Capitolo 5

La classe Transportation

In questo capitolo l'attenzione sarà concentrata sugli strumenti necessari per risolvere il modello di trasporto. Tale problema è un problema di flusso multicommodity di costo minimo; pertanto, verranno inizialmente approfonditi gli aspetti basilari di questa categoria di problemi (che in seguito abbrevieremo con la sigla MMCF). MMCF è da tempo oggetto di studio al Dipartimento di Informatica [Fr-97]; tali lavori hanno prodotto esperienze e codici che sono stati utili per la nostra tesi. In particolare, abbiamo sfruttato la classe di interfaccia MMCFClass e la classe Graph per la risoluzione del problema di trasporto. Tali classi consentono di interfacciarsi con un risolutore di problemi MMCF senza conoscerne i dettagli. Pertanto, il codice da noi sviluppato potrebbe in futuro utilizzare solutori diversi rispetto a quello da noi utilizzato, CPLEX 6.0, che è un generico solutore di problemi di P.L. Intera.

5.1 Problemi di flusso multicommodity

I problemi di flusso multicommodity sono una generalizzazione dei problemi di flusso single-commodity in cui flussi di natura diversa (commodities) coesistono in una data rete. Ogni commodity ha le sue equazioni di conservazione del flusso, ma i diversi flussi sono in competizione per le risorse, che sono rappresentate dalla capacità degli archi. Diamo ora una definizione analitica di questa classe di problemi.

Dato un grafo diretto $G(N, A)$, con n nodi e m archi, e l'insieme $H = \{ 1\ldots. h \}$,un flusso multicommodity su G è un vettore $\mathbf{x} = [\mathbf{x}^1 \ldots \mathbf{x}^h]$ di h distinti vettori di flusso $\mathbf{x} : A \rightarrow \Re$ .La formulazione del problema è la seguente:

(MMCF) $\min \sum_h \sum_{i,j} c_{ij}^h x_{ij}^h$

$$\sum_j x_{ij}^h - \sum_j x_{ji}^h = b_i^h \quad \forall (i,h) \qquad \text{(a)}$$

$$0 \le x_{ij}^h \le u_{ij}^h \quad \forall (i,j,h) \qquad \text{(b)}$$

$$\sum_h x_{ij}^h \le u_{ij} \quad \forall (i,j) \qquad \text{(c)}$$

la quale richiede l'instradamento delle h commodities su G al minimo costo totale (lineare), rispettando sia i vincoli di bilancio ai nodi e i vincoli di capacità individuale $0 \le x_{ij}^h \le u_{ij}^h$, che i vincoli di capacità mutua $\sum_h x_{ij}^h \le u_{ij}$.

Nel nostro problema i flussi di diversa natura che circolano sulla rete sono i diversi tipi di legno prodotti dalle origini nei vari intervalli di tempo in cui queste sono state suddivise. Se riprendiamo la Fig.2.2 del Cap.2, possiamo vedere che non tutti gli archi del grafo sono soggetti a vincoli di capacità mutua; in particolare, gli unici archi interessati a questo vincolo sono gli archi (o, o') i quali indicano quanti camion di prodotto vengono caricati in un certo intervallo di tempo e da una data origine. Il vincolo di capacità mutua su questi archi, che rispecchia una restrizione del problema, consente il caricamento di un solo camion.

5.2 La classe Graph

Questa classe può essere considerata di «servizio» per solutori di problemi di flusso multicommodity di costo minimo. Essa, infatti, fornisce un metodo per la lettura e la memorizzazione della descrizione di problemi MMCF. La classe Graph rappresenta quindi una interfaccia che può essere utilizzata da un qualunque solutore di problemi di questo tipo.

Essa permette, inoltre, di avere un controllo completo dei dati delle istanze (leggerli o modificarli) ed è prevista una fase di pre-processing che ha lo scopo di rendere la specifica istanza del problema più semplice da risolvere, ad esempio identificando e successivamente eliminando i vincoli di capacità mutua che risultano ridondanti.

E' importante sottolineare che in molte applicazioni reali, quali la nostra, il problema MMCF rappresenta una parte di un problema più complesso di programmazione lineare intera. In questo caso è richiesto che il flusso circolante sul grafo sia intero.

Un tipico esempio può essere quello in cui sono presenti costi fissi di carico (Fixed Charge) sugli archi, ossia costi fissi di costruzione degli archi, che il flusso di ogni commodity può attraversare.

La classe Graph prevede anche una eventualità di questo genere, supporta infatti anche grafi con costi fissi di carico sugli archi.

I costruttori presenti nella classe sono due. Il primo permette di leggere la descrizione delle istanze MMCF da file in vari formati : una volta letta, l'istanza è disponibile in memoria , quindi il solutore può leggerla più facilmente in un unico modo, indipendentemente dal formato originario (ci sono infatti diversi formati usati per rappresentare problemi MMCF). Il secondo costruttore della classe permette di avere in input una istanza direttamente dalla memoria. In tal caso l'utente della classe deve opportunamente definire delle strutture dati che descrivono le caratteristiche del grafo che si vuole costruire. Le strutture necessarie sono le seguenti:

- n : numero dei nodi della rete;
- m : numero degli archi della rete;
- comm : numero delle commodities;
- Def : matrice (comm x m) dei deficit ai nodi, in particolare Def[k][i] è il deficit del nodo i relativo alla commodity k;
- S : vettore m-dimensionale dei nodi testa degli archi;
- E : vettore m-dimensionale dei nodi coda degli archi;
- CapTot : vettore m-dimensionale delle capacita' mutue degli archi;
- Cap : matrice (comm x m) delle capacità degli archi. Cap[k][i] è la capacità superiore dell'arco i relativamente alla commodity k;
- Cost : matrice (comm x m) dei costi degli archi. Cost[k][i] è il costo dell'arco i relativamente alla commodity k;
- Drct : è una variabile booleana che deve essere settata a TRUE se il grafo è orientato, FALSE altrimenti.
- FxCCost : vettore m-dimensionale dei costi fissi di carico sugli archi. Se FxCCost = NULL allora questi costi non sono presenti.

Una volta creato l'oggetto, indipendentemente dal costruttore utilizzato, i dati dell'istanza MMCF possono essere agevolmente letti per mezzo di metodi pubblici quali "GetCost( )". I dati possono anche essere modificati, ed esistono metodi come "PreProcess( )" e "MakeSingleSourced( )" che permettono utili operazioni sull'istanza.

5.3 La classe MMCFClass

E' una classe base astratta che definisce l'interfaccia per solutori dei problemi MMCF. Essa definisce solamente i metodi che permettono al solutore di leggere la descrizione dell'istanza attraverso l'oggetto «grafo», descritto nella sezione precedente, risolvere il problema, leggere i risultati e cambiare i dati se necessario. In particolare questa interfaccia è usata dai solutori dei seguenti problemi:

- problemi di flusso multicommodity lineari continui;
- rilasciamenti del problema precedente come il «Flow Relaxation», ottenuto rilassando i vincoli di capacità mutuale, oppure il «Knapsack Relaxation», ottenuto rilassando i vincoli di conservazione del flusso;
- estensioni del problema come MMCF intero oppure MMCF Fixed Charge.
- Inoltre la classe supporta variabili «extra» in aggiunta alle variabili di flusso multicommodity standard.

Oltre al costruttore della classe, il metodo più importante è il «SolveMMCF(…) », che permette di risolvere il problema. Qui il concetto di «soluzione» varia a seconda del tipo di problema che si esamina, in quanto si può voler minimizzare il valore della funzione obiettivo oppure si cerca la soddisfattibilità dei vincoli..

Il metodo restituisce una variabile che comunica all'utente l'esito della chiamata al solutore, questa variabile può assumere i seguenti valori:

- kOK : se l'ottimizzazione ha avuto successo;
- kStopped : se il procedimento di ottimizzazione è stato arrestato in base alle condizioni di arresto che il «solver» applica. Ad esempio perchè è stato raggiunto il numero massimo di iterazioni, il che non necessariamente significa che ci sono errori nella procedura.
- kUnfeasible: se l'istanza MMCF che si sta esaminando è (primale) inammissibile;
- kUnbounded: se l'istanza MMCF che si sta esaminando è (primale) illimitata;
- kError: ci sono stati errori (dipendenti dall'implementazione).

L'interfaccia consente anche di leggere e modificare i dati dell'istanza e tutte le informazioni relative alla soluzione (primale e duale) del problema: ciò avviene attraverso metodi pubblici che sono implementati dalle classi derivate, come descritto nel prossimo paragrafo.

5.4 La classe MMCFCplex

Vengono qui implementati i metodi definiti nella classe base utilizzando funzioni appartenenti alla «Callable Library» della versione 6.0 di CPLEX [CPLEX®].

5.4.1 CPLEX

Si tratta di uno strumento per la risoluzione di problemi di ottimizzazione lineare, meglio conosciuti come problemi di programmazione lineare, della forma:

(Massimizzazione o Minimizzazione) $\mathbf{c_1x_1 + c_2x_2 + \ldots + c_nx_n}$

Vincoli
$\mathbf{a_{11}x_1 + a_{12}x_2 + \ldots + a_{1n}x_n \sim b_1}$
$\mathbf{a_{21}x_1 + a_{22}x_2 + \ldots + a_{2n}x_n \sim b_2}$
$\mathbf{\ldots}$
$\mathbf{\ldots}$
$\mathbf{a_{m1}x_1 + a_{m2}x_2 + \ldots + a_{mn}x_n \sim b_m}$

Bounds
$\mathbf{l_1 \le x_1 \le u_1}$
$\mathbf{\ldots}$
$\mathbf{l_n \le x_n \le u_n}$

dove ~ può essere **≤**, **≥**, oppure **=**, mentre le limitazioni superiori e inferiori (rispettivamente u ed l) possono essere un qualunque numero reale, infinito negativo o anche infinito positivo. I termini noti di questo problema sono:

coefficienti della funzione obiettivo	**(c_1,…,c_n)**
coefficienti dei vincoli	**(a_{11},…a_{mn})**
RHS	**(b_1,…b_m)**
limiti sup. e inf.	**(u_1,…,u_n** e **l_1,…l_n)**

L'oggetto su cui CPLEX lavora è quindi una matrice, mentre il nostro problema è sotto forma di grafo di flusso multicommodity.

Il nostro lavoro è stato quello di realizzare una interfaccia tra il nostro grafo e CPLEX: attraverso questa interfaccia a CPLEX viene passata la matrice di incidenza corrispondente al grafo, ossia un oggetto della forma vista in 4.4.1. Per risolvere il problema, per leggere i dati, o per cambiarli, sono stati definiti dei metodi i quali a loro volta effettuano delle chiamate a routines di CPLEX.
Vediamo più in dettaglio come si passa dal grafo di flusso alla matrice. La matrice dei vincoli presenta una forma molto sparsa, ossia una alta percentuale di coefficienti uguali a zero; ci sarebbe quindi un grosso spreco di memoria se fosse necessario fornire in input a CPLEX tutti questi coefficienti. Per questo motivo CPLEX consente di specificare solamente i valori diversi da zero della matrice. Si tratta di un formato «column-oriented» il che significa che il problema è specificato per colonna (quindi specificando le variabili del problema) anziché per riga (quindi attraverso i vincoli).
La rappresentazione del problema si appoggia sulle seguenti strutture dati:
objsen : intero che indica se il problema è un problema di massimizzazione o di minimizzazione.
obj : array contenente i coefficienti della funzione obiettivo;
rhs : array contenente i valori costanti (RHS) per ogni vincolo della matrice;
sense : array contenente il senso di ogni vincolo della matrice. In particolare:

sense[i] = 'L' vincolo ≤;
sense[i] = 'E' vincolo =;
sense[i] = 'G' vincolo ≥;
sense[i] = 'R' «ranged constraint»;

lb : array contenente il limite inferiore per ogni variabile
ub : array contenente il limite superiore per ogni variabile
matbeg, matcnt, matind, matval : questi quattro vettori descrivono i coefficienti dei vincoli della matrice Questi coefficienti sono raggruppati per colonna, e vengono memorizzati nell'array matval. Ogni colonna deve essere memorizzata in locazioni sequenziali di questo array, inoltre matbeg[j] contiene l'indice dal quale comincia la colonna j, e matcnt[j] contiene il numero di elementi (non-zero) presenti nella colonna . I componenti del vettore matbeg devono essere in

ordine ascendente: per ogni k matind[k] indica il numero di riga del corrispondente coefficiente matval[k].

Il costruttore della classe MMCFCplex a partire dal grafo di flusso che prende come parametro (si tratta di un oggetto della classe Graph) costruisce le strutture descritte. Il primo problema che viene affrontato nel costruttore è il dimensionamento di queste strutture. Il vettore dei costi (obj) ha tanti elementi quanti sono gli archi del grafo (se un arco è condiviso da più commodities, nella matrice questo arco è replicato; se un arco non esiste per una certa commodity il suo costo risulterà uguale a C_INF).
Nel caso in cui il grafo non è orientato, questo numero va raddoppiato in quanto ogni arco può essere percorso in entrambi i sensi.
La dimensione del vettore dei termini noti (rhs) è data dal numero di righe della matrice; poiché il numero di nodi viene replicato per ogni commodity (Fig._) risulta che il numero di righe della matrice è ottenuto sommando le seguenti quantità:

- numero di nodi moltiplicato per il numero di commodity;
- numero di vincoli attivi, ossia numero di archi che hanno associato il vincolo di capacità mutua.

Il vettore dei segni (sense) ha la stessa dimensione di rhs. Matind e matval hanno tanti elementi quanti i coefficienti diversi da zero della matrice. Per ogni arco ci sono due elementi («1» e «-1»), ed essendoci una colonna per ogni arco risulta che il numero di elementi diversi da zero presenti nella parte «a blocchi» della matrice è il doppio delle colonne. Inoltre se un arco ha associato il vincolo di capacità mutua c'è un «1» per ogni commodity in cui l'arco esiste (cioè il costo è minore di C_INF). Infine la dimensione di matbeg e matcnt è pari al numero di colonne della matrice.
Una volta dimensionate e definite tali strutture dati vengono chiamate le seguenti routines di CPLEX:

CREAZIONE AMBIENTE

env = CPXopenCPLEXdevelop(….) la quale restituisce un puntatore all'ambiente di Cplex

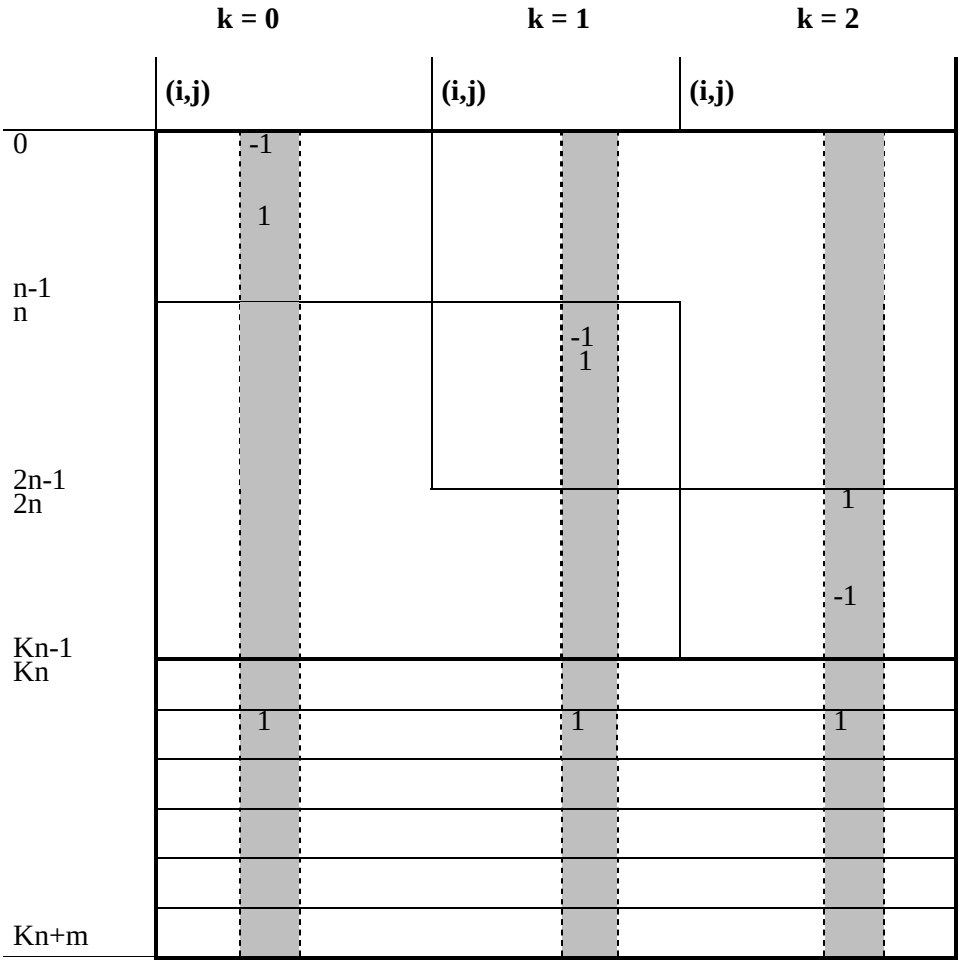
CREAZIONE PROBLEMA OGGETTO

lp = CPXcreateprob(…) la quale restituisce un puntatore che può essere passato ad altre routines di Cplex per identificare il problema oggetto che è stato creato.

DEFINIZIONE PROBLEMA

CPXcopylp(env, lp, numcols, numrows , objsense(= 1), obj, rhs, sense, matbeg, matcnt, matind, matval, lb, ub, NULL**)** la quale copia i dati che definiscono il problema LP in un problema oggetto di CPLEX.

Numero Commodity = k (= 3)
Numero Nodi = n
Numero di archi che hanno associato il vincolo di capacità mutua = m



[Fig.: Matrice da passare a CPLEX]

Attraverso i metodi della classe possiamo ottenere la soluzione del problema passato a CPLEX, oppure modificare o anche leggere i dati del problema quali il vettore dei costi o il vettore dei vincoli di capacità mutua. Andremo ora ad esaminare i metodi più importanti della classe.

METODI DELLA CLASSE

SolveMMCF(…) : a seconda della scelta dell'utente questo metodo può risolvere il problema usando diversi algoritmi risolutivi che CPLEX mette a disposizione, pertanto in questo metodo può venire effettuata una delle seguenti chiamate:

- CPXprimopt(env, lp) che risolve il problema sfruttando il simplesso primale;
- CPXdualopt(env, lp) che risolve il problema col simplesso duale;

- CPXbaropt(env, lp) che risolve il problema con l'algoritmo «barrier», il quale è un'alternativa al metodo del simplesso e genera una sequenza di soluzioni primali e duali strettamente interne al poliedro;
- CPXmipopt(env, lp) che risolve problemi interi misti;
- CPXhybnetopt (env, lp ,'p') la quale riconosce se parte della matrice dei vincoli del problema può essere risolta come un problema di flusso, lo risolve col simplesso per flussi ottenendo così una soluzione di base da cui partire per risolvere l'intero problema con il simplesso primale;
- CPXhybnetopt (env, lp ,'d') che funziona come la precedente, ma usando il simplesso duale dopo quello per flussi.

Questo metodo restituisce lo stato di successo, insuccesso o di errore della chiamata a una delle suddette routines (come visto nella sezione precedente).

GetOptVal(…) : sfruttando questo metodo, che ha senso solo se viene invocato dopo SolveMMCF(…), avremo come risultato il valore ottimo della funzione obiettivo del problema LP ottenuto dalla chiamata CPXgetobjval(env, lp, &objval_p).

GetCosts(…) : permette di leggere i costi associati a tutte le variabili del problema oppure solo ad una parte di esse. Si basa sulla chiamata CPXgetobj(env, lp, …).

GetICaps(…) : permette di leggere le capacità individuali associate a tutte le variabili del problema oppure solo ad una parte di esse, ed è basata sulla chiamata CPXgetobj(env, lp, …).

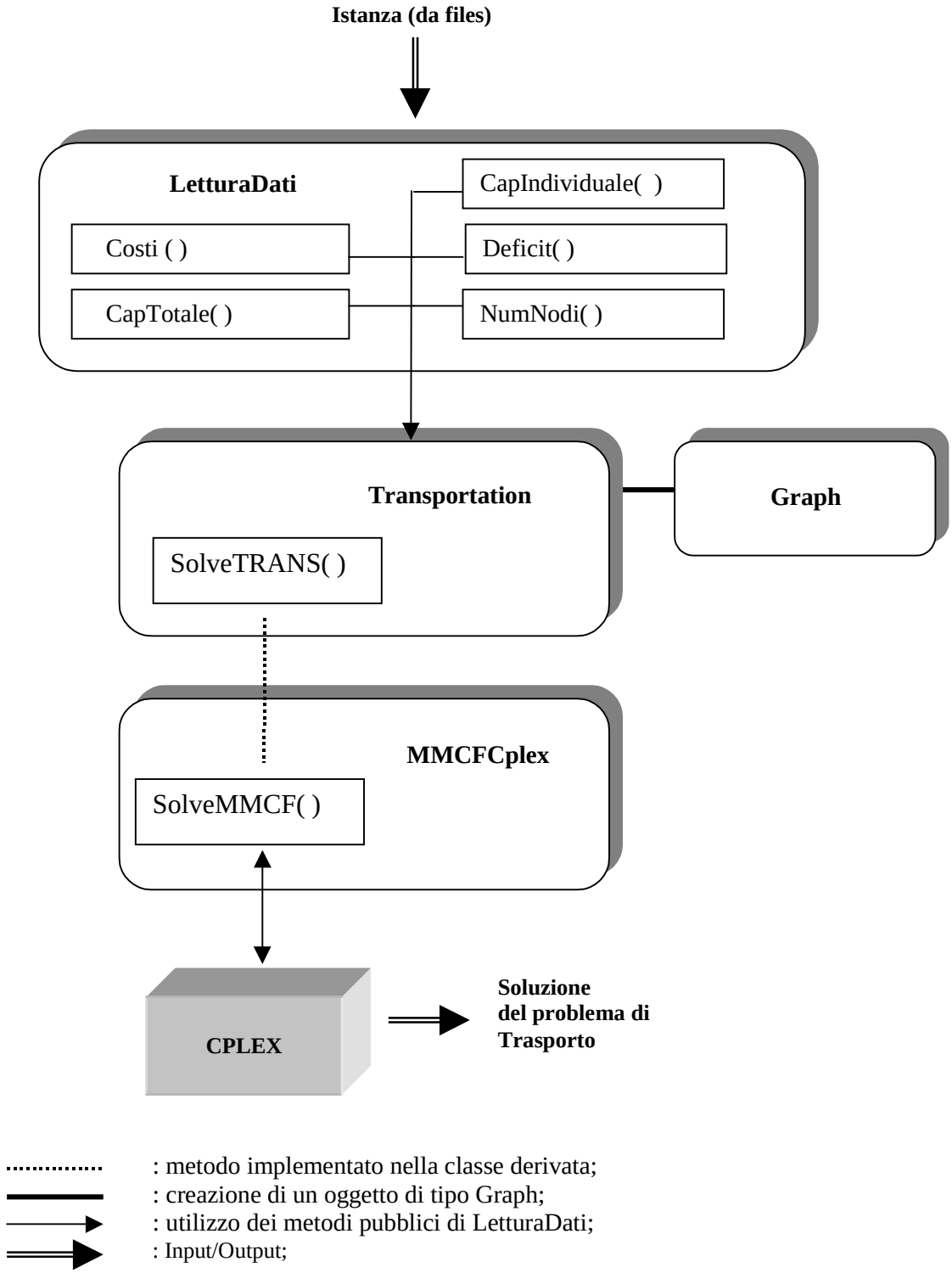
GetMCaps(…) : legge le capacità mutue delle variabili di flusso nell'istanza MMCF; si basa sulla chiamata CPXgetrhs(env, lp,…).

I metodi **:ChgCosts(…)** ,**ChgICaps(…)** ,**ChgMCaps(…)** permettono di modificare rispettivamente i costi, le capacità individuali e le capacità mutue associate a tutti gli archi o solo ad una parte di essi. Si basano rispettivamente sulle chiamate a CPXchgcoef(env, lp, …), CPXchgbds(env, lp, …),CPXchgrhs(env, lp, …).

Come è stato più volte detto, questa classe risolve il nostro modello di trasporto; inoltre, essa può essere sfruttatata da un qualunque utente che si trovi di fronte a problemi dello stesso genere. Infatti è già stata usata con successo da un altro tesista per risolvere una parte del suo modello matematico. Bisogna inoltre notare che tra le varie fasi che hanno portato alla realizzazione di questo strumento, grande importanza va alla fase di «testing» della classe. Il solutore «SolveMMCF( )» è stato testato su un'ampia categoria di istanze MMCF ancor prima che il resto della classe Transportation venisse scritto. Ciò ha permesso un più efficiente test del codice.

5.5 La classe Transportation

In questa classe vengono utilizzate tutte le procedure descritte sinora. Per rendere più chiaro il modo in cui interagiscono tra di loro le diverse classi illustrate abbiamo schematizzato la situazione in Fig,_.



[Fig._ : Schema del problema di trasporto]

Al costruttore della classe Transportation viene passato come parametro un oggetto della classe LetturaDati. Transportation crea le strutture necessarie per la costruzione di un oggetto di tipo Graph; a questo punto viene creato l'oggetto MMCFCplex, al quale viene passato l'oggetto Graph come parametro. Per risolvere il problema la classe Transportation ha a disposizione il metodo "SolveTRANS( )" che risolve il problema MMCF invocando la «SolveMMCF( )» la quale a sua volta sfrutta il solutore CPLEX.

Si noti come la modularizzazione da noi scelta possa permettere di programmare facilmente approcci alternativi. Se, ad esempio, si volessero sperimentare altri tipi di solutori diversi da CPLEX aventi la stessa interfaccia MMCFClass, le modifiche alla classe Transportation sarebbero minime, mentre LetturaDati rimarrebbe invariata.