Capitolo 6

La classe Scheduling

Esaminiamo ora l'implementazione del modello di schedulazione. Come abbiamo visto nel Cap.2 bisogna risolvere R problemi di flusso di costo minimo. Ci siamo basate su un solutore di problemi di flusso di costo minimo, RelaxIV, che descriveremo insieme alla sua interfaccia astratta, McfClass. Il ruolo della classe Scheduling è quello di creare un oggetto nella forma in cui il RelaxIV lo richiede. In particolare, in base ai viaggi risultanti dalla soluzione del modello di trasporto, verrà creato un array di puntatori ad oggetti di tipo RelaxIV e un metodo "SolveSCHED( )" che con opportune chiamate al solutore Mcf risolve gli R problemi di assegnamento.

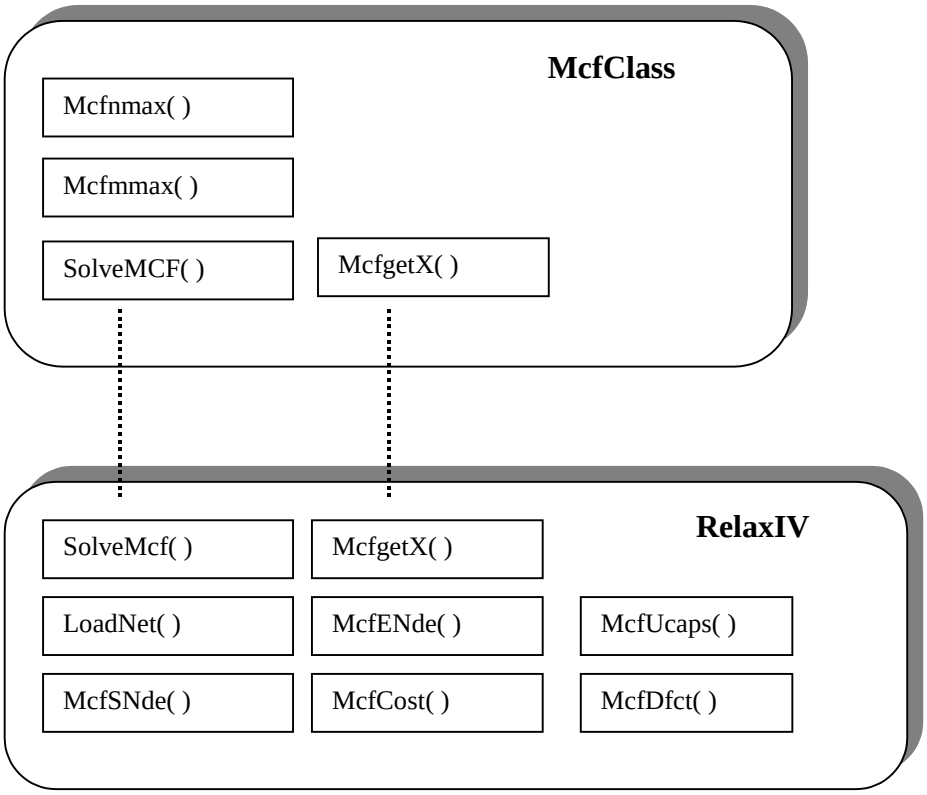
6.1 La classe McfClass e la classe RelaxIV

Il ruolo di questa classe per risolvere problemi di flusso di costo minimo può essere considerato equivalente a quello della classe MMCFClass nella risoluzione di problemi MMCF.

Si tratta, infatti, di una classe base astratta che definisce un'interfaccia tra un generico solutore (MCF) e i programmi applicativi, con lo scopo di rendere questi ultimi il più possibile indipendenti dai dettagli del particolare solutore che viene utilizzato.

In tale interfaccia è presente un insieme di tipi "virtualizzati" con la funzione di consentire una maggiore flessibilità nella scelta dei tipi (interi o "floating") e nella precisione dei numeri (costi, flussi, indici,…) e soprattutto con lo scopo di rendere il codice estremamente portatile rispetto alla specifica macchina o ad una particolare applicazione.

I metodi definiti in questa classe sono implementati nella classe RelaxIV ossia il vero e proprio solutore di problemi MCF; in Fig.6.1 è schematizzata l'interazione tra le due classi. Il grafo di flusso che viene definito tramite il costruttore della classe RelaxIV non necessariamente deve conservare la sua topologia durante le diverse fasi delle applicazioni; è infatti possibile cambiarne la struttura, inserendo e togliendo archi o anche distruggendo e inserendo nuovi nodi, durante l'esecuzione.



[Fig.6.1:Schematizzazione delle classi McfClass e RelaxIV]

Al costruttore RelaxIV devono essere necessariamente passate le seguenti informazioni:

- pnmax: massimo numero di nodi della rete;
- pmmax: massimo numero di archi della rete

Tutti gli altri parametri che descrivono la rete iniziale sono opzionali: essi possono essere passati tramite una o più chiamate del metodo “LoadNet( )” che andremo a descrivere.

Ovviamente tutti i dati che descrivono la rete devono essere memorizzati prima della chiamata del solutore. Il ruolo del costruttore RelaxIV è solo quello di allocare strutture dati della massima dimensione calcolata sulla base di n_max ed m_max.

Descriviamo ora le strutture che devono essere definite e poi passate al metodo “LoadNet( )”:

- pn : numero dei nodi della rete;
- pm : numero degli archi della rete;
- pU : vettore m-dimensionale delle capacità superiori degli archi: le capacità devono essere ≥ 0 e “finite”, ossia più piccole della quantità F_INF;

- pC : vettore m_dimensionale dei costi degli archi;
- pDfct : vettore n-dimensionale dei deficit ai nodi; i nodi "sorgente" devono avere deficits negativi mentre i nodi "pozzo" hanno deficits positivi;
- pSn : vettore m-dimensionale dei nodi testa degli archi;
- pEn : vettore m-dimensionale dei nodi coda degli archi.

Una volta memorizzata la rete il metodo per risolvere il problema MCF è il "SolveMCF( )" basato sull'implementazione di un algoritmo ideato da D.Bertsekas [Be-91] per la risoluzione di problemi di flusso di costo minimo.

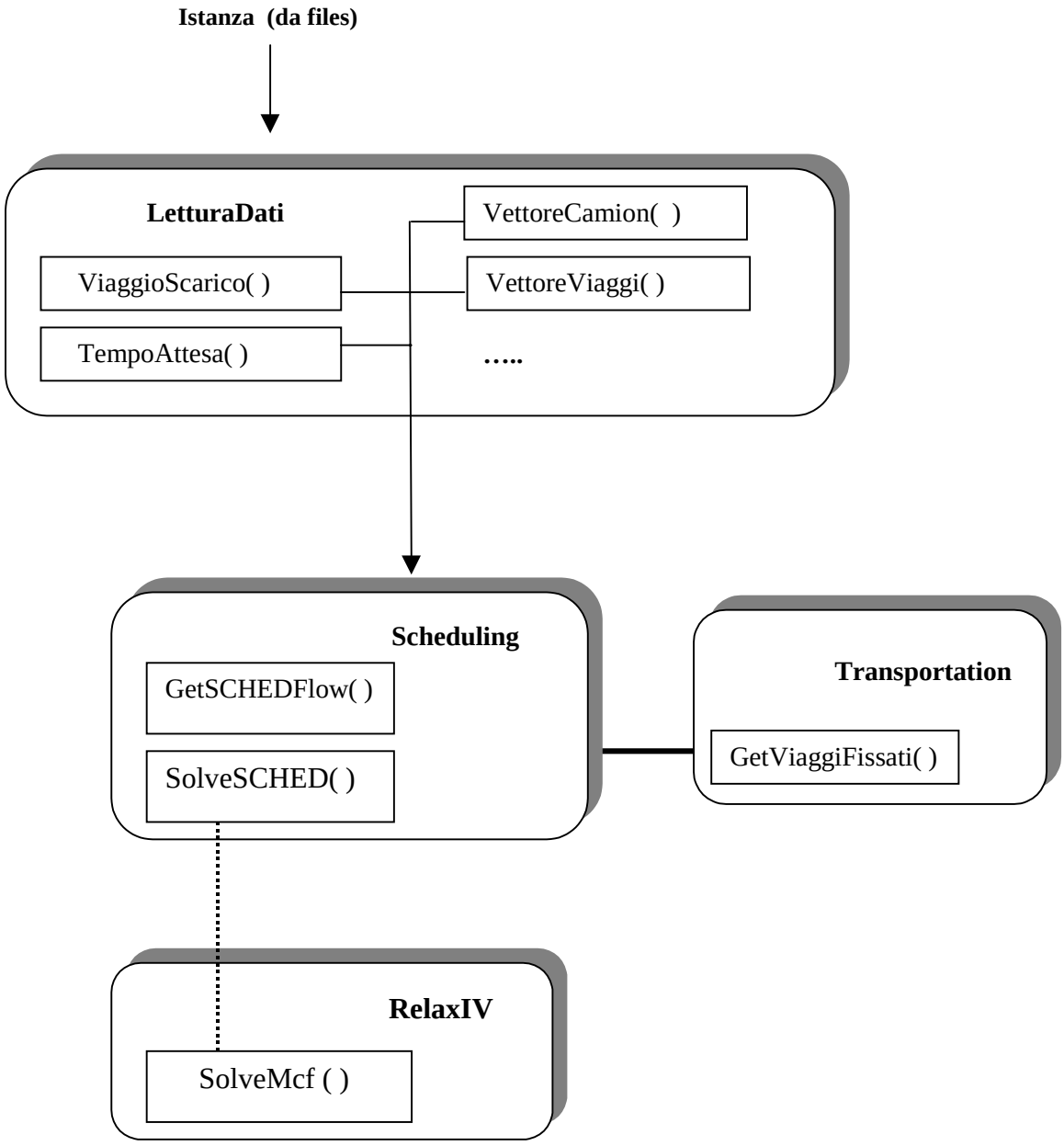
Il metodo "SolveMCF( )" restituisce un valore che indica lo stato di successo o insuccesso della chiamata al metodo stesso. In particolare se il valore restituito è 0 questo vuol dire che è stata trovata una soluzione ottima. La soluzione primale e duale può poi essere letta con i metodi …, inoltre metodi come "ChgCosts(…)","ChgDfcts(…)", "ChgCaps(…)" permettono di cambiare rispettivamente i costi, i deficit e le capacità superiori del problema MCF.

6.2 La classe Scheduling

Vengono qui sfruttati gli strumenti finora descritti. Per una maggiore chiarezza in Fig.6.2 è illustrato il modo in cui interagiscono le classi esaminate.

Al costruttore della classe Scheduling viene passato un oggetto di tipo LetturaDati ed un vettore di indici di viaggi ottenuto tramite il metodo "GetViaggiFissati( )" della classe Trasportation (che può essere invocato solo dopo il metodo "SolveTRANS( )"). Dalla classe LetturaDati il costruttore otterrà tutte le informazioni riguardanti i viaggi che nella prima fase del problema si è deciso di compiere. Queste informazioni sono in gran parte memorizzate nella struttura VettoreViaggi (contenente oggetti di tipo ViaggioScarico). Altre verranno ottenute sfruttando i metodi pubblici sempre della classe LetturaDati. Risulta evidente che anche qui, così come era per la classe Transportation, il ruolo della classe LetturaDati è fondamentale affinché le diverse fasi risolutive siano il più possibile scollegate. Sulla base delle informazioni suddette il costruttore Scheduling costruisce un array di oggetti di tipo RelaxIV, uno per ogni tipo di camion.

In seguito con il metodo "SolveSCHED(…)" , che effettua a sua volta una chiamata al solutore MCF, viene calcolata la soluzione ottima per ognuno degli R problemi.

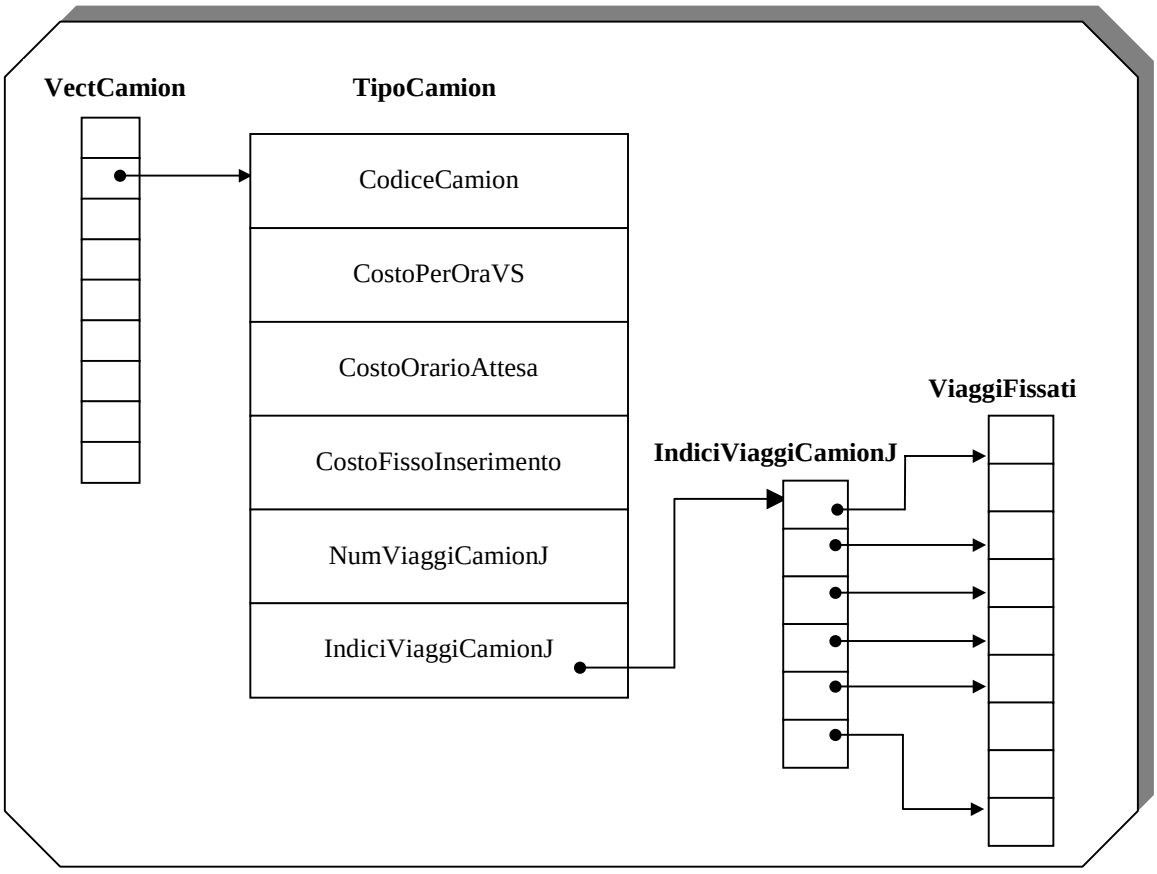


[Fig.6.2 : Schema del problema di assegnamento]

6.3 Costruttore della classe Scheduling

Come abbiamo già accennato il costruttore della classe Scheduling costruisce in grafo per ogni tipo di camion, che in base ai risultati del modello di Trasporto, risulta compiere almeno un viaggio. A partire

dagli indici di viaggi contenuti nel vettore ViaggiFissati, vengono calcolati quali sono (**IndiciViaggiCamionJ**), e quanti sono (**NumViaggiCamionJ**), i viaggi riguardanti ogni camion. Per memorizzare queste ed altre informazioni diverse per ogni tipo di camion viene creata una struttura TipoCamion analoga a quella vista nel Capitolo 4, schematizzata in Fig.6.3.



[Fig.6.3: schema della struttura TipoCamion]

Buona parte di tali campi sono riempiti utilizzando il metodo pubblico "VettoreCamion( )" della classe LetturaDati. Come si è visto nel Cap.2, il modello matematico prevede un **CostoFissoInserimento** che si sostiene nel momento in cui un nuovo mezzo viene inserito nel sistema. Poiché nelle istanze inviateci questa informazione non era presente, abbiamo supposto tale costo pari al costo di quattro ore di attesa: ciò è giustificabile se si pensa che in media, nella realtà, il viaggio dal deposito alle origini e analogamente dalle destinazioni al deposito è di circa due ore.

Un'altra informazione che otteniamo tramite il metodo NumTir( ) della classe LetturaDati è il numero di diversi tipi di camion a disposizione. Il secondo passo è quello di recuperare, tramite il metodo pubblico "VettoreViaggi( )" della classe LetturaDati, la struttura VettoreViaggi, la quale per ogni indice di viaggio possibile contiene tutte le informazioni che lo riguardano e che sono necessarie al modello di Scheduling. A questo punto abbiamo tutto il necessario per costruire un oggetto di tipo RelaxIV, che in questo caso è il grafo bipartito esaminato nel Cap.2.
Il procedimento è il seguente:

```
∀ tipo di camion J
 { ViaggiCamionJ = VectCamion[j].NumViaggiCamionJ
    Se ViaggiCamionJ > 0
     { m = NUMArchiGrafoCamionJ (J);
       n  = NUMNodiGrafoCamionJ(J);
       Defct = CreaVectDeficit(n);
         /*  ARCHI DI USCITA DAL DEPOSITO */
         k = 0;
        from  i  = 0 to i < (n-2)/2
                {Sn[ k ] = n-2;
                 En[ k ] = (n-2)/2 + i;
                 U[ k ] = 1;
                 C[ k ] = 0;}
        /* ARCHI DI RIENTRO AL DEPOSITO */
         from  i  = 0 to i < (n-2)/2
                {Sn[ k ] = i;
                 En[ k ] = n-1;
                 U[ k ] = 1;
                 C[ k ] = 0;}
        /* ARCO "SPECIALE" TRA I NODI DEPOSITO */
                 Sn[ k ] = n-1;
                 En[ k ] = n-2;
                 U[ k ] = ViaggiCamionJ;  /* ponendo come upper-bound il numero dei viaggi
                                             facciamo si che nella più costosa delle soluzioni
                                             venga usato un camion per ogni viaggio da compiere */
                 C[ k ] = VectCamion[j].CostoFissoInserimento;
```

```
    /*ARCHI DI VIAGGIO SCARICO TRA DESTINAZIONE E
      ORIGINE DEL VIAGGIO SUCCESSIVO */

    { from i = 0 to ViaggiCamionJ
          IndiceDes = VectCamion[ j ].ViaggiFissatiCamionJ[ i ];
          IndiceSucc = ( i + 1 ) mod ViaggiCamionJ;

           While ( IndiceSucc ≠ i )
            {
             IndiceOri = VectCamion[ j ].ViaggiFissatiCamionJ[ IndiceSucc ];
             Se Compatibili(i, j )
               {Sn[ k ] = i;
                En[ k ] = IndiceSucc + ViaggiCamionJ;
                U[ k ] = 1;
                C[ k ] = CostoViaggioScarico(j, IndiceOri, IndiceDes ); }
             IndiceSucc = ( IndiceSucc + 1) mod ViaggiCamionJ;
            }
      }

   /* a questo punto per il camion di indice j posso costruire un oggetto RelaxIV */

    ReIV = new RelaxIV(n,m,n,m,U,C,Dfct,Sn,En);
    R[ j ] = ReIV;
  }

}
```

Nella procedura descritta sono usati i seguenti metodi privati della classe scheduling:

- **NUMArchiGrafoCamionJ(j)** : che calcola il numero totale di archi del grafo relativo al camion di indice j. Questo valore è ottenuto sommando le quantità (A1 + A2 + A3 + D) dove:
    - A1 = Numero di archi di viaggio scarico relativi al camion j ottenuto controllando le condizioni di compatibilità descritte nel Cap.2;
    - A2 = Archi per il viaggio dal deposito alle origini dei viaggi;
    - A3 = Archi per il viaggio dalle destinazioni dei viaggi al deposito;
    - D = 1 è l'arco di collegamento tra i nodi speciali deposito.
- **NUMNodiGrafoCamionJ(j)** : calcola il numero di nodi del grafo relativo al camion di indice j nel seguente modo: NumNodi = (VectCamion[j].NumViaggiCamionJ x 2) + 2: i nodi duplicati sono quelli corrispondenti, mentre i restanti due sono i nodi speciali di deposito.
- **CreaVectDeficit(n)** : che crea il vettore dei deficit ai nodi. Per comprendere meglio il suo funzionamento bisogna sapere come sono stati numerati i nodi: i nodi che vanno da 0 a (n/2)-1 corrispondono ai nodi s_z definiti nel Cap.2 ed a questi è stato associato deficit –1; i nodi che vanno da (n/2)-1 corrispondono ai nodi e_u definiti nel Cap.2 ed a questi viene associato deficit 1; infine i

nodi numerati con (n-1) ed (n-2) sono i nodi speciali di deposito e quindi gli è stato attribuito deficit 0, sempre secondo quanto stabilito dal modello matematico esaminato nel Cap.2.

- **Compatibile(IndiceOri, IndiceDes)** : che controlla se il viaggio scarico tra l'origine IndiceOri e la destinazione IndiceDes rispetta le seguenti condizioni di ammissibilità:

  1. ViaggioScarico(IndiceOri,IndiceDes) < 23.59 (ossia il viaggi non è considerato impossibile);
  2. (VettoreViaggi[IndiceDes].FineIntDes + TempoViaggioScarico) ≤ VettoreViaggi[IndiceOri].InizioIntOri;
  3. VettoreViaggi[IndiceOri].InizioIntOri ≤ (VettoreViaggi[IndiceDes].FineIntDes + TempoViaggioScarico + epsilon;

- **CostoViaggioScarico(IndiceOri, IndiceDes) :** che calcola il costo del viaggio scarico tra IndiceOri e IndiceDes includendo anche il costo dell'eventuale attesa all'origine.

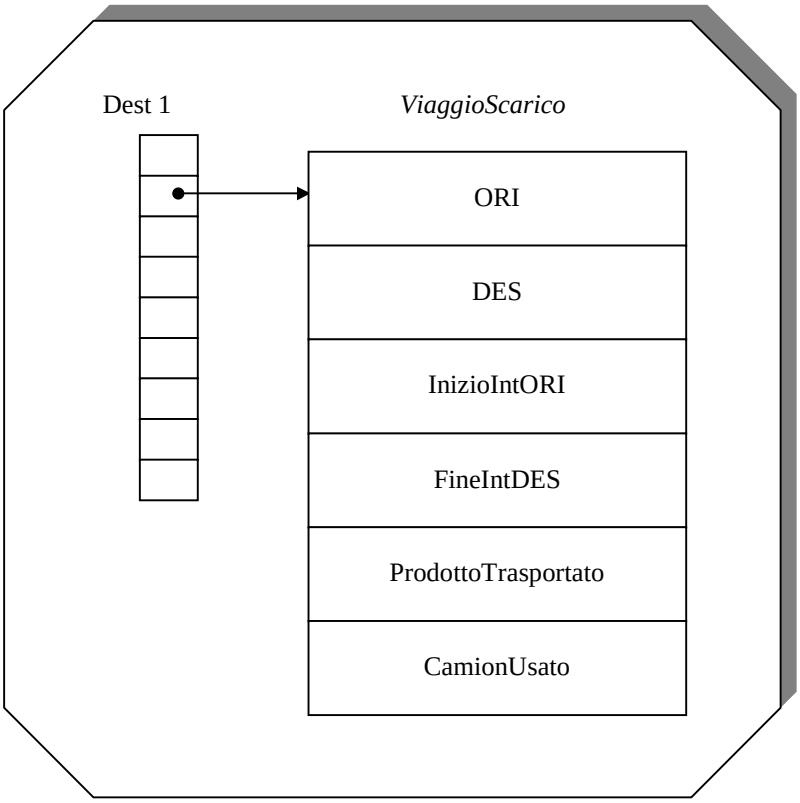
6.4 Metodi pubblici della classe Scheduling

Tramite i metodi pubblici della classe Scheduling possiamo calcolare la soluzione ottima per ogni problema di assegnamento e anche ottenere il vettore del flusso ottimo associato aglia rchi del grafo bipartito associato ad ogni camion. In particolare:

- SolveSCHED(j): che a partire dall'oggetto R[j] creato nel costruttore invoca il metodo SolveMCF(..) per ottenere il valore ottimo del problema;
- Get_SCHED_Flow(j): che dopo che è stato calcolato il flusso ottimo ci permette di leggere i valori del flusso associati ad ogni arco tramite una chiamata al metodo pubblico McfGetX(..) della classe RelaxIV.

6.5 Interfaccia per la classe Scheduling

Come vedremo nel Capitolo 6, quello che serve alla classe Scheduling è l'insieme dei viaggi possibili , ossia l'insieme degli archi origine-destinazione che è stato possibile inserire nel grafo "spazio-tempo" descritto nel paragrafo precedente. Ogni viaggio deve essere correlato da una serie
di informazioni necessarie per la risoluzione del modello di assegnamento: tali informazioni sono state memorizzate nella struttura **ViaggioScarico**, schematizzata in Fig._.



[Fig._ : Schema della struttura ViaggioScarico]