OTTIMIZZAZIONE

Programmazione Lineare Intera (PLI)

1) Imposto il sistema di disequazioni ricavandolo dalle condizioni che trovo nel testo dell'esercizio. Una di queste condizioni (implicita) è che le variabili siano non nulle (≥ 0) e intere
2) Disegno la regione ammissibile F, identificata da un poliedro (solitamente a 3 o 4 lati)
3) Disegno il fascio di rette della funzione obiettivo ψ (ovvero quello che richiede l'esercizio, ad esempio massimizzare il profitto) ponendo ψ = costante
4) Determino la direzione di miglioramento del fascio di rette, ovvero in che direzione devo spostare la mia retta affinché soddisfi la richiesta dell'esercizio
5) Determino l'ultima intersezione del fascio di rette con la regione ammissibile lungo la direzione di miglioramento; questa sarà la soluzione ottima (frazionaria)
6) Applico il metodo Branch-And-Bound per trovare la soluzione ottima intera:
    a. Inizio ad applicare una nuova restrizione alla variabile imposta dal testo dell'esercizio
    b. Questa restrizione mi genera due nuovi sottoproblemi (branch = ramifica): considero per primo quello imposto dal testo
    c. Disegno la nuova regione ammissibile
    d. Trovo la soluzione ottima del rilassamento continuo della nuova regione ammissibile
        i. Se è continua, ho finito di esplorare questo nodo e esploro i rimanenti
        ii. Se è frazionaria, calcolo l'upper bound di questa soluzione (U) sostituendo i suoi valori alla funzione obiettivo e arrotondando all'intero più vicino per difetto; poi ricomincio dal punto a.
        iii. Se la regione ammissibile è vuota, scrivo Ø a fianco del nodo (bound = limita) ed esploro i rimanenti
    e. Finite le iterazioni, la soluzione ottima sarà quella intera di valore più elevato se voglio massimizzare la funzione obiettivo o quella di valore più basso se voglio minimizzarla.

Trucco: ogni upper bound mi dice quanto vale al massimo la soluzione ottima da quel nodo in poi; pertanto, se visitando un altro nodo trovo una soluzione ottima intera pari o superiore all'upper bound precedente, posso evitare di esplorare il nodo in quanto il problema mi chiede UNA soluzione ottima; il nodo precedente viene quindi UCCISO.

Ulteriori richieste

Se ci viene chiesto di studiare la soluzione ottima del rilassamento continuo con una restrizione con un parametro al posto del termine noto ($5x_1 + 4x_2 \leq \alpha$), guardo com'è la regione ammissibile al variare di α

- se la regione ammissibile ha la forma di un quadrilatero, la soluzione ottima sarà l'intersezione tra le due rette (quella nota e quella con il parametro)
- se la regione ammissibile si riduce a un triangolo, la soluzione ottima avrà una coordinata nulla trovo quindi il caso limite, ovvero il valore di α per cui dal quadrilatero passo al triangolo e scrivo la soluzione ottima generale al variare di α.

Se ci viene chiesto di studiare la soluzione ottima del rilassamento continuo per una funzione obiettivo con parametro come coefficiente ($\beta x_1 + x_2$), stabilisco come varia la mia retta al variare di β. In generale, la retta ruoterà e distinguerò alcuni casi limite (ad esempio, se è parallela a una certa retta piuttosto che a un'altra)

che mi faranno variare le soluzioni ottime. Come prima, alla fine unisco tutti i risultati e ottengo le soluzioni ottime al variare del parametro β.

Problemi sui grafi

- Albero ricoprente di costo minimo (SST – Shortest Spanning Tree) → si applica solo a grafi non orientati → **Algoritmo di Prim**
- Cammino più corto tra due vertici (SP – Shortest Path) → **Algoritmo di Dijkstra**

Algoritmo di Prim – Strutture dati

- E' → insieme dei lati scelti (Ø all'inizio)
- W → insieme dei vertici contenuti nell'albero "in costruzione" (all'inizio W = $\{v_1\}$)
- b(v) (v ∈ V\W) → vertice di W più vicino al vertice v

Algoritmo di Prim – Codice

```
Begin
  W:={v1};
  E':= Ø;
  for each v ∈ V\{v1} do b(v):= v1;
  while (w≠V) do begin
    determina v* ∈ V\W tale che c(v*,b(v*)) è minimo;
    w:=w U {v*};
    E':=E' U {(v*,b(v*))};
    for each v ∈ V\W do
      if c(v,v*)<c(v,b(v)) then b(v):=b*;
  end;
End.
```

Nota: per trovare l'albero ricoprente di costo massimo, mi basta considerare i costi come negativi e applicare Prim (ovviamente il costo totale andrà nuovamente cambiato di segno per risultare positivo!), oppure sostituire, nell'algoritmo di Prim, al posto della parola "minimo" la parola "massimo".

Algoritmo di Dijkstra – Strutture dati

- s → vertice iniziale; t → vertice finale
- W → insieme dei vertici raggiunti con il cammino minimo da s
- pred(v) (v ∈ V) → vertice che precede v nel cammino "corrente" da s a v
- L(v) (v ∈ V) → costo del cammino minimo da s a v che passa solo per vertici di W

Algoritmo di Dijkstra – Codice

```
Begin
  W:={s};
  pred(s):=NULL;
  L(s):=0;
  for each v ∈ V\{s} do begin
    pred(v):=s;
    L(v):=c(s,v);
  end;
  while W≠V do begin
```

```
   determina v* Є V\W tale che L(v*) è minimo;
   W:=W U {v*};
   for each v Є V\W do
    if (L(v*)+c(v*,v)<L(v)) then begin
     L(v):=L(v*)+c(v*,v);
     pred(v):=v*;
    end;
  end;
End.
```

Nota: in realtà, l'algoritmo di Dijkstra trova il cammino minimo da s a tutti gli altri vertici.

Nota: per trovare il cammino da s a t tale che il massimo costo di un arco sia minimizzato, mi basta sostituire al posto di *if L(v*)+c(v*,v)<L(v) then L(v):=L(v*)+c(v*,v); pred(v):=v** le istruzioni *if L(v)>max{L(v*),c(v*,v)} then L(v):=max{L(v*),c(v*,v)}; pred(v):=v*;*

Metodo CPM (Cost Plus Method) – Cammino massimo da s a t

Il grafico a cui lo applico deve essere <u>orientato</u> e <u>aciclico</u> (non deve contenere cicli orientati)

1) Procedura NUMBER: numero i vertici in modo che da nessun vertice punti a uno di indice inferiore
   a. Considero un vertice con nessun arco entrante e gli do il primo indice libero
   b. Cancello il vertice considerato e tutti i suoi archi
   c. Ripeto il procedimento finché non ho coperto tutti i vertici
2) Calcolo i TMIN per ogni vertice

   $TMIN_0$:=0;

   for k:=1 to n+1 do

   $TMIN_k$:= max {$TMIN_i + d(v_i,v_k)$, … }; (per ogni arco entrante in v_k)
3) Memorizzando pred(v_i) per ogni vertice ricostruisco facilmente i cammini

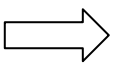
Memorizzazione di grafi – Metodo Forward Star

Il mio grafo viene memorizzato attraverso tre vettori:

- f o u: indica a quali vertici è collegato ciascun vertice; per utilizzarlo, numero da 1 a n i valori presenti e considero il secondo vettore.
- p: definito in modo tale che l'insieme dei vertici collegati a v_i va da p[i] fino a p[i+1]-1 in f.
- c: indica i costi di ciascun arco (in corrispondenza con f)

Esempio

f = [2 4 5 2 3 2 3 4]
p = [1 4 4 5 6 9]
c = [7 4 3 1 7 2 9 1]



1 2 3 4 5 6 7 8
f = [2 4 5 | | 2 | 3 | 2 3 4]
v_1 v_2 v_3 v_4 v_5
c = [7 4 3 | | 1 | 7 | 2 9 1]