UNIVERSITÀ DEGLI STUDI DI TORINO

Corso di Laurea Magistrale in Informatica

Tesina di Ottimizzazione Combinatoria

Multi Commodity Min Cost Flow

Alberto Campini
885128

Alessandro Cavaglià
882567

Anno Accademico 2022/2023

Abstract

Il seguente lavoro è sviluppato come tesina compilativa relativa al corso Ottimizzazione Combinatoria e verte nell'analisi e approfondimento del problema di flusso su rete conosciuto come Multi Commodity Min Cost flow. Dopo aver introdotto dal punto di vista teorico il problema appoggiandosi alla sua versione generalizzata si entrerà nel dettaglio cercando di individuare gli elementi complicanti, i possibili rilassamenti, approcci risolutivi e implementazione tecnica di alcune di esse. Il lavoro si conclude con una comparazione con istanze note del problema degli algoritmi implementati

Indice

1 Introduzione

Il problema del Multi commodity Min cost flow, che da ora in poi per semplicità identificheremo con la sigla MC-MCF nasce come generalizzazione del più comune problema Min cost flow (MCF). Questo è un problema di flusso su reti che consiste nell'instradamento di flusso su una rete cercando di minimizzare il costo complessivo degli archi utilizzati. E' un problema noto in letteratura ed è molto utile per svolgere diversi task come ad esempio:

- Instradamento di merci su reti stradali
- Gestione di reti di comunicazioni
- Gestione del traffico stradale

Affronteremo prima la definizione del MCF per capirne la struttura e il modello di programmazione lineare, successivamente analizzeremo come e in cosa cambia la sua variante Multi commodity.

1.1 MCF: Min Cost Flow

Dato un Grafo orientato $G = (N, A)$ dove ad ogni *nodo* $\in$N è associato un bilanciamento b_i che rappresenta la differenza tra il flusso entrante e il flusso uscente dal nodo. Ad ogni arco (i, j) $\in$A sono associati il costo unitario c_{ij} che rappresenta il costo per ogni unità di flusso che passerà per l'arco e la capacità u_{ij} che rappresenta il quantitativo massimo di flusso che può passare per quell'arco.
All'interno del grafo sono caratterizzati nodi di diverso tipo, abbiamo:

- Nodi origini o sorgenti identificati generalmente con la t e caratterizzati da un bilancio associato $b_i < 0$, questa tipologia di nodi immette flusso nel sistema.
- Nodi destinazione o pozzo identificati generalmente con la s e caratterizzati da un bilancio associato $b_i > 0$, questi nodi ricevono flusso dal sistema
- Nodi di transito e sono caratterizzati da un bilancio associato $b_i = 0$, cioè tutto il flusso che entra deve anche uscire, appunto fanno scorrere il flusso.

I bilanci associati ai nodi dunque governano il flusso sulla rete imponendo regole precise, mentre gli archi definiscono i limiti di flusso che può circolare e il suo costo.

Il problema del Min Cost Flow consiste nel determinare il flusso massimo sugli archi della rete in modo che dalle sorgenti parta tutto il flusso a disposizione, e

alle origini arrivi tutto il flusso richiesto, senza che le capacità u_{ij} degli archi e i bilanciamenti b_i dei nodi vengano violati cercando di minimizzare il costo complessivo dell'instradamento del flusso sulla rete.

1.1.1 Problema PL

Data questa descrizione possiamo formalizzare una definizione del problema del Min Cost Flow come un problema di programmazione lineare, utilizzando le variabili di flusso x_{ij} sugli archi a rappresentare il quantitativo di flusso che percorrerà l'arco. Possiamo formulare il modello di programmazione lineare che governa come segue

$$min \sum_{(i,j)\in A} c_{ij}x_{ij} \tag{1}$$

subject to

$$-\sum_{(i,j)\in FS} x_{ij} + \sum_{(j,i)\in BS} x_{ji} = b_i \quad \forall i \in N \tag{2}$$

$$0 \leq x_{ij} \leq u_{ij} \quad \forall (i,j) \in A \tag{3}$$

La funzione obiettivo (1) punta a minimizzare la somma del costo su tutti gli archi percorsi da flusso.

La funzione obiettivo è vincolata dai vincoli (2) e (3), il primo definisce il vincolo sui bilanci dei nodi appartenenti ad N andando a modellare il flusso sulla rete, infatti esso non può mai violare il bilancio relativo al nodo che sta attraversando.

Il secondo vincolo invece gestisce la capacità degli archi appartenenti ad A, su ogni arco non può mai passare più flusso della sua capacità massima.

1.1.2 Condizioni di ottimalità

Dato un problema di MCF partiamo dal calcolare una soluzione di flusso ammissibile per il problema x'.

Costruiamo poi il grafo residuale della soluzione inserendo però per ogni arco un valore g_{ij} definito come segue:

$$\begin{array}{c} c_{ij} se (i,j) \in A^+(X) (\text{Ovvero se l'arco è concorde}) \\ -c_{ij} se (i,j) \in A^-(X) (\text{Ovvero se l'arco è discorde}) \end{array} \tag{4}$$

Data questa formulazione, l'unico modo per modificare la soluzione senza violare i vincoli di conservazione del flusso consiste nel variare il flusso su un ciclo di una stessa quantità θ.

Variando il flusso di un valore θ otterremo un nuovo flusso sulle variabili x''_{ij}.

La funzione obiettivo dopo il cambiamento del flusso subirà una variazione pari a:

$$\sum_{(i,j)\in A} c_{ij}x''_{ij} - \sum_{(i,j)\in A} c_{ij}x'_{ij} = \sum_{(i,j)\in A} g_{ij}\theta \tag{5}$$

Quindi vi è un miglioramento della funzione obiettivo solo se il ciclo ha somma dei costi residuali negativi. Il flusso ammissibile x' e' quindi ottimo se e solo se il grafo residuale $G^R(x')$ non contiene cicli la cui somma dei costi residuali è negativa.

1.2 MC-MCF: Multi commodity Min Cost Flow

La precedenza formulazione del MCF ha una assunzione sottintesa, cioè il fatto che il flusso circolante nella rete è omogeneo. Questo però in molti casi reali non è vero e dunque non possono essere modellati con questa ipotesi restrittiva. E' necessario quindi un modello in grado di considerare flussi distinti che concorrono all'utilizzo delle risorse comuni. Da qui nasce il nostro problema, il Multi Commodity Min Cost Flow.

Dato un Grafo orientato $G = (N, A)$ e K commodity, ad ogni nodo è associato un bilanciamento b_i^k per ogni flusso $k \in K$ e ad ogni arco (i,j) sono associati il costo unitario c_{ij} e la capacità u_{ij}.

Notiamo che con l'introduzione del concetto di commodity, la definizione per gli archi del grafo non cambia, infatti gli archi non sono influenzati da quale commodity li attraversa, i nodi al contrario hanno un bilancio specifico per ogni comodity $k \in K$. Quindi lo stesso nodo può essere contemporaneamente sia un nodo sorgente, un nodo destinazione oppure di passaggio in base alla specifica commodity k che viene valutata. L'obiettivo del problema resta equivalente, ovvero di minimizzare il costo del flusso che percorre la rete, però in questa versione avremo K comodities che concorreranno per ottenere le risorse a messe a disposizione della rete.

1.2.1 Problema PL

Data questa descrizione possiamo formalizzare il MCMCF come un problema di programmazione lineare tramite variabili di flusso x_{ij}^k sugli archi, che denotano il quantitativo del flusso k che circola sull'arco (i,j).
Possiamo formulare il modello di programmazione lineare come segue.

$$min \sum_{(i,j)\in A} \sum_{k\in K} c_{ij}x_{ij}^k \tag{6}$$

subject to

$$-\sum_{(i,j)\in FS} x_{ij}^k + \sum_{(j,i)\in BS} x_{ji}^k = b_i^k \quad \forall i \in N \quad \forall k \in K \tag{7}$$

$$0 \leq \sum_{k\in K} x_{ij}^k \leq u_{ij} \quad \forall (i,j) \in A \tag{8}$$

La funzione obiettivo (6) dunque punta a minimizzare la somma del costo su tutti gli archi percorsi dal flusso di tutte le k comodity presenti nella rete.

La funzione obiettivo è vincolata dai vincoli (7) e (8).

Il primo vincolo (7) definisce il controllo sui bilanci dei nodi appartenenti ad N andando a modellare il flusso sulla rete.

Rispetto a un MCF, nel problema di flusso multi-prodotto si replicano i vincoli di conservazione del flusso per ogni prodotto k impedendogli di violare il bilancio b_i^k cioè il bilancio del nodo i assocaito alla commodity k.

Il vincolo (8) e' chiamato anche *Boundle contraint* e gestisce la capacità degli archi appratenti ad A, su ogni arco non può mai passare più flusso della sua capacità massima, anche in questo caso viene calcolata la somma per ogni prodotto k. Per le successive elaborazioni che faremo di questo modello, può essere utile andare ad effettuare una standardizzazione del vincolo (8) sfruttando una variabile di slack s_{ij} che permette di trasformare il vincolo come segue:

$$\sum_{1\leq k\leq K} x_{ij}^k + s_{ij} = u_{ij} \quad \forall (i,j) \in A \tag{9}$$

La struttura del problema così formalizzato impone quindi di trovare contemporaneamente i flussi per tutti i prodotti.

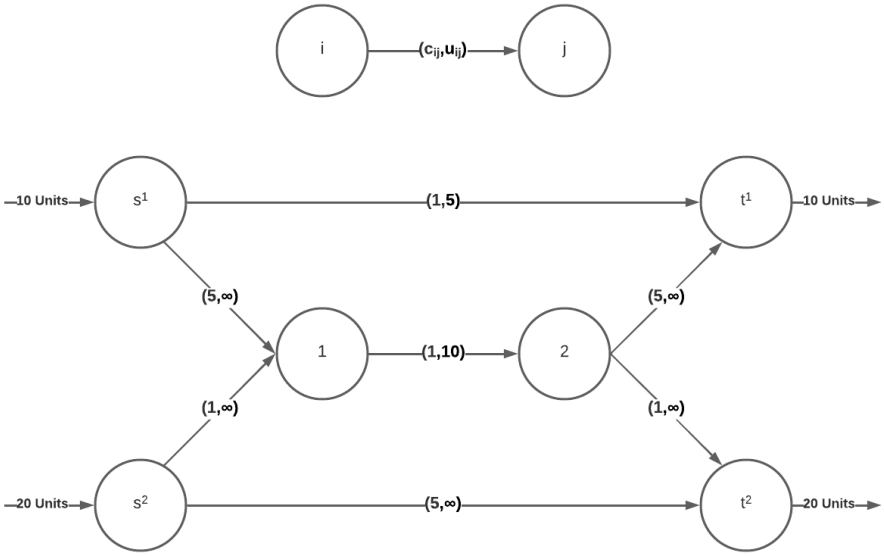

Figure 1: Esempio della struttura del problema di Multi commodity min cost flow

1.2.2 Assunzioni sulla formulazione

Prima di affrontare i possibili approcci risolutivi al problema è necessario discutere di una serie di assunzioni indotte dal modello di programmazione lineare sopra proposto.

Homogeneus goods assumption. Assumiamo che ogni unità di flusso per ogni commodity k utilizzi una unità della capacità di ogni arco su cui scorre. Una formulazione più generale del problema potrebbe permettere che in base alla specifica commodity k il suo flusso utilizzi unità differenti di capacità dell'arco. Questo è ottenibile generalizzando con una costante p_{ij}^k, ovvero un moltiplicatore associato all'arco (i,j) per la commodity k. Questa variazione porterebbe un cambiamento del vincolo *Boundle contraint* 8.

No Congestion assumption. Assumiamo che abbiamo una capacità fissata e restrittiva per ogni singolo arco (i,j) del nostro grafo, inoltre il costo del flusso è lineare per l'intera rete. Questa assunzione serve perché in alcune situazioni reali le singole commodities interagiscono con la rete in modo non lineare andando ad aumentare o diminuire il costo del flusso durante un percorso. Questo rende di fatto obbligatorio gestire la funzione obiettivo come la minimizzazione della congestione della rete per evitare traffico.

Indivisible goods assumption. Il modello proposto assume che la variabile di flusso possa essere continua e non per forza intera. In alcune applicazioni questo non è ammissibile, tutta via il fatto che sia un numero frazionario rappresenta sicuramente un upper bound ad una eventuale soluzione intera dello stesso problema. Tramite tecniche note come il branch and bound è possibile, partendo da una soluzione ottima non intera , ottenere l'ottimo intero del problema.

1.2.3 Condizioni di ottimalità

Andiamo ora ad analizzare le condizioni che definiscono l'ottimalità di un problema di MC-MCF che data una soluzione ammissibile del problema, ci permette di identificare se è anche una soluzione ottima. Oltre a questo le condizioni di ottimalità sono fondamentali per trattare i possibili apporci risolutivi dato che ognuno di essi cerca di risolvere codeste condizioni.

Per discutere delle condizioni di ottimalità di questo problema assumiamo che le variabili di flusso x_{ij}^k non abbiano un limite individuale di flusso, sia dunque $u_{ij}^k = \infty$ partendo dalla formulazione del programma lineare in sezione 1.2.1.

Poiché il problema del flusso multi commodity è un programma lineare, possiamo sfruttare le condizioni di ottimalità della programmazione lineare per caratterizzare le soluzioni ottimali di questo specifico problema.

Data la formulazione del programma lineare del problema (sezione 1.2.1), si nota come abbia un vincolo di bundle per ogni arco (i,j) presente nella rete e un vincolo di bilanciamento per ogni combinazione di nodo e commodity k. Il programma lineare duale invece ha due tipi di variabili duali:

- un prezzo w_{ij} su ogni arco (i, j)
- un potenziale $\pi^k(i)$ per ogni combinazione di commodity k e nodo i.

Utilizzando queste variabili duali, definiamo il costo ridotto $c_{ij}^{\pi,k}$ dell'arco (i, j) rispetto alla commodity k nel seguente modo:

$$c_{ij}^{\pi,k} = c_{ij}^k + w_{ij} - \pi^k(i) + \pi^k(j) \tag{10}$$

Notiamo che fissando una specifica commodity k, il costo ridotto $c_{ij}^{\pi,k}$ è simile al costo ridotto che abbiamo utilizzato nella trattazione delle condizioni di ottimalità del problema MCF (sezione 1.1.2); la differenza è che in questo caso andiamo ad aggiungere il prezzo dell'arco w_{ij} al costo dell'arco c_{ij}^k.

Notiamo che proprio come il vincolo di bundle forniva un collegamento tra le variabili di flusso x_{ij}^k, altrimenti indipendenti tra loro, anche i prezzi degli archi w_{ij} forniscono un collegamento tra i costi ridotti di commodity altrimenti indipendenti.

Sfruttiamo ora la teoria della dualità della programmazione lineare per caratterizzare le soluzioni ottimali del problema del flusso multicommodity, come prima cosa scriviamo il duale del problema del flusso multicommodity definito in sezione 1.2.1:

$$max - \sum_{(i,j)\in A} u_{ij} w_{ij} \sum_{k\in 1}^{K} b^k \pi^k \tag{11}$$

subject to

$$c_{ij}^{\pi,k} = c_{ij}^k + w_{ij} - \pi^k(i) + \pi^k(j) \geq 0 \quad \forall i \in N \quad \forall k \in K \tag{12}$$

$$w_{i,j} \geq 0 \quad \forall (i,j) \in A \tag{13}$$

Le condizioni di ottimalità per un programma lineare, chiamate condizioni di complementarietà delle slacks (complementary slackness conditions), affermano che una soluzione primale ammissibile x e una soluzione duale ammissibile (w, π^k) sono ottimali per i rispettivi problemi se e solo se il prodotto di ogni variabile primale (duale) e la slack nel vincolo duale (primale) corrispondente è zero.

Le condizioni di complementarità delle slacks per la coppia primale-duale del problema del flusso multicommodity assumono la seguente forma speciale. (Utilizzeremo y_{ij}^k per indicare un valore specifico della variabile di flusso x_{ij}^k.)

Multicommodity flow complementary slackness conditions. I flussi delle commodity y_{ij}^k sono ottimali per il problema del flusso multicommodity (1.2.1) con ogni $u_{ij}^k = \infty$, se e solo se sono ammissibili e per una qualche scelta di prezzi (non negativi) degli archi w_{ij} e potenziali dei nodi $\pi^k(i)$ (non limitati nel segno). A questo punto i costi ridotti e i flussi degli archi soddisfano le seguenti condizioni di complementarità delle slacks:

1.
$$w_{ij}(\sum_{1\leq k\leq K} y_{ij}^k - u_{ij}) = 0 \quad \forall (i,j) \in A \tag{14}$$

2.
$$c_{ij}^{\pi,k} \geq 0 \quad \forall i \in N \quad \forall k \in K \tag{15}$$

3.
$$c_{ij}^{\pi,k} y_{ij}^k = 0 \quad \forall i \in N \quad \forall k \in K \tag{16}$$

Ci riferiamo a qualsiasi insieme di prezzi degli archi e potenziali dei nodi che soddisfano le condizioni di complementarità delle slacks come prezzi ottimali degli archi e potenziali ottimali dei nodi. Il seguente teorema mostra la connessione tra il problema del flusso multicommodity e il problema del flusso a singola commodity.

Teorema della Partial Dualization. Sia y_{ij}^k il flusso ottimo e w_{ij} i prezzi ottimali degli archi per il problema del flusso MC MCF (1.2.1). Allora per ogni commodity k, la variabile di flusso y_{ij}^k per $(i,j) \in A$ risolve il seguente minimum cost flow problem:

$$min\{ \sum_{(i,j)\in A} (c_{ij}^k + w_{ij}) x_{ij}^k : \mathscr{N} x^k = b, x_{ij}^k \geq 0 \quad \forall (i,j) \in A\} \tag{17}$$

Questa proprietà mostra come possiamo utilizzare un approccio sequenziale per ottenere i prezzi ottimali degli archi e i potenziali ottimali dei nodi: prima troviamo i prezzi ottimali degli archi e successivamente cerchiamo di trovare i potenziali ottimali dei nodi e i flussi risolvendo i problemi del flusso a costo minimo per singola commodity.

2 Apporci risolutivi

Avendo definito formalmente i due problemi affrontiamo in questa sezione gli apporci risolutivi proposti che saranno alla base della trattazione degli algoritmi per risolvere il Multi commodity min cost flow. Essendo questo un problema generalizzato di MCF è bene studiare brevemente gli apporci risolutivi per MCF dato che diverse intuizioni verranno utilizzate per decomporre e di conseguenza semplificare MC-MCF.

2.1 Soluzioni per MCF

In questa sezione tratteremo brevemente le principali tecniche risolutive per problemi di Min Cost Flow, questa sezione sarà utile per sfruttare parte della trattazione nella definizione delle soluzioni per il problema generalizzato Multi Commodity. Affronteremo tre tipologie differenti di approcci risolutivi che si basano su algoritmi di diverso tipo via via migliori:

- Basic algorithms approach
- Polynominal algorithms approach
- Network simplex algorithms

Andremo a ora a dare cenni delle intuizioni dietro ad ogni famiglia di algortmi sopra elencata.

2.1.1 Basic algorithms approach

Gli approci più diretti per risolvere problemi MCF si basano fortemente su strumenti e metodi di analisi attingendo a tecniche e strategie risolutive applicabili ad altri problemi noti come il Max Flow e lo Shortest Path. Ad esempio, al fine di fornirci una base solida per lo sviluppo di algoritmi per risolvere problemi di flusso a costo minimo, nella Sezione 1.1.2 abbiamo stabilito le condizioni di ottimalità per i problemi di flusso a costo minimo, basate sulla nozione di potenziali associati ad ogni nodo associati ai nodo del grafo. Questi potenziali dei nodi sono generalizzazioni del concetto di etichette di distanza che vengono utilizzate nello studio dei problemi dello shortest path.

L'ottimalità di uno shortest path è caratterizzata dalle etichette di distanza; possiamo sfruttare le condizioni di ottimalità delle etichette di distanza come punto di partenza per lo sviluppo degli algoritmi di *iterative label-setting* e *label-correcting* alla base della risoluzione dei problemi del shortest path.

Utilizziamo il potenziale dei nodi in modo simile per i problemi MCF. Tuttavia anche in questo caso, la connessione con i problemi di shortest path è molto più profonda di questa semplice analogia tra i potenziali dei nodi e le etichette di distanza. Ad esempio, è noto come interpretare e trovare i potenziali dei nodi ottimali per un problema di flusso a costo minimo sia possibile risolvendo un appropriato problema shortest path. Vale la seguente proprietà: i potenziali dei nodi ottimali per un MCF sono uguali al negativo delle etichette di distanza ottimali del relativo problema shortest path.

Come accennato all'inizio, non viene sfruttato solo le intuizioni e tecniche dietro al shortest path, ma molti approcci basici sfruttano anche le tecniche del flusso massimo. Infatti un gran numero di questi algoritmi risolvono una sequenza di problemi shortest path rispetto a grafi residuali simili a flussi massimi e cammini aumentanti. (In realtà, per definire il grafo residuale in questo contesto, consideriamo sia considerazioni di costo che di capacità.)

Seguono ora quattro algoritmi che sfruttano queste intuizioni per risolvere gli MCF:

- *cycle-canceling algorithm* utilizza i calcoli dei vari shortest path per trovare cicli aumentati con costi di flusso negativi; quindi aumenta i flussi lungo questi cicli e ripete iterativamente questi calcoli per individuare via via cicli di costo negativo e aumentare i flussi nella rete.

- *successive shortest path algorithm* carica incrementalmente il flusso sulla rete da un nodo sorgente a un nodo pozzo, selezionando ogni volta un percorso più breve opportunamente definito.

- *Primal-dual algorithm* e *out-ofkilter algorithm* utilizzano una strategia algoritmica simile: ad ogni iterazione, risolvono un problema shortest path e aumentano il flusso lungo i percorsi più brevi (che possono essere uno o più). Tuttavia, differiscono nelle loro strategie. L'algoritmo primale-duale utilizza un calcolo del flusso massimo per aumentare contemporaneamente il flusso lungo diversi shortest path.

- *Out-ofkilter algorithm* diversamente dagli altri consente ai flussi degli archi di violare i loro limiti di flusso. Utilizza calcoli dei shortest path per trovare flussi che soddisfano sia i limiti di flusso che le condizioni di ottimalità basate su costo e capacità.

Il fatto che possiamo implementare algoritmi iterativi del shortest path in così tanti modi dimostra la versatilità disponibile nella risoluzione dei problemi di flusso a costo minimo.

Poiché i problemi di flusso a costo minimo sono programmi lineari, di cui abbiamo dato la definizione in sezione 1.1.1 non sorprende scoprire che possiamo anche utilizzare metodologie di programmazione lineare per risolvere i problemi di flusso a costo minimo. Notiamo infatti che molte delle varie condizioni di ottimalità sono casi particolari delle condizioni di ottimalità più generali della programmazione lineare. Inoltre, possiamo interpretare molti di questi risultati nel contesto di una teoria generale di dualità per i programmi lineari.

2.1.2 Polynominal algorithms approach

Gli apporci che abbiamo appena citato garantiscono una convergenza finita solo quando i dati del problema sono integrali, ma queste soluzioni non sono necessariamente polinomiali. Questi approcci, dunque, non ci forniscono alcuna assicurazione teorica che gli algoritmi si comportino bene su tutte le istanze di problemi che potremmo incontrare.

Vorremmo capire quali sono gli elementi che ci servono per sviluppare algoritmi polinomiali per risolvere un MCF. La strategia che viene adottata più comunemente è la scalatura dei dati di capacità e/o dei dati di costo della rete.

La scalatura è un'idea potente che ha portato a miglioramenti algoritmici per molti problemi di ottimizzazione combinatoria. Ad alto livello possiamo considerare gli algoritmi di scalatura nel seguente modo: partiamo dalle condizioni di ottimalità per il problema di flusso di rete che stiamo esaminando, in questo caso il nostro MCF, ma anziché imporre queste condizioni esattamente, generiamo una soluzione "approssimata" (o anche detta rilassata) che può violare una (o più) delle condizioni di un'entità Δ.

Inizialmente, scegliendo Δ abbastanza grande, ad esempio come C oppure U, saremo facilmente in grado di trovare una soluzione di partenza che soddisfa le condizioni di ottimalità rilassate. Quindi reimpostiamo il parametro Δ a $\frac{\Delta}{2}$ e ottimizziamo nuovamente in modo che la soluzione approssimata violi le condizioni di ottimalità per al massimo $\frac{\Delta}{2}$. Ripetiamo quindi la procedura, ottimizzando nuovamente fino a quando la soluzione approssimata viola le condizioni per al massimo $\frac{\Delta}{4}$, e così via. Questa strategia di soluzione è piuttosto flessibile e porta a diversi algoritmi a seconda di quali condizioni di ottimalità rilassiamo e di come eseguiamo le nuove ottimizzazioni.

La strategia dunque è quella di rilassare il problema e ad ogni "iterazione" far calare il nostro upper bound ammissibile modulato dal parametro Δ fino a raggiungere il "limite" per il quale il problema rimane valido.

2.1.3 Network simplex algorithms approach

Poiché i problemi di flusso a costo minimo definiscono una classe speciale di programmi lineari, ci potremmo aspettare che il metodo del simplesso sia la procedura risolutiva migliore.

Ragionando notiamo però che i problemi di flusso di rete hanno una struttura ben specifica e dunque potremmo chiederci se il metodo del simplesso possa competere con altri metodi "combinatoriali", come le molte varianti dei "successive shortest path algorithm", che sfruttano la struttura di rete sottostante al problema. Il metodo del simplesso generale, quando implementato in modo classico e che quindi non sfrutta la struttura di rete sottostante al problema, non è per definizione una buona procedura risolutiva ed è dunque poco competitiva per risolvere problemi MCF.

Tuttavia, se interpretiamo correttamente i concetti fondamentali alla base del metodo del simplesso come operazioni di rete, possiamo adattarlo e semplificarlo per sfruttare al meglio la struttura di rete sottostante al MCF, ottenendo un algoritmo molto più efficiente.

L'algoritmo del simplesso su rete si basa sul concetto centrale delle soluzioni di tipo spanning tree. Un spanning tree è un sottografo connesso che contiene tutti i nodi di una rete senza cicli. Nel contesto dell'algoritmo del simplesso su rete, le soluzioni di tipo spanning tree sono ottenute fissando il flusso su ogni arco che non fa parte di un spanning tree, sia a valore zero che alla capacità massima dell'arco stesso.

L'obiettivo è risolvere in modo univoco il flusso su tutti gli archi dello spanning tree. Inoltre, viene dimostrato che il problema del flusso a costo minimo ha sempre almeno una soluzione ottimale rappresentata da uno spanning tree. È possibile trovare tale soluzione "spostandoci" da un'altra soluzione ottimale dello spanning tree, introducendo ad ogni passo un nuovo arco che non appartiene ancora allo spanning tree.

2.2 Soluzioni per MC MCF

La ricerca su questo problema è molto ricca e sono stati proposti diversi approcci per risolvere il Multi Commodity Min Cost Flow Problem, le principali proposte ricadono in tre grandi famiglie:

- Price-directive decomposition
- Resource-directive decomposition

- Partitioning methods

Andremo ora a dettagliare i singoli approcci che verranno poi sfruttati nei capitoli successivi per costruire algoritmi risolutivi del problema.

2.2.1 Price-directive decomposition

I metodi di Price-directive decomposition pongono moltiplicatori di Lagrange detti anche prezzi sui vincoli di bundle e li introducono nella funzione obiettivo in modo che il problema risultante si decomponga in un problema separato di flusso a costo minimo per ogni commodity k. In altre parole, questi metodi rimuovono i vincoli di capacità e invece "addebitano" a ciascuna commodity l'uso della capacità di ciascun arco. Questi metodi cercano di trovare prezzi appropriati in modo che una soluzione ottimale del "problema dei prezzi" risultante o del sotto-problema lagrangiano risolva anche il problema complessivo del flusso multicommodity.

Nella sezione 3.1 descriviamo l'applicazione del rilassamento lagrangiano per trovare i prezzi corretti.

La Dantzig-Wolfe decomposition è un altro approccio per trovare i prezzi corretti, questo metodo è un approccio più generale per decomporre problemi che presentano un insieme di vincoli "facili" e un insieme di vincoli "difficili" detti anche vincoli complicanti del problema.

Per i problemi di flusso multicommodity, i vincoli di flusso di rete sono i vincoli facili e i vincoli di bundle sono i vincoli difficili. L'approccio inizia, come il rilassamento lagrangiano, ignorando o imponendo prezzi sui vincoli di bundle e risolvendo sotto-problemi lagrangiani con solo i vincoli di flusso di rete per singola commodity.

Le soluzioni risultanti potrebbero non soddisfare i vincoli di bundle, questo metodo sfrutta la programmazione lineare per aggiornare i prezzi in modo che le soluzioni generate dai sotto-problemi soddisfino i vincoli di bundle del problema principale.

Il metodo risolve iterativamente due problemi diversi: un sotto-problema lagrangiano e un programma lineare per impostare i prezzi. Questo metodo ha giocato un ruolo importante nel campo dell'ottimizzazione sia perché l'algoritmo stesso si è dimostrato molto utile, sia perché ha stimolato molti altri approcci alla decomposizione dei problemi.

2.2.2 Resource-directive decomposition

Possiamo considerare il problema del flusso multicommodity come un problema di allocazione di capacità. Tutte le commodity competono per la capacità fissa (la capacità dell'arco) u_{ij} di ogni arco (i, j) della rete.

Qualsiasi soluzione ottimale al problema del flusso multicommodity prescriverà un flusso specifico su ogni arco (i, j) per ciascuna commodity k, che rappresenta la capacità appropriata da allocare a quella commodity. Se iniziamo allocando queste capacità alle commodity e poi risolviamo i risultanti problemi di flusso a singola commodity (indipendenti tra loro), saremo in grado di risolvere il problema abbastanza facilmente come un insieme di problemi di flusso a singola commodity indipendenti.

I metodi Resource-directive decomposition forniscono un approccio generale per implementare questa idea. L'idea alla base è di iniziare allocando le capacità alle commodity e poi utilizzano le informazioni ricavate dalla soluzione ai problemi di flusso a singola commodity risultanti per riallocare le capacità in modo tale da migliorare il costo complessivo del sistema. Nella Sezione 3.4 mostriamo come risolvere il problema del flusso multicommodity utilizzando questo particolare approccio.

2.2.3 Partitioning methods

I Partitioning methods sfruttano il fatto che il problema del flusso multicommodity è un programma lineare strutturato con una serie di problemi di flusso di rete incorporarti al suo interno. Come abbiamo visto nel nella sezione precedente (Sol MCF), per risolvere qualsiasi problema di flusso a singola commodity, possiamo utilizzare il metodo del simplesso di rete, che funziona generando una sequenza di soluzioni di alberi di spanning che migliorano via via.

Nella Sezione 2.1.3 abbiamo accennato alla approccio di risoluzione del MCF che si basa sul simplesso su reti, esprimendo le soluzioni di base ammissibili del problema del MCF come soluzioni di spanning tree. Questa osservazione solleva le seguenti domande:

1. Possiamo adottare un approccio simile per risolvere il problema del flusso multicommodity?

2. Possiamo in qualche modo utilizzare le soluzioni di alberi di spanning per i vincoli di flusso di rete $\mathscr{N}x^k = b^k$?

Il metodo di partizionamento è un approccio di programmazione lineare che ci permette di rispondere in modo affermativo a entrambe queste domande. Mantiene una base di programmazione lineare composta da basi (spanning tree) dei singoli problemi di flusso a singola commodity, oltre ad archi aggiuntivi che sono necessari per "legare" queste soluzioni insieme per soddisfare i vincoli di bundle.

3 Tecniche Risolutive

In questa sezione intendiamo dettagliare le tecniche risolutive per risolvere in modo efficiente il multi commodity min cost flow problem. Ognuna delle tecniche che dettaglieremo si baserà fortemente su una o più idee introdotte nella sezione precedente appoggiandosi in alcuni casi anche a strategie e tecniche risolutive del MCF.

Le tecniche introdotte approcciano il problema in modo differente proponendo delle riformulazioni che ci permettono di risolverlo per vie traverse o di decomporlo in problemi più semplici che ci aiutano a risolvere il problema principale.

3.1 Rilassamento Lagrangiano

La prima tecnica che introduciamo è il rilassamento lagrngiano. Ricordiamo che il rilassamento lagrangiano è una strategia generale di risoluzione per programmi lineari che ci permette di decomporre i problemi per sfruttarne la struttura specifica. In quanto tale, questo approccio di risoluzione è perfettamente adattato per risolvere molti modelli con una struttura di rete sottostante al problema come il nostro. La strategia di soluzione lagrangiana ha diversi vantaggi significativi, ne elenchiamo i principali:

1. Poiché spesso è possibile scomporre i modelli in diversi modi e applicare il rilassamento lagrangiano a ogni differente scomposizione del problema, il rilassamento lagrangiano è un approccio di risoluzione molto flessibile. Infatti, grazie alla sua flessibilità, il rilassamento lagrangiano è meglio definito come un framework di soluzione piuttosto che una singola tecnica .

2. Nella scomposizione dei problemi, il rilassamento lagrangiano risolve gli n sottoproblemi principali come modelli indipendenti. Di conseguenza, l'approccio di soluzione ci permette di sfruttare qualsiasi metodologia o algoritmo noto per risolvere i sottoproblemi. In particolare, quando i sottoproblemi sono modelli di rete, l'approccio di risoluzione lagrangiano può sfruttare i vari algoritmi che abbiamo sviluppato in precedenza.

3. Come abbiamo già osservato, il rilassamento lagrangiano ci permette di sviluppare limiti sul valore della funzione obiettivo ottimale e spesso di convergere rapidamente soluzioni buone anche se non necessariamente ottimali, con garanzie di prestazioni associate, cioè un limite su quanto la soluzione potrebbe essere lontana dall'ottimalità (in termini di valore della funzione obiettivo). In molti casi nel contesto della programmazione intera, i limiti forniti dai metodi

di rilassamento lagrangiano sono migliori di quelli generati risolvendo il rilassamento lineare del problema in modo classico e, di conseguenza, il rilassamento lagrangiano è spesso un'alternativa interessante alla programmazione lineare come meccanismo di limitazione nei metodi di branch-and-bound per risolvere i problemi interi.

4. In molti casi possiamo utilizzare i metodi di rilassamento lagrangiano per sviluppare metodi di soluzione euristici efficaci per risolvere problemi complessi di ottimizzazione combinatoria e programmi interi. Andando quindi a risolvere sotto-componenti che ci facilitano il lavoro con il problema principale.

3.1.1 Subgradient Optimization Technique

In questa sezione approcciamo le tecniche di rilassamento lagrangiano e più nello specifico la Subgradient Optimization Technique, tutto questo servirà nelle sezioni successive come appoggio per diverse tecniche risolutive più avanzate per risolvere il problema multi commodity flow.

Come prima cosa partiamo da una formulazione generale di un modello di ottimizzazione che utilizzeremo come punto di partenza di questa trattazione, il seguente modello è formulato come un vettore di variabili decisionali x:

$$z^* = min\ cx \tag{18}$$

subject to

$$\mathcal{A}x = b \tag{19}$$

$$x \in X \tag{20}$$

Il modello appena descritto ha una funzione obiettivo lineare cx e un insieme di vincoli lineari espliciti $\mathcal{A}x = b$. Le variabili decisionali x sono anche vincolate a trovarsi in un dato insieme di vincoli X che, come vedremo, spesso modella la struttura come rete in cui scorre il flusso. Ad esempio, l'insieme di vincoli $X = \{x : \mathcal{N}x = q, 0 \leq x \leq u\}$ potrebbe contenere tutte le soluzioni ammissibili per un problema di flusso di rete con un vettore di domanda/offerta q. Oppure, l'insieme X potrebbe contenere i vettori di incidenza di tutti gli alberi di copertura o accoppiamenti di un dato grafo. A meno che non sia diversamente specificato, assumiamo che l'insieme X sia finito (ad esempio, per problemi di flusso di rete, porremo l'insieme finito delle soluzioni degli alberi di copertura).

Come suggerisce il nome, la procedura di rilassamento lagrangiano utilizza l'idea di rilassare i vincoli lineari espliciti portandoli nella funzione obiettivo con i moltiplicatori lagrangiani associati:

$$min\ cx + \mu(\mathcal{N}x - b) \tag{21}$$

subject to

$$x \in X \tag{22}$$

Il rilassamento lagrangiano applicato al problema appena enunciato detto anche sotto-problema lagrangiano è:

$$L(\mu) = min\{cx + \mu(\mathcal{N}x - b) : x \in X\} \tag{23}$$

Nella funzione lagrangiana è importante notare che poiché, nel formulare il rilassamento lagrangiano, abbiamo eliminato i vincoli $\mathcal{A}x = b$ dalla formulazione principale del problema, la soluzione del sottoproblema lagrangiano potrebbe non essere ammissibile per il problema originale. È possibile ottenere informazioni utili sul problema originale anche quando la soluzione del sottoproblema lagrangiano non è ammissibile per il problema originale?

Per rispondere a questa domanda ci viene in soccorso il lemma (**Lagrangian Bounding Principle**): *Per qualsiasi vettore μ di moltiplicatori lagrangiani, il valore $L(\mu)$ della funzione Lagrangiana è un lower bound del valore ottimo della funzione obiettivo z^* del problema di ottimizzazione originale.*

Come abbiamo visto dunque, per qualsiasi valore del moltiplicatore di Lagrange, $L(\mu)$ è un lower bound del valore ottimale della funzione obiettivo del problema originale. Per ottenere il limite inferiore più preciso possibile, dovremmo passare alla formulazione duale e risolvere il seguente problema di ottimizzazione:

$$L^* = max_\mu\ L(\mu) \tag{24}$$

Questo appena definito prende il nome di *Lagrangian multiplier problem* associato al problema di ottimizzazione originale. Sapendo la proprietà di **Weak Duality** affermiamo che il valore ottimo della funzione obiettivo L^* del *Lagrangian multiplier problem* è sempre un lower bound del valore ottimo della funzione obiettivo del problema (cioè, $L^* \leq z^*$).

Subgradient Optimization Technique. Nella risoluzione dei problemi di ottimizzazione con la funzione obiettivo $f(x)$ non lineare quindi con un vettore x di n dimensioni, ricercatori e professionisti spesso utilizzano variazioni della seguente idea

classica: formare il gradiente $\nabla f(x)$ di f definito come un vettore riga che ha come componenti $(\frac{\partial f(x)}{\partial x_1}, \frac{\partial f(x)}{\partial x_2} ... \frac{\partial f(x)}{\partial x_n})$. Ricordiamo dall'analisi avanzata che la derivata direzionale di f nella direzione d soddisfa l'uguaglianza:

$$\lim_{\theta \to 0} \frac{f(x + \theta d) - f(x)}{\theta} = \nabla f(x) d \tag{25}$$

Quindi, se scegliamo la direzione d in modo che $\nabla f(x) d > 0$ e ci spostiamo nella direzione d con una "step length" θ sufficientemente piccola, cioè cambiamo la nostra x in $x + \theta d$, ci muoviamo verso l'alto nella nostra funzione.

Questa semplice osservazione è alla base di una considerevole letteratura in programmazione non lineare nota come metodi del gradiente.

Supponiamo che nella risoluzione del problema dei moltiplicatori lagrangiani ci troviamo in un punto in cui la funzione Lagrangiana $L(\mu) = min\{cx + \mu(\mathcal{A}x - b) : x \in X\}$ ha una soluzione unica e differenziabile x. Poiché $L(\mu) = c\overline{x} + \mu(\mathcal{A}\overline{x} - b)$ e la soluzione x rimane ottimale per piccoli cambiamenti nel valore di μ, il gradiente in questo punto è $(\mathcal{A}\overline{x} - b)$, quindi un metodo del gradiente modificherebbe il valore di μ come segue:

$$\mu \leftarrow \mu + \theta(\mathcal{A}\overline{x} - b) \tag{26}$$

In questa espressione, θ è uno step length (uno scalare) che specifica quanto ci spostiamo nella direzione del gradiente. Si noti che questa procedura ha una interpretazione intuitiva. Se $(\mathcal{A}\overline{x} - b)_i = 0$, la soluzione x utilizza esattamente le unità richieste della risorsa i-esima e manteniamo il moltiplicatore di Lagrange μ_i di quella risorsa al suo valore attuale (non si sposta).

Se $(\mathcal{A}\overline{x} - b)_i < 0$, la soluzione x utilizza meno unità rispetto alle unità disponibili della risorsa i-esima e diminuiamo il moltiplicatore di Lagrange μ_i per quella risorsa; se invece $(\mathcal{A}\overline{x} - b)_i > 0$, la soluzione x utilizza più unità rispetto alle unità disponibili della risorsa i-esima e aumentiamo il moltiplicatore di Lagrange μ_i per quella risorsa.

Per risolvere il problema dei moltiplicatori di Lagrange, adottiamo un'estensione piuttosto naturale di questo approccio di soluzione. Facciamo sì che μ^0 sia una scelta iniziale qualsiasi del moltiplicatore lagrangiano; determiniamo i valori successivi μ^k per k = 1, 2, ..., dei moltiplicatori lagrangiani secondo la seguente formula:

$$\mu^{k+1} = \mu^k + \theta_k(\mathcal{A}x^k - b) \tag{27}$$

In questa espressione, x^k è qualsiasi soluzione del sottoproblema lagrangiano quando $\mu = \mu^k$ e θ^k è la dimensione del passo all'iterazione k.

Per garantire che questo metodo risolva il problema dei moltiplicatori lagrangiani, è necessario prestare attenzione nella scelta delle dimensioni dei passi. Se le scegliamo passi troppo "corti", l'algoritmo potrebbe rimanere bloccato al punto corrente e non convergere; se le dimensioni dei passi sono troppo "lunghi, le l'iterazione potrebbero superare la soluzione ottimale e magari oscillare tra due soluzioni non ottimali senza riuscire a convergere. Il compromesso seguente assicura che l'algoritmo trovi un equilibrio appropriato tra questi estremi e converga verso l'ottimo:

$$\theta_k \to 0 \text{and} \sum_{j=1}^{k} \theta_j \to \infty \tag{28}$$

Vediamo adesso come queste idee possono essere applicate al MC-MCF e come ci possono aiutare a risolvere in maniera efficiente il problema.

3.1.2 Riformulazione Programma Lineare

Per applicare il rilassamento lagrangiano al problema è necessario associare un moltiplicatore lagrangiano non negativo associato al nostro bundle constraint (3) definendo il seguente sotto problema lagrangiano:

$$L(w) = min \sum_{1 \le k \le K} c^k x^k \sum_{(i,j) \in A} w_{ij} (\sum_{1 \le k \le K} x_{ij}^k - u_{ij}) \tag{29}$$

equivalente a

$$L(w) = min \sum_{1 \le k \le K} \sum_{(i,j) \in A} (c_{ij}^k + w_{ij}) x_{ij}^k - \sum_{(i,j) \in A} w_{ij} u_{ij} \tag{30}$$

subject to

$$- \sum_{(i,j) \in FS} x_{ij}^k + \sum_{(j,i) \in BS} x_{ji}^k = b_i^k \quad \forall i \in N \quad \forall k \in K \tag{31}$$

$$x_{ij}^k \ge 0 \quad \forall (i,j) \in A \tag{32}$$

3.1.3 Tecnica risolutiva

Notiamo che nella seconda riformulazione (30) del programma lineare

$- \sum_{(i,j) \in A} w_{ij} u_{ij}$ nella funzione obiettivo del sottoproblema Lagrangiano è costante per qualsiasi scelta dei moltiplicatori di lagrangiani, dunque per un valore fissato di questi moltiplicatori, tale termine è costante e quindi possiamo ignorarlo.

La funzione obiettivo risultante per il sotto-problema Lagrangiano ha un costo di $c_{ij}^k + w_{ij}$ associato ad ogni variabile di flusso x_{ij}^k. Poiché nessuno dei vincoli in questo problema contiene le variabili di flusso per più di una delle commodity, il nostro problema si scompone in n problemi separati di flusso di costo minimo, uno per ogni commodity presente. La conseguenza di ciò è che per applicare la procedura di ottimizzazione del sotto-gradiente (descritta brevemente nella sezione 3.1.2) a questo problema, dovremmo alternativamente:

1. risolvere un insieme di problemi di flusso a costo minimo (per un valore fissato dei moltiplicatori di Lagrange w) con i coefficienti di costo $c_{ij}^k + w_{ij}$
2. aggiornare i moltiplicatori tramite le procedure algoritmiche descritte nella Sezione 3.1.2.

In questo caso, se y_{ij}^k denota la soluzione ottimale ai sotto-problemi di flusso a costo minimo quando i moltiplicatori di Lagrange hanno il valore w_{ij}^q all'iterazione q-esima, la formula di aggiornamento del sotto-gradiente diventa:

$$w_{ij}^{q+1} = [w_{ij}^q + \theta_q(\sum_{1 \leq k \leq K} y_{ij}^k - u_{ij})]^+ \tag{33}$$

Nella precedente espressione notiamo la dicitura $[\alpha]^+$, essa è da intendere come "la parte positiva di α" cioè il massimo tra $(\alpha, 0)$.

Il termine θ_q è uno scalare e indica di quanto ci stiamo spostando dalla soluzione corrente w_{ij}^q. Notiamo che questa formula di aggiornato del peso aumenta il moltiplicatore w_{ij}^q sull'arco (i,j) di $\sum_{1 \leq k \leq K} y_{ij}^k - u_{ij}$ se le soluzioni del sotto-problema y_{ij}^k stanno utilizzando più della capacità massima u_{ij} disponibile per l'arco (i,j), oppure il riduce moltiplicatore w_{ij}^q sull'arco (i,j) di $\sum_{1 \leq k \leq K} y_{ij}^k - u_{ij}$ se le soluzioni del sotto-problema y_{ij}^k stanno utilizzando meno della capacità massima u_{ij} disponibile per l'arco (i,j). Tuttavia, se la riduzione farebbe diventare negativo il moltiplicatore w_{ij}^{q+1}, riduciamo il moltiplicatore al valore zero in accordo con il concetto di parte positiva visto precedentemente.

È bene a questo punto della trattazione considerare il metodo dell'ottimizzazione del sotto-gradiente per il problema di flusso multicommodity come una procedura di soluzione valida per risolvere il programma lineare definito 1.2.1, questo grazie alla capacita di sfruttare la struttura speciale dei vincoli di bilancio di massa non rilassati specifici del problema.

3.2 Column Generation approch

Andremo ora a trattare una tecnica utilizzata per risolvere problemi di programmazione lineare di grandi dimensioni chiamata Column Generation approach. Questa tecnica si basa sul concetto di generare e aggiungere progressivamente nuove colonne al modello di programmazione lineare, focalizzandosi solo sulle colonne effettivamente rilevanti al fine di trovare la soluzione ottima.

Il processo di Column Generation incomincia con un set iniziale di colonne nel modello. Successivamente, viene risolto un problema di programmazione duale ristretto, noto in letteratura anche come problema maestro, per determinare i valori dei moltiplicatori di Lagrange corrispondenti ai vincoli del nostro problema primale. Utilizzando questi moltiplicatori, viene risolto un problema di generazione di colonne, noto come sotto-problema, che mira a trovare la migliore colonna aggiuntiva da aggiungere al modello. La nuova colonna viene quindi generata e aggiunta al modello originale.

Questo processo di generazione di colonne e aggiunta prosegue fino a quando non viene trovata una soluzione ottima o, in alternativa, viene soddisfatto un criterio di terminazione definito a priori. Inoltre, durante il processo, vengono gestite opportune variabili slack o variabili di surplus per garantire l'ammissibilità delle soluzioni intermedie.

Quello che andremo a fare è cercare di utilizzare questa strategia con il problema del MC MCF che come vedremo servirà come punto di partenza per applicare altre tecniche derivanti da essa come ad esempio la Dantzig-Wolfe Decomposition (3.3).

3.2.1 Riformulazione con Path Flows

Per semplificare la nostra discussione in questa sezione, consideriamo un caso particolare del problema di flusso multicommodity: assumiamo che ogni commodity k abbia un singolo nodo sorgente s^k e un singolo nodo destinazione t^k e uno specifico requisito di flusso di d^k unità tra i nodi sorgente e destinazione.

Assumiamo inoltre di non imporre limiti di flusso sulle singole commodity al di fuori dei vincoli di bundle. Pertanto, per ogni commodity k, i vincoli del sotto-problema $\mathscr{N} x^k = b^k, x^k \geq 0$ definiscono un problema di shortest path. In questo specifico modello, qualsiasi sia la scelta di w_{ij} dei moltiplicatori lagrangiani per i vincoli di bundle, il rilassamento lagrangiano richiede la soluzione di una serie di problemi di shortest path, uno per ogni commodity.

A questo punto, fatte le opportune assunzioni sul problema possiamo procedere a riformulare il problema di multi commodity min cost flow in termini di percorsi e

cicli di flusso piuttosto che con archi di flusso.

Sappiamo che possiamo formulare qualsiasi problema di flusso di rete utilizzando percorsi e cicli di flusso. In questo caso al fine di semplificare ulteriormente la nostra trattazione, assumiamo che per ogni commodity k il costo di ogni ciclo W nella rete sottostante sia non negativo. Il problema soddisfa questa condizione specifica, se ad esempio, tutti i costi di flusso degli archi sono non negativi.

Se imponiamo questa condizione di ciclo con costo non negativo, allora in una soluzione ottimale del nostro problema, il flusso su ogni ciclo è zero, quindi possiamo trascurare le variabili di flusso dei cicli.

Pertanto, in questa sezione, assumiamo che possiamo rappresentare qualsiasi soluzione potenzialmente ottimale come somma di flussi su percorsi diretti, dato che abbiamo detto che possiamo trascurare le variabili di flusso dei cicli.

Per ogni commodity k, definiamo p^k come la collezione di tutti i percorsi diretti dal nodo sorgente s^k al nodo destinazione t^k nella rete sottostante definita come G = (N, A). Nella formulazione del path flow, ogni variabile decisionale $f(P)$ rappresenta il flusso su un certo percorso P per la k-esima commodity, noi definiamo questa variabile per ogni percorso diretto P in P^k.

Definiamo ora $\delta_{ij}P$ come una variabile booleana che ci dice se P contiene o meno l'arco, quindi $\delta_{ij}P$ vale 1 se l'arco (i, j) è contenuto nel percorso P, altrimenti vale 0.

Il teorema di decomposizione del flusso di rete afferma che possiamo sempre decomporre un flusso di arco ottimale x_{ij}^k in path flows $f(P)$ come segue:

$$f(P) = x_{ij}^k \sum_{P \in P^k} \delta_{ij}(P) f(P) \tag{34}$$

Definiamo $c^k(P) = \sum_{(i,j)\in A} c_{ij}^k \delta_{ij}(P) = \sum_{(i,j)\in P} c_{ij}^k$ come il costo unitario del flusso sul percorso $P \in P^k$ rispetto alla commodoty k. Osserviamo che per ogni commodity k, se sostituiamo le variabili di flusso degli archi nella funzione obiettivo, scambiamo l'ordine delle sommatorie e raggruppiamo i termini, otteniamo che:

$$\sum_{(i,j)\in A} c_{ij}^k \delta_{ij} = \sum_{(i,j)\in A} c_{ij}^k [\sum_{P \in P^k} \delta_{ij}(P) f(P)] = \sum_{P \in P^k} c^k(P) f(P) \tag{35}$$

Questa osservazione mostra che possiamo esprimere il costo di qualsiasi soluzione sia come il costo dei flussi sugli archi sia come il costo dei flussi sui percorsi.

Sostituendo le path variable nella formulazione del flusso multi commodity, otteniamo la seguente formulazione equivalente del problema:

$$\sum_{1 \le k \le K} \sum_{P \in P^k} c^k(P) f(P) \tag{36}$$

subject to

$$\sum_{1\leq k\leq K}\sum_{P\in P^k}\delta_{ij}(P)f(P)\leq u_{ij}\quad\forall(i,j)\in A \tag{37}$$

$$\sum_{P\in P^k}f(P)=d^k\quad\forall k\in K \tag{38}$$

$$f(P)\geq 0\quad\forall k\in K\quad\forall P\in P^k \tag{39}$$

Nella formulazione di questo problema abbiamo utilizzato il *teorema di decomposizione del flusso*, secondo il quale possiamo scomporre qualsiasi arco di flusso ammissibile del sistema $\mathscr{N}x^k = b^k$ in un insieme di percorsi e cicli di flusso in modo tale che i path flows soddisfino la condizione di bilancio di massa (38).

Si osservi che la formulazione del flusso di percorso del problema di flusso multi commodity ha una struttura di vincoli molto semplice. Il problema ha un singolo vincolo per ogni arco (i,j) che afferma che la somma dei path flows che passano attraverso l'arco è al massimo u_{ij}, la capacità dell'arco.

Inoltre, il problema ha un singolo vincolo (38) per ogni commodity k che afferma che il flusso totale su tutti i percorsi che collegano il nodo sorgente s^k e il nodo destinazione t^k della commodity k deve essere uguale alla domanda d^k per questa specifica commodity.

Per una rete con n nodi, m archi e K commodity, la formulazione del path flows contiene $m + K$ vincoli (oltre alle restrizioni di non negatività imposte ai valori di flusso di percorso). In contrasto, la formulazione ad arco (6) contiene $m+nK$ vincoli in quanto contiene un vincolo di bilancio di massa per ogni combinazione di nodo e commodity.

Ragionando vediamo come tramite questa riformulazione riusciamo a ridurre di molto i vincoli presenti del problema. Provando a fare un esempio numerico per comprendere al meglio il valore dell'affermazione, prendiamo una rete con n = 1000 nodi e m = 5000 archi e con una commodity tra ogni coppia di nodi, ha approssimativamente K = n2 = 1.000.000 commodity. Pertanto, la formulazione del path flows contiene circa 1.005.000 vincoli. La formulazione per arch-flow contiene circa 1.000.005.000 di vincoli. Ma la differenza è ancora più evidente: poiché nessun percorso appare in più di uno dei vincoli 38, possiamo applicare una versione specializzata del metodo del simplesso, noto come metodo del simplesso di superiori generalizzati, per risolvere la formulazione del flusso di percorso in modo molto efficiente.

3.2.2 Condizioni di ottimalità

Ricordiamo che il metodo del simplesso rivisto della programmazione lineare mantiene una base ad ogni passo e utilizzando questa base determina un vettore di moltiplicatori del simplesso per i vincoli del problema.

Poiché la formulazione del flusso di percorso (3.2.1) contiene un vincolo di bundle per ogni arco e un vincolo di domanda (38) per ogni commodity, il programma lineare duale ha una variabile duale w_{ij} per ogni arco (questo è lo stesso prezzo dell'arco che abbiamo introdotto in precedenza durante la trattazione del rilassamento lagrangiano) e un'altra variabile duale σ^k per ogni commodity k = 1, 2, ..., K.

Rispetto a queste variabili duali, il costo ridotto $c_p^{\sigma,w}$ per ogni variabile del path flow f(P) è:

$$c_p^{\sigma,w} = c^k(P) + \sum_{(i,j)\in P} w_{ij} - \sigma^k \tag{40}$$

Le slackness conditions per la formulazione basata sugli archi assumono quindi la seguente formula:

Path flow complementary slackness conditions. I path flows delle commodity f(P) sono ottimali nella formulazione path flows (3.2.1) del problema di flusso multi commodity se e solo se, per alcuni prezzi degli archi w_{ij} e prezzi delle commodity σ^k, i costi ridotti e i flussi degli archi soddisfano le seguenti complementary slackness conditions:

1.
$$w_{ij}[\sum_{1\le k\le K}\sum_{p\in P^k}\delta_{ij}(P)f(P) - u_{ij}] = 0 \quad \forall (i,j)\in A \tag{41}$$

2.
$$c_p^{\sigma,w} \ge 0 \quad \forall k \in K \quad \forall P \in P^k \tag{42}$$

3.
$$c_p^{\sigma,w} f(P) = 0 \quad \forall k \in K \quad \forall P \in P^k \tag{43}$$

La condizione (1) afferma che il prezzo w_{ij} dell'arco (i,j) è zero se la soluzione ottimale f(P) non utilizza tutta la capacità u_{ij} dell'arco. In altre parole, se la soluzione ottimale non utilizza completamente la capacità di quell'arco, possiamo ignorare il vincolo (non assegnare un prezzo ad esso). Poiché il costo $c^k(P)$ del percorso P è semplicemente la somma dei costi degli archi contenuti in quel percorso, , possiamo

scrivere il costo ridotto del percorso P come:

$$c_p^{\sigma,w} = \sum_{(i,j)\in P} (c_{i,j}^k + w_{ij}) - \sigma^k \tag{44}$$

3.3 Dantzig-Wolfe Decomposition

Nella precedente sezione abbiamo mostrato come utilizzare una procedura column generation puo essere un'idea valida per risolvere il problema di flusso multicommodity formulato sfruttando l'Intuizione dei path flows.

In questa sezione, riprendiamo questo approccio di soluzione ma in un altro contesto, noto come decomposizione Dantzig-Wolfe. Immaginiamo che K diversi decision maker, così come un "coordinatore", stiano risolvendo il problema di flusso K-commodity. Ogni attore del problema ha un ruolo speciale nella risoluzione del problema. Il compito del coordinatore è quello di risolvere la formulazione a path flows del problema, che chiamiamo "master" o "problema di coordinazione". Nella risoluzione del problema, tuttavia, il coordinatore non genera le colonne del problema principale; invece, i K decision maker, con l'orientamento del coordinatore sotto forma di prezzi degli archi, generano queste colonne, con il decision maker k-esimo che genera le colonne del problema principale corrispondenti alla k-esima commodity.

Ciò che stiamo facendo in pratica è di scomporre il problema in due problemi principali (master e slave) dove gli slave grazie alla guida del master aumentano le colonne del problema principale per raggiungere una soluzione ottima del problema di coordinazione.

In generale, il coordinatore ha a disposizione solo un sottoinsieme delle colonne del problema principale. Dal momento che il coordinatore può al massimo risolvere il programma lineare limitato a questo sottoinsieme di colonne, ci riferiamo a questo programma lineare più piccolo come *restricted master problem*.

La formulazione a path flows del problema di flusso multicommodity ha $m + K$ vincoli:

1. uno per ogni commodity k, che specifica che il flusso della commodity k è d^k
2. uno per ogni arco (i,j), che specifica che il flusso totale su quell'arco è al più u_{ij}.

Il master risolve il problema principale limitato all'ottimalità utilizzando una qualsiasi tecnica di programmazione lineare, come ad esempio, l'algoritmo del simplesso, e

quindi deve determinare se la soluzione del problema principale limitato è ottimale per il problema originale o se qualche altra colonna ha un costo ridotto negativo.

A tal fine, il master trasmette il set ottimale di moltiplicatori simplex (o prezzi) del problema principale limitato, ovvero trasmette un prezzo dell'arco w_{ij} associato all'arco (i,j) e una lunghezza del percorso σ^k associata a ogni commodity k.

Dopo che il master ha trasmesso i prezzi, il decision maker per la commodity k determina il percorso a costo minore per trasportare d^k unità dal nodo di origine s^k al nodo destinazione t^k della commodity k, assumendo che ogni arco (i,j) abbia un costo associato di w_{ij} oltre al suo costo di arco c_{ij}^k.

Se il costo di questo shortest path è inferiore a c_{ij}^k, il decision maker k-esimo segnalerà questa soluzione al master come una soluzione migliorante del problema. Se il costo di questo percorso è uguale a c_{ij}^k, il decision maker k-esimo non deve segnalare nulla al coordinatore. (Il costo non sarà mai maggiore di c_{ij}^k perché, come mostrato dalla nostra discussione delle condizioni di ottimalità nella Sezione 3.2.2, il master sta già utilizzando un percorso di costo c_{ij}^k per la k-esima commodity nella sua soluzione ottimale.)

3.3.1 Riformulazione programma lineare

Riformuliamo adesso il problema modellando esattamente l'intuizione presentata mediante la rappresentazione shortest path:

$$min \sum_{1 \le k \le K} \sum_{p \in P^k} [c^k(P) + \sum_{(i,j) \in P} w_{ij}] f(P) \tag{45}$$

subject to

$$\sum_{p \in P^k} f(P) = d^k \quad \forall k \in K \tag{46}$$

$$f(P) \ge 0 \quad \forall k \in K \quad \forall P \in P^k \tag{47}$$

Questo corrisponde, a trovare il modo migliore per soddisfare la domanda per la k-esima commodity, dato che $c^k(P) = \sum_{(i,j) \in P} c_{ij}^k$, assumendo inoltre un costo di $c^k(P) + w_{ij}$ associato ad ogni arco (i,j).

Il percorso più breve per la coomodity k avrà un costo ridotto negativo se e solo se il costo per il problema shortest path k è inferiore a σ^k.

Ci riferiamo ai problemi risolti da ciascuno dei K decisori come sotto-problemi, poiché li utilizziamo esclusivamente allo scopo di generare nuove colonne per il problema principale ristretto o per dimostrare l'ottimalità della soluzione attuale.

Per concludere la trattazione riassumiamo l'intuizione che stiamo sfruttando. Nella Sezione 3.2 abbiamo interpretato questa procedura come una procedura di generazione di colonne che determina le variabili in ingresso risolvendo K diversi shortest path problem. Lo shortest path k corrisponde al problema del percorso più breve per la commodity k con l'aggiunta del costo w_{ij} associato all'arco (i,j).

Nell'interpretazione di Dantzig-Wolfe della procedura, un coordinatore, o master, centrale sta risolvendo il problema del flusso multicommodity riformulato con i percorsi limitato alle colonne che ha a disposizione.

Dopo aver ottenuto una soluzione ottimale per il problema principale ristretto, il coordinatore chiede a ciascuno dei K decision maker di risolvere un problema shortest path utilizzando un costo aggiuntivo dell'arco w_{ij} su ciascun arco (i,j), ponendo sostanzialmente una sorta di lower-bound alla soluzione del sotto-problema.

Ciascun decision maker (cioè, sotto-problema) fornisce quindi al coordinatore un nuovo percorso o afferma di non poter generare un percorso più breve rispetto a quelli attualmente utilizzati dal coordinatore.

Il metodo presentato è molto flessibile su diversi aspetti. Quando implementato come il metodo del simplesso rivisto, l'algoritmo manterrebbe una base di programmazione lineare e introdurrebbe una colonna ad ogni iterazione. Possiamo interpretare questo approccio come il mantenimento di un problema principale ristretto ad ogni passo con una sola colonna che non è in base.

Nella sua formulazione classica, l'algoritmo scarta qualsiasi colonna che lascia la base di programmazione lineare. Tuttavia, generare nuove colonne in generale richiedere tempo, quindi potrebbe essere vantaggioso conservare le vecchie colonne, poiché potrebbero successivamente avere un costo ridotto negativo e di conseguenza servire nuovamente al problema.

Per ovviare a questa inefficienza, nell'implementazione dell'algoritmo di decomposizione, si tende a conservare alcune o tutte le colonne che abbiamo generato in precedenza, implementando una sorta di "programmazione dinamica" spostando la complessità dal tempo allo spazio per salvare le colonne scartate.

Di conseguenza, quando risolviamo il problema principale ristretto, generalmente effettuiamo più di un cambio di base (per risolvere completamente il problema principale ristretto) prima di diffondere nuovi prezzi e risolvere nuovamente i sotto-problemi shortest path.

La procedura di generazione di colonne e la procedura di decomposizione condividono un altro potenziale vantaggio: la possibilità di risolvere il problema utilizzando

processori paralleli. Una volta determinati i prezzi degli archi w_{ij} in entrambe le procedure, i sotto-problemi shortest path sono indipendenti tra loro, quindi potremmo risolverli simultaneamente, utilizzando un processore separato per ogni problema.

Poiché ogni colonna del problema principale corrisponde a uno dei numeri finiti di percorsi per ciascuna commodity, la tecnica di soluzione della decomposizione sarà finita finché non scarteremo mai alcuna colonna del problema principale ristretto. (Alla fine, genereremmo ogni colonna nel problema principale completo). In alternativa, potremmo scartare le colonne solo quando la soluzione del problema principale ristretto migliora in modo significativo.

Notiamo infine che i K sotto-problemi generati non sono altro che il problema del rilassamento lagrangiano con un moltiplicatore di w_{ij} imposto su ogni arco (i,j). Di conseguenza, potremmo considerare il master come colui che imposta i moltiplicatori di Lagrange e risolve il problema dei moltiplicatori lagrangiani. In effetti, la Dantzig-Wolfe decomposition è un metodo efficiente per risolvere il problema dei moltiplicatori lagrangiani se misuriamo l'efficienza dal numero di iterazioni che un algoritmo esegue.

L'algoritmo di Dantzig-Wolfe risolve il problema dei moltiplicatori lagrangiani poiché il problema del flusso multicommodity è un programma lineare e, per i programmi lineari, il problema dei moltiplicatori lagrangiani ha lo stesso valore di funzione obiettivo del programma lineare. Purtroppo, nell'applicare la Dantzig-Wolfe decomposition, ad ogni iterazione il coordinatore deve risolvere un programma lineare con $m + K$ vincoli, e questo passaggio di aggiornamento dei moltiplicatori simplex è molto costoso.

Risolvere un programma lineare richiede molto più tempo rispetto all'aggiornamento dei moltiplicatori utilizzando il subgradient optimization. Poiché ogni aggiornamento dei moltiplicatori per la Dantzig-Wolfe decomposition è computazionalmente molto costoso, il metodo di decomposizione di Dantzig-Wolfe non si è dimostrato generalmente un metodo efficiente per risolvere il problema MC-MCF; tuttavia, Dantzig-Wolfe decomposition ha un vantaggio importante che la distingue dagli altri algoritmi basati su Lagrangiano: l'algoritmo di decomposizione di Dantzig-Wolfe mantiene sempre una soluzione feasible del problema. La soluzione dei sotto-problemi ci fornisce un lower bound sul valore ottimale del problema, quindi ad ogni passo abbiamo anche un limite su quanto la soluzione attuale è lontana dall'ottimalità in termini di valore della funzione obiettivo.

Pertanto, possiamo terminare l'algoritmo in qualsiasi passo non solo con una soluzione ammissibile, ma anche con la garanzia di quanto, in termini di valore della funzione obiettivo, quella soluzione sia lontana dall'ottimalità.

3.4 Resource-Directive Decomposiotion

Fino ad ora abbiamo affrontato due metodi che rientrano nella famiglia dei metodi di decomposizione del prezzo, infatti sia il rilassamento lagrangiano che la Dantzig-Wolfe decomposition sfruttano l'intuizione proposta nella sezione 2.2.1.

In questa sezione andiamo a considerare il Resource-Directive Decomposiotion che adotta un approccio diverso da quello presentato sino ad ora, di cui abbiamo dato un overview nella sezione 3.4.

Questo approccio risolutivo, invece di utilizzare i prezzi per decomporre il problema, assegna la capacità di ciascun arco alle singole commodity. Applicato alla formulazione del problema 1.2.1, l'approccio Resource-Directive Decomposiotion assegna $r_{ij}^k \leq u_{ij}^k$ unità della capacità di insieme u_{ij}^k dell'arco (i,j) alla commodity k, producendo la seguente riformulazione del problema Resource-Directive.

3.4.1 Riformulazione programma lineare

$$z = min \sum_{1 \leq k \leq K} c^k x^k \tag{48}$$

subject to

$$\sum_{1 \leq k \leq K} r_{ij}^k \leq u_{ij} \quad \forall (i,j) \in A \tag{49}$$

$$\mathscr{N} x^k = b^k \quad \forall k \in K \tag{50}$$

$$0 \leq x_{ij}^k \leq r_{ij}^k \quad \forall (i,j) \in A \quad \forall k \in K \tag{51}$$

Analizzando la riformulazione del problema si noti che il vincolo (49) assicura che l'allocazione complessiva delle risorse per l'arco (i, j) non superi la capacità congiunta di quell'arco. Indichiamo con $r = (r_{ij}^k)$ il vettore delle allocazioni di risorse.

Ora facciamo alcune osservazioni elementari su questo problema.

Il problema Resource-Directive Decomposiotion è equivalente al problema originale del flusso multicommodity (1.2.1) nel senso che:

1. se (x, r) è ammissibile in il problema Resource-Directive, allora x è ammissibile e ha la stessa funzione obiettivo value nel problema originale, problema Resource-Directive.

2. se x è ammissibile nel problema originale e settiamo $r = x$, allora (x, r) è ammissibile e ha la stesso valore di funzione obiettivo del problema Resource-Directive.

Ora consideriamo il seguente approccio sequenziale per risolvere il problema Resource-Directive Decomposiotion. Invece di risolvere il problema scegliendo contemporaneamente i vettori r e x, li scegliamo in modo sequenziale. Prima fissiamo le allocazioni di risorse r_{ij}^k e poi scegliamo il flusso x_{ij}^k. Indichiamo con $z(r)$ il valore ottimale del problema Resource-Directive per un valore fissato dell'allocazione delle risorse r e consideriamo il seguente problema:

$$min\ z(r) \tag{52}$$

subject to

$$\sum_{1 \leq k \leq K} r_{ij}^k \leq u_{ij} \quad \forall (i,j) \in A \tag{53}$$

$$0 \leq x_{ij}^k \leq r_{ij}^k \quad \forall (i,j) \in A \quad \forall k \in K \tag{54}$$

La funzione obiettivo $z(r)$ per questo problema è complicata. Conosciamo il suo valore solo implicitamente come soluzione di un problema di ottimizzazione nelle variabili di flusso x_{ij}^k.

Inoltre, si noti che per ogni valore fisso delle variabili della risorsa r_{ij}^k, il problema Resource-Directive si decompone in un sotto-problema di flusso di rete separato per ogni commodity. Quindi, $z(r) = \sum_{k \in K} z^k(r^k)$, dove il valore $z^k(r^k)$ del k-esimo sotto-problema è dato da:

$$z^k(r^k) = min \sum_{1 \leq k \leq K} c^k x^k \tag{55}$$

subject to

$$\mathscr{N} x^k = b^k \quad \forall k \in K \tag{56}$$

$$0 \leq x_{ij}^k \leq r_{ij}^k \quad \forall (i,j) \in A \quad \forall k \in K \tag{57}$$

Il Resource-Directive Decomposiotion è equivalente al problema di allocazione delle risorse nel senso che:

1. se (x, r) è ammissibile nel problema Resource-Directive, allora r è ammissibile nel problema di allocazione delle risorse e $z(r) \leq cx$

2. se r è ammissibile nel problema di allocazione delle risorse, allora per qualche vettore x, (x, r) è ammissibile nel problema originale ed $z(r) = cx$

Le proprietà ottenute successivamente alla riformulazione sopra descritta implicano che invece di risolvere direttamente il problema del flusso multicommodity,

possiamo decomporlo in un problema di allocazione delle risorse con una struttura di vincoli molto semplice, dove abbiamo un unico vincolo di disuguaglianza, ma con una funzione obiettivo complessa $z(r)$.

Sebbene la struttura complessiva della funzione obiettivo sia complicata, è valutarla è semplice: per trovarne il valore per qualsiasi scelta del vettore r di allocazione delle risorse, dobbiamo semplicemente risolvere K problemi di flusso a costo minimo con singola commodity.

Un altro modo di vedere la funzione obiettivo $z(r)$ è come il costo del programma lineare in funzione dei parametri del lato destro r. Cioè, ogni valore r per il vettore di allocazione definisce i valori dei parametri del lato destro per questo programma lineare. Un risultato ben noto nel programma lineare ci mostra che la funzione ha una forma speciale. Esponiamo questo risultato per un problema di programmazione lineare generale che contiene il problema del flusso multicommodity come caso speciale.

3.4.2 Risoluzione Resource-Directive Decomposiotion

Vi sono un certo numero di approcci algoritmici per risolvere i modelli Resource-Directive che abbiamo introdotto in questa sezione.

Nella seguente sezione, delineeremo alcuni approcci di base.

Poiché la funzione $z(r)$ non è differenziabile (perché è lineare a tratti), non possiamo utilizzare metodi di ottimizzazione del gradiente dalla programmazione non lineare per risolvere il problema, come quelli affrontati nella sezione del rilassamento lagrangiano.

Potremmo, invece, utilizzare diversi altri approcci che non implicano lo studio della funzione $z(r)$. Ad esempio, potremmo approcciare il problema con la tecnica del miglioramento locale in $z(r)$ utilizzando un metodo euristico.

Una tale possibilità, potrebbe essere l'utilizzo di un "arc-at-a-time approach" aggiungendo 1 a $r_{pq}^{k'}$ e sottraendo 1 a $r_{pq}^{k''}$ per un certo arco (P, q) per due commodity generiche k' e k", scegliendo l'arco e le commodities ad ogni passo, utilizzando un criterio (ad esempio, le scelte che danno la maggiore diminuzione del valore della funzione obiettivo ad ogni passo). Questo approccio è facile da implementare, ma non garantisce la convergenza a una soluzione ottimale.

Si noti che possiamo considerare questo approccio come la variazione dell'allocazione delle risorse ad ogni passo secondo la formula:

$$r \leftarrow r + \theta\gamma \tag{58}$$

con una lunghezza del passo di $\theta = 1$ e una direzione di spostamento $\gamma = \gamma_{pq}^{k}$ data da $\gamma_{pq}^{k'} = 1$, $\gamma_{pq}^{k''} = -1$ e $\gamma_{pq}^{k} = 0$ per tutte le altre combinazioni di arco-commodity.

Tuttavia, prendendo in prestito idee dalla subgradient optimization, potremmo utilizzare un approccio di ottimizzazione scegliendo la direzione di spostamento γ come un subgradient corrispondente all'allocazione delle risorse r.

Un approccio valido potrebbe essere quello di cercare un subgradient o comunque una direzione di spostamento γ e una specifica lunghezza del passo θ che mantengano contemporaneamente sia l'ammissibilità che la convergenza a una soluzione ottimale r del problema Resource-Directive.

Adotteremo una strategia a due fasi, come prima cosa andiamo a determinare una direzione del subgradient e una lunghezza di passo θ che garantiscono la convergenza, a condizione che vi fossero vincoli imposti sulle allocazioni.

Se il passaggio a $r+\theta\gamma$ ci dà una soluzione non ammissibile, trasformiamo r in un punto r' che mantenga l'ammissibilità, ma garantisca la convergenza. Per applicare questo approccio, dobbiamo essere in grado di rispondere a due domande di base:

1. Come possiamo trovare un subgradient γ di $z(r)$ in un punto r dato?

2. Come potremmo trasformare qualsiasi allocazione di risorse non ammissibile $r + \theta\gamma$ in modo che diventi ammissibile per il problema di resource-choice, ovvero soddisfi i tutti i vincoli del problema?

Notiamo che quando abbiamo applicato la subgradient optimization al problema dei moltiplicatori lagrangiani nella sezione (3.1.2), i moltiplicatori lagrangiani erano o senza vincoli o vincolati solamente a essere non negativi, quindi abbiamo utilizzato direttamente la formula di aggiornamento $r \leftarrow r+\theta\gamma$ o una sua leggera modifica per garantire che i moltiplicatori di Lagrange rimanessero non negativi. Pertanto, nel contesto del rilassamento lagrangiano, la fattibilità era facile da garantire.

Per rispondere alla prima domanda, ricordiamo che un sotto-gradiente γ di $z(r)$ nel punto $r = \overline{r}$ è qualsiasi vettore che soddisfa la seguente condizione:

$$z(r) \geq z(\overline{r}) + \gamma(r - \overline{r}) \quad \forall\ r = (r^1, r^2, ..., r^k) \text{ con } r^k \in R^k \tag{59}$$

In questa espressione, R^k è l'insieme delle allocazioni di risorse per la commodity k per cui il sotto-problema (52) è ammissibile. La seguente proprietà mostra che per trovare un qualsiasi sotto-gradiente del genere, possiamo lavorare con ciascuna delle funzioni costituenti $z^k(r^k)$ in modo indipendente.

Sia γ^k un sotto-gradiente di $z^k(r^k)$ nel punto $\overline{r}^k$, allora $\gamma = (\gamma^1, \gamma^2, ..., \gamma^k)$ sono dei sotto-gradienti di $z(r)$ con $\overline{r} = (\overline{r}^1, \overline{r}^2, ..., \overline{r}^k)$

Questo risultato ci mostra che per ottenere un sotto-gradiente di $z(r)$, possiamo trovare un sotto-gradiente per ciascuna funzione $z^k(r^k)$. Fortunatamente, come mostrato dal seguente risultato, queste informazioni saranno un sottoprodotto di quasi ogni procedura di soluzione per risolvere il sotto-problema (55). Poiché questo risultato è generale e si applica a contesti applicativi più ampi, lo esprimiamo per un problema generale di flusso di rete (ovvero, includiamo l'indice k).

Consideriamo il problema di flusso di rete $z(q) = min\{cx : Xx = b, o \leq X \leq q\}$, con un vettore parametrico q di upper bound non negativi sui flussi degli archi. Supponiamo che x^* sia una soluzione ottima di questo problema quando $q = q^*$, e che x^* insieme al vettore dei costi ridotti c^π soddisfino le condizioni di ottimalità per il flusso a costo minimo. Definiamo i vettori $\mu = (\mu_{ij})$ settando $\mu_{ij} = 0$ se $x^*_{ij} < q^*_{ij}$ e $\mu_{ij} = c^\pi_{ij}$ se $x^*_{ij} = q^*_{ij}$. Allora per qualsiasi vettore non negativo q' per cui il problema è ammissibile, $z(q') \geq < (q^*) + \mu(q' - q^*)$. Quindi μ è il sotto gradiente della funzione obiettivo parametrica $z(\mu)$ con $\mu = \mu^*$.

Per applicare la proprietà sopra definita, è necessario risolvere ciascun sotto-problema (55), indipendente. Dopo aver risolto il sottoproblema per la k-esima commodity con $q = r^k$, poniamo $\gamma?k$ uguale al vettore μ. Successivamente, utilizziamo la Proprietà di decomposizione dei gradienti e poniamo $\gamma = (\gamma^1, \gamma^2, ..., \gamma^k)$. Questi risultati ci mostrano come utilizzare qualsiasi algoritmo di flusso a costo minimo che produca costi ridotti $c?\pi$ così come un vettore di flusso ottimale per trovare un sotto-gradiante.

Una volta ottenuto un sotto-gradiante γ di $z(r)$, possiamo utilizzare la formula del sotto-gradiante $r \leftarrow r + \theta\gamma$ per trovare la nuova allocazione delle risorse r. Se il vettore di allocazione delle risorse risultante r è ammissibile (cioè soddisfa il vincolo $\sum_{1 \leq k \leq K} r^k_{ij} \leq u_{ij}$), ci spostiamo su questo punto.

Tuttavia, se la nuova allocazione delle risorse r non ammissibile, dobbiamo modificarla per garantire l'ammissibile, spostandoci invece su un altro punto r.

Un approccio che assicura che l'algoritmo converga con la scelta corretta delle dimensioni dei passi è quello di scegliere r come il punto ammissibile che è il più vicino possibile a r nel senso di minimizzare la quantità$\sum_{1 \leq k \leq K} \sum_{)i,j) \in A} (r^k_{ij} - \overline{r}^k_{ij})^2$ (Distanza euclidea).

4 Implementazione

Questo capitolo termina la fase teorica della tesina, procediamo quindi con una serie di sperimentazioni svolte con l'obiettivo di applicare le tecniche teoriche discusse a problemi reali.

4.1 Strumenti utilizzati

Durante la fase sperimentativa abbiamo deciso di utilizzare i seguenti strumenti:

- Cplex: Solver di proprietà IBM utile alla risoluzione efficiente di problemi di programmazione lineare.
- Python: Linguaggio di programmazione molto utilizzato in ambiente scientifico
- [3]Pyomo : Libreria per il linguaggio python utile alla formulazione di problemi di programmazione lineare e alla loro risoluzione tramite molteplici solver supportati tra cui Cplex.

Abbiamo deciso di utilizzare Python e la Libreria Pyomo in modo da poter parsificare agevolmente i dati in input da fornire al solver e per poter scrivere algoritmi che ragionino sulle soluzioni fornite dai solver. Inoltre questa soluzione ci permette di essere indipendenti dallo specifico solver sottostante e quindi permette di poter valutare anche le performance di molteplici solver.

4.2 Dataset

Per la ricerca del dataset su cui svolgere la sperimentazione ci siamo riferiti ai dataset messi a disposizione sul sito dell'università di Pisa [2].
Tra i vari dataset a disposizione abbiamo scelto "The hydrotermal problem" in quanto rappresenta dati provenienti da una istanza reale del problema. Le istanze di questo problema sono state messe a disposizione da Jordi Castro e sono relative all'ambito di generazione di energia idrotermica.
Il dataset si compone di 4 file di input di varie dimensioni specifiche in formato PPRN:

- cinca2.dat: Numero Nodi: 37, Numero Archi: 153, Numero commodities: 4
- minsil7.dat: Numero Nodi: 99, Numero Archi: 315, Numero commodities: 4
- navia90.dat: Numero Nodi: 37, Numero Archi: 117, Numero commodities: 4

- minsil19.dat: Numero Nodi: 685, Numero Archi: 2141, Numero commodities: 4

Queste istanze non sono particolarmente complesse ma sono state scelte per la loro rappresentatività di problemi reali.
Per la parsificazione del formato PPRN ci siamo riferiti alla documentazione dello strumento Graph, software dell'università di pisa per la parsificazione di problemi di programmazione lineare.

4.3 Soluzioni proposte

4.3.1 Programmazione Lineare

Il primo approccio risolutivo esplorato è stato la risoluzione della formulazione originale del problema (6) tramite il solver Cplex.
Questa soluzione si è formalizzata quindi in un programma python in grado di parsificare i dataset in input e tradurli nel formato pyomo in modo che possa occuparsi della risoluzione tramite solver Cplex. La formulazione in programmazione lineare del problema del MC MCF viene tradotta nel seguente modo:

- Funzione obiettivo (6):

```
{(
    sum(model.x[edge, k]*model.edgeCosts[edge]
    for edge in model.edges for k in commodities)
    )}.
```

- Vincoli di bilanciamento (7):

```
for k in model.commodities:
for n in model.nodes:
    enteringEdges = []
    exitingEdges = []
    for edge in model.edges:
        if startNodes[edge] == n:
            exitingEdges.append(edge)
        if endNodes[edge] == n:
            enteringEdges.append(edge)
```

```
model.balancesConstraint.add(
    (sum(model.x[edge, k] for edge in enteringEdges)
    - sum(model.x[edge, k] for edge in exitingEdges)) ==
    model.nodesBalances[(k, n)]
)
```

- Vincoli di bundle (8):

```
for edge in edges:
model.balances_constraint.add(
    sum(model.x[edge, k] for k in commodities) <=
    model.edge_capacity[edge]
)
```

Questo primo approccio risolutivo ha fornito ottimi risultati applicativi in quanto risolve quasi istantaneamente ogni istanza del problema all'ottimo:

- cinca2.dat ticks: 0.96
- minsil7.dat ticks: 8.01
- navia90.dat ticks: 1.59
- minsil19.dat ticks: 240.34

La velocità media del calcolatore su cui abbiamo risolto queste istanze lavora in media a (1000 ticks/s) quindi tutte soluzioni sono state fornite ampiamente sotto il secondo di calcolo.

Questo risultato è dovuto fortemente dalla dimensione contenuta dei problemi presi in causa.
Per proseguire lo studio di questo approccio risolutivo abbiamo svolto un po' di sperimentazione su questa formulazione cercando di limitare il più possibile l'ottimizzazione automatica svolta dal solver cplex e cercando quindi di risolvere il problema con approcci più simili a quelli studiati durante il corso.
Disabilitando il pre-processing e forzando il solver ad usare il simplesso primale però le prestazioni non sono variate significativamente.

4.3.2 Rilassamento Lagrangiano

Il secondo approccio risolutivo analizzato è il rilassamento lagrangiano.
Applicare questo approccio richiede due modifiche all' approccio discusso nella sezione precedente:

- Modifica della formulazione lineare: E' necessario modificare la formulazione del problema passando alla definizione (30).
- Implementazione di un approccio di discesa del gradiente.

Per il primo punto la riformulazione si è svolta eliminando il vincolo di bundle e modificando la funzione obiettivo secondo la nuova definizione (30):

```
(
    sum(model.x[edge, k]*(model.edge_costs[(k,edge)]+coefficient[edge])
    for edge in model.edges for k in commodities)
)
```

E' importante notare come nella funzione obiettivo non sia presente il termine $\sum_{(i,j)\in A} w_{ij}u_{ij}$.
Questo termine non è inserito in quanto esso è costante e quindi può essere ignorato nella valutazione della funzione obiettivo, come enunciato nel corrispondente capitolo.
Per il secondo punto, ovvero l'implementazione della discesa del gradiente, abbiamo deciso di implementare una versione molto basilare della tecnica per valutarne le prestazioni.
La nostra implementazione inizializza tutti i coefficienti lagrangiani per gli archi a 0, imposta il valore θa 0.1 e imposta il numero massimo di iterazioni a 50.
L'approccio consiste quindi nella risoluzione del problema lagrangiano con i relativi coefficienti, nell'analisi della soluzione proposta e della sua ammissibilità, dell'aggiornamento dei pesi con le regole discusse nella parte teorica con il θ costante e nella riesecuzione finchè non si ha una soluzione ammissibile o si supera il numero massimo di iterazioni.
Ovviamente, data la semplicità di come è stato scelto il parametro θ costante non riusciamo mai a raggiungere una soluzione ammissibile in quanto abbiamo notato come l'algoritmo tende a oscillare attorno a punti di ammissibilità ma non li raggiunge.
Analizzando i risultati dal punto di vista dell'ammissibilità otteniamo però comunque dei buoni risultati:

- cinca2.dat: Numero di archi non ammissibili: 7, 5%

- minsil7.dat: Numero di archi non ammissibili: 50, 18%
- navia90.dat: Numero di archi non ammissibili: 17, 15%
- minsil9.dat: Numero di archi non ammissibili: 342, 16%

Secondo i risultati ottenuti in media abbiamo solo il 13.5% degli archi inammissibili rispetto ai vincoli di bundle, mentre i restanti rispettano i vincoli imposti.
Dal punto di vista prestazionale invece non otteniamo risultati positivi. In media, sui dataset a nostra dispozione, la risoluzione del problema lineare rilassato richiede $\frac{1}{10}$ del tempo di risoluzione del problema lineare non rilassato. Applicando quindi 50 iterazioni di algoritmo tendiamo a richiedere 5 volte più tempo rispetto alla risoluzione del problema originale.
Secondo le nostre analisi, questo è causato da due fattori principali:

- Dimensioni del problema: Le istanze su cui lavoriamo sono di dimensioni contenute, soprattutto sul numero di commodities K che è costante a 4 per tutti i problemi. Sulla base di ciò i vincoli di bundle devono legare solamente 4 sottoproblemi di MCF assieme, non complicando troppo la struttura del problema. Sulla base della stessa intuizione quando svolgiamo il rilassamento Lagrangiano dividiamo il problema originale in 4 sottoproblemi indipendenti, quindi non possiamo aspettarci speed-up elevati dato il basso numero di parallelismo ottenuto.
- Discesa del gradiente: L'approccio utilizzato per la discesa del gradiente è basilare, sfruttando approcci più elaborati si potrebbe ottenere una discesa più veloce e un rallentamento nell'intorno della soluzione ammissibile. Questo permetterebbe sia di diminuire il numero di iterazioni sia di raggiungere effettivamente la soluzione ammissibile.

References

[1] P. Erdős, *A selection of problems and results in combinatorics*, Recent trends in combinatorics (Matrahaza, 1995), Cambridge Univ. Press, Cambridge, 2001, pp. 1–6.

[2] Università di Pisa 02/11/2021, *Multicommodity Flow Problems*, https://commalab.di.unipi.it/datasets/mmcf/

[3] Pyomo, *Python-based, open-source optimization modeling language*, https://pyomo.readthedocs.io/en/stable Sandia National Laboratories