MDMM Project

Matino Ruben
Ritacco Antonio

June 2020

1 Problema (N. 4)

A shipping company has to send a set of orders over a multi-modal transportation network (trains + trucks), represented as a directed graph where each arc has a unit transportation cost and a maximum capacity (total amount of goods that can travel along that arc). Train arcs have far lower transportation cost, have stricter capacity (railways are few w.r.t. roads) and are "longer" (only reach a subset of nodes), while truck arcs have a higher transportation cost but larger capacity and connect all nodes. Each order is represented by a given amount of goods, a starting node and an ending node. To avoid possible misservices to users, all the goods pertaining to the same order have to be shipped together, i.e., along a single path; furthermore, the length of the path must not exceed a prescribed constant k (nor very small, but not very large either). The problem is to find a minimum-cost set of paths with the required property that allow to satisfy all the orders while satisfying the arc capacity constraints.

2 Modello matematico

Il modello matematico che ci è sembrato più adeguato è il Minimum Cost Multicommodity Flow adattandolo al nostro caso ed assumendo che dato un arco esso possa essere o solo di tipo stradale o solo di tipo ferroviario. Inoltre per soddisfare l'unicità dei cammini abbiamo aggiunto un vincolo (6) in cui imponiamo che dato un ordine, ogni nodo appartenente a quel cammino possa avere al più un arco uscente.

2.1 Parametri

- N è l'insieme dei nodi.
- A è l'insieme degli archi.
- H è l'insieme degli ordini
- u_{ij} capacità del collegamento $(i, j) > 0$

- c_{ij} costo del collegamento (i, j)
- d_h quantita' merce relativa ad ordine h

2.2 Variabili Decisionali

- $y_{ij}^h \in \{0, 1\}$ arco di attivazione del collegamento (i, j) relativa all'ordine h

2.3 Modello

$$min(\sum_{(i,j)\in A}\sum_{h=1}^{H} c_{ij} \cdot y_{ij}^h) \tag{1}$$

$$(\sum_j y_{ji}^h) - (\sum_j y_{ij}^h) = b_i^h, \ \ \forall i \in N, h = 1, ...H \tag{2}$$

$$\sum_{(h=1)}^{H} y_{ij}^h \cdot d^h \leq u_{ij}, \ \ \forall (i, j) \in A \tag{3}$$

$$\sum_{(i,j)} y_{ij}^h \leq K, \ \ h = 1, ...H \tag{4}$$

$$\sum_j y_{ij}^h \leq 1, \ \ \forall i \in N, h = 1, ...H \tag{5}$$

$$y_{ij}^h \in \{0, 1\}, \ \ \forall (i, j) \in A, h = 1, ...H \tag{6}$$

- In (1) è presentata la funzione obiettivo. Questa consiste nella minimizzazione, per ogni ordine h, del costo di instradare la merce lungo una tratta composta possibilmente da ferrovie e/o strade.
- In (2) è presentato il vincolo di bilancio di flusso per ogni ordine $h \in H$.

$$b_i = \begin{cases} -1, i = origine \\ 1, i = destinazione \\ 0, altrimenti \end{cases}$$

- Le disequazioni (3) rappresenta i vincoli sulla capacita' degli archi.
- La disequazione (4) modella il vincolo di lunghezza massima del cammino $\leq K$.
- Infine, la disequazione (5) modella il vincolo di unicita' del cammino per ogni commodity.
- In (6) è invece presente il vincolo di interezza per le attivazioni degli archi.

3 Generatore di istanze

La funzione *instance_generator* all'interno di **instance.py** viene utilizzata per la generazione dei files .cmpl mentre la funzione *generator* all'interno dello script file **generator.py** si occupa della generazione dei parametri necessari all'istanza per essere creata generando direttamente il file .cmpl per ogni istanza del problema, con annesso modello matematico. Ogni istanze generata è infatti pronta per essere data in pasto ad un solutore che accetti istanze in formato .cmpl. I nostri test presentati nel paragrafo successivo sono stati effettuati utilizzando i solver *glpk* e *cbc*. In questa sezione ci proponiamo di svolgere, quindi, i punti 2 e 3 del progetto.

3.1 Istance.py

Per ottenere le istanze del problema abbiamo deciso di implementare, utilizzando il linguaggio python, un generatore di file .cmpl che tenesse conto delle specifiche del problema per generare delle istanze realistiche. Il problema richiede infatti, che le strade colleghino ogni coppia di nodi e che le ferrovie siano presenti solo sui collegamenti "lunghi". Il nostro generatore, data una coppia (i, j) crea un arco ferroviario tra i e j solo quando la seguente condizione è verificata:

$$random.uniform[0,1] < \frac{|i-j|}{Nodes-1} \cdot perc\_att\_ferrovie \tag{7}$$

Dove, Nodes è il numero di nodi e $perc\_att\_ferrovie$ è un parametro scelto dall'utente in fase di creazione dell'istanza e compreso tra 0 ed 1. In questo modo, la probabilità che venga a crearsi un collegamento ferroviario tra i e j è tanto più alta quanto più i due nodi sono lontani. La probabilità è uguale ad 1 quando i due nodi sono il primo (1) e l'ultimo (Nodes). L'utente tramite il paramentro $perc\_att\_ferrovie$ è in grado di controllare ulteriormente il numero di ferrovie che possono venire a crearsi.
Il nostro generatore, genera quindi per ogni coppia di nodi, una strada e con probabilità variabile una ferrovia. Per scoraggiare l'utilizzo indiscriminato di strade su collegamenti lunghi (visto che le specifiche del problema indicavano che ogni coppia di nodi dovesse essere collegata almeno da una strada), abbiamo moltiplicato il costo del collegamento (stradale o ferroviario) per $|i-j|$ così che utilizzare più collegamenti, ma corti, abbia un costo generalmente inferiore.
Visto che le specifiche richiedevano che i costi e le capacità delle ferrovie fossero minori delle strade, il generatore di istanze genera i costi e le capacità in maniera randomica ma all'interno di intervalli predefiniti, così da impedire che esista un collegamento stradale tra due nodi con un costo (capacità) minore della corrispettiva ferrovia (se creata).
Altro parametro a cui abbiamo fatto attenzione è K ossia il numero massimo di collegamenti servibili in un solo percorso. Nel nostro generatore K è generato 4 volte per ogni istanza tenendo conto del numero di nodi e facendo in modo di avere K crescenti a partire da valori bassi sino a valori leggermente più alti. In questo modo abbiamo potuto osservare come per una stessa istanza, K

influezzasse il risultato in termini di costo minimo e tempo di esecuzione.
Infine, il numero di ordini H è generato come : $\lceil \frac{Nodes}{2} \rceil$

3.2 Generator.py

Per generare una sequenza di istanze .cmpl basta lanciare lo script generator.py con i seguenti parametri:

- -nodes : Numero di nodi della rete, default [10,50,100]
- -patt : percentuale di attivazione delle ferrovie, default 0.9

Non viene esposto all'utente la possibilità di cambiare i costi e e le capacità minime e massime, così come la demand. Dopo alcuni tests ci è sembrato ragionevole utilizzare i seguenti valori di min e max come range per la generazione di valori random:

- min_demand : 50
- max_demand : 100
- min_costs_strade : 50
- max_costs_strade : 75
- min_costs_ferrovie : 25
- max_costs_ferrovie : 45

Riguardo le capacità delle ferrovie abbiamo scelto di utilizzare il valore *max_demand* come **capacità minima ferroviaria** (così da non rischiare la creazione di ferrovie potenzialmente non utilizzabili per nessun ordine), mentre per la **capacità massima ferroviaria** abbiamo moltiplicato la capacità minima per 1.5. Riguardo le strade, la **capacità minima stradale** è ottenuta sommando 1 alla capacità massima delle ferrovie, mentre la **capacità massima stradale** è ottenuta raddoppiando la capacità minima.

4 Implementazione e risultati

Per testare come il generatore creasse istanze effettivamente più difficili da risolvere al variare dei parametri esposti all'utente, abbiamo lanciato una serie di esperimenti con i due solver : *cbc* e *glpk* ed usando i seguenti parametri del generatore:

- Numero di nodi : 10, 50, 100
- Percentuale di attivazione delle ferrovie : 0.1, 0.5, 0.9
- K : variabile tra 1 e 25 in base al numero dei nodi, ma comunque usando 3 o 4 diversi K per ogni configurazione.

In totale abbiamo generato 66 diverse istanze .cmpl (per la stessa combinazione di $Nodes$ e $perc\_att\_ferr$ ci sono 3 o 4 istanze con gli stessi dati di input e solo K diverso) i cui risultati (completi) sono mostrati in section 6. In generale, abbiamo notato che utilizzando il solver *glpk* la soluzione ottima viene trovata in un tempo inferiore rispetto a quella trovata con il solver *cbc*, fatta eccezione per casi sporadici. Come tempo massimo per la risoluzione del problema abbiamo utilizzato 300 secondi e le caratteristiche del pc sul quale abbiamo lanciato le esecuzioni sono le seguenti:

- S.O. : Windows 10
- Processore : intel core i7-7700HQ 4 core (8 Threads)
- Ram : 16gb ddr4
- S.video : gtx 1060m

La percentuale di attivazione delle ferrovie influenza molto la facilità con la quale è trovata la soluzione ottima. Con 100 nodi e percentuale di attivazione = 0.1 come visibile in table 1, nessuno dei due solvers riesce a trovare la soluzione ottima in un tempo ragionevole tranne che per il caso con $k = 2$ e come solver *glpk*, questo perchè un K così basso rispetto al numero di possibili collegamenti riduce di molto lo spazio di ricerca. Con una percentuale di attivazione delle ferrovie = 0.5 e = 0.9 le istanze da 100 nodi, vengono invece solo risolte da *glpk* come visibile in table 2 e table 3 e con tempi generalmente inferiori. Il parametro k che controlla il numero massimo di archi attivabili per un singolo ordine, abbiamo notato che non influisce molto una volta che diventa sufficientemente grande. I miglioramenti più importanti in termini di costo minimo li abbiamo verificati quando si passava da $k = 2$ a $k = 6$. Probabilmente aumentando la distanza minima tra i nodi sorgente e nodi destinazione avremmo potuto vedere miglioramenti più consistenti in termini di costo minimo all'aumentare di k, ma al momento la generazione dei bilanci degli ordini è lasciata random.
Per quanto riguarda le esecuzioni fallite con 100 nodi (table 1), riducendo il numero di collegamenti stradali ,passando quindi da un grafo completo ad uno sparso (impostando la probabilità di creare un collegamento stradale da 1 ad 0.1), il solver *glpk* riesce in poco tempo a trovare la soluzione ottima. Non abbiamo però incluso questa serie di esperimenti, poichè le istanze non sarebbero state aderenti alla formulazione del problema.

Osservando le tabelle è possibile notare un notevole incremento del tempo di calcolo al crescere delle dimensioni del problema. Per problemi di piccole dimensioni il solver si comporta bene ottenendo soluzioni ottime in tempi accettabili (meno di 300 secondi); tuttavia aumentando le dimensioni del problema, ed in particolare utilizzando 100 nodi, il tempo di calcolo cresce in maniera evidente, impendendo al solutore di trovare soluzioni ottime restando nel time-limit. Il gap ottenuto nelle istanze non risolte ottimamente è particolarmente elevato quando le istanze sono caratterizzate da un basso numero di collegamenti ferroviari.

5 Riformulazione e Risultati

Allo scopo di ridurre il GAP tra il lower-bound e la soluzione trovata nei 5 minuti a disposizione del solutore abbiamo provato due strategie:

- Provare a ridurre l'upper-bound aggiungendo valid inequalities
- Utilizzare nella formulazione originale le soluzioni di un rilassamento lagrangeano per il nostro problema e provare ad alzare il lower-bound

Riguardo la prima strategia abbiamo pensato di aggiungere al modello un vincolo che impedisca al solutore di cercare tra le soluzioni che avessero un ciclo di lunghezza 3. Più formalmente:

$$\sum_{(h=1)}^{H} y^h_{i_1 i_2} + y^h_{i_2 i_3} + y^h_{i_3 i_1} \leq 2, \;\; \forall (i_1,i_2),(i_2,i_3),(i_3,i_1) \in A, i_1 < i_2 < i_3 \quad (8)$$

Purtroppo, questo vincolo che in teoria avrebbe dovuto sveltire la risoluzione del problema, ha fatto aumentare di troppo i tempi di creazione del modello e non ci ha reso possibile vedere quanto avremmo potuto guadagnare in termini di riduzione del gap lavorando sull'upper bound.

Riguardo la seconda strategia, ossia quella tesa a trovare un lower-bound migliore (più alto), abbiamo rilassato la formulazione del problema spostando i vincoli di capacità nella funzione obiettivo. La soluzione ottima del problema rilassato è un bound duale al valore ottimo del problema originale. Il duale Lagrangeano del nostro modello è definito a partire dal rilassamento Lagrangeano dei vincoli complicanti

$$\sum_{(h=1)}^{H} y^h_{ij} \cdot d^h \leq u_{ij}, \;\; \forall (i,j) \in A \quad (9)$$

ovvero:

$$\phi(w) = min(\sum_{(i,j)\in A} \sum_{h=1}^{H} c_{ij} \cdot y^h_{ij} + \sum_{(i,j)\in A} w_{ij} \cdot (\sum_{h=1}^{H} y^h_{ij} \cdot d^h - u_{ij}) \quad (10)$$

$$(\sum_j y^h_{ji}) - (\sum_j y^h_{ij}) = b^h_i, \;\; \forall i \in N, h = 1,...H \quad (11)$$

$$\sum_{(i,j)} y^h_{ij} \leq K, \;\; h = 1,...H \quad (12)$$

$$\sum_j y^h_{ij} \leq 1, \;\; \forall i \in N, h = 1,...H \quad (13)$$

$$y^h_{ij} \in \{0,1\}, \;\; \forall (i,j) \in A, h = 1,...H \quad (14)$$

dove le variabili w_{ij} sono i moltiplicatori lagrangeani (non negativi) associati ai vincoli rilassati. Se z è il valore della funzione obiettivo del nostro modello

all'ottimo, vale la relazione:

$$\phi(w) \leq z \quad \forall w \geq 0 \tag{15}$$

Nel nostro caso non abbiamo utilizzato algoritmi di ottimizzazione per i moltiplicatori lagrangeani, ma li abbiamo scelti provandone solo alcuni valori ed in modo che fossero costanti per tutti gli archi. In particolare, i valori testati nel rilassamento sono stati: 1, 0.5, 0.4 . Questa scelta è stata guidata dal fatto che le capacità degli archi per le istanze sono tutte dello stesso ordine di grandezza e direttamente proporzionali ai costi (per archi della stessa lunghezza). Lo script *lagrange.py*, legge la soluzione del problema duale in .xml e la sostituisce nella formulazione originale del problema utilizzando il corrispondente file .cmpl.

Come è possibile vedere in table 4 risolvendo il duale lagrangeano si trovano lower-bound con valori maggiori rispetto a quelli trovati utilizzando la formulazione originale (che viene rilassata trasformando i vincoli binari in vincoli reali). Per la maggior parte delle istanze dove il gap era abbastanza alto (>3%) lo stesso è stato ridotto notevolmente mentre per le istanze con gap già abbastanza piccolo il guadagno non è stato molto importante. In particolare il gap era già abbastanza piccolo nella versione originale in tutte le istanze con una percentuale di attivazione delle ferrovie elevata. Riguardo il parametro K abbiamo notato che nelle istanze dove la percentuale di attivazione delle ferrovie era piccola, usare un K troppo grande ha effetti negativi sulla risoluzione del problema in quanto lo spazio di ricerca viene inutilmente ingrandito. Questo fatto è confermato anche dalle soluzioni trovate nel problema duale che oltre un certo K smettono di scendere nei valori. Lo script *eval_ activation.py* prende in input una serie di soluzioni in formato .xml e ne estrae il numero di ferrovie e strade attivate così come il numero di ordini da soddisfare. Nelle istanze dove la riduzione del gap è stata importante (prime quattro istanze in Table 4)

6 Tabelle e Immagini

Nodes	PercFerr	K	NOrder	Solver	Solution	LowerBound	Time	Memory	Gap
10	0.1	1	5	glpk	1407	1407	0s	0,8Mb	0%
10	0.1	1	5	cbc	1407	1407	0,08s	-	0%
10	0.1	2	5	glpk	1033	1033	0s	0,8Mb	0%
10	0.1	2	5	cbc	1033	1033	0,1s	-	0%
10	0.1	3	5	glpk	1014	1014	0s	0,8Mb	0%
10	0.1	3	5	cbc	1014	1014	1,73s	-	0%
50	0.1	2	25	glpk	17459	17459	17,2s	93,6Mb	0%
50	0.1	2	25	cbc	17459	17459	35s	-	0%
50	0.1	4	25	glpk	14784	14784	6,9s	87,3Mb	0%
50	0.1	4	25	cbc	14784	14784	41s	-	0%
50	0.1	6	25	glpk	14734	14734	6,5s	89,9Mb	0%
50	0.1	6	25	cbc	14734	14734	29s	-	0%
50	0.1	13	25	glpk	14734	14734	6,1s	89,9Mb	0%
50	0.1	13	25	cbc	14734	14734	29s	-	0%
100	0.1	2	50	glpk	60239	59167	/	1042,5Mb	1.8%
100	0.1	2	50	cbc	NoSol	NoSol			-
100	0.1	7	50	glpk	52563	50858	/	725,7Mb	3.10%
100	0.1	7	50	cbc	NoSol	NoSol			-
100	0.1	12	50	glpk	53101	50867	/	684,6Mb	4.10%
100	0.1	12	50	cbc	NoSol	NoSol			-
100	0.1	25	50	glpk	53283	50867	/	684,6Mb	4.50%
100	0.1	25	50	cbc	NoSol	NoSol			-

Table 1: Risultati delle istanze con percentuale di attivazione ferrovie = 0.1

Nodes	PercFerr	K	NOrder	Solver	Solution	LowerBound	Time	Memory	Gap
10	0.5	1	5	glpk	874	874	0s	0,8Mb	0%
10	0.5	1	5	cbc	874	874	0,12s	-	0%
10	0.5	2	5	glpk	740	740	0s	0,9Mb	0%
10	0.5	2	5	cbc	740	740	1s	-	0%
10	0.5	3	5	glpk	740	740	0s	0,9Mb	0%
10	0.5	3	5	cbc	740	740	0,13s	-	0%
50	0.5	2	25	glpk	15140	15140	9s	88,6Mb	0%
50	0.5	2	25	cbc	15140	15140	35s	-	0%
50	0.5	4	25	glpk	13620	13620	77s	120,6Mb	0%
50	0.5	4	25	cbc	13620	13620	40s	-	0%
50	0.5	6	25	glpk	13571	13571	39s	105,6Mb	0%
50	0.5	6	25	cbc	13571	13571	37s	-	0%
50	0.5	13	25	glpk	13571	13571	34s	102,6Mb	0%
50	0.5	13	25	cbc	13571	13571	34s	-	0%
100	0.5	2	50	glpk	56257	56257	198s	722,5Mb	0%
100	0.5	2	50	cbc	NoSol	NoSol		-	-
100	0.5	7	50	glpk	52700	52229	/	704,3Mb	0.9%
100	0.5	7	50	cbc	NoSol	NoSol			-
100	0.5	12	50	glpk	52925	52233	/	704,3 Mb	1.30%
100	0.5	12	50	cbc	NoSol	NoSol			-
100	0.5	25	50	glpk	52925	52233	/	704.3Mb	1.30%
100	0.5	25	50	cbc	NoSol	NoSol			-

Table 2: Risultati delle istanze con percentuale di attivazione ferrovie = 0.5

Nodes	PercFerr	K	NOrder	Solver	Solution	LowerBound	Time	Memory	Gap
10	0.9	1	5	glpk	1303	1303	0s	0,9Mb	0%
10	0.9	1	5	cbc	1303	1303	0,11s	-	0%
10	0.9	2	5	glpk	852	852	0s	0,9Mb	0%
10	0.9	2	5	cbc	852	852	0,04s	-	0%
10	0.9	3	5	glpk	852	852	0s	0,9Mb	0%
10	0.9	3	5	cbc	852	852	0,04s	-	0%
50	0.9	2	25	glpk	15439	15439	6s	90,8Mb	0%
50	0.9	2	25	cbc	15439	15439	24,12s	-	0%
50	0.9	4	25	glpk	14535	14535	9s	94,3Mb	0%
50	0.9	4	25	cbc	14535	14535	28,64s	-	0%
50	0.9	6	25	glpk	14515	14515	8s	94,1Mb	0%
50	0.9	6	25	cbc	14515	14515	28,97s	-	0%
50	0.9	13	25	glpk	14515	14515	8s	94,3Mb	0%
50	0.9	13	25	cbc	14515	14515	31s	-	0%
100	0.9	2	50	glpk	50915	46290	/	883,9Mb	4.1%
100	0.9	2	50	cbc	NoSol	NoSol			-
100	0.9	7	50	glpk	45635	45329	/	795,4Mb	0.50%
100	0.9	7	50	cbc	NoSol	NoSol			-
100	0.9	12	50	glpk	45635	45329	/	822,2Mb	0.50%
100	0.9	12	50	cbc	NoSol	NoSol			-
100	0.9	25	50	glpk	45635	45329	/	859.4 Mb	0.50%
100	0.9	25	50	cbc	NoSol	NoSol			-

Table 3: Risultati delle istanze con percentuale di attivazione ferrovie = 0.9

Nodes	PercFerr	K	NOrder	Solver	Solution	LowerBound	SolutionRelax	NewLower	Gap	NewGap
100	0.1	2	50	glpk	60239	59167	66078	59891	1.8%	0.6%
100	0.1	7	50	glpk	52563	50858	60090	51563	3.10%	1.9%
100	0.1	12	50	glpk	53101	50867	60090	51563	4.10%	2.8%
100	0.1	25	50	glpk	53283	50867	60090	51563	4.50%	3.2%
100	0.5	7	50	glpk	52700	52229	56962	52306	0.9%	0.7%
100	0.5	12	50	glpk	52925	52233	56962	52306	1.30%	1.10%
100	0.5	25	50	glpk	52925	52233	56962	52306	1.30%	1.10%
100	0.9	2	50	glpk	50915	46290	55225	49081	4.1%	3.6%
100	0.9	7	50	glpk	45635	45329	49092	45482	0.50%	0.3%
100	0.9	12	50	glpk	45635	45329	49092	45482	0.50%	0.3%
100	0.9	25	50	glpk	45635	45329	49092	45482	0.50%	0.3%

Table 4: In giallo soluzione trovata dal solutore nel limite dei 300 secondi. In rosso il nuovo gap calcolato utilizzando come nuovo lowerbound la soluzione del problema lagrangeano all'interno della funzione obiettivo del problema originale