatto un linguaggio ad alto livello e non fa uso di particolari strutture dati perché utilizza solamente file di testo sia per l'implementazione del modello sia

per la memorizzazione dei dati. In aggiunta, per affermare l'esigenza già ricordata di rendere un modello indipendente dai dati, permette di tenere distinta la struttura del modello dai dati numerici che sono memorizzati separatamente. Inoltre, permette di richiamare al suo interno diversi tra i più efficienti solutori per problemi di Programmazione Matematica di cui attualmente si dispone. È disponibile per i sistemi operativi Windows, Unix/Linux e Mac OS X e tutte le informazioni sono reperibili sul sito `http://www.ampl.com`. Sono disponibili la AMPL IDE version e la AMPL Command Line version. Nel seguito, si farà esplicito riferimento a sistemi Windows e alla versione "command line". Gli studenti possono scaricare la Demo version. Tale versione è limitata nel numero delle variabili e dei vincoli. In particolare, per problemi lineari fino a 500 variabili e 500 vincoli; per problemi non lineari fino a 300 variabili e 300 vincoli. È anche disponibile una versione di prova completa senza limitazioni valida 30 giorni (AMPL Free 30-day trials).

Il testo di riferimento è:

> Robert Fourer, David M. Gay, Brian W. Kernighan. AMPL *A Modeling Language For Mathematical Programming*, Second Edition, Duxbury Thomson, 2003

disponibile sul sito di AMPL al link `http://ampl.com/resources/the-ampl-book`.

Allo stesso link si possono trovare i file di tutti gli esempi riportati nel libro. Sul sito è inoltre disponibile ulteriore materiale riguardante AMPL e i solutori supportati.

2.1 INSTALLAZIONE E AVVIO DI AMPL

Per quanto riguarda la versione Windows a riga di comando, andrà scaricato il file `ampl-demo-mswin.zip` che è un archivio contenente tutti i file necessari. Una volta estratto l'archivio in una directory, saranno create due directory (MODELS e TABLES) che contengono numerosi utili esempi e dei file eseguibili. In particolare il file `ampl.exe` è l'eseguibile di AMPL che chiamato dal prompt di comandi avvierà il programma. Gli altri file eseguibili `cplex.exe`, `gurobi.exe`, `minos.exe` e `lpsolve.exe` sono i solutori disponibili, rispettivamente CPLEX 12.6, Gurobi 6.0, MINOS 5.51 e LP_SOLVE 4.0. Si tratta di una versione a linea di comando e quindi deve essere utilizzata dalla finestra del *prompt di comandi* (finestra nera). Quindi aprire tale finestra, portarsi nella directory dove è stato estratto il file `ampl-demo-mswin.zip` (oppure aggiungere questa directory nel `PATH` di Windows) ed avviare il programma digitando il comando `ampl`. Apparirà così il prompt dei comandi di AMPL:

```
ampl:
```

A questo punto si è nell'ambiente AMPL, ovvero possono essere digitati i comandi di AMPL. Alternativamente, per avviare AMPL si può cliccare due volte su sw.exe (*scrolling-window utility*), digitando poi `ampl` sul prompt che appare.

2.2 UN PRIMO ESEMPIO

In linea generale (anche se assai poco pratico) un modello potrebbe essere inserito dal prompt di comandi di AMPL, ovvero digitando, ad esempio, sulla riga di comando i seguenti comandi in successione:

```
ampl: var x1;
ampl: var x2;
ampl: maximize funzione_obiettivo: x1 + x2;
ampl: subject to vincolo1: x1 + x2 <= 1;
ampl: subject to vincolo2: x1 - x2 <= 2;
ampl: subject to x1_non_neg: x1 >= 0;
ampl: subject to x2_non_neg: x2 >= 0;
```

Vedremo più avanti come sia possibile scrivere tali comandi in un file, ma per il momento soffermiamoci sull'analisi dei comandi AMPL utilizzati nell'esempio e sulle parole chiave del linguaggio. Inanzitutto osserviamo che

- ogni comando termina con un "`;`"
- sono presenti due variabili (x_1 e x_2) e queste sono dichiarate con i comandi `var x1;` e `var x2;`
- la funzione obiettivo è introdotta dalla parola chiave `maximize` in quanto trattasi di un problema di massimizzazione (per problemi di minimizzazione la parola chiave sarà `minimize`). Tale parola chiave è seguita da una etichetta proposta dall'utente (nel caso dell'esempio è `funzione_obiettivo`) seguita da "`:`" che introducono all'espressione della funzione obiettivo
- i vincoli sono elencati di seguito: ciascun vincolo è introdotto dalla parola chiave `subject to` (che può anche esser abbreviata in `s.t.`), seguita dall'etichetta che si vuole dare al vincolo e da "`:`" dopo il quale è riportata l'espressione del vincolo.

Questo semplice esempio ci ha permesso di introdurre la struttura tipica di un modello nel linguaggio AMPL, struttura che ricalca fedelmente quella standard di un modello di Programmazione Matematica come riportato nella Tabella 1.3.1.

Ovvero, dopo aver introdotto le *variabili di decisione*, deve essere formalizzata la *funzione obiettivo* e i *vincoli*.
Ora che abbiamo il modello implementato in AMPL, naturalmente viene spontaneo domandarsi come può essere determinata la soluzione del problema di ottimizzazione rappresentato dal modello (che nel caso dell'esempio è un problema di Programmazione Lineare). Come già detto da AMPL sono disponibili alcuni solutori. Nella directory dove risiede AMPL oltre al file `ampl.exe` sono presenti altri file eseguibili che rappresentano i solutori; essi sono CPLEX,GUROBI, MINOS e LPSOLVE. Ciascuno di essi risolve alcune classe di problemi di Programmazione Matematica. Nel seguito riporteremo nel dettaglio le tipologie di problemi risolte da ciascun solutore. Ora vogliamo introdurre il comando per selezionare il solutore da utilizzare, ovvero

`option solver` *nome_solutore*;

Quindi digitando:

```
ampl: option solver cplex;
```

stiamo comunicando ad AMPL di voler utilizzare il solutore CPLEX. Quest'ultimo è un solutore per problemi di Programmazione Lineare (e non solo) e può quindi essere utilizzato per risolvere il problema riportato nell'esempio.
A questo punto è sufficiente dare il comando per risolvere il problema che è

`solve;`

Digitando questo comando al promtp dei comandi, ovvero

```
ampl: solve;
```

si ottiene il seguente messaggio:

```
CPLEX 12.6.1.0: optimal solution; objective 1;
0 dual simplex iterations (0 in phase I)
```

con il quale AMPL ci comunica che il solutore ha determinato una soluzione ottima con valore della funzione obiettivo pari a 1. Per vedere il valore delle variabili all'ottimo è necessario un altro comando, ovvero

`display` *nome_variabile*;

Quindi digitando, ad esempio,

```
ampl: display x1;
```

otteniamo il valore di x_1^*. Analogamente per l'altra variable x_2^*. Abbiamo così ottenuto il punto di massimo che stavamo cercando.

Scrivere il modello utilizzando la linea di comando, ovviamente, è piuttosto scomodo, soprattutto perché in questo modo, una volta usciti da AMPL il modello

è completamente perso. Comviene, pertanto, scrivere il modello in un file testo che deve avere estensione `.mod`. Quindi utilizzando un qualsiasi editor di testo possiamo riscrivere il modello nel seguente modo:

esempio.mod

```
var x1;
var x2;

maximize funzione-obiettivo: x1 + x2;

subject to vincolo1: x1 + x2 <= 1;
subject to vincolo2: x1 - x2 <= 2;
subject to x1_non_neg: x1 >= 0;
subject to x2_non_neg: x2 >= 0;
```

A questo punto, dal prompt di AMPL è sufficiente scrivere il seguente comando per leggere il file del modello:

`model` *<PATH>\nome_file.mod*

Quindi, assumendo che la directory dove risiede il file del modello `esempio.mod` sia *C:\MODELLI*, digitando,

```
ampl: model C:\MODELLI\esempio.mod;
```

carichiamo il modello. A questo punto, si procede come descritto in precedenza per risolvere il problema e stampare i risultati ottenuti.

Si può inoltre creare un file contenente i comandi da dare (in modo da non doverli digitare ogni volta). Tale file deve avere estensione `.run`. Un esempio di questo file relativamente all'esempio fino ad ora esaminato è il seguente:

esempio.run

```
reset;
model C:\MODELLI\esempio.mod;
option solver cplex;
solve;
display x1;
display x2;
display funzione_obiettivo;
```

Si noti che nel file è stato aggiunto il comando `reset;` che ha la funzione di eliminare da AMPL dati relativi ad un modello precedentemente risolto. È conveniente introdurlo all'inizio di ogni file `.run`.
Per lanciare il file `.run` *nell'ambiente* AMPL si utilizza il comando

`include` <*PATH*> \ *nome_file.run*

Alternativamente, *fuori dell'ambiente* AMPL è sufficiente avere tale file nella directory che contiene AMPL (oppure in un'altra directory se si è introdotto nel *PATH* il percorso dove risiede AMPL), aprire un terminale del prompt dei comandi in tale directory e dare il comando `ampl esempio.run`.

Infine, per uscire dall'ambiente AMPL è sufficiente il comando `quit;`.

2.3 I SOLUTORI

Abbiamo già menzionato il fatto che alcuni solutori sono disponibili nell'installazione Demo versiondi AMPL. Essi sono:

- CPLEX
- GUROBI
- LPSOLVE
- MINOS.

CPLEX.
Risolve problemi di Programmazione Lineare e problemi di Programmazione Quadratica convessi utilizzando il metodo del simplesso e metodi a punti interni. Inoltre risolve anche problemi di Programmazione Lineare e problemi di Programmazione Quadratica convessi con variabili intere utilizzando procedure di tipo Branch-and-Bound. La versione distribuita con la Demo version di AMPL è limitata a 500 variabili e 500 vincoli.

GUROBI.
Risolve problemi di Programmazione Lineare con il metodo del simplesso e con metodi a punti interni. Risolve inoltre problemi di Programmazione Lineare Misti Interi utilizzanzo procedure di tipo Branch-and-Bound. Risolve inoltre anche problemi di Programmazione Programmazione Quadratica e problemi di Programmazione Quadratica Misti Interi. Utilizzando la "student version" di AMPL, GUROBI limita la dimensione dei problemi a 500 variabili e 500 vincoli.

LPSOLVE.
Risolve problemi di Programmazione Lineare e Programmazione Lineare Intera di dimensioni moderate.

MINOS.
Risolve problemi di Programmazione Lineare attraverso il metodo del simplesso e problemi di Programmazione Non Lineare utilizzando metodi di tipo gradiente ridotto.

Esistono altri solutori dei quali è stato previsto l'uso con AMPL, ma non tutti sono disponibili gratuitamente.

2.4 ALCUNI ESEMPI DI MODELLI DI PROGRAMMAZIONE LINEARE

Esaminiamo ora alcuni semplici modelli di Programmazione Lineare costruendo prima la formulazione algebrica e poi realizzandone un'implementazione in AMPL.

Esempio 2.4.1 *Un'industria chimica fabbrica 4 tipi di fertilizzanti,* **Tipo 1**, **Tipo 2**, **Tipo 3**, **Tipo 4**, *la cui lavorazione è affidata a due reparti dell'industria: il reparto produzione e il reparto confezionamento. Per ottenere fertilizzante pronto per la vendita è necessaria naturalmente la lavorazione in entrambi i reparti. La tabella che segue riporta, per ciascun tipo di fertilizzante i tempi (in ore) necessari di lavorazione in ciascuno dei reparti per avere una tonnellata di fertilizzante pronto per la vendita.*

	Tipo 1	**Tipo 2**	**Tipo 3**	**Tipo 4**
Reparto produzione	2	1.5	0.5	2.5
Reparto confezionamento	0.5	0.25	0.25	1

Dopo aver dedotto il costo del materiale grezzo, ciascuna tonnellata di fertilizzante dà i seguenti profitti (prezzi espressi in Euro per tonnellata)

	Tipo 1	**Tipo 2**	**Tipo 3**	**Tipo 4**
profitti netti	250	230	110	350

Determinare le quantità che si devono produrre settimanalmente di ciascun tipo di fertilizzante in modo da massimizzare il profitto complessivo, sapendo che ogni settimana, il reparto produzione e il reparto confezionamento hanno una capacità lavorativa massima rispettivamente di 100 e 50 ore.

Si tratta di un problema di pianificazione della produzione industriale in cui le incognite, che saranno le variabili del problema, sono le quantità di fertilizzante di ciascun tipo che si devono produrre. Costruiamo un modello di Programmazione Matematica rappresentante il problema in analisi supponendo di voler pianificare la produzione settimanale.

– *Variabili di decisione.* È naturale introdurre le variabili reali x_1, x_2, x_3, x_4 rappresentanti rispettivamente le quantità di prodotto del **Tipo 1**, **Tipo 2**, **Tipo 3**, **Tipo 4** da fabbricare in una settimana.

– *Funzione Obiettivo.* Ciascuna tonnellata di fertilizzante contribuisce al profitto totale secondo la tabella data. Quindi il profitto totale sarà

$$250x_1 + 230x_2 + 110x_3 + 350x_4. \tag{2.4.1}$$

L'obiettivo dell'industria sarà quello di scegliere le variabili x_1, x_2, x_3, x_4 in modo che l'espressione (2.4.1) del profitto sia massimizzata. La (2.4.1) rappresenta la funzione obiettivo.

– *Vincoli.* Ovviamente la capacità produttiva della fabbrica limita i valori che possono assumere le variabili x_j, $j = 1, \ldots, 4$; infatti si ha una capacità massima lavorativa in ore settimanali di ciascun reparto. In particolare per il reparto produzione si hanno a disposizione al piú 100 ore settimanali e poiché ogni tonnellata di fertilizzante di **Tipo 1** utilizza il reparto produzione per 2 ore, ogni tonnellata di fertilizzante di **Tipo 2** utilizza il reparto produzione per 1.5 ore e così via per gli altri tipi di fertilizzanti si dovrà avere

$$2x_1 + 1.5x_2 + 0.5x_3 + 2.5x_4 \leq 100. \tag{2.4.2}$$

Ragionando in modo analogo per il reparto confezionamento si ottiene

$$0.5x_1 + 0.25x_2 + 0.25x_3 + x_4 \leq 50. \tag{2.4.3}$$

Le espressioni (2.4.2), (2.4.3) costituiscono i vincoli del modello. Si devono inoltre esplicitare vincoli dovuti al fatto che le variabili x_j, $j = 1, \ldots 4$, rappresentando quantità di prodotto non possono essere negative e quindi vanno aggiunti i vincoli di non negatività

$$x_1 \geq 0, \quad x_2 \geq 0, \quad x_3 \geq 0, \quad x_4 \geq 0.$$

La formulazione finale quindi può essere scritta in questa forma

$$\begin{cases} \max\left(250x_1 + 230x_2 + 110x_3 + 350x_4\right) \\ 2x_1 + 1.5x_2 + 0.5x_3 + 2.5x_4 \leq 100 \\ 0.5x_1 + 0.25x_2 + 0.25x_3 + x_4 \leq 50 \\ x_1 \geq 0, x_2 \geq 0, x_3 \geq 0, x_4 \geq 0. \end{cases}$$

Si riporta di seguito il file `.mod` che implementa questo modello:

fertilizzanti.mod

```
var x1;
var x2;
var x3;
var x4;

maximize profitto: 250*x1+230*x2+110*x3+350*x4;

subject to vincolo1: 2*x1+1.5*x2+0.5*x3+2.5*x4 <= 100;
s.t. vincolo2: 0.5*x1+0.25*x2+0.25*x3+x4<= 50;
s.t. x1_nonneg: x1 >=0;
s.t. x2_nonneg: x2 >=0;
s.t. x3_nonneg: x3 >=0;
s.t. x4_nonneg: x4 >=0;
```

Esempio 2.4.2 *Un'industria manifatturiera possiede due impianti di produzione e fabbrica due tipi di prodotti* $\mathbf{P_1}$ *e* $\mathbf{P_2}$ *utilizzando due macchine utensili: una per la levigatura e una per la pulitura. Per avere un prodotto finito è necessaria l'utilizzazione di entrambe le macchine. Il primo impianto ha una disponibilità massima settimanale di 80 ore della macchina per la levigatura e di 60 ore della macchina per la pulitura. Le disponibilità orarie delle due macchine nel secondo impianto sono rispettivamente di 60 e 70 ore settimanali. La tabella che segue riporta, per ciascun prodotti, il numero di ore di lavorazione necessarie su ciascuna macchina per ottenere un prodotto finito (poiché le macchine possedute dal secondo impianto sono piú vecchie, i tempi di utilizzo sono maggiori)*

	Impianto 1		Impianto 2	
	$\mathbf{P_1}$	$\mathbf{P_2}$	$\mathbf{P_1}$	$\mathbf{P_2}$
levigatura	*4*	*2*	*5*	*3*
pulitura	*2*	*5*	*5*	*6*

Inoltre ciascuna unità di prodotto utilizza 4 Kg di materiale grezzo. Il profitto netto (in Euro) ottenuto dalla vendita di una unità di Prodotti $\mathbf{P_1}$ *e* $\mathbf{P_2}$ *è rispettivamente di 10 e 15.*

(a) *Costruire un modello lineare che permetta di massimizzare il profitto complessivo ottenuto dalla vendita dei prodotti in ciascun impianto sapendo che settimanalmente l'industria decide di assegnare 75 Kg di materiale grezzo al primo impianto e 45 Kg di materiale grezzo al secondo impianto.*

(b) *Costruire un modello lineare che permetta di massimizzare il profitto complessivo ottenuto dalla vendita dei prodotti supponendo che l'industria non allochi a priori 75 Kg di materiale grezzo nel primo impianto e di 45 Kg di materiale grezzo nel secondo impianto, ma lasci al modello la decisione di come ripartire tra i due impianti 120 Kg complessivi disponibili di questo materiale grezzo.*

Si riportano sinteticamente le formulazioni nei due casi.

Formulazione del caso (a)
Questo caso, nella pratica, corrisponde a costruire due modelli indipendenti: uno riferito al primo impianto, uno riferito al secondo impianto. Una "risorsa" (il materiale grezzo) è già allocata a priori.

Impianto 1: La formulazione relativa al primo impianto è:

$$\begin{cases} \max(10x_1 + 15x_2) \\ 4x_1 + 4x_2 \leq 75 \\ 4x_1 + 2x_2 \leq 80 \\ 2x_1 + 5x_2 \leq 60 \\ x_1 \geq 0, x_2 \geq 0 \end{cases}$$

A questo punto, prima di passare all'impianto 2, vediamo come fare per scrivere in AMPL il precedente problema di PL. Per fare questo è sufficiente creare il seguente file di modello (un semplice file di testo ma con estensione `.mod`) ad esempio `impianto1.mod`.

`impianto1.mod`

```
var x1;
var x2;

maximize profitto: 10*x1 + 15*x2;

subject to   m_grezzo: 4*x1 + 4*x2 <= 75;
subject to levigatura: 4*x1 + 2*x2 <= 80;
s.t.         pulitura: 2*x1 + 5*x2 <= 60;
s.t.         x1_non_neg: x1 >= 0;
s.t.         x2_non_neg: x2 >= 0;
```

Risolvendo il problema, si ottiene che all'ottimo l'impianto 1 ha un profitto di 225 Euro, ottenuto fabbricando $11,25$ unità di $\mathbf{P_1}$ e $7,5$ di $\mathbf{P_2}$. In particolare si è utilizzato tutto il materiale grezzo e sono state sfruttate tutte le ore di pulitura a disposizione. Le macchine per la levigatura, invece, sono state sottoutilizzate essendo avanzate 20 ore di tempo macchina.

Impianto 2: La formulazione relativa al secondo impianto è:

$$\begin{cases} \max(10x_3 + 15x_4) \\ 4x_3 + 4x_4 \leq 45 \\ 5x_3 + 3x_4 \leq 60 \\ 5x_3 + 6x_4 \leq 70 \\ x_3 \geq 0, x_4 \geq 0 \end{cases}$$

Il modello in AMPL per l'impianto 2 è:

`impianto2.mod`

```
var x3;
var x4;

maximize profitto: 10*x3 + 15*x4;
```

```
subject to   m_grezzo: 4*x3 + 4*x4 <= 45;
subject to levigatura: 5*x3 + 3*x4 <= 60;
s.t.         pulitura: 5*x3 + 6*x4 <= 70;
s.t.         x3_non_neg: x3 >= 0;
s.t.         x4_non_neg: x4 >= 0;
```

Risolvendo il problema si ha che all'ottimo l'impianto 2 ha un profitto di $168,75$ Euro ottenuto fabbricando $11,25$ unità di $\mathbf{P_2}$ e nessuna unità di $\mathbf{P_1}$. In particolare è stato utilizzato tutto il materiale grezzo ma, sia le macchine per la levigatura che quelle per la pulitura, sono state sottoutilizzate essendo avanzate rispettivamente $26,25$ e $7,5$ ore di tempo macchina.

Formulazione del caso (b)
Questo caso corrisponde a costruire un unico modello comprendente entrambi gli impianti. L'allocazione della "risorsa" data dal materiale grezzo è lasciata al modello stesso.

La formulazione relativa a questo caso è:

$$\begin{cases} \max & (10x_1 & + & 15x_2 & + & 10x_3 & + & 15x_4) & & \\ & 4x_1 & + & 4x_2 & + & 4x_3 & + & 4x_4 & \leq & 120 \\ & 4x_1 & + & 2x_2 & & & & & \leq & 80 \\ & 2x_1 & + & 5x_2 & & & & & \leq & 60 \\ & & & & & 5x_3 & + & 3x_4 & \leq & 60 \\ & & & & & 5x_3 & + & 6x_4 & \leq & 70 \\ & x_1 \geq 0, & & x_2 \geq 0, & & x_3 \geq 0, & & x_4 \geq 0 & & \end{cases}$$

in AMPL si avrà:

impianto1e2.mod

```
var x1;
var x2;
var x3;
var x4;

maximize profitto: 10*(x1+x3) + 15*(x2+x4);

s.t. m_grezzoTOT:   4*(x1+x2+x3+x4) <= 120;
s.t. levigatura1:   4*x1 + 2*x2 <= 80;
s.t. pulitura1:     2*x1 + 5*x2 <= 60;
```

```
s.t. levigatura2:   5*x3 + 3*x4 <= 60;
s.t. pulitura2:     5*x3 + 6*x4 <= 70;

s.t. x1_non_neg:        x1 >= 0;
s.t. x2_non_neg:        x2 >= 0;
s.t. x3_non_neg:        x3 >= 0;
s.t. x4_non_neg:        x4 >= 0;
```

3

Gli insiemi e i parametri in AMPL

AMPL permette di scrivere il modello in forma parametrica. Questo, nella sostanza, significa scrivere il modello nel file `.mod` senza specificare i dati che vengono invece scritti separatamente in un file `.dat`. Praticamente nel file del modello si effettuano le *dichiarazioni*, mentre nel file dei dati si effettuano le *assegnazioni*. Fatto questo, dal prompt di AMPL oltre al comando

`model` <*PATH*> $\setminus$*nome_file.mod*

utilizzato per caricare il modello, sarà necessario un comando per caricare il file dei dati. Tale comando è:

`data` <*PATH*> $\setminus$*nome_file.dat*

Quindi, assumendo che la directory dove risiedono i file del modello (`esempio.mod`) e dei dati (`esempio.dat`) sia *C:*$\setminus$*MODELLI*, digitando anche

```
ampl: data C:\MODELLI\esempio.dat;
```

carichiamo anche i dati del modello. A questo punto, si procede come descritto in precedenza per risolvere il problema e stampare i risultati ottenuti.

Per introdurre l'uso di insiemi e parametri, consideriamo di nuovo i modelli presentati nell'Esempio 2.4.2 del capitolo precedente. Infatti, AMPL consente di scrivere il problema in modo diverso, utilizzando un concetto molto utilizzato in AMPL: gli *insiemi*. Grazie ad esso, è possibile tenere separati il modello dai dati. Il modello relativo all'impianto 1 può essere infatti riscritto come segue:

impianto.mod

```
set Prodotti;
set Risorse;

param q_max{Risorse} >=0;
param richiesta{Risorse,Prodotti} >=0;
param prezzo{Prodotti} >=0;

var x{Prodotti} >=0;

maximize profitto: sum{i in Prodotti} prezzo[i]*x[i];

s.t. vincolo_risorsa {j in Risorse}:
sum{i in Prodotti} richiesta[j,i]*x[i] <= q_max[j];
```

Ora analiziamo le istruzione del file `impianto.mod`. Anzitutto notiamo le istruzioni

```
set Prodotti;
set Risorse;
```

con le quali si dichiarano `Prodotti` e `Risorse` come insiemi di un numero imprecisato di elementi (prodotti e risorse). Subito dopo abbiamo tre istruzioni che ci servono per definire altrettanti parametri del modello. La prima di queste

```
param q_max{Risorse}>=0;
```

definisce un vettore di parametri con tante componenti quanti sono gli elementi dell'insieme `Risorse` e definisce le quantità massime disponibili di ciascuna risorsa.
Il $>= 0$ specifica che i valori immessi devono essere non negativi, AMPL eseguirà automaticamente il controllo sui dati.
L'istruzione successiva ovvero

```
param richiesta{Risorse,Prodotti}>=0;
```

definisce invece una matrice di parametri con tante righe quanti sono le risorse e tante colonne quante sono i prodotti, specificando per ogni prodotto la quantità di risorsa necessaria alla sua realizzazione.

L' istruzione

```
param prezzo{Prodotti} >=0;
```

definisce un vettore di parametri con tante componenti quanti sono gli elementi dell'insieme `Prodotti` specificando il prezzo di ogni prodotto.

La funzione obiettivo `profitto` è definita dalla istruzione

```
maximize profitto: sum{i in Prodotti} prezzo[i]*x[i];
```

I vincoli sono definiti atttraverso l'istruzione che segue:

```
s.t. vincolo_risorsa {j in Risorse}:
     sum{i in Prodotti} richiesta[j,i]*x[i] <= q_max[j];
```

Con essa si definiscono tanti vincoli quanti sono gli elementi dell'insieme `Risorse`.

A questo punto, completato il file della definizione del modello (`.mod`) è necessario definire un file con estensione `.dat` dove vengono specificati i valori per gli insiemi e i parametri del modello. Per quanto riguarda l'impianto 1 si avrà:

impianto1.dat

```
set Prodotti := P1 P2;
set Risorse  := m_grezzo levigatura pulitura;

param         q_max :=
m_grezzo        75
levigatura      80
pulitura        60;

param  richiesta: P1   P2 :=
m_grezzo          4     4
levigatura        4     2
pulitura          2     5;

param  prezzo :=
P1       10
P2       15;
```

Facciamo notare che in questo modo, avendo cioè a disposizione il file `impianto.mod` contenente il modello separato dai dati del problema (contenuti invece nel file `impianto1.dat`), possiamo risolvere differenti problemi di massimizzazione del profitto semplicemente cambiando i dati contenuti nel file con estensione `.dat`.

Sfruttando questo fatto è banale risolvere il problema relativo all'impianto 2: si utilizza lo stesso file `.mod` e un diverso file `.dat`, che è il seguente:

impianto2.dat

```
set Prodotti := P1 P2;
set Risorse  := m_grezzo levigatura pulitura;

param         q_max :=
m_grezzo        45
levigatura      60
pulitura        75;


param  richiesta: P1   P2 :=
m_grezzo          4     4
levigatura        5     3
pulitura          5     6;


param  prezzo :=
P1       10
P2       15;
```

Nello stesso modo, cioè specificando il solo file dei dati, è possibile risolvere il problema completo:

impianto.dat

```
set Prodotti :=
       P1_impianto1
       P2_impianto1
       P1_impianto2
       P2_impianto2;

set Risorse :=
       m_grezzo
       levigatura1
       pulitura1
       levigatura2
       pulitura2;
```

```
param         q_max :=
      m_grezzo         120
      levigatura1      80
      pulitura1        60
      levigatura2      60
      pulitura2        75;


param  richiesta: P1_impianto1   P2_impianto1  P1_impianto2  P2_impianto2:=
 m_grezzo           4                4             4               4
 levigatura1        4                2             0               0
 pulitura1          2                5             0               0
 levigatura2        0                0             5               3
 pulitura2          0                0             5               6;


param          prezzo :=
        P1_impianto1       10
        P2_impianto1       15
        P1_impianto2       10
        P2_impianto2       15;
```

Si può ovviamente modificare il file `.run` scritto in precedenza aggiungendo l'istruzione necessaria per caricare il file dei dati.

È importante ribadire che nel file `.mod` vengono effettuate le *dichiarazioni* di insiemi e parametri, mentre nel file `.dat` vengono effettuate le *assegnazioni*.

Esaminiamo nel dettaglio nei paragrafi che seguono gli elementi base della sintassi di AMPL riguardante *insiemi*, *parametri*, *variabili*, *funzione obiettivo* e *vincoli*.

3.1 GLI INSIEMI

Gli insiemi definiscono gli elementi di base con i quali si possono indicizzare variabili, parametri e vincoli del modello.

La dichiarazione di un *insieme generico* (nel file `.mod`) si effettua come segue:

```
set NomeInsieme;
```

L'assegnazione di un *insieme generico* (nel file `.dat`) si effettua come segue:

```
set NomeInsieme := e1 e2 e3 e4 e5 ;
```

Questa istruzione specifica che l'insieme `NomeInsieme` è composto dagli elementi $\{e_1, e_2, e_3, e_4, e_5\}$. Gli insiemi così definiti corrispondono agli insiemi generici. Esiste la possibilità di definire altri tipi di insiemi:

- *insiemi ordinati*

  ```
  set NomeInsieme ordered;
  ```

- *insiemi numerici*

  ```
  set NomeInsieme := 1 .. N;
  set NomeInsieme := 1 .. N by p;
  ```

- *insiemi ordinati e ciclici*

  ```
  set NomeInsieme circular;
  ```

Per quanto riguarda gli insiemi numerici la notazione `NomeInsieme := 1 .. N` definisce tutti i numeri interi tra 1 e N, mentre `NomeInsieme := 1 .. N by p` definisice tutti i numeri interi tra 1 e N distanti fra di loro di p numeri. Si osservi, quindi, che per quanto riguarda gli insiemi numerici la dichiarazione di fatto è un'assegnazione e quindi l'assegnazione di un insieme numerico *non* deve essere ripetuta nel file `.dat`.

Riportiamo ora gli operatori e le funzioni più comuni tra due **insiemi generici**:

Operatore/Funzione	**Significato**
`A union B`	insieme di elementi che stanno in A o B
`A inter B`	insieme di elementi che stanno sia in A che in B
`A within B`	A sottoinsieme di B
`A diff B`	insieme di elementi che stanno in A ma non in B
`A symdiff B`	insieme di elementi che stanno in A o in B ma non in entrambi
`card(A)`	numero di elementi che stanno in A

Di seguito gli operatori più comuni utilizzati in un **insieme ordinato** A:

Funzione	Significato
`first(A)`	primo elemento di A
`last(A)`	ultimo elemento di A
`next(a,A)`	elemento di A dopo a
`prev(a,A)`	elemento di A prima di a
`next(a,A,k)`	k-esimo elemento di A dopo a
`prev(a,A,k)`	k-esimo elemento di A prima di a
`ord(a,A)`	posizione di a in A
`ord0(a,A)`	come `ord(a,A)` ma restituisce 0 se a non è in A
`member(k,A)`	elemento di A in k-esima posizione

3.1.1 Gli insiemi multidimensionali

In AMPL è possibile definire insiemi a più dimensioni. Tale dimensione deve essere specificata nella dichiarazione di insieme (file `.mod`). La dicharazione

```
set INSIEME dimension p;
```

si usa per indicare che l'insieme `INSIEME` è costituito da $p-$uple *ordinate*, ovvero i suoi elementi sono della forma $(a_1, a_2, \ldots, a_p)$. Un esempio potrebbe essere:

```
set TERNE dimension 3;
```

che indica che l'insieme `TERNE` è formato da tutte terne ordinate di valori che verranno poi specificate nel file `.dat`, ad esempio, nel seguente modo:

```
set TERNE := (a,b,c) (d,e,f) (g,h,i) (l,m,n);
```

oppure

```
set TERNE :=
a b c
d e f
g h i
l m n;
```

Un modo alternativo di ottenere insiemi multidimensionali di dimensione p è l'utilizzazione del prodotto cartesiano di p insiemi. Il prodotto cartesiano in AMPL si indica con `cross`. Quindi se, ad esempio, A, B e C sono stati già dichiarati come insiemi, l'istruzione

```
set TERNE := A cross B cross C;
```

indica l'insieme delle terne ordinate (a, b, c) con $a \in A$, $b \in B$ e $c \in C$.

Partendo da un insieme multidimensionale è possibile ottenere gli insiemi componenti l'insieme multidimensionale effettuando un procedimento inverso al precedente. Con riferimento all'esempio precedente, dall'insieme `TERNE` si possono ottenere i tre insiemi di partenza nel seguente modo con l'uso di `setof`:

```
set A := setof{(i,j,k) in TERNE} i;
set B := setof{(i,j,k) in TERNE} j;
set C := setof{(i,j,k) in TERNE} k;
```

È inoltre possibile definire insiemi in modo implicito, ovvero senza specificarne il nome, ma solo indicando la loro espressione. Ad esempio, dati A e B due insiemi,

```
{A,B}
{a in A, b in B}
```

sono due espressioni equivalenti per definire il prodotto cartesiano $A \times B$, ovvero l'insieme di tutte le coppie ordinate (a, b) con $a \in A$ e $b \in B$. Un altro esempio è dato dall'espressione

```
{a in A : costo[a]>=5}
```

che sta ad indicare tutti gli elementi a dell'insieme A tali che il `costo` di a sia maggiore o uguale a 5, dove, come vedremo più avanti, `costo` è un parametro indicizzato sull'insieme A.

3.2 I PARAMETRI

I parametri rappresentano i dati di un problema. Una volta che vengono assegnati (nel file `.dat`), non vengono in nessun caso modificati dal solutore. Un parametro deve essere *dichiarato* nel file `.mod` e *assegnato* nel file `.dat`. L'istruzione

```
param T;
```

utilizzata nel file `.mod` effettua la dichiarazione del parametro T. È possibile anche dichiarare vettori e matrici di parametri. Quindi se `PROD` e `ZONA` sono due insiemi le istruzioni

```
set PROD;
set ZONA;
param T;
param costi{PROD};
param prezzo{PROD,ZONA};
param domanda{PROD,ZONA,1..T};
```

oltre che dichiarare gli insiemi, definiscono:

- il parametro `T`;

- il parametro `costi` indicizzato dall'insieme `PROD`, ovvero `costi` è un vettore che ha tante componenti quanti sono gli elementi di `PROD`;
- il parametro a due dimensioni `prezzo`, indicizzato dagli insiemi `PROD` e `ZONA`;
- il parametro a tre dimensioni `domanda`, indicizzato dagli `PROD`, `ZONA` e dall'insieme dei numeri interi che vanno da 1 a T.

L'assegnazione dei parametri avviene nel file `.dat`. Un esempio di assegnazione dei parametri prima introdotti è il seguente:

```
set PROD := p1 p2;
set ZONA := z1 z2;
param T := 2;

param costi :=
   p1 5
   p2 4;

param prezzo: z1    z2  :=
   p1          2     7
   p2          5     9;

param domanda:=
  [*,*,1] :   z1    z2:=
     p1       10    15
     p2       13    22

  [*,*,2] :   z1    z2:=
     p1       32    25
     p2       18    15;
```

Si osservi, innanzitutto, che la scelta di scrivere i dati incolonnati ha il solo scopo di rendere il file più leggibile: nessuna formattazione particolare è richiesta da AMPL. Inoltre è opportuno soffermarci sull'uso dei ":"; infatti i ":" sono obbligatori se si assegnano valori a due o più vettori di parametri monodimensionali indicizzati sullo stesso insieme, come nel caso di `prezzo` e `domanda`.

Il parametro `domanda` può essere alternativamente assegnato nel seguente modo:

```
param : domanda :=
   p1  z1  1  10
   p1  z1  2  32
   p1  z2  1  15
   p1  z2  2  25
```

```
p2  z1  1  13
p2  z1  2  18
p2  z2  1  22
p2  z2  2  15;
```

Nella dichiarazione dei parametri (file `.mod`) si possono anche includere controlli o restrizioni sui parametri stessi. Ad esempio, le istruzioni

```
param T > 0;
param N integer, <= T;
```

controllano che il parametro T sia positivo e che il parametro N sia un numero intero minore o uguale a T. Esiste anche l'istruzione `check` per effettuare controlli contenenti espressioni logiche. Un esempio potrebbe essere il seguente:

```
set PROD;
param offertatot >0;
param offerta{PROD} >0;
check: sum{p in PROD} offerta[p]=offertatot;
```

Esiste, infine, la possibilità di dichiarare parametri "calcolati" come nel seguente esempio:

```
set PROD;
param offerta{PROD};
param offertatot:=sum{p in PROD} offerta[p];
```

3.3 LE VARIABILI

Le variabili rappresentano le incognite del problema e il loro valore è calcolato dal solutore. Una volta che la soluzione è stata determinata, il valore delle variabili all'ottimo rappresenta la soluzione del problema. Si osservi, quindi, la differenza con i parametri: questi ultimi vengono assegnati dall'utente e rimangono costanti, mentre il valore delle variabili cambia durante le iterazioni del solutore. Alle variabili l'utente può eventualmente (ma è del tutto opzionale) assegnare dei valori iniziali che sono poi modificati dal solutore.

La dichiarazione delle variabili è obbligatoria. Per default, una variabile è considerata *reale*, oppure può essere specificata *intera* o *binaria* (a valori in $\{0, 1\}$). L'esempio che segue riporta la dichiarazione di variabili con la specifica del tipo di variabili:

```
var x;
```

```
var n integer;
var d binary;
```

Analogamente a quanto avviene per i parametri, anche le variabili possono essere indicizzate da insiemi come riportato nel seguente esempio:

```
set PROD;
set OPERAI;
param dom{PROD};
var num{PROD} integer;
var assegnamento{PROD,OPERAI} binary;
```

Anche nel caso delle variabili si possono introdurre dei controlli contestualmente alla loro dichiarazione come riportato nell'esempio seguente:

```
set PROD;
param dom{PROD};
var x >=0;
var quantita{p in PROD} >=0, <= dom[p];
```

È possibile *fissare* una variabile ad un valore mediante l'istruzione `fix` che può essere utilizzata nel file `.dat` o nel file `.run`. Quindi data la variabile x, l'istruzione

```
fix x :=4;
```

assegna il valore 4 ad x. In questo caso il solutore *non cambierà il valore di tale variabile* che verrà considerata fissata al valore assegnato. Esiste poi il comando opposto `unfix` che sblocca una variabile precedentemente fissata. Nel caso dell'esempio si avrebbe

```
unfix x;
```

Infine, c'è la possibilità di inizializzare una variabile ad un determinato valore con il comando `let` da utilizzare nel file `.dat` o nel file `.run`. Se scriviamo

```
let x:= 10;
```

stiamo inizializzando la variabile x al valore 10. Questo vuole dire che l'algoritmo risolutivo assegnerà 10 come valore iniziale della variabile x, valore che sarà cambiato dal solutore nel corso delle sue iterazioni. Si osservi, quindi, la differenza fondamentale tra il "fixing" di una variabile che non permette di modificare il valore assegnato a quella variabile e l'assegnazione di un valore iniziale ad una variabile.

3.4 LA FUNZIONE OBIETTIVO E I VINCOLI

La funzione obiettivo è sempre presente in un modello di Programmazione Matematica e rappresenta ciò che vogliamo massimizzare o minimizzare. Essa deve

essere specificata nel file del modello (`.mod`). La parola chiave del linguaggo AMPL per introdurre la funzione obiettivo è `minimize` o `maximize` a seconda che ci trovi di fronte ad un problema di minimizzazione o massimizzazione. La sintassi è

```
maximize nome_funzione_obiettivo :  espressione_aritmetica ;
```

oppure

```
minimize nome_funzione_obiettivo :  espressione_aritmetica ;
```

Quindi un esempio potrebbe essere:

```
maximize profitto_totale : sum{i in PROD} prezzo[i]*quantita[i];
```

I vincoli sono parte integrante di un modello di Programmazione Matematica e sono specificati anch'essi nel file del modello (`.mod`). La parola chiave è `subject to` che può essere abbreviata in `s.t.`. La sintassi è

```
subject to nome_vincolo :  espressione_aritmetica e/o logica;
```

Se x è una variabile reale, la più semplice espressione di un vincolo potrebbe essere

```
subject to vincolo : x >=0;
```

In realtà, questo tipo di vincoli semplici viene spesso inserito come restrizione nella dichiarazione della variabile x. Anche i vincoli possono essere indicizzati e quindi invece di scrivere tanti vincoli, in una sola espressione si possono esplicitare più vincoli. Un esempio di ciò è il seguente:

```
set PROD;
set REPARTI;

param orelavoro{PROD,REPARTI};
param maxore{REPARTI};
param prezzo{PROD};

var x{PROD};

maximize ricavo : sum{in PROD} prezzo[i]*x[i];

subject to vincoli{j in REPARTI} : sum{i in PROD}
                        orelavoro[i,j]*x[i] <= maxore[j];
```

In questo esempio, con un unico vincolo si sono scritti tanti vincoli quanti sono gli elementi dell'insieme `REPARTI` al posto dei vincoli

```
subject to vincolo_1 : sum{i in PROD} orelavoro[i,"R1"]*x[i]
                            <= maxore["R1"];
subject to vincolo_2 : sum{i in PROD} orelavoro[i,"R2"]*x[i]
                            <= maxore["R2"];
   .
   .
   .
   .
subject to vincolo_m : sum{i in PROD} orelavoro[i,"Rm"]*x[i]
                            <= maxore["Rm"];
```

avendo supposto che l'insieme `REPARTI` sia composto dagli elementi $\{R_1, R_2, \ldots, R_m\}$. Quest'ultima scrittura non darebbe luogo a messaggi di errore in AMPL, ma oltre che essere molto lunga presenta l'ovvio inconveniente di dover conoscere già nel file del modello (`.mod`) quali sono gli elementi dell'insieme `REPARTI`. Questo contravviene al fatto che un *modello deve essere indipendente dai dati*; infatti, non utilizzando la scrittura indicizzata dei vincoli, un cambio degli elementi dell'insieme `REPARTI` (che ricordiamo è assegnato nel file dei dati (`.dat`)) non permetterebbe più al modello implementato di essere corretto.

3.5 LE ESPRESSIONI

Nella costruzione della funzione obiettivo e dei vincoli, così come nell'imporre condizioni sui parametri e sulle variabili si utilizzano espressioni aritmetiche, funzioni e operatori di diverso tipo. Abbiamo già riportato in precedenza i principali operatori e funzioni sugli insiemi. Le principali funzioni e operatori aritmetici e logici sono riportati nelle tabelle che seguono (per una elenco completo si fa riferimento al testo [Fourer et al., 2003]):

Funzione	Significato
`abs(x)`	valore assoluto di x
`sin(x)`	$\sin(x)$
`cos(x)`	$\cos(x)$
`tan(x)`	$\tan(x)$
`asin(x)`	$\arcsin(x)$
`acos(x)`	$\arccos(x)$
`atan(x)`	$\arctan(x)$
`exp(x)`	$\exp(x)$
`sqrt(x)`	radice quadrata di x, $\sqrt{x}$
`log(x)`	logaritmo naturale di x, $\ln(x)$
`log10(x)`	logaritmo in base 10 di x, $\log(x)$
`ceil(x)`	parte intera superiore di x, $\lceil x \rceil$
`floor(x)`	parte intera inferiore di x, $\lfloor x \rfloor$

Operatori aritmetici	Significato
`^`	potenza
`+`	somma
`-`	sottrazione
`*`	prodotto
`/`	divisione
`div`	divisione intera
`mod`	modulo
`sum`	sommatoria
`prod`	produttoria
`min`	minimo
`max`	massimo
$>$, $>=$	maggiore, maggiore o uguale
$<$, $<=$	minore, minore o uguale
`=`	uguale
`<>`, $!=$	diverso

Operatori logici	Significato
`not`	negazione logica
`or`	"or" logico
`and`	"and" logico
`exists`	quantificatore esistenziale logico
`forall`	quantificatore universale logico
`if then else`	espressione condizionale

3.6 DUE ESEMPI DI MODELLI DI PROGRAMMAZIONE LINEARE

3.6.1 Un problema di pianificazione dei trasporti

Esaminiamo ora un esempio di un importante classe di problemi della Ricerca Operativa: *il problema dei trasporti.*

Esempio 3.6.1 *Si devono pianificare i trasporti di un tipo di merce da cinque città, Asti, Bergamo, Como, Domodossola, Lecce (città origini) ad altre quattro città, Mantova, Napoli, Olbia, Palermo (città destinazioni). La tabella che segue riporta il costo unitario del traporto della merce da ciascuna città origine a ciascuna città destinazione, insieme alla domanda di merce da soddisfare esattamente di ogni città destinazione e alla quantità massima di merce disponibile presso ciascuna città origine*

	Mantova	Napoli	Olbia	Palermo	*disponibilità max*
Asti	1	3.5	4	4.5	100
Bergamo	0.1	3	4.5	4.8	110
Como	0.3	2.8	4.7	4.9	130
Domodossola	1	2.2	3.9	4	85
Lecce	1.7	2.2	5	5.3	120
domanda	30	18	45	56	

Come si può osservare, la somma dei quantitativi di merce disponibili nelle città origini è maggiore alla somma delle domande delle città destinazione e questo perché nelle città orgini sono disponibili dei depositi dove immagazzinare la merce. Si devono pianificare i trasporti dalle città origini alle città destinazioni in modo da minizzare il costo totale dei trasporti.

Formulazione

Introdotte le varibili di decisione x_{ij}, $i = 1, 2, 3, 4, 5$, $j = 1, 2, 3, 4$, associate alla quantità di merce da trasportare dalla città origine i-esima alla città destinazione j-esima, indicando con c_{ij} il costo unitario del trasporto dalla città origine i-esima alla città destinazione j-esima e indicando rispettivamente con a_i, $i = 1, 2, 3, 4, 5$ e b_j, $j = 1, 2, 3, 4$ la disponibilità massima di merce nelle origini e la domanda di merce nelle destinazioni, un modello lineare che rappresenta il problema in analisi è il seguente:

$$\begin{cases} \min \left(\sum_{i=1}^{5} \sum_{j=1}^{4} c_{ij} x_{ij} \right) \\ \sum_{j=1}^{4} x_{ij} \leq a_i \qquad i = 1, \ldots, 5 \\ \sum_{i=1}^{5} x_{ij} = b_j \qquad j = 1, \ldots, 4 \\ x_{ij} \geq 0 \end{cases}$$

Segue il file `trasporto1.mod` che realizza un'implementazione in AMPL del modello ora costruito.

trasporto1.mod

```
####    SEZIONE PER LA DICHIARAZIONE DEGLI INSIEMI    ####

set ORIGINI;         # introduce l'insieme delle origini
set DESTINAZIONI;    # introduce l'insieme delle destinazioni


####    SEZIONE PER LA DICHIARAZIONE DEI PARAMETRI    ####

param  costo_origdest {ORIGINI,DESTINAZIONI} >= 0
                            # vettore di parametri a 2
                            # indici (matrice). Rappresenta
                            # i costi unitari di trasporto
                            # (non negativi) dalle origini
                            # alle destinazioni.

param  max_uscita {ORIGINI} >= 0;
                            # vettore di parametri ad un
                            # indice. Rappresenta la quantita'
                            # (non negativa) massima di merce
                            # che puo' essere trasportata da
                            # ciascuna origine.

param  domanda {DESTINAZIONI} >=0;
                            # vettore di parametri ad un
                            # indice. Rappresenta la quantita'
                            # (non negativa) di merce che deve
```

```
                              # essere trasportata a ciascuna
                              # destinazione.

####   SEZIONE PER LA DICHIARAZIONE DELLE VARIABILI  ####

var x {i in ORIGINI,j in DESTINAZIONI} >= 0;
                      # x[i,j] e' l'elemento di una matrice di
                      # variabili (2 indici) e rappresenta il
                      # quantitativo di merce trasportato dalla
                      # origine 'i' alla destinazione 'j'.


####  SEZIONE PER LA DICHIARAZIONE DELLA FUNZIONE ####
####             OBIETTIVO E DEI VINCOLI          ####

minimize cost_trasporto:  sum{i in ORIGINI,j in DESTINAZIONI}
                          costo_origdest[i,j]*x[i,j];

                  # a ciascun trasporto origine/destinazione
                  # e' associato un costo, i costi poi vengono
                  # sommati

s.t. origini {i in ORIGINI}:
     sum{j in DESTINAZIONI} x[i,j] <= max_uscita[i] ;
              # da ciascuna origine non e' possibile inviare
              # piu' di un determinato quantitativo di merce.


s.t. destinazioni {j in DESTINAZIONI}:
     sum{i in ORIGINI} x[i,j] = domanda[j] ;
          # a ciascuna destinazione deve arrivare
          # esattamente una quantita' determinata di merce.
```

Il file `trasporto1.dat` che permette di specificare i dati da introdurre nel modello precedente è il seguente

`trasporto1.dat`

```
set ORIGINI := asti bergamo como domodossola lecce;
                      # l'insieme ORIGINI ha 5 elementi
```

```
set DESTINAZIONI := mantova napoli olbia palermo;
                      # l'insieme DESTINAZIONI ha 4 elementi

param            max_uscita :=
     asti              100
     bergamo           110
     como              130
     domodossola        85
     lecce             120       ;
                      # il vettore di parametri 'max_uscita'
                      # ha 5 elementi (tanti quante le origini).


param               domanda   :=
     mantova            30
     napoli             18
     olbia              45
     palermo            56        ;
                    # il vettore di parametri 'domanda'
                    # ha 4 elementi (tanti quante le  destina-
                    # zioni). Si noti che quando si assegnano
                    # gli elementi di un solo vettore non sono
                    # necessari i ':' dopo la parola chiave
                    # 'param'


param costo_origdest : mantova  napoli  olbia  palermo :=
          asti            1       3.5     4        4.5
          bergamo          .1     3       4.5      4.8
          como             .3     2.8     4.7      4.9
          domodossola     1       2.2     3.9      4.0
          lecce           1.7     2.2     5.0      5.3   ;
           # la matrice di parametri 'costo_origdest' contiene
           # 4*5 = 20 elementi, uno per ogni coppia origine/de-
           # stinazione; non e' necessario specificare
           # lo '0' (zero) davanti alla virgola.
```

Supponiamo ora che nelle origini ci sia un costo di prelevamento della merce di cui tenere conto nel calcolo del costo totale, aggiuntivo al costo dei trasporti. Nella tabella che segue si riporta i costi unitari di prelevamento da ciascuna città origine.

	Asti	Bergamo	Como	Domodossola	Lecce
costo prelevamento	2	3	4	7	6

Modificare il file `.mod` e il file `.dat` in modo da tener conto di questi costi aggiuntivi.

Chiamati c_i, $i = 1, 2, 3, 4, 5$, i costi unitari di prelevamento nella i-esima città origine, la funzione obiettivo che rappresenta il costo complessivo diventa

$$\sum_{i=1}^{5}\sum_{j=1}^{4} c_{ij}x_{ij} + \sum_{i=1}^{5} c_i \sum_{j=1}^{4} x_{ij}$$

Il file `trasporto1es.mod` che segue riporta il file del modello modificato in accordo con quanto richiesto dall'esercizio

trasporto1es.mod

```
set ORIGINI;
set DESTINAZIONI;

param  costo_origdest {ORIGINI,DESTINAZIONI} >= 0;

param  costo_origine {ORIGINI} >= 0;

param  max_uscita {ORIGINI} >= 0;

param  domanda {DESTINAZIONI} >=0;

var x {i in ORIGINI,j in DESTINAZIONI} >= 0;

minimize cost_trasporto: sum{i in ORIGINI,j in DESTINAZIONI}
           costo_origdest[i,j]*x[i,j] +  sum{i in ORIGINI}
           costo_origine[i]* sum{j in DESTINAZIONI} x[i,j];
```

```
s.t. origini {i in ORIGINI}:
    sum{j in DESTINAZIONI} x[i,j] <= max_uscita[i] ;

s.t. destinazioni {j in DESTINAZIONI}:
    sum{i in ORIGINI} x[i,j] = domanda[j] ;
```

Come si può vedere è stato aggiunta l'istruzione

```
param costo_origine {ORIGINI}
```

per rappresentare i costi di prelevemento da ciascuna città origine. Inoltre è stata modificata la funzione obiettivo.

Il file `trasporto1es.dat` che segue riporta il nuovo file di dati per il modello modificato come richiesto dall'esercizio.

trasporto1es.dat

```
set ORIGINI := asti bergamo como domodossola lecce;

set DESTINAZIONI := mantova napoli olbia palermo;

param:           costo_origine  max_uscita :=
     asti              2          100
     bergamo           3          110
     como              4          130
     domodossola       7           85
     lecce             6          120       ;

param               domanda   :=
     mantova          30
     napoli           18
     olbia            45
     palermo          56        ;

param costo_origdest : mantova  napoli  olbia  palermo :=
          asti            1       3.5     4        4.5
          bergamo          .1     3       4.5      4.8
          como             .3     2.8     4.7      4.9
```

```
domodossola     1       2.2     3.9     4.0
lecce           1.7     2.2     5.0     5.3   ;
```

Introduciamo ora ulteriori modifiche del modello dei trasporti considerato, prendendo in considerazione elementi aggiuntivi che sono di solito presenti nei problemi di trasporto. Supponiamo quindi che ci sia

1. un costo aggiuntivo da imputarsi a tasse portuali ad ogni unità di merce trasportata alla città di Olbia pari a 0.1;
2. un vincolo che impone che almeno i 4/5 della merce trasportata provenga da città origine del nord Italia;
3. un vincolo che impone che almeno i 2/3 della merce trasportata raggiunga città destinazione del sud Italia e delle isole.
4. un ulteriore costo di trasporto per ogni unità di merce trasportata per qualche coppia di città origine e città destinazione da imputarsi, ad esempio, a pedaggi autostradali; la tabella che segue riporta per quali coppia di città esiste questo costo aggiuntivo e a quanto ammonta

Bergamo—Napoli	2	Domodossola—Mantova	0.5	Lecce—Olbia	3.5
Bergamo—Olbia	3.5	Domodossola—Palermo	3.8	Lecce—Palermo	3.1

Per tener conto di queste esigenze aggiuntive, sarà necessario introdurre nuovi insiemi nel modello in AMPL: l'insieme `NORD` delle città del nord, l'insieme `SUD` delle città del sud e l'insieme `ISOLE` delle città che si trovano nelle isole. Inoltre si dovrà fare uso degli operatori tra insiemi di *intersezione* e *unione* (`inter` e `union`). L'uso di questi operatori è molto semplice e permette di definire gli insiemi intersezione e unione, ad esempio, nel seguente modo:

```
set ORIGINI;            # insieme delle citta' origini
set DESTINAZIONI;       # insieme delle citta' destinazioni
set NORD;               # insieme di alcune citta' del nord
set SUD;                # insieme di alcune citta' del sud
set ISOLE;              # insieme di alcune citta' delle isole

set ORI_NORD := ORIGINI inter NORD;
                        # insieme intersezione degli
                        # insiemi ORIGINI e NORD

set DEST_SUDISOLE := DESTINAZIONI inter {SUD union ISOLE};
```

```
                        # insieme intersezione dell'insieme
                        # DESTINAZIONI con l'insieme
                        # (SUD unione ISOLE)
```

Il file `trasporto2.mod` che segue riporta il nuovo modello di trasporti che tiene conto delle nuove esigenze sia nella funzione obiettivo sia per la presenza di nuovi vincoli.

trasporto2.mod

```
####    SEZIONE PER LA DICHIARAZIONE DEGLI INSIEMI    ####

set ORIGINI;
set DESTINAZIONI;
set NORD;
set SUD;
set ISOLE;

set ORI_NORD := ORIGINI inter NORD;

set DEST_SUDISOLE := DESTINAZIONI inter {SUD union ISOLE};
                        # introduce l'insieme intersezione
                        # dell'insieme DESTINAZIONI con l'insieme
                        # (SUD unione ISOLE)

      # le due dichiarazioni che seguono definiscono l'insieme
      # COPPIE formato da coppie ordinate di elementi: il primo
      # appartenente ad ORIGINI, il secondo a DESTINAZIONI.
      # Tuttavia, mentre la prima dichiarazione definisce COPPIE
      # come prodotto cartesiano e quindi contiene TUTTE le coppie
      # possibili, la seconda lo definisce solo come un sottoinsieme
      # del prodotto cartesiano (i cui elementi devono essere
      # specificati nel file .dat)

#set COPPIE := {ORIGINI,DESTINAZIONI};
set COPPIE within {ORIGINI,DESTINAZIONI};

param  costo_origdest {ORIGINI,DESTINAZIONI} >= 0;

param  costo_origine {ORIGINI} >= 0;
```

```
param  max_uscita {ORIGINI} >= 0;

param  domanda {DESTINAZIONI} >=0;

param costo_coppie {COPPIE} >=0;
            # per gli elementi  appartenenti all'insieme
            # COPPIE vi e' un ulteriore costo
            # di trasporto (pedaggio autostradale)

param costo_portuale;
            # costo unitario da imputarsi ad ogni unita' di
            # prodotto trasportato alla destinazione "olbia"

var x {i in ORIGINI,j in DESTINAZIONI} >= 0;

minimize cost_trasporto:  sum{i in ORIGINI,j in DESTINAZIONI}
       costo_origdest[i,j]*x[i,j] +  sum{i in ORIGINI}
       costo_origine[i]* sum{j in DESTINAZIONI} x[i,j] +
       sum{(i,j) in COPPIE} costo_coppie[i,j] * x[i,j] +
       sum{(i,"olbia") in {ORIGINI,DESTINAZIONI}}
                          costo_portuale * x[i,"olbia"];

               # a ciascun trasporto origine/destinazione
               # e' associato un costo, i costi poi vengono
               # sommati; agli elementi appartenenti allo
               # insieme COPPIE e' associato un costo in piu'.

s.t. origini {i in ORIGINI}:
     sum{j in DESTINAZIONI} x[i,j] <= max_uscita[i] ;

s.t. destinazioni {j in DESTINAZIONI}:
     sum{i in ORIGINI} x[i,j] = domanda[j] ;

s.t. origini_nord: sum {i in ORI_NORD,j in DESTINAZIONI}
      x[i,j] >= 4/5*sum {i in ORIGINI,j in DESTINAZIONI} x[i,j] ;
         # dalle citta' di origine del nord vengono effettuati
         # almeno i 4/5 dei trasporti totali.

s.t. destinazioni_isole:
      sum {i in ORIGINI,j in DESTINAZIONI : j in {SUD union ISOLE}}
      x[i,j] >= 2/3 * sum {i in ORIGINI,j in DESTINAZIONI} x[i,j] ;
```

```
          # verso le citta' di destinazione del sud e delle
          # isole vengono effettuati almeno i 2/3 dei trasporti
          # totali.
```

Il file di dati `trasporto2.dat` che segue permette di introdurre i dati assegnati nel modello.

`trasporto2.dat`

```
set ORIGINI := asti bergamo como domodossola lecce;

set DESTINAZIONI := mantova napoli olbia palermo;

set NORD := asti varese bergamo brescia como cremona pavia
            domodossola mantova bergamo padova
            sondrio torino milano venezia;

set SUD := bari lecce napoli cosenza foggia salerno
           reggiocalabria taranto crotone;

set ISOLE := cagliari olbia palermo sassari trapani;

param:           costo_origine  max_uscita :=
     asti              2          100
     bergamo           3          110
     como              4          130
     domodossola       7           85
     lecce             6          120       ;

param               domanda   :=
     mantova          30
     napoli           18
     olbia            45
     palermo          56        ;

param costo_origdest : mantova  napoli  olbia  palermo :=
          asti            1       3.5     6.0      4.5
          bergamo          .1     3       4.5      4.8
          como             .3     2.8     4.7      4.9
          domodossola     1       2.2     3.9      4.0
```

```
          lecce          1.7     2.2     5.0      5.3   ;

param: COPPIE :                  costo_coppie   :=
        bergamo napoli              2
        bergamo olbia               3.5
        domodossola mantova         0.5
        domodossola palermo         3.8
        lecce olbia                 3.5
        lecce palermo               3.1          ;
           # le possibili coppie origine/destinazione sono
           # 5*4 = 20 ma l'insieme COPPIE ne contiene solo 6

param costo_portuale := 0.1 ;
   # per ogni unita' di merce che ha come destinazione "olbia"
   # vi e' questo costo aggiuntivo;
```

Si osservi che in questo file dei dati non è stato assegnato esplicitamente l'insime `COPPIE` con un'istruzione del tipo

```
set COPPIE :=
   (bergamo , napoli) (bergamo , olbia) (domodossola , mantova)
   (domodossola , palermo)  (lecce , olbia)  (lecce , palermo);
```

oppure

```
set COPPIE :=
   bergamo napoli
   bergamo olbia
   domodossola mantova
   domodossola palermo
   lecce olbia
   lecce palermo;
```

Infatti, AMPL, in questo caso prenderà come elementi dell'insieme `COPPIE` (che ricordiamo è stato dichiarato come *sottoinsieme* del prodtto cartesiano) tutte e sole le coppie specificate nell'assegnazione del parametro `costo_coppie`.

3.6.2 Un problema di produzione industriale multiperiodo

Esaminiamo ora un modello per la produzione industriale *multiperiodo* che implementeremo in AMPL utilizzando alcune delle strutture del linguaggio che abbiamo introdotto nel capitolo precedente. In particolare, vedremo l'uso oltre che degli insiemi generici, anche degli insiemi numerici, ordinati e ciclici.

Esempio 3.6.2 *Una fabbrica produce due tipi di pneumatici A e B ed ha una gestione trimestrale della produzione. Per i prossimi tre mesi deve soddisfare il seguente ordine (espresso in numero di pneumatici richiesti ciascun mese)*

	tipo A	**tipo B**
ottobre	*16000*	*14000*
novembre	*7000*	*4000*
dicembre	*4000*	*6000*

Per la produzione di questi pneumatici la fabbrica dispone di due linee di produzione **L1** *e* **L2**. *Per avere un pneumatico finito e pronto per essere venduto, è necessaria la lavorazione di materiale grezzo su solo una delle due linee di produzione. Il numero di ore in cui le linee di produzione sono disponibili ciascun mese sono riportate nella seguente tabella*

	L1	**L2**
ottobre	*2000*	*3000*
novembre	*400*	*800*
dicembre	*200*	*1000*

I tempi necessari per produrre questi pneumatici variano a seconda del tipo e della linea di produzione usata. Tali tempi sono riportati nella seguente tabella (in ore)

	L1	**L2**
tipo A	*0.10*	*0.12*
tipo B	*0.12*	*0.18*

Il costo di ogni ora di lavorazione su una linea di produzione è uguale per entrambe le linee ed è pari a 6 euro. Il costo del materiale grezzo necessario per produrre ciascun pneumatico è di euro 2.50 per il tipo A e di euro 4.00 per il tipo B.
Nel primo e nel secondo mese del trimestre è possibile produrre più di quanto richiesto nello stesso mese; la produzione in eccesso deve essere immagazzinata per essere usata nel mese successivo. Ogni mese, il costo di tale immagazzinamento è di euro 0.35 per ciascun pneumatico immagazzinato. Si assuma che all'inizio del trimestre non ci sia nessun prodotto immagazzinato e analogamente alla fine del trimestre non rimanga nessun prodotto immagazzinato.
Costruire un modello lineare che permetta di pianificare la produzione trimestrale minimizzando il costo complessivo trascurando l'interezza dei prodotti.

Formulazione.
Si tratta di un problema di allocazione ottima di risorse nel quale si deve tenere presente la possibilità dell'immagazzinamento del prodotto in eccesso (allocazione ottima multiperiodo).

– *Variabili.* Si introducono le variabili A_{Li}^{ott}, A_{Li}^{nov}, A_{Li}^{dic} che indicano la quantità di pneumatici di tipo A prodotti dalla $i-$esima linea produzione ($i = 1, 2$) rispettivamente nei mesi di ottobre, novembre e dicembre. Analogamente B_{Li}^{ott}, B_{Li}^{nov}, B_{Li}^{dic} indicheranno le quantità di pneumatici di tipo B prodotti dalla $i-$esima linea di produzione ($i = 1, 2$) rispettivamente nei mesi di ottobre, novembre e dicembre. Si indichino inoltre con A_{im}^{ott}, A_{im}^{nov}, B_{im}^{ott}, B_{im}^{nov} le quantità di pneumatici di tipo A e B da immagazzinare nei mesi di ottobre e novembre.

– *Funzione obiettivo.* La funzione obiettivo da minimizzare è data dal costo complessivo di produzione. Poiché un'ora di lavorazione su una linea di produzione costa 6 euro, e poiché i tempi di lavorazione cambiano a seconda della linea di produzione utilizzata, per produrre ciascun pneumatico di tipo A si spende euro 0.60 se si utilizza la linea **L1** e euro 0.72 se si utilizza la linea **L2**. Analogamente, il costo di ciascun pneumatico del tipo B è di euro 0.72 se si utilizza la macchina 1, e di euro 1.08 se si utilizza la linea **L2**. Quindi tenendo conto del costo del materiale grezzo e dell'immagazzinamento, il costo complessivo sarà

$$0.6(A_{L1}^{ott} + A_{L1}^{nov} + A_{L1}^{dic}) + 0.72(A_{L2}^{ott} + A_{L2}^{nov} + A_{L2}^{dic}) +$$

$$+0.72(B_{L1}^{ott} + B_{L1}^{nov} + B_{L1}^{dic}) + 1.08(B_{L2}^{ott} + B_{L2}^{nov} + B_{L2}^{dic}) +$$

$$+2.50(A_{L1}^{ott} + A_{L1}^{nov} + A_{L1}^{dic} + A_{L2}^{ott} + A_{L2}^{nov} + A_{L2}^{dic}) +$$

$$+4.00(B_{L1}^{ott} + B_{L1}^{nov} + B_{L1}^{dic} + B_{L2}^{ott} + B_{L2}^{nov} + B_{L2}^{dic}) +$$

$$+0.35(A_{im}^{ott} + A_{im}^{nov} + B_{im}^{ott} + B_{im}^{nov}).$$

– *Vincoli.* I vincoli dovuti alla disponibilità limitata delle macchine sono

$$\begin{aligned} 0.10A_{L1}^{ott} + 0.12B_{L1}^{ott} &\leq 2000 \\ 0.10A_{L1}^{nov} + 0.12B_{L1}^{nov} &\leq 400 \\ 0.10A_{L1}^{dic} + 0.12B_{L1}^{dic} &\leq 200 \end{aligned}$$

$$\begin{aligned} 0.12A_{L2}^{ott} + 0.18B_{L2}^{ott} &\leq 3000 \\ 0.12A_{L2}^{nov} + 0.18B_{L2}^{nov} &\leq 800 \\ 0.12A_{L2}^{dic} + 0.18B_{L2}^{dic} &\leq 1000. \end{aligned}$$

Si hanno inoltre i seguenti vincoli dovuti alla richiesta e all'immagazzinamento:

$$A_{L1}^{ott} + A_{L2}^{ott} = 16000 + A_{im}^{ott}$$

$$\begin{aligned} A_{L1}^{nov} + A_{L2}^{nov} + A_{im}^{ott} &= 7000 + A_{im}^{nov} \\ A_{L1}^{dic} + A_{L2}^{dic} + A_{im}^{nov} &= 4000 \end{aligned}$$

$$\begin{aligned} B_{L1}^{ott} + B_{L2}^{ott} &= 14000 + B_{im}^{ott} \\ B_{L1}^{nov} + B_{L2}^{nov} + B_{im}^{ott} &= 4000 + B_{im}^{nov} \\ B_{L1}^{dic} + B_{L2}^{dic} + B_{im}^{nov} &= 6000. \end{aligned}$$

Si hanno infine i vincoli di non negatività sulle variabili. Quindi il modello finale è:

$$\begin{cases} \min\Big(3.1(A_{L1}^{ott} + A_{L1}^{nov} + A_{L1}^{dic}) + 3.22(A_{L2}^{ott} + A_{L2}^{nov} + A_{L2}^{dic}) + \\ \quad +4.72(B_{L1}^{ott} + B_{L1}^{nov} + B_{L1}^{dic}) + 5.08(B_{L2}^{ott} + B_{L2}^{nov} + B_{L2}^{dic}) + \\ \quad +0.35(A_{im}^{ott} + A_{im}^{nov} + B_{im}^{ott} + B_{im}^{nov})\Big) \\ 0.10A_{L1}^{ott} + 0.12B_{L1}^{ott} \leq 2000 \\ 0.10A_{L1}^{nov} + 0.12B_{L1}^{nov} \leq 400 \\ 0.10A_{L1}^{dic} + 0.12B_{L1}^{dic} \leq 200 \\ 0.12A_{L2}^{ott} + 0.18B_{L2}^{ott} \leq 3000 \\ 0.12A_{L2}^{nov} + 0.18B_{L2}^{nov} \leq 800 \\ 0.12A_{L2}^{dic} + 0.18B_{L2}^{dic} \leq 1000 \\ A_{L1}^{ott} + A_{L2}^{ott} = 16000 + A_{im}^{ott} \\ A_{L1}^{nov} + A_{L2}^{nov} + A_{im}^{ott} = 7000 + A_{im}^{nov} \\ A_{L1}^{dic} + A_{L2}^{dic} + A_{im}^{nov} = 4000 \\ B_{L1}^{ott} + B_{L2}^{ott} = 14000 + B_{im}^{ott} \\ B_{L1}^{nov} + B_{L2}^{nov} + B_{im}^{ott} = 4000 + B_{im}^{nov} \\ B_{L1}^{dic} + B_{L2}^{dic} + B_{im}^{nov} = 6000 \\ A_{Li}^{ott} \geq 0, A_{Li}^{nov} \geq 0, A_{Li}^{dic} \geq 0, \ i = 1, 2 \\ B_{Li}^{ott} \geq 0, B_{Li}^{nov} \geq 0, B_{Li}^{dic} \geq 0, \ i = 1, 2. \\ A_{im}^{ott} \geq 0, A_{im}^{nov} \geq 0, B_{im}^{ott} \geq 0, B_{im}^{nov} \geq 0, \end{cases}$$

Decidiamo di definire un insieme *generico* che chiameremo `tipi` per quanto riguarda il tipo di pneumatico (tipo A e tipo B). Per quanto riguarda le linee, pensando all'attribuzione a ciascuna linea di un numero intero, definiamo l'insieme delle linee (`linee`) come un insieme numerico formato dai numeri interi da 1 a `numlinee`, dove `numlinee` è un parametro che corrisponde al numero di linee presenti e che verrà assegnato nel file dei dati. Naturalmente, la dichiarazione del parametro `numlinee` dovrà precedere quella dell'insieme `linee`. Si ribadisce che in questo modo il modello definito nel file dei dati è indipendente dai dati che saranno introdotti nel file `.dat`. Infine, si sceglie di utilizzare un insieme (ordinato) ciclico per quanto riguarda l'insieme dei mesi. Questa scelta è dettata dai vincoli che dovranno essere implementati e sarà esaminata nel dettaglio nel seguito.

Per quanto riguarda le variabili, sono state introdotte le variabili con tre indici `x{tipi,linee,mesi}` e le variabili `x_im{tipi,mesi}` relative alle quantità immagazzinate. Poiché nell'ultimo prediodo non è previsto l'immagazzinamento, sarà necessario imporre che il quantitativo di pneumatici immagazzinato nell'ultimo periodo sia pari a 0.

Un file `.mod` che rappresenta bene il modello in esame è riportato di seguito.

pneumatici.mod

```
set tipi;   # insieme generico

param numlinee integer;   # parametro numlinee con
                          # controllo dell'interezza

set linee:= 1..numlinee;    # insieme numerico

set mesi circular;   # insieme ordinato ciclico

param ordine{mesi,tipi}>=0;
param disponibilita{mesi,linee}>=0;
param tempi{tipi,linee}>=0;

param costo_orario>=0;
param costo_immagazzinamento>=0;

param costo_materiale_grezzo{tipi}>=0;

var x{tipi,linee,mesi}>=0;  # quantita' di pneumatici di
                            # ciascun tipo prodotti da
                            # ciascuna linea in ciascun mese

var x_im{tipi,mesi}>=0; # quantita' di pneumatici di ciascun
                        # tipo immagazzinato in ciascun mese
                        # ove questo e' possibile
                        # (sara' quindi necessario porre
                        # x_im[i,last(mesi)]=0
                        # al variare di {i in tipi})


minimize costo_totale:
 sum{i in tipi, j in linee, k in mesi}
                         costo_orario*tempi[i,j]*x[i,j,k] +
```

```
 sum{i in tipi, j in linee, k in mesi}
                          costo_materiale_grezzo[i]*x[i,j,k]+
 sum{i in tipi, l in mesi} costo_immagazzinamento*x_im[i,l];

# vincoli disponibilita' delle macchine

s.t. vincoli_disponibilita_linee{k in mesi, j in linee}:
      sum{i in tipi} tempi[i,j]*x[i,j,k] <= disponibilita[k,j];



# vincoli dovuti alla richiesta e all'immagazzinamento

s.t. vincoli_richiesta{k in mesi, i in tipi}:
        x_im[i,prev(k,mesi)]+sum{j in linee} x[i,j,k] =
                                  ordine[k,i]+x_im[i,k];
```

Vale la pena soffermarci sulla definizione dei vincoli. Iniziamo dai vincoli sulla disponibilità delle macchine. Si tratta di esprimere un vincolo per ogni linea di produzione. Quindi, in prima analisi, potevamo scrivere i seguenti vincoii:

```
s.t. vincoli_disponibilita_linea1{k in mesi}:
      sum{i in tipi} tempi[i,1]*x[i,1,k] <= disponibilita[k,1];

s.t. vincoli_disponibilita_linea2{k in mesi}:
      sum{i in tipi} tempi[i,2]*x[i,2,k] <= disponibilita[k,2];
    .
    .
    .
```

Ma questa scrittura, oltre che essere lunga, prevede di conoscere già nel file del modello quante sono le linee di produzione, dato che invece deve essere un parametro assegnato nel file dei dati.

Per quanto riguarda i vincoli dovuti alla richiesta e all'immagazzinamento, essendo l'insieme `mesi` un insieme ciclico, possiamo evitare di tenere separati i vincoli riguardanti il primo e l'ultimo mese purché si imponga che il quantitativo di pneumatici immagazzinato nell'ultimo periodo sia pari a 0. Anche in questo caso, per ragioni analoghe, si è evitato di scrivere i vincoli nella forma

```
s.t. vincoli_richiesta_tipoA{k in mesi}:
```

```
            x_im["tipoA",prev(k,mesi)]+
            sum{j in linee} x["tipoA",j,k] =
            ordine[k,"tipoA"]+x_im["tipoA",k];

s.t. vincoli_richiesta_tipoB{k in mesi}:
            x_im["tipoB",prev(k,mesi)]+
            sum{j in linee} x["tipoB",j,k] =
            ordine[k,"tipoB"]+x_im["tipoB",k];
    .
    .
    .
```

L'assegnazione dei parametri nel file `.dat` è molto standard. Come abbiamo già detto dobbiamo *fissare* a 0 le variabili che identificano le quantità di pneumatici immagazzinati nell'ultimo mese. Il fixing delle variabili si può effettuare o nel file `.dat` o nel file `.run`. In questo caso, trattandosi di una richiesta parte integrante del modello, lo effettuiamo nel file `.dat`. Si tratterebbe di scrivere le istruzioni

```
fix x_im["tipoA",last(mesi)]:=0;
fix x_im["tipoB",last(mesi)]:=0;
  .
  .
  .
```

Anche in questo caso, sia per brevità di scrittura, ma soprattutto per rendere il modello valido anche nel caso in cui si cambino i dati, scegliamo di fissare le variabili lasciando variare il *tipo* all'interno dell'insieme `tipi`. In questo caso si può procedere utilizzando una sintassi del linguaggio AMPL non ancora introdotta: si tratta dell'istruzione `for`. Il suo uso verrà specificato meglio nel seguito, ma la sintassi è abbastanza standard. In questo caso è

```
for {i in tipi}{fix x_im[i,last(mesi)]:=0;}
```

Quindi il file del dati per il modello in esame può essere scritto come segue:

pneumatici.dat

```
set mesi := ott nov dic;
set tipi := tipoA tipoB;

param numlinee := 2;

param ordine:   tipoA tipoB :=
       ott     16000  14000
```

```
       nov      7000   4000
       dic      4000   6000;

param disponibilita:       1     2 :=
                ott      2000  3000
                nov       400   800
                dic       200  1000;

param tempi:                 1     2 :=
                tipoA      0.10  0.12
                tipoB      0.12  0.18;

param costo_orario := 6;

param costo_immagazzinamento := 0.35;

param costo_materiale_grezzo :=
                        tipoA 2.50
                        tipoB 4;

for {i in tipi}{fix x_im[i,last(mesi)]:=0;}
```

4
I principali comandi AMPL

Abbiamo già utilizzato alcuni comandi AMPL per poter determinare la soluzione ottima di alcuni esempi di modelli. Ad esempio, abbiamo già utilizzato il comando `solve` per invocare il solutore, oppure il comando `option solver cplex` per definire il solutore da utilizzare ed anche il comando `display` per visualizzare il risultato ottenuto. Vogliamo ora dare un quadro più generale riguardante l'uso di questi comandi.
Innanzitutto, i comandi vengono dati su riga di comando al prompt di AMPL, oppure sono inseriti in un file `.run`. Essi possono essere di fatto utilizzati per scrivere degli *script* che permettono di realizzare un'implementazione di veri e propri programmi in AMPL.

4.1 IL COMANDO `OPTION`

Il comando `option` serve per visualizzare o cambiare il valore delle *opzioni*. Le *opzioni* sono *variabili di stato* dell'ambiente AMPL. Ciascuna di esse ha un nome ed un valore che può essere un numero o una stringa di caratteri. Per aver un'idea di quali sono le variabili di stato in AMPL digitare sul prompt dei comandi di AMPL il comando

```
option;
```

Verrà visualizzato un lungo elenco di tutte le variabili di stato di AMPL e il loro valore corrente. Il comando `option` senza ulteriore specificazione serve infatti per visualizzare il valore delle variabili di stato. Il comando `option` accetta una "wild card" che è rappresentata dal carattere "$*$" ed è utilizzato per rappresentare qualsiasi stringa. Quindi, ad esempio, con il comando `option presolve*` si

otterrà la lista di tutte le opzioni il cui nome inizia per `presolve` e il loro valore corrente.

Per visualizzare il valore corrente di un'opzione specifica si dovrà specificare il `nomeopzione`. Quindi per visualizzare il valore dell'opzione `nomeopzione` sarà sufficiente specificare

```
option nomeopzione;
```

Per modificare il valore dell'opzione `nomeopzione` sarà sufficiente il comando che indichi questo nuovo valore:

```
option nomeopzione nuovovalore;
```

Abbiamo già visto un esempio di questo comando quando abbiamo selezionato il solutore da utilizzare con il comando `option solver cplex`. In questo caso la variabile di stato è `solver` che viene impostata al valore `cplex`.

Prestare molta attenzione al fatto che il comando `option` non controlla subito che il valore assegnato abbia senso o meno; un messaggio di errore si avrà solo in fase di esecuzione.

Infine, per riportare tutte le opzioni al loro valore di default si utilizza il comando

```
reset options;
```

Per un elenco completo di tutte le opzioni si rimanda al testo [Fourer et al., 2003]. Ne riportiamo di seguito solamente tre di uso frequente:

- `solver` specifica il solutore. Ha un suo valore default e può essere cambiato utililzzando il nome degli altri solutori.
- `presolve` specifica le opzioni del preprocessamento. Il preprocessamento è un'operazione che AMPL può effettuare allo scopo di ridurre il problema, ad esempio, eliminando vincoli ridondanti, fissando valori di alcune variabili, etc. Tale procedimento è molto utile (e a volte indispensabile) nella risoluzione di problemi a grandi dimensioni. Il valore di default è 10. Per inibire il preprocessamento è sufficiente assegnare valore 0 a `presolve`.
- `show_stats` specifica il livello di dettaglio delle informazioni sul problema e sulla soluzione che devono essere visualizzate. Il valore di default è 0, in corrispondenza del quale vengono visualizzate informazione minime. Assegnando il valore 1 o superiori a `show_stats` aumenta il livello di dettaglio delle informazioni visualizzate.

Attraverso il comando `option` si possono inoltre specificare le opzioni relative al solutore utilizzato.

4.2 IL COMANDO DISPLAY

Il comando `display` si utilizza per visualizzare oggetti presenti nel modello come, ad esempio, gli elementi di un insieme, il valore delle variabili, dei parametri, della funzione obiettivo, dei vincoli. Nella sua versione più semplice consente di visualizzare il valore di un oggetto denominato `nomeoggetto` tramite il comando

```
display nomeoggetto;
```

Dopo il comando `display` possono essere anche elencati un certo numero di oggetti da visualizzare separati dalla virgola. Con il comando `display` possono essere anche utilizzate espressioni algebriche o logiche come riportato negli esempi che seguono (facendo riferimento agli oggetti utilizzati nell'Esempio 3.6.2):

```
display mesi;
display x;
display costo_totale;
display {i in tipi} x[i,1,"nov"];

display sum{i in tipi, j in linee, k in mesi}
                    costo_materiale_grezzo[i]*x[i,j,k];

display {i in tipi, k in mesi : x[i,1,k] > 100};
```

Non forniamo spiegazioni dettagliate di queste istruzioni perché sono molto intuitive. Ci soffermiamo solamente sull'uso dei ":" che può essere introdotto nei costrutti logici, come nell'ultimo comando `display` dell'esempio, con il significato di "*tale che*".

Le opzioni del comando `display` riguardano la formattazione delle informazione da visualizzare e l'approssimazione utilizzata nell'arrotondamento dei valori numerici da visualizzare. Per esse si fa riferimento al Capitolo 12 di [Fourer et al., 2003] ed in particolare alle Tabelle 12-1 e 12-2.

4.3 REINDIRIZZAMENTO DELL'OUTPUT DEI COMANDI

È molto utile poter reindirizzare l'output dei comandi in un file nel quale conservare tale output. Questo vale per tutti i comandi, ma in particolare per il comando `display`. Infatti, in questo modo si può facilmente memorizzare la soluzione ottima e altre informazioni. Per creare un file testo di output `nomefile.out` nel quale reindirizzare l'output del comando `display` è sufficiente il comando

```
display oggetto > nomefile.out;
```

In questo modo viene creato o eventualmente sovrascritto (se già esistente) il file `nomefile.out` nel quale verrà scritto l'output del comando. Se si vuole "appendere", ovvero aggiungere alla fine del file, altri output è sufficiente il comando

```
display oggetto2 >> nomefile.out;
```

In questo modo, nel file `nomefile.out`, dopo il/i valore/i di `oggetto` compariranno il/i valore/i di `oggetto2`.

Quindi, sempre in riferimento all'Esempio 3.6.2, è possibile aggiungere nel file `.run` (ovviamente dopo il comando `solve`) i comandi

```
display x > risultati.out;
display x_im >> risultati.out;
display costo_totale >> risultati.out;
```

per creare il file `risultati.out` contenente i valori della variabili all'ottimo e il valore ottimo della funzione obiettivo.

4.4 IL COMANDO DISPLAY PER VISUALIZZARE ALTRE GRANDEZZE RELATIVE ALLE VARIABILI E AI VINCOLI

Nella risoluzione di problemi di Programmazione Lineare, AMPL oltre a fornire (ove esista) la soluzione ottima del problema, permette di visualizzare anche altri elementi del problema come i prezzi ombra i costi ridotti associati alla soluzione ottima. Per visualizzare questi elementi è sufficiente aggiungere dei suffissi al nome della variable. In particolare, se x è una variabile del problema, possiamo utilizzare il comando

```
display x.lb, x.ub;
```

per visualizzare il lower bound e l'upper bound della variabile x. Quindi, ad esempio, se x è una variabile definita non negativa, il comando fornirà il valore `0` per il lower bound e il valore `Infinity` per l'upper bound. Il comando

```
display x.slack;
```

visualizza la differenza tra il valore della variabile e il più vicino bound.

Il concetto di "bound" e di "slack" ha un'interpretazione analoga anche per i vincoli di un modello. Ovvero si pensa al vincolo scritto nella forma

$$\textit{lower bound} \quad \leq \quad \textit{body} \quad \leq \quad \textit{upper bound}$$

e quindi, se `vinc` è l'etichetta data ad un vincolo, il comando

```
display vinc.lb, vinc.body, vinc.ub;
```

visualizza il lower bound del vincolo, il valore della parte variabile del vincolo e l'upper bound del vincolo. Il comando

```
display vinc.slack;
```

visualizza la differenza tra il valore del vincolo e il più vicino bound.

Il comando `display` si può utilizzare anche per avere informzioni sulle quantità *duali* associate al problema. Si ricorda che nella Programmazione Lineare associata a ciascun vincolo c'è una variabile duale e il valore ottimo di tale variabile viene chiamato *prezzo ombra* o *valore marginale*. Con il comando

```
display vincolo;
```

si visualizza tale valore, ovviamente senza la necessità di dover costruire esplicitamente il problema duale. Similmente il comando

```
display x.rc;
```

visualizza il *costo ridotto* associato alla variabile x.

L'uso delle quantità duali e la loro interpretazione verrà trattata nel dettaglio nel prossimo Capitolo 5.3 nel quale verrà affrontata *l'analisi di sensitività* della soluzione ottima rispetto a parametri di un problema di Programmazione Lineare.

4.5 COMANDI PER AGGIORNARE IL MODELLO: RESET, DROP E RESTORE

Sono disponibili comandi per modificare anche solo parzialmente un modello. Il comando `reset`, già utilizzato, cancella completamente il modello e i dati. Equivale ad uscire (con il comando `quit`) da AMPL e rientrare. Esistono poi comandi per far in modo che alcuni vincoli siano ignorati. Il comando è `drop`. Quindi utilizzando i comandi

```
drop vincolo1;
drop vincoli_risorse{i in RISORSE};
drop vincolo_budget["periodo1"];
```

si ottiene che i vincoli corrispondenti vengano ignorati. Con il comando `restore` si ripristinano vincoli che fossero stati eventualmente in precedenza "ignorati".

4.6 ALTRI UTILI COMANDI: SHOW, XREF, EXPAND

Sono comandi che servono per visualizzare componenti del modello.

- Il comando `show` visualizza tutte le componenti del modello, ovvero parametri, insiemi, variabili, vincoli e funzione obiettivo.

- Il comando `xref` visualizza tutte le componenti del modello che dipendono da una specifica componente.
- Il comando `expand` applicato ad una componente scritta con una espressione parametrica genera esplicitamente tutti i termini dell'espressione. Applicato ad una variabile, visualizza tutti i coefficienti non nulli di questa variabile nei termini lineari della funzione obiettivo e dei vincoli. Se inoltre la variabile compare anche in espressioni non lineari allora viene aggiunto all'output l'espressione `+ nonlinear`.

4.7 NOMI GENERICI PER VARIABILI, VINCOLI, E FUNZIONI OBIETTIVO

AMPL rende disponibili parametri che forniscono il *numero* delle variabili, dei vincoli e delle funzioni obiettivo del problema:

- `_nvars`: numero delle variabili del problema
- `_ncons`: numero dei vincoli del problema
- `_nobjs`: numero delle funzioni obiettivo del problema.

Sono disponibili inoltre parametri che forniscono i *nomi* delle componenti del problema:

- `_varname`: nomi delle variabili del problema
- `_conname`: nomi dei vincoli del problema
- `_objname`: nomi delle funzioni obiettivo del problema.

Infine, sono disponibili sinoninmi per le componenti del problema:

- `_var`: sinonimo per le variabili del problema
- `_con`: sinonimo per i vincoli del problema
- `_obj`: sinonimo per le funzioni obiettivo del problema

Utilizzando questi *sinonimi* è possibile scrivere un file `.run` che può essere utilizzato per la soluzione di problemi diversi senza dover indicare volta per volta il nome specifico delle variabili e della funzione obiettivo nel comando `display`. Un esempio di un tale file `.run` è il seguente:

```
reset;
model modello.mod;
data modello.dat;
```

```
option solver cplex;
solve;

display _varname, _var;
display _objname, _obj;
```

5

Modelli di Programmazione Lineare

Abbiamo già visto alcuni esempi di modelli di Programmazione Lineare nei paragrafo 3.6. Come è ben noto si tratta di problemi di Programmazione Matematica nei quali tutte le funzioni che compaiono sono lineari nelle variabili di decisione. Ovviamente, il solutore da utilizzare andrà scelto tra i solutori di Problemi di Programmazione Lineare disponibili. In questo capitolo vedremo altri esempi di tali modelli. Verrà inoltre trattata la *dualità* e l'*analisi della sensitività* nella Programmazione Lineare.

5.1 UN PROBLEMA DI PIANIFICAZIONE DELLA PRODUZIONE

Esempio 5.1.1 *Un'azienda metallurgica deve decidere il piano di produzione per la prossima settimana. Prendendo come input lastre di acciaio, l'azienda produce tre tipi di semilavorati: nastri, bobine e placche. Il processo di trasformazione dell'acciaio si compone di due fasi: il riscaldamento e l'arrotolamento. I tassi di produzione, ovvero le tonnellate prodotte in un'ora lavorativa per ciascuna fase sono riportate nella tabella seguente:*

	fase 1 (riscaldamento)	fase 2 (arrotolamento)
nastri	200	200
bobine	200	140
placche	200	160

Con la vendita dei prodotti si ottengono i seguenti profitti espressi in Euro a tonnellata:

	profitto
nastri	25
bobine	30
placche	29

Inoltre, per ogni prodotto è noto l'ordinativo, ovvero la richiesta massima del mercato di ciascun prodotto:

	ordini
nastri	6000
bobine	4000
placche	3500

Il problema dell'azienda è il seguente: se sono disponibili solo 35 ore di tempo per il riscaldamento (fase 1) e 40 per l'arrotolamento (fase 2), qual è il piano di produzione che massimizza il profitto totale ?

- *Formulare il problema come modello di Programmazione Lineare.*
- *Scrivere il modello in* AMPL *in forma parametrica e risolverlo (fornire i file* `acciaio.mod`*,* `acciaio.dat` *e* `acciaio.run`*).*

Supponiamo, ora di voler introdurre le modifiche elencate nel seguito:

1. *Modificare il modello per tenere conto che per ragioni contrattuali l'azienda deve soddisfare il seguente ordine minimo per ogni prodotto: 1000 ton. di nastri, 500 ton. di bobine e 750 ton. di placche e determinare la soluzione ottima.*
2. *Modificare il modello per imporre che, per ogni fase produttiva, la quantità prodotta uguagli la capacità massima produttiva, determinare la soluzione ottima e confrontarla con quella ottenuta al punto precedente.*
3. *Modificare il modello per tenere conto che, per questioni di magazzino la produzione totale non può superare un numero massimo di 6500 ton. Spiegare come questo nuovo vincolo cambia i risultati.*
4. *Cambiare l'obiettivo del modello: supponiamo che l'azienda voglia ora massimizzare la produzione totale. Come cambia la soluzione ottima rispetto al problema originale ?*

Torniamo ora a considerare il modello costruito al punto 1. ed effettuiamo le seguenti modifiche:

5. *Cambiare la restrizione di ordine minimo con la restrizione che ogni prodotto venga prodotto almeno in una percentuale minima rispetto a quella totale riportata nella tabella seguente:*

	perc
nastri	0.4
bobine	0.1
placche	0.4

Determinare la soluzione ottima. Se la percentuale di nastri *e* placche *viene fissata a 0.5, come cambia la soluzione ?*

6. *Introdurre una nuova fase di lavorazione, necessaria solo per il prodotto* placche*: il tasso di produzione è di 150 ton/ore e il numero di ore disponibile per questo tipo di lavorazione è pari a 20.*

Si riportano di seguito il file `acciaio.mod` e `acciaio.dat` che implementano questo modello, incluse le modifiche (queste ultime sono riportane nei file commentate, ovvero precedute dal carattere `#`).

`acciaio.mod`

```
# INSIEMI
set PROD;   # prodotti
set FASI;  # fasi di lavorazione

# PARAMETRI
param tasso {PROD,FASI} >= 0; # ton per ora in ogni fase
param disp {FASI} >= 0;       # ore disponibili a settimana
                              # in ogni fase
param profitto {PROD};        # profitto per ton
param ord_min {PROD} >= 0;    # ordini minimi in ton a settimana
param ord {PROD} >= 0;    # ordine in ton a settimana

### Modifica 3.: produzione tot. <= produzione max
#param produzione_massima >=0;

### Modifica 5.: aggiunta di vincoli percentuali sulla produzione
#param perc{PROD};

# VARIABILI
var x {p in PROD} >=0, <= ord[p]; # ton prodotte

### Modifica 1.: ordine minimo da soddisfare
#var x {p in PROD} >= ord_min[p], <= ord[p] # ton prodotte
```

```
# FUNZIONE OBIETTIVO
maximize profitto_totale: sum {p in PROD} profitto[p] * x[p];
              # Obiettivo:  massimizzare il profitto totale

### Modifica 4.: massimizzare la quantita' complessiva prodotta
#maximize produzione_complessiva: sum {p in PROD} x[p];

# VINCOLI
#s.t. tempo {s in FASI}:
#      sum {p in PROD} (1/tasso[p,s]) * x[p] <= disp[s];
              # In ogni fase il numero totale di ore non deve eccedere
   # le ore disponibili
### Modifica 2.:
#    sum {p in PROD} (1/tasso[p,s]) * x[p] = disp[s];

### controllo sul denominatore nullo
s.t. tempo {s in FASI}:
      sum {p in PROD : tasso[p,s]>0 } (1/tasso[p,s]) * x[p] <= disp[s];
              # In ogni fase il numero totale di ore non deve eccedere
   # le ore disponibili

### Modifica 3.: (produzione massima <= 6500)
#s.t. produzione_totale : sum{p in PROD} x[p] <= produzione_massima;

### Modifica 5.: vincoli percentuali sulla produzione
#s.t. vincoli_percentuali{p in PROD} : x[p] >= perc[p] * sum{q in PROD} x[q];
### Modifica 5 bis.:
#s.t. vincolo_perc1 : x["nastri"] = 0.5  * sum{q in PROD} x[q];
#s.t. vincolo_perc2 : x["placche"] = 0.5 * sum{q in PROD} x[q];
```

acciaio.dat

```
set PROD := nastri bobine placche;
set FASI := fase1 fase2;

param tasso:  fase1  fase2 :=
  nastri        200    200
  bobine        200    140
  placche       200    160 ;
```

```
# Modifica 6.: ulteriore fase per le placche
#set FASI := fase1 fase2 fase3;

#param tasso:  fase1  fase2 fase3 :=
#  nastri        200    200    0
#  bobine        200    140    0
#  placche       200    160   150;


param:    profitto  ord   :=
  nastri     25     6000
  bobine     30     4000
  placche    29     3500  ;

#param:    profitto  ord_min  ord   perc:=
#  nastri     25      1000    6000    0.4
#  bobine     30       500    4000    0.1
#  placche    29       750    3500    0.4;

param disp :=
  fase1   35
  fase2   40 ;

#  Modifica 6.: ulteriore fase per le placche
#  param disp :=
#  fase1   35
#  fase2   40
#  fase3   20;


#param produzione_massima := 6500;
```

5.2 UN PROBLEMA DI MISCELAZIONE

Esempio 5.2.1 *Un'azienda produttrice di bibite vuole creare un composto contenente tre tipi di zucchero (di canna, di mais e di barbabietola) per usarla nei suoi prodotti. I fornitori di zucchero vendono delle miscele di questi tre tipi di zucchero e non una varietà pura. Nella tabella che segue vengono riportate le percentuali delle tre tipologie di zucchero presenti nelle miscele vendute dai vari fornitori (indicati con le lettere da A a G) e i prezzi:*

	A	B	C	D	E	F	G
zucchero di canna	10%	10%	20%	30%	40%	20%	60%
zucchero di mais	30%	40%	40%	20%	60%	60%	10%
zucchero di barbabietola	60%	50%	40%	50%	0%	20%	30%
costo $/ton	10	11	12	13	14	12	15

L'azienda vuole ottenere una miscela contenente 52 tonnellate di zucchero di canna, 56 tonnellate di zucchero di mais e 59 tonnellate di zucchero di barbabietola minimizzando il costo d'acquisto.

- *Formulare il problema come modello di Programmazione Lineare.*
- *Scrivere il problema in* AMPL *in forma parametrica e risolverlo (fornire i file* `zucchero1.mod`*,* `zucchero1.dat` *e* `zucchero1.run`*).*

Cosiderare ora le seguenti modifiche:

1. *Per ragioni contrattuali l'azienda deve effettuare un ordine minimo per ogni fornitore:*

	ton
A	1
B	5
C	7
D	10
E	5
F	8
G	10

   Modificare il modello e scrivere i file `zucchero2.mod`*,* `zucchero2.dat` *e aggiornare* `zucchero.run` *per tener conto di questa restrizione. Determinare la soluzione ottima.*

2. *Si supponga ora che l'azienda abbia la possibilità di acquistare direttamente un dolcificante artificiale (che denoteremo con H) da un altro fornitore al prezzo di 12 dollari alla tonnellata. Per misurare la qualità del dolcificante*

l'azienda utilizza un sofisticato macchinario che ne determina la sua bontà. Supponendo che la bontà del dolcificante artificiale è 6 e tenendo conto che le altre miscele hanno tutte una bontà di 9, si modifichi il modello precedente aggiungendo il nuovo dolcificante/miscela e inserendo un vincolo di qualità minima sul prodotto finale pari a 8 (si ipotizzi che la qualità finale dipenda linearmente dalla qualità dei suoi componenti). Chiaramente ora l'azienda intende sempre avere una quantità pari a 167 tonnellate di dolcificante, ma una parte sarà composta dal dolcificante artificiale, mentre l'altra parte sarà composta dai tre tipi di zucchero nelle stesse proporzioni richieste precedentemente. Scrivere i file `zucchero3.mod`, `zucchero3.dat` *e aggiornare* `zucchero.run` *per tener conto di questa nuova situazione e analizzare la soluzione ottima.*

Si riportano di seguito i file che realizzano un'implementazione in AMPL del problema ora esposto.

zucchero1.mod

```
set ZUCCHERO;   # tipi di zucchero
set MISCELE;    # tipo miscela

param percentuale {ZUCCHERO,MISCELE} >= 0, <= 1;
          # percentuale del tipo di zucchero nella miscela

param prezzo {MISCELE} >= 0;
          # prezzo di una tonnellata della miscela

param richiesta {ZUCCHERO} >= 0;
          # richiesta del tipo di zucchero

check {f in MISCELE}: sum {z in ZUCCHERO} percentuale[z,f] = 1;
          # controllo sulla percentuale

var  Qta {f in MISCELE} >= 0;
          # ton acquistate per miscela

minimize Costo: sum {f in MISCELE} prezzo[f] * Qta [f];
       # funzione obiettivo (costo di acquisto) da minimizzare

subject to Richiesta_Finale {z in ZUCCHERO}:
   sum {f in MISCELE} percentuale[z,f] * Qta[f] = richiesta[z];
          # le quantita' finali di zucchero sono fissate
```

zucchero1.dat

```
set ZUCCHERO := canna mais barbabietola;
set MISCELE := A B C D E F G;

param percentuale: A B C D E F G:=
       canna   0.10    0.10    0.20    0.30   0.40   0.2   0.60
        mais   0.30    0.40    0.40    0.20   0.60   0.6   0.10
barbabietola   0.60    0.50    0.40    0.50   0      0.2   0.30;

param:     prezzo :=
    A 10
    B 11
    C 12
    D 13
    E 14
    F 12
    G 15;

param: richiesta:=
    canna         52
    mais          56
    barbabietola  59;
```

Per realizzare i file `zucchero2.mod` e `zucchero2.dat` è sufficiente aggiungere al modello il parametro `acquisto_min` e i vincoli `contratti`. Per fare ciò è sufficiente scrivere i seguenti file:

zucchero2.mod

```
param acquisto_min {MISCELE} >= 0;
     # quantita' minime da acquistare per i vari fornitori

subject to contratti {f in MISCELE}: Qta[f] >= acquisto_min[f];
      # necessario acquistare almeno una certa quantita' da
      # ciascun fornitore
```

zucchero2.dat

```
param:     acquisto_min :=
   A 1
   B 5
```

```
C 7
D 10
E 5
F 8
G 10;
```

Tali file possono essere caricati da AMPL con gli usuali comandi `model` e `data` *dopo* aver precedentemente caricato i file `zucchero1.mod` e `zucchero1.dat`. Le nuove componenti del modello verranno aggiunte al modello già caricato.

Si riportano di seguito i file `zucchero3.mod` e `zucchero3.dat` che realizzano quanto richiesto al punto 2.

zucchero3.mod

```
set ZUCCHERO;   # tipi di zucchero
set MISCELE;  # tipo miscela
set DOLCIFICANTE_ARTIFICIALE;  # dolcificante artificiale: un
            # insieme costiturito da un solo elemento ( 'H' )

set FORNITORI := MISCELE union DOLCIFICANTE_ARTIFICIALE;

param percentuale{ZUCCHERO,MISCELE} >= 0, <= 1;
            # percentuale del tipo di zucchero nella miscela

param prezzo{FORNITORI} >= 0;    # prezzo di una tonnellata
                # (di miscela e di dolcificanti artificiali)

param richiesta{ZUCCHERO} >= 0;  # richiesta del tipo di zucchero

param acquisto_min{FORNITORI} >= 0;
             # quantita' minime da acquistare per i vari fornitori

param quantita_totale:=sum{z in ZUCCHERO} richiesta[z];
             # quantita' totale di dolcificante da ottenere

param bonta{FORNITORI};
             # bonta' di ciascuna miscela e del dolcificante
             # artificiale

param bonta_min;

check{f in MISCELE}: sum{z in ZUCCHERO} percentuale[z,f] = 1;
```

```
             # controllo sulla percentuale

var  Qta{f in FORNITORI} >= 0; # ton acquistate per miscela

var frazione_dol_art{ZUCCHERO} >= 0;
             # frazione del dolcificante articificiale
             # sostitutivo dello zucchero dei vari tipi

minimize Costo: sum{f in FORNITORI} prezzo[f] * Qta [f];
             # minimizzazione del costo di acquisto

s.t. Richiesta_Finale {z in ZUCCHERO}:
     sum {f in MISCELE} percentuale[z,f] * Qta[f] =
                        richiesta[z]-frazione_dol_art[z];
             # le quantita' finali di zucchero sono fissate

s.t. Contratti{f in MISCELE}: Qta[f] >= acquisto_min[f];
             # bisogna acquistare almeno una certa quantita'
             # di miscela

s.t. Vincolo_bonta : sum{f in FORNITORI} bonta[f]*Qta[f] >=
                     bonta_min * sum{f in FORNITORI} Qta[f];
             # vincolo sulla bonta

s.t. Vincolo_quantita : sum{f in FORNITORI} Qta[f]=
                                              quantita_totale;
             # vincolo sulla quantita' totale

s.t. Totale_dol_art : sum{z in ZUCCHERO} frazione_dol_art[z] =
                                                    Qta['H'];
             # la somma delle frazioni di dolcificante
             # artificiale ('H') deve uguagliare la quantita'
             # totale di 'H'
```

zucchero3.dat

```
set ZUCCHERO := canna mais barbabietola;
set MISCELE := A B C D E F G;
```

```
param percentuale: A B C D E F G:=
       canna   0.10    0.10    0.20    0.30   0.40   0.2   0.60
        mais   0.30    0.40    0.40    0.20   0.60   0.6   0.10
barbabietola   0.60    0.50    0.40    0.50   0      0.2   0.30;

param:     prezzo :=
    A 10
    B 11
    C 12
    D 13
    E 14
    F 12
    G 15
    H   12;

param: richiesta:=
    canna         52
    mais          56
    barbabietola  59;

param:     bonta :=
    A   9
    B   9
    C   9
    D   9
    E   9
    F   9
    G   9
    H   6;

param:     acquisto_min :=
    A 1
    B 5
    C 7
    D 10
    E 5
    F 8
    G 10;

param bonta_min := 8;
```

Segue il file `zucchero.run` che richiama e risolve di seguito i modelli ora introdotti.

zucchero.run

```
reset;
model zucchero1.mod;
data zucchero1.dat;

option solver cplex;
solve;

display  _nvars, _ncons, _nobjs;
display _varname, _var;
display  _conname, _con.lb, _con.body, , _con.ub;
display _objname, _obj;

# senza il comando 'reset'
model zucchero2.mod;
data zucchero2.dat;

option solver cplex;
solve;

display  _nvars, _ncons, _nobjs;
display _varname, _var;
display  _conname, _con.lb, _con.body, _con.ub;
display _objname, _obj;

reset;
model zucchero3.mod;
data zucchero3.dat;

option solver cplex;
solve;

display  _nvars, _ncons, _nobjs;
display _varname, _var;
display  _conname, _con.lb, _con.body, _con.ub;
display _objname, _obj;
```

Notare che prima di caricare `zucchero2.mod` e `zucchero2.dat` *non* è stato introdotto il comando `reset` come negli altri casi perché questi due file contengono componenti del modello da aggiungere a quelle già caricate contenute nei file `zucchero1.mod` e `zucchero1.dat`. Si lascia alla cura dello studente l'analisi dei risultati ottenuti dai tre differenti modelli.

5.3 DUALITÀ E ANALISI DELLA SENSITIVITÀ NELLA PROGRAMMAZIONE LINEARE

Come abbiamo visto nel capitolo precedente, attraverso il comando `display` è possibile ottenere informazioni sul problema duale di un problema di Programmazione Lineare senza implementare il problema duale. Si tratta di informazioni molto utili per l'*analisi della sensitività* ed, in particolare, dei *prezzi ombra* e dei *costi ridotti*. Ricordiamo che

- il *prezzo ombra* associato ad un vincolo rappresenta il cambiamento della funzione obiettivo all'ottimo per un incremento unitario del termine noto del vincolo (ovvero della limitazione superiore di quel vincolo) mantenendo tutti gli altri dati del problema immutati;

- il *costo ridotto* associato al vincolo di non negatività di una variabile rappresenta il cambiamento nella funzione obiettivo per un incremento unitario del lower bound della variabile.

5.4 INTERPRETAZIONE DELLA DUALITÀ

Per esaminare come, assegnato un problema di Programmazione Lineare, le informazioni sul corrispondente problema duale sono facilmente ottenibili utilizzando AMPL, facciamo riferimento ad un semplice esempio di modello di pianificazione della produzione

Esempio 5.4.1 *Un'industria produce due tipi di prodotti: un tipo* de luxe *e un tipo* standard. *Per avere un prodotto finito di ciascuno dei due tipi sono necessari due ingredienti grezzi* $\mathbf{I_1}$ *e* $\mathbf{I_2}$ *e la lavorazione su una macchina. La tabella che segue riporta le quantità in Kg di ciascuno degli ingredienti e le ore di lavorazione sulla macchina necessarie per ottenere un prodotto finito di ciascuno dei due tipi.*

	de luxe	*standard*
$\mathbf{I_1}$	3	2
$\mathbf{I_2}$	4	1
ore lavoraz.	2	1

Settimanalmente si hanno a disposizione al piú 1200 Kg dell'ingrediente $\mathbf{I_1}$ *al piú 1000 Kg dell'ingrediente* $\mathbf{I_2}$ *mentre la disponibilità massima settimanale di ore lavorative della macchina è pari a 700. Un prodotto* de luxe *è venduto a 24 Euro e un prodotto* standard *è venduto a 14 Euro. Si vuole pianificare la produzione settimanale in modo da massimizzare il profitto complessivo assumendo che i prodotti siano frazionabili.*

Si tratta di un problema di allocazione ottima di risorse limitate che può essere formulato come problema di Programmazione Lineare nel seguente modo:

$$(\mathrm{P}) \qquad \begin{cases} \max 24x_1 + 14x_2 \\ 3x_1 + 2x_2 \leq 1200 \\ 4x_1 + x_2 \leq 1000 \\ 2x_1 + x_2 \leq 700 \\ x_1 \geq 0, x_2 \geq 0 \end{cases}$$

dove le variabili x_1 e x_2 rappresentano le quantità di prodotti rispettivamente del tipo *de luxe* e del tipo *standard* da fabbricare settimanalmente.

Il problema duale del problema ora formulato è

$$(\mathrm{D}) \qquad \begin{cases} \min 1200u_1 + 1000u_2 + 700u_3 \\ 3u_1 + 4u_2 + 2u_3 \geq 24 \\ 2u_1 + u_2 + u_3 \geq 14 \\ u_1 \geq 0, u_2 \geq 0, u_3 \geq 0. \end{cases}$$

La soluzione ottima del primale è

$$x_1^\star = 160, \qquad x_2^\star = 360$$

e il valore ottimo della funzione obiettivo primale è pari a 8880.

La soluzione ottima del duale è

$$u_1^\star = 6.4, \qquad u_2^\star = 1.2, \qquad u_3^\star = 0$$

e il valore ottimo della funzione obiettivo duale è pari a 8880. Quindi il Teorema della Dualità Forte è verificato.
Scriviamo, ora, le condizioni di complementarità:

$$\begin{aligned} x_1^\star(3u_1^\star + 4u_2^\star + 2u_3^\star - 24) &= 0 \\ x_2^\star(2u_1^\star + u_2^\star + u_3^\star - 14) &= 0 \\ u_1^\star(1200 - 3x_1^\star - 2x_2^\star) &= 0 \\ u_2^\star(1000 - 4x_1^\star - x_2^\star) &= 0 \\ u_3^\star(700 - 2x_1^\star - x_2^\star) &= 0 \end{aligned}$$

Si verifica immediatamente che tali condizioni sono soddisfatte. Si osservi che tutte le equazioni tranne l'ultima sono verificate in quanto si annulla il secondo dei due fattori moltiplicativi. Questo significa, in particolare, che il primo e il secondo vincolo del problema primale sono attivi nella soluzione ottima, cioè verificati all'uguaglianza. L'ultima equazione invece è verificata per il fatto che è nulla all'ottimo la variabile duale $u_3^\star$ mentre il vincolo corrispondente primale (cioè il terzo vincolo del problema primale) non è verificato all'uguaglianza. Infatti in corrispondenza della soluzione ottima il valore ottenuto è $2x_1^\star + x_2^\star = 680$.

Poiché la disponibilità di ore lavorative è pari a 700 ore, si hanno ancora 20 ore disponibili (surplus). Quindi l'industria, per aumentare il profitto, potrebbe acquistare altre quantità di ingredienti grezzi e quindi aumentare la disponibilità settimanale di questi ingredienti e utilizzare le ore di lavorazione ancora rimaste disponibili. Poiché i valori all'ottimo della funzione obiettivo primale e della funzione obiettivo duale coincidono e poiché la funzione obiettivo duale è

$$1200u_1 + 1000u_2 + 700u_3,$$

essendo $u_1^\star = 6.4$, $u_2^\star = 1.2$, $u_3^\star = 0$, l'aumento di 1 Kg della disponibilità di ingrediente $\mathbf{I_1}$ (da 1200 a 1201 Kg) porta ad un incremento di 6.4 Euro nel profitto complessivo. Analogamente per l'ingrediente $\mathbf{I_2}$: un incremento di 1 Kg (da 1000 a 1001 Kg) porta ad un incremento del profitto complessivo di 1.2 Euro. Risulta evidente che il prezzo ombra rappresenta il prezzo massimo unitario al quale risulti conveniente acquistare ulteriore risorsa. Questo è il motivo per cui le variabili duali sono anche chiamate *prezzi ombra* e determinano il *valore marginale* delle risorse.
Ovviamente il fatto che $u_3^\star = 0$ significa che l'aumento della disponibilità di ore lavorative non porta a nessun incremento del profitto, ma questo è ovvio in quanto ore lavorative inutilizzate sono già disponibili.
Nell'ipotesi che, ad esempio, si possa incrementare la disponibilità di una sola delle risorse, naturalmente esaminando i prezzi ombra, si deduce che conviene aumentare la disponibilità dell'ingrediente $\mathbf{I_1}$ che porta ad un maggiore incremento del profitto complessivo.
Si osservi che il fatto che ad un incremento pari a δ nel termine noto del primo vincolo corrisponda un incremento pari a 6.4δ nel valore ottimo della funzione obiettivo, è valido fin tanto che la variabile duale all'ottimo $u_1^\star$ associata al primo vincolo rimane pari al valore 6.4. Si noti che la variazione del termine noto del vincolo corrispondente alla disponibilità dell'ingrediente $\mathbf{I_1}$ porta anche ad un cambiamento nella formulazione del problema duale: infatti un cambiamento nel termine noto di un vincolo primale corrisponde ad un cambiamento in un coefficiente della funzione obiettivo del problema duale. Pertanto c'è la possibilità che se la variazione è ampia, cambi il punto di ottimo del problema duale e quindi, in particolare, cambi il prezzo ombra $u_1^\star$ associato al primo vincolo. In questo caso, naturalmente, la variazione del valore della funzione obiettivo all'ottimo non può essere piú proporzionale al valore 6.4.

Implementiamo ora il problema primale nel linguaggio AMPL. Seguono il file `.mod` e `.dat`.

primale1.mod

```
####    SEZIONE PER LA DICHIARAZIONE DEGLI INSIEMI    ####

set PRODOTTI;      # introduce l'insieme dei prodotti
set RISORSE;   # introduce l'insieme delle risorse

####    SEZIONE PER LA DICHIARAZIONE DEI PARAMETRI    ####

param max_dispo {RISORSE}>=0;
          # array di parametri ciascuno dei quali indica
          # la disponibilita' di ciascuna risorsa.

param profitto {PRODOTTI} >=0;
          # ciascuna sua componente indica il profitto
          # marginale per uno dei prodotti.

param richieste {RISORSE,PRODOTTI}>=0;
                 # l'elemento (i,j) indica la quantita' di
                 # risorsa i-esimo richiesta per produrre
                 # un'unita' di prodotto j-esimo.


####    SEZIONE PER LA DICHIARAZIONE DELLE VARIABILI   ####

var x {j in PRODOTTI} >=0;
         # la variabile x[j] rappresenta il quantitativo
         # di prodotto j-esimo.


####    FUNZIONE OBIETTIVO E VINCOLI   ####

maximize profitto_totale: sum {j in PRODOTTI} profitto[j]*x[j];
                 # rappresenta il profitto totale.

s.t. vincoli {i in RISORSE}:
        sum {j in PRODOTTI} richieste[i,j]*x[j] <= max_dispo[i];
             # vincoli sul massimo quantitativo disponibile delle
             # risorse (ingredienti e di ore lavorative)
```

primale1.dat

```
set PRODOTTI:= de_luxe  standard;
set RISORSE:= I1 I2 ore_lav;

param    max_dispo :=
  I1       1200
  I2       1000
  ore_lav   700 ;

param:         profitto:=
    de_luxe           24
    standard          14    ;

param richieste:  de_luxe   standard :=
      I1            3         2
      I2            4         1
      ore_lav       2         1  ;
```

Risolviamo ora questo problema e analizziamo i risultati che AMPL produce.

```
profitto_totale = 8880

x [*] :=
 de_luxe  160
standard  360

vincoli [*] :=
     I1  6.4
     I2  1.2
ore_lav  0
```

Insieme al punto di ottimo

$$x_1^\star = 160, \qquad x_2^\star = 360$$

e al valore ottimo della funzione obiettivo 8880 abbiamo riportato altre informazioni riguardanti il problema duale associato. In particolare vengono riportati

i valori ottimi della variabili duali associate ai vincoli del problema originario (prezzi ombra). La soluzione ottima del duale risulta quindi

$$u_1^\star = 6.4, \qquad u_2^\star = 1.2, \qquad u_3^\star = 0$$

Per verificare ciò, implementiamo direttamente il problema duale. Seguono il file `.mod` e `.dat` che realizzano una possibile implementazione del problema duale.

duale1.mod

```
####    SEZIONE PER LA DICHIARAZIONE DEGLI INSIEMI    ####
set COLONNE;      # introduce l'insieme delle colonne.
set RIGHE;        # introduce l'insieme delle righe.

####    SEZIONE PER LA DICHIARAZIONE DEI PARAMETRI    ####
param matr_coeff {RIGHE,COLONNE}>=0;
                  # dichiara la matrice dei coefficienti per
                  # il problema duale.

param noti {RIGHE} >=0;
                  # dichiara il vettore dei termini noti.

param coeff_obiett {COLONNE} >=0;
                  # dichiara i coefficienti della funzione
                  # obiettivo.

####    SEZIONE PER LA DICHIARAZIONE DELLE VARIABILI   ####
var u {j in COLONNE} >=0;
                  # dichiara il vettore delle variabili duali.

####    FUNZIONE OBIETTIVO E VINCOLI   ####
minimize funz_obj_duale:
              sum {j in COLONNE} coeff_obiett[j]*u[j];
                  # funzione obiettivo del problema duale.

s.t. vincoli {i in RIGHE}:
           sum {j in COLONNE} matr_coeff[i,j]*u[j] >= noti[i];
                           # vincoli del problema duale.
```

duale1.dat

```
set COLONNE:= COL1  COL2  COL3;
set RIGHE:= RIGA1   RIGA2;

param matr_coeff:    COL1  COL2  COL3:=
         RIGA1         3     4     2
         RIGA2         2     1     1  ;

param:  noti:=
  RIGA1  24
  RIGA2  14  ;

param:   coeff_obiett:=
   COL1      1201
   COL2      1000
   COL3       700     ;
```

Risolvendo questo problema si hanno i seguenti risultati:

```
funz_obj_duale = 8880

u [*] :=
COL1  6.4
COL2  1.2
COL3  0

vincoli [*] :=
RIGA1  160
RIGA2  360
```

Questi risultati confermano che il valore ottimo delle variabili duali è $u_1^\star = 6.4$, $u_2^\star = 1.2$, $u_3^\star = 0$ e che il valore ottimo del problema primale e del problema duale coincide ed è pari a 8880. Naturalmente, in modo del tutto simmetrico, nel risultati relativi al problema duale possiamo avere informazioni relative al duale del problema duale, cioè al problema primale. Infatti è riportato il valore ottimo delle variabili primali che infatti risultano pari a $x_1^\star = 160$, $x_2^\star = 360$.
Abbiamo già detto che le variabili duali costituiscono i *prezzi ombra* e rappresentano l'effetto dei cambiamenti nel termine noto dei vincoli: ad esempio, incrementando di un valore Δ la disponibilità dell'ingrediente $\mathbf{I_1}$ si ottiene un incremento

del profitto pari a 6.4Δ cioè proporzionale a $u_1^\star$ che è la variabile duale associata al primo vincolo. Verifichiamo ciò cambiando il termine noto del vincolo relativo alla disponibilità di ingrediente $\mathbf{I_1}$ dato da $3x_1 + 2x_2 \leq 1200$; in particolare aumentando di una unità il termine noto ($\Delta = 1$), trasformando cioè il vincolo in $3x_1 + 2x_2 \leq 1201$, ci si aspetta un incremento del valore ottimo della funzione obiettivo pari a 6.4 e quindi un valore pari a 8886.4. Per verificare ciò è sufficiente modificare il file `primale1.dat` (lasciando ovviamente immutato il file `primale1.mod`), cambiando il valore del parametro `max_dispo` relativo a $\mathbf{I_1}$ e scrivendo quindi

```
param    max_dispo :=
  I1       1201
  I2       1000
  ore_lav   700 ;
```

Risolvendo di nuovo il problema, si ottengono i risultati di seguito riportati che confermano quanto ci si aspettava.

```
profitto_totale = 8886.4

x [*] :=
 de_luxe  159.8
standard  360.8

vincoli [*] :=
     I1  6.4
     I2  1.2
ore_lav  0
```

5.5 UN ALTRO ESEMPIO SULL'INTERPRETAZIONE DELLE QUANTITÀ DUALI

Esula dallo scopo di queste note la motivazione teorica dettagliata della validità dell'interpretazione data delle variabili duali, a partire da una soluzione ottima, come prezzi ombra rappresentanti i *valori marginali* dei termini noti dei vincoli solo per piccole perturbazioni di questi termini noti ed anche la determinazione dell'intervallo $[\delta_l, \delta_u]$ in cui può variare la perturbazione δ rimanendo valida tale l'interpretazione.
Si riporta tuttavia di seguito un esempio geometrico di questa interpretazione delle variabili duali che dovrebbe chiarire, almeno in un caso particolare, quanto illustrato in precedenza.

Si consideri il seguente problema di Programmazione Lineare

$$\begin{cases} \max 3x_1 + 2x_2 \\ x_1 + x_2 \leq 4 \\ 2x_1 + x_2 \leq 5 \\ -x_1 + 4x_2 \geq 2 \\ x_1 \geq 0, x_2 \geq 0. \end{cases} \tag{5.5.1}$$

Si verifica facilmente che i prezzi ombra associati ai vincoli sono rispettivamente $u_1^\star = 1$, $u_2^\star = 1$ e $u_3^\star = 0$. Questi possono naturalmente essere ricavati scrivendo il problema duale del problema dato

$$\begin{cases} \min 4u_1 + 5u_2 - 2u_3 \\ u_1 + 2u_2 + u_3 \geq 3 \\ u_1 + u_2 - 4u_3 \geq 2 \\ u_1 \geq 0, u_2 \geq 0 \end{cases}$$

e determinandone la soluzione ottima. Geometricamente si può determinare facilmente la soluzione ottima del problema assegnato (primale) (5.5.1) che risulta essere nel punto $P(1, 3)$ a cui corrisponde un valore ottimo della funzione obiettivo pari a 9 (Figura 5.5.1).
Ora, consideriamo il primo vincolo $x_1 + x_2 \leq 4$ e facciamo variare di un valore $\delta = 0.5$ il termine noto che passa da 4 a 4.5; rappresentando geometricamente il nuovo vincolo $x_1 + x_2 \leq 4.5$ si determina la nuova regione ammissibile del problema in cui il segmento $\overline{PQ}$ è mutato nel segmento $\overline{P'Q'}$. Il punto di ottimo del nuovo problema è $P'(0.5, 4)$ e corrispondentemente risulta un incremento del valore della funzione obiettivo proporzionale al valore del prezzo ombra $u_1^\star = 1$ associato al primo vincolo cioè dato da $u_1^\star \delta$, cioè pari a 0.5: infatti il valore ottimo passa da 9 a 9.5. È facile verificare che il cambiamento effettuato nel termine noto non ha fatto variare il punto di ottimo del problema duale, cioè i costi ombra sono rimasti invariati.

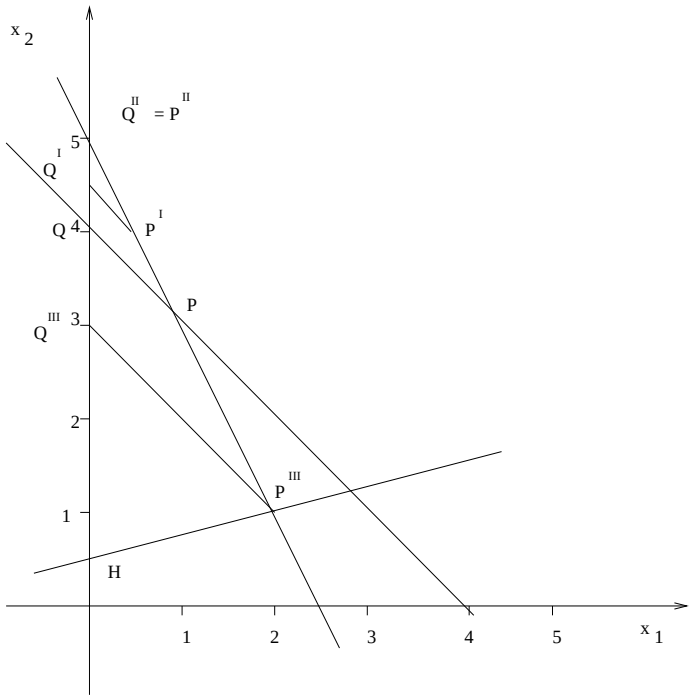


Figura 5.5.1 Rappresentazione geometrica del problema (5.5.1)

Se si continua a rilassare il vincolo considerato facendolo diventare $x_1 + x_2 \leq 5$, considerando la nuova regione ammissibile così ottenuta, si osserva che il segmento $\overline{PQ}$ degenera in un punto $Q'' = P''(0, 5)$ con conseguente incremento proporzionale del valore della funzione obiettivo nel nuovo punto di ottimo dove è pari a 10. (Nemmeno con questa variazione si è avuto un cambiamento nei prezzi ombra.) Come si deduce facilmente dalla rappresentazione geometrica, non ci sarà nessun effetto nell'incrementare ulteriormente il termine noto del primo vincolo, in quanto il punto soluzione continuerà a rimanere $Q'' = P''$. Quindi la limitazione superiore alla variazione del δ in questa caso risulta pari a $\delta_u = 1$.
Se invece si diminuisce il termine noto del vincolo $x_1 + x_2 \leq 4$, la regione ammissibile progressivamente muta fino a che il segmento $\overline{PQ}$ coincide con $\overline{P'''Q'''}$ e questo accade quando il valore il termine noto del vincolo è pari a 3. In corrispondenza di questo valore il punto di ottimo è $P'''(2, 1)$ a cui corrisponde un valore della funzione obiettivo pari a 8. Anche in questo caso la variazione ha lasciato invariato i prezzi ombra.
Fino al raggiugimento del valore 3 nel termine noto del vincolo considerato, ogni decrescita di questo termine noto porta ad un decremento della funzione obiettivo proporzionale ad prezzo ombra associato ($u_1^\star = 1$). Il valore 3 assunto dal termine noto del vincolo è un "valore di soglia" al di sotto del quale la funzione obiettivo decrescerà con una "rapidità" completamente diversa; infatti dalla rappresenta-

zione geometrica si deduce facilmente che la situazione cambia drasticamente al di sotto di questo valore e la rapidità di decrescita non è prevedibile a partire dalla conoscenza della soluzione ottima P. Si verifica infatti che, ad esempio, variando il termine noto considerato dal valore 3 al valore 2.5, il valore ottimo della funzione obiettivo passa dal valore 8 al valore 6.6 e i prezzi ombra sono mutati in $u^\star = (2.8, 0, 0.2)$.
Geometricamente si può verificare ciò osservando come al variare del termine noto del primo vincolo dal valore 3 al valore 5, i corrispondenti punti di ottimo appartengono al segmento $\overline{P''P'''}$ mentre per valori inferiori al valore 3 i punti corrispondenti di ottimo appartengono al segmento $\overline{HP'''}$. Quindi il valore della limitazione inferiore dell'intervallo di variabilità di δ può essere considerata $\delta_l = -1$. Quindi l'intervallo ammesso perché le considerazioni fatte sui prezzi ombra valgano, in questo caso è l'intervallo $[\delta_l, \delta_u] = [-1, 1]$.

Si evince quindi che volendo disegnare il grafico del valore ottimo del profitto al variare del termine noto di un vincolo, i prezzi ombra rappresentano i coefficienti angolari delle rette che rappresentano queste variazioni. Il grafico complessivo del valore ottimo del profitto al variare del termine noto di un vincolo è quindi una curva ***lineare a tratti convessa*** nel caso di problema di massimizzazione, ***concava*** nel caso di problemi di minimizzazione. I punti nodali di questa funzione lineare a tratti rappresentano i punti in cui si verifica il cambio del prezzo ombra.

Queste considerazioni qui dedotte solo in modo geometrico fanno parte della cosiddetta *analisi della sensitività* o analisi post-ottimale che affronta lo studio di come varia la soluzione ottima di un problema al variare oltre che dei termini noti dei vincoli, anche al variare dei coefficienti di costo della funzione obiettivo, oppure aggiungendo nuove variabili o nuovi vincoli. Ovviamente una trattazione rigorosa di queste problematiche esula dallo scopo di queste note e perciò si rimanda ai testi di specifici.

È lasciata alla cura dello studente l'implementazione in AMPL di questo problema e la verifica delle conclusioni ora ottenute modificando direttamente il file dei dati del problema.

Esercizio 5.5.1 *Considerare il modello di Programmazione Lineare proposto nell'Esempio 5.1.1 ed, in particolare, il modello ottenuto al punto* 6. *di quell'esempio. Rispondere ai quesiti che seguono riguardanti l'analisi della sensitività:*

7. *Spiegare come cambia il profitto incrementando progressivamente il tempo massimo disponibile per la fase 1 (da 35 a 36, a 37, a 38, a 39 ore).*

8. *Disegnare il grafico del profitto massimo al variare del tempo disponibile per la fase 1 nell'intervallo* $[35, 39]$.

9. *Estendere tale grafico anche all'intervallo* $[12, 35]$.

10. *Spiegare che cosa accade quando il tempo massimo per la fase 1 viene fissato ad 11.*

6

Modelli di Programmazione Lineare Intera

Come è noto, quando tutte le variabili di un problema di Programmazione Lineare sono vincolate ad assumere valori interi, si parla di Programmazione Lineare Intera. Moltissimi problemi reali possono essere rappresentati da modelli di Programmazione Lineare Intera; tipicamente si tratta di problemi in cui le variabili di decisione rappresentano quantità indivisibili (come il numero di automobili, di persone addette a certe mansioni, etc.) oppure sono problemi caratterizzati dalla necessità di scegliere tra un numero finito di alternative diverse. In quest'ultimo caso, in particolare, si avranno problemi di Programmazione Lineare 0–1, cioè problemi in cui le variabili sono binarie e assumono valore 0 oppure 1. Molto spesso si hanno problemi di Programmazione Lineare Mista Intera, ovvero problemi che presentano alcune variabili continue e altre intere.

6.1 VARIABILI INTERE PER RAPPRESENTARE QUANTITÀ INDIVISIBILI

Un numero molto elevato di problemi reali è caratterizzato dalla indivisibilità del bene da produrre o della risorsa da utilizzare. Di qui la necessità di rappresentare tali problemi attraverso modelli di Programmazione Lineare con variabili intere. Questo tipo di problemi riguardano molte applicazioni reali: dai problemi in ambito industriale come la distribuzione dei beni e il sequenziamento delle attività produttive, ai problemi economici come la gestione ottima di un portafoglio titoli; dai problemi di progettazione ottima ai problemi inerenti la biologia e la fisica delle alte energie.
Esempi di modelli di Programmazione Lineare Intera caratterizzati da variabili di decisione associate a quantità indivisibili sono già stati presi in esame trascurando

il vincolo di interezza sulle variabili; si veda l'Esempio 3.6.2 nel quale è stata presa in esame la produzione di pneumatici che ovviamente rappresentano prodotti non divisibili.

L'implementazione in AMPL di questi problemi di Programmazione Lineare Intera non rappresenta una difficoltà. Infatti, sarà sufficiente definire le variabili in questioni come *variabili intere*. Questo si ottiene nella dichiarazione delle variabili introducendo la parola chiave `integer`. Quindi la dichiarazione di una variabile intera non negativa può essere realizzata nel seguente modo:

```
var x >=0, integer;
```

Se invece una variabile è di tipo binario, ovvero a valori in $\{0, 1\}$, essa sarà dichiarata utilizzando la parola chiave `binary`, ovvero nel seguente modo:

```
var x binary;
```

Il solutore `cplex` può essere utilizzato per risolvere problemi di questo tipo. Ovviamente i metodi risolutivi utilizzati cambiano rispetto alla Programmazione Lineare e saranno basati su tecniche di tipo *"branch and bound"* e altro.

È importante soffermarci sul criterio di arresto utilizzato da `cplex` nel caso della Programmazione Linera Intera. Esso è ovviamente basato sul "gap" utilizzato come "certificato di qualità" di una soluzione ammissibile del problema. Ricordiamo, infatti che, dato un problema di Programmazione Lineare Intera nella forma di minimizzazione,

$$\min c^T x$$
$$x \in S,$$

se $x_0 \in S$ è una soluzione ammissibile del problema e x_* la soluzione ottima, si ha

$$LB \leq c^T x_* \leq c^T x_0,$$

dove LB indica un *lower bound* del valore della soluzione ottima. Il *gap*

$$c^T x_0 - LB$$

"certifica" la qualità di x_0: se il gap è piccolo allora x_0 è una "buona" soluzione del problema; se il gap è nullo, allora x_0 è ottima. Ovviamente si tratta in maniera analoga il caso in cui il problema si presenti in forma di massimizzazione, facendo uso di *upper bound* invece che di lower bound.

Il solutore *cplex* utilizza un criterio di arresto basato su due valori: il *gap assoluto* (*absolute mixed-integer optimality gap tolerance*) e il *gap relativo* (*relative mixed-integer optimality gap tolerance*), rispettivamente:

`absmipgap=r1` (default 0.0) , `mipgap=r2` (default 1.0e-4).

L'algoritmo si arresta se

$$|best\_node - best\_integer| < r1 \qquad \text{o} \qquad \frac{|best\_node - best\_integer|}{1.0 + |best\_node|} < r2,$$

dove *best_integer* è il valore di una soluzione ammissibile ottenuta fino a quella iterazione (ottimo corrente) e *best_node* è il *lower bound* o l'*upper bound* a seconda che il problema sia di minimizzazione o di massimizzazione. In questo modo si garantisce che il gap (assoluto) sia inferiore ad $r1$ oppure che il valore della funzione obiettivo sia entro il 100 $r2$ percento dall'ottimo. È indicato l'utilizzo del gap assoulto nel caso in cui ci si aspetti un valore della funzione obiettivo all'ottimo non troppo grande. Valori validi per $r2$ sono tra 10^{-9} e 1.0. Aumentare i valori di $r1$ e $r2$ permette di accettare soluzioni "meno buone", ma questo, a volte, si rende necessario quando il tempo di CPU per ottenere una soluzione più accurata è molto lungo. Per poter assegnare valori a queste due tolleranze si usa il comando che permette di definire i parametri del solutore `cplex`, e quindi, ad esempio, si può aggiungere nel file `.run` la seguente riga di comando:

```
option cplex_options 'absmipgap=1.0e-1 mipgap=5.00e-2';
```

per assegnare nel criterio di stop un gap assoluto pari 10^{-1} e un gap relativo del 5%.

Le *options* di `cplex` sono molte e si possono trovare elencate nel file `README.cplex.txt` presente nella directory di AMPL. Ne citiamo solo un'altra che riguarda la visualizzazione, ovvero la frequenza con la quale vengono fornite informazioni sul *branch and bound*. Si utilizzano le opzioni

`mipdisplay` (default 0) , `mipinterval` (default 1).

Se `mipdisplay` è impostato a 0 non verrano mai fornite tali informazioni; altrimenti, se `mipdisplay=1` le informazioni vengono visualizzate ogni soluzione ammissibile intera, oppure se `mipdisplay=2` le informazioni vengono visualizzate ogni "`mipinterval`" nodi. Un esempio potrebbe essere il seguente:

```
option cplex_options 'mipdisplay=1 mipinterval=1';
```

Per ogni altro dettaglio sulle optioni di `cplex` si rimanda alla guida [IBM ILOG AMPL, 2010].
È lasciata alla cura dello studente riconsiderare alcuni dei modelli di Programmazione Lineare già esaminati nei quali si era trascurato il vincolo di interezza delle variabili, aggiungendo tale vincolo e analizzando la soluzione ottenuta.

6.2 VARIABILI BINARIE PER RAPPRESENTARE SCELTE ALTERNATIVE

In molte applicazioni reali, i problemi si presentano come problemi di decisione nei quali si deve modellare il fatto che un certo evento possa verificarsi oppure no. La natura binaria del problema suggerisce l'idea di modellare questa dicotomia per mezzo di un variabile binaria $\delta \in \{0,1\}$; si porrà $\delta = 1$ se l'evento si verifica e $\delta = 0$ altrimenti. Esempi tipici sono i *Problemi di Knapsack binario* e i *Problemi di assegnamento*. Vediamo un esempio di un problema di assegnamento.

Esempio 6.2.1 *Una compagnia aerea cerca di pianificare le proprie linee aeree creando un areoporto centrale e cercando di avere un elevato numero di voli in arrivo in questo areoporto in una certa fascia oraria ed un elevato numero di partenze nella fascia oraria immediatamente successiva. Questo permette ai passeggeri di avere un elevato numero di combinazioni tra città di partenza e città di destinazione con una sola coincidenza e al più un cambio di aereo nell'areoporto centrale. Il fine è quello di creare una tale struttura in modo da minimizzare i cambi di aerei e quindi il movimento di bagagli nell'areoporto centrale. Supponiamo che la compagnia aerea abbia cinque voli che arrivano tra le 8 e le 8.30 nell'areoporto centrale e che poi gli stessi aerei partono per altre diverse destinazioni tra le 8.40 e le 9.20. La tabella che segue riporta il numero medio di passeggeri che arrivano con uno dei voli in arrivo* **A1**, **A2**, **A3**, **A4**, **A5** *e che rimarrebbero a bordo dell'aeromobile se questo fosse assegnato rispettivamente ai voli in partenza* **P1**, **P2**, **P3**, **P4**, **P5**

	P1	**P2**	**P3**	**P4**	**P5**
A1	15	20	8	16	12
A2	17	9	15	25	12
A3	12	32	16	9	20
A4	-	15	9	7	30
A5	-	-	35	10	18

Il volo **A4** *arriva troppo tardi e non permette di prendere il volo in partenza* **P1**; *analogamente il volo* **A5** *non permette coincidenze con i voli in partenza* **P1** *e* **P2**. *Supponendo che tutti gli aerei sono identici, il problema consiste nell'assegnare ciascun aereo in arrivo ad uno dei voli in partenza in modo da massimizzare il numero delle persone che non devono cambiare aeromobile. Costuire un modello di Programmazione Lineare a variabili* $\{0,1\}$ *per risolvere il problema (file* `aeroporto.mod` *e* `aeroporto.dat`*).*

Modificare il file `aeroporto.mod` *eliminando la dichiarazione di variabili* binarie *(*$x \in \{0,1\}$*) e sostituendola con la dichiarazione di variabili continue in* $[0,1]$. *Risolvere questo problema rilassato e confrontare la soluzione ora ottenuta con quella ottenuta con il modello a variabili* $\{0,1\}$ *e giustificare le conclusioni tratte.*

Per completezza riportiamo brevemente la formulazione algebrica e poi i file in AMPL.

– *Variabili.* Introduciamo le variabili di decisione x_{ij} definite come segue

$$x_{ij} = \begin{cases} 1 & \text{se l'aereo del volo } \mathbf{Ai} \text{ è assegnato al volo } \mathbf{Pj} \\ 0 & \text{altrimenti.} \end{cases}$$

– *Funzione obiettivo.* Definiamo come funzione obiettivo il numero di passeggeri che non devono cambiare aereo:

$$\begin{aligned} & 15x_{11} + 20x_{12} + 8x_{13} + 16x_{14} + 12x_{15} + \\ + \; & 17x_{21} + 9x_{22} + 15x_{23} + 25x_{24} + 12x_{25} + \\ + \; & 12x_{31} + 32x_{32} + 16x_{33} + 9x_{34} + 20x_{35} + \\ + \; & 15x_{42} + 9x_{43} + 7x_{44} + 30x_{45} + \\ + \; & 35x_{53} + 10x_{54} + 18x_{55}. \end{aligned}$$

Tale funzione deve naturalmente essere massimizzata.

– *Vincoli.* I vincoli saranno

$$\begin{aligned} x_{11} + x_{12} + x_{13} + x_{14} + x_{15} &= 1 \\ x_{21} + x_{22} + x_{23} + x_{24} + x_{25} &= 1 \\ x_{31} + x_{32} + x_{33} + x_{34} + x_{35} &= 1 \\ x_{42} + x_{43} + x_{44} + x_{45} &= 1 \\ x_{53} + x_{54} + x_{55} &= 1 \\ x_{11} + x_{21} + x_{31} &= 1 \\ x_{12} + x_{22} + x_{32} + x_{42} &= 1 \\ x_{13} + x_{23} + x_{33} + x_{43} + x_{53} &= 1 \\ x_{14} + x_{24} + x_{34} + x_{44} + x_{54} &= 1 \\ x_{15} + x_{25} + x_{35} + x_{45} + x_{55} &= 1. \end{aligned}$$

aeroporto.mod

```
set ARRIVI;
set PARTENZE;

param passeggeri{ARRIVI,PARTENZE}>=0;

var x{ARRIVI,PARTENZE} binary;
#var x{ARRIVI,PARTENZE} >=0, <= 1;
```

```
maximize passeggeri_che_non_cambiano_aereo :
    sum{i in ARRIVI, j in PARTENZE} passeggeri[i,j]*x[i,j];

s.t. vincoliA{i in ARRIVI} : sum{j in PARTENZE} x[i,j] = 1;
s.t. vincoliP{j in PARTENZE}  : sum{i in ARRIVI} x[i,j] = 1;
```

aeroporto.dat

```
set ARRIVI := A1, A2, A3, A4, A5;
set PARTENZE := P1, P2, P3, P4, P5;

param passeggeri : P1   P2   P3   P4   P5 :=
                A1  15   20   8    16   12
                A2  17   9    15   25   12
                A3  12   32   16   9    20
                A4  0    15   9    7    30
                A5  0    0    35   10   18;

# fissare le 3 variabili a zero

fix x["A4","P1"]:=0;
fix x["A5","P1"]:=0;
fix x["A5","P2"]:=0;
```

6.3 VARIABILI BINARIE COME VARIABILI INDICATRICI

Un altro classico uso di variabili $0-1$, consiste nell'indicare le relazioni di dipendenza tra alcune grandezze di un problema. In questo caso, le variabili binarie vengono utilizzate come *variabili indicatrici*. Per completezza, richiamiamo di seguito questo uso delle variabili binarie.

Supponiamo che la variabile $x \geq 0$ rappresenti una grandezza del problema e di conoscere un limite superiore di tale variabile, cioè un valore M positivo maggiore del più grande valore che può assumere la x. Allora, data una variabile $\delta \in \{0,1\}$ può risultare molto utile modellare attraverso un vincolo lineare l'implicazione logica

$$x > 0 \Rightarrow \delta = 1 \tag{6.3.1}$$

(che è equivalente alla condizione $\delta = 0 \Rightarrow x = 0$). Si vede facilmente che tale implicazione può essere modellata con il vincolo lineare data dalla seguente

diseguaglianza lineare

$$x - M\delta \leq 0.$$

In altri casi, può essere necessario modellare la condizione

$$\delta = 1 \Rightarrow x > 0 \tag{6.3.2}$$

(che è equivalente alla condizione $x_i = 0 \Rightarrow \delta = 0$, poiché, per ipotesi, $x \geq 0$).
La condizione logica (6.3.2) non si può facilmente rappresentare con un vincolo. Supponiamo, ad esempio, che in un problema di miscelazione una variabile x rappresenti la quantità di un ingrediente da includere nella miscela e quindi si ha $x \geq 0$; si può usare una variabile indicatrice $\delta \in \{0, 1\}$ per distinguere tra il caso in cui $x = 0$ e $x > 0$. La condizione logica (6.3.2) afferma che se $\delta = 1$ allora l'ingrediente rappresentato da x deve apparire nella miscela, ma non fornisce nessuna indicazione sulla quantità dell'ingrediente. In realtà, è più verosimile imporre una condizione logica del tipo

$$\delta = 1 \Rightarrow x \geq \epsilon > 0 \tag{6.3.3}$$

cioè se $\delta = 1$ allora la variabile x assume un valore almeno pari ad ϵ.
La (6.3.3) è rappresentabile dal vincolo

$$x - \epsilon\delta \geq 0. \tag{6.3.4}$$

Riepilogando possiamo considerare il seguente schema: se x è una variabile non negativa e $\delta \in \{0, 1\}$ ed inoltre $x_i < M$ e $\epsilon > 0$, allora

$$x - M\delta \leq 0 \quad \Leftrightarrow \quad \begin{cases} x > 0 \Rightarrow \delta = 1 \\ \delta = 0 \Rightarrow x_i = 0 \end{cases}$$

$$x - \epsilon\delta \geq 0 \quad \Leftrightarrow \quad \begin{cases} \delta = 1 \Rightarrow x \geq \epsilon \\ x = 0 \Rightarrow \delta = 0. \end{cases}$$

Analizziamo, ora, un esempio di miscelazione in cui applichiamo quanto appena esposto.

6.3.1 Un esempio: il problema della dieta

Esempio 6.3.1 *Sia data la seguente tavola di valori nutrizionali che riporta il tipo di alimento, il costo unitario, le unità di sostanze (proteine, carboidrati, grassi, vitamine, calcio) per unità di alimento*

	costo	**prot.**	**carb.**	**grassi**	**vitam.**	**calcio**
1	0.15	0	7	1	1	0
2	0.23	1	0	3	1	4
3	0.79	5	0	4	0	1
4	0.47	2	2	1	3	0
5	0.52	0	3	0	2	1

Formulare un problema di Programmazione Lineare Intera che permetta di trovare una dieta di costo minimo sapendo che si devono assumere almeno 3 unità di proteine, 10 unità di carboidrati, 2 unità di grasso, 3 unità di vitamine e 2 unità di calcio e sapendo che se è presente l'alimento 1 la dieta non può contenere l'alimento 5.

Formulazione.

È un classico problema di miscelazione; le quantità di alimenti presenti nella dieta si suppongono frazionabili. A causa della presenza di una condizione logica, è necessario utilizzare, in aggiunta alle variabili del problema, una variabile $0-1$ per modellarla cioè per esprimere con un vincolo il legame tra la presenza nella dieta dell'alimento 1 e dell'alimento 5.

– *Variabili di decisione.* Introduciamo come variabili del problema le unità di alimenti presenti nella dieta, x_i con $i = 1, \ldots, 5$. Inoltre, introduciamo la variabile booleana $\delta \in \{0, 1\}$.

– *Vincoli.* Si hanno i seguenti vincoli:

- Vincoli di qualità: la dieta deve contenere alcuni valori minimi di sostanze nutrizionali; dalla tabella si ottiene che deve essere

$$x_2 + 5x_3 + 2x_4 \geq 3$$

$$7x_1 + 2x_4 + 3x_5 \geq 10$$

$$x_1 + 3x_2 + 4x_3 + x_4 \geq 2$$

$$x_1 + x_2 + 3x_4 + 2x_5 \geq 3$$

$$4x_2 + x_3 + x_5 \geq 2$$

- Vincolo logico: se nella dieta è presente l'alimento 1 allora non deve esserci l'alimento 5. Vogliamo quindi definire dei vincoli che consentano di esprimere le seguenti condizioni logiche

$$x_1 > 0 \quad \Rightarrow \quad \delta = 1$$
$$\delta = 1 \quad \Rightarrow \quad x_5 = 0$$

Secondo quanto descritto, ciò può essere modellato introducendo i vincoli

$$x_1 - M\delta \leq 0$$

$$x_5 - M(1-\delta) \leq 0$$

dove M è un numero positivo maggiore del più grande valore che possono assumere le variabili.

- Vincoli di non negatività: Si tratta di quantità di alimenti, e quindi deve essere
$$x_i \geq 0 \qquad i = 1, \ldots, 5.$$

– *Funzione obiettivo.* È il costo da minimizzare ed è data da

$$0.15x_1 + 0.23x_2 + 0.79x_3 + 0.47x_4 + 0.52x_5.$$

Complessivamente la formulazione di PLI per questo problema può essere scritta

$$\begin{cases} \min\left(0.15x_1 + 0.23x_2 + 0.79x_3 + 0.47x_4 + 0.52x_5\right) \\ x_2 + 5x_3 + 2x_4 \geq 3 \\ 7x_1 + 2x_4 + 3x_5 \geq 10 \\ x_1 + 3x_2 + 4x_3 + x_4 \geq 2 \\ x_1 + x_2 + 3x_4 + 2x_5 \geq 3 \\ 4x_2 + x_3 + x_5 \geq 2 \\ x_1 - M\delta \leq 0 \\ x_5 - M(1-\delta) \leq 0 \\ x_i \geq 0 \quad i = 1, \ldots, 5 \\ \delta \in \{0, 1\} \end{cases}$$

dove M è una quantità che supera il valore che possono assumere le variabili x_i.

Si riportano di seguito i file `dieta.mod` e `dieta.dat` che realizzano un implementazione AMPL di questo problema.

`dieta.mod`

```
set ALIMENTI;
set SOSTANZE;

param contenuti{ALIMENTI,SOSTANZE}>=0;
param costi{ALIMENTI}>=0;
param contenuti_min{SOSTANZE};
param BigM>0;
```

```
var x{ALIMENTI}>=0;
var d binary;

minimize costo : sum{i in ALIMENTI} costi[i]*x[i];

s.t. vincoli_qualita{j in SOSTANZE} :
   sum{i in ALIMENTI} contenuti[i,j]*x[i] >= contenuti_min[j];

s.t. vincolo_logico1 : x["A1"] - BigM * d <=0;
s.t. vincolo_logico2 : x["A4"] - (1-d)*BigM <=0;
```

dieta.dat

```
set ALIMENTI:= A1 A2 A3 A4 A5;
set SOSTANZE:= prot carb grassi vitam calcio;

param BigM := 10000;

param costi :=
A1  0.15
A2  0.23
A3  0.79
A4  0.47
A5  0.52;

param contenuti_min :=
prot 3
carb 10
grassi 2
vitam 3
calcio 2;


param contenuti : prot carb grassi vitam calcio :=
A1  0 7 1 1 0
A2  1 0 3 1 4
A3  5 0 4 0 1
A4  2 2 1 3 0
A5  0 3 0 2 1;
```

6.4 PROBLEMI DI COSTO FISSO

Un altro esempio di applicazione di variabili indicatrici è il *problema del costo fisso* che ora richiamiamo.
Nei modelli di PL la funzione obiettivo è una funzione lineare nelle variabili di decisione che, di solito, rappresentano livelli di attività. Questa ipotesi, in molti problemi pratici, non è verosimile: può infatti accadere che il costo di un'attività abbia un costo iniziale (set-up), ad esempio l'acquisto di un macchinario, che esiste solo se quell'attività è svolta a livello non nullo.
In riferimento ad un'applicazione industriale, indichiamo con c il costo della manifattura per unità di prodotto, con $f \geq 0$ il costo di set-up (costo fisso) e con $x \geq 0$ la quantità di prodotto da fabbricare.
Quindi se $x = 0$ il costo totale è ovviamente nullo; se $x > 0$ allora il costo totale è dato da $cx + f$. Quindi la funzione obiettivo è data dall'espressione

$$f(x) = \begin{cases} cx + f & \text{se} \quad x > 0 \\ 0 & \text{se} \quad x = 0. \end{cases}$$

Tale funzione ha una discontinuità nell'origine e quindi non è lineare. Per formulare questo problema in termini di Programmazione Lineare, introduciamo una variabile indicatrice $\delta \in \{0, 1\}$ tale che, se il prodotto rappresentato dalla x è fabbricato in una qualsiasi quantità allora $\delta = 1$; se il prodotto non è fabbricato allora $\delta = 0$. Dobbiamo, quindi modellare con un vincolo le condizioni logiche

$$x > 0 \Rightarrow \delta = 1 \tag{6.4.1}$$

$$x = 0 \Rightarrow \delta = 0. \tag{6.4.2}$$

L'implicazione (6.4.1) si realizza introducendo il vincolo

$$x - M\delta \leq 0$$

dove M è un numero positivo maggiore del più grande valore che può assumere la x. Per realizzare l'implicazione (6.4.2), si dovrebbe introdurre un vincolo del tipo $x - \epsilon\delta \geq 0$ con $\epsilon > 0$; in realtà, ciò non è necessario perché la condizione (6.4.2) discende direttamente dal fatto che ci troviamo in un problema di minimizzazione. Infatti, il problema può essere formulato come

$$\min (cx + f\delta)$$

con vincolo aggiuntivo

$$x - M\delta \leq 0$$

con $x \geq 0$ e $\delta \in \{0, 1\}$ e quindi dalla struttura della funzione obiettivo discende immediatamente che se $x = 0$ allora, poiché si tratta di un problema di minimo,

all'ottimo deve essere $\delta = 0$, essendo $f \geq 0$. Quindi non è necessario introdurre nella formulazione la condizione logica (6.4.2).

Si può facilmente generalizzare il problema del costo fisso al caso di n attività.

Vediamo ora un esempio di *problema di costo fisso.*

Esempio 6.4.1 *In una centrale elettrica sono a disposizione tre generatori e ogni giorno si deve decidere quali usare di giorno e quali di notte per assicurare una produzione di almeno 4000 megawatts di giorno e di almeno 2800 megawatts di notte. L'uso di un generatore comporta la presenza di personale tecnico che sorvegli il suo funzionamento; tale personale viene retribuito in maniera diversa tra il giorno e la notte e a seconda del tipo di generatore; tali costi di attivazione sono riportati nella tabella che segue (in euro) insieme al costo (in euro) per ogni megawatt prodotta e alla massima capacità di produzione in megawatts per ogni singolo periodo (giorno/notte).*

	Costo attivazione		**Costo per**	**Capacità**
	giorno	**notte**	**megawatt**	**max**
Generatore A	750	1000	3	2000
Generatore B	600	900	5	1700
Generatore C	800	1100	6	2500

Formulare un modello di PLI che permetta di rappresentare il problema in analisi.

È un problema di costo fisso e può essere formulato in termini di Programmazione Lineare Intera come appena descritto in generale. Per brevità di notazione, chiameremo 1^o periodo il giorno e 2^o periodo la notte.

– *Variabili.* Indichiamo con x_{A_i}, x_{B_i} e x_{C_i}, $i = 1, 2$, i megawatts generati rispettivamente dai generatori A, B e C nel periodo i. Inoltre, per ottenere una formulazione lineare, è necessario introdurre sei variabili $0 - 1$, δ_{A_i}, δ_{B_i} e δ_{C_i}, $i = 1, 2$, definite come segue :

$$\delta_{A_i} = \begin{cases} 1 & \text{se il generatore A è attivato nell'}i\text{-esimo periodo} \\ 0 & \text{se nell'}i\text{-esimo periodo il generatore A non è attivato} \end{cases} \qquad i = 1, 2.$$

Analoga è la definizione per le altre variabili δ_{B_i} e δ_{C_i}, $i = 1, 2$.

– *Funzione obiettivo.* La funzione obiettivo da minimizzare può esser scritta

$$3x_{A_1} + 3x_{A_2} + 5x_{B_1} + 5x_{B_2} + 6x_{C_1} + 6x_{C_2} + 750\delta_{A_1} +$$

$$+1000\delta_{A_2} + 600\delta_{B_1} + 900\delta_{B_2} + 800\delta_{C_1} + 1100\delta_{C_2}.$$

– *Vincoli.* Si devono considerare i vincoli sulla richiesta cioè

$$x_{A_1} + x_{B_1} + x_{C_1} \geq 4000$$

$$x_{A_2} + x_{B_2} + x_{C_2} \geq 2800.$$

Inoltre, per quanto esposto nel caso generale si devono considerare i vincoli

$$x_{A_i} - 2000\delta_{A_i} \leq 0 \qquad i = 1, 2$$

$$x_{B_i} - 1700\delta_{B_i} \leq 0 \qquad i = 1, 2$$

$$x_{C_i} - 2500\delta_{C_i} \leq 0 \qquad i = 1, 2.$$

Quindi la formulazione complessiva può essere scritta

$$\begin{cases} \min\Big(3x_{A_1} + 3x_{A_2} + 5x_{B_1} + 5x_{B_2} + 6x_{C_1} + 6x_{C_2} + \\ \qquad +750\delta_{A_1} + 1000\delta_{A_2} + 600\delta_{B_1} + 900\delta_{B_2} + 800\delta_{C_1} + 1100\delta_{C_2}\Big) \\ x_{A_1} + x_{B_1} + x_{C_1} \geq 4000 \\ x_{A_2} + x_{B_2} + x_{C_2} \geq 2800 \\ x_{A_1} - 2000\delta_{A_1} \leq 0 \\ x_{B_1} - 1700\delta_{B_1} \leq 0 \\ x_{C_1} - 2500\delta_{C_1} \leq 0 \\ x_{A_2} - 2000\delta_{A_2} \leq 0 \\ x_{B_2} - 1700\delta_{B_2} \leq 0 \\ x_{C_2} - 2500\delta_{C_2} \leq 0 \\ x_{A_i} \geq 0, x_{B_i} \geq 0, x_{C_i} \geq 0, \qquad i = 1, 2 \\ \delta_{A_i} \in \{0, 1\}, \delta_{B_i} \in \{0, 1\}, \delta_{C_i} \in \{0, 1\} \qquad i = 1, 2. \end{cases}$$

Si riportano di seguito i file `centrale.mod` e `centrale.dat` che rapprentano una implementazione AMPL del problema in esame.

`centrale.mod`

```
set GENERATORI;
set PERIODI;

param costo_fisso{GENERATORI,PERIODI}>=0;
param costo_unitario{GENERATORI}>=0;
param capacita_max{GENERATORI};
param produzione_min{PERIODI};
param BigM = 10^6;

var x{GENERATORI,PERIODI} >=0;
var d{GENERATORI,PERIODI} binary;

minimize costo_totale : sum{i in GENERATORI, j in PERIODI}
```

```
        (costo_unitario[i]*x[i,j]+costo_fisso[i,j]*d[i,j]);

s.t. produzione_minima{j in PERIODI} : sum{i in GENERATORI}
          x[i,j]>= produzione_min[j];

#s.t. vincoli_capacita{i in GENERATORI, j in PERIODI} :
#     x[i,j] - capacita_max[i]<=0;
#
#s.t. vincoli_logici{i in GENERATORI, j in PERIODI} :
#     x[i,j] - BigM*d[i,j] <=0;

# oppure
s.t. vincoli_logici{i in GENERATORI, j in PERIODI} :
     x[i,j] - capacita_max[i]*d[i,j] <=0;
```

centrale.dat

```
set GENERATORI := A, B, C;
set PERIODI := giorno notte;

param : costo_unitario    capacita_max :=
  A       3                   2000
  B       5                   1700
  C       6                   2500;

param produzione_min :=
   giorno     4000
   notte      2800;

param costo_fisso : giorno    notte :=
           A          750      1000
           B          600       900
           C          800      1100;
```

6.5 PROBLEMI DI LOCALIZZAZIONE DI IMPIANTI

Esaminiamo un'altra importante classe di problemi che possono essere formulati come problemi di Programmazione Lineare Intera facendo uso delle variabili indicatrici: i cosiddetti *problemi di localizzazione*. Si tratta di problemi che nascono nell'ambito della pianificazione industriale che possono essere schematizzati nel seguente modo: sono date n aree $\mathbf{A}_1, \mathbf{A}_2, \ldots, \mathbf{A}_n$, distribuite in un territorio. In ciascuna di esse è possibile costruire una fabbrica che produce merce. Per ciascuna area $\mathbf{A_i}$, $i = 1, \ldots n$ è nota la massima capacità produttiva p_i, $i = 1, \ldots, n$ che una fabbrica avrebbe se fosse localizzata in $\mathbf{A_i}$. Sia inoltre f_i il costo fisso di costruzione della fabbrica nell'area $\mathbf{A_i}$. Sono inoltre dati m siti $\mathbf{C}_1, \mathbf{C}_2, \ldots, \mathbf{C}_m$, ove risiedono clienti ai quali deve essere trasportate la merce prodotta. Per ciascun sito $\mathbf{C}_j$ è assegnato un quantitativo r_j, $j = 1, \ldots, m$, di una data merce richiesta presso il sito $\mathbf{C}_j$. Tale richiesta deve essere soddisfatta esattamente. Per soddisfare questa richiesta possono essere costruite $q \leq n$ fabbriche che producono la merce. Esistono altri costi fissi dovuti alla eventuale costruzione di una strada dall'area $\mathbf{A}_i$ al sito $\mathbf{C}_j$, per ogni $i = 1, \ldots, n$ e $j = 1, \ldots m$; indicheremo questi costi fissi con f_{ij}. Siano inoltre c_{ij} il costo necessario per trasportare una unità di merce dalla fabbrica costruita nell'area $\mathbf{A}_i$ al sito $\mathbf{C}_j$ e M_{ij} il quantitativo massimo di merce trasportabile. Il problema consiste nel determinare quante fabbriche e su quali aree costruirle, insieme a quali strade di collegamento costruire, in modo da soddisfare le richieste di i siti minimizzando i costi di costruzione delle fabbriche, delle strade di collegamento e il costo del trasporto della merce una volta che le costruzioni delle fabbriche sono state ultimate determinando al tempo stesso il piano per il trasporto della merce per soddisfare tutte le richieste. Questo problema può essere formulato come problema di Programmazione Lineare Intera nel seguente modo: si introducono le seguenti variabili

$$\delta_i = \begin{cases} 1 & \text{se una fabbrica è costruita sull'area } \mathbf{A}_i \\ 0 & \text{altrimenti;} \end{cases}$$

$$y_{ij} = \begin{cases} 1 & \text{se una strada è costruita da } \mathbf{A}_i \text{ a } \mathbf{C}_j \\ 0 & \text{altrimenti.} \end{cases}$$

Si introducono inoltre le variabili x_{ij} che rappresentano la quantità di merce trasportata dalla fabbrica costruita nell'area $\mathbf{A}_i$ al sito $\mathbf{C}_j$.
I vincoli sono innanzitutto i vincoli di richiesta

$$\sum_{i=1}^{n} x_{ij} = r_j \quad \text{per ogni} \quad j = 1, \ldots, m.$$

Inoltre per ogni $i = 1, \ldots, m$, si vuole che se $\sum_{j=1}^{m} x_{ij} > 0$ allora $\delta_i = 1$. Questa implicazione si realizza con i vincoli

$$\sum_{j=1}^{m} x_{ij} - p_i \delta_i \leq 0 \qquad i = 1, \ldots, n.$$

Ragionando analogamente si ottengono i vincoli

$$x_{ij} - M_{ij}y_{ij} \leq 0 \qquad i = 1, \dots n, \quad j = 1, \dots m.$$

Infine dovrà essere

$$\sum_{i=1}^{n} \delta_i \leq q.$$

Si devono poi esplicitare i vincoli $x_{ij} \geq 0$ e $\delta_i \in \{0, 1\}$, $y_{ij} \in \{0, 1\}$, $i = 1, \dots n$, $j = 1, \dots m$.
La funzione obiettivo si può quindi scrivere

$$\sum_{i=1}^{n}\sum_{j=1}^{m} c_{ij}x_{ij} + \sum_{i=1}^{n} f_i\delta_i + \sum_{i=1}^{n}\sum_{j=1}^{m} f_{ij}y_{ij}.$$

Esaminiamo ora un semplice esempio.

Esempio 6.5.1 *Una compagnia di distribuzione deve rifornire i suoi clienti* $\mathbf{C_1}$, $\mathbf{C_2}$, $\mathbf{C_3}$, $\mathbf{C_4}$ *e* $\mathbf{C_5}$ *che sono dislocati in località diverse di una regione. Per ottimizzare il rifornimento la compagnia vuole costruire un numero di depositi non superiore a due disponendo di tre possibili zone dove costruirli. A seconda della zona in cui vengono costruiti, i tre possibili depositi hanno un costo di costruzione e una capacità massima diversi. La tabella che segue riporta questi costi in migliaia di euro e queste capacità in tonnellate.*

	Costo costruzione	**Capacità massima**
Deposito 1	10000	180
Deposito 2	15000	230
Deposito 3	13000	500

Il quantitativo di merce (in tonnellate) richiesto da ciascun cliente è riportato nella tabella che segue insieme ai costi (in migliaia di euro) del trasporto di una unità di merce da ciascuno dei possibili depositi a ciascun cliente.

	$\mathbf{C_1}$	$\mathbf{C_2}$	$\mathbf{C_3}$	$\mathbf{C_4}$	$\mathbf{C_5}$
Richiesta	91	170	135	153	110
Deposito 1	15	13	27	9	7
Deposito 2	12	21	34	21	3
Deposito 3	7	10	2	17	12

Costruire un modello lineare che rappresenti il problema in analisi per soddisfare esattamente la richiesta minimizzando il costo complessivo trascurando la possibilità di costruire ulteriori collegamenti rispetto a quelli esistenti e supponendo che non ci siano limitazioni sulle quantità massime di merci trasportabili.

Anche in questo caso riportiamo di seguito i file del modello e dei dati che realizzano una implementazione AMPL del problema in analisi.

localizzazione.mod

```
set CLIENTI;
set ZONE;

param costo_costruzione{ZONE}>=0;
param capacita_max{ZONE}>=0;
param richieste{CLIENTI}>=0;
param costi_trasp{ZONE,CLIENTI}>=0;

var x{ZONE,CLIENTI}>=0;
var d{ZONE} binary;

minimize costo_totale : sum{i in ZONE, j in CLIENTI}
 costi_trasp[i,j]*x[i,j]+sum{i in ZONE}costo_costruzione[i]*d[i];

s.t. vincoli_domanda{j in CLIENTI} :
                      sum{i in ZONE} x[i,j]=richieste[j];
s.t. vincoli_logici{i in ZONE} : sum{j in CLIENTI}
                      x[i,j] - capacita_max[i]*d[i]<=0;
s.t. max_depositi : sum{i in ZONE} d[i] <= 2;
```

localizzazione.dat

```
set CLIENTI:= C1 C2 C3 C4 C5;
set ZONE := dep1 dep2 dep3;

param : costo_costruzione capacita_max :=
dep1 10000  180
dep2 15000  230
dep3 13000  500;

param richieste :=
C1  91
C2  170
C3  135
C4  153
C5  110;

param costi_trasp : C1   C2  C3  C4  C5  :=
dep1  15 13 27 9 7
dep2  12 21 34 21 3
```

```
dep3  7  10  2 17 12;
```

7

Modelli di Programmazione Non lineare

7.1 CENNI AI MODELLI DI PROGRAMMAZIONE NON LINEARE

La trattazione dei modelli di Programmazione Non Lineare esula dai contenuti di questo corso perché tale argomento verrà trattato nel corso di *Ottimizzazione* della Laurea Magistrale. Tuttavia se ne riporta un esempio, per mettere in luce le problematiche inerenti ai modelli di Programmazione Non Lineare.

Esempio 7.1.1 *Un'industria deve costruire un serbatoio di forma cilindrica per contenere grandi quantitativi di un liquido che verrà poi distribuito in piccole confezioni pronte per la vendita al minuto. Tale serbatoio deve essere posto in un magazzino appoggiato su una delle basi. Tale magazzino è a pianta rettangolare di dimensioni metri* 20×10 *ed ha un tetto spiovente lungo il lato di 10 metri, che ha altezza massima di metri* 5 *e altezza minima di metri* 3*. Per costruire questo serbatoio deve essere usato del materiale plastico sottile flessibile che può essere tagliato, modellato e incollato saldamente. Sapendo che si dispone di non più di* 200 m^2 *di tale materiale plastico si costruisca un modello che permetta di determinare le dimensioni del serbatoio (raggio di base ed altezza) in modo da massimizzare la quantità di liquido che può esservi contenuto.*

La formulazione algebrica di questo problema è la seguente:

– *Variabili di decisione.* È immediato introdurre due variabili x_1 e x_2 che rappresentano rispettivamente la lunghezza (in metri) del raggio di base e dell'altezza del contenitore cilindrico.

– *Funzione obiettivo.* La funzione obiettivo è rappresentata dal volume del

contenitore cilindrico ed è data da

$$\pi x_1^2 x_2.$$

– *Vincoli.* Il diametro della base non può superare le dimensioni del magazzino e quindi deve essere

$$2x_1 \leq 10.$$

La limitazione dell'altezza del contenitore varia al variare del diametro di base in quanto il tetto è spiovente. Dato che la pendenza del tetto è del 20%, dovrà risultare

$$x_2 \leq 5 - 0.2 \cdot 2x_1.$$

Inoltre disponendo solo di una quantità limitata di materiale plastico la superficie totale del contenitore cilindrico non può superare $200m^2$ e quindi deve risultare

$$2\pi x_1^2 + 2\pi x_1 x_2 \leq 200.$$

Si devono infine esplicitare i vincoli di non negatività $x_1 \geq 0$, $x_2 \geq 0$.
La formulazione complessiva risulta quindi

$$\begin{cases} \max\ \pi x_1^2 x_2 \\ x_1 \leq 5 \\ x_2 \leq 5 - 0.2 \cdot 2x_1 \\ 2\pi x_1^2 + 2\pi x_1 x_2 \leq 200 \\ x_1 \geq 0, \quad x_2 \geq 0. \end{cases}$$

Si riporta di seguito un'implementazione AMPL di tale modello.

serbatoio.mod

```
var x1>=0;
var x2 >=0;

param pi := 3.14159265358979; # approssimazione di pi-greco
param lato_base_corto;
param lato_base_lungo;
param altezza_max;
param altezza_min;
param disponibilita_materiale;

param pendenza:= (altezza_max-altezza_min)/lato_base_corto;
            # si assume tetto spiovente lungo il lato corto

maximize volume :  pi*x1^2 * x2;
```

```
s.t. dim_base : 2*x1 <= lato_base_corto;

s.t. altezza : x2 <= altezza_max - pendenza * 2*x1;

s.t. sup_totale : 2*pi*x1^2 + 2 * pi * x1 * x2
                  <= disponibilita_materiale;
```

serbatoio.dat

```
param lato_base_corto := 10;
param lato_base_lungo := 20;
param altezza_max := 5;
param altezza_min := 3;
param disponibilita_materiale := 200;
```

Naturalmente per risolvere questo problema sarà necessario utilizzare un solutore non lineare, ad esempio `MINOS` o `SNOPT`. Utilizzando `MINOS` si ottiene il seguente risultato:

```
MINOS 5.5: optimal solution found.
0 iterations, objective 0
Nonlin evals: obj = 3, grad = 2, constrs = 3, Jac = 2.
: _varname _var    :=
1   x1       0
2   x2       0
;

: _objname _obj    :=
1   volume   0
;
```

Chiaramente il solutore non è riuscito a determinare una soluzione ottima del problema. Senza entrare in nessun dettaglio riguardante gli algoritmi di Programmazione Non Lineare, diciamo solamente che a tale spiacevole situazione si può porre rimedio cambiando il punto iniziale dell'algoritmo, ovvero assegnando un valore iniziale alle variabili diverso da quello assegnato per default che è lo zero per tutte le variabili. Si ricorda che per assegnare un valore iniziale alle variabili il comando da utilizzare è `let`. In questo caso, assegnando ad esempio

```
let x1:=1;
```

```
let x2:=1;
```

il risultato che si ottiene è il seguente:

```
MINOS 5.5: optimal solution found.
35 iterations, objective 185.5942843
Nonlin evals: obj = 105, grad = 104, constrs = 105, Jac = 104.
: _varname    _var      :=
1   x1       4.22456
2   x2       3.31017
;

: _objname    _obj      :=
1   volume   185.594
;
```

La situazione è analoga se si utilizza `SNOPT`. Anche in questo caso, utilizzando il punto iniziale $x_1 = 1$ e $x_2 = 1$ si ottiene lo stesso risultato.

```
SNOPT 7.2-8 : Optimal solution found.
7 iterations, objective 185.5942843
Nonlin evals: obj = 12, grad = 11, constrs = 12, Jac = 11.
: _varname    _var      :=
1   x1       4.22456
2   x2       3.31017
;

: _objname    _obj      :=
1   volume   185.594
;
```

Come già detto, non accenniamo neanche alle problematiche tipiche della Programmazione Non Lineare legate all'esistenza di soluzioni ottime, alla presenza di ottimi locali e ottimi globali, alla natura degli algoritmi e tantissime altre.

8

Approfondimenti sui parametri. Script in AMPL

8.1 PARAMETRI SIMBOLICI

In alcuni casi può essere molto utile rendere ancora più flessibile la scrittura di un modello in AMPL introducendo dei parametri che sostanzialmente possono essere visti come "parametri variabili" il cui valore è assegnato nel file `.dat`. Tali parametri rappresentano, di fatto, *stringhe di caratteri*. Ciò si ottiene con la parola chiave `symbolic` che può essere associata alla dichiarazione dei parametri. Vengono di solito utilizzati per selezionare elementi di un insieme oppure per associare stringhe di caratteri agli elementi di un insieme.

Un esempio può essere il seguente: nel file `.mod`

```
set INSIEME;
param elemento symbolic in INSIEME;
```

e nel file `.dat`

```
set INSIEME = e1 e2 e3 d4 e5;
param elemento := e3;
```

Un semplice esempio di uso di parametri simbolici può riguardare il loro utilizzato per riformulare i vincoli logici del problema della dieta dell'Esempio 6.3.1 senza dover specificare nel file del modello i due alimenti alternativi, ma parametrizzzando anche la loro scelta che verrà quindi specificata nel file dei dati. Si riportano di seguito i file di tale modello.

dieta2.mod

```
set ALIMENTI;
set SOSTANZE;

param contenuti{ALIMENTI,SOSTANZE}>=0;
param costi{ALIMENTI}>=0;
param contenuti_min{SOSTANZE};
param BigM>0;

param alimento1 symbolic in ALIMENTI;
param alimento2 symbolic in ALIMENTI;

var x{ALIMENTI}>=0;
var d binary;

minimize costo : sum{i in ALIMENTI} costi[i]*x[i];

s.t. vincoli_qualita{j in SOSTANZE} : sum{i in ALIMENTI}
                 contenuti[i,j]*x[i] >= contenuti_min[j];

s.t. vincolo_logico1 : x[alimento1] - BigM * d <=0;
s.t. vincolo_logico2 : x[alimento2] - (1-d)*BigM <=0;
```

dieta2.dat

```
set ALIMENTI:= A1 A2 A3 A4 A5;
set SOSTANZE:= prot carb grassi vitam calcio;

param BigM := 10000;

param alimento1 := A2;
param alimento2 := A3;

param costi :=
A1  0.15
A2  0.23
A3  0.79
A4  0.47
```

```
A5  0.52;

param contenuti_min :=
prot 3
carb 10
grassi 2
vitam 3
calcio 2;


param contenuti : prot carb grassi vitam calcio :=
A1  0 7 1 1 0
A2  1 0 3 1 4
A3  5 0 4 0 1
A4  2 2 1 3 0
A5  0 3 0 2 1;
```

Vediamo ora un altro esempio in cui utilizziamo i parametri simbolici.

Esempio 8.1.1 *Una azienda automobilistica produce tre diversi modelli di autovettura: un modello economico, uno normale ed uno di lusso. Ogni autovettura viene lavorata da tre robot:* **A**, **B** *e* **C**. *I tempi necessari alla lavorazione sono riportati, in minuti, nella tabella seguente insieme al profitto netto realizzato per autovettura*

	Economica	**Normale**	**Lusso**
A	20	30	62
B	31	42	51
C	16	81	10
Prezzo	1000	1500	2200

I robot **A** *e* **B** *sono disponibili per 8 ore al giorno mentre il robot* **C** *è disponibile per 5 ore al giorno. Il numero di autovetture di lusso prodotte non deve superare il 20% del totale mentre il numero di autovetture economiche deve costituire almeno il 40% della produzione complessiva. Se si produce piú del 50% di autovetture economiche, allora c'è da pagare una penale pari a 10000. Supponendo che tutte le autovetture vengano vendute, formulare un problema di Programmazione Lineare che permetta di decidere le quantità giornaliere da produrre per ciascun modello in modo tale da massimizzare i profitti rispettando i vincoli di produzione.*

Prima di scrivere i file del modello e dei dati, si riporta la formulazione algebrica di questo problema. Indichiamo con x_1, x_2, x_3, rispettivamente il numero di autovetture del modello economico, normale e di lusso da produrre giornalmente e con $\delta \in \{0, 1\}$ una variabile binaria così definita:

$$\delta = \begin{cases} 1 & \text{se si produce piú del 50\% di autovetture economiche} \\ 0 & \text{altrimenti} \end{cases}$$

Con questa scelta delle variabili una formulazione del problema è la seguente:

$$\begin{cases} \max\left(1000x_1 + 1500x_2 + 2200x_3 - 10000\delta\right) \\ 20x_1 + 30x_2 + 62x_3 \leq 480 \\ 31x_1 + 42x_2 + 51x_3 \leq 480 \\ 16x_1 + 81x_2 + 10x_3 \leq 300 \\ x_3 \leq 0.2\left(x_1 + x_2 + x_3\right) \\ x_1 \geq 0.4\left(x_1 + x_2 + x_3\right) \\ x_1 - 0.5(x_1 + x_2 + x_3) \leq M\delta \\ x_1 \geq 0, x_2 \geq 0, x_3 \geq 0 \quad \textit{intere} \\ \delta \in \{0, 1\}. \end{cases}$$

Si riportano di seguito i file `autovetture.mod` e `autovetture.dat` che implementano questo problema facendo uso dei parametri simbolici.

autovetture.mod

```
####   SEZIONE DEFINIZIONE SET   ####

set AUTOVETTURE  ;
set ROBOT;

####   SEZIONE DEFINIZIONE E TEST PARAMETRI    #####

param par1 symbolic in AUTOVETTURE;
param par2 symbolic in AUTOVETTURE;

param prezzi{AUTOVETTURE}>=0;

param dispo_max{ROBOT}>=0;

param rich_tempo{ROBOT,AUTOVETTURE}>=0 , integer;

param perc_max_lusso >=0;
param perc_min_economica >=0;
```

```
param penale >0;
param big_M:= 10^6;

####   SEZIONE DICHIARAZIONE VARIABILI    ####

var x{AUTOVETTURE}>=0, integer;
var delta >=0, binary;

####   SEZIONE VINCOLI E FUNZ. OBIETTIVO   ####

maximize profitto: sum{j in AUTOVETTURE}
     prezzi[j]*x[j] - penale*delta;

s.t. v_min_robot {i in ROBOT}: sum{j in AUTOVETTURE}
     rich_tempo[i,j]*x[j] <= dispo_max[i];

s.t. max_lusso : x[par1] <= perc_max_lusso *
                 (sum{j in AUTOVETTURE} x[j]);


s.t. min_economiche : x[par2]>= perc_min_economica *
                      (sum{j in AUTOVETTURE} x[j]);

s.t. penalita: x[par2] - .5*sum{j in AUTOVETTURE} x[j]
               <=big_M*delta;
```

autovetture.dat

```
set AUTOVETTURE := economica normale lusso;
set ROBOT := A B C;

param par1 := lusso;
param par2 := economica;

param:    prezzi :=
economica    1000
normale      1500
lusso        2200     ;

param: dispo_max :=
   A           480
```

```
   B           480
   C           300     ;

param rich_tempo :  economica    normale     lusso :=
          A       20        30          62
          B       31        42          51
          C       16        81          10   ;

param perc_max_lusso := 0.2;
param perc_min_economica := 0.4;
param penale := 10000;
```

8.2 PARAMETRI RANDOM

Abbiamo già parlato in precedenza della possibilità di introdurre parametri *calcolati*, ovvero definiti attraverso un'espressione algebrica che ne definisce il valore. È inoltre possibile definire valori dei parametri in maniera random, ovvero generati in accordo ad opportune distribuzioni di probabilità. Questo può essere di utilità quando in un modello alcune quantità non sono deterministiche, ma si conosce la distribuzione di probabilità che le caratterizza. Ad esempio, la disponibilità di una risorsa potrebbe essere definita non in maniera deterministica, ma potrebbe essere noto che tale disponibilità è distribuita secondo una distribuzione di probabilità (distribuzione che può essere, ad esempio, dedotta dall'osservazione di dati storici). In questo caso si può, ad esempio, definire

```
param media >0;
param varianza >0;
param disponibilita = Normal(media,varianza);
```

In questo modo, alla disponibilità verrà assegnato un valore casuale estratto dalla distribuzione Normale a media e varianza assegnata. Per questa funzionalità AMPL dispone di funzioni "built–in" che generano osservazioni estratte da distribuzioni di probabilità. Le distribuzioni implementate sono le più comunemente utilizzate. Per una descrizione completa si veda la Tabella A-3 dell'Appendice di [Fourer et al., 2003].

Come è noto la generazione di numeri pseudo–casuali dipende da un *seme* (seed) iniziale da assegnare al generatore. Con i comandi

```
option randseed;
option randseed valore;
```

si visualizza il seme utilizzato e si assegna il seme ad un dato valore.

8.3 SCRIPT IN AMPL

Gli *script* sono sequenze di istruzioni memorizzate in un file `.run` che vengono eseguite una dopo l'altra e risultano molto utili per implementare l'esecuzione sequenziale di operazioni che possono anche essere ripetute più volte. Abbiamo di fatto già realizzato degli script scrivendo i file `.run` che fino ad ora abbiamo di solito utilizzato per

- caricare il file del modello;
- caricare il file dei dati;
- specificare il solutore da utilizzare;
- specificare eventuali opzioni;
- lanciare il solutore;
- stampare i risultati ottenuti.

Un esempio di file `.run` che effettua questa sequenza di operazioni è quello utilizzato per risolvere il problema dell Esempio 7.1.1:

esempio.run

```
model serbatoio.mod;
data serbatoio.dat;

option show_stats 1;
option solver minos;

let x1:=1;
let x2:=1;

solve;

display _varname, _var;
display _objname, _obj;
```

Gli script in AMPL possono includere anche costrutti sintattici più complessi che fanno uso di istruzioni che permettono di generare cicli e/o ripetizioni di istruzioni basandosi sul soddisfacimento o meno di condizioni logiche. Si tratta delle istruzioni *for*, *repeat while* e *repeat until*, e dell'istruzione *if then else*.

8.3.1 Istruzione `for`

L'istruzione `for` permette di ripetere una sequenza di istruzioni una volta per ogni membro di un insieme. Ovvero, l'istruzione `for` esegue al variare di un elemento all'interno di un insieme una sequenza di istruzioni. La sintassi è la seguente:

```
for {elemento in INSIEME}{
      istruzione1;
      istruzione2;
      istruzione3;
           .
           .
           .
      }
```

Un esempio banale dell'uso del `for` è il seguente: avendo definito

```
set INSIEME = {1..10};
var x{INSIEME};
```

invece di scrivere:

```
display x[1];
display x[2];
display x[3];
 .
 .
 .
```

si può scrivere in maniera molto più sintetica

```
for {i in INSIEME}{
  display x[i];
  }
```

Oppure il `for` potrebbe, ad esempio, essere utilizzato per risolvere più volte un problema cambiando ogni volta uno o più dati. Supponiamo di avere un modello di allocazione ottima di risorse nel quale sono definite delle disponibilità massime di ciascuna delle risorse. In questo caso si potrebbe scrivere uno script di questo tipo:

```
for {i in 1..10}{
     let disponibilita["risorsa1"]:=disponiblita["risorsa1"]+1;
     solve;
```

```
display i, profitto;
}
```

In questo modo il problema viene risolto dieci volte aumentando ogni volta di una unità la disponibilità della `risorsa1`.

8.3.2 Istruzione `repeat while` e `repeat until`

L'istruzione `repeat` permette di ripetere l'esecuzione di una sequenza di istruzioni fino al verificarsi di una certa condizione logica. La differenza tra il `repeat while` e `repeat until` risiede nel fatto che il ciclo definito dal `while` viene ripetuto in presenza di una condizione logica ***vera***, mentre il ciclo definito dall'`until` viene ripetuto in presenza di una condizione logica ***falsa***. La sintassi del `repeat while` è la seguente:

```
repeat while condizione_logica { istruzioni };

# oppure

repeat { istruzioni } while condizione_logica;
```

In questo modo le `istruzioni` vengono ripetute fintantoché la `condizione_logica` è ***vera***. La sintassi del `repeat until` è analoga:

```
repeat until condizione_logica { istruzioni };

# oppure

repeat { istruzioni } until condizione_logica;
```

In questo modo le `istruzioni` vengono ripetute fintantoché la `condizione_logica` è ***falsa***. Si osservi la possibilità di avere la verifica della condizione logica prima o dopo le istruzioni. Infatti in talune circostanze può risultare utile una posizione piuttosto che l'altra.

Quindi se, ad esempio, vogliamo far eseguire una sequenza di istruzioni fino a quando un certo `valore` diventi zero, ovvero vogliamo che la ripetizioni si arresti quando `valore=0`, possiamo scrivere una delle seguenti quattro istruzioni:

```
repeat while valore > 0 { . . . };
repeat until valore = 0 { . . . };
```

```
repeat { . . . } while valore > 0;
repeat { . . . } until valore = 0;
```

8.3.3 Istruzione `if then else`

L'istruzione `if` permette di effettuare un controllo condizionale dell'esecuzione di una o più istruzioni. Nel caso più semplice l'`if` permette di valutare una condizione logica ed eseguire delle istruzioni se questa condizione è vera:

```
if condizione_logica then { . . . }
```

Può inoltre essere presente l'`else` per indicare il verificarsi della condizione alternativa:

```
if condizione_logica then { . . . }
else { . . .}
```

Oppure si può considerare il verificarsi di più condizioni:

```
if condizione_logica1 then { . . . }
else if condizione_logica2 then  { . . .}
else if condizione_logica3 then  { . . .}
   .
   .
   .
else  { . . . }
```

8.3.4 Terminazione di cicli con istruzioni `continue` e `break`

Per rendere più flessibile il modo di realizzare cicli all'interno di script esistono due istruzioni che si possono utilizzare per interrompere i cicli `for` e i cicli `repeat`. Si tratta delle istruzioni `continue` e `break`. La prima termina un ciclo saltando tutte le istruzioni che seguono portando il controllo all'esecuzione del test da effettuare prima del successivo ciclo. La seconda termina un ciclo in maniera definitiva. La sintassi è la seguente:

```
repeat while condizione_logica1 {
     . . .
     if condizione_logica2 then continue;
     . . .}
```

In questo caso le istruzioni che seguono il `continue` non verranno eseguite e il controllo passa alla verifica della `condizione_logica1` che viene effettuata all'inizio di ogni ciclo. Altrimenti, utilizzando `break`, ovvero

```
repeat while condizione_logica1 {
     . . .
     if condizione_logica2 then break;
     . . .}
```

il ciclo `repeat` è definitivamente interrotto. In modo del tutto analogo, `continue` e `break` si utilizzano all'interno di un ciclo `repeat until` o di un ciclo `for`.

8.3.5 Un esempio di script

Per scrivere un esempio di script in AMPL consideriamo di nuovo il modello dell'Esempio 8.1.1, supponendo di voler risolvere più volte il problema variando la disponibilità massima del robot **C**. In particolare, innanzitutto si vuole risolvere il problema assegnato; poi, dopo aver verificato che tale disponibilità è almeno pari a 300, si vuole risolvere di nuovo il problema diminuendo la disponibilità di volta in volta di 10, di 20, di 30, ... arrestandosi quando tale disponibilità dovesse scendere al di sotto di 150.

esempio1.run

```
reset;

model autovetture.mod;
data autovetture.dat;

option show_stats 1;
option solver cplex;

solve;

display _varname, _var;
display _objname, _obj;

if dispo_max["C"] >= 300 then {
    for { i in 10..50 by 10}{
       let dispo_max["C"]:= dispo_max["C"]-i;
       display i;
       display dispo_max["C"];
       solve;
       display _varname, _var;
       display _objname, _obj;
```

```
        if dispo_max["C"] < 150  then break;
        }
        }
else {display dispo_max["C"]}
```

8.3.6 Il comando `printf`

Nello script proposto nel paragrafo precedente sono state inserire istruzioni di visualizzazione dei risultati attraverso il comando `display`. Tale comando, in realtà, non è molto flessibile perché non permette né di specificare il formato di scrittura, né di inserire frasi di commento nella visualizzazione. In alternativa si può utilizzare il comando `printf` che pur essendo analogo al `display`, permette all'utente di decidere il formato di scrittura. Si utilizza molto semplicemente inserendo una stringa da visualizzare tra apici. La stringa è composta da due oggetti: caratteri da visualizzare e il carattere speciale `%` seguito da una specifica di formato. Segue poi una virgola e le variabili da visualizzare associate nell'ordine specificato all'interno della stringa. Un semplice esempio è il seguente:

```
printf "Profitto totale = %d \n" , profitto;
```

Il valore da visualizzare è contenuto nella variabile `profitto` che verrà visualizzata nella stringa di output al posto della specifica di formato `%d`. L'istruzione che segue `\n` ha solo la funzione di andare alla riga successiva nella visualizzazione dell'output. In questo caso se il valore di `profitto` è pari a 1532.55, l'output sarebbe

```
Profitto totale = 1532.55
```

La specifica di formato `%d` indica il formato decimale standard con la virgola. Sono disponibili diversi altri formati. Alcuni esempi sono:

- `%f` per il formato in doppia precisione floating point
- `%e` per il formato in doppia precisione floating point con notazione esponenziale
- `%s` per il formato stringa.

Per l'elenco completo dei formati del comando `printf` si fa riferimento alla Tabella A-10 dell'Appendice di [Fourer et al., 2003].

Come altro esempio dell'uso di `printf`, usiamo questo comando per visualizzare le disponibilità dei robot nell'Esempio 8.1.1:

```
for {j in ROBOT}{
printf "La dispon. max di %s e' uguale a %d\n",
j, dispo_max[j];
}
```

Possiamo, ora, riscrivere lo script riportato nel paragrafo precedente utilizzando il comando `printf`. L'output sarà sicuramente migliore, presentando una maggiore leggibilità.

esempio2.run

```
if dispo_max["C"] >= 300 then {
    printf "La disponib. max di %s e' pari a %d\n",
                                "C", dispo_max["C"];
    printf "possiamo ridurla \n";
    for { i in 10..100 by 10}{
         let dispo_max["C"]:= dispo_max["C"]-i;

         printf " *** i= %d *** \n", i;
         printf " *** Dispon. max di %s impostata a %d *** \n",
                                          "C", dispo_max["C"];
         if dispo_max["C"] < 150  then {
             printf " ***** STOP ***** \n";
             break;
             }
         solve;
         display _varname, _var;
         display _objname, _obj;
         }
      }
else {printf "Dispon. max di %s pari a %d : inferiore a 300 \n",
                                "C", dispo_max["C"]}
```

8.3.7 Esempi

Esaminiamo ora alcuni semplici modelli in cui utilizziamo uno script per risolvere più volte uno stesso problema variando alcuni valori dei parametri.

Esempio 8.3.1 *Un'acciaieria acquista rottame di quattro tipi differenti (T1, T2, T3, T4) per ottenere due leghe (L1, L2) con caratteristiche chimiche differenti. I quattro tipi di rottame hanno le seguenti caratteristiche, cioè il contenuto percentuale di Piombo, di Zinco, di Stagno e il prezzo unitario di acquisto (espresso in migliaia di Euro la tonnellata).*

	T1	T2	T3	T4
Piombo	40%	30%	25%	38%
Zinco	35%	40%	35%	32%
Stagno	25%	30%	40%	30%
prezzo	2.5	1.8	2	2.2

Le due leghe devono avere le seguenti caratteristiche: la lega L1 deve avere un contenuto non superiore al 30% di piombo, al 60% di zinco e al 42% di stagno. La lega L2 deve avere un contenuto non superiore al 46% di piombo, al 38% di zinco e al 56% di stagno. Si tratta di definire le quantità di ciascun tipo di rottame da utilizzare in ciascuna delle leghe in modo da minimizzare il costo complessivo e in modo da soddisfare esattamente un ordine di 1500 tonnellate di lega L1 e 2000 tonnellate di lega L2.

1. *Scrivere* in forma parametrica *i file* `.mod`, `.dat` *e il file* `.run` *che realizzano un'implementazione in* AMPL *di questo problema.*
2. *Risolvere quindi il problema riportando il valore ottimo ottenuto e il valore delle variabili all'ottimo.*
3. *Realizzare nel file* `.run` *uno script in* AMPL *che permetta di risolvere dieci volte il problema, aumentando ogni volta di 100 tonnellate l'ordine di ciascuna lega.*

Si riportano di seguito i file richiesti.

rottame.mod

```
set ROTTAME;
set LEGA;
set INGREDIENTI;

param contenuti{INGREDIENTI,ROTTAME};
param contenuti_max{INGREDIENTI,LEGA};
```

```
param prezzo{ROTTAME};

param ordine{LEGA};

var x{ROTTAME,LEGA}>=0;

minimize costo : sum{i in ROTTAME} prezzo[i]*sum{j in LEGA} x[i,j];

s.t. qualita1{i in INGREDIENTI, l in LEGA} : sum{j in ROTTAME}
                                        contenuti[i,j] * x[j,l] <=
                      contenuti_max[i,l]*sum{j in ROTTAME} x[j,l];

s.t. vinc_ordine{k in LEGA} : sum{i in ROTTAME} x[i,k] = ordine[k];
```

rottame.dat

```
set ROTTAME :=  T1 T2 T3 T4;
set LEGA := L1 L2;
set INGREDIENTI := piombo zinco stagno;

param contenuti: T1  T2  T3 T4:=
piombo          .40  .30  .25 .38
zinco           .35  .40  .35 .32
stagno          .25  .30  .40 .30;

param contenuti_max : L1  L2 :=
piombo     .30   .46
zinco      .60   .38
stagno     .42   .56;

param prezzo :=
T1  2.5
T2  1.8
T3  2.0
T4  2.2;

param ordine
L1  1500
L2  2000;
```

Si riporta di seguito la soluzione ottenuta utilizzando il solutore CPLEX:

```
Presolve eliminates 4 constraints.
Adjusted problem:
8 variables, all linear
4 constraints, all linear; 15 nonzeros
2 equality constraints
2 inequality constraints
1 linear objective; 8 nonzeros.

CPLEX 12.6.0.1: optimal solution; objective 6460
2 dual simplex iterations (0 in phase I)
:      _varname      _var    :=
1   "x['T1','L1']"      0
2   "x['T1','L2']"      0
3   "x['T2','L1']"   1500
4   "x['T2','L2']"   1200
5   "x['T3','L1']"      0
6   "x['T3','L2']"    800
7   "x['T4','L1']"      0
8   "x['T4','L2']"      0
;

: _objname   _obj    :=
1   costo    6460
;
```

Di seguito il file `.run` richiesto al punto 3.

rottame.run

```
reset;

model rottame.mod;
data rottame.dat;

for { j in LEGA}{
    printf "*** ordine di %s impostato a %d *** \n",
                   j , ordine[j];
}

option show_stats 1;
option solver cplex;

solve;

display _varname, _var;
display _objname, _obj;


printf "Aumento ordine \n";
   for { i in 1..10 }{
       printf "*** i = %d *** \n", i;
       for { j in LEGA }{
   let ordine[j]:= ordine[j]+100;
       printf "*** ordine di %s impostato a %d *** \n",
                   j , ordine[j];
        }
solve;

display _varname, _var;
display _objname, _obj;
}
```

Esempio 8.3.2 *Una fabbrica produce divani in tessuto acquistando da un magazzino all'ingrosso i quantitativi di tessuto che gli occorrono settimanalmente. La lavorazione è divisa in due fasi: nella prima fase il tessuto viene tagliato secondo le misure richieste per i vari modelli e nella seconda fase viene effettuata la cucitura. I tipi di divani attualmente in produzione sono 4: D1, D2, D3 , D4. Il prezzo di vendita unitario dei divani del tipo D2 e D3 è pari a 1500 Euro. I divani del tipo D1 sono venduti a 1600 Euro ciascuno e i divani del tipo D4 sono venduti a 1550 Euro ciascuno. Nella tabella che segue sono riportati i tempi di lavorazione (in ore) di ciascuna delle due fasi di lavorazione necessari per ottenere un divano di ciascun tipo finito pronto per la vendita insieme al quantitativo in m^2 di tessuto necessario e alla quantità minima di ciascun tipo di divano che deve essere fabbricata settimanalmente:*

	D1	D2	D3	D4
prima fase	1	0.75	1.25	1
seconda fase	1.25	1.5	1.3	1
tessuto necessario	3.7	3.8	3.5	3.8
quantità minime	5	7	5	4

Alla prima fase della lavorazione sono addetti 8 operai mentre alla seconda fase sono addetti 10 operai (si assuma che, settimanalmente, ogni operaio lavori 35 ore). Il quantitativo di tessuto disponibile settimanalmente è pari a 500 m^2 e il costo è di 150 Euro al m^2. Per motivi di mercato, la quantità di ciascun tipo di divano fabbricato non può superare il 35% *del totale. Si vuole costruire un modello lineare che permetta di pianificare la produzione settimanale della fabbrica, ovvero le quantità (non necessariamente intere) di divani da fabbricare (e quindi da vendere) settimanalmente in modo da massimizzare il profitto netto complessivo (ricavo dalla vendita sottratto dei costi del tessuto effettivamente utilizzato).*

1. *Scrivere* in forma parametrica *i file* `.mod`, `.dat` *e il file* `.run` *che realizzano un'implementazione in* AMPL *di questo problema. Risolvere quindi il problema riportando il valore ottimo ottenuto.*

2. *Determinare se acquistare settimanalmente un quantitativo di tessuto superiore a 500 m^2 porterebbe ad un aumento del profitto. In caso affermativo specificare il massimo prezzo unitario al quale è conveniente acquistare quantità di tessuto aggiuntivo, spiegandone brevemente il motivo.*

3. *Realizzare nel file* `.run` *uno script in* AMPL *che permetta di risolvere più volte il problema, aumentando ogni volta di 100 m^2 la disponibilità settimanale di pellame fino a quando a tale incremento non corrisponde più un aumento del profitto. Riportare il valore della disponibilità così ottenuta al quale non corrisponde più un aumento del profitto.*

Riportiamo di seguito i file richiesti.

divani.mod

```
set DIVANI;
set RISORSE;

param prezzi{DIVANI}>=0;
param disponibilita{RISORSE}>=0;
param utilizzo_risorse{RISORSE,DIVANI}>=0;
param quantita_minime{DIVANI};
param costo_tessuto>=0;

var x{DIVANI} >=0;

maximize profitto : sum{i in DIVANI} prezzi[i] * x[i]-
 costo_tessuto*sum{j in DIVANI} utilizzo_risorse["tessuto",j]*x[j];

s.t. vincoli_risorse{i in RISORSE} : sum{j in DIVANI}
     utilizzo_risorse[i,j] * x[j] <= disponibilita[i];

s.t. vincoli_quantita{j in DIVANI} : x[j]>= quantita_minime[j];

s.t. vincolo_percentuale{j in DIVANI} : x[j] <=
                                    0.35 * sum{k in DIVANI} x[k];
```

divani.dat

```
set DIVANI := D1 D2 D3 D4;
set RISORSE := fase1 fase2 tessuto;

param prezzi
D1   1600
D2   1500
D3   1500
D4   1550;

param disponibilita
fase1  280
fase2  350
tessuto 500;

param utilizzo_risorse :  D1 D2 D3 D4 :=
```

```
fase1   1   0.75   1.25   1
fase2   1.25   1.5   1.3  1
tessuto 3.7   3.8   3.5   3.8;

param quantita_minime
D1   5
D2   7
D3   5
D4   4;

param costo_tessuto:=150;
```

Si riporta di seguito la soluzione ottenuta utilizzando il solutore CPLEX:

```
Presolve eliminates 4 constraints.
Adjusted problem:
4 variables, all linear
7 constraints, all linear; 28 nonzeros
7 inequality constraints
1 linear objective; 4 nonzeros.

CPLEX 12.6.1.0: optimal solution; objective 136398.6339
5 dual simplex iterations (1 in phase I)
:   _varname     _var      :=
1   "x['D1']"   47.8142
2   "x['D2']"    7
3   "x['D3']"   47.8142
4   "x['D4']"   33.9836
;

:   _objname    _obj     :=
1   profitto   136399
;
```

Il massimo prezzo unitario al quale è conveniente acquistare quantità di tessuto aggiuntivo si determina visualizzando il *prezzo ombra* associato al vincolo sulla disponibilità di tessuto:

```
ampl: display vincoli_risorse["tessuto"];
vincoli_risorse["tessuto"] = 273.497
```

Di seguito il file `.run` richiesto al punto 3.

divani.run

```
reset;

model divani.mod;
data divani.dat;

option show_stats 1;
option solver cplex;

solve;

display _varname, _var;
display _objname, _obj;

repeat while vincoli_risorse['tessuto'] > 0 {
   let disponibilita["tessuto"] := disponibilita["tessuto"] +100;
   printf" DISPONIBILITA' di TESSUTO = %d\n",
                                        disponibilita["tessuto"];
   solve;
   display _varname, _var;
   display _objname, _obj;
};
```

Il valore della disponibilità ottenuta al quale non corrisponde più un aumento del profitto è pari a 1100:

```
DISPONIBILITA' di TESSUTO = 1100

Presolve eliminates 4 constraints.
```

```
Adjusted problem:
4 variables, all linear
7 constraints, all linear; 28 nonzeros
7 inequality constraints
1 linear objective; 4 nonzeros.

CPLEX 12.6.1.0: optimal solution; objective 285155.832
1 dual simplex iterations (0 in phase I)
:   _varname      _var      :=
1   "x['D1']"   100.591
2   "x['D2']"    57.916
3   "x['D3']"    28.3048
4   "x['D4']"   100.591
;

:   _objname    _obj     :=
1   profitto   285156
;
```

Esempio 8.3.3 *Un'industria produce un modello di scooter in cinque cilindrate diverse (50cc, 100cc, 125cc, 150cc, 250cc). Ciascuno scooter, per essere pronto per la vendita, deve essere lavorato in tutti e tre i reparti (Rep1, Rep2, Rep3) di cui questa industria dispone. I tempi di lavorazione di ciascuno scooter in ogni reparto differiscono a seconda della cilindrata. La tabella che segue riporta questi tempi (in ore) insieme al massimo numero di ore disponibili settimanalmente in ciascun reparto e al prezzo di vendita unitario (in Euro) del modello nelle varie cilindrate.*

	50cc	**100cc**	**125cc**	**150cc**	**250cc**	ore
Rep1	8	12	14	15	18	3200
Rep2	8	9	10	12	16	2800
Rep3	4	5	7	8	13	2400
prezzo di vendita	1600	2000	2800	3000	4000	

Sapendo che il numero di scooter di ciascuna cilindrata non deve superare rispettivamente il 30% , il 25%, il 35%, il 45% e il 40% della produzione totale, si vuole pianificare la produzione settimanale, cioè si vuole determinare la quantità di scooter di ogni cilindrata da produrre, in modo da massimizzare il profitto complessivo, trascurando l'interezza delle variabili.

1. *Scrivere* in forma parametrica *i file* `.mod`, `.dat` *che realizzano un'implementazione in* AMPL *di questo problema e il file* `.run` *che contenga uno script che permetta di risolvere il problema.*

2. *Risolvere il problema e riportare di seguito il valore ottimo della funzione obiettivo e il valore della variabili in corrispondenza dell'ottimo determinato.*

3. *Determinare in quale reparto è conveniente avere un aumento della disponibilità oraria, indicando di seguito anche il prezzo massimo che è conveniente pagare per l'incremento di una ora di tale disponibilità.*

4. *Modificare il file* `.mod` *per aggiungere la presenza della limitazione commerciale che consiste nell'imporre che se si produce il modello nella cilindrata 250cc allora non si può produrre il modello nella cilindrata 125cc e viceversa. Risolvere di nuovo il problema riportando di seguito il nuovo valore ottimo della funzione obiettivo e il valore della variabili in corrispondenza dell'ottimo determinato.*

5. *Realizzare nel file* `.run` *uno script in* AMPL *che permetta di risolvere più volte il problema originale (senza la modifica di cui al punto 4.), aumentando ogni volta di 10 ore la disponibilità oraria settimanale del Reparto 1 fino a quando a tale incremento non corrisponde più un aumento del profitto. Riportare di seguito il valore della disponibilità così ottenuta.*

Si riportano di seguito i file richiesti.

scooter.mod

```
set CILINDRATE;
set REPARTI;

param prezzo{CILINDRATE};
param tempi{REPARTI,CILINDRATE};

param disp_ore{REPARTI};
param perc_max{CILINDRATE};
param bigM=1000;

var x{CILINDRATE}>=0;
var delta binary;

maximize profitto : sum{j in CILINDRATE} prezzo[j]*x[j];

s.t. vinc_disp{i in REPARTI} : sum{j in CILINDRATE}
                          tempi[i,j]*x[j] <= disp_ore[i];

s.t. vinc_perc{j in CILINDRATE} : x[j] <=
                    perc_max[j]*sum{k in CILINDRATE} x[k];

#s.t. vinc_logico1 : x["250cc"]-bigM*delta <= 0;
#s.t. vinc_logico2 : x["125cc"]-bigM*(1-delta) <= 0;
```

scooter.dat

```
set CILINDRATE := 50cc 100cc 125cc 150cc 250cc;
set REPARTI := Rep1 Rep2 Rep3;

param : prezzo perc_max :=
50cc    1600   .30
100cc   2000   .25
125cc   2800   .35
150cc   3000   .45
250cc   4000   .40
;
```

```
param tempi : 50cc 100cc 125cc 150cc 250cc :=
Rep1  8  12  14  15  18
Rep2  8  9   10  12  16
Rep3  4  5    7   8  13
;

param disp_ore:=
Rep1  3200
Rep2  2800
Rep3  2400
;
```

Si riporta di seguito la soluzione ottenuta utilizzando il solutore CPLEX.

```
5 variables, all linear
8 constraints, all linear; 40 nonzeros
8 inequality constraints
1 linear objective; 5 nonzeros.

CPLEX 12.6.0.1: optimal solution; objective 677101.4493
3 dual simplex iterations (1 in phase I)
:     _varname      _var      :=
1   "x['50cc']"    69.5652
2   "x['100cc']"    0
3   "x['125cc']"   69.5652
4   "x['150cc']"    0
5   "x['250cc']"   92.7536
;

:   _objname    _obj     :=
1   profitto   677101
;
```

Per il punto 3, sarà sufficiente visualizzare i *prezzi ombra*:

```
ampl: display vinc_disp;
vinc_disp [*] :=
Rep1  211.594
Rep2    0
Rep3    0
;
```

L'unico prezzo ombra non nullo riguarda il reparto R1 e quindi il reparto nel quale è conveniente avere un aumento della disponibilità oraria è il reparto R1. Il prezzo massimo che è conveniente pagare per l'incremento di una ora di tale disponibilità è pari al prezzo ombra, ovvero a 211.594.

Lo script richiesto al punto 5 è riportato di seguito:

scooter.run

```
reset;

model scuter.mod;
data scooter.dat;

option show_stats 1;
option solver cplex;

solve;

display _varname, _var;
display _objname, _obj;

repeat while vinc_disp['Rep1'] > 0 {
let disp_ore['Rep1'] := disp_ore['Rep1'] + 10;
printf" DISPONIBILITA' REPARTO 1 = %d\n",  disp_ore['Rep1'];
solve;
display _varname, _var;
display _objname, _obj;
display vinc_disp['Rep1'];
};
```

Il valore della disponibilità oraria del reparto R1 in corrispondenza della quale non corrisponde più un aumento del profitto è pari a 3530:

```
 DISPONIBILITA' REPARTO 1 = 3530

5 variables, all linear
8 constraints, all linear; 40 nonzeros
8 inequality constraints
1 linear objective; 5 nonzeros.

CPLEX 12.6.3.0: optimal solution; objective 724297.5207
1 dual simplex iterations (0 in phase I)
:     _varname       _var      :=
1   "x['50cc']"      0
2   "x['100cc']"     0
3   "x['125cc']"    80.9917
4   "x['150cc']"   104.132
5   "x['250cc']"    46.281
;

:   _objname    _obj     :=
1   profitto   724298
;

vinc_disp['Rep1'] = 0
```

9
Esercizi di riepilogo

9.1 ESERCIZI SVOLTI

In questo paragrafo sono riportati alcuni esempi di modelli di Programmazione Lineare e di Programmazione Lineare Intera e la loro implementazione in AMPL, ovvero i relativi file `.mod` e `.dat`.

Esercizio 9.1.1 *Un'industria manifatturiera può fabbricare 5 tipi di prodotti che indichiamo genericamente con* **P1**, **P2**, **P3**, **P4**, **P5** *usando 2 processi di produzione che avvengono mediante l'uso di due macchine che indichiamo con* **M1** *e* **M2**. *Dopo aver dedotto il costo del materiale grezzo, ciascuna unità di prodotto dà i seguenti profitti (in migliaia di lire)*

P1	**P2**	**P3**	**P4**	**P5**
250	*300*	*500*	*450*	*180*

Ciascuna unità di prodotto richiede un certo tempo di ciascuno dei 2 processi; la tabella che segue riporta i tempi (in ore) di lavorazione di ciascuna macchina per ottenere una unità di ciascuno dei prodotti finiti

	P1	**P2**	**P3**	**P4**	**P5**
M1	*10*	*15*	*7*	*18*	–
M2	*9*	*13*	–	–	*20*

Inoltre, l'assemblaggio finale per ciascuna unità di ciascun prodotto richiede 18 ore di lavoro di un operaio. La fabbrica possiede 4 macchine **M1** *e 3 macchine*

M2 *che sono in funzione 5 giorni alla settimana per 2 turni di 8 ore al giorno. Gli operai impiegati nell'assemblaggio sono 10 e ciascuno di essi lavora 5 giorni alla settimana per un turno di 8 ore al giorno. Trovare la quantità che conviene produrre di ciascun prodotto per massimizzare il profitto totale.*

industria.mod

```
set PROD;
set MACCH;

param Profitto{PROD};
param Tempo_macch{PROD,MACCH};
param Tempo_oper;
param Numero_macch{MACCH};
param Numero_oper;
param Ore_max_macch;
param Ore_max_oper;

var Quantita{PROD} >= 0;

maximize profitto_tot: sum{p in PROD} Profitto[p]*Quantita[p];

subject to limite_macchine {m in MACCH}:
  sum{p in PROD} Tempo_macch[p,m]*Quantita[p] <=
  Numero_macch[m]*Ore_max_macch;

subject to limite_oper:
  Tempo_oper*sum{p in PROD} Quantita[p] <=
  Numero_oper*Ore_max_oper;
```

industria.dat

```
set PROD := P1 P2 P3 P4 P5;
set MACCH := M1 M2;

param:  Profitto :=
P1      250
P2      300
P3      500
P4      450
P5      180;
```

```
param Tempo_macch:  M1  M2 :=
P1                  10  9
P2                  15  13
P3                  7   0
P4                  18  0
P5                  0   20;

param Tempo_oper := 18;

param:  Numero_macch :=
M1      4
M2      3;

param Numero_oper := 10;
param Ore_max_macch := 80; # giorni*turni*ore
param Ore_max_oper := 40; # giorni*turni*ore
```

Esercizio 9.1.2 *Un'industria produce 4 tipi di elettrodomestici* **E1**, **E2**, **E3**, **E4** *ed è divisa in 3 reparti. Ciascun reparto può fabbricare ciascuno tipo di elettrodomestico. Questa industria dispone di 100 operai cosí ripartiti: 40 nel reparto 1, 35 nel reparto 2 e 25 nel reparto 3. Ciascun operaio lavora 5 giorni la settimana, per 8 ore al giorno. La tabella che segue riporta, per ciascun tipo di elettrodomestico e per ciascun reparto, il tempo di lavorazione (in ore) necessario per ottenere un elettrodomestico pronto per la vendita, insieme al prezzo di vendita unitario in migliaia di lire.*

	E1	**E2**	**E3**	**E4**
Reparto 1	*1*	*1.5*	*0.5*	*1.6*
Reparto 2	*1.2*	*1.3*	*0.6*	*1*
Reparto 3	*0.8*	*1.7*	*0.7*	*1.3*
prezzo di vendita	*800*	*1200*	*950*	*1100*

Questa industria deve pianificare la sua produzione settimanale, deve cioè determinare il numero di ciascuno degli elettrodomestici che deve essere fabbricato da ciascun reparto in modo da soddisfare un ordine di almeno 1000, 600, 300, 200 elettrodomestici rispettivamente del tipo **E1**, **E2**, **E3**, **E4** *e in modo da massimizzare il profitto complessivo ricavato dalla vendita.*

elettrodomestici.mod

```
set ELET;
set REP;

param NumOper{REP};
param OreOper;
param OreProd{ELET,REP};
param Prezzo{ELET};
param Domanda{ELET};

var Qta{ELET,REP} >= 0, integer;

maximize profitto: sum{e in ELET, r in REP} Prezzo[e]*Qta[e,r];

subject to produzione {r in REP}:
   sum{e in ELET} OreProd[e,r]*Qta[e,r] <= NumOper[r]*OreOper;

subject to domanda_min {e in ELET}:
   sum{r in REP} Qta[e,r] >= Domanda[e];
```

elettrodomestici.dat

```
set ELET := E1 E2 E3 E4;
set REP := R1 R2 R3;

param:  NumOper :=
R1      40
R2      35
R3      25;

param OreOper := 40;

param OreProd:  R1  R2  R3 :=
E1              1   1.2 0.8
E2              1.5 1.3 1.7
E3              0.5 0.6 0.7
E4              1.6 1   1.3;

param:  Prezzo  Domanda :=
E1      800     1000
E2      1200    600
```

```
E3      950     300
E4      1100    200;
```

Esercizio 9.1.3 *Una raffineria produce quattro tipi di benzine grezze (*$\mathbf{B_1}$, $\mathbf{B_2}$, $\mathbf{B_3}$, $\mathbf{B_4}$*) e le miscela allo scopo di ottenere carburanti di due diverse qualità (*$\mathbf{C_1}$, $\mathbf{C_2}$*). Le quantità di benzine grezze non utilizzate nella produzione delle miscele possono essere vendute direttamente. La seguente tabella riassume i dati delle benzine grezze, cioè il numero di ottani, la quantità (in ettolitri) che si può produrre al giorno e il costo (in migliaia di lire) di un ettolitro di ciascuna benzina.*

	$\mathbf{B_1}$	$\mathbf{B_2}$	$\mathbf{B_3}$	$\mathbf{B_4}$
n. ottani	*90*	*73*	*79*	*86*
ettolitri	*3500*	*6000*	*4500*	*5200*
costo	*260*	*210*	*190*	*220*

Nella seguente tabella sono riportate le caratteristiche che devono avere le miscele cioè il minimo numero di ottani e il prezzo di vendita di un ettolitro di carburante (in migliaia di lire)

	$\mathbf{C_1}$	$\mathbf{C_2}$
min. n. ottani	*80*	*85*
prezzo	*350*	*520*

Inoltre il mercato è in grado di assorbire non più di 25000 ettolitri al giorno del carburante $\mathbf{C_1}$*, mentre richiede almeno 10000 ettolitri al giorno di carburante* $\mathbf{C_2}$*. Infine, i quantitativi di benzine grezze prodotti ma non utilizzati nella preparazione delle miscele sono rivenduti al prezzo di 280 migliaia di lire per ettolitro se il numero di ottani è non inferiore a 80, e a 250 migliaia di lire per ettolitro altrimenti. Occorre pianificare la produzione giornaliera della raffineria, cioè le quantità e le composizioni delle due miscele, massimizzando il profitto ottenuto dalla vendita dei prodotti. Assumere che il numero di ottani di ciascuna miscela dipenda in modo lineare dalle gradazioni delle benzine componenti.*

raffineria.mod

```
set BENZ;
set CARB;

param Ottani{BENZ};
param Ettol_disp{BENZ};
```

```
param Costo{BENZ};
param Prezzo_benz{BENZ};
param Min_ottani{CARB};
param Prezzo_carb{CARB};
param Dom_min{CARB};
param Dom_max{CARB};

var Qta{BENZ,CARB} >= 0;
var Qta_grez{BENZ} >= 0;

maximize ricavo:
 (sum{b in BENZ, c in CARB} (Prezzo_carb[c]-Costo[b])*Qta[b,c])+
 (sum{b in BENZ} (Prezzo_benz[b]-Costo[b])*Qta_grez[b]);

subject to qualita {c in CARB}:
        sum{b in BENZ} (Ottani[b]-Min_ottani[c])*Qta[b,c] >= 0;

subject to disponibilita {b in BENZ}:
        sum{c in CARB} Qta[b,c] + Qta_grez[b] <= Ettol_disp[b];

subject to domanda {c in CARB}:
        Dom_min[c] <= sum{b in BENZ} Qta[b,c] <= Dom_max[c];
```

raffineria.dat

```
set BENZ := B1 B2 B3 B4;
set CARB := C1 C2;

param:  Ottani  Ettol_disp  Costo  Prezzo_benz :=
B1      90      3500        260    280
B2      73      6000        210    250
B3      79      4500        190    250
B4      86      5200        220    280;

param:  Min_ottani  Prezzo_carb  Dom_min  Dom_max :=
C1      80          350          0        25000
C2      85          520          10000    Infinity;
```

Esercizio 9.1.4 *Un'industria produce un preparato chimico utilizzando due impianti di produzione* **I1**, **I2**. *Da questi impianti tutto il preparato chimico prodotto viene trasportato in due magazzini* **M1**, **M2** *che si trovano in differenti località. In questi magazzini una parte del preparato è venduta all'ingrosso direttamente, un'altra parte viene inviata a quattro centri di distribuzione* **D1**, **D2**, **D3**, **D4** *che effettuano la vendita al minuto. Questi centri necessitano ripettivamente di almeno 150, 190, 220, 170 quintali di preparato chimico che vendono rispettivamente a Lire 350000, 280000, 200000, 270000 al quintale. La tabella che segue riporta i costi (in migliaia di lire) necessari per trasportare un quintale di preparato da ciascun impianto a ciascun magazzino.*

	M1	**M2**
I1	*21*	*25*
I2	*27*	*22*

Nella seguente tabella si riportano i costi (in migliaia di lire) necessari per trasportare un quintale di preparato chimico da ciascun magazzino a ciascun centro di distribuzione.

	D1	**D2**	**D3**	**D4**
M1	*33*	*31*	*36*	*30*
M2	*27*	*30*	*28*	*31*

L'impianto di produzione **I1** *può fabbricare al piú 3000 quintali di preparato, l'impianto* **I2** *può fabbricare al piú 2000 quintali di preparato. I prezzi di vendita all'ingrosso effettuati presso i magazzini* **M1** *e* **M2** *sono rispettivamente di Lire 150000 e 170000 al quintale. Per ragioni commerciali i quantitativi di preparato chimico venduti all'ingrosso in ciascun magazzino devono essere pari ad almeno 500 quintali ed inoltre tutto il preparato contenuto nei magazzini dovrà essere o venduto o trasportato ai centri di distribuzione per non avere rimanenze non vendute. Costruire un modello lineare che permetta di determinare le quantità di preparato chimico che devono essere prodotte in ciascun impianto e come esse devono essere ripartite tra i magazzini e i centri di distribuzione in modo da massimizzare il profitto netto complessivo.*

distribuzione.mod

```
set IMP;
set MAG;
set DIS;

param Offerta{IMP};
param Costo_trasp1{IMP,MAG};
```

```
param Domanda_mag{MAG};
param Prezzo_mag{MAG};
param Costo_trasp2{MAG,DIS};
param Domanda_dis{DIS};
param Prezzo_dis{DIS};

var x{IMP,MAG} >= 0;
var y{MAG,DIS} >= 0;
var z{m in MAG} >= Domanda_mag[m];

maximize profitto:
    sum{m in MAG} (Prezzo_mag[m]*z[m] +
                sum{d in DIS} Prezzo_dis[d]*y[m,d]) -
    sum{i in IMP,m in MAG} Costo_trasp1[i,m]*x[i,m] -
    sum{m in MAG,d in DIS} Costo_trasp2[m,d]*y[m,d];

subject to offerta_imp {i in IMP}:
                    sum{m in MAG} x[i,m] <= Offerta[i];

subject to transito {m in MAG}:
    sum{i in IMP} x[i,m] = sum{d in DIS} y[m,d] + z[m];

subject to domanda_dis {d in DIS}:
    sum{m in MAG} y[m,d] >= Domanda_dis[d];
```

distribuzione.dat

```
set IMP := I1 I2;
set MAG := M1 M2;
set DIS := D1 D2 D3 D4;

param:  Offerta :=
I1      3000
I2      2000;

param Costo_trasp1:  M1  M2 :=
I1                   21  25
I2                   27  22;

param:  Domanda_mag  Prezzo_mag :=
M1      500          150
M2      500          170;
```

```
param Costo_trasp2:  D1  D2  D3  D4 :=
M1                   33  31  36  30
M2                   27  30  28  31;

param:  Domanda_dis  Prezzo_dis :=
D1      150          350
D2      190          280
D3      220          200
D4      170          270;
```

Esercizio 9.1.5 *Un'industria produce computer utilizzando due fabbriche (F1, F2) dislocate in due differenti regioni. Si prende in considerazione un particolare modello di computer per la cui produzione queste due fabbriche hanno le seguenti caratteristiche (costo di produzione unitario in euro), numero massimo di computer che si possono produrre settimanalmente)*

	F1	*F2*
costo di produzione	*450*	*490*
capacità massima	*800*	*100*

Ogni computer dopo la fabbricazione viene trasportato a centri di imballaggio dove viene preparato per la vendita. L'industria dispone di 3 centri di imballaggio (C1, C2, C3) che possono essere utilizzati. La tabella che segue riporta i costi di trasporto di un computer da ciascuna fabbrica a ciascun centro di imballaggio (in euro), insieme al numero di operai necessari in ogni centro, al numero di ore necessarie per imballare un computer e alla capacità massima di ciascun centro:

	C1	*C2*	*C3*
F1	*10*	*9.5*	*8*
F2	*7.5*	*8.5*	*9*
numero operai	*10*	*8*	*9*
ore per l'imballaggio	*1.5*	*1.1*	*1.2*
capacità massima	*400*	*500*	*450*

I centri di imballaggio hanno una capacità limitata in quanto ogni operaio lavora 35 ore settimanali. Costruire un modello lineare che permetta di formulare un piano settimanale della produzione di questa industria che permetta di decidere settimanalmente quali centri di imballaggio attivare in modo da massimizzare il profitto netto complessivo sapendo che tutti i computer trasportati ai centri settimanalmente vengono venduti nella settimana stessa e che il prezzo di vendita

è di euro 900 ogni computer. Tenere inoltre conto del fatto che al più due dei tre centri possono essere attivati in una stessa settimana.

computer.mod

```
set FABBRICHE;
set CENTRI;

param costo_prod{FABBRICHE};
param capacita_max_fab{FABBRICHE};

param costo_trasp{FABBRICHE,CENTRI};
param capacita_max_centri{CENTRI};
param dispon_oraria{CENTRI};
param tempo_imball{CENTRI};
param prezzo;

var x{FABBRICHE,CENTRI}>=0, integer;
var d{CENTRI} binary;

maximize profitto : prezzo*sum{i in FABBRICHE, j in CENTRI}
    x[i,j]-sum{i in FABBRICHE} (costo_prod[i]*sum{j in CENTRI}
    x[i,j])-sum{i in FABBRICHE, j in CENTRI}
    costo_trasp[i,j]*x[i,j];

s.t. capacita_fab{i in FABBRICHE} : sum{j in CENTRI} x[i,j]
                                    <= capacita_max_fab[i];
s.t. ore_disponibili{j in CENTRI} :
        tempo_imball[j]*sum{i in FABBRICHE} x[i,j]
                                <= dispon_oraria[j];
s.t. vincoli_logici{j in CENTRI} : sum{i in FABBRICHE} x[i,j]-
                            capacita_max_centri[j]*d[j]<=0;
s.t. numero_centri : sum{j in CENTRI} d[j]<=2;
```

computer.dat

```
set FABBRICHE := F1 F2;
set CENTRI C1 C2 C3;
```

```
param : costo_prod capacita_max_fab :=
F1 450 800
F2 490 100;

param costo_trasp : C1 C2 C3 :=
F1 10  9.5 8
F2 7.5 8.5 9
;

param : capacita_max_centri dispon_oraria tempo_imball :=
C1   400   350  1.5
C2   500   280  1.1
C3   450   315  1.2
;

param prezzo := 900;
```

9.2 ESERCIZI PROPOSTI

Esercizio 9.2.1 *Un'industria produce un preparato chimico utilizzando due impianti di produzione* **I1**, **I2**. *Da questi impianti tutto il preparato chimico prodotto viene trasportato in due magazzini* **M1**, **M2** *che si trovano in differenti località. In questi magazzini una parte del preparato è venduta all'ingrosso direttamente, un'altra parte viene inviata a quattro centri di distribuzione* **D1**, **D2**, **D3**, **D4** *che effettuano la vendita al minuto. Questi centri necessitano ripettivamente di almeno 150, 190, 220, 170 quintali di preparato chimico che vendono rispettivamente a Euro 350, 280, 200, 270 al quintale. La tabella che segue riporta i costi (in Euro) necessari per trasportare un quintale di preparato da ciascun impianto a ciascun magazzino.*

	M1	**M2**
I1	21	25
I2	27	22

Nella seguente tabella si riportano i costi (in Euro) necessari per trasportare un quintale di preparato chimico da ciascun magazzino a ciascun centro di distribuzione.

	D1	**D2**	**D3**	**D4**
M1	33	31	36	30
M2	27	30	28	31

L'impianto di produzione **I1** *può fabbricare al piú 3000 quintali di preparato, l'impianto* **I2** *può fabbricare al piú 2000 quintali di preparato. I prezzi di vendita all'ingrosso effettuati presso i magazzini* **M1** *e* **M2** *sono rispettivamente di Euro 150 e 170 al quintale. Per ragioni commerciali i quantitativi di preparato chimico venduti all'ingrosso in ciascun magazzino devono essere pari ad almeno 500 quintali ed inoltre tutto il preparato contenuto nei magazzini dovrà essere o venduto o trasportato ai centri di distribuzione per non avere rimanenze non vendute. Si vuole costruire un modello lineare che permetta di determinare le quantità di preparato chimico che devono essere prodotte in ciascun impianto e come esse devono essere ripartite tra i magazzini e i centri di distribuzione in modo da massimizzare il profitto netto complessivo.*

1. *Scrivere* in forma parametrica *i file* `.mod`, `.dat` *e il file* `.run` *che realizzano un'implementazione in* AMPL *di questo problema.*

2. *Senza modificare il file* `.mod`, *determinare la soluzione ottima del problema ottenuta imponendo che non ci sia vendita all'ingrosso in nessuno dei due magazzini.*

3. *Scrivere un file* `.run` *contenente uno* script *in* AMPL *che permetta di risolvere più volte il problema variando ogni volta il quantitativo di preparato chimico venduto all'ingrosso in ciascuno dei due magazzini ponendolo pari a* 100, 200, 300, 400 *e* 500 *quintali con istruzioni di visualizzazione dei risultati ottenuti.*

Esercizio 9.2.2 *Il prodotto finale di una fabbrica è ottenuto raffinando materie prime grezze e miscelandole insieme. Queste materie prime possono essere di due categorie: naturali e sintetizzate. In particolare, sono disponibili tre materie prime naturali (***N1***, **N2**, **N3***) e due materie prime sintetizzate (***S1***, **S2***). Le materie prime naturali e quelle sintetizzate richiedono differenti linee di produzione. Ogni settimana è possibile raffinare non più di 500 quintali di materie prime naturali e non più di 300 quintali di materie prime sintetizzate. Si assume che non ci sia perdita di peso durante la raffinazione e che si possa trascurare il costo di raffinazione. Inoltre esiste una restrizione tecnologica sulla gradazione del prodotto finale: nell'unità di misura in cui questa gradazione è misurata, essa deve essere tra 2 e 7; si assume che tale gradazione nella miscela finale dipenda linearmente dalle singole gradazioni delle materie prime componenti. Nella tabella che segue è riportato il costo (in Euro) per quintale e la gradazione delle materie prime grezze.*

	N1	**N2**	**N3**	**S1**	**S2**
costo	160	180	150	200	230
grad.	6.0	1.9	8.5	5.0	3.5

Il prodotto finale viene venduto a 350 Euro per quintale. Si vuole costruire un modello lineare per determinare la pianificazione della produzione settimanale in modo da massimizzare il profitto netto.

1. *Scrivere* in forma parametrica *i file* `.mod`, `.dat` *e il file* `.run` *che realizzano un'implementazione in* AMPL *di questo problema.*

2. *Modificare il file* `.mod` *in modo da implementare la richiesta che solamente una delle materie prime naturali sia presente nella miscela.*

3. *Utilizzando il modello fino ad ora implementato, modificare* solamente *il file* `.run` *in modo che dopo aver risolto il problema, risolva ancora il problema:* a) *senza il vincolo introdotto al punto precedente e senza i vincoli sulla gradazione finale;* b) *con il vincolo introdotto al punto precedente ma senza i vincoli sulla gradazione finale;*

Appendice A: NEOS Server

A.1 NEOS SERVER FOR OPTIMIZATION

È disponibile un "free Internet-based service" per risolvere problemi di ottimizzazione. Si tratta del *Network-Enabled Optimization System* (NEOS) Server al quale si può accedere dall'indirizzo `http://www.neos-server.org`. È un servizio gratuito che offre la possibilità di risolvere una grande varietà di problemi di ottimizzazione mettendo a disposizione numerosi solutori che rappresentano lo stato dell'arte degli algoritmi di soluzione.
L'accesso ai solutori è molto semplice in quanto viene fornita la possibiltà di sottomettere un problema scritto in alcuni formati standard, fra i quali AMPL. In questo caso l'utente, dopo aver scelto il solutore che vuole utilizzare, esegue un upload dei file `.mod` e `.dat` (e opzionalmente del file `.run`) relativo ad un problema e il NEOS Server collegherà il problema al solutore e determinerà la soluzione ottima. Sono disponibili solutori per le seguenti classi di problemi di ottimizzazione:

- Bound Constrained Optimization
- Combinatorial Optimization and Integer Programming
- Complementarity Problems
- Global Optimization
- Linear Network Programming

- Linear Programming
- Mixed Integer Linear Programming
- Mixed Integer Nonlinearly Constrained Optimization
- Mixed-Integer Optimal Control Problems
- Nonlinearly Constrained Optimization
- Nondifferentiable Optimization
- Semidefinite Programming
- Semi-infinite Optimization
- Stochastic Linear Programming
- Second Order Conic Programming
- Unconstrained Optimization

Bibliografia

Fourer, R., Gay, D., Kernighan, B. (2003). AMPL *A Modeling language for Mathematical Programming*. Duxbury Thomson, USA. Disponibile gratuitamente su sito `http://www.ampl.com`.

IBM ILOG AMPL (2010). *User's Guide. Standard (Command–line) Version Including CPLEX Directives*. IBM.

Indice