Appunti per il corso di Ricerca Operativa 2

Capitolo 1

Introduzione

In questo corso ci occuperemo di problemi di *ottimizzazione*. Tali problemi sono tipicamente raggruppati in classi ognuna delle quali é formata da piú istanze. Diamo ora una definizione precisa di classe di problemi di ottimizzazione e di istanze di tale classe.

Definizione 1 *Una classe di problemi di ottimizzazione é formata da un insieme $\mathcal{I}$ di istanze. Ogni istanza $I \in \mathcal{I}$ é rappresentata da una coppia (f_I, S_I) dove*

$$f_I : S_I \to R$$

é detta funzione obiettivo *e S_I é detta* regione ammissibile. *Se il problema é di massimo, allora il problema si rappresenta nel modo seguente*

$$\max_{x \in S_I} f_I(x)$$

e risolvere l'istanza vuol dire trovare un $x_I^* \in S_I$ tale che

$$f_I(x_I^*) \geq f_I(x) \quad \forall\ x \in S_I;$$

se il problema é di minimo, allora il problema si rappresenta nel modo seguente

$$\min_{x \in S_I} f_I(x)$$

e risolvere l'istanza vuol dire trovare un $x_I^* \in S_I$ tale che

$$f_I(x_I^*) \leq f_I(x) \quad \forall\ x \in S_I.$$

In entrambi i casi il punto x_I^* viene chiamato ottimo (globale) *dell'istanza I del problema, mentre $f_I(x_I^*)$ viene detto* valore ottimo *dell'istanza.*

Alcuni problemi giá incontrati sono problemi di ottimizzazione. In particolare i problemi di PL (Programmazione Lineare) formano una classe le cui istanze

hanno la seguente forma:

$$\begin{gathered}\max\ c_I x \\ A_I x \leq b_I \\ x \geq 0\end{gathered}$$

dove

$$f_I(x) = c_I x \quad S_I = \{x \in R^n \ : \ A_I x \leq b_I,\ x \geq 0\}.$$

Un'altra classe di problemi di ottimizzazione é quella dei problemi di PLI (Programmazione Lineare Intera) le cui istanze hanno la seguente forma:

$$\begin{gathered}\max\ c_I x \\ A_I x \leq b_I \\ x \geq 0 \qquad x \in Z^n\end{gathered}$$

dove

$$f_I(x) = c_I x \quad S_I = \{x \in Z^n \ : \ A_I x \leq b_I,\ x \geq 0\}.$$

All'interno delle classi dei problemi di ottimizzazione possiamo riconoscere due grandi categorie.

Ottimizzazione Combinatoria In ogni istanza I la regione ammissibile S_I contiene un numero finito o un'infinitá numerabile di punti.

Ottimizzazione Continua La regione ammissibile S_I puó contenere un'infinitá non numerabile di punti.

Alla prima categoria appartiene la classe dei problemi di PLI, alla seconda la classe dei problemi di PL. In questo corso ci occuperemo prevalentemente di problemi di ottimizzazione combinatoria ma dedicheremo una parte anche all'ottimizzazione continua. Nel capitolo 2 introdurremo tre esempi di problemi di ottimizzazione combinatoria, il problema dell'albero di supporto a costo minimo, il problema dello zaino e il problema del commesso viaggiatore, che ci aiuteranno ad introdurre diversi concetti legati alla complessitá dei problemi. Nel capitolo 3 ci concentreremo su alcuni problemi "facili", il problema del flusso a costo minimo, del flusso massimo, di matching, del trasporto e di assegnamento. Nei capitoli 4 e 5 sposteremo l'attenzione su problemi "difficili" introducendo esempi di approcci esatti (branch-and-bound e programmazione dinamica) nel capitolo 4 e algoritmi di approssimazione e tecniche euristiche nel capitolo 5. Infine nel capitolo 6 introdurremo alcuni risultati di ottimizzazione continua.

Capitolo 2

Cenni di complessitá dei problemi di ottimizzazione

In questo capitolo vogliamo associare ad ogni problema di ottimizzazione combinatoria un grado di difficoltá. Ci renderemo conto che esistono delle notevoli differenze nella difficoltá di risoluzione dei diversi problemi di ottimizzazione combinatoria. Tali differenze vengono formalizzate nella teoria della complessitá. Per cominciare introdurremo come esempi di diversi gradi di difficoltá tre problemi di ottimizzazione combinatoria.

2.1 Esempi di problemi di Ottimizzazione Combinatoria

2.1.1 Albero di supporto a peso minimo

Prima di tutto richiamiamo alcune definizioni.

Definizione 2 *Sia dato un grafo $G = (V, A)$ con $\mid V \mid = n$ dove $\mid V \mid$ denota la cardinalitá (il numero di elementi) dell'insieme V. Si dice che G é un albero se soddisfa le seguenti condizioni (equivalenti tra loro)*

1. *G é privo di cicli e connesso (ovvero esiste un cammino tra ogni coppia di nodi);*
2. *G é privo di cicli e $\mid A \mid = n - 1$;*
3. *G é connesso e $\mid A \mid = n - 1$;*
4. *esiste un unico cammino che congiunge ogni coppia di nodi.*

Definizione 3 *Sia dato un grafo generico $G = (V, A)$. Si definisce albero di supporto di G un sottografo $T = (V, A_T)$ di G (quindi con $A_T \subseteq A$) che é un albero.*

Si noti che un albero di supporto di G deve contenere tutti i nodi di G e che in virtú del punto 2. (o del punto 3.) della Definizione 2, si dovrá avere $\mid A_T \mid = \mid V \mid -1$.

Esempio 1 *Sia dato il grafo* $G = (V, A)$ *con*

$$V = \{a, b, c, d\} \quad A = \{(a, b); (b, c); (b, d); (a, d); (c, d)\}$$

illustrato in Figura 2.1. Un albero di supporto di G *é il sottografo* $T_1 = (V, A_{T_1})$ *con*

$$A_{T_1} = \{(b, c); (b, d); (a, d)\}$$

un altro é il sottografo $T_2 = (V, A_{T_2})$ *con*

$$A_{T_2} = \{(a, b); (b, c); (c, d)\}$$

I due alberi di supporto di G *sono illustrati in Figura 2.1.*

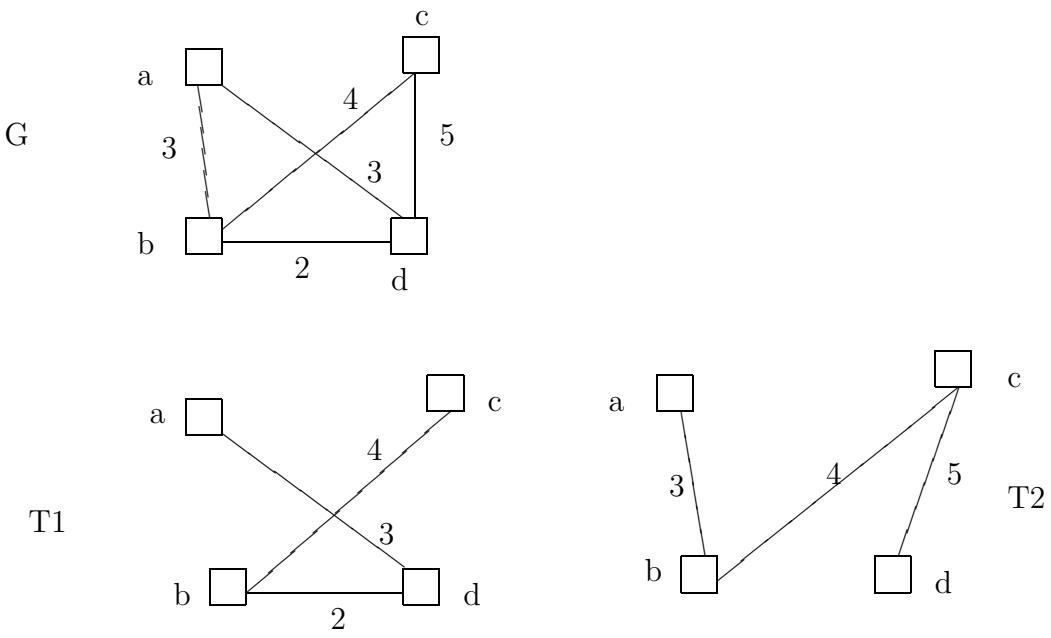


Figura 2.1: Un grafo G e due suoi alberi di supporto T_1 e T_2.

Definiamo ora una funzione

$$w : A \to R$$

che ad ogni arco $e \in A$ del grafo associa un peso $w(e)$ (in Figura 2.1 tale valore é riportato sopra ciascun arco). Nel problema dell'albero di supporto a peso minimo la regione ammissibile S é composta da tutti i possibili alberi di supporto del grafo G, che sono in numero finito. Per ogni albero di supporto

$T = (V, A_T) \in S$ la funzione f é definita come somma dei pesi degli archi dell'albero, cioé

$$f(T) = \sum_{e \in A_T} w(e).$$

Nel grafo di Figura 2.1 si ha $f(T_1) = 9$ e $f(T_2) = 12$. Il problema dell'albero di supporto a peso minimo o Minimum Spanning Tree (abbreviato nel seguito con MST) consiste nel determinare un albero di supporto $T^* = (V, A_{T^*})$ tale che

$$f(T^*) \leq f(T) \quad \forall\ T \in S$$

2.1.2 Il problema dello zaino

Sia dato un insieme $\{1, \ldots, n\}$ di oggetti. Ad ogni oggetto i é associato un peso $p_i > 0$ ed un valore $v_i > 0$. Si ha inoltre a disposizione uno zaino con capacitá (peso massimo trasportabile) pari a $b > 0$. La regione ammissibile S é formata da tutti i sottinsiemi di oggetti il cui peso complessivo non supera la capacitá dello zaino e quindi

$$S = \{N \subseteq \{1, \ldots, n\} : \sum_{i \in N} p_i \leq b\}.$$

Si noti che S é un insieme finito. Infatti

$$\mid S \mid \leq 2^n.$$

La funzione obiettivo é definita dalla somma dei valori degli oggetti nel sottinsieme, ovvero

$$f(N) = \sum_{i \in N} v_i.$$

Il problema dello zaino, indicato nel seguito come problema $KNAPSACK$, é un problema di massimo e consiste nel determinare un sottinsieme $N^* \in S$ tale che

$$f(N^*) \geq f(N) \quad \forall\ N \in S,$$

cioé si ricerca il sottinsieme di oggetti il cui peso complessivo non superi la capacitá dello zaino ed il cui valore complessivo sia massimo.

Esempio 2 *Si consideri l'insieme* $\{1, 2, 3, 4\}$ *di oggetti con i seguenti valori e pesi*

$$\begin{array}{llll} v_1 = 8 & v_2 = 6 & v_3 = 10 & v_4 = 1 \\ p_1 = 7 & p_2 = 7 & p_3 = 13 & p_4 = 4 \end{array}$$

Si supponga inoltre che la capacitá dello zaino sia $b = 16$. *Si ha che* $N_1 = \{1, 4\} \in S$ *con* $f(N_1) = 9$, $N_2 = \{2, 4\} \in S$ *con* $f(N_2) = 7$, *mentre* $N_3 = \{1, 3\} \notin S$.

2.1.3 Problema del commesso viaggiatore

Prima di tutto dobbiamo dare la definizione di *circuito hamiltoniano* in un grafo.

Definizione 4 *Dato un grafo orientato $G = (V, A)$, un circuito hamiltoniano é un sottografo $C = (V, A_C)$ i cui archi formano un cammino orientato all'interno del grafo che partendo da un nodo dello stesso tocca tutti gli altri nodi una ed una sola volta e ritorna infine al nodo di partenza.*

In Figura 2.2 é rappresentato un grafo G e due suoi circuiti hamiltoniani C_1 e C_2. Nel problema del commesso viaggiatore la regione ammissibile S é costituita da tutti i possibili circuiti hamiltoniani del grafo. Se il grafo é completo (cioé esiste un arco orientato tra ogni coppia ordinata di nodi) il numero di tali circuiti é pari a

$$(n-1)! \tag{2.1}$$

dove n é il numero di nodi. Ad ogni arco $(i, j) \in A$ é associato un valore d_{ij} che rappresenta la "distanza" da percorrere lungo tale arco (indicata accanto a ciascun arco in Figura 2.2). La funzione obiettivo f associa ad ogni circuito hamiltoniano $C = (V, A_C)$ la somma delle "distanze" degli n archi attraversati dal circuito, cioé

$$f(C) = \sum_{(i,j)\in A_C} d_{ij}$$

e rappresenta quindi la "distanza" complessiva percorsa lungo il circuito. Il problema del commesso viaggiatore o Travelling Salesman Problem (abbreviato con TSP nel seguito) é un problema di minimo in cui si cerca un circuito hamiltoniano $C^* = (V, A^*_C)$ con "distanza" complessiva minima, cioé

$$f(C^*) \leq f(C) \quad \forall\ C \in S.$$

In Figura 2.2 abbiamo che $f(C_1) = 19$ e $f(C_2) = 14$. Un caso particolare di problema TSP é quello in cui per ogni coppia di archi $(i, j),\ (j, i) \in A$ si ha $d_{ij} = d_{ji}$, cioé la distanza da percorrere da i verso j é uguale a quella da j verso i per ogni coppia di nodi $i, j \in V$. In tal caso si parla di problema TSP *simmetrico*. Tale problema si puó rappresentare con un grafo non orientato con un solo arco non orientato tra ogni coppia di nodi e il numero totale di circuiti hamiltoniani é dimezzato rispetto al numero (2.1) del caso asimmettrico.

2.2 Difficoltá dei problemi di ottimizzazione combinatoria

Teoricamente tutte le istanze di ognuno dei tre problemi descritti in precedenza é risolvibile in modo esatto. Infatti le regioni ammissibili sono tutte costituite da un numero finito di elementi e la seguente procedura, detta di *enumerazione completa*, risolve il problema: valutare la funzione f per ogni elemento in S e restituire l'elemento con valore di f minimo (o massimo se il problema é di

massimo). Tuttavia una semplice osservazione metterá in luce i limiti di questo approccio. Consideriamo il problema TSP su un grafo completo con numero di nodi $n = 22$. In base alla formula (2.1) sappiamo che il numero di circuiti hamiltoniani é pari a $(n-1)! = 21! > 10^{19}$. Supponiamo (ottimisticamente) che la valutazione della funzione obiettivo per un singolo circuito hamiltoniano richieda un nanosecondo (10^{-9} secondi). Ne consegue che la valutazione di tutti i circuiti hamiltoniani richiede piú di 10^{10} secondi che corrispondono ad un tempo superiore ai 200 anni. Ció dimostra che sebbene sia sempre possibile teoricamente risolvere tali problemi, in pratica dobbiamo fare i conti con dei limiti temporali e quindi, da un punto di vista pratico, si parlerá di problema risolvibile con una data procedura solo nel caso in cui la procedura restituisca una soluzione in tempi ragionevoli. Ció rende la procedura di enumerazione completa accettabile solo per istanze di problemi di dimensioni limitate (quindi per grafi con pochi nodi per i problemi MST e TSP e per istanze con pochi oggetti per il problema $KNAPSACK$). La domanda successiva é quindi la seguente: esistono altre procedure che consentono di risolvere in tempi ragionevoli istanze dei tre problemi visti con dimensioni molto piú elevate? La risposta é: dipende dal problema. É in generale vero che esistono procedure molto piú efficienti dell'enumerazione completa ma mentre per uno dei problemi (il problema MST) si possono risolvere in tempi ragionevoli istanze di grandi dimensioni, per il problema $KNAPSACK$ e, ancor piú, per il problema TSP puó essere difficile risolvere anche istanze di dimensioni non troppo elevate. Tutto ció ha una formulazione ben precisa nella teoria della complessitá, della quale si daranno brevi cenni di seguito.

2.2.1 Complessitá degli algoritmi di risoluzione

Ad ogni istanza I di un problema di ottimizzazione combinatoria é associata una dimensione $dim(I)$ che corrisponde alla quantitá di memoria necessaria per memorizzare (in codifica binaria) tale istanza. Per esempio, l'istanza del problema $KNAPSACK$ nell'Esempio 2 ha una dimensione pari alla quantitá di memoria necessaria per memorizzare in codice binario i pesi ed i valori degli oggetti e la capacitá dello zaino. Consideriamo ora una procedura di risoluzione o algoritmo $\mathcal{A}$ per il problema (si pensi, ad esempio, alla procedura piú semplice, quella di enumerazione completa). La risoluzione dell'istanza I con l'algoritmo $\mathcal{A}$ richiederá un certo numero di operazioni (e quindi un certo tempo) indicato con $numop_{\mathcal{A}}(I)$. Fissata una dimensione k vi saranno diverse istanze di dimensione k. L'analisi *worst case* definisce il tempo $t_{\mathcal{A}}(k)$ necessario all'algoritmo $\mathcal{A}$ per risolvere istanze di dimensione k come il massimo tra tutti i tempi di esecuzione di istanze di dimensione k, cioé

$$t_{\mathcal{A}}(k) = \max_{I:\ dim(I)=k} numop_{\mathcal{A}}(I).$$

Tipicamente non si conosce il valore esatto della funzione $t_{\mathcal{A}}(k)$ ma se ne conosce l'ordine di grandezza. Si dice che la funzione $t_{\mathcal{A}}(k) = O(g(k))$, ovvero che $t_{\mathcal{A}}(k)$ é dell'ordine di grandezza della funzione $g(k)$, se esiste una costante $u > 0$ tale

$g(k)$	$k=10$	$k=20$	$k=30$	$k=40$	$k=50$
$\log_2(k)$	3.32	4.32	4.90	5.32	5.64
k	10	20	30	40	50
k^2	100	400	900	1600	2500
k^3	1000	8000	27000	64000	125000
2^k	1024	$> 10^6$	$> 10^9$	$> 10^{12}$	$> 10^{15}$
$k!$	3628800	$> 10^{18}$	$> 10^{32}$	$> 10^{47}$	$> 10^{64}$

Tabella 2.1: Crescita di alcune funzioni g al crescere di k

che

$$t_{\mathcal{A}}(k) \leq ug(k).$$

Costruiamo ora la Tabella 2.2.1 con diverse possibili funzioni g e diversi valori di k. Come si vede dalla tabella, mentre fino a $g(k) = k^3$ la crescita di g al crescere di k é ragionevole, per $g(k) = 2^k$ e ancor piú per $g(k) = k!$, la crescita é rapidissima. Un algoritmo per il quale $t_{\mathcal{A}}(k)$ fosse dell'ordine di grandezza di 2^k oppure $k!$ si dice che ha *complessitá esponenziale*. É evidente che un tale algoritmo consente di risolvere in tempi ragionevoli solo istanze di dimensioni limitate. Per questa ragione si cercano per i problemi algoritmi di risoluzione con *complessitá polinomiale*, in cui cioé la funzione $t_{\mathcal{A}}$ é dell'ordine di grandezza di un polinomio k^p per un qualche esponente p. Ovviamente, tanto maggiore é l'esponente p del polinomio, quanto piú rapidamente cresce il numero di operazioni dell'algoritmo al crescere di k e quindi quanto piú piccole sono le dimensioni dei problemi che l'algoritmo é in grado di risolvere in tempi ragionevoli (giá per $p = 4$ la crescita é piuttosto rapida). Tuttavia la crescita polinomiale é sempre preferibile ad una esponenziale. A questo punto ci possiamo chiedere se, dato un problema di ottimizzazione combinatoria, possiamo sempre trovare un algoritmo con complessitá polinomiale che lo risolva. La risposta é: forse, ma é molto improbabile. Vedremo ora di chiarire meglio nel seguito questa risposta.

2.2.2 La classe P

Vogliamo ora definire la classe P di problemi di ottimizzazione.

Definizione 5 *Un problema appartiene alla classe P se esiste almeno un algoritmo di complessitá polinomiale che lo risolve.*

I problemi di PL appartengono alla classe P. Un po' a sorpresa l'algoritmo del simplesso non ha complessitá polinomiale. Esistono peró altri algoritmi che risolvono i problemi di PL e che hanno complessitá polinomiale. Va detto comunque che ció non toglie validitá all'algoritmo del simplesso. Le istanze per le quali esso richiede un tempo di esecuzione esponenziale sono molto particolari e difficilmente riscontrabili nella pratica, mentre su istanze tipiche il comportamento dell'algoritmo del simplesso é piuttosto buono.
Non é invece noto, ma é ritenuto improbabile, se i problemi di PLI appartengano

alla classe P.
Prendiamo ora in esame il primo problema di ottimizzazione combinatoria che abbiamo introdotto, il problema MST. Ci chiediamo se tale problema appartiene alla classe P. Per dare risposta affermativa a tale domanda é sufficiente presentare almeno un algoritmo di complessitá polinomiale che lo risolva. Prendiamo in esame il seguente algoritmo detto Algoritmo Greedy (nel capitolo 5 il concetto di scelta greedy verrá illustrato piú in dettaglio).

Algoritmo greedy per il problema MST

Passo 1 Dato il grafo $G = (V, A)$, si ordinino tutti gli $m =\mid A \mid$ archi del grafo in ordine non decrescente rispetto al peso, cioé

$$w(e_1) \leq w(e_2) \leq \cdots \leq w(e_{m-1}) \leq w(e_m).$$

Si parta con un insieme A_T di archi vuoto, cioé $A_T = \emptyset$. Sia $k = 1$

Passo 2 Se $\mid A_T \mid=\mid V \mid -1$ ci si arresta e si restituisce l'albero $T = (V, A_T)$ come soluzione. Altrimenti si vada al Passo 3.

Passo 3 Se e_k non forma cicli con gli archi in A_T, lo si aggiunga ad A_T, cioé si ponga $A_T = A_T \cup \{e_k\}$. Altrimenti si lasci A_T invariato.

Passo 4 Si ponga $k = k + 1$ e si ritorni al Passo 2.

Si puó dimostrare che tale algoritmo risolve in modo esatto il problema MST. Se ora ne analizziamo la complessitá, notiamo che l'operazione piú costosa, almeno nel caso di grafi densi, ovvero con un numero elevato di archi, é quella di ordinamento degli archi secondo il costo non decrescente. Questo richiede un numero di operazioni pari a $O(\mid A \mid \log(\mid A \mid))$. Il ciclo che segue é di lunghezza non superiore a $\mid A \mid$ e, utilizzando opportune strutture dati, il numero di operazioni da esso richieste non supera $O(\mid A \mid \log(\mid A \mid))$. Quindi la complessitá totale dell'algoritmo é anch'essa $O(\mid A \mid \log(\mid A \mid))$ ed é polinomiale. Ció é giá sufficiente per il nostro scopo principale, cioé quello di inserire MST nella classe P. Tuttavia é opportuno fare un'ulteriore osservazione. Nel risolvere un problema vorremmo sempre eseguire il minor numero possibile di operazioni. Quindi, dato un problema e un algoritmo che lo risolve, ci si puó sempre chiedere se esista un algoritmo di complessitá migliore che risolva il problema. Nel caso del MST tale algoritmo esiste e ha complessitá pari a $O(\mid V \mid^2)$ che, almeno per grafi densi, é migliore di $O(\mid A \mid \log(\mid A \mid))$. Si puó anche dimostrare che, in un senso che non specificheremo, questo algoritmo ha complessitá ottima, ovvero non esiste un algoritmo che risolva il problema MST con complessitá migliore di questa.

2.2.3 La classe NP e i problemi $NP-completi$

Introduciamo ora una nuova classe di problemi, la classe NP. Diamo una versione molto semplificata del criterio di appartenenza di un problema alla classe NP.

Definizione 6 *Una classe di problemi con insieme di istanze $\mathcal{I}$ si dice che appartiene alla classe NP se, per ogni istanza I del problema, essendo nota la soluzione ottima x_I^* dell'istanza, il calcolo del valore ottimo $f_I(x_I^*)$ dell'istanza puó essere effettuato in un tempo polinomiale rispetto alla dimensione dell'istanza.*

Possiamo subito notare che $P \subseteq NP$, cioé tutti i problemi nella classe P fanno anche parte della classe NP. Infatti, per tutti i problemi nella classe P il calcolo del valore ottimo di ogni istanza del problema puó essere effettuato in tempo polinomiale persino *a prescindere* dalla conoscenza della soluzione ottima x_I^*. Abbiamo quindi, in particolare, che

$$MST \in P \quad \Rightarrow \quad MST \in NP.$$

Ma cosa possiamo dire al riguardo del problema $KNAPSACK$ e del problema TSP? Non sappiamo se tali problemi appartengono alla classe P. Ció significa che al momento non sono stati individuati algoritmi di complessitá polinomiale in grado di risolverli e come vedremo é altamente improbabile che esistano. Tuttavia entrambi appartengono alla classe NP come dimostrano le due seguenti proposizioni.

Proposizione 1 $KNAPSACK \in NP$.

Dimostrazione
Data una soluzione ottima N_I^* di un'istanza I del problema di $KNAPSACK$, si ha

$$f_I(N_I^*) = \sum_{i \in N_I^*} v_i$$

e quindi il calcolo richiede $\mid N_I^* \mid \leq n$ somme e quindi un numero polinomiale di operazioni rispetto alla dimensione dell'istanza.

Proposizione 2 $TSP \in NP$.

Dimostrazione
Data una soluzione ottima $C_I^* = (V_I, A_{C_I^*})$, dove $\mid A_{C_I^*} \mid = \mid V_I \mid$, di un'istanza I, relativa al grafo $G_I = (V_I, A_I)$, del problema di TSP, si ha

$$f_I(C_I^*) = \sum_{e \in A_{C_I^*}} w_e$$

e quindi il calcolo richiede $\mid V_I \mid$ somme e quindi un numero polinomiale di operazioni rispetto alla dimensione dell'istanza.

Stabilito che $P \subseteq NP$, ci si puó chiedere se tutti i problemi in NP sono risolvibili in tempo polinomiale, cioé $P = NP$, oppure se esistono problemi in NP che non sono risolvibili in tempo polinomilae, cioé $P \neq NP$. In realtá con le conoscenze attuali non si puó ancora rispondere a tale domanda. Tuttavia tra le due possibili risposte quella che si ritiene la piú probabile é che $P \neq NP$. All'interno della classe NP una grande importanza hanno i problemi $NP - completi$ definiti nel modo seguente.

Definizione 7 *Un problema si dice $NP - completo$ se soddisfa le seguenti due proprietá:*

- *appartiene alla classe NP;*
- *se esistesse un algoritmo di complessitá polinomiale che lo risolve, allora $P = NP$.*

In base alla definizione e al fatto che non sia chiaro se $P = NP$ o $P \neq NP$, possiamo dire che non sono al momento noti algoritmi di complessitá polinomiale in grado di risolvere problemi $NP - completi$. Inoltre, se, come si ritiene, $P \neq NP$, allora non esisterebbero algoritmi di complessitá polinomiale per tali problemi. Ne risulta quindi che la classe dei problemi $NP - completi$ é una classe di problemi "difficili" nel senso che, a meno che non sia $P = NP$, non possiamo risolverli in tempo polinomiale. Esiste un gran numero di problemi $NP - completi$. Tra questi anche i problemi $KNAPSACK$ e TSP come sancito dalla seguente proposizione (non dimostrata).

Proposizione 3 $KNAPSACK, TSP \in NP - completi$.

Possiamo quindi dire che dei tre problemi di esempio che abbiamo introdotto, uno, MST, puó essere catalogato tra quelli "facili" (ovvero risovibili in tempo polinomiale), gli altri due, $KNAPSACK$ e TSP, sono catalogabili tra quelli difficili (ovvero quelli appartenenti alla classe dei problemi $NP - completi$). Ma la catalogazione puó procedere ulteriormente. É possibile introdurre ulteriori distinzioni tra i problemi "difficili". A questo scopo introdurremo ora i concetti di problemi e algoritmi di approssimazione.

2.2.4 Problemi e algoritmi di approssimazione

Sia data una classe di problemi di ottimizzazione con insieme $\mathcal{I}$ di istanze e tale che

$$\forall\ I = (f_I, S_I) \in \mathcal{I}\ \ \forall\ x \in S_I\ \ :\ \ f_I(x) \geq 0.$$

Per ogni istanza $I = (f_I, S_I) \in \mathcal{I}$ sia x_I^* la soluzione ottima dell'istanza e sia

$$opt(I) = f_I(x_I^*)$$

il valore ottimo dell'istanza. Il problema di ε-approssimazione associato a questo problema di ottimizzazione é definito come segue.

Definizione 8 *Per i problemi di massimo il problema di ε-approssimazione, $\varepsilon \geq 0$, consiste nel determinare un punto $\overline{x}_I \in S_I$ tale che*

$$\frac{opt(I)}{f_I(\overline{x}_I)} \leq 1 + \varepsilon. \tag{2.2}$$

Per i problemi di minimo il problema di ε-approssimazione consiste nel determinare un punto $\overline{x}_I \in S_I$ tale che

$$\frac{f_I(\overline{x}_I)}{opt(I)} \leq 1 + \varepsilon. \tag{2.3}$$

In entrambi i casi il punto $\overline{x}_I$ viene definito soluzione ε-approssimata del problema.

Si noti che per $\varepsilon = 0$ il problema coincide con quello di ottimizzazione, ma per $\varepsilon > 0$ si richiede qualcosa di meno rispetto al problema di ottimizzazione: non si cerca la soluzione ottima ma una soluzione che non si discosti troppo da quella ottima. In particolare, da (2.2) e (2.3) si ricava che in una soluzione ε-approssimata il valore f_I in corrispondenza di tale soluzione differisce, sia per i problemi di massimo che per quelli di minimo, per al piú $\varepsilon opt(I)$ dal valore ottimo $opt(I)$ dell'istanza. Chiaramente, tanto maggiore é il valore di ε, quanto minore é la precisione garantita da una soluzione ε-approssimata.
Un algoritmo A si definisce *algoritmo di ε-approssimazione* per una classe di problemi di ottimizzazione, se risolve il problema di ε-approssimazione associato ad ogni istanza $I \in \mathcal{I}$ del problema di ottimizzazione. Cioé se indichiamo con $\overline{x}_I \in S_I$ la soluzione restituita dall'algoritmo A e con $A(I) = f_I(\overline{x}_I)$ il corrispondente valore della funzione obiettivo f_I, si ha che per ogni istanza $I \in \mathcal{I}$

$$\frac{opt(I)}{A(I)} \leq 1 + \varepsilon,$$

se il problema é di massimo, oppure

$$\frac{A(I)}{opt(I)} \leq 1 + \varepsilon,$$

se il problema é di minimo. A questo punto ci si puó chiedere, dato un problema $NP - completo$, qual é la complessitá dei corrispondenti problemi di ε-approssimazione per diversi possibili valori di ε. Possiamo riconoscere quattro diversi possibili casi che elenchiamo ora in ordine crescente di difficoltá.

Caso 1 Per ogni $\varepsilon > 0$ esiste un algoritmo di ε-approssimazione che richiede tempi polinomiali sia rispetto alla dimensione delle istanze, sia rispetto all'inverso $\frac{1}{\varepsilon}$ della precisione richiesta. In tal caso si dice che il problema ammette uno *schema di approssimazione completamente polinomiale*.

Caso 2 Per ogni $\varepsilon > 0$ esiste un algoritmo di ε-approssimazione che richiede tempo polinomiale rispetto alla dimensione delle istanze ma esponenziale rispetto all'inverso $\frac{1}{\varepsilon}$ della precisione richiesta. In tal caso si dice che il problema ammette uno *schema di approssimazione polinomiale*.

Caso 3 Per valori piccoli di ε, anche il problema di ε-approssimazione é $NP-completo$, mentre per valori di ε piú elevati é risolvibile in tempo polinomiale.

Caso 4 Per ogni valore di ε il problema di ε-approssimazione é $NP-completo$.

Il problema $KNAPSACK$ rientra nel Caso 1, cioé anche se la determinazione di una soluzione ottima del problema $KNAPSACK$ é un problema $NP-completo$, é sempre possibile determinare una soluzione ε-approssimata per ogni $\varepsilon > 0$ tramite un algoritmo polinomiale rispetto alla dimensione dell'istanza e rispetto a $\frac{1}{\varepsilon}$. Non vedremo tale algoritmo di approssimazione, denominato algoritmo *scaling-rounding*, ma ne riportiamo la complessitá $O\left(\frac{n}{\varepsilon^2}\right)$.
All'estremo opposto, cioé nel Caso 4, troviamo il problema TSP (anche nel sottocaso simmetrico): per esso non é possibile risolvere in tempi polinomiali, a meno che non sia $P = NP$, neppure il problema di ε-approssimazione per *ogni* valore di ε. Possiamo quindi dire che, pur appartenendo entrambi i problemi di $KNAPSACK$ e di TSP alla classe dei problemi "difficili" o, piú precisamente, $NP-completi$, il problema $KNAPSACK$ e quello TSP hanno livelli di difficoltá agli estremi opposti all'interno della classe dei problemi $NP-completi$.
Nel capitolo 5 introdurremo un caso particolare di problema TSP, chiamato problema TSP *metrico*, che, come vedremo, rientra nel Caso 3.

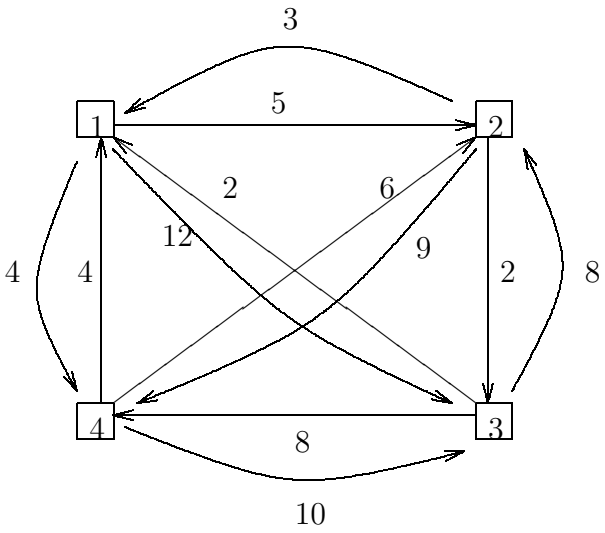
2.2.5 Tecniche euristiche

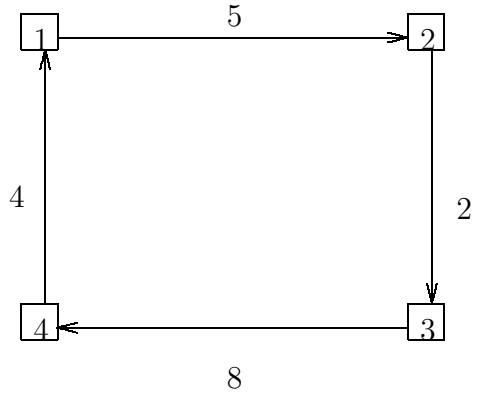
Quando affrontiamo un problema vorremmo avere risposte in "tempi ragionevoli". Questo é tanto piú difficile, quanto piú difficile é il problema che cerchiamo di risolvere. Quindi possiamo aspettarci che per problemi nella classe P si possano risolvere in modo esatto in "tempi ragionevoli" anche istanze molto grandi, mentre per problemi $NP-completi$ questo é piú complicato, in taluni casi persino se ci si accontenta di una soluzione approssimata. Tuttavia questa non é una regola sempre valida. Il tutto, infatti, é strettamente legato a cosa si intende per "tempi ragionevoli". In alcune applicazioni si devono risolvere problemi appartenenti alla classe P ma si desiderano risposte in tempo reale (frazioni di secondo), che neppure un algoritmo di complessitá polinomiale é in grado di fornire. In altre applicazioni si devono risolvere problemi difficili ma i tempi richiesti per la risposta sono molto larghi (giorni o persino mesi) ed in tal caso si puó anche sperare di risolvere tali problemi in modo esatto nei limiti di tempo richiesti.
In ogni caso ci si pone la seguente domanda: cosa possiamo fare se i "tempi ragionevoli" entro cui si desidera una risposta sono troppo brevi per poter sperare di ottenere una soluzione esatta o approssimata del problema che stiamo affrontando? In questo caso si possono utilizzare delle *tecniche euristiche*.
Un'euristica si puó definire come un compromesso tra tempi di esecuzione e qualitá della soluzione trovata e deve quindi soddisfare i seguenti due requisiti in conflitto tra loro:

1. essere eseguibile in tempi ragionevoli (in particolare deve avere complessitá polinomiale);

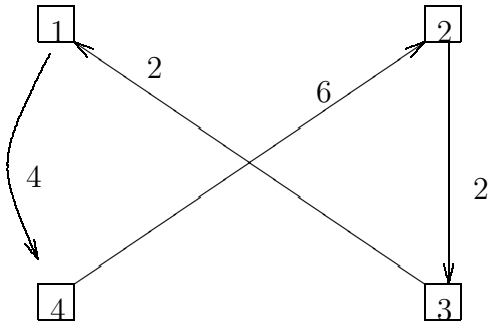
2. deve restituire su molte istanze del problema una soluzione ottima o vicina a quella ottima.

Si noti che il secondo requisito non richiede che su *tutte* le istanze del problema l'euristica restituisca buone soluzioni. Su alcune istanze puó anche resituire soluzioni non particolarmente buone ma questo é l'inevitabile prezzo da pagare per soddisfare il primo requisito e cioé quello di ottenere una risposta in tempi brevi.
Lo sviluppo di tali euristiche é tipicamente strettamente legato al particolare problema che si deve risolvere ma si fonda spesso su alcuni concetti di base di cui vedremo alcuni esempi (quali tecniche greedy e mosse locali) nel capitolo 5.





C1



C2

Figura 2.2: Un grafo G e i suoi due circuiti hamiltoniani C_1 e C_2.

Capitolo 3

Problemi nella classe P

In questo capitolo ci occuperemo di alcuni problemi di ottimizzazione combinatoria che si incontrano spesso nelle applicazioni e per i quali si é dimostrata l'appartenenza alla classe P. I problemi che tratteremo sono quelli di flusso a costo minimo, flusso massimo, matching, trasporto e assegnamento. Questi si vanno ad aggiungere ai generici problemi di PL e al problema MST di cui si é giá detto che appartengono alla classe P. Prima peró di entrare nel dettaglio di ciascuno di questi problemi, introduciamo il concetto di chiusura convessa di un insieme e le matrici *totalmente unimodulari* che ci permetteranno in molti casi di catalogare immediatamente un problema nella classe P.

3.1 Problemi di PLI e chiusure convesse

Consideriamo il seguente problema di PL

$$\begin{aligned} &\max cx \\ &Ax \leq b \\ &x \geq 0 \end{aligned}$$

con regione ammissibile

$$S_a = \{x \in R^n : \ Ax \leq b, \ x \geq 0\}. \tag{3.1}$$

Se si aggiungono i vincoli di interezza sulle variabili la regione ammissibile diventa

$$Z_a = S_a \cap Z^n$$

dove Z rappresenta l'insieme degli interi. Il problema di PLI corrispondente é il seguente

$$\max_{x \in Z_a} cx \tag{3.2}$$

Diamo ora la definizione di insieme convesso e di *chiusura convessa* di un insieme.

Definizione 9 *Un insieme S si dice* convesso *se:*

$$\forall\ x_1, x_2 \in S\ \ \forall\ \lambda \in [0,1]\ :\quad \lambda x_1 + (1-\lambda)x_2 \in S.$$

Definizione 10 *Dato un insieme T, la chiusura convessa di T, indicata con $conv(T)$, é il piú piccolo insieme convesso contenente T.*

In Figura 3.1 é riportato un insieme T e la sua chiusura convessa. Si consideri

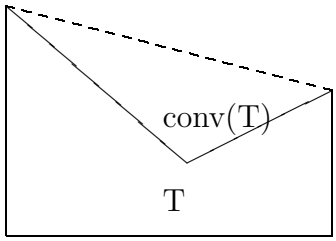


Figura 3.1: Un insieme T e la sua chiusura convessa.

ora la chiusura convessa $conv(Z_a)$ di Z_a. Si puó dimostrare che, se $Z_a \neq \emptyset$, $conv(Z_a)$ é un poliedro convesso o un troncone, cioé esiste una matrice A' ed un vettore b' tale che

$$conv(Z_a) = \{x \in R^n :\ A'x \leq b',\ x \geq 0\}. \tag{3.3}$$

Si consideri ora il seguente problema di PL

$$\max_{x \in conv(Z_a)} cx \tag{3.4}$$

Si puó dimostrare che:

- il problema (3.2) ammette soluzione se e solo se la ammette il problema (3.4);
- ogni soluzione ottima del problema (3.2) é soluzione ottima del problema (3.4);
- esiste almeno una soluzione ottima (di base) del problema (3.4) che é anche soluzione ottima del problema (3.2).

Esempio 3 *Si consideri il seguente problema di PLI,*

$$\begin{aligned}
&\max x_1 + x_2\\
&x_1 + \frac{1}{2}x_2 \leq \frac{7}{4}\\
&\frac{1}{2}x_1 + x_2 \leq \frac{7}{4}\\
&x_1, x_2 \geq 0 \ \textit{interi}
\end{aligned}$$

Si ha $S_a = \{(x_1, x_2) : \ Ax \leq b, \ x_1, x_2 \geq 0\}$, *con*

$$A = \begin{bmatrix} 1 & \frac{1}{2} \\ \frac{1}{2} & 1 \end{bmatrix} \qquad b = \begin{bmatrix} \frac{7}{4} \\ \frac{7}{4} \end{bmatrix}.$$

L'insieme $Z_a = S_a \cap Z^2$ *é formato dai quattro punti*

$$(0,0) \quad (0,1) \quad (1,0) \quad (1,1)$$

Si ha che

$$conv(Z_a) = \{(x_1, x_2) : \ x_1 \leq 1, \ x_2 \leq 1, \ x_1, x_2 \geq 0\}.$$

(vedi Figura 3.2). In tal caso

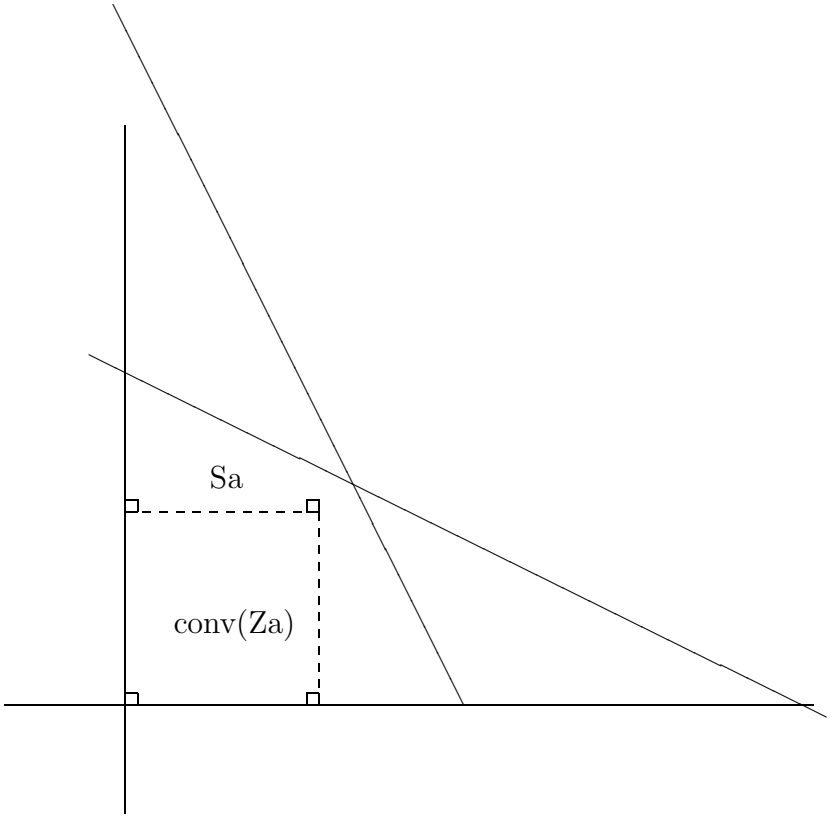


Figura 3.2: La chiusura convessa di Z_a per l'esempio considerato.

$$A' = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} \qquad b' = \begin{bmatrix} 1 \\ 1 \end{bmatrix}.$$

Si noti che indipendentemente dalla funzione obiettivo i problemi (3.2) e (3.4) hanno sempre soluzione in questo caso. Con la funzione obiettivo $x_1 + x_2$ *entrambi i problemi hanno l'unica soluzione* $(1, 1)$*. Se si considera invece l'obiettivo* x_1*, si ha che il problema (3.2) ha soluzioni* $(1, 0)$ *e* $(1, 1)$*, mentre il problema (3.4) ha come soluzioni l'intero segmento avente come estremi* $(1, 0)$ *e* $(1, 1)$*. É comunque sempre vero che esiste almeno una soluzione del problema (3.4) che é anche soluzione del problema (3.2).*
Se si risolve il problema con l'algoritmo di Gomory, si puó vedere che il primo taglio é dato dalla seguente disequazione $3x_1 + 4x_2 \leq 7$*. Aggiungendo tale taglio si vede, in Figura 3.3, come si ottenga un'approssimazione piú precisa rispetto a* S_a *di* $conv(Z_a)$*.*

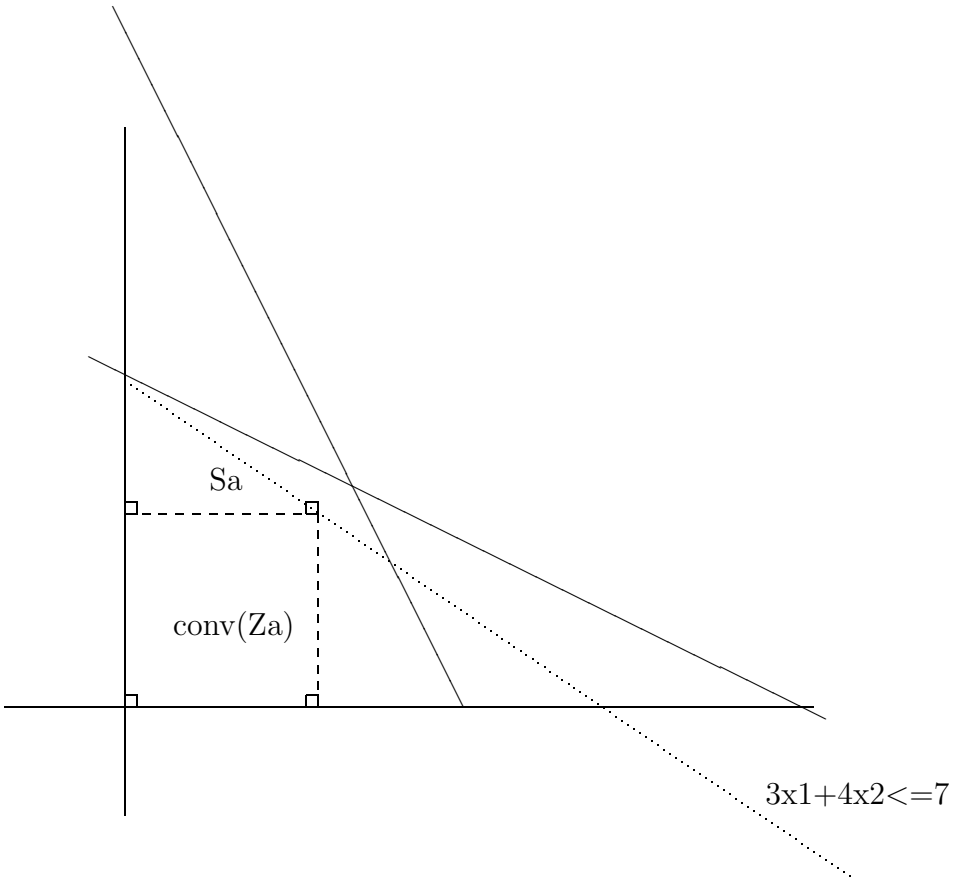


Figura 3.3: L'aggiunta di un taglio fornisce un'approssimazione della chiusura convessa di Z_a migliore di S_a.

Se conoscessimo la matrice $\mathbf{A}'$ ed il vettore $\mathbf{b}'$ che definiscono $conv(Z_a)$, potremmo risolvere il problema (PLI) risolvendo il problema (3.4). Il problema

è che in molti casi $conv(Z_a)$ non è noto. In altri casi $conv(Z_a)$ è noto ma è formato da un numero esponenziale di vincoli, il che *può* rendere inefficiente la risoluzione del problema (3.4). Se l'insieme di vincoli $\mathbf{A'x} \leq \mathbf{b'}$ che definisce (3.4) contiene un numero esponenziale di vincoli, è in alcuni casi comunque possibile risolvere in tempo polinomiale il problema (3.4). Più precisamente vale la seguente osservazione.

Osservazione 1 *Se il problema (3.4) contiene un numero esponenziale di vincoli, è possibile risolverlo in tempo polinomiale* se e solo se *è possibile risolvere in tempo polinomiale il seguente problema (detto* problema di separazione*):*

dato $\mathbf{x} \in R^n$, è vero che $\mathbf{A'x} \leq \mathbf{b'}$?

Si noti che dal momento che il numero di vincoli è esponenziale è ovviamente impossibile risolvere questo problema in tempo polinomiale verificando ad uno ad uno che tutti i vincoli siano soddisfatti.
Citiamo infine il fatto che gli algoritmi di taglio visti nel corso precedente rappresentano un tentativo di approssimare in modo sempre piú preciso $conv(Z_a)$ attraverso l'aggiunta di tagli successivi.

Dopo aver chiarito cosa sia la chiusura convessa di un insieme e l'importanza di tale chiusura per i problemi di PLI, ci concentreremo ora su una speciale classe di problemi di PLI, quelli per cui la chiusura convessa di $Z_a = S_a \cap Z^n$ coincide con S_a, ovvero

$$conv(Z_a) = conv(S_a \cap Z^n) = S_a. \tag{3.5}$$

Si noti che nell'esempio precedente ció non era vero. I problemi di PLI per cui si verifica questa condizione sono importanti perché sono molto piú semplici da risolvere rispetto agli altri problemi di PLI. Infatti, essendo $conv(Z_a) = S_a$ e viste le strette relazioni tra le soluzioni dei problemi (3.2) e (3.4), possiamo risolvere tali problemi semplicemente eliminando i vincoli di interezza sulle variabili, ovvero risolvendo il problema di PL

$$\max_{x \in S_a} cx.$$

Si noti che essendo tutti problemi di PL nella classe P, anche tutti questi problemi risulteranno appartenere alla classe P. Inoltre, in molti casi questi problemi hanno strutture particolari che consentono di risolverli attraverso algoritmi specifici che sono per lo piú specializzazioni di metodi per risolvere problemi di PL (come il metodo del simplesso), dove la particolare struttura dei problemi consente di implemementare in modo piú efficiente i passi di tali metodi.
Nel seguito mostreremo un importante classe di casi in cui la condizione (3.5) é soddisfatta.

3.2 Matrici totalmente unimodulari

Prima di approfondire ulteriormente il discorso sui problemi per cui $conv(Z_a) = S_a$, introduciamo il concetto di matrice *totalmente unimodulare*.

Definizione 11 *Una matrice A si dice totalmente unimodulare (TU nel seguito) se ogni sua sottomatrice quadrata ha determinante pari a 0, +1 o -1.*

Si noti che una matrice TU puó avere come elementi solo i valori 0, +1 e -1 visto che ogni suo elemento é una particolare sottomatrice quadrata di ordine 1×1. Si citano di seguito alcune proprietá delle matrici TU.

Proprietá 1 *Se A é una matrice TU si ha che*

1. A^T *é TU*
2. $[A\ I]$, *dove I é la matrice identica, é TU*
3. *una matrice ottenuta duplicando righe e/o colonne di A é ancora TU*
4. *una matrice ottenuta moltiplicando righe e/o colonne di A per -1 é ancora TU*
5. *una matrice ottenuta scambiando righe di A (oppure colonne di A) tra loro é ancora TU*
6. *una matrice ottenuta da A mediante un'operazione di cardine é ancora TU*

Ci chiediamo ora come sia possibile riconoscere una matrice TU senza dover calcolare i determinanti di tutte le sottomatrici quadrate. Esistono alcune regole, tra cui la seguente.

Osservazione 2 *Sia A una matrice i cui elementi sono tutti uguali a 0, +1 o -1 e lungo ogni colonna non vi sono piú di due elementi diversi da 0. Allora A é TU se e solo se l'insieme delle righe di A puó essere suddiviso in due sottinsiemi Q_1 e Q_2 tali che se una colonna contiene due elementi diversi da 0 si ha che:*

- *se i due elementi hanno lo stesso segno allora una delle due righe in cui si trovano é in Q_1 e l'altra in Q_2;*
- *se hanno segno opposto le righe corrispondenti sono entrambe contenute in Q_1 od entrambe in Q_2.*

Esempio 4 *Sia A la seguente matrice*

$$A = \begin{bmatrix} 1 & 1 & 0 & 0 \\ 0 & 0 & 1 & 1 \\ 1 & 0 & 1 & 0 \\ 0 & 1 & 0 & 1 \end{bmatrix}$$

Prendendo $Q_1 = \{1, 2\}$ e $Q_2 = \{3, 4\}$ si verifica immediatamente che la condizione é soddisfatta e quindi A é TU.

Vale il seguente corollario.

Corollario 1 *Sia A una matrice i cui elementi sono tutti uguali a 0, +1 o -1 e lungo ogni colonna non vi sono piú di due elementi diversi da 0. Se nelle colonne con due elementi diversi da 0 la somma di tali elementi é uguale a 0 (ovvero un elemento é uguale a +1 e l'altro a -1), allora A é TU.*

Dimostrazione É sufficiente utilizzare l'Osservazione 2 ponendo

$$Q_1 = \{\text{tutte le righe di } A\} \quad Q_2 = \emptyset.$$

Il corollario ci dice ad esempio che tutte le matrici di incidenza nodo-arco di un grafo orientato sono matrici TU.

Esempio 5 *Sia A la seguente matrice*

$$A = \begin{bmatrix} 1 & 1 & 0 & 0 \\ 0 & -1 & 1 & 1 \\ -1 & 0 & 0 & 0 \\ 0 & 0 & 0 & -1 \end{bmatrix}$$

In base al corollario tale matrice é TU.

Un altro corollario é il seguente:

Corollario 2 *La matrice di incidenza nodo-arco per un grafo bipartito non orientato $G = (V_1 \cup V_2, A)$, dove V_1 e V_2 sono le due classi di bipartizione, é TU.*

Dimostrazione Si ricordi che per grafi non orientati la matrice di incidenza nodo-arco é una matrice con entrate pari a 0 o 1, con tante righe quanti sono i nodi del grafo e tante colonne quanti sono gli archi del grafo. Lungo la colonna relativa ad un arco $(i, j) \in A$ la matrice ha un $+1$ all'altezza delle righe i e j e 0 in tutte le altre righe. Per dimostrare il corollario é sufficiente utilizzare l'Osservazione 2 ponendo

$$Q_1 = V_1 \quad Q_2 = V_2.$$

Ma perché sono importanti le matrici TU? La loro importanza é legata a questo teorema (non dimostrato).

Teorema 1 *Sia*

$$S_a(b_1, b_2, b_3) = \{x \in R^n : \ A_1 x \leq b_1, \ A_2 x \geq b_2, \ A_3 x = b_3 \ x \geq 0\}$$

e

$$Z_a(b_1, b_2, b_3) = S_a(b_1, b_2, b_3) \cap Z^n,$$

con b_1, b_2, b_3 vettori di interi. Si dimostra che A_1, A_2, A_3 sono TU se e solo se per tutti i vettori di interi b_1, b_2, b_3 per cui $S_a(b_1, b_2, b_3) \neq \emptyset$ si ha che

$$conv(Z_a(b_1, b_2, b_3)) = S_a(b_1, b_2, b_3).$$

In altre parole questo ci dice che la soluzione di un problema di PLI con matrici dei vincoli TU e vettori dei termini noti intero puó essere ottenuta semplicemente eliminando i vincoli di interezza.

Esempio 6 *Si consideri il seguente problema*

$$\begin{aligned}
&\max\ x_1 + x_2 + x_3 + x_4 \\
&x_1 + x_2 = 2 \\
&-x_1 + x_3 = 4 \\
&-x_2 + x_4 = 3 \\
&-x_3 - x_4 = 2 \\
&x_1, x_2, x_3, x_4 \geq 0 \quad \textit{interi}
\end{aligned}$$

Il problema di PL ottenuto eliminando i vincoli di interezza ha regione ammissibile

$$S_a(b_3) = \{x \in R^n :\ A_3 x = b_3,\ x \geq 0\}$$

con

$$A_3 = \begin{bmatrix} 1 & 1 & 0 & 0 \\ -1 & 0 & 1 & 0 \\ 0 & -1 & 0 & 1 \\ 0 & 0 & -1 & -1 \end{bmatrix}$$

e

$$b_3 = \begin{bmatrix} 2 \\ 4 \\ 3 \\ 2 \end{bmatrix}$$

Si puó verificare attraverso il Corollario 1 che A_3 é TU. Essendo b_3 un vettore di interi, il problema di PLI puó essere risolto eliminando semplicemente i vincoli di interezza.

3.3 Problemi di flusso a costo minimo

Sia data una rete (grafo orientato e connesso) $G = (V, A)$ come quella mostrata in Figura 3.4. Si consideri il seguente problema:

$$\begin{aligned}
&\min\ \textstyle\sum_{(i,j)\in A} c_{ij} x_{ij} \\
&\textstyle\sum_{j:(i,j)\in A} x_{ij} - \sum_{j:(j,i)\in A} x_{ji} = b_i \quad \forall\ i \in V \\
&x_{ij} \geq 0 \text{ interi} \qquad\qquad \forall\ (i,j) \in A
\end{aligned}$$

con b_i interi e tali che $\sum_{i \in V} b_i = 0$. Il problema viene interpretato come segue. Dovete inviare un flusso (di prodotti, di informazione, eccetera) attraverso la rete. Un'unitá di flusso inviata lungo l'arco (i, j) ha costo pari a c_{ij}. La variabile x_{ij} rappresenta la quantitá di flusso inviata lungo l'arco (i, j). La somma

$$\sum_{j:(j,i) \in A} x_{ji}$$

rappresenta il flusso complessivo *entrante* nel nodo i, la somma

$$\sum_{j:(i,j) \in A} x_{ij}$$

rappresenta il flusso complessivo *uscente* dal nodo i e quindi il vincolo

$$\sum_{j:(i,j) \in A} x_{ij} - \sum_{j:(j,i) \in A} x_{ji} = b_i$$

dice che la differenza tra flusso uscente e flusso entrante nel nodo i deve essere pari a b_i. Se $b_i > 0$ il flusso uscente supera quello entrante e quindi il nodo viene detto nodo *sorgente*. Se $b_i < 0$ il flusso entrante supera quello uscente ed il nodo viene detto nodo *destinazione*. Se $b_i = 0$ i due flussi entrante ed uscente si equivalgono ed il nodo viene detto di *transito*. In pratica ci sono nodi in cui il flusso viene prodotto (i nodi sorgente), altri in cui transita (i nodi transito) ed altri ancora verso cui viene convogliato (i nodi destinazione). Inviare un flusso x_{ij} lungo il generico arco (i, j) ha un costo pari a $c_{ij}x_{ij}$. Il problema da risolvere consiste nel trasportare attraverso la rete il flusso realizzato nei nodi sorgente (pari complessivamente a $\sum_{i \in V \,:\, b_i > 0} b_i$) facendolo giungere ai nodi destinazione (che, in virtú della condizione $\sum_{\in V} b_i = 0$, richiedono esattamente la stessa quantitá prodotta nei nodi sorgente) ed eseguire tale operazione in modo tale da avere un costo complessivo del flusso inviato lungo i diversi archi che sia il piú piccolo possibile.

Esempio 7 *Sia data la rete in Figura 3.4. I valori b_i sono riportati di fianco ai nodi mentre lungo gli archi sono riportati i valori c_{ij}. I nodi 1,2 e 3 sono nodi sorgente mentre i nodi 4 e 5 sono nodi destinazione (non vi sono nodi transito). Il problema corrispondente é il seguente*

$$\begin{gathered}
\min \; 5x_{12} - 4x_{23} + 6x_{42} - 2x_{13} + 0x_{34} + 2x_{15} + 4x_{53} + 3x_{45} \\
x_{12} + x_{13} + x_{15} = 2 \\
x_{23} - x_{12} - x_{42} = 5 \\
x_{34} - x_{13} - x_{23} - x_{53} = 1 \\
x_{42} + x_{45} - x_{34} = -4 \\
x_{53} - x_{15} - x_{45} = -4 \\
x_{12}, x_{23}, x_{42}, x_{13}, x_{34}, x_{15}, x_{53}, x_{45} \geq 0 \; \textit{interi}
\end{gathered}$$

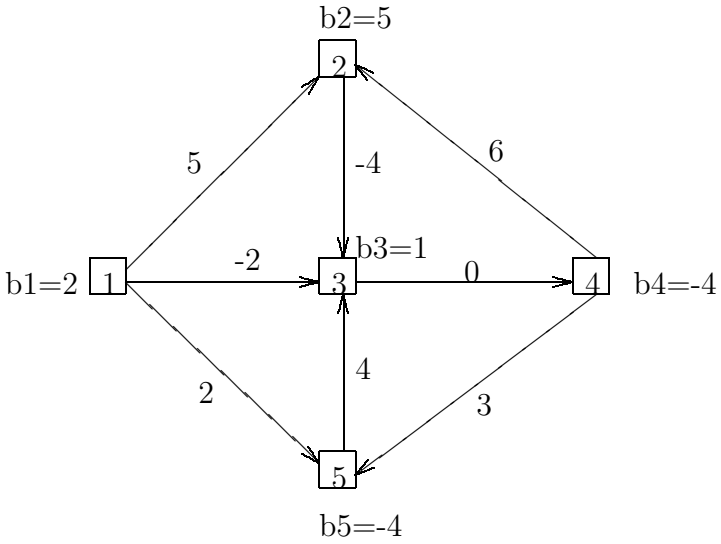


Figura 3.4: Una rete con i relativi valori b_i associati ai nodi ed i costi unitari di trasporto lungo gli archi.

Analizziamo ora la matrice dei vincoli per questi problemi. In essa avremo tante righe quanti sono i nodi della rete e tante colonne quanti sono gli archi della rete. Tale matrice é la *matrice di incidenza nodo-arco* della rete. Essa avrá nella colonna relativa all'arco (i, j) due soli elementi diversi da 0, un $+1$ nella riga i relativa al nodo da cui l'arco esce e un -1 nella riga j relativa al nodo in cui l'arco entra. Ma allora ci ritroviamo nella situazione di una matrice con elementi tutti uguali a 0, $+1$ o -1, con non piú di due elementi diversi da 0 lungo ogni colonna e con tali elementi di segno opposto. Il Corollario 1 ci dice che tale matrice é TU e quindi, essendo tutti i b_i interi per ipotesi, possiamo eliminare i vincoli di interezza sulle variabili. Quindi, i problemi di flusso a costo minimo, pur essendo problemi di PLI, sono problemi piú semplici dei generici problemi di PLI in quanto risolvibili come se fossero problemi di PL.

Esempio 8 *Nel nostro esempio la matrice di incidenza nodo-arco é la seguente:*

$$A = \begin{bmatrix} 1 & 1 & 1 & 0 & 0 & 0 & 0 & 0 \\ -1 & 0 & 0 & 1 & -1 & 0 & 0 & 0 \\ 0 & -1 & 0 & -1 & 0 & 1 & -1 & 0 \\ 0 & 0 & 0 & 0 & 1 & -1 & 0 & 1 \\ 0 & 0 & -1 & 0 & 0 & 0 & 1 & -1 \end{bmatrix} \tag{3.6}$$

Come giá anticipato, i problemi di PLI risolvibili come problemi di PL eliminando i vincoli di interezza hanno tipicamente una struttura tale da consentire di sviluppare tecmiche specifiche per tali problemi piú efficienti del simplesso o quantomeno di applicare il simplesso stesso con tecniche piú legate alla struttura del problema. Ció che vedremo nel seguito é il metodo del simplesso su reti,

ovvero il metodo del simplesso adattato a problemi di flusso a costo minimo su reti. Prima peró accenniamo brevemente ad un risultato sul rango della matrice di incidenza nodo-arco di una rete.

3.3.1 Rango della matrice di incidenza nodo-arco di una rete

Se sommiamo tra loro tutte le $\mid V \mid$ righe della matrice otteniamo il vettore nullo. Infatti in ogni colonna ci sono esattamente un +1 e un -1 (si faccia la verifica sull'esempio). Quindi le $\mid V \mid$ righe sono tra loro linearmente dipendenti ed il rango non potrá essere superiore a $\mid V \mid -1$. Si puó dimostrare (ma non lo faremo) che il rango é esattamente pari a $\mid V \mid -1$. Il fatto che $\sum_{i \in V} b_i = 0$ (e quindi non solo le righe della matrice sono linearmente dipendenti ma anche le equazioni stesse dei vincoli sono tra loro linearmente dipendenti) ci mostra che uno (ed un solo) vincolo del problema puó essere eliminato in quanto ridondante. Non importa quale vincolo si elimina. Come convenzione si puó fissare di eliminare l'ultima equazione. Nel seguito quindi l'ultimo vincolo si intenderá soppresso e quando si parlerá di matrice dei vincoli si intenderá la matrice di incidenza nodo-arco privata dell'ultima riga. Nel nostro esempio quindi la matrice dei vincoli sará

$$A = \begin{bmatrix} 1 & 1 & 1 & 0 & 0 & 0 & 0 & 0 \\ -1 & 0 & 0 & 1 & -1 & 0 & 0 & 0 \\ 0 & -1 & 0 & -1 & 0 & 1 & -1 & 0 \\ 0 & 0 & 0 & 0 & 1 & -1 & 0 & 1 \end{bmatrix}$$

ovvero la matrice (3.6) in cui é stata soppressa l'ultima riga.

3.3.2 Basi per il problema di flusso a costo minimo

Vedremo ora come il simplesso possa essere adattato ai problemi di flusso a costo minimo su reti. Cominceremo con lo stabilire il legame esistente tra basi del simplesso ed alberi di supporto della rete (rimandiamo alla sezione 2.1.1 per un richiamo al concetto di albero di supporto).
Si noti che, essendo il numero di righe (ed il rango) della matrice dei vincoli pari a $\mid V \mid -1$, le basi sono sempre formate da $\mid V \mid -1$ variabili. Ma non tutti gli aggregati di $\mid V \mid -1$ variabili danno origine ad una base. Per formare una base devono anche soddisfare la proprietá che la matrice ottenuta considerando le sole colonne relative ad esse nella matrice dei vincoli sia invertibile. Nel nostro esempio, se si considera l'albero di supporto in Figura 3.5 si ha che le colonne ad esso relativo nella matrice dei vincoli formano la seguente matrice

$$\begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & -1 & 1 & 0 \\ 0 & 0 & -1 & 1 \end{bmatrix}$$

che é invertibile e quindi le variabili corrispondenti formano una base.

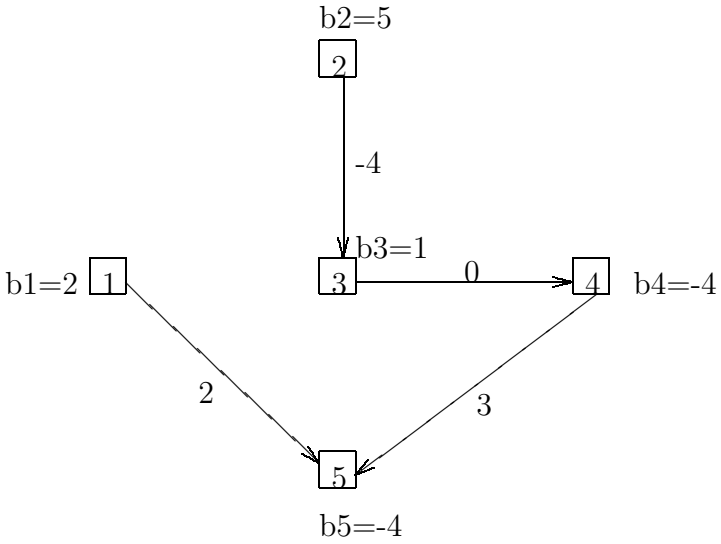


Figura 3.5: Un albero di supporto per la rete del nostro esempio.

3.3.3 Relazioni tra basi ed alberi di supporto

Si consideri un generico albero di supporto della rete. Esso sará formato da $\mid V \mid -1$ archi. Si puó dimostrare (ma non lo faremo) che la soluzione ottenuta mettendo le variabili relative agli archi dell'albero di supporto in base e tutte le altre fuori base si ottiene una soluzione di base (non necessariamente ammissibile e quindi non necessariamente un vertice) del problema. Abbozzeremo invece una dimostrazione del viceversa e cioé che data una qualsiasi base, i $\mid V \mid -1$ archi relativi alle variabili in base formano un albero di supporto. Per dimostrarlo ragioniamo per assurdo e supponiamo che gli archi non formino un albero di supporto. Poiché gli archi sono $\mid V \mid -1$, se non formano un albero di supporto devono formare almeno un ciclo. Vediamo cosa succede in presenza di un ciclo sul nostro esempio, precisando che quanto vedremo su tale esempio puó essere generalizzato a tutti i casi in cui compaia un ciclo. Supponiamo che le $\mid V \mid -1 = 4$ variabili in base siano quelle relative agli archi

$$(1,2)\ (2,3)\ (5,3)\ (1,5)$$

che formano il ciclo mostrato in Figura 3.6. Fissiamo un arco del ciclo, ad esempio $(1,2)$ ed imponiamo che il verso di percorrenza del ciclo sia quello dell'arco $(1,2)$. Per ogni colonna nella matrice dei vincoli relativa ad un arco del ciclo la moltiplichiamo per +1 se il ciclo attraversa l'arco nel suo verso, per -1 se lo attraversa nel verso opposto. Poi sommiamo i vettori ottenuti in questo modo. Nel nostro caso moltiplicheremo per +1 le colonne relative agli archi

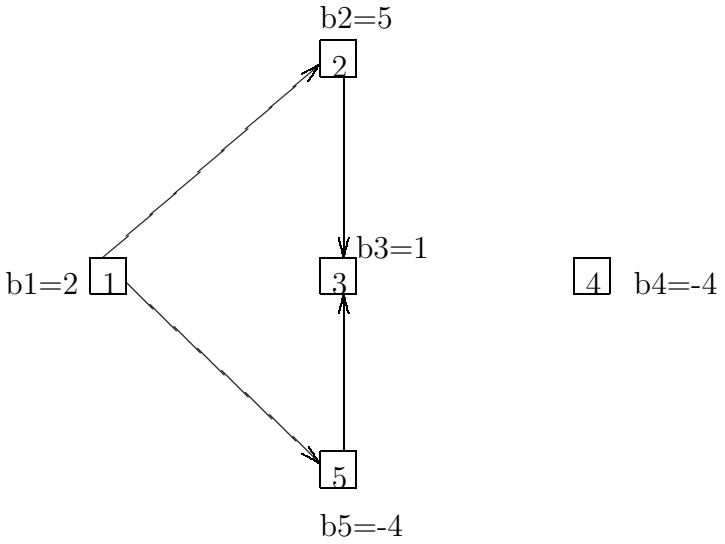


Figura 3.6: Un insieme di archi che formano un ciclo non possono dare origine ad una base.

$(1,2)$ e $(2,3)$ e per -1 quelle relative agli archi $(5,3)$ e $(1,5)$. Quindi avremo

$$+1\begin{bmatrix}1\\-1\\0\\0\end{bmatrix}+1\begin{bmatrix}0\\1\\-1\\0\end{bmatrix}-1\begin{bmatrix}0\\0\\-1\\0\end{bmatrix}-1\begin{bmatrix}1\\0\\0\\0\end{bmatrix}=\begin{bmatrix}0\\0\\0\\0\end{bmatrix}$$

Ció dimostra che esiste una combinazione lineare non nulla delle colonne che restituisce il vettore nullo. Quindi tali colonne non formano una matrice invertibile e non rappresentano una base. Come detto, é possibile generalizzare questo risultato: ogni qualvolta gli archi relativi ad un insieme di variabili formano un ciclo, le corrispondenti colonne della matrice dei vincoli sono linearmente dipendenti e quindi le variabili non formano una base. L'unica possibilitá per avere una base é che gli archi non formino alcun ciclo. Ma in base alla definizione di albero, in un grafo con $\mid V \mid$ nodi, $\mid V \mid -1$ archi che non formino alcun ciclo rappresentano un albero di supporto.
Abbiamo quindi mostrato il seguente importante risultato.

Osservazione 3 *In un problema di flusso su rete a costo minimo vi é una corrispondenza uno a uno tra basi ed alberi di supporto, ovvero ad ogni insieme di* $\mid V \mid -1$ *variabili che formano una base corrisponde un albero di supporto e viceversa.*

Quindi, per i problemi di flusso su reti a costo minimo sará indifferente parlare di basi o di alberi di supporto.

3.3.4 Alberi di supporto e soluzione di base corrispondente

Supponiamo ora di avere un albero di supporto (vedi Figura 3.5) nel nostro esempio e poniamoci la seguente domanda: in corrispondenza di tale albero e quindi di tale soluzione di base, qual é il valore delle variabili? Per prima cosa le variabili associate ad archi che non appartengono all'albero di supporto avranno associato un valore pari a 0. Quindi nel nostro esempio:

$$x_{12} = x_{13} = x_{42} = x_{53} = 0.$$

A questo punto sostituiamo tali valori nulli nei vincoli del problema. Quello che si ottiene é un sistema di $\mid V \mid -1$ variabili (quelle relative agli archi dell'albero di supporto) e $\mid V \mid -1$ equazioni. L'unica soluzione di tale sistema fornisce i valori delle variabili in base. Nel nostro esempio ponendo a 0 i valori delle variabili relative ad archi che non appartengono all'albero di supporto, otteniamo il seguente sistema:

$$\begin{aligned} x_{15} &= 2 \\ x_{23} &= 5 \\ x_{34} - x_{23} &= 1 \\ x_{45} - x_{34} &= -4 \end{aligned}$$

la cui soluzione é:

$$x_{15} = 2 \quad x_{23} = 5 \quad x_{34} = 6 \quad x_{45} = 2$$

Quindi la soluzione relativa all'albero di supporto dell'esempio é data da

$$x_{15} = 2 \quad x_{23} = 5 \quad x_{34} = 6 \quad x_{45} = 2 \quad x_{12} = x_{13} = x_{42} = x_{53} = 0$$

Si noti che tutte le variabili sono non negative e quindi in questo caso si parla di soluzione di base o albero di supporto *ammissibile* (e quindi si tratta di un vertice della regione ammissibile).
NOTA BENE Nel caso in cui una o piú delle variabili relative all'albero di supporto fossero uguali a 0 avremmo una soluzione *degenere*.

3.3.5 Calcolo dei coefficienti di costo ridotto

Come avete visto nel corso precedente, una condizione sufficiente per stabilire se, data una soluzione di base ammissibile (un vertice), ci troviamo in una soluzione ottima in un problema di PL, é controllare se tutti i coefficienti di costo ridotto sono tutti non positivi in un problema di massimo oppure tutti non negativi in un problema di minimo. Nella tabella del simplesso i coefficienti di costo ridotto appaiono nell'ultima riga della tabella. Nel simplesso su rete non abbiamo alcuna tabella e dobbiamo quindi vedere come calcolare tali valori. Per prima cosa ricordiamo che vanno calcolati per le sole variabili fuori base. Quindi i coefficienti vanno calcolati per le sole variabili associate ad archi che non fanno

parte dell'albero di supporto. La procedura per tale calcolo verrá illustrata sul nostro esempio. Prendiamo una qualsiasi variabile fuori base e quindi un qualsiasi arco che non faccia parte dell'albero di supporto, ad esempio l'arco $(1,3)$. Per prima cosa aggiungiamo l'arco all'albero. Si formerá esattamente un ciclo che verrá orientato nel verso dell'arco $(1,3)$ e quindi il ciclo sará

$$1 \to 3 \to 4 \to 5 \to 1$$

come si vede da Figura 3.7. Si noti che il ciclo attraversa gli archi $(1,3)$, $(3,4)$

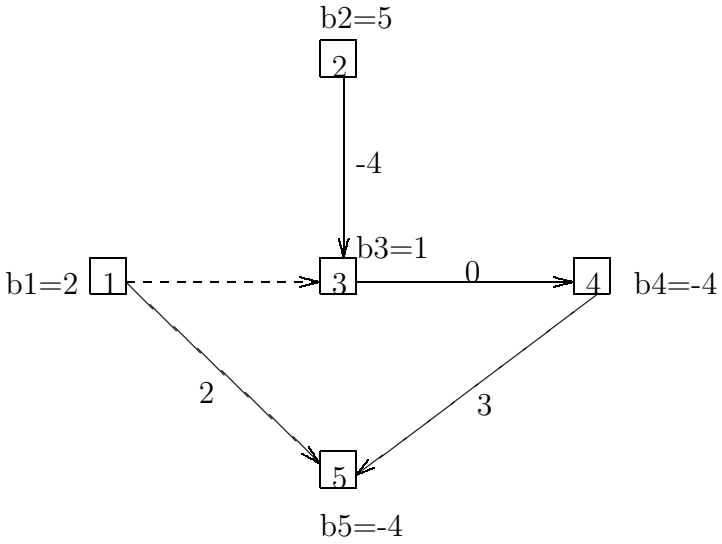


Figura 3.7: Il ciclo che si forma aggiungendo l'arco $(1,3)$.

e $(4,5)$ nel loro verso, mentre attraversa l'arco $(1,5)$ nel suo verso opposto. Il coefficiente di costo ridotto relativo all'arco $(1,3)$, indicato con $\overline{c}_{13}$ verrá calcolato sommando tra loro tutti i costi relativi agli archi attraversati dal ciclo nel loro stesso verso e sottraendo al risultato i costi degli archi attraversati dal ciclo in senso opposto al loro verso. Quindi

$$\overline{c}_{13} = c_{13} + c_{34} + c_{45} - c_{15} = -2 + 0 + 3 - 2 = -1.$$

Si noti che il coefficiente di costo ridotto é negativo e questo ci dice immediatamente che non possiamo concludere che la soluzione di base corrente é ottima. Possiamo ripetere la procedura per tutti gli archi fuori base per calcolare tutti i coefficienti di costo ridotto. Si ottengono i seguenti risultati:

$$\overline{c}_{42} = 2 \quad \overline{c}_{12} = 2 \quad \overline{c}_{53} = 7.$$

Come giá osservato, la presenza di un coefficiente di costo ridotto negativo ($\overline{c}_{13}$) ci impedisce di concludere che la soluzione di base corrente é ottima. In questi casi nel metodo del simplesso si procede ad un cambio di base attraverso un'operazione di cardine sulla tabella. Vediamo ora come questo viene fatto nel simplesso su rete.

3.3.6 Cambio di base ovvero l'operazione di cardine nel simplesso su rete

Per il cambio di base dovremo dare una regola per stabilire quale variabile fuori base far entrare in base e quale in base dovrá uscire dalla base. Per quanto riguarda la variabile fuori base da far entrare in base, la scelta é ristretta alle sole variabili con coefficiente di costo ridotto negativo (le sole incrementando le quali si puó far diminuire il costo complessivo del flusso). Tra queste fisseremo come regola di scegliere quella (o una di quelle, se sono piú di una) con il coefficiente di costo ridotto il piú negativo possibile. Nel nostro esempio non abbiamo alcuna scelta da fare visto che la sola variabile fuori base con coefficiente di costo ridotto negativo é quella relativa all'arco $(1,3)$. Aggiungiamo tale arco all'albero e riotteniamo la Figura 3.7. Inizialmente il flusso lungo l'arco $(1,3)$ é nullo ($x_{13}=0$). Incrementiamo a Δ il valore di tale flusso. Quindi avremo un nuovo flusso pari a Δ in uscita dal nodo 1 ed in entrata al nodo 3. Per poter continuare a rispettare i vincoli relativi al nodo 1 e 3 dovremo diminuire di Δ il flusso lungo l'arco $(1,5)$ ed aumentare di Δ il flusso lungo l'arco $(3,4)$. A questo punto per soddisfare il vincolo relativo al nodo 4 dobbiamo incrementare di Δ il flusso lungo l'arco $(4,5)$. Si noti che il vincolo relativo al nodo 5 é ancora soddisfatto poiché nel nodo 5 arriva un flusso pari a Δ in piú dal nodo 4 ma anche un flusso ancora pari a Δ in meno dal nodo 1. Gli archi relativi al nodo 2 non subiscono variazioni e quindi il vincolo relativo al nodo 2 continua ad essere soddisfatto. Si puó riassumere quanto visto nel modo seguente. Una volta aggiunto l'arco $(1,3)$ si forma un ciclo che viene orientato nel verso dell'arco $(1,3)$ stesso. Il flusso viene incrementato di Δ lungo ogni arco che il ciclo attraversa nel suo stesso verso e decrementato di Δ lungo gli archi che vengono attraversati in verso opposto. Quindi nel nostro esempio:

$$x_{13}=\Delta \quad x_{34}=6+\Delta \quad x_{45}=2+\Delta \quad x_{15}=2-\Delta.$$

A questo punto possiamo incrementare il valore di Δ arrestandoci nel momento in cui un flusso lungo un arco del ciclo si annulla. Nel nostro caso possiamo incrementare Δ fino a 2 ma non oltre in quanto incrementandolo oltre il flusso relativo all'arco $(1,5)$ diventerebbe negativo. La prima variabile che diventa nulla incrementando Δ corrisponderá alla variabile da far uscire di base. Se piú variabili diventano nulle contemporaneamente incrementando Δ (caso degenere) se ne seleziona una di esse arbitrariamente. L'albero di supporto corrispondente alla nuova base sará quello ottenuto inserendo l'arco relativo alla variabile fatta entrare in base (l'arco $(1,3)$ nel nostro esempio) e rimuovendo l'arco della variabile fatta uscire di base (l'arco $(1,5)$ nel nostro esempio). Per il nostro esempio la nuova base é quella riportata in Figura 3.8 ed i nuovi valori delle variabili sono i seguenti

$$x_{13}=2 \quad x_{23}=5 \quad x_{34}=8 \quad x_{45}=4 \quad x_{12}=x_{15}=x_{42}=x_{53}=0$$

NOTA BENE Se il ciclo ottenuto aggiungendo all'albero di supporto l'arco relativo alla variabile fuori base avesse tutti gli archi orientati nello stesso verso

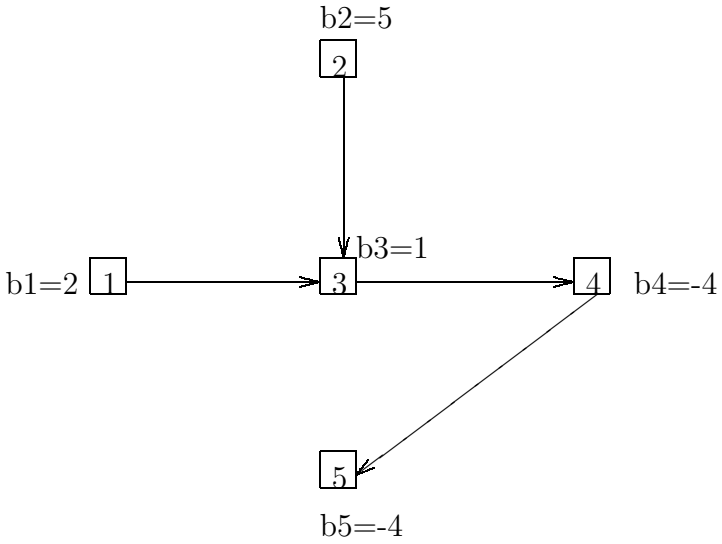


Figura 3.8: La nuova base (albero di supporto) del problema.

del ciclo stesso (vedi Figura 3.9) allora potrei far crescere Δ all'infinito senza che nessun flusso si annulli (tutti i flussi lungo il ciclo vengono incrementati). Ció corrisponde al caso di problema con obiettivo illimitato.

Possiamo ora concludere il nostro esempio andando a calcolare i nuovi coefficienti di costo ridotto. I risultati sono i seguenti.

$$\overline{c}_{42} = 2 \;\; \overline{c}_{15} = 1 \;\; \overline{c}_{12} = 3 \;\; \overline{c}_{53} = 7.$$

Essendo tutti non negativi si conclude che la soluzione corrente é ottima. Piú precisamente, essendo tutti non solo non negativi ma anche strettamente positivi, si conclude che la soluzione é anche l'unica soluzione ottima.

3.3.7 Determinazione di una soluzione di base ammissibile iniziale

Nella descrizione del simplesso su rete siamo partiti assumendo di avere giá a disposizione un albero di supporto ammissibile. Non sempre peró questo é vero e non é neppure detto che una soluzione ammissibile esista. Avremo quindi bisogno di una procedura che ci dica se ci sono soluzioni ammissibili e, nel caso esistano, ce ne restituisca una. Utilizzeremo una tecnica due fasi. Nella prima fase aggiungiamo alla nostra rete un nuovo nodo q e congiungiamo tale nodo con ogni nodo i della rete tale che $b_i < 0$ attraverso l'arco (q, i), mentre lo congiungiamo con ogni nodo i della rete tale che $b_i \geq 0$ attraverso l'arco (i, q). I valori b_i vengono lasciati invariati, mentre si pone $b_q = 0$. I costi dei flussi unitari saranno posti uguali a 1 per tutti gli archi incidenti sul nodo q e 0 per

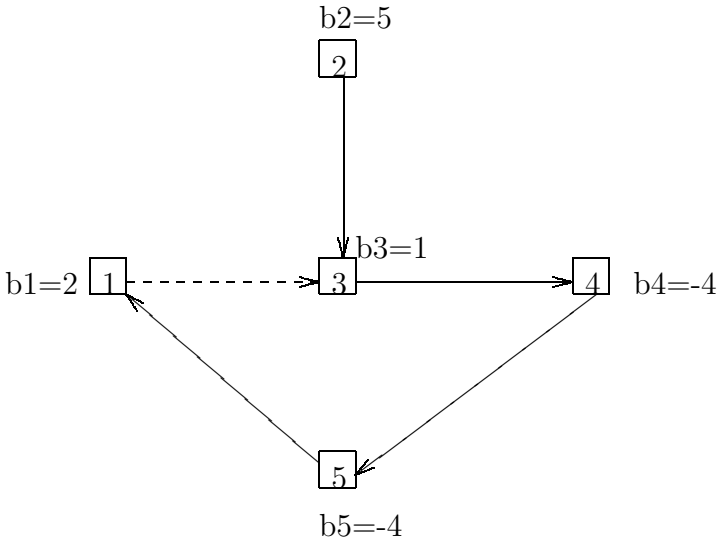


Figura 3.9: Tutti gli archi del ciclo hanno lo stesso orientamento: il problema ha obiettivo illimitato.

tutti gli archi della rete originaria. Per il nostro esempio la nuova rete sará quella in Figura 3.10. Per questo problema si ha immediatamente a disposizione un albero di supporto ammissibile, quello formato da tutti gli archi incidenti su q, con i seguenti valori delle variabili:

$$x_{qi} = -b_i \quad \forall\, i:\ b_i < 0$$
$$x_{iq} = b_i \quad \forall\, i:\ b_i \geq 0$$

mentre tutte le altre variabili sono nulle. A questo punto risolviamo questo problema con il simplesso su rete nel modo giá visto in precedenza. Se la soluzione ottima di tale problema é maggiore di 0, allora il problema originario ha regione ammissibile vuota. Se invece la soluzione ottima é pari a 0 e l'albero di supporto ottimo contiene solo uno dei nuovi archi (quelli incidenti su q), eliminando tale arco si ottiene un albero di supporto ammissibile per il problema originario. A questo punto possiamo eliminare il nodo q e tutti gli archi incidenti su di esso, ripristinare gli originari costi degli archi e cominciare a risolvere il problema (seconda fase del metodo). Non illustreremo la prima fase del metodo sul nostro solito esempio in quanto ci sarebbero troppi calcoli da fare. La illustreremo su un esempio di piú piccole dimensioni.

Esempio 9 *Si consideri la rete in Figura 3.11. Nella prima fase aggiungiamo il nodo q e gli archi incidenti su di esso ed aggiorniamo i costi dei flussi come indicato in Figura 3.12. Per il problema della prima fase un albero di supporto ammissibile é quello formato dagli archi incidenti su q, ovvero* $(1, q)$, $(2, q)$ *e*

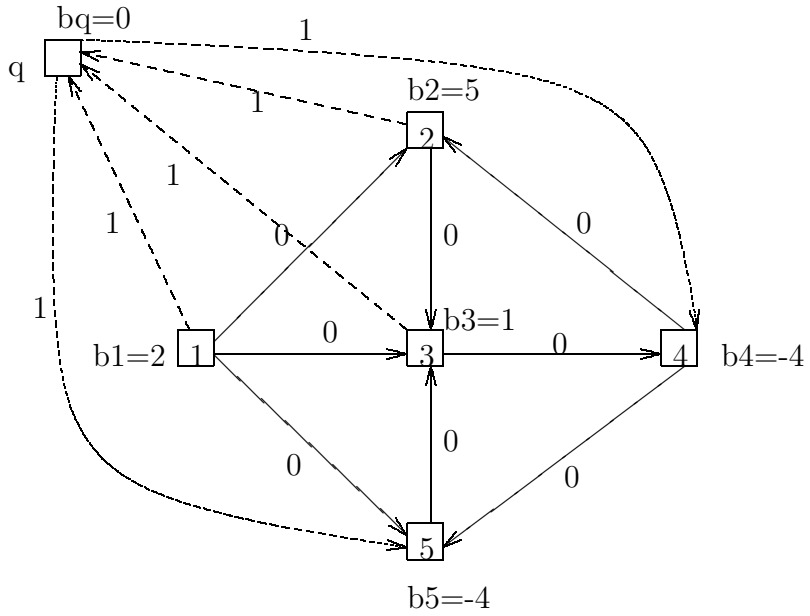


Figura 3.10: Il problema di prima fase per determinare una soluzione ammissibile iniziale.

$(q,3)$. *La soluzione iniziale é*

$$x_{q3} = 4 \; x_{2q} = 3 \; x_{1q} = 1,$$

tutte le altre variabili nulle. Il calcolo dei coefficienti di costo ridotto resituisce

$$\overline{c}_{12} = 0 \; \overline{c}_{13} = -2 \; \overline{c}_{23} = -2.$$

La soluzione non é ottima in quanto abbiamo coefficienti di costo ridotto negativi. Scelgo una delle variabili fuori base con coefficiente di costo ridotto piú negativo, ad esempio quella associata all'arco $(1,3)$. *Applicando la procedura per il cambio di base ottengo il nuovo albero di supporto* $(1,3)$, $(2,q)$ *e* $(q,3)$. *La nuova soluzione é*

$$x_{q3} = 3 \; x_{2q} = 3 \; x_{13} = 1,$$

tutte le altre variabili nulle. Il calcolo dei coefficienti di costo ridotto resituisce

$$\overline{c}_{12} = 2 \; \overline{c}_{1q} = 2 \; \overline{c}_{23} = -2.$$

La soluzione non é ottima in quanto abbiamo coefficienti di costo ridotto negativi. Scelgo una delle variabili fuori base con coefficiente di costo ridotto piú negativo, in tal caso c'é solo quella associata all'arco $(2,3)$. *Applicando la procedura per il cambio di base ottengo il nuovo albero di supporto* $(1,3)$, $(2,3)$ *e* $(q,3)$. *La nuova soluzione é*

$$x_{q3} = 0 \; x_{23} = 3 \; x_{13} = 1,$$

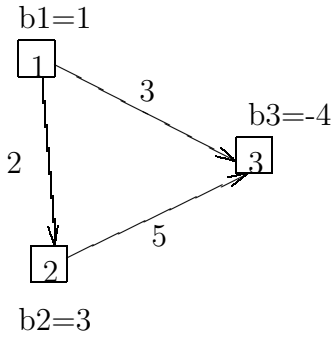


Figura 3.11: Una rete.

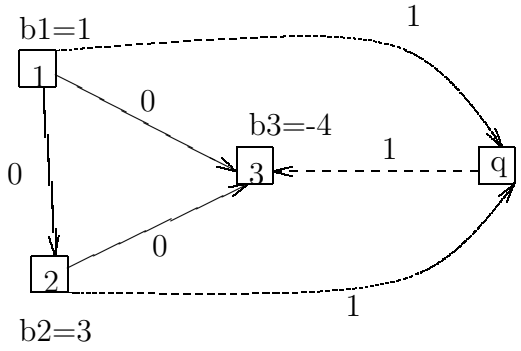


Figura 3.12: La rete ausiliaria per determinare un flusso ammissibile iniziale per la rete di Figura 3.11.

tutte le altre variabili nulle. Il calcolo dei coefficienti di costo ridotto resituisce

$$\overline{c}_{12} = 0 \; \overline{c}_{1q} = 2 \; \overline{c}_{2q} = 2.$$

La soluzione é ottima ed é pari a 0. Quindi il problema ammette soluzioni ammissibili. Inoltre, poiché la soluzione ottima contiene un solo arco incidente sul nodo q, eliminando tale arco ottengo immediatamente un albero di supporto ammissibile per il problema originario (quello formato dagli archi $(1,3)$ *e* $(2,3)$*) e con tale albero di supporto ammissibile sono pronto ad entrare nella seconda fase e risolvere il problema originario.*

3.3.8 Problemi di flusso a costo minimo con capacitá limitate sugli archi

Un'importante variante dei problemi di flusso a costo minimo che abbiamo trattato sino ad ora é quella in cui esistono dei limiti di capacitá associati agli archi. In pratica ad ogni arco $(i,j) \in A$ della rete é associato un valore intero, indicato

nel seguito con d_{ij}, che rappresenta il flusso massimo di prodotto inviabile lungo quell'arco. Se pensate all'esempio pratico di una rete di comunicazione, non potete inviare una quantitá infinita di prodotto nell'unitá di tempo lungo un cavo della rete, cioé avete proprio un limite di capacitá del vostro collegamento. Il modello matematico del problema viene modificato con la sola aggiunta dei vincoli di capacitá e diventa:

$$\begin{array}{cc} \min \sum_{(i,j)\in A} c_{ij}x_{ij} & \\ \sum_{j:(i,j)\in A} x_{ij} - \sum_{j:(j,i)\in A} x_{ji} = b_i & \forall\ i \in V \\ x_{ij} \leq d_{ij} & \forall\ (i,j) \in A \\ x_{ij} \geq 0 \text{ interi} & \forall\ (i,j) \in A \end{array}$$

In forma matriciale possiamo riscrivere i vincoli aggiuntivi di capacitá come segue:

$$IX \leq D$$

dove I é la matrice identica di ordine $\mid A \mid$, X il vettore con le $\mid A \mid$ componenti x_{ij} e D il vettore con le $\mid A \mid$ componenti intere d_{ij}. L'unimodularitá della matrice identica e l'interezza delle componenti di D consentono di estendere immediatamente anche al caso a capacitá limitate la possibilitá di risolverlo semplicemente eliminando i vincoli di interezza sulle variabili.

Dal punto di vista della risoluzione non potremo piú utilizzare l'algoritmo del simplesso come lo abbiamo visto in precedenza ma dovremo utilizzare una sua variante in grado di trattare variabili con limitazioni superiori. Resta del tutto invariata la corrispondenza uno ad uno tra basi e alberi di supporto. Ció che cambia é che se in precedenza in corrispondenza di ogni base una variabile poteva trovarsi in uno solo tra due possibili stati (in base oppure fuori base), ora i possibili stati di una variabile x_{ij} sono tre:

1. in base;
2. fuori base *a valore nullo* (ovvero $x_{ij} = 0$);
3. fuori base *a valore pari al proprio limite superiore* (ovvero $x_{ij} = d_{ij}$).

Mentre i primi due casi coincidono con quelli giá noti, la novitá é rappresentata dal terzo caso. Nel seguito indicheremo con B l'insieme delle variabili in base, con N_0 quello delle variabili fuori base a valore nullo, con N_1 quello delle variabili fuori base a valore pari al proprio limite superiore. Se nel caso precedente (senza limiti di capacitá) una soluzione di base associata ad una base data era immediatamente identificata una volta note quali erano le variabili in base, ora per poter calcolare la soluzione di base ammissibile é necessario anche sapere per ogni variabile fuori base se essa lo sia a valore nullo oppure a valore pari al proprio limite superiore. Una soluzione di base si definisce ammissibile se tutte le variabili in base hanno valore compreso tra 0 e la propria limitazione superiore. Inoltre, si parlerá di soluzione di base ammissibile non degenere

nel caso in cui nessuna variabile in base abbia valore pari a 0 o alla propria limitazione superiore. Di seguito introdurremo un esempio ed illustreremo su di esso quanto detto.

Esempio 10 *Si consideri un problema di flusso a costo minimo su una rete con 5 nodi con i seguenti valori b_i:*

$$b_1 = 8 \quad b_4 = -8 \quad b_2 = b_3 = b_5 = 0$$

e archi aventi i seguenti costi unitari di trasoprto e limiti di capacitá:

	$(1,2)$	$(1,3)$	$(1,4)$	$(2,4)$	$(3,4)$	$(3,5)$	$(5,4)$
c_{ij}	5	2	15	3	2	3	2
d_{ij}	9	7	3	11	2	7	7

Il modello matematico di tale problema (una volta eliminato il vincolo relativo al nodo 5, cosa che é possibile effettuare anche in questa variante del problema) é il seguente

$$\begin{gathered}
\min \ 5x_{12} + 2x_{13} + 15x_{14} + 3x_{24} + 2x_{34} + 3x_{35} + 2x_{54} \\
x_{12} + x_{13} + x_{14} = 8 \\
x_{24} - x_{12} = 0 \\
x_{34} + x_{35} - x_{13} = 0 \\
-x_{14} - x_{24} - x_{34} - x_{54} = -8 \\
x_{12} \leq 9 \quad x_{13} \leq 7 \\
x_{14} \leq 3 \quad x_{24} \leq 11 \\
x_{34} \leq 2 \quad x_{35} \leq 7 \\
x_{54} \leq 7 \\
x_{12}, x_{13}, x_{14}, x_{24}, x_{34}, x_{35}, x_{54} \geq 0 \text{ interi}
\end{gathered}$$

Consideriamo ora la base formata dalle 4 variabili $\{x_{12}, x_{13}, x_{24}, x_{35}\}$ (verificate che si tratta di un albero di supporto). Si sa inoltre che delle variabili fuori base due sono fuori a valore nullo (la x_{34} e la x_{54}), mentre una é fuori base a valore pari al proprio limite superiore (la x_{14}). Quindi abbiamo

$$B = \{x_{12}, x_{13}, x_{24}, x_{35}\} \quad N_0 = \{x_{34}, x_{54}\} \quad N_1 = \{x_{14}\}.$$

A questo punto, per ottenere il valore delle variabili in base non devo fare altro che sostituire nei vincoli ad ogni variabile fuori base il valore corrispondente (0 o il limite superiore della variabile) e risolvere quindi il sistema risultante. Nel nostro caso il sistema da risolvere sará il seguente:

$$\begin{gathered}
x_{12} + x_{13} + 3 = 8 \\
x_{24} - x_{12} = 0 \\
x_{35} - x_{13} = 0 \\
-3 - x_{24} = -8
\end{gathered}$$

da cui si ottiene la soluzione di base:

$$x_{12} = 5 \quad x_{24} = 5 \quad x_{13} = 0 \quad x_{35} = 0 \quad x_{34} = x_{54} = 0 \quad x_{14} = 3$$

con valore dell'obiettivo pari a 85. Tale soluzione di base é ammissibile ed é degenere (le variabili in base x_{13} e x_{35} hanno valore nullo).

Il calcolo dei coefficienti di costo ridotto é del tutto identico a quanto giá visto. In particolare per la base del nostro esempio abbiamo:

$$\overline{c}_{14} = 7 \quad \overline{c}_{34} = -4 \quad \overline{c}_{54} = -1$$

Per le variabili fuori base a valore nullo vale il discorso giá fatto in precedenza: tra queste quelle che conviene far crescere se vogliamo ridurre il valore dell'obiettivo sono quelle a coefficiente di costo ridotto negativo. La novitá é rappresentata dal fatto che per le variabili fuori base con valore pari al proprio limite superiore non é ovviamente possibile farle crescere (sono, appunto, giá al loro limite superiore) ma solo farle decrescere. Ne consegue che per variabili fuori base al proprio limite superiore quelle la cui variazione (ovvero diminuzione) consente un decremento del valore dell'obiettivo sono quelle con coefficiente di costo ridotto positivo. Avremo quindi la seguente condizione di ottimalitá: una soluzione di base ammissibile é soluzione ottima del problema se:

a) tutte le variabili fuori base a valore nullo hanno coefficiente di costo ridotto non negativo;

b) tutte le variabili fuori base a valore pari al limite superiore hanno coefficiente di costo ridotto non positivo.

Formalmente:

$$\overline{c}_{ij} \geq 0 \ \ \forall\ (i,j) \in N_0 \qquad \overline{c}_{ij} \leq 0 \ \ \forall\ (i,j) \in N_1.$$

Nel nostro esempio la condizione non é soddisfatta e si dovrá procedere ad un cambio di base.

La variabile che si decide di far entrare in base é la piú promettente (quella la cui variazione modifica piú rapidamente il valore dell'obiettivo). Questa la si identifica prendendo quella che fornisce il massimo tra i coefficienti di costo ridotto delle variabili fuori base a valore pari al limite superiore e quelli cambiati di segno delle variabili fuori base a valore nullo. Formalmente:

$$(i,j) \in \arg\max\{\max_{(i,j)\in N_1} \overline{c}_{ij}, \max_{(i,j)\in N_0} -\overline{c}_{ij}\}$$

Nel nostro esempio si tratta di confrontare tra loro il coefficiente di costo ridotto della variabile fuori base x_{14} (pari a 7) e i coefficienti di costo ridotto cambiati di segno di x_{35} e x_{54} (rispettivamente pari a 4 e 1). Il massimo é raggiunto da x_{14} e quindi questa sará la variabile che tenteremo di far entrare in base (in caso

di paritá tra piú variabili se ne sceglie una a caso).

Per quanto riguarda la scelta della variabile che dovrá uscire dalla base, il procedimento é simile a quanto visto nel caso senza capacitá sugli archi ma con qualche variante. Se la variabile che decidiamo di far entrare in base é fuori base a valore nullo, allora si procede come nel caso senza capacitá, ovvero si fa passare da 0 a Δ il flusso lungo l'arco che si fa entrare in base e si modificano secondo le regole viste in precedenza i flussi lungo gli archi del ciclo che si viene a formare aggiungendo l'arco che si vuol far entrare in base all'albero di supporto relativo alla base attuale. Invece nel caso in cui la variabile che stiamo cercando di far entrare in base sia fuori base a valore pari al proprio limite superiore si riduce il valore di tale variabile dal proprio limite superiore d_{ij} a $d_{ij} - \Delta$ mentre sugli archi del ciclo che si forma con l'aggiunta dell'arco relativo a questa variabile i flussi sono aggiornati secondo regole inverse rispetto a quelle viste in precedenza (ovvero il flusso viene diminuito di Δ lungo gli archi attraversati secondo il proprio verso dal ciclo ed incrementato di *Delta* lungo gli archi attarversati dal ciclo in verso opposto al proprio).
Ma fino a quanto possiamo far crescere il valore Δ? La crescita puó avvenire fino a quando si verifica uno dei seguenti eventi:

1. una variabile in base si annulla (in tal caso essa uscirá dalla base e diventerá fuori base a valore nullo, cioé passerá in N_0, mentre in B entrerá la nostra variabile fuori base);

2. una variabile in base raggiunge il proprio limite superiore (in tal caso essa uscirá dalla base e diventerá fuori base a valore pari al proprio limite superiore, cioé passerá in N_1, mentre in B entrerá la nostra variabile fuori base);

3. la variabile fuori base che stiamo cercando di far entrare in base raggiunge il proprio limite superiore (se era a valore nullo) o si annulla (se era a valore pari al proprio limite superiore): in tal caso la base B non cambia, cambia solamente lo stato della variabile fuori base che abbiamo tentato di far entrare in base, la quale passa da fuori base a valore nullo a fuori base a valore pari al proprio limite superiore (cioé da N_0 in N_1) o viceversa.

Ma vediamo di capire cosa succede nel nostro esempio. Se cerchiamo di far entrare in base x_{14} abbiamo le seguenti variazioni nei valori delle variabili

$$x_{12} = 5 + \Delta \quad x_{24} = 5 + \Delta \quad x_{13} = 0 \quad x_{35} = 0 \quad x_{34} = x_{54} = 0 \quad x_{14} = 3 - \Delta$$

con un valore dell'obiettivo pari a $85 - 7\Delta$. Si vede che Δ puó crescere al massimo fino al valore 3 (in corrispondenza di tale valore la variabile x_{14} si annulla). Quindi la base non cambia ma cambia lo stato della variabile x_{14} che da fuori base al proprio limite superiore passa a fuori base a valore nullo. La nuova soluzione di base é la seguente:

$$x_{12} = 8 \quad x_{24} = 8 \quad x_{13} = 0 \quad x_{35} = 0 \quad x_{34} = x_{54} = x_{14} = 0$$

con un valore dell'obiettivo pari a 64. In questo momento abbiamo:

$$B = \{x_{12}, x_{13}, x_{24}, x_{35}\} \quad N_0 = \{x_{14}, x_{34}, x_{54}\}.$$

A questo punto la procedura viene iterata. Calcoliamo i coefficienti di costo ridotto (in realtá non essendo cambiata la base essi rimangono identici a prima):

$$\overline{c}_{14} = 7 \quad \overline{c}_{34} = -4 \quad \overline{c}_{54} = -1.$$

La condizione di ottimalitá non é soddisfatta (i coefficienti di costo ridotto di x_{34} e x_{54}, entrambe in N_0, sono negativi). La variabile che si decide di far entrare in base é la x_{34}. Se cerchiamo di far entrare in base x_{34} abbiamo le seguenti variazioni nei valori delle variabili

$$x_{12} = 8 - \Delta \quad x_{24} = 8 - \Delta \quad x_{13} = \Delta \quad x_{35} = 0 \quad x_{34} = \Delta \quad x_{54} = x_{14} = 0$$

con un valore dell'obiettivo pari a $64-4\Delta$. Si vede che Δ puó crescere al massimo fino al valore 2 (in corrispondenza di tale valore la variabile x_{34} raggiunge il proprio limite superiore). Quindi la base non cambia ma cambia lo stato della variabile x_{34} che da fuori base al proprio limite superiore passa a fuori base a valore nullo. La nuova soluzione di base é la seguente:

$$x_{12} = 6 \quad x_{24} = 6 \quad x_{13} = 2 \quad x_{35} = 0 \quad x_{34} = 2 \quad x_{54} = x_{14} = 0$$

con un valore dell'obiettivo pari a 56. In questo momento abbiamo:

$$B = \{x_{12}, x_{13}, x_{24}, x_{35}\} \quad N_0 = \{x_{14}, x_{54}\} \quad N_1 = \{x_{34}\}.$$

Continuiamo con la nostra procedura. Calcoliamo i coefficienti di costo ridotto (in realtá non essendo cambiata la base essi continuano a rimanere identici a prima):

$$\overline{c}_{14} = 7 \quad \overline{c}_{34} = -4 \quad \overline{c}_{54} = -1.$$

La condizione di ottimalitá non é soddisfatta (il coefficiente di costo ridotto di x_{54}, che é in N_0, é negativo). La variabile che si decide di far entrare in base é la x_{54}. Se cerchiamo di far entrare in base x_{54} abbiamo le seguenti variazioni nei valori delle variabili

$$x_{12} = 6 - \Delta \quad x_{24} = 6 - \Delta \quad x_{13} = 2 + \Delta \quad x_{35} = \Delta \quad x_{54} = \Delta \quad x_{34} = 2 \quad x_{14} = 0$$

con un valore dell'obiettivo pari a $56-\Delta$. Si vede che Δ puó crescere al massimo fino al valore 5 (in corrispondenza di tale valore la variabile in base x_{13} raggiunge il proprio limite superiore). Quindi la base ora cambia in quanto in essa la variabile x_{13} viene sostituita dalla x_{54}. La nuova soluzione di base é la seguente:

$$x_{12} = 1 \quad x_{54} = 5 \quad x_{13} = 7 \quad x_{35} = 5 \quad x_{34} = 2 \quad x_{24} = 1 \quad x_{14} = 0$$

con un valore dell'obiettivo pari a 51. In questo momento abbiamo:

$$B = \{x_{12}, x_{24}, x_{54}, x_{35}\} \quad N_0 = \{x_{14}\} \quad N_1 = \{x_{13}, x_{34}\}.$$

Calcoliamo i coefficienti di costo ridotto:

$$\overline{c}_{14} = 7 \quad \overline{c}_{34} = -3 \quad \overline{c}_{13} = -1.$$

Da essi possiamo concludere che la solzuione di base ammissibile attuale é soluzione ottima (unica) del nostro problema.

Il problema di stabilire se esistono o meno soluzioni ammissibili del problema si risolve con una procedura a due fasi del tutto analoga a quella vista per il problema senza vincoli di capacitá sugli archi (nel problema di I fase agli archi incidenti sul nodo aggiuntivo q si assegna capacità infinita, mentre quelli originari mantengono la loro capacità).

3.4 Il problema di flusso massimo

Si consideri una rete, ovvero un grafo orientato $G = (V, A)$. Attraverso tale rete si fa viaggiare quello che chiameremo genericamente un *flusso* che puó essere, a seconda delle applicazioni, un flusso di prodotti se la rete é una rete stradale, di informazione se é una rete di comunicazione, di acqua se é una rete idraulica e cosí via. Tra i nodi della rete si riconoscono:

- un nodo *sorgente*, che nel seguito indicheremo con S, da cui il flusso parte;
- un nodo *destinazione*, che nel seguito indicheremo con D, a cui il flusso arriva.

Tutti gli altri nodi vengono detti *intermedi* e sono caratterizzati dal fatto che in essi la quantitá di flusso entrante é sempre pari a quella uscente (vincoli di *equilibrio*). Gli archi della rete hanno una *capacitá* limitata che rappresenta la quantitá massima di flusso che puó attraversare tali archi. Il *problema di massimo flusso* consiste nel determinare la quantitá massima di flusso che partendo dal nodo sorgente si puó far giungere fino al nodo destinazione, tenuto conto dei vincoli di capacitá sugli archi e di quelli di equilibrio nei nodi intermedi.
NB Abbiamo parlato di un solo nodo sorgente e di un solo nodo destinazione nella rete. In problemi reali il flusso puó essere generato da piú sorgenti e/o essere ricevuto da piú destinazioni. Tali casi si possono facilmente ricondurre a quello di una sola sorgente e di una sola destinazione attraverso l'introduzione di una sorgente fittizia collegata tramite archi fittizi a capacitá infinita a ciascuna sorgente reale e, analogamente, attraverso l'introduzione di una destinazione fittizia alla quale si giunge tramite archi fittizi a capacitá infinita a partire da ciascuna destinazione reale (vedi Figura 3.13).

3.4.1 Modello matematico del problema

Vediamo ora di introdurre un modello matematico che rappresenti il nostro problema. Associamo ad ogni arco della rete $(i, j) \in A$ una variabile:

$$x_{ij} = \text{flusso inviato lungo l'arco } (i, j)$$

Tali variabili saranno vincolate ad essere non negative (non ha senso parlare di un flusso negativo). Se indichiamo con c_{ij} la capacitá dell'arco (i, j) si dovrá anche avere

$$x_{ij} \leq c_{ij} \quad \forall\ (i, j) \in A,$$

cioé il flusso lungo ogni arco non ne puó superare la capacitá. In molti casi le variabili possono assumere solo valori interi ma, come vedremo in seguito, non sará necessario imporre esplicitamente il vincolo di interezza sulle variabili. Il nostro obiettivo é quello di massimizzare la quantitá di flusso uscente dal nodo sorgente S:

$$\sum_{j:\ (S,j)\in A} x_{Sj}$$

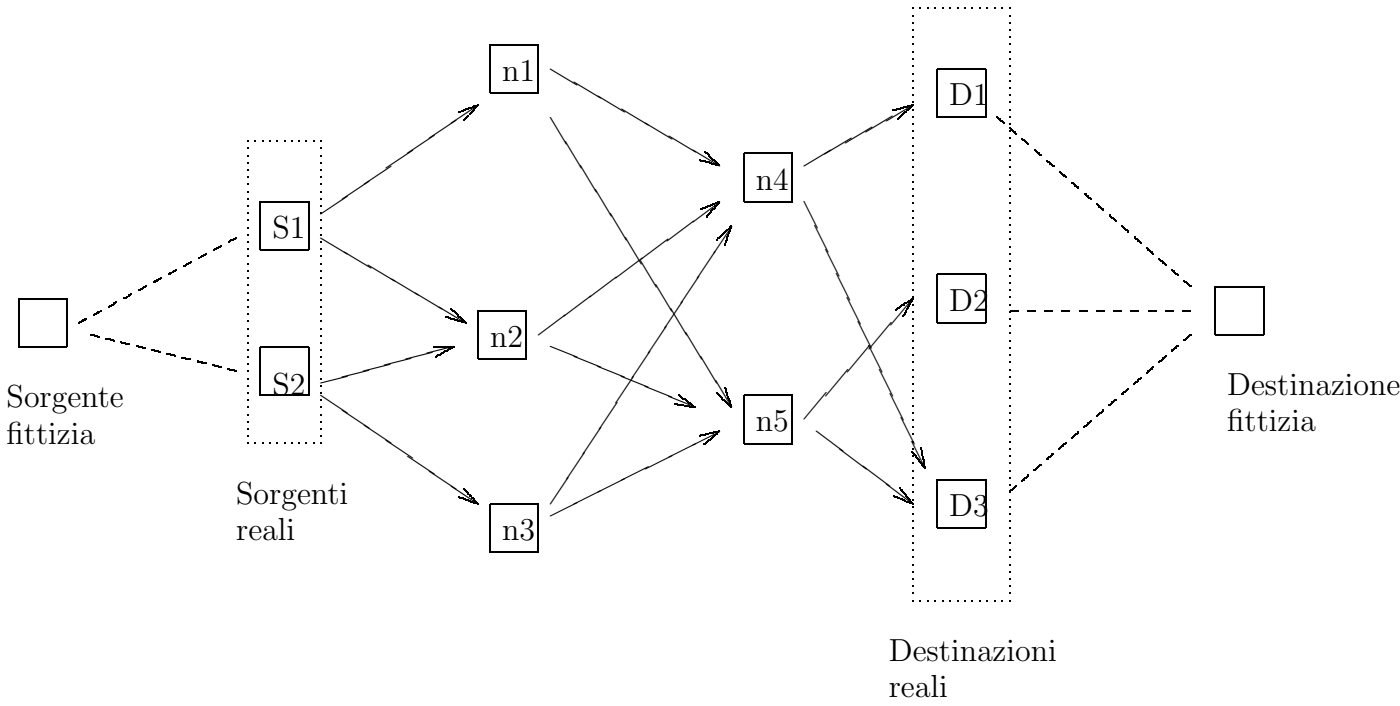


Figura 3.13: La riduzione al caso di una sola sorgente e una sola destinazione.

o, equivalentemente, quella entrante nel nodo destinazione D:

$$\sum_{j:\ (j,D)\in A} x_{jD}$$

(le due quantitá sono uguali). Restano ancora da esprimere i vincoli di equilibrio nei nodi intermedi. Questi possono essere tradotti nelle seguenti equazioni:

$$\underbrace{\sum_{j:\ (k,j)\in A} x_{kj}}_{\text{flusso uscente da } k} = \underbrace{\sum_{j:\ (j,k)\in A} x_{jk}}_{\text{flusso entrante in } k} \qquad \forall\ k \in V \setminus \{S, D\}.$$

Riassumendo, il modello matematico del problema di massimo flusso é il seguente:

$$\begin{array}{ll} \max & \sum_{j:\ (S,j)\in A} x_{Sj} \\ & \sum_{j:\ (k,j)\in A} x_{kj} = \sum_{j:\ (j,k)\in A} x_{jk} \quad \forall\ k \in V \setminus \{S, D\} \\ & 0 \le x_{ij} \le c_{ij} \qquad\qquad\qquad\quad \forall\ (i,j) \in A \end{array} \tag{3.7}$$

In forma matriciale il problema puó essere scritto nella seguente forma:

$$\begin{aligned} \max \quad & \delta^S X \\ & MX = 0 \\ & IX \leq C \\ & X \geq 0 \end{aligned}$$

dove:

- X é il vettore di variabili di dimensione $\mid A \mid$ con componenti le variabili x_{ij};
- δ^S é un vettore di dimensione $\mid A \mid$ la cui componente associata all'arco (i, j) é la seguente:
$$\delta^S_{ij} = \begin{cases} 1 & \text{se } i = S \\ 0 & \text{altrimenti} \end{cases}$$
- I é la matrice identica di ordine $\mid A \mid$;
- C il vettore di dimensione $\mid A \mid$ con componenti le capacitá c_{ij};
- M é una matrice di ordine $(\mid V \mid -2) \times \mid A \mid$ che coincide con la matrice di incidenza nodo-arco della rete dopo che a questa sono state rimosse le righe relative ai nodi S e D.

Da questa riformulazione in forma matriciale possiamo notare un fatto importante. La matrice I é una matrice TU. Lo stesso vale per la matrice M (ogni matrice di incidenza nodo-arco di un grafo orientato é TU e si puó vedere, come conseguenza del Corollario 1, che rimuovendo righe da una tale matrice si ottiene ancora una matrice TU).
Quindi sulla base del Teorema 1 con $A_1 \equiv I$, $b_1 \equiv C$, $A_3 \equiv M$ e $b_3 \equiv 0$, si ha che se il vettore C di capacitá é un vettore di interi (come spesso succede nei problemi reali), allora tutti i vertici ottimi del problema di flusso massimo sono a coordinate intere. Abbiamo quindi dimostrato il seguente teorema.

Teorema 2 *Se le capacitá c_{ij} degli archi sono tutti valori interi, allora tutti i vertici ottimi del problema di flusso massimo (3.7) sono a coordinate intere.*

Esempio 11 *Sia data la rete in Figura 3.14. I numeri sugli archi ne indicano le capacitá. Il modello matematico del problema é il seguente problema di PL (essendo le capacitá valori interi non avremo bisogno di imporre vincoli di*

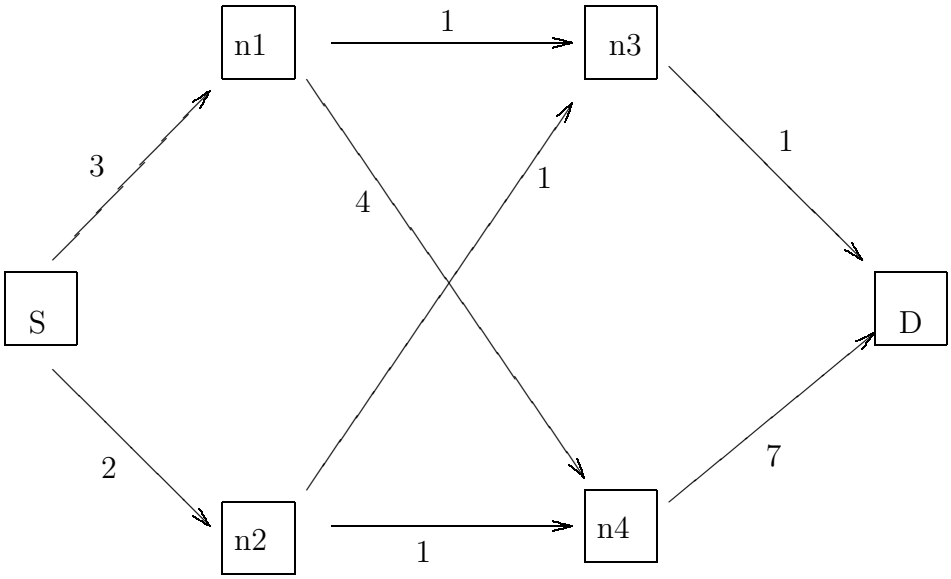


Figura 3.14: Una rete con le capacitá degli archi indicati al loro fianco.

interezza sulle variabili):

$$
\begin{aligned}
\max \quad & x_{S1} + x_{S2} \\
& x_{13} + x_{14} = x_{S1} \\
& x_{23} + x_{24} = x_{S2} \\
& x_{3D} = x_{13} + x_{23} \\
& x_{4D} = x_{14} + x_{24} \\
& 0 \leq x_{S1} \leq 3 \\
& 0 \leq x_{S2} \leq 2 \\
& 0 \leq x_{13} \leq 1 \\
& 0 \leq x_{14} \leq 4 \\
& 0 \leq x_{23} \leq 1 \\
& 0 \leq x_{24} \leq 1 \\
& 0 \leq x_{3D} \leq 1 \\
& 0 \leq x_{4D} \leq 7
\end{aligned}
$$

La matrice M per questo esempio é la seguente:

M :

	$(S,1)$	$(S,2)$	$(1,3)$	$(1,4)$	$(2,3)$	$(2,4)$	$(3,D)$	$(4,D)$
1	-1	0	1	1	0	0	0	0
2	0	-1	0	0	1	1	0	0
3	0	0	-1	0	-1	0	1	0
4	0	0	0	-1	0	-1	0	1

3.4.2 Tagli e problema del taglio minimo

Prima di descrivere un algoritmo per risolvere il problema di flusso massimo accenniamo ad un altro problema che, come vedremo, é strettamente legato a quello del flusso massimo. Si consideri $U \subset V$ con la proprietá che:

$$S \in U \quad D \notin U.$$

L'insieme di archi

$$\mathcal{S}_U = \{(i,j) \in A \ : \ i \in U, \ j \notin U\},$$

ovvero gli archi con il primo estremo in U e l'altro al di fuori di U, viene detto *taglio* della rete. Si noti che eliminando tutti gli archi di un taglio dalla rete rendo impossibile raggiungere D a partire da S in quanto ció equivale ad eliminare tutti gli archi che vanno da U (contenente S) al suo complementare $\overline{U} = V \setminus U$ (contenente D). Ad un taglio si associa un *costo* pari alla somma delle capacitá degli archi del taglio, cioé:

$$C(\mathcal{S}_U) = \sum_{(i,j) \in \mathcal{S}_U} c_{ij}.$$

Nell'esempio in Figura 3.14, l'insieme $U = \{S, n_1, n_2\}$ induce il taglio $\mathcal{S}_U = \{(n_1,n_3),(n_1,n_4),(n_2,n_3),(n_2,n_4)\}$ con capacitá $C(\mathcal{S}_U) = 7$. Si puó notare che, dato un taglio $\mathcal{S}_U$ qualsiasi, il valore del flusso massimo nella rete non puó superare quello del costo del taglio. Infatti, per poter passare dall'insieme di nodi U contenente la sorgente S al suo complementare $\overline{U}$ contenente la destinazione D il flusso puó solo passare attraverso gli archi del taglio $\mathcal{S}_U$ e quindi il flusso non puó superare la somma delle capacitá di tali archi, ovvero il costo del taglio. Quindi il costo di ogni taglio rappresenta un limite superiore per il valore del flusso massimo (nel nostro esempio sappiamo quindi giá che il flusso massimo non puó superare il valore 7, ovvero il costo del taglio indotto da $U = \{S, n_1, n_2\}$). Possiamo anche spingerci piú in lá e dire che tra tutti i tagli ve ne é almeno uno il cui costo é *esattamente pari* a quello del flusso massimo. Piú precisamente, consideriamo il problema del taglio di costo minimo che consiste nel determinare, tra tutti i tagli possibili all'interno di una rete, quello il cui costo sia il piú piccolo possibile, ovvero

$$\min_{U \subset V \ : \ S \in U, \ D \notin U} C(\mathcal{S}_U). \tag{3.8}$$

Avremo modo di dimostrare che i valori ottimi del problema di flusso massimo (3.7) e quello di taglio a costo minimo (3.8) sono uguali tra loro. Non solo, l'algoritmo con cui risolveremo il problema di flusso massimo ci dará immediatamente anche una soluzione per il problema di taglio a costo minimo. Richiamando le nozioni di dualitá nella PL, possiamo dire che il problema di taglio a costo minimo é un problema *duale* del problema di flusso massimo.

3.4.3 Algoritmo di Ford-Fulkerson

Ci occuperemo ora di descrivere un possibile algoritmo di risoluzione per il problema di flusso massimo, chiamato *algoritmo di Ford-Fulkerson*. Prima di descrivere l'algoritmo abbiamo bisogno di introdurre alcuni concetti. Supponiamo di avere un flusso ammissibile:

$$\overline{X} = (\overline{x}_{ij})_{(i,j)\in A},$$

ovvero un flusso che soddisfa i vincoli di equilibrio e quelli di capacitá degli archi. Se $\overline{x}_{ij} = c_{ij}$ diremo che l'arco (i, j) é *saturo*. Si consideri ora un cammino orientato nella rete dal nodo S al nodo D:

$$S = q_0 \to q_1 \to \cdots \to q_r \to q_{r+1} = D,$$

privo di archi saturi, ovvero nesuno degli archi

$$(q_i, q_{i+1}) \in A \quad i = 0, \ldots, r$$

é saturo. In tal caso il flusso $\overline{X}$ non é ottimo. Infatti, posso aumentare il flusso lungo ciascun arco del cammino di una quantitá Δ definita nel seguente modo:

$$\Delta = \min_{i=0,\ldots,r}[c_{q_i q_{i+1}} - \overline{x}_{q_i q_{i+1}}],$$

senza violare i vincoli di capacitá degli archi. Si noti anche che i vincoli di equilibrio continuano ad essere rispettati in quanto in ogni nodo intermedio del grafo facente parte del cammino si ha che il flusso in entrata aumenta di Δ ma contemporaneamente aumenta di Δ anche quello in uscita. Essendo gli archi del cammino non saturi si ha che $\Delta > 0$ e il flusso totale da S a D viene aumentato proprio di questa quantitá. Per illustrare questa situazione si consideri l'esempio in Figura 3.15 dove di fianco agli archi sono riportati due numeri, il primo corrispondente al flusso attuale $\overline{x}_{ij}$ lungo l'arco, l'altro corrispondente alla capacitá c_{ij} dell'arco. Attuamente il flusso in uscita dal nodo S é pari a 2 (1 unitá lungo l'arco (S, n_1) e 1 lungo l'arco (S, n_2)). Questo flusso non é ottimo. Notiamo infatti che esiste almeno un cammino orientato da S a D con archi non saturi, ad esempio il cammino

$$S \to n_1 \to n_2 \to D$$

Lungo tale cammino si ha

$$\Delta = \min\{c_{Sn_1} - \overline{x}_{Sn_1},\ c_{n_1n_2} - \overline{x}_{n_1n_2},\ c_{n_2D} - \overline{x}_{n_2D}\} = 1.$$

Quindi posso incrementare di 1 unitá il flusso lungo gli archi del cammino, cioé avró il seguente aggiornamento del flusso:

$$\overline{x}_{Sn_1} = \overline{x}_{Sn_1} + \Delta \quad \overline{x}_{Sn_2} = \overline{x}_{Sn_2} \quad \overline{x}_{n_1n_2} = \overline{x}_{n_1n_2} + \Delta$$

$$\overline{x}_{n_1D} = \overline{x}_{n_1D} \quad \overline{x}_{n_2D} = \overline{x}_{n_2D} + \Delta$$

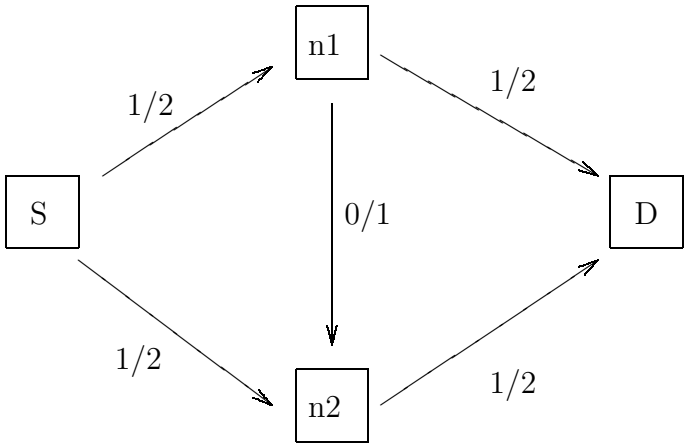


Figura 3.15: I numeri a fianco di ogni arco rappresentano rispettivamente il flusso lungo di esso e la sua capacitá.

La nuova situazione é illustrata in Figura 3.16. Ora il flusso totale uscente da S é salito anch'esso della quantitá Δ passando da 2 a 3. Possiamo notare che ora tutti i cammini orientati da S a D contengono almeno un arco saturo. Possiamo allora concludere che il flusso attuale é il massimo possibile? La risposta é no e lo dimostreremo nel seguito. Prima definiamo un nuovo grafo orientato $G(\overline{X}) = (V, A(\overline{X}))$ detto *grafo associato al flusso* $\overline{X}$. Il nuovo grafo ha gli stessi nodi della rete originaria e ha il seguente insieme di archi:

$$A(\overline{X}) = \underbrace{\{(i,j) \ : \ (i,j) \in A, \ \overline{x}_{ij} < c_{ij}\}}_{A_f(\overline{X})} \cup \underbrace{\{(i,j) \ : \ (j,i) \in A, \ \overline{x}_{ji} > 0\}}_{A_b(\overline{X})},$$

ovvero $A(\overline{X})$ contiene tutti gli archi di A non saturi (l'insieme $A_f(\overline{X})$, detto insieme degli archi *forward*) e tutti gli archi di A lungo cui é stata inviata una quantitá positiva di flusso, cambiati peró di verso (l'insieme $A_b(\overline{X})$, detto insieme degli archi *backward*). Vediamo di generare il grafo associato al flusso attuale $\overline{X}$ del nostro esempio. Si ha che:

$$A_f(\overline{X}) = \{(S, n_2), \ (n_1, D)\}$$

$$A_b(\overline{X}) = \{(n_1, S), \ (n_2, S), \ (n_2, n_1), \ (D, n_1), \ (D, n_2)\}.$$

Il grafo é rappresentato in Figura 3.17. Poniamoci ora la seguente domanda: esiste sul nuovo grafo orientato $G(\overline{X})$ un cammino orientato da S a D? La risposta é affermativa. Esiste infatti il cammino orientato:

$$S \to n_2 \to n_1 \to D.$$

Indichiamo con $P = \{(S, n_2), \ (n_2, n_1), \ (n_1, D)\}$ l'insieme degli archi di tale cammino. Possiamo a questo punto modificare il nostro flusso nel modo seguente.

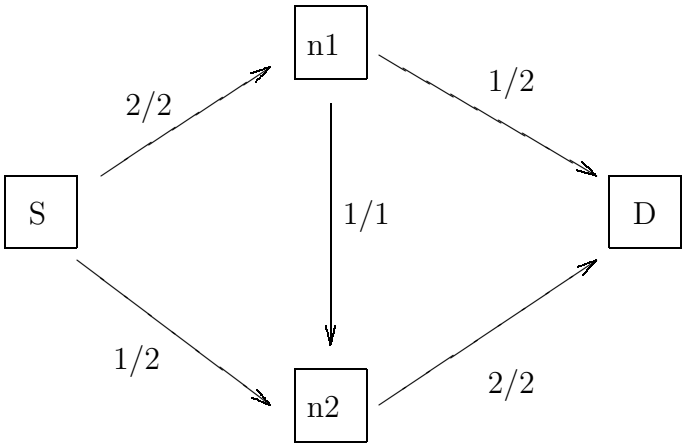


Figura 3.16: La nuova situazione dopo l'incremento del flusso.

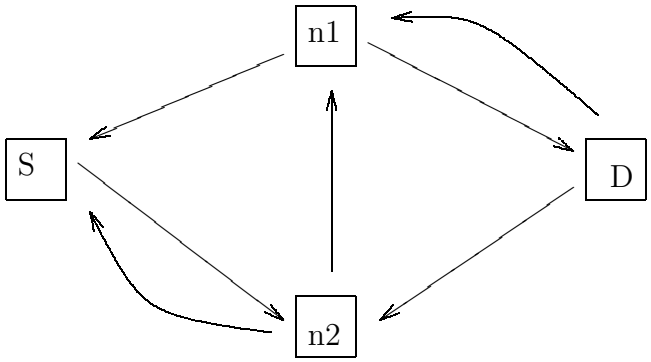


Figura 3.17: Il grafo associato al flusso corrente.

Per ogni arco (i, j) del cammino P si calcoli il seguente valore:

$$\alpha_{ij} = \begin{cases} c_{ij} - \overline{x}_{ij} & \text{se } (i,j) \in A_f(\overline{X}) \cap P \\ \overline{x}_{ji} & \text{se } (i,j) \in A_b(\overline{X}) \cap P \end{cases}$$

e quindi sia

$$\Delta = \min_{(i,j) \in P} \alpha_{ij}$$

il minimo di tali valori. Per gli archi forward il valore α_{ij} rappresenta quanto flusso posso ancora inviare lungo l'arco $(i, j) \in A$ della rete originaria, per gli archi backward il valore α_{ij} rappresenta quanto flusso posso rispedire indietro lungo l'arco $(j, i) \in A$ della rete originaria. Nel nostro caso si ha:

$$(S, n_2) \in A_f(\overline{X}) \rightarrow \ \alpha_{Sn_2} = 2 - 1 = 1 \quad (n_2, n_1) \in A_b(\overline{X}) \rightarrow \ \alpha_{n_2 n_1} = 1$$

$$(n_1, D) \in A_f(\overline{X}) \rightarrow \ \alpha_{n_1 D} = 2 - 1 = 1$$

Il minimo tra questi tre valori é $\Delta = 1$. Ora il flusso viene aggiornato nel modo seguente:

$$\overline{x}_{ij} = \begin{cases} \overline{x}_{ij} + \Delta & \text{se } (i,j) \in A_f(\overline{X}) \cap P \\ \overline{x}_{ij} - \Delta & \text{se } (j,i) \in A_b(\overline{X}) \cap P \\ \overline{x}_{ij} & \text{altrimenti} \end{cases} \tag{3.9}$$

Quindi nel nostro esempio avremo:

$$\overline{x}_{Sn_1} = 2 \quad \overline{x}_{Sn_2} = 1 + 1 = 2 \quad \overline{x}_{n_1 n_2} = 1 - 1 = 0$$

$$\overline{x}_{n_1 D} = 1 + 1 = 2 \quad \overline{x}_{n_2 D} = 2.$$

La nuova situazione é illustrata in Figura 3.18. In pratica sulla rete originaria

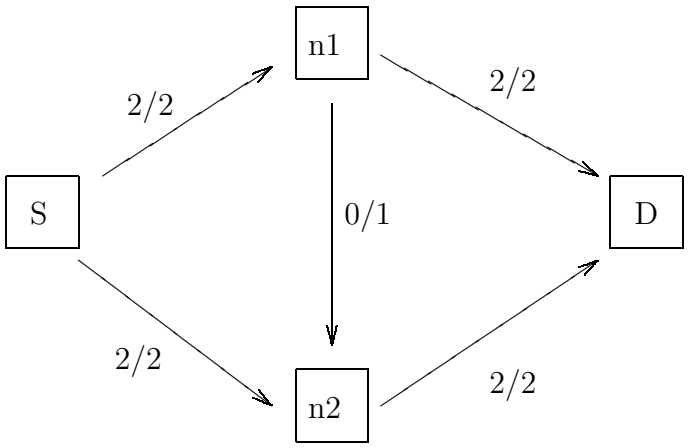


Figura 3.18: La nuova situazione dopo l'aggiornamento del flusso.

incrementiamo di Δ il flusso lungo gli archi corrispondenti ad archi forward del grafo associato al flusso attuale appartenenti al cammino P, rispediamo indietro Δ unitá di flusso lungo gli archi che, cambiati di verso, corrispondono ad archi backward del grafo associato appartenenti al cammino P, e quindi decrementiamo il flusso della stessa quantitá lungo tali archi. Infine, lungo gli archi che non fanno parte del cammino P il flusso rimane invariato. Si puó verificare che il nuovo flusso é ammissibile. In particolare, per come é stato scelto il valore Δ il nuovo flusso lungo ciascun arco é non negativo e non supera la capacitá dell'arco. Inoltre, continuano anche ad essere soddisfatti i vincoli di equilibrio nei nodi intermedi. La quantitá di flusso uscente da S é anch'essa aumentata proprio della quantitá Δ e quindi passa da 3 a 4, il che dimostra che il flusso precedente non era ottimo. Con il nuovo flusso possiamo ripetere quanto visto. Costruiamo il grafo $G(\overline{X})$ associato al nuovo flusso $\overline{X}$. Ora avremo:

$$A_f(\overline{X}) = \{(n_1, n_2)\} \quad A_b(\overline{X}) = \{(n_1, S),\ (n_2, S),\ (D, n_1),\ (D, n_2)\}.$$

Il grafo é illustrato in Figura 3.19. Possiamo ora chiederci se il nuovo flus-

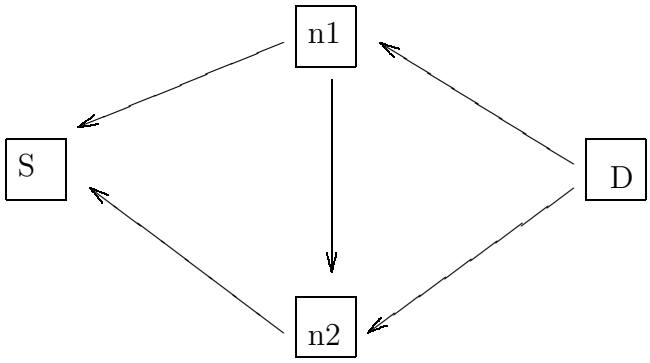


Figura 3.19: Il grafo associato al nuovo flusso.

so é quello ottimo. La risposta la possiamo dare sulla base della seguente fondamentale osservazione.

Osservazione 4 *Dato un flusso $\overline{X}$, esso é ottimo se e solo se nel grafo associato $G(\overline{X})$ non esiste alcun cammino orientato da S a D.*

Dalla Figura 3.19 si puó notare che per il grafo associato al nostro nuovo flusso non esiste alcun cammino orientato dal nodo S al nodo D e quindi il flusso attuale é ottimo. Quanto visto rappresenta la base dell'algoritmo di Ford-Fulkerson. In pratica, ad ogni iterazione di tale algoritmo si eseguono i seguenti passi.

Passo 1 Si costruisca il grafo $G(\overline{X})$ associato al flusso attuale $\overline{X}$.

Passo 2 Se non esiste alcun cammino orientato da S a D in $G(\overline{X})$, allora STOP: il flusso attuale $\overline{X}$ é ottimo. Altrimenti, si individui un tale cammino P e si aggiorni il flusso $\overline{X}$ come indicato in (3.9) e si ritorni al Passo 1.

Se quella sopra é la struttura dell'algoritmo, vediamo ora di entrare nei suoi dettagli.

ALGORITMO DI FORD-FULKERSON

Passo 0 Si parta con un flusso ammissibile $\overline{X}$. Si noti che é sempre possibile partire con il flusso nullo $\overline{X} = 0$.

Passo 1 Si associ alla sorgente S l'etichetta (S, ∞). L'insieme R dei nodi *analizzati* é vuoto, ovvero $R = \emptyset$, mentre l'insieme E dei nodi *etichettati* contiene il solo nodo S, ovvero $E = \{S\}$.

Passo 2 Si verifichi se $E \setminus R \neq \emptyset$, ovvero se vi sono nodi etichettati non ancora analizzati. Se non ne esistono il flusso attuale $\overline{X}$ é ottimo. Altrimenti si selezioni un nodo $i \in E \setminus R$, cioé etichettato, con etichetta (k, Δ), ma non

ancora analizzato e lo si analizzi. Analizzare il nodo i vuol dire compiere le seguenti operazioni. Per ogni nodo $j \notin E$, cioé non ancora etichettato, e tale che $(i,j) \in A(\overline{X})$, si etichetti j con la seguente etichetta:

$$(i, \min(\Delta, c_{ij} - \overline{x}_{ij})) \quad \text{se } (i,j) \in A_f(\overline{X})$$

$$(i, \min(\Delta, \overline{x}_{ji})) \quad \text{se } (i,j) \in A_b(\overline{X})$$

Quindi si ponga:

$$E = E \cup \{j \notin E \ : \ (i,j) \in A(\overline{X})\} \qquad R = R \cup \{i\}.$$

Se $D \in E$, cioé se la destinazione é stata etichettata, si vada al Passo 3, altrimenti si ripeta il Passo 2.

Passo 3 Si ricostruisca un cammino orientato da S a D in $G(\overline{X})$ procedendo a ritroso da D verso S ed usando le prime componenti delle etichette. Piú precisamente, si cominci col considerare l'etichetta (q_r, Δ) di D. Allora nel cammino orientato da S a D il nodo D é preceduto da q_r. Per conoscere da quale nodo é preceduto q_r se ne consideri la prima componente dell'etichetta, indicata qui con q_{r-1}. Si ripeta quindi la stessa procedura con q_{r-1}. Ci si arresta non appena si arriva ad un nodo q_1 la cui prima componente dell'etichetta é il nodo sorgente S. A questo punto il cammino

$$S \to q_1 \to \cdots \to q_{r-1} \to q_r \to D$$

con insieme di archi

$$P = \{(S, q_1), \ldots, (q_{r-1}, q_r), (q_r, D)\}$$

é un cammino orientato da S a D in $G(\overline{X})$. Ora possiamo aggiornare $\overline{X}$ come indicato in (3.9), dove il valore Δ è dato dalla seconda componente dell'etichetta del nodo D, e ritornare al Passo 1.

Quando l'algoritmo termina abbiamo giá anche una soluzione del problema di taglio minimo, come dimostra il seguente teormema.

Teorema 3 *Se si pone $U = E$, dove E é l'insieme dei nodi etichettati al momento della terminazione dell'algoritmo, si ha anche che il taglio $\mathcal{S}_U$ indotto da U é soluzione ottima del problema di taglio minimo (3.8).*

Dimostrazione Per prima cosa si noti che al momento della terminazione dell'algoritmo si ha $S \in E$ (il nodo S viene sempre etichettato al Passo 1) e $D \notin E$ (altrimenti dovremmo andare al Passo 3 ed aggiornare il flusso attuale). Quindi l'insieme E induce effettivamente un taglio. Vediamo ora qual é il valore di questo taglio. Se riusciamo a dimostrare che esso coincide con il valore del flusso uscente da S, avendo giá osservato che il costo di ogni taglio é non inferiore al valore del flusso massimo, possiamo concludere che esso é il taglio a costo minimo. Per prima cosa valutiamo il flusso uscente da S ed entrante in D.

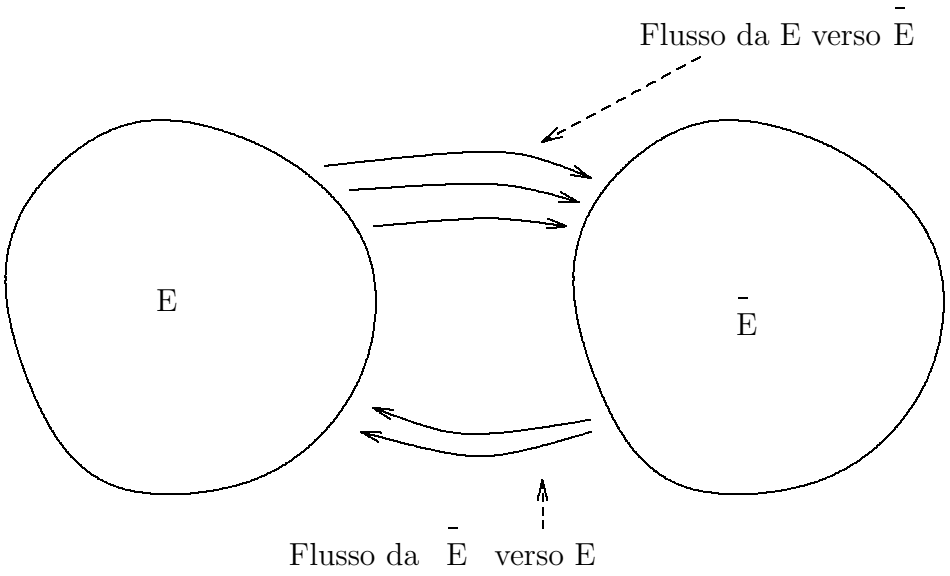


Figura 3.20: Il flusso totale dalla sorgente alla destinazione è pari alla differenza tra il flusso totale da E verso $\overline{E}$ e il flusso totale da $\overline{E}$ verso E.

Esso coincide con tutto il flusso che dai nodi in E viene spostato verso i nodi nel complemento $\overline{E} = V \setminus E$ di E meno il flusso che va in senso opposto, cioé quello dai nodi nel complemento $\overline{E}$ verso i nodi in E (si veda la Figura 3.20). In formule il flusso uscente da S ed entrante in D é quindi pari a:

$$\sum_{(i,j)\ :\ (i,j)\in A,\ i\in E,\ j\in\overline{E}} \overline{x}_{ij} - \sum_{(j,i)\ :\ (j,i)\in A,\ i\in E,\ j\in\overline{E}} \overline{x}_{ji}. \tag{3.10}$$

Ma vediamo ora come devono essere i valori $\overline{x}_{ij}$ per $(i,j) \in A,\ i \in E,\ j \in \overline{E}$ e i valori $\overline{x}_{ji}$ per $(j,i) \in A,\ i \in E,\ j \in \overline{E}$. Si deve avere che

$$\forall\ (i,j) \in A,\ i \in E,\ j \in \overline{E} \quad \overline{x}_{ij} = c_{ij}. \tag{3.11}$$

Infatti, per assurdo si supponga che esista un $(i_1, j_1) \in A,\ i_1 \in E,\ j_1 \in \overline{E}$ con $\overline{x}_{i_1 j_1} < c_{i_1 j_1}$. In tal caso $(i_1, j_1) \in A_f(\overline{X})$ e quindi al Passo 2 dovremmo assegnare un'etichetta a j_1, il che contraddice l'ipotesi che $j_1 \notin E$. Analogamente, si deve avere che

$$\forall\ (j,i) \in A,\ i \in E,\ j \in \overline{E} \quad \overline{x}_{ji} = 0. \tag{3.12}$$

Infatti, per assurdo si supponga che esista un $(j_1, i_1) \in A,\ i_1 \in E,\ j_1 \in \overline{E}$ con $\overline{x}_{j_1 i_1} > 0$. In tal caso $(i_1, j_1) \in A_b(\overline{X})$ e quindi al Passo 2 dovremmo assegnare un'etichetta a j_1, il che contraddice ancora l'ipotesi che $j_1 \notin E$. Sostituendo (3.11) e (3.12) in (3.10) si ottiene che il valore del flusso é pari a

$$\sum_{(i,j)\ :\ (i,j)\in A,\ i\in E,\ j\in\overline{E}} c_{ij} = C(\mathcal{S}_E),$$

Tabella 3.1:

	E	R	S	n_1	n_2	n_3	n_4	D
Passo 1	S	$\emptyset$	(S,∞)	-	-	-	-	-
Passo 2	S, n_1, n_2	S	(S,∞)	$(S,3)$	$(S,2)$	-	-	-
Passo 2	S, n_1, n_2, n_3, n_4	S, n_1	(S,∞)	$(S,3)$	$(S,2)$	$(n_1,1)$	$(n_1,3)$	-
Passo 2	S, n_1, n_2, n_3, n_4	S, n_1, n_2	(S,∞)	$(S,3)$	$(S,2)$	$(n_1,1)$	$(n_1,3)$	-
Passo 2	S, n_1, n_2, n_3, n_4, D	S, n_1, n_2, n_3	(S,∞)	$(S,3)$	$(S,2)$	$(n_1,1)$	$(n_1,3)$	$(n_3,1)$

cioé é pari al costo del taglio indotto da E, il che conclude la dimostrazione.

Dobbiamo ancora precisare la complessitá dell'algoritmo. Se si considera la rete $G = (V, A)$ con

$$V = \{S, 1, 2, D\} \quad A = \{(S,1), (S,2), (1,2), (1,D), (2,D)\}$$

e capacità degli archi:

Archi	$(S,1)$	$(S,2)$	$(1,2)$	$(1,D)$	$(2,D)$
Capacità	M	M	1	M	M

posso scegliere come percorso aumentante alternativamente $S \to 1 \to 2 \to D$ e $S \to 2 \to 1 \to D$ e in tal caso incremento di una sola unità il flusso ad ogni iterazione. Il numero di iterazioni è quindi pari a $2M$ (esponenziale ripsetto alla dimensione dell'istanza). Si puó però dimostrare in generale che se la scelta del nodi $i \in E \setminus R$ da analizzare viene fatta secondo una disciplina FIFO, cioé i nodi vengono analizzati nello stesso ordine in cui vengono etichettati, allora il numero di operazioni richieste dall'algoritmo é pari a $O(\mid A \mid\mid V \mid^2)$. L'algoritmo ha quindi complessitá polinomiale. Va anche sottolineato come questa non sia la migliore complessitá possibile per questo problema. Esistono algoritmi piú sofisticati con una complessitá pari a $O(\mid V \mid^3)$.

3.4.4 Un esempio di applicazione del problema di massimo flusso

Vediamo ora di risolvere il problema di flusso massimo in Figura 3.14 utilizzando l'algoritmo di Ford-Fulkerson. Cominciamo con il flusso nullo $\overline{X}$. In tal caso si ha che il grafo associato a tale flusso nullo coincide con il grafo originario, ovvero $G(\overline{X}) \equiv G$. Il primo ciclo dell'algoritmo é descritto nella Tabella 3.1. Il valore Δ é pari a 1 ed il cammino orientato in $G(\overline{X})$ da S a D é il seguente:

$$S \to n_1 \to n_3 \to D.$$

Ció porta al seguente aggiornamento del flusso:

$$\overline{x}_{Sn_1} = 1 \;\; \overline{x}_{Sn_2} = 0 \;\; \overline{x}_{n_1n_3} = 1 \;\; \overline{x}_{n_1n_4} = 0$$

$$\overline{x}_{n_2n_3} = 0 \;\; \overline{x}_{n_2n_4} = 0 \;\; \overline{x}_{n_3D} = 1 \;\; \overline{x}_{n_4D} = 0$$

Tabella 3.2:

	E	R	S	n_1	n_2	n_3	n_4	D
Passo 1	S	$\emptyset$	(S,∞)	-	-	-	-	-
Passo 2	S, n_1, n_2	S	(S,∞)	$(S,2)$	$(S,2)$	-	-	-
Passo 2	S, n_1, n_2, n_4	S, n_1	(S,∞)	$(S,2)$	$(S,2)$	-	$(n_1,2)$	-
Passo 2	S, n_1, n_2, n_3, n_4	S, n_1, n_2	(S,∞)	$(S,2)$	$(S,2)$	$(n_2,1)$	$(n_1,2)$	-
Passo 2	S, n_1, n_2, n_3, n_4, D	S, n_1, n_2, n_4	(S,∞)	$(S,2)$	$(S,2)$	$(n_2,1)$	$(n_1,2)$	$(n_4,2)$

Il grafo $G(\overline{X})$ associato al nuovo flusso é illustrato in Figura 3.21. A questo

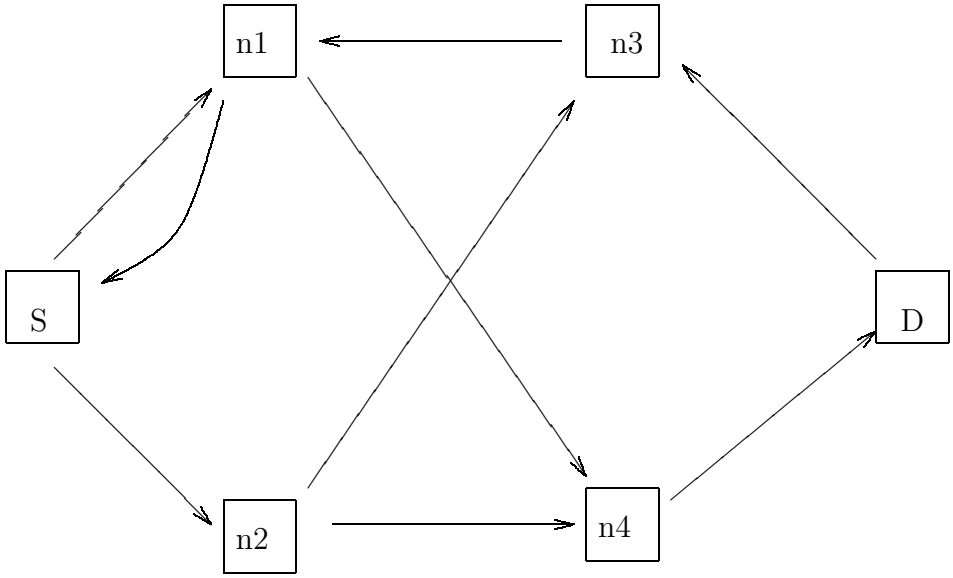


Figura 3.21: Il grafo associato al nuovo flusso.

punto si ripete la procedura con i passi indicati in Tabella 3.2. Il valore Δ é pari a 2 ed il cammino orientato in $G(\overline{X})$ da S a D é il seguente:

$$S \to n_1 \to n_4 \to D.$$

Ció porta al seguente aggiornamento del flusso:

$$\overline{x}_{Sn_1} = 3 \;\; \overline{x}_{Sn_2} = 0 \;\; \overline{x}_{n_1n_3} = 1 \;\; \overline{x}_{n_1n_4} = 2$$

$$\overline{x}_{n_2n_3} = 0 \;\; \overline{x}_{n_2n_4} = 0 \;\; \overline{x}_{n_3D} = 1 \;\; \overline{x}_{n_4D} = 2$$

Il grafo $G(\overline{X})$ associato al nuovo flusso é illustrato in Figura 3.22. A questo

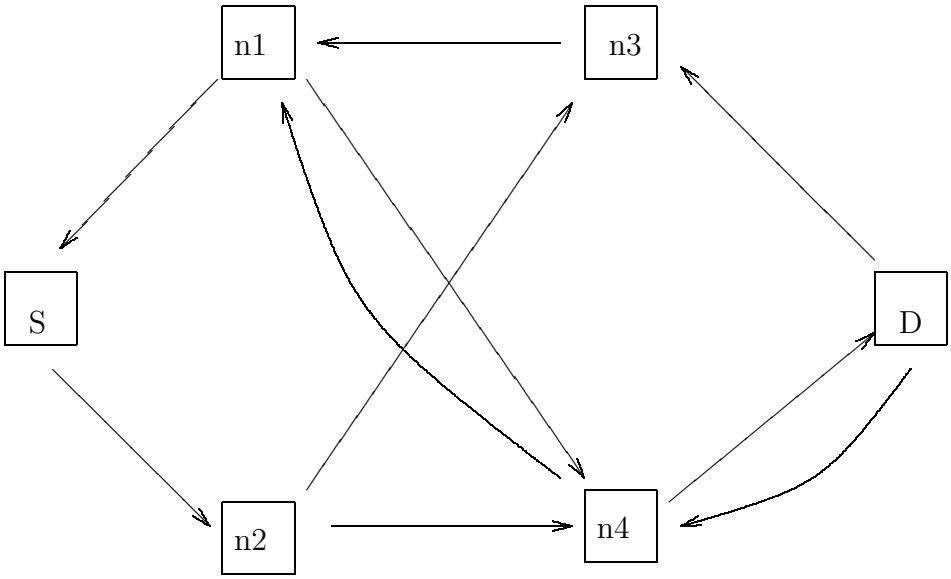


Figura 3.22:

Tabella 3.3:

	E	R	S	n_1	n_2	n_3	n_4	D
Passo 1	S	$\emptyset$	(S,∞)	-	-	-	-	-
Passo 2	S, n_2	S	(S,∞)	-	$(S,2)$	-	-	-
Passo 2	S, n_2, n_3, n_4	S, n_2	(S,∞)	-	$(S,2)$	$(n_2,1)$	$(n_2,1)$	-
Passo 2	S, n_2, n_3, n_4, n_1	S, n_2, n_3	(S,∞)	$(n_3,1)$	$(S,2)$	$(n_2,1)$	$(n_2,1)$	-
Passo 2	S, n_2, n_3, n_4, n_1, D	S, n_2, n_3, n_4	(S,∞)	$(n_3,1)$	$(S,2)$	$(n_2,1)$	$(n_2,1)$	$(n_4,1)$

punto si ripete la procedura con i passi indicati in Tabella 3.3. Il valore Δ é pari a 1 ed il cammino orientato in $G(\overline{X})$ da S a D é il seguente:

$$S \to n_2 \to n_4 \to D.$$

Ció porta al seguente aggiornamento del flusso:

$$\overline{x}_{Sn_1} = 3 \;\; \overline{x}_{Sn_2} = 1 \;\; \overline{x}_{n_1n_3} = 1 \;\; \overline{x}_{n_1n_4} = 2$$

$$\overline{x}_{n_2n_3} = 0 \;\; \overline{x}_{n_2n_4} = 1 \;\; \overline{x}_{n_3D} = 1 \;\; \overline{x}_{n_4D} = 3$$

Il grafo $G(\overline{X})$ associato al nuovo flusso é illustrato in Figura 3.23. A questo punto si ripete la procedura con i passi indicati in Tabella 3.4. Il valore Δ é pari a 1 ed il cammino orientato in $G(\overline{X})$ da S a D é il seguente:

$$S \to n_2 \to n_3 \to n_1 \to n_4 \to D.$$

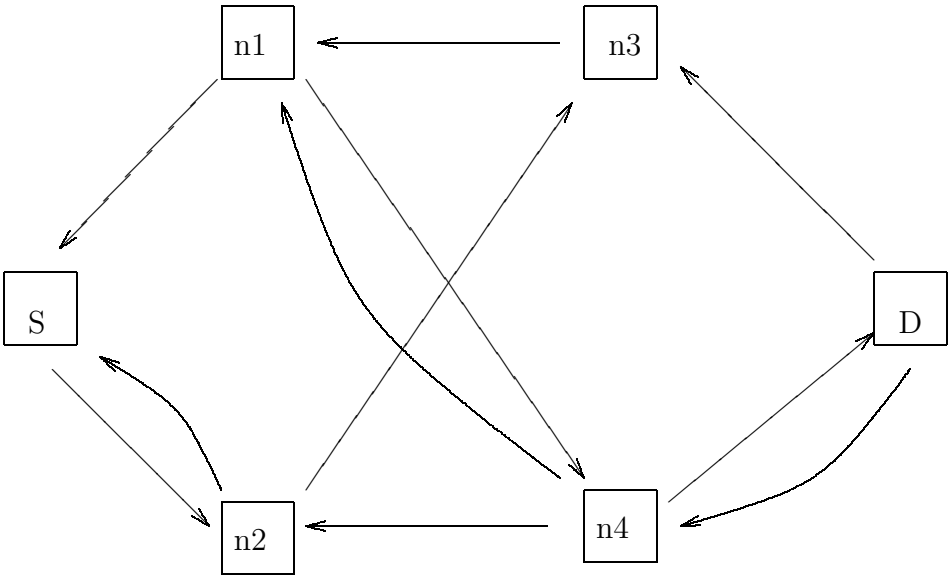


Figura 3.23:

Ció porta al seguente aggiornamento del flusso:

$$\overline{x}_{Sn_1} = 3 \;\; \overline{x}_{Sn_2} = 2 \;\; \overline{x}_{n_1n_3} = 0 \;\; \overline{x}_{n_1n_4} = 3$$

$$\overline{x}_{n_2n_3} = 1 \;\; \overline{x}_{n_2n_4} = 1 \;\; \overline{x}_{n_3D} = 1 \;\; \overline{x}_{n_4D} = 4$$

Il grafo $G(\overline{X})$ associato al nuovo flusso é illustrato in Figura 3.24. A questo punto si ripete la procedura con i passi indicati in Tabella 3.5. Arriviamo a $E \setminus R = \emptyset$ e quindi possiamo fermarci ed affermare che il flusso attuale é quello ottimo. Non solo, sappiamo che il taglio indotto dal sottinsieme di nodi $E = \{S\}$ é quello a costo minimo. Si puó infatti verificare che il valore del flusso uscente da S e il costo del taglio indotto da $\{S\}$ sono entrambi pari a 5.

Tabella 3.4:

	E	R	S	n_1	n_2	n_3	n_4	D
Passo 1	S	$\emptyset$	(S,∞)	-	-	-	-	-
Passo 2	S, n_2	S	(S,∞)	-	$(S,1)$	-	-	-
Passo 2	S, n_2, n_3	S, n_2	(S,∞)	-	$(S,1)$	$(n_2,1)$	-	-
Passo 2	S, n_2, n_3, n_1	S, n_2, n_3	(S,∞)	$(n_3,1)$	$(S,1)$	$(n_2,1)$	-	-
Passo 2	S, n_2, n_3, n_1, n_4	S, n_2, n_3, n_1	(S,∞)	$(n_3,1)$	$(S,1)$	$(n_2,1)$	$(n_1,1)$	-
Passo 2	S, n_2, n_3, n_1, n_4, D	S, n_2, n_3, n_1, n_4	(S,∞)	$(n_3,1)$	$(S,1)$	$(n_2,1)$	$(n_1,1)$	$(n_4,1)$

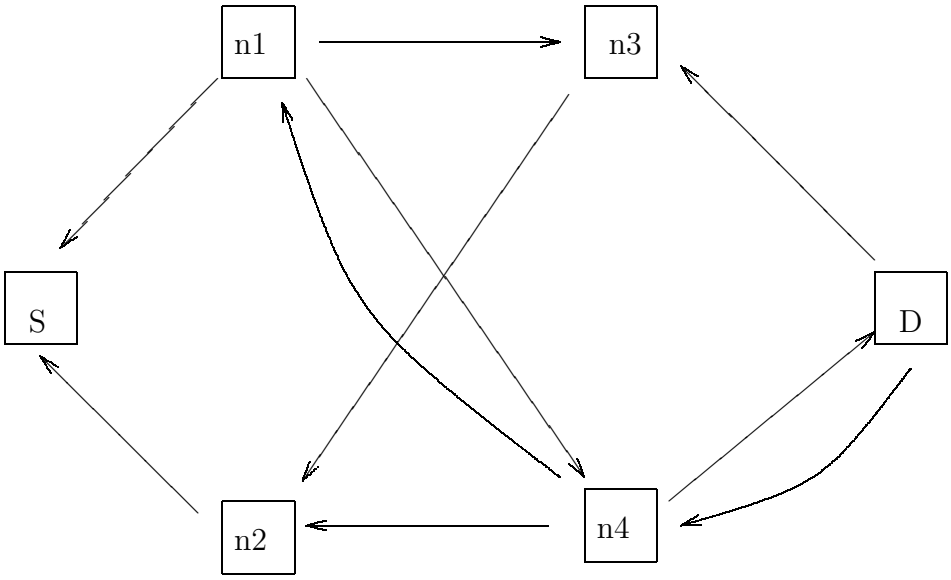


Figura 3.24:

Tabella 3.5:

	E	R	S	n_1	n_2	n_3	n_4	D
Passo 1	S	$\emptyset$	(S, ∞)	-	-	-	-	-
Passo 2	S	S	(S, ∞)	-	-	-	-	-

3.5 Il problema di matching

Dato un grafo non orientato $G = (V, A)$ un matching su tale grafo é un sottinsieme $M \subseteq A$ dell'insieme di archi A tale che non ci sono in M coppie di archi adiacenti, ovvero con un nodo in comune. Se associamo ad ogni arco $e \in A$ un peso $w_e > 0$ possiamo associare un peso anche a un matching M pari alla somma dei pesi degli archi in M, cioé:

$$w(M) = \sum_{e \in M} w_e.$$

Nel problema di *matching pesato* si vuole determinare tra tutti i possibili matching in un grafo quello di peso massimo, cioé si vuole risolvere il seguente problema:

$$\max_{M \subseteq A \text{ é un matching}} w(M).$$

Esempio 12 *Nel grafo in Figura 3.25, in cui i pesi degli archi sono riportati di fianco ad essi, gli archi* (a, b), (c, d), (e, f) *formano un matching con peso pari a 11, mentre gli archi* (a, f), (e, d), (b, c) *formano un matching con peso pari a 10. Nel problema di matching pesato si cerca tra tutti i possibili matching quello di peso massimo.*

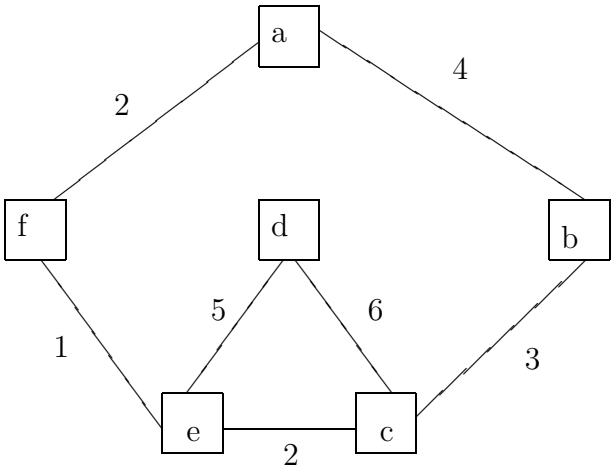


Figura 3.25:

Un caso particolare si ha quando $w_e = 1 \ \forall \ e \in A$. In tal caso il peso di un matching coincide con la sua cardinalitá $\mid M \mid$ e si parla di problema di *matching di cardinalitá massima*.

3.5.1 Un modello matematico per il problema di matching

Ad ogni arco e associamo una variabile binaria x_e definita nel modo seguente:

$$x_e = \begin{cases} 0 & \text{se } e \text{ non viene inserito nel matching} \\ 1 & \text{altrimenti} \end{cases}$$

Per garantire che le variabili x_e con valore pari a 1 formino effettivamente un matching introduciamo dei vincoli. Per avere un matching su ogni nodo del grafo deve incidere al massimo un arco del matching. Indichiamo per ogni nodo $i \in V$ il seguente insieme degli archi incidenti su di esso:

$$\delta(i) = \{e \in A \ : \ e \text{ incide su } i\}.$$

Se vogliamo che al massimo un nodo del matching incida su ogni nodo i del grafo, dovremo imporre i seguenti vincoli:

$$\sum_{e \in \delta(i)} x_e \leq 1 \quad \forall \ i \in V.$$

Infine l'obiettivo del problema é il peso totale del matching che é dato dalla seguente espressione:

$$\sum_{e \in A} w_e x_e.$$

Riassumendo, il modello del problema di matching é il seguente:

$$\begin{array}{ll} \max & \sum_{e \in A} w_e x_e \\ & \sum_{e \in \delta(i)} x_e \leq 1 \quad \forall \ i \in V \\ & x_e \in \{0,1\} \quad \forall \ e \in A \end{array}$$

Per il nostro esempio abbiamo il seguente modello matematico:

$$\begin{array}{ll} \max & 4x_{ab} + 2x_{af} + 3x_{bc} + 6x_{cd} + 2x_{ce} + 5x_{de} + x_{ef} \\ & x_{ab} + x_{af} \leq 1 \\ & x_{ab} + x_{bc} \leq 1 \\ & x_{bc} + x_{cd} + x_{ce} \leq 1 \\ & x_{cd} + x_{de} \leq 1 \\ & x_{ce} + x_{de} + x_{ef} \leq 1 \\ & x_{af} + x_{ef} \leq 1 \\ & x_{ab}, x_{af}, x_{bc}, x_{cd}, x_{ce}, x_{de}, x_{ef} \in \{0,1\} \end{array}$$

Se introduciamo la seguente notazione:

- $\mathbf{x}$ il vettore di ordine $\mid A \mid$ le cui componenti sono le variabili x_e, $e \in A$;
- $\mathbf{w}$ il vettore di ordine $\mid A \mid$ le cui componenti sono i pesi w_e, $e \in A$;

- **C** la matrice dei vincoli di ordine $\mid V \mid \times \mid A \mid$ (che coincide con la *matrice di incidenza nodo-arco del grafo*);
- **0** e **1** rispettivamente il vettore le cui componenti sono tutte pari a 0 e quello le cui componenti sono tutte pari a 1;
- **I** la matrice identica;

il modello matematico del problema di matching in forma matriciale è il seguente:

$$\begin{aligned} \max \quad & \mathbf{wx} \\ & \mathbf{Cx} \leq \mathbf{1} \\ & \mathbf{Ix} \leq \mathbf{1} \\ & \mathbf{x} \geq \mathbf{0} \quad \text{intero} \end{aligned}$$

dove:

- $\mathbf{wx} \leftrightarrow \sum_{e \in A} w_e x_e$;
- $\mathbf{Cx} \leq \mathbf{1} \leftrightarrow \sum_{e \in \delta(i)} x_e \leq 1 \quad \forall\, i \in V$;
- $\mathbf{Ix} \leq \mathbf{1}$ e $\mathbf{x} \geq \mathbf{0}$ intero $\leftrightarrow x_e \in \{0, 1\} \quad \forall\, e \in A$

Nell'esempio abbiamo il vettore di pesi **w**:

$$(4 \;\; 2 \;\; 3 \;\; 6 \;\; 2 \;\; 5 \;\; 1)$$

La matrice **C**:

	x_{ab}	x_{af}	x_{bc}	x_{cd}	x_{ce}	x_{de}	x_{ef}
a	1	1	0	0	0	0	0
b	1	0	1	0	0	0	0
c	0	0	1	1	1	0	0
d	0	0	0	1	0	1	0
e	0	0	0	0	1	1	1
f	0	1	0	0	0	0	1

3.5.2 Complessità

Il problema di matching appartiene alla classe P, ovvero esiste una procedura di complessitá polinomiale che lo risolve. Nel caso di grafi bipartiti, dove **C** è TU, possiamo sfruttare il solito risultato sulle matrici TU. *Non* possiamo invece dimostrare questo per grafi generici utilizzando i risultati visti sulle matrici TU (per grafi non orientati generici la matrice di incidenza nodo-arco **C** non è necessariamente TU). Per grafi generici i vincoli già introdotti non sono sufficienti per definire la chiusura convessa, occorre introdurre altri vincoli. Dato

$U \subseteq V$, indichiamo con $E(U)$ l'insieme di tutti gli archi del grafo con entrambi gli estremi in U. I vincoli aggiuntivi sono i seguenti:

$$\sum_{e \in E(U)} x_e \leq \left\lfloor \frac{|\, U \,|}{2} \right\rfloor \quad \forall\, U \subseteq V,\ |\, U \,| \ \text{ dispari}$$

Nel nostro esempio abbiamo:

$$\begin{aligned} U = \{a,b,c\} &\ \rightarrow \ x_{ab} + x_{bc} \leq 1 \\ U = \{a,b,d\} &\ \rightarrow \ x_{ab} \leq 1 \\ U = \{a,b,e\} &\ \rightarrow \ x_{ab} \leq 1 \\ U = \{a,b,f\} &\ \rightarrow \ x_{ab} + x_{af} \leq 1 \\ U = \{a,c,d\} &\ \rightarrow \ x_{cd} \leq 1 \\ U = \{a,c,e\} &\ \rightarrow \ x_{ce} \leq 1 \\ U = \{a,c,f\} &\ \rightarrow \ 0 \leq 1 \end{aligned}$$

eccetera ...

Quindi, il problema (3.4) per problemi di matching su grafi generici è il seguente:

$$\begin{array}{lll} \max & \sum_{e \in A} w_e x_e & \\ & \sum_{e \in \delta(i)} x_e \leq 1 & \forall\, i \in V \\ & \sum_{e \in E(U)} x_e \leq \left\lfloor \frac{|U|}{2} \right\rfloor & \forall\, U \subseteq V,\ |\, U \,| \ \text{ dispari} \\ & x_e \geq 0 & \forall\, e \in A \end{array}$$

Il numero dei vincoli aggiuntivi è esponenziale rispetto alla dimensione del problema di matching. Tuttavia tale problema è risolvibile in tempo polinomiale in quanto, dato un generico vettore $\mathbf{x} \in R^{|A|},\ \mathbf{x} \geq \mathbf{0}$, è possibile stabilire in tempo polinomiale se questo soddisfa o meno tutti i vincoli aggiuntivi (si ricordi quanto stabilito nell'Osservazione 1).

3.5.3 Il problema di matching di cardinalitá massima su grafi bipartiti

La nostra attenzione sará ora rivolta ai casi in cui tutti i pesi sono uguali a 1 (matching di cardinalitá massima) e il grafo $G = (V, A)$ é un grafo bipartito con le due classi di bipartizione V_1 e V_2. Supponiamo di avere giá a disposizione un matching M. Scegliere $M = \emptyset$ é sempre possibile ma la seguente semplice procedura ci consente di determinare rapidamente un matching migliore.

Inizializzazione Si ponga $M = \emptyset$.

1. Se esiste un arco $\overline{e} \in A \setminus M$ che non sia adiacente ad alcun arco in M, allora porre $M = M \cup \{\overline{e}\}$ e ripetere il passo 1. Altrimenti: STOP.

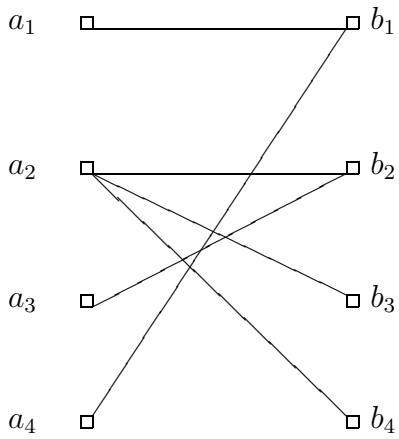


Figura 3.26:

Illustriamo ora questa semplice procedura sul grafo bipartito in Figura 3.26.

Esempio 13 *Sia dato il grafo bipartito in Figura 3.26. La procedura viene illustrata in Figura 3.27 per il nostro esempio. Essa seleziona dapprima l'arco*

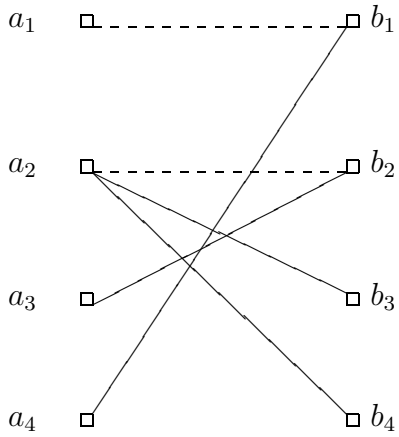


Figura 3.27:

(a_1, b_1) *(*$M = \{(a_1, b_1)\}$*). Poi seleziona l'arco* (a_2, b_2) *che non è adiacente ad* (a_1, b_1) *(*$M = \{(a_1, b_1),\ (a_2, b_2)\}$*). A questo punto non restano piú archi non adiacenti ad archi in* M *e la procedura si arresta.*

L'insieme M individuato con la procedura vista sopra ci serve come punto di partenza per l'algoritmo che porterá ad individuare un matching di cardinalitá massima. L'algoritmo per la risoluzione del problema è il seguente.

0. Tutti i vertici del grafo sono non etichettati.

1. Se non c'è alcun vertice del grafo in V_1 che soddisfa le seguenti proprietá

- è non etichettato;
- su di esso non incide alcun arco in M,

allora: STOP. Altrimenti si seleziona un vertice del grafo in V_1 con tali proprietá e gli si assegna etichetta $(E, -)$. Si inizializzi l'insieme R dei vertici *analizzati* con $R = \emptyset$.

2. Se tutti i vertici etichettati sono stati analizzati, ritornare al Passo 1.; altrimenti selezionare un vertice k etichettato ma non analizzato ed analizzarlo, cioé si ponga $R = R \cup \{k\}$. Analizzare un vertice vuol dire compiere le seguenti operazioni:

a) Se la prima componente dell'etichetta di k è una E, allora si assegna un'etichetta (O, k) a tutti i vertici del grafo adiacenti a k e non ancora etichettati; quindi si ripete il Passo 2.

b) Se la prima componente dell'etichetta di k è una O, allora sono possibili due casi

Caso 1 C'è un arco marcato incidente su k: in tal caso si assegna l'etichetta (E, k) al vertice unito a k dall'arco in M e si ripete il Passo 2.;

Caso 2 Non ci sono archi in M incidenti su k. In tal caso si va al Passo 3.

3. Utilizzando la seconda componente delle etichette risalire dal vertice k in cui ci si è bloccati al Passo 2. fino al vertice s con seconda componente pari a $-$. In questo modo si è individuato un cammino da s a k che inizia con un arco non appartenente a M, prosegue alternando archi in M e archi non appartenenti a M, e termina in k con un arco non appartenente a M. A questo punto si aggiorna M invertendo l'appartenenza e non appartenenza a M per i soli archi lungo tale cammino. Quindi, tutti gli archi non appartenenti a M lungo il cammino vengono inseriti in M e viceversa. Infine si cancellano tutte le etichette ripartendo dal Passo 1.

Si puó dimostrare che quando tale algoritmo si arresta, l'insieme degli archi in M coincide con un matching di cardinalitá massima. La procedura ora descritta verrá illustrata sul nostro esempio

Esempio 14 *Partendo dall'insieme M individuato con la procedura vista in precedenza, i vertici in V_1 che non sono toccati da archi in M sono a_3, a_4. Al Passo 1. selezioniamo, arbitrariamente, il vertice a_3 e gli assegniamo etichetta $(E, -)$. Passiamo quindi al Passo 2. L'unico vertice etichettato e non analizzato è il vertice a_3 e quindi selezioniamo questo per analizzarlo. Essendo un vertice con prima componente pari a E l'analisi consisterá nell'etichettare con l'etichetta (O, a_3) tutti i vertici ad esso adiacenti non ancora etichettati . In questo caso verrá etichettato l'unico vertice adiacente ad a_3, cioè b_2. A questo*

punto dobbiamo ripetere il Passo 2. Abbiamo ancora un solo vertice etichettato e non analizzato e cioè b_2 (a_3 è etichettato ma è giá stato analizzato). Quindi si analizza b_2. Il vertice b_2 ha prima componente dell'etichetta O e su di esso incide un arco in M (l'arco (b_2, a_2)), quindi dobbiamo assegnare l'etichetta (E, b_2) al vertice unito a b_2 tramite l'arco marcato, ovvero il vertice a_2. A questo punto ripetiamo il Passo 2. Abbiamo di nuovo un solo vertice etichettato e non analizzato, il vertice a_2. Analizziamo tale vertice: ha prima componente E e quindi dobbiamo etichettare con (O, a_2) tutti i vertici ad esso adiacenti non ancora etichettati. Gli unici vertici adiacenti ad a_2 non ancora etichettati sono b_3 e b_4. A questo punto ripeto il Passo 2. Gli unici vertici etichettati e non analizzati sono b_3 e b_4. Scelgo di analizzare b_3. Ha prima componente O e nessun arco in M incide su di esso, quindi devo andare al Passo 3. Utilizzando le seconde componenti delle etichette ricostruisco un cammino da un vertice con seconda componente $-$ fino al vertice b_3. Dalla seconda componente dell'etichetta di b_3 vedo che nel cammino b_3 è preceduto da a_2; dalla seconda componente per a_2 vedo che a_2 è preceduto da b_2; dalla seconda componente di b_2 vedo che è preceduto da a_3; infine la seconda componente di a_3 è pari a $-$. Quindi il cammino cercato è $a_3 \to b_2 \to a_2 \to b_3$ composto dagli archi $(a_3, b_2), (b_2, a_2), (a_2, b_3)$. Tra questi archi (a_3, b_2) e (a_2, b_3) non sono in M e vengono quindi spostati in M, mentre (b_2, a_2) è in M e viene quindi rimosso da M. Il nuovo insieme M é quindi $M = \{(a_1, b_1),\ (a_3, b_2),\ (a_2, b_3)\}$. Completata questa operazione, si cancellano tutte le etichette e si ripete tutto a partire dal Passo 1. Ora vi

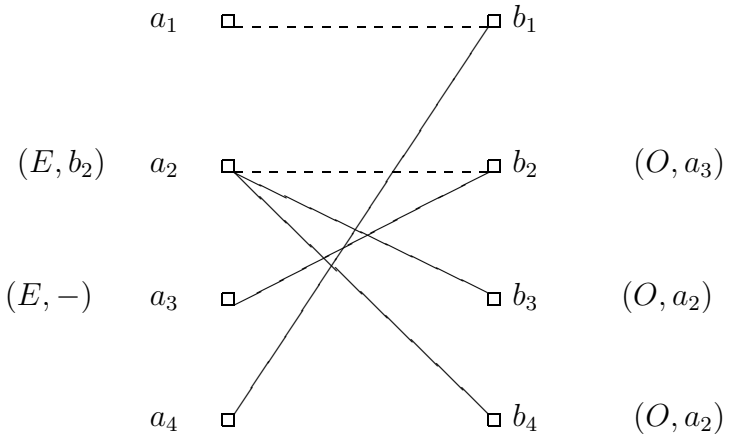


Figura 3.28:

è un solo vertice in V_1 su cui non incide nessun arco nel nuovo insieme M: a_4 (vedi Figura 3.29). Seleziono a_4 e gli assegno etichetta $(E, -)$. Al Passo 2. analizzo a_4 assegnando etichetta (O, a_4) all'unico vertice ad esso adiacente, il vertice b_1. Ripeto il Passo 2. analizzando b_1 ed assegnando quindi etichetta (E, b_1) al vertice a_1 unito a b_1 da un arco in M. Ripeto il Passo 2. analizzando a_1. Poichè a_1 non ha vertici adiacenti non ancora etichettati la sua analisi è

immediatamente conclusa. A questo punto non ho piú vertici etichettati e non analizzati e l'algoritmo mi prescrive di tornare al Passo 1. a cercare un vertice non ancora etichettato su cui non incidano archi marcati. Ma anche di questi non ve ne sono piú. A questo punto l'algoritmo prescrive di arrestarsi. Siamo

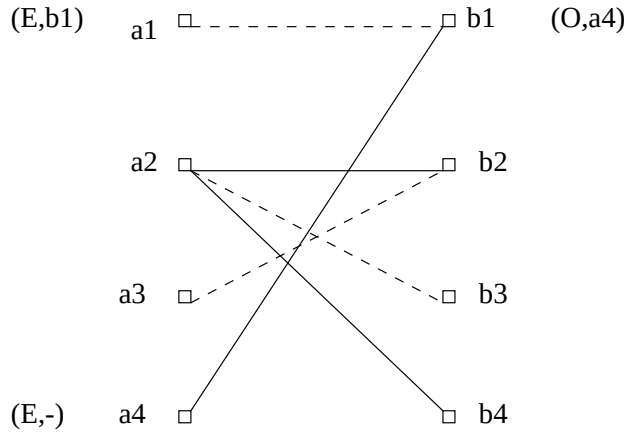


Figura 3.29:

quindi arrivati alla determinazione di un matching M di cardinalitá massima.

Si puó dimostrare che questo algoritmo richiede un numero di operazioni $O(\min(|\ V_1\ |, |\ V_2\ |)\ |\ A\ |)$ e ha quindi complessitá polinomiale. Va sottolineato che non é il meglio che si possa ottenere per questo problema. Esiste infatti anche un altro algoritmo, che non vedremo, la cui complessitá é pari a $O(|\ V\ |^{1/2}|\ A\ |)$.

3.6 Problema del trasporto

Supponiamo di avere m *depositi* in cui é immagazzinato un prodotto e n negozi che richiedono tale prodotto. Nel deposito i é immagazzinata la quantitá a_i di prodotto. Nel negozio j si richiede la quantitá b_j di prodotto. É noto che il costo di trasporto di un'unitá di prodotto dal deposito i al negozio j é pari a c_{ij}. Il problema del trasporto consiste nel determinare quale quantitá di prodotto inviare da ciascun deposito verso ciascun negozio in modo tale da minimizzare il costo complessivo di trasporto, rispettando i vincoli sulle quantitá di prodotto presenti in ciascun deposito e quelli di richieste di ciascun negozio. Si suppone che la quantitá totale immagazzinata in tutti i depositi sia pari alla quantitá totale richiesta da tutti i magazzini, ovvero

$$\sum_{i=1}^{m} a_i = \sum_{j=1}^{n} b_j. \tag{3.13}$$

Non si tratta comunque di un'ipotesi restrittiva dal momento che ci si puó sempre ricondurre ad essa. Infatti, si supponga che

$$\sum_{i=1}^{m} a_i > \sum_{j=1}^{n} b_j.$$

cioé nei depositi vi sia piú prodotto di quanto effettivamente richiesto dai negozi. Per soddisfare l'ipotesi (3.13) basta aggiungere un negozio fittizio $n+1$ con

$$b_{n+1} = \sum_{i=1}^{m} a_i - \sum_{j=1}^{n} b_j,$$

e con $c_{i,n+1} = 0$ per ogni $i,\ i = 1, \ldots, m$, cioé il costo del trasporto verso il negozio fittizio é pari a 0. La quantitá di prodotto che un deposito invia a un negozio fittizio resta in realtá immagazzinata nel deposito. Analogamente, si supponga che

$$\sum_{i=1}^{m} a_i < \sum_{j=1}^{n} b_j.$$

cioé nei depositi vi sia meno prodotto di quanto effettivamente richiesto dai negozi. Per soddisfare l'ipotesi (3.13) basta aggiungere un deposito fittizio $m+1$ con

$$a_{m+1} = \sum_{j=1}^{n} b_j - \sum_{i=1}^{m} a_i,$$

e con $c_{m+1,j} = 0$ per ogni $j,\ j = 1, \ldots, n$, cioé il costo del trasporto dal deposito fittizio é pari a 0. La quantitá di prodotto che un negozio riceve da un deposito fittizio equivale in realtá a una richiesta non soddisfatta per quel negozio.

3.6.1 Modello matematico del problema

Vediamo ora di formulare il modello matematico del problema del trasporto. Ad ogni coppia deposito i-negozio j associamo una variabile x_{ij} che corrisponde alla quantitá di prodotto inviata dal deposito i verso il negozio j. Tale quantitá dovrá essere ovviamente non negativa, ovvero:

$$x_{ij} \geq 0 \quad i = 1, \ldots, m \;\; j = 1, \ldots, n.$$

Se inviare un'unitá di prodotto dal deposito i al negozio j ha costo pari a c_{ij}, inviarne una quantitá x_{ij} ha costo pari a $c_{ij}x_{ij}$. Sommando su tutte le possibili coppie deposito-negozio, abbiamo la seguente formula per l'obiettivo:

$$\sum_{i=1}^{m} \sum_{j=1}^{n} c_{ij} x_{ij}.$$

Quello che desideriamo é minimizzare questo costo totale di trasporto. Per ogni deposito i la quantitá totale di prodotto inviata da esso deve essere pari alla quantitá di prodotto a_i in esso immagazzinata. La quantitá totale di prodotto inviata dal deposito i é data dalla formula $\sum_{j=1}^{n} x_{ij}$ e quindi per ogni deposito i avremo il seguente vincolo:

$$\sum_{j=1}^{n} x_{ij} = a_i \quad i = 1, \ldots, m.$$

Analogamente, per ogni negozio j la quantitá totale di prodotto ricevuta da esso deve essere pari alla quantitá di prodotto b_j da esso richiesta. La quantitá totale di prodotto ricevuta dal negozio j é data dalla formula $\sum_{i=1}^{m} x_{ij}$ e quindi per ogni negozio j avremo il seguente vincolo:

$$\sum_{i=1}^{m} x_{ij} = b_j \quad j = 1, \ldots, n.$$

Riassumendo, il modello matematico del problema del trasporto é il seguente problema di PL:

$$\begin{array}{ll} \min & \sum_{i=1}^{m} \sum_{j=1}^{n} c_{ij} x_{ij} \\ & \sum_{j=1}^{n} x_{ij} = a_i \qquad i = 1, \ldots, m \\ & \sum_{i=1}^{m} x_{ij} = b_j \qquad j = 1, \ldots, n \\ & x_{ij} \geq 0 \;\text{ interi} \qquad i = 1, \ldots, m \; j = 1, \ldots, n \end{array}$$

Possiamo anche associare al problema un grafo bipartito completo $K_{m,n}$ dove su un lato della bipartizione compaiono i nodi corrispondenti ai depositi, numerati da 1 a m, mentre sull'altro lato della bipartizione compaiono i nodi corrispondenti ai negozi, numerati da $m + 1$ a $m + n$ (si veda la Figura 3.30). In forma

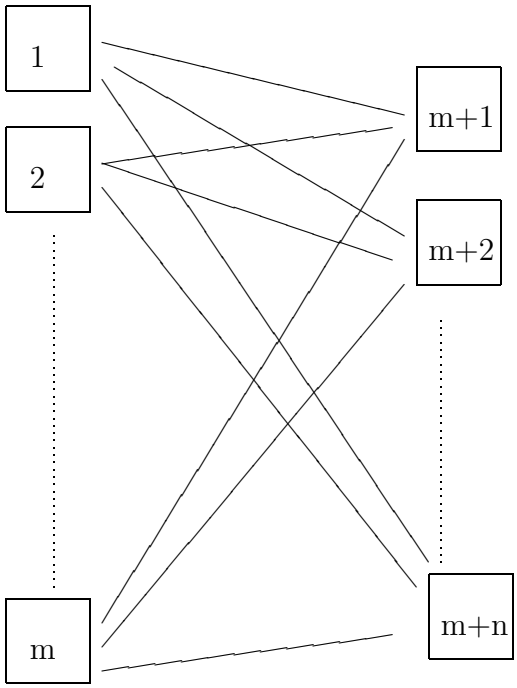


Figura 3.30: Il grafo bipartito associato a un problema del trasporto.

matriciale il problema si scrive nella seguente forma:

$$\begin{aligned} \min \quad & CX \\ & MX = D \\ & X \geq 0 \end{aligned}$$

dove:

- X é il vettore di variabili di dimensione mn con componenti le variabili x_{ij};
- C é un vettore di dimensione mn la cui componente associata alla coppia deposito i-negozio j é pari a c_{ij};
- M é una matrice di ordine $(m+n) \times mn$ che coincide con la matrice di incidenza nodo-arco del grafo bipartito completo associato al problema del trasporto. La colonna relativa alla coppia deposito i-negozio j di tale matrice e quindi alla variabile x_{ij} é data dal vettore

  $$e_i + e_{m+j}$$

  dove e_k, $k \in \{1, \ldots, m+n\}$, é un vettore di dimensione $m+n$ le cui componenti sono tutte nulle tranne la k-esima che é pari a 1.

Tabella 3.6:

	1	2	3
1	$c_{11} = 4$	$c_{12} = 7$	$c_{13} = 5$
2	$c_{21} = 2$	$c_{22} = 4$	$c_{23} = 3$

- D é un vettore di dimensione $m + n$ la cui i-esima componente, $i = 1, \ldots, m$, é pari ad a_i, mentre la sua $(m + j)$-esima componente, $j = 1, \ldots, n$, é pari a b_j.

In base al Corollario 2 si ha che M é TU. Quindi se il vettore D é a componenti tutte intere, cioé i valori a_i e b_j sono quantitá tutte intere, possiamo utlizzare il Teorema 1 con $A_3 \equiv M$, $b_3 \equiv D$, per dimostrare il seguente risultato che ci consente di non introdurre esplicitamente vincoli di interezza delle variabili nel problema di trasporto.

Teorema 4 *Se i valori a_i, $i = 1, \ldots, m$, e b_j, $j = 1, \ldots, n$, sono tutti interi, allora ogni vertice ottimo del problema del trasporto é a coordinate intere.*

Esempio 15 *Si consideri il problema del trasporto con 2 depositi e 3 negozi e con*

$$a_1 = 30 \quad a_2 = 20 \quad b_1 = 15 \quad b_2 = 10 \quad b_3 = 25.$$

Per prima cosa notiamo che

$$\sum_{i=1}^{2} a_i = 30 + 20 = \sum_{j=1}^{3} b_j = 15 + 10 + 25.$$

Si supponga inoltre che i costi unitari di trasporto per le diverse coppie deposito-negozio siano date dalla Tabella 3.6 Il modello matematico di questo problema é il seguente:

$$\begin{aligned} \min \quad & 4x_{11} + 7x_{12} + 5x_{13} + 2x_{21} + 4x_{22} + 3x_{23} \\ & x_{11} + x_{12} + x_{13} = 30 \\ & x_{21} + x_{22} + x_{23} = 20 \\ & x_{11} + x_{21} = 15 \\ & x_{12} + x_{22} = 10 \\ & x_{13} + x_{23} = 25 \\ & x_{11}, x_{12}, x_{13}, x_{21}, x_{22}, x_{23} \geq 0 \end{aligned}$$

La matrice dei vincoli é data nella Tabella 3.7 e si puó vedere che coincide con la matrice di incidenza nodo-arco del grafo bipartito completo $K_{2,3}$ associato a questo problema e illustrato in Figura 3.31.

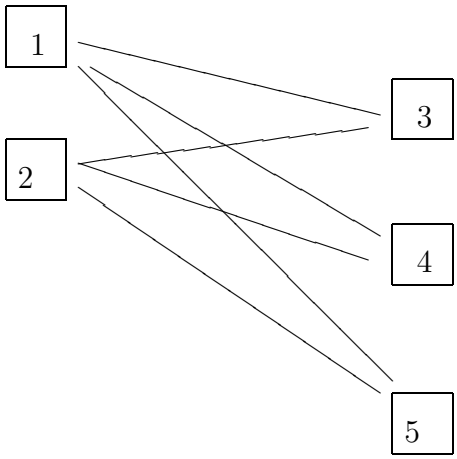


Figura 3.31: Il grafo bipartito associato al problema del trasporto dell'esempio.

Tabella 3.7:

	x_{11}	x_{12}	x_{13}	x_{21}	x_{22}	x_{23}
Deposito 1	1	1	1	0	0	0
Deposito 2	0	0	0	1	1	1
Negozio 1	1	0	0	1	0	0
Negozio 2	0	1	0	0	1	0
Negozio 3	0	0	1	0	0	1

Resta ancora da sottolineare che quando la condizione (3.13) é soddisfatta (e soltanto in questo caso), il problema del trasporto ammette sempre almeno una soluzione ammissibile e almeno una soluzione ottima. Infatti, la condizione (3.13) garantisce l'esistenza di una soluzione ammissibile (nel seguito verrá anche data una procedura per determinare una tale soluzione). Inoltre, si ha, ad esempio, che:

$$x_{ij} \geq 0, \quad \sum_{j=1}^{n} x_{ij} = a_i \quad \Rightarrow \quad x_{ij} \leq a_i \quad i = 1, \ldots, m \;\; j = 1, \ldots, n,$$

ovvero tutte le variabili hanno valori limitati e la regione ammissibile é dunque essa stessa un insieme limitato. In base a ben note proprietá dei problemi di PL in generale, questo garantisce l'esistenza di una soluzione ottima. Ció esclude a priori i casi di illimitatezza e la verifica se la regione ammissibile é vuota oppure no si riduce alla verifica della condizione (3.13).

3.6.2 Basi del problema del trasporto

Si puó dimostrare che il rango della matrice dei vincoli M in un problema del trasporto é pari a $m + n - 1$. Una prima definizione, standard, di base per il problema del trasporto é la seguente.

Definizione 12 *Una base per il problema del trasporto é un insieme di $m+n-1$ variabili le cui corrispondenti colonne nella matrice dei vincoli sono linearmente indipendenti.*

Tuttavia vedremo che é possibile dare una rappresentazione grafica piú intuitiva di una base. Consideriamo una tabella con i depositi sulle righe e i negozi sulle colonne. Dato un insieme di $m+n-1$ variabili, associamo ad esso un grafo nel modo seguente: associamo un nodo ad ogni cella relativa ad una delle $m+n-1$ variabili e colleghiamo tra loro mediante archi nodi successivi che si trovano lungo la stessa riga o lungo la stessa colonna. Si noti che non si tracciano archi tra due nodi lungo la stessa riga (o colonna) se in mezzo ad essi sulla stessa riga (o colonna) vi é un altro nodo. In Figura 3.32 é rappresentato il grafo corrsipondente alle $m + n - 1 = 4$ variabili del nostro esempio $\{x_{12}, x_{13}, x_{22}, x_{23}\}$, mentre in Figura 3.33 é rappresentato il grafo corrsipondente alle $m+n-1 = 4$ variabili $\{x_{11}, x_{12}, x_{13}, x_{23}\}$. In entrambi i casi abbiamo $m+n-1$ variabili, ma

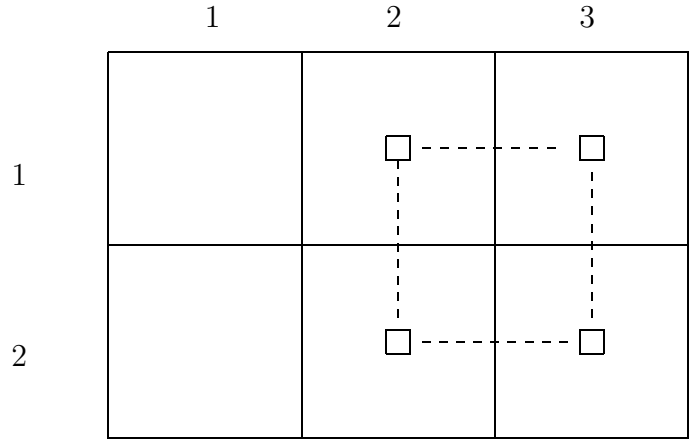


Figura 3.32: Il grafo associato alle variabili $\{x_{12}, x_{13}, x_{22}, x_{23}\}$.

non tutti gli insiemi di $m+n-1$ variabili formano una base, é anche necessario che le colonne corrispondenti nella matrice dei vincoli siano tra loro linearmente indipendenti. Ci chiediamo allora se sia possibile verificare questa condizione di indipendenza lineare basandosi sulla sola osservazione del grafo associato. Nel primo grafo, associato alle variabili alle variabili $\{x_{12}, x_{13}, x_{22}, x_{23}\}$, notiamo che é presente un ciclo. Per prima cosa tralasciamo i nodi del ciclo che si trovino sulla stessa riga o colonna di altri due nodi del ciclo e siano in mezzo a questi. Ad esempio, in Figura 3.34 tralasciamo i nodi nelle celle $(1, 2)$ e $(2, 1)$. Una

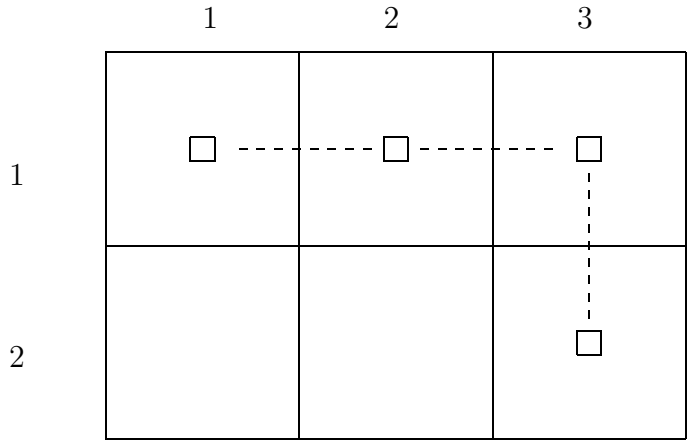


Figura 3.33: Il grafo associato alle variabili $\{x_{11}, x_{12}, x_{13}, x_{23}\}$.

volta eliminati tali nodi (ma non ne sono presenti nell'esempio di Figura 3.32 che stiamo considerando), prendiamo un nodo del ciclo ed assegniamo il coefficiente +1 alla colonna relativa a tale nodo. Quindi ci muoviamo lungo il ciclo assegnando alternativamente alle colonne relative ai nodi del ciclo i coefficienti -1 e +1. Ricordando che la colonna relativa ad una variabile x_{ij} della matrice dei vincoli é data dalla somma dei vettori $e_i + e_{m+j}$ e partendo dal nodo nella cella $(1, 2)$ avremo nel nostro esempio

$$+1(e_1 + e_4) - 1(e_1 + e_5) + 1(e_2 + e_5) - 1(e_2 + e_4) = 0.$$

Questo ci dice che le colonne non sono tra loro linearmente indipendenti. Quanto visto nell'esempio vale in generale: se il grafo associato a $m + n - 1$ variabili forma un ciclo, le $m + n - 1$ variabili non formano una base. Quindi il grafo associato ad una base deve essere privo di cicli. Si puó inoltre dimostrare che esso deve soddisfare anche le seguenti due condizioni:

A: contiene un nodo in ogni riga e in ogni colonna della tabella;

B: é connesso.

Vale anche il viceversa, ovvero dato un grafo sulla tabella con $m + n - 1$ nodi, privo di cicli e che soddisfa le condizioni A e B, le $m + n - 1$ variabili relative ai nodi formano una base. Riassumendo, abbiamo quindi visto il seguente risultato.

Osservazione 5 *Nel problema del trasporto $m + n - 1$ variabili formano una base se e solo se il grafo ad esso associato é privo di cicli e soddisfa le condizioni A e B.*

Nel nostro esempio notiamo che il grafo associato alle variabili $\{x_{11}, x_{12}, x_{13}, x_{23}\}$ é privo di cicli e soddisfa le condizioni A e B. Quindi tali variabili formano una

base. Una base verrá detta ammissibile se ponendo uguali a 0 tutte le variabili fuori base nei vincoli, il sistema risultante restituisce valori non negativi per le variabili in base. Nel nostro caso ponendo a 0 le variabili fuori base si ottiene il seguente sistema:

$$\begin{aligned} x_{11} + x_{12} + x_{13} &= 30 \\ x_{23} &= 20 \\ x_{11} &= 15 \\ x_{12} &= 10 \\ x_{13} + x_{23} &= 25 \end{aligned}$$

che ha come soluzione

$$x_{11} = 15 \quad x_{12} = 10 \quad x_{13} = 5 \quad x_{23} = 20.$$

Dunque la base é ammissibile.

3.6.3 Il metodo del simplesso per il trasporto

I passi del metodo del simplesso per il problema di trasporto sono praticamente gli stessi di quelli del metodo del simplesso per i generici problemi di PL. La sola differenza é l'assenza, nel simplesso per il trasporto, della verifica di illimitatezza (abbiamo visto che il problema del trasporto ammette sempre soluzioni ottime). Per gli altri passi cambia solo il modo in cui essi sono implementati (l'implementazione sfrutta la particolare struttura del problema del trasporto). Quindi anche qui dovremo intanto determinare una base ammissibile iniziale. Poi ad ogni iterazione dovremo:

- calcolare i coefficienti di costo ridotto delle variabili fuori base;
- verificare se sono soddisfatte le condizioni di ottimalitá;
- se non sono soddisfatte scegliere una variabile da far entrare in base;
- determinare la variabile da far uscire di base;
- effettuare il cambio di base e ripetere tutti i passi con la nuova base.

Nel seguito vedremo in dettaglio come si esegue ciascuno di questi passi.

Determinazione di una base ammissibile iniziale

Come giá detto, se é soddisfatta la condizione (3.13) il problema del trasporto ammette sempre una soluzione ammissibile e quindi, in base ad un teorema della PL, ammette anche almeno una base ammissibile. Tra le possibili regole per

determinare una base ammissibile iniziale descriveremo ora le cosidetta *regola dell'angolo nord-ovest*. Inizialmente si pone

$$\hat{a}_i = a_i, \quad i = 1, \ldots, m \quad \hat{b}_j = b_j, \quad j = 1, \ldots, n.$$

Si parte dalla cella $(1, 1)$ (appunto la cella nell'angolo nord-ovest) e la prima variabile inserita nella base é la corrispondente variabile x_{11} a cui viene assegnato il valore:

$$x_{11} = \min\{\hat{a}_1, \hat{b}_1\},$$

e si pongono

$$\hat{a}_1 = \hat{a}_1 - x_{11} \quad \hat{b}_1 = \hat{b}_1 - x_{11}.$$

A questo punto se $\hat{a}_1 > \hat{b}_1$ ci si sposta nella cella $(1, 2)$ e la variabile successiva posta nella base é x_{12} con valore:

$$x_{12} = \min\{\hat{a}_1, \hat{b}_2\},$$

ponendo poi

$$\hat{a}_1 = \hat{a}_1 - x_{12} \quad \hat{b}_2 = \hat{b}_2 - x_{12}.$$

Se invece $\hat{a}_1 < \hat{b}_1$ ci si sposta nella cella $(2, 1)$ e la variabile successiva posta nella base é x_{21} con valore:

$$x_{21} = \min\{\hat{a}_2, \hat{b}_1\},$$

ponendo poi

$$\hat{a}_2 = \hat{a}_2 - x_{21} \quad \hat{b}_1 = \hat{b}_1 - x_{21}.$$

Se infine $\hat{a}_1 = \hat{b}_1$ (caso degenere) ci si sposta indifferentemente nella cella $(1, 2)$ facendo entrare in base x_{12} con valore 0 oppure nella cella $(2, 1)$ facendo entrare in base x_{21} sempre con valore 0.
Supponiamo ora di esserci spostati nella cella $(1, 2)$. A questo punto se $\hat{a}_1 > \hat{b}_2$ ci si sposta nella cella $(1, 3)$ e la variabile successiva posta nella base é x_{13} con valore:

$$x_{13} = \min\{\hat{a}_1, \hat{b}_3\},$$

ponendo poi

$$\hat{a}_1 = \hat{a}_1 - x_{13} \quad \hat{b}_3 = \hat{b}_3 - x_{13}.$$

Se invece $\hat{a}_1 < \hat{b}_2$ ci si sposta nella cella $(2, 2)$ e la variabile successiva posta nella base é x_{22} con valore:

$$x_{22} = \min\{\hat{a}_2, \hat{b}_2\},$$

ponendo poi

$$\hat{a}_2 = \hat{a}_2 - x_{22} \quad \hat{b}_2 = \hat{b}_2 - x_{22}.$$

Se infine $\hat{a}_1 = \hat{b}_2$ (caso degenere) ci si sposta indifferentemente nella cella $(1, 3)$ facendo entrare in base x_{13} con valore 0 oppure nella cella $(2, 2)$ facendo entrare in base x_{22} sempre con valore 0.

Si procede in questo modo fino a quando con questo processo si arriva alla cella (m, n), l'ultima cella in basso a destra. A questo punto le celle attraversate durante il processo formano una base.
Illustereremo ora tale regola sul nostro esempio. Poniamo:

$$\hat{a}_1 = 30 \quad \hat{a}_2 = 20 \quad \hat{b}_1 = 15 \quad \hat{b}_2 = 10 \quad \hat{b}_3 = 25.$$

Partiamo dalla cella $(1, 1)$ inserendo nella base la variabile x_{11} con valore:

$$x_{11} = \min\{\hat{a}_1, \hat{b}_1\} = 15,$$

e ponendo:

$$\hat{a}_1 = 30 - 15 = 15 \qquad \hat{b}_1 = 15 - 15 = 0.$$

A questo punto si ha $\hat{a}_1 > \hat{b}_1$ e quindi ci si sposta nella cella $(1, 2)$ e la variabile successiva posta nella base é x_{12} con valore:

$$x_{12} = \min\{\hat{a}_1, \hat{b}_2\} = 10,$$

ponendo poi

$$\hat{a}_1 = 15 - 10 = 5 \qquad \hat{b}_2 = 10 - 10 = 0.$$

A questo punto si ha $\hat{a}_1 > \hat{b}_2$ e quindi ci si sposta nella cella $(1, 3)$ e la variabile successiva posta nella base é x_{13} con valore:

$$x_{13} = \min\{\hat{a}_1, \hat{b}_3\} = 5,$$

ponendo poi

$$\hat{a}_1 = 5 - 5 = 0 \qquad \hat{b}_3 = 25 - 5 = 20.$$

A questo punto si ha $\hat{a}_1 < \hat{b}_3$ e quindi ci si sposta nella cella $(2, 3)$ e la variabile successiva posta nella base é x_{23} con valore:

$$x_{23} = \min\{\hat{a}_2, \hat{b}_3\} = 20,$$

ponendo poi

$$\hat{a}_2 = 20 - 20 = 0 \qquad \hat{b}_3 = 20 - 20 = 0.$$

Essendo arrivati nella cella in basso a destra ci arrestiamo. L'insieme di variabili $\{x_{11}, x_{12}, x_{13}, x_{23}\}$ forma una base ammissibile con il grafo associato illsutrato in Figura 3.33 e con i seguenti valori della variabili in base:

$$x_{11} = 15 \quad x_{12} = 10 \quad x_{13} = 5 \quad x_{23} = 20.$$

mentre le variabili fuori base x_{21} e x_{22} hanno entrambe valore nullo.

Calcolo dei coefficienti di costo ridotto

Ad ogni variabile fuori base é associato un coefficiente di costo ridotto il cui valore indica di quanto varierebbe il valore dell'obiettivo (ovvero il costo totale di trapsorto) se facessimo crescere a 1 il valore della corrispondente variabile fuori base. Se tutti i coefficienti di costo ridotto sono non negativi la base corrente é ottima dal momento che ogni incremento delle variabili fuori base non ridurrebbe il costo totale di trasporto. In particolare, se tutti i coefficienti di costo ridotto sono positivi, la soluzione di base corrispondente é l'unica soluzione ottima del problema.
Se invece esistono coefficienti di costo ridotto negativi, facendo entrare in base una variabile fuori base con coefficiente di costo ridotto negativo potremmo ridurre il costo totale di trasporto. Quindi la conoscenza dei coefficienti di costo ridotto é necessaria sia per la verifica della condizione di ottimalitá (tutti i coefficienti di costo ridotto sono non negativi) sia per la determinazione della variabile da far entrare in base qualora non sia soddisfatta la condizione di ottimalitá. Vediamo allora come vengono calcolati questi coefficienti. Nel grafo relativo alla base ammissibile corrente aggiungiamo un nodo nella cella corrispondente alla variabile fuori base di cui vogliamo calcolare il coefficiente di costo ridotto e congiungiamo tale nodo con i nodi della base ad esso adiacenti lungo la riga e la colonna in cui si trova la cella. Nel nostro esempio se partiamo dalla base ammissibile individuata con la regola dell'angolo nord-ovest e vogliamo calcolare il coefficiente di costo ridotto di x_{21}, il grafo ottenuto aggiungendo il nodo nella cella $(2, 1)$ é quello illustrato in Figura 3.35. L'aggiunta del nodo e degli archi forma un ciclo dal quale escludiamo, come giá visto in precedenza, ogni nodo che si trovi nel mezzo di altri due lungo una riga o una colonna, congiungendo direttamente questi ultimi. Per il nostro esempio questa operazione conduce al grafo in Figura 3.36. A questo punto il coefficiente di costo ridotto della variabile si calcola partendo dal costo unitario della cella relativa alla variabile fuori base, sottraendo il costo unitario della cella successiva nel ciclo, sommando quello della cella ancora successiva e cosí via alternando sottrazioni e addizioni fino a quando non si chiude il ciclo. Nel nostro esempio avremo:

$$\overline{c}_{21} = c_{21} - c_{11} + c_{13} - c_{23} = 2 - 4 + 5 - 3 = 0.$$

Ripetiamo ora la procedura per il calcolo del coefficiente di costo ridotto dell'altra variabile fuori base, la x_{22}. Si ha:

$$\overline{c}_{22} = c_{22} - c_{12} + c_{13} - c_{23} = 4 - 7 + 5 - 3 = -1.$$

Come si puó vedere si ha un coefficiente di costo ridotto negativo. Questo ci impedisce di concludere che la base é ottima. Come variabile da far entrare in base possiamo scegliere tra quelle con coefficiente di costo ridotto negativo (nell'esempio la sola x_{22}). Come regola di scelta useremo la seguente: entra in base la variabile con coefficiente di costo ridotto piú piccolo (con scelta arbitraria in caso di paritá).

Scelta della variabile uscente dalla base

Una volta scelta quale variabile entrerá in base nel modo che abbiamo visto, dobbiamo ora decidere quale variabile uscirá dalla base. Come giá visto in precedenza, aggiungiamo un nodo nella cella corrispondente alla variabile che vogliamo far entrare in base e congiungiamo tale nodo con i nodi della base ad esso adiacenti lungo la riga e la colonna in cui si trova la cella. Nel nostro esempio il grafo ottenuto aggiungendo il nodo nella cella $(2, 2)$ é quello illustrato in Figura 3.37. L'aggiunta del nodo e degli archi forma un ciclo dal quale escludiamo, come giá visto in precedenza, ogni nodo che si trovi nel mezzo di altri due lungo una riga o una colonna, congiungendo direttamente questi ultimi (ma nel nostro esempio non ce ne sono). A questo punto se facciamo crescere a Δ il valore della variabile che facciamo entrare in base, dobbiamo, per poter rispettare i vincoli, aggiornare il valore delle altre variabili nel ciclo nel modo seguente: la prima che si incontra nel ciclo viene decrementata di Δ, la seconda incrementata di Δ e cosí via alternando decrementi e incrementi fino a quando si chiude il ciclo. Nel nostro esempio avremo:

$$x_{22} = \Delta \quad x_{12} = 10 - \Delta \quad x_{13} = 5 + \Delta \quad x_{23} = 20 - \Delta.$$

mentre le variabili in base fuori dal ciclo (nell'esempio la sola $x_{11} = 15$) e le altre variabili fuori base (nell'esempio la sola $x_{21} = 0$) rimangono invariate. Il valore Δ puó essere fatto crescere fino a quando tutte le variabili in base hanno valore non negativo. La prima variabile che si annulla al crescere di Δ é la variabile che dovrá uscire di base. Se se ne annulla piú di una contemporaneamente (caso degenere) la scelta di quale fare uscire di base é arbitraria. Nel nostro esempio possiamo far crescere Δ fino a 10. A questo punto la sola x_{12} si annulla e viene portata fuori base. La nuova base sará quindi $\{x_{11}, x_{13}, x_{22}, x_{23}\}$ con il relativo grafo in Figura 3.38 e i seguenti valori delle variabili in base:

$$x_{11} = 15 \quad x_{22} = 10 \quad x_{13} = 15 \quad x_{23} = 10.$$

Vediamo ora di terminare l'esempio. Dobbiamo ripetere con la nuova base quanto fatto con quella precedente. Cominciamo dal calcolo dei coefficienti di costo ridotto per le variabili fuori base x_{12} e x_{21}. Avremo:

$$\overline{c}_{21} = c_{21} - c_{11} + c_{13} - c_{23} = 2 - 4 + 5 - 3 = 0.$$

$$\overline{c}_{12} = c_{12} - c_{13} + c_{23} - c_{22} = 7 - 5 + 3 - 4 = 1.$$

Come si puó vedere tutti i coefficienti di costo ridotto sono non negativi. Questo ci consente di concludere che la base é ottima. Si noti peró che la presenza di un coefficiente di costo ridotto nullo indica la possibile presenza di altre soluzioni ottime. In realtá, essendo la soluzione di base corrispondente non degenere, possiamo garantire che esistono altre soluzioni ottime. Volendo individuare un'altra soluzione ottima possiamo far entrare in base la variabile x_{21} con coefficiente di costo ridotto nullo.

Il valore ottimo si ottiene andando a sostituire nell'obiettivo i valori delle variabili in corrispondenza della soluzione ottima:

$4x_{11}+7x_{12}+5x_{13}+2x_{21}+4x_{22}+3x_{23} = 4\times15+7\times0+5\times15+2\times0+4\times10+3\times10 = 205.$

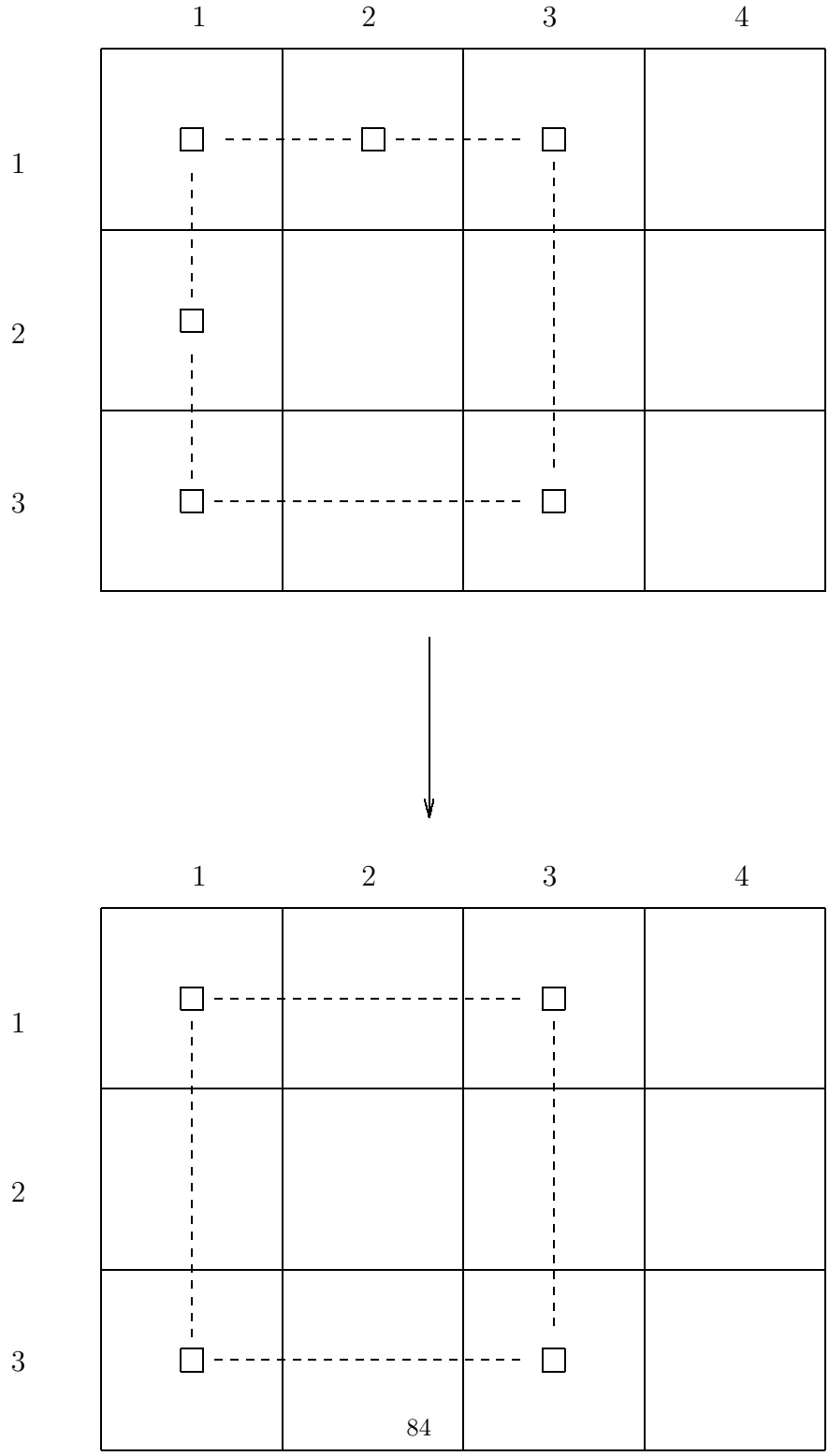


Figura 3.34: Esclusione di nodi del grafo.

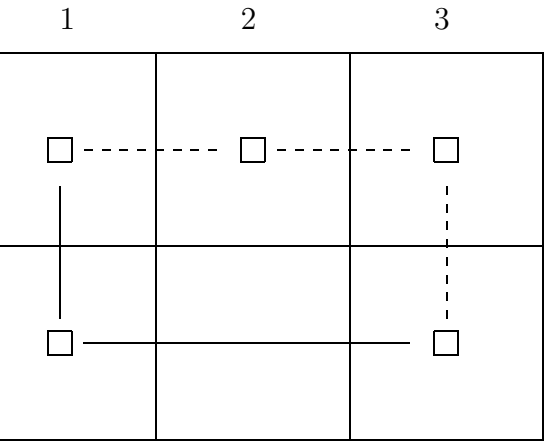


Figura 3.35: Il grafo ottenuto aggiungendo il nodo relativo alla variabile x_{21}.

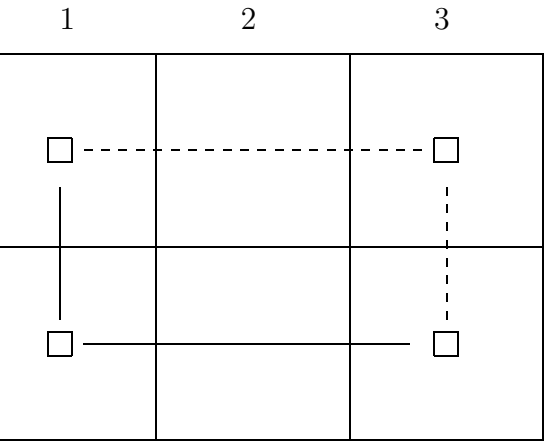


Figura 3.36: Eliminazione di nodi nel ciclo.

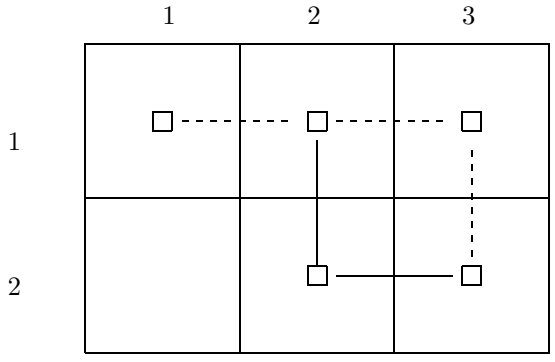


Figura 3.37: L'aggiunta del nodo relativo alla variabile x_{22}.

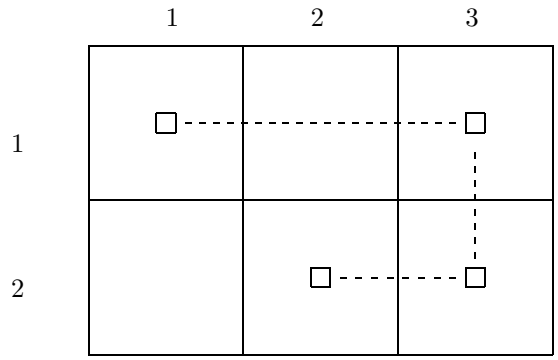


Figura 3.38: La nuova base $\{x_{11}, x_{22}, x_{13}, x_{23}\}$

3.7 Il problema di assegnamento

Siano dati due insiemi A e B entrambi di cardinalitá n. Ad ogni coppia $(a_i, b_j) \in A \times B$ è associato un valore $d(a_i, b_j) \geq 0$ che misura la "incompatibilitá" tra a_i e b_j (tanto piú $d(a_i, b_j)$ e' grande, quanto piú a_i e b_j sono tra loro incompatibili). Il problema di assegnamento è il seguente:

Problema 1 *Individua n coppie di elementi appartenenti ad $A \times B$ in modo tale che ogni elemento di A e di B deve appartenere ad una ed una sola coppia ed in modo tale da minimizzare la "incompatibilitá" totale, data dalla somma delle "incompatibilitá" di ogni singola coppia.*

Nel caso $n = 3$, le seguenti tre coppie

$$(a_1, b_2) \quad (a_2, b_3) \quad (a_3, b_1)$$

rappresentano una soluzione ammissibile del problema, in quanto ogni elemento di A e B è contenuto in una ed una sola coppia. Per tale soluzione la "incompatibilitá" totale è pari a

$$d(a_1, b_2) + d(a_2, b_3) + d(a_3, b_1).$$

Le tre coppie

$$(a_1, b_2) \quad (a_1, b_3) \quad (a_3, b_1)$$

non rappresentano invece una soluzione ammissibile in quanto, per esempio, l'elemento a_1 è presente in due coppie.
Un tipico esempio di problema di assegnamento è quello in cui si hanno n lavori da compiere (insieme A) e n lavoratori (insieme B). Dato il lavoratore b_j ed il lavoro a_i, il valore $d(a_i, b_j)$ misura l'attitudine del lavoratore b_j a compiere il lavoro a_i (tanto maggiore è tale valore, quanto minore è l'attitudine). Il problema dell'assegnamento quindi cerca di accoppiare lavori e lavoratori in modo tale da rendere minima l'incompatibilitá complessiva tra questi.

Osservazione 6 *Si è supposto che A e B abbiano la stessa cardinalitá n. Vi sono peró casi in cui questo non è vero. Nell'esempio dei lavori, vi possono essere piú lavori che lavoratori ($| A | > | B |$), o piú lavoratori che lavori ($| A | < | B |$). Questi casi possono* **sempre** *essere ricondotti al caso $| A | = | B |$ aggiungendo elementi fittizi. Per esempio, nel caso vi siano piú lavoratori che lavori, si aggiungono $| B | - | A |$ lavori fittizi a_i e per ognuno di questi la sua "incompatibilitá" $d(a_i, b_j)$ viene fissata a 0 per ogni lavoratore b_j. In questo modo ci si è ricondotti al caso con $| A | = | B |$. Assegnare un lavoro fittizio ad un lavoratore equivale a non assegnargli alcun lavoro.*

A un problema di assegnamento si associa un grafo bipartito completo $K_{n,n}$ con n nodi associati agli elementi a_i dell'insieme A su un lato della bipartizione e n nodi associati agli elementi b_j dell'insieme B sull'altro lato della bipartizione.

Esempio 16 *Si consideri l'insieme A formato dagli elementi a_1, a_2, a_3, a_4 e l'insieme B formato dagli elementi b_1, b_2, b_3, b_4. I valori di incompatibilitá $d(a_i, b_j)$, $i, j = 1, 2, 3, 4$, tra elementi dell'insieme A ed elementi dell'insieme B sono riportati nella seguente matrice:*

$$\begin{bmatrix} 2 & 3 & 4 & 5 \\ 6 & 2 & 2 & 2 \\ 7 & 2 & 3 & 3 \\ 2 & 3 & 4 & 5 \end{bmatrix}, \tag{3.14}$$

mentre il grafo bipartito completo $K_{4,4}$ associato al problema é riportato in Figura 3.39.

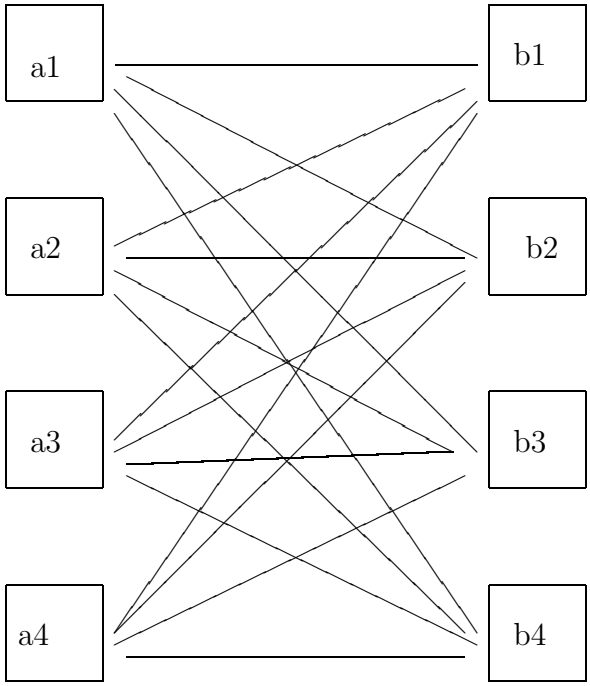


Figura 3.39: Il grafo bipartito completo $K_{4,4}$ associato al problema dell'esempio.

3.7.1 Formulazione del problema mediante la PL

Indichiamo piú semplicemente con d_{ij} i valori $d(a_i, b_j)$. Ad ogni coppia $(a_i, b_j) \in A \times B$ si associa una variabile x_{ij} con i seguenti possibili valori:

$$x_{ij} = \begin{cases} 1 & \text{se } a_i \text{ è assegnato a } b_j \\ 0 & \text{altrimenti} \end{cases}$$

Quindi $x_{ij} \in \{0,1\}$. I vincoli sono i seguenti. Ad ogni elemento a_i è assegnato uno ed un solo b_j e quindi avremo i seguenti n vincoli

$$\sum_{j=1}^{n} x_{ij} = 1 \quad \forall\, i \in \{1,\ldots,n\};$$

ad ogni elemento b_j è assegnato uno ed un solo a_i e quindi avremo i seguenti n vincoli

$$\sum_{i=1}^{n} x_{ij} = 1 \quad \forall\, j \in \{1,\ldots,n\};$$

Il contributo all'incompatibilitá totale di (a_i, b_j) é d_{ij} se a_i viene assegnato a b_j, cioé $x_{ij} = 1$, ed é 0 se a_i non viene assegnato a b_j, cioé $x_{ij} = 0$. In entrambi i casi il contributo all'incompatibilitá totale di (a_i, b_j) é $d_{ij}x_{ij}$. Sommando su tutte le possibili coppie si ottiene l'obiettivo del problema:

$$\sum_{i=1}^{n}\sum_{j=1}^{n} d_{ij}x_{ij}.$$

Quindi, riassumendo, il problema di assegnamento è un problema di PLI con la seguente forma

$$\begin{array}{ll} \min & \sum_{i=1}^{n}\sum_{j=1}^{n} d_{ij}x_{ij} \\ & \sum_{j=1}^{n} x_{ij} = 1 \qquad \forall\, i \in \{1,\ldots,n\} \\ & \sum_{i=1}^{n} x_{ij} = 1 \qquad \forall\, j \in \{1,\ldots,n\} \\ & x_{ij} \in \{0,1\} \qquad \forall\, i,j \end{array}$$

Vediamo ora di rilassare il vincolo di interezza sulle variabili. Questo vuol dire che i vincoli $x_{ij} \in \{0,1\}$ vengono sostituiti con i vincoli $0 \le x_{ij} \le 1$, ottenendo quindi il seguente problema di PL:

$$\begin{array}{ll} \min & \sum_{i=1}^{n}\sum_{j=1}^{n} d_{ij}x_{ij} \\ & \sum_{j=1}^{n} x_{ij} = 1 \qquad \forall\, i \in \{1,\ldots,n\} \\ & \sum_{i=1}^{n} x_{ij} = 1 \qquad \forall\, j \in \{1,\ldots,n\} \\ & 0 \le x_{ij} \le 1 \qquad \forall\, i,j \end{array}$$

Ma possiamo notare che:

$$x_{ij} \ge 0, \quad \sum_{j=1}^{n} x_{ij} = 1 \quad \forall\, i \in \{1,\ldots,n\} \quad \Rightarrow \quad x_{ij} \le 1 \ \ \forall\, i,j \in \{1,\ldots,n\},$$

il che significa che i vincoli $x_{ij} \le 1$ sono del tutto inutili e possono quindi essere eliminati. Possiamo quindi riscrivere il problema di PL nella seguente forma:

$$\begin{array}{ll} \min & \sum_{i=1}^{n}\sum_{j=1}^{n} d_{ij}x_{ij} \\ & \sum_{j=1}^{n} x_{ij} = 1 \qquad \forall\, i \in \{1,\ldots,n\} \\ & \sum_{i=1}^{n} x_{ij} = 1 \qquad \forall\, j \in \{1,\ldots,n\} \\ & x_{ij} \ge 0 \qquad \forall\, i,j \end{array} \tag{3.15}$$

Una volta riscritto in questo modo, notiamo che il problema di PL é un caso particolare di problema del trasporto dove vi sono n depositi e n negozi, ogni deposito ha una sola unitá di prodotto a disposizione e ogni negozio richiede una sola unitá di prodotto. In base a quanto visto in generale sul problema del trasporto, possiamo concludere che tutti i vertici ottimi del problema di PL (3.15) sono a coordinate intere e in particolare dovranno avere coordinate pari a 0 oppure 1. Questo ci consente di dire che risolvere il problema di PL (3.15) é del tutto equivalente a risolvere il nostro originario problema di assegnamento e ci consente di inserire anche il problema di assegnamento nella classe P.
Notiamo anche che, essendo il problema un caso particolare di problema del trasporto, potremmo usare per risolverlo l'algoritmo del simplesso per il problema del trasporto. La difficoltá in questo caso è la forte degenerazione di tutti i vertici del problema. Infatti, una soluzione non degenere dovrebbe avere $2n-1$ variabili (quelle in base) diverse da zero, ma i vertici (che coincidono con le soluzioni ammissibili del problema di assegnamento) possono avere solo n variabili uguali a 1 (ci sono n coppie nell'assegnamento) e quindi abbiamo una forte degenerazione. Conviene dunque utilizzare altri algoritmi per la risoluzione del problema di assegnamento. Qui si presenterá un algoritmo chiamato *algoritmo ungherese*. Accenniamo brevemente anche al fatto che il problema di assegnamento puó anche essere visto come caso particolare di problema di matching, ovvero come problema di matching pesato su grafi bipartiti completi.

3.7.2 Algoritmo ungherese

Come il simplesso per il problema del trasporto é un adattamento del metodo del simplesso alla particolare struttura del problema del trasporto, allo stesso modo l'algoritmo ungherese puó essere visto come adattamento alla particolare struttura del problema di assegnamento di una variante del metodo del simplesso, il metodo del simplesso primale-duale, che non é stato visto a lezione. Qui ci limiteremo a descrivere i passi dell'algoritmo senza dimostrarne la derivazione dal metodo del simplesso primale-duale.
Si parte dalla matrice T_0 di ordine $n \times n$ che ha come elemento nella posizione (i,j) il valore d_{ij}. Il primo passo consiste nel trasformare la matrice T_0 in una nuova matrice. Si comincia a calcolare per ogni colonna j il minimo su tale colonna

$$d_j^0 = \min_i \ d_{ij};$$

tale valore verrá sottratto ad ogni elemento della colonna j e questo viene fatto per tutte le colonne. Si ottiene quindi una nuova matrice T_1 che nella posizione (i,j) ha il valore $d_{ij} - d_j^0$. La matrice T_1 viene ulteriormente trasformata andando a calcolare il minimo su ogni sua riga i

$$d_i^1 = \min_j \ [d_{ij} - d_j^0]$$

e sottraendo questo ad ogni elemento della riga i. Il risultato è una matrice T_2 che nella posizione (i,j) ha il valore

$$d_{ij}^2 = d_{ij} - d_j^0 - d_i^1 \geq 0.$$

É importante notare che, per come sono stati ottenuti, tutti gli elementi d^2_{ij} sono non negativi. Il passaggio dalla matrice T_0 alla corrispondente T_2 verrá ora mostrato sull'esempio precedentemente introdotto.

Esempio 17 *Nel nostro esempio la matrice T_0 é riportata in (3.14). Si osserva che $d^0_1 = d^0_2 = d^0_3 = d^0_4 = 2$, da cui si ottiene la matrice T_1*

$$T_1 \ : \ \begin{bmatrix} 0 & 1 & 2 & 3 \\ 4 & 0 & 0 & 0 \\ 5 & 0 & 1 & 1 \\ 0 & 1 & 2 & 3 \end{bmatrix}$$

Si osserva che $d^1_1 = d^1_2 = d^1_3 = d^1_4 = 0$, da cui si ottiene la matrice T_2 identica a T_1

$$T_2 \ : \ \begin{bmatrix} 0 & 1 & 2 & 3 \\ 4 & 0 & 0 & 0 \\ 5 & 0 & 1 & 1 \\ 0 & 1 & 2 & 3 \end{bmatrix}$$

NB: il fatto che T_2 sia identica a T_1 <u>non</u> è un fatto generalizzabile.

Torniamo ora alla trattazione generale. Osserviamo che

$$d_{ij} = d^2_{ij} + d^0_j + d^1_i.$$

Andando a sostituire nell'obiettivo al posto di d_{ij} si ottiene

$$\sum_{i=1}^{n}\sum_{j=1}^{n} d_{ij}x_{ij} =$$
$$\sum_{i=1}^{n}\sum_{j=1}^{n} d^2_{ij}x_{ij} + \sum_{j=1}^{n}\sum_{i=1}^{n} d^0_j x_{ij} + \sum_{i=1}^{n}\sum_{j=1}^{n} d^1_i x_{ij} =$$
$$\sum_{i=1}^{n}\sum_{j=1}^{n} d^2_{ij}x_{ij} + \sum_{j=1}^{n} d^0_j \sum_{i=1}^{n} x_{ij} + \sum_{i=1}^{n} d^1_i \sum_{j=1}^{n} x_{ij}.$$

Ogni soluzione ammissibile ha $\sum_{i=1}^{n} x_{ij} = 1$ per ogni j e $\sum_{j=1}^{n} x_{ij} = 1$ per ogni i, quindi l'obiettivo è, per ogni soluzione ammissibile, uguale a

$$\sum_{i=1}^{n}\sum_{j=1}^{n} d^2_{ij}x_{ij} + \sum_{j=1}^{n} d^0_j + \sum_{i=1}^{n} d^1_i.$$

Ponendo

$$D_0 = \sum_{j=1}^{n} d^0_j \qquad D_1 = \sum_{i=1}^{n} d^1_i,$$

si ha infine

$$\sum_{i=1}^{n}\sum_{j=1}^{n} d_{ij}x_{ij} = \sum_{i=1}^{n}\sum_{j=1}^{n} d^2_{ij}x_{ij} + D_0 + D_1.$$

Poichè, come giá osservato, $d_{ij}^2 \geq 0$ per ogni i, j, e poichè si ha anche che $x_{ij} \geq 0$ per ogni i, j, si ha che

$$\sum_{i=1}^{n} \sum_{j=1}^{n} d_{ij} x_{ij} \geq D_0 + D_1,$$

per ogni soluzione ammissibile del problema. A questo punto se trovo una soluzione ammissibile del problema con valore dell'obiettivo pari a $D_0 + D_1$, questa è certamente anche una soluzione ottima. Quindi la domanda che ci poniamo ora è la seguente: esiste o meno una soluzione ammissibile con valore $D_0 + D_1$? In caso di risposta positiva, abbiamo una soluzione ottima del problema, in caso di risposta negativa ci dovremo poi porre la questione di cosa fare se non esiste. La risposta a tale domanda è strettamente legata alla soluzione di questo problema:

Problema 2 *Determinare un sottinsieme Δ di cardinalitá massima degli 0 della matrice T_2 tale che presi due elementi qualsiasi di Δ essi sono* **indipendenti**, *ovvero appartengono a righe e colonne diverse.*

Nel nostro esempio gli 0 nelle posizioni $(1, 1)$ e $(2, 2)$ sono indipendenti, mentre non lo sono gli 0 nelle posizioni $(1, 1)$ e $(4, 1)$ in quanto sono entrambi nella colonna 1. Ci occuperemo in seguito di come si risolve questo problema. Per il momento supponiamo che il problema ammetta una soluzione Δ con $\mid \Delta \mid = n$. Consideriamo allora la seguente soluzione:

$$\overline{x}_{ij} = \begin{cases} 1 & \text{se } (i, j) \in \Delta \\ 0 & \text{altrimenti} \end{cases}$$

Per prima cosa dimostriamo che tale soluzione è ammissibile. Supponiamo per assurdo che non lo sia. Per esempio supponiamo che per qualche j si abbia

$$\sum_{i=1}^{n} \overline{x}_{ij} \neq 1.$$

Sono possibili due casi:

Caso I $\sum_{i=1}^{n} \overline{x}_{ij} = 0$: in tal caso non c'è nessun elemento di Δ nella colonna j. Quindi ve ne dovranno essere n nella restanti $n - 1$ colonne. Ció vuol dire che almeno una colonna contiene due elementi in Δ, ma questo è assurdo in quanto gli elementi di Δ devono essere tra loro indipendenti e quindi non possono appartenere ad una stessa colonna.

Caso II $\sum_{i=1}^{n} \overline{x}_{ij} \geq 2$: in tal caso ci sono due elementi di Δ nella colonna j e si ha una contraddizione identica a quella vista per il Caso I.

In modo del tutto analogo si vede che i vincoli

$$\sum_{j=1}^{n} \overline{x}_{ij} = 1 \quad \forall\, i,$$

non possono essere violati. Quindi la soluzione è ammissibile. Andiamo ora a calcolarne il valore dell'obiettivo. Si nota che le $\overline{x}_{ij}$ sono uguali a 1 solo in corrispondenza di coppie (i, j) per cui si ha $d^2_{ij} = 0$. Quindi

$$\sum_{i=1}^{n}\sum_{j=1}^{n} d_{ij}\overline{x}_{ij} = \sum_{i=1}^{n}\sum_{j=1}^{n} d^2_{ij}\overline{x}_{ij} + D_0 + D_1 = D_0 + D_1.$$

ovvero il valore dell'obiettivo in corrispondenza di tale soluzione ammissibile è $D_0 + D_1$ e per quanto giá osservato la soluzione è ottima.

A questo punto dobbiamo peró porci la questione di come calcolare Δ e, nel caso in cui si abbia $\mid \Delta \mid < n$, come procedere nella soluzione del problema di assegnamento.

Determinazione dell'insieme Δ

Il primo passo consiste nel costruire un grafo *bipartito* nel modo seguente: i due insiemi di vertici sono rispettivamente rappresentati dall'insieme A e dall'insieme B; tra il vertice a_i ed il vertice b_j si traccia un arco (non orientato) se e solo se $d^2_{ij} = 0$. Nel nostro esempio il grafo bipartito sará quello di Figura 3.26. Un insieme indipendente di 0 equivale a un matching su tale grafo bipartito. Infatti, cercare insiemi di 0 nella tabella che non siano mai sulla stessa riga e colonna equivale a cercare insiemi di archi nel grafo bipartito che non abbiano nodi in comune, ovvero equivale a cercare dei matching. Quindi, determinare il massimo insieme di 0 indipendenti equivale a risolvere un problema di matching di cardinalitá massima sul grafo bipartito. Questo lo possiamo fare utilizzando la procedura vista nella sezione 3.5.3.
Se l'insieme Δ ottenuto ha cardinalitá n abbiamo giá visto che possiamo immediatamente trovare una soluzione ottima per il nostro problema. Se si risolve il problema di matching di cardinalitá massima si puó vedere che questo non succede per il nostro esempio dove $n = 4$ ma $\mid \Delta \mid = 3$. Si noti che tale problema coincide esattamente con l'esempio riportato e risolto nella sezione 3.5.3. Resta quindi da stabilire che fare se Δ non ha cardinalitá n. L'obiettivo finale sará quello di giungere ad un'ulteriore trasformazione della matrice T_2. Per arrivare a questa è necessario un passaggio ulteriore in cui si sfrutta l'insieme Δ trovato. Si tratterá di risolvere il seguente problema.

Problema 3 *Determinare un insieme minimo di righe e colonne tali che ricoprendole si ricoprono tutti gli 0 della matrice T_2*

Nel seguito si parlerá genericamente di *linee*, dove una linea puó essere indifferentemente una riga od una colonna. Il problema 3 è strettamente lagato a quello della determinazione dell'insieme Δ. Infatti, consideriamo le etichette ottenute all'ultima iterazione dell'algoritmo per il massimo matching sul grafo bipartito costruito per determinare Δ. Si può dimostrare la seguente

Proprietá 2 *Un ricoprimento ottimo per il problema 3 è formato da esattamente Δ linee ed è costituito da:*

(i) Le righe a_i corrispondenti a nodi non etichettati.

(ii) le colonne b_i corrispondenti a nodi etichettati.

Nel nostro caso, $\{a_2, a_3, b_1\}$ è un ricoprimento ottimo.

Aggiornamento della matrice T_2

Vediamo ora a cosa ci serve il ricoprimento con un numero minimo di linee ottenuto con la procedura appena descritta. Esso ci serve per aggiornare la matrice T_2 e trasformarla in una nuova matrice T_3. La trasformazione avviene seguendo questi passi.

a) Determinare il valore minimo λ tra tutti gli elementi di T_2 non ricoperti da alcuna linea. Si noti che essendo gli 0 di T_2 tutti ricoperti, gli elementi non ricoperti sono tutti positivi e quindi λ stesso è positivo.

b) Gli elementi d_{ij}^3 della nuova matrice T_3 sono definiti in questo modo:

$$d_{ij}^3 = d_{ij}^2 + d_i^3 + d_j^3,$$

dove

$$d_i^3 = \begin{cases} 0 & \text{se la riga } a_i \text{ è nel ricoprimento} \\ -\lambda & \text{altrimenti} \end{cases}$$

e

$$d_j^3 = \begin{cases} \lambda & \text{se la colonna } b_j \text{ è nel ricoprimento} \\ 0 & \text{altrimenti} \end{cases}$$

Sebbene apparentemente complicata, la determinazione degli elementi d_{ij}^3 è semplice se si utilizza la seguente regola:

- gli elementi ricoperti da due linee in T_2 devono essere incrementati di λ;
- gli elementi non ricoperti da alcuna linea vengono decrementati di λ;
- tutti gli altri (gli elementi ricoperti da una sola linea) non cambiano

Da questa regola si vede anche che i soli elementi a cui viene sottratto qualcosa sono quelli non ricoperti e ad essi viene sottratto il minimo di tutti gli elementi non ricoperti. Ció significa che tutti gli elementi d_{ij}^3 di T_3 saranno non negativi come lo erano quelli di T_2.

Ora ricordiamo che il nostro obiettivo era stato riscritto in questo modo:

$$\sum_{i=1}^{n}\sum_{j=1}^{n} d_{ij}x_{ij} = \sum_{i=1}^{n}\sum_{j=1}^{n} d_{ij}^2 x_{ij} + D_0 + D_1.$$

Osserviamo ora che $d_{ij}^2 = d_{ij}^3 - d_i^3 - d_j^3$ ed andando a sostituire otteniamo:

$$\sum_{i=1}^{n}\sum_{j=1}^{n} d_{ij}x_{ij} =$$
$$\sum_{i=1}^{n}\sum_{j=1}^{n} d_{ij}^3 x_{ij} - \sum_{j=1}^{n}\sum_{i=1}^{n} d_j^3 x_{ij} - \sum_{i=1}^{n}\sum_{j=1}^{n} d_i^3 x_{ij} + D_0 + D_1 =$$
$$\sum_{i=1}^{n}\sum_{j=1}^{n} d_{ij}^3 x_{ij} - \sum_{j=1}^{n} d_j^3 \sum_{i=1}^{n} x_{ij} - \sum_{i=1}^{n} d_i^3 \sum_{j=1}^{n} x_{ij} + D_0 + D_1.$$

e quindi, per ogni soluzione ammissibile, si avrá

$$\sum_{i=1}^{n}\sum_{j=1}^{n} d_{ij}x_{ij} = \sum_{i=1}^{n}\sum_{j=1}^{n} d_{ij}^3 x_{ij} - \sum_{j=1}^{n} d_j^3 - \sum_{i=1}^{n} d_i^3 + D_0 + D_1.$$

Vediamo ora di calcolare $\sum_{j=1}^{n} d_j^3$ e $\sum_{i=1}^{n} d_i^3$. Indichiamo con h_1 il numero di righe nel ricoprimento e con h_2 il numero di colonne nel ricoprimento. Si noti che $h_1 + h_2 =\mid \Delta \mid$. Si ha:

$$\sum_{i=1}^{n} d_i^3 = -\lambda \times (\text{numero righe che non sono nel ricoprimento}) = -\lambda(n - h_1),$$

e

$$\sum_{j=1}^{n} d_j^3 = \lambda \times (\text{numero colonne che sono nel ricoprimento}) = \lambda h_2.$$

Quindi

$$\sum_{i=1}^{n}\sum_{j=1}^{n} d_{ij}x_{ij} = \sum_{i=1}^{n}\sum_{j=1}^{n} d_{ij}^3 x_{ij} + \lambda(n - h_1) - \lambda h_2 + D_0 + D_1,$$

da cui

$$\sum_{i=1}^{n}\sum_{j=1}^{n} d_{ij}x_{ij} = \sum_{i=1}^{n}\sum_{j=1}^{n} d_{ij}^3 x_{ij} + \lambda(n- \mid \Delta \mid) + D_0 + D_1.$$

A questo punto ci ritroviamo nella stessa situazione in cui ci si è trovati in partenza con T_2. Dal momento che $d_{ij}^3 \geq 0$ e $x_{ij} \geq 0$ si ha che un limite inferiore per il problema di assegnamento è $D_0 + D_1 + \lambda(n- \mid \Delta \mid)$. Se riesco a trovare un assegnamento che ha come valore dell'obiettivo proprio tale limite inferiore ho determinato una soluzione ottima. La verifica se un tale assegnamento esista puó essere fatto cercando un sottinsieme indipendente di cardinalitá massima in T_3, esattamente come si era fatto in T_2. Se tale sottinsieme ha cardinalitá n si è trovato un assegnamento ottimo, altrimenti sfruttando tale sottinsieme e passando attraverso la determinazione di un insieme minimo delle linee di ricoprimento di T_3 si determina una nuova matrice T_4 e si itera in questo modo la

procedura fino a che si è determinato un assegnamento ottimo. Ci si puó chiedere se la procedura termina oppure no, cioè se si arriva infine ad una matrice T_h che contiene un sottinsieme di 0 indipendenti a due a due di cardinalitá n. Nel caso in cui tutti i d_{ij} siano interi si ha certamente terminazione finita. Lo si puó vedere da come aumentano le limitazioni inferiori ad ogni iterazione. Con T_2 avevamo una limitazione inferiore per la soluzione ottima pari a $D_0 + D_1$; con T_3 si è passati ad una limitazione inferiore pari a $D_0 + D_1 + \lambda(n - \mid \Delta \mid)$. La limitazione inferiore cresce quindi almeno di una quantitá pari a $\lambda > 0$. Nel caso in cui i d_{ij} siano interi λ deve essere anch'esso un intero e quindi è certamente maggiore o uguale a 1. Se per assurdo la procedura dovesse essere iterata infinite volte, la limitazione inferiore stessa crescerebbe all'infinito ma questo è assurdo in quanto un qualsiasi assegnamento (ad esempio l'assegnamento (a_i, b_i) per ogni $i \in \{1, \ldots, n\}$) ha un valore finito ed il minimo di tali assegnamenti non puó quindi crescere all'infinito. Qui ci siamo limitati a dimostare che la procedura termina in un numero finito di iterazioni. In realtá si puó dimostrare che essa richiede un numero $O(n^3)$ di operazioni ed é quindi una procedura di complessitá polinomiale, a ulteriore conferma dell'appartenenza alla classe P del problema di assegnamento.

Ora vediamo di ottenere la matrice T_3 per il nostro esempio. Innanzitutto osserviamo che il valore minimo λ in T_2 tra quelli non ricoperti è pari a 1. Quindi sottraendo 1 a tutti gli elementi non ricoperti e sommandolo a tutti quelli ricoperti due volte si ottiene la nuova matrice T_3

T_3 :

	b_1	b_2	b_3	b_4
a_1	0	0	1	2
a_2	5	0	0	0
a_3	6	0	1	1
a_4	0	0	1	2

A questo punto si tratta di costruire, a partire dalla matrice T_3 un grafo bipartito per determinare un sottinsieme indipendente di 0 in T_3. Si noti che non occorre utilizzare la procedura di inizializzazione per determinare un insieme indipendente iniziale. Infatti, si puó dimostrare che gli 0 dell'insieme Δ dell'iterazione precedente sono sempre presenti in T_3 e si possono quindi utilizzare questi come insieme indipendente iniziale (o, equivalentemente, come matching iniziale sul grafo bipartito). L'esempio non viene ulteriormente risolto e viene lasciato come esercizio.

Capitolo 4

Algoritmi esatti per problemi *NP*-completi

Per problemi difficili come il problema $KNAPSACK$ o il problema TSP sappiamo, in base ai risultati presentati in precedenza, che é molto improbabile essere in grado di risolvere istanze del problema di dimensioni molto elevate in tempi ragionevoli. Tuttavia questo non vuol dire che si deve sempre rinunciare a risolvere tali problemi in modo esatto. Esistono algoritmi esatti per tali problemi che sono in grado di risolvere anche istanze di dimensioni tutt'altro che banali. Con algoritmi particolarmente sofisticati e con una elevata potenza di calcolo si sono risolti anche problemi TSP con 15.000 nodi. Tipicamente questi algoritmi si basano sul concetto di *enumerazione implicita* delle soluzioni, utilizzano cioé metodi che consentono di scartare sottinsiemi di elementi della regione ammissibile senza dover valutare esplicitamente per ciascuno di essi la funzione obiettivo, avendo stabilito che in tali sottinsiemi non vi possono essere soluzioni migliori rispetto alla miglior soluzione nota. In questo capitolo vedremo due classi di algoritmi esatti: gli algoritmi *branch-and-bound* e quelli di *programmazione dinamica*. Per illustrare tali algoritmi, mostreremo un algoritmo branch-and-bound applicato al problema $KNAPSACK$ e uno applicato al problema TSP, mentre per la programmazione dinamica mostreremo un esempio applicato al problema $KNAPSACK$. Prima peró di addentrarci nella descrizione di questi algoritmi introdurremo i modelli matematici dei due problemi NP-completi che abbiamo incontrato, il problema $KNAPSACK$ e il problema TSP.

4.1 Il problema $KNAPSACK$ e il suo modello matematico

In questa sezione vogliamo introdurre il modello matematico del problema $KNAPSACK$. Associamo ad ogni oggetto i una variabile binaria x_i tale che

$$x_i = \begin{cases} 1 & \text{se l'oggetto } i \text{ viene messo nello zaino} \\ 0 & \text{altrimenti} \end{cases}$$

Si avrá dunque $x_i \in \{0, 1\}$. L'unico vincolo nel problema $KNAPSACK$ é quello che il peso complessivo degli oggetti inseriti nello zaino non deve superare la capacitá dello zaino e quindi:

$$\sum_{i=1}^{n} p_i x_i \leq b.$$

Il valore contenuto nello zaino é dato da:

$$\sum_{i=1}^{n} v_i x_i,$$

e tale valore é da massimizzare. Il problema é quindi un problema di PLI (in particolare di PL binaria, essendo tutte le variabili binarie) formulato in questo modo:

$$\begin{array}{ll} \max & \sum_{i=1}^{n} v_i x_i \\ & \sum_{i=1}^{n} p_i x_i \leq b \\ & x_i \in \{0, 1\} \qquad \forall\ i \in \{1, \ldots, n\} \end{array} \tag{4.1}$$

Si introduce ora un particolare esempio che verrá anche utilizzato in seguito.

Esempio 18 *Se ci riferiamo al problema $KNAPSACK$ dell'Esempio 2 con $n = 4$, il modello matematico del problema é il seguente:*

$$\begin{array}{ll} \max & 8x_1 + 6x_2 + 10x_3 + x_4 \\ & 7x_1 + 7x_2 + 13x_3 + 4x_4 \leq 16 \\ & x_1, x_2, x_3, x_4 \in \{0, 1\} \end{array} \tag{4.2}$$

Si noti l'estrema semplicitá del modello per il problema di $KNAPSACK$ dove appare un solo vincolo, che fa da contrasto con la difficoltá del problema stesso.

4.2 Il problema TSP e il suo modello matematico

Il modello matematico per il problema TSP é decisamente piú complesso. Dato il grafo completo orientato $G = (V, A)$, Associamo ad ogni arco $(i, j) \in A$ la

variabile x_{ij} che potrá assumere i seguenti valori:

$$x_{ij} = \begin{cases} 1 & \text{se l'arco } (i,j) \text{ fa parte del circuito hamiltoniano} \\ 0 & \text{altrimenti} \end{cases}$$

Quindi $x_{ij} \in \{0,1\}$. Vediamo ora quali vincoli devono essere soddisfatti dai circuiti hamiltoniani. Prendiamo in esame la Figura 4.1 dove gli archi tratteggiati rappresentano il circuito hamiltoniano

$$1 \to 3 \to 2 \to 4 \to 1.$$

sul grafo completo con insieme di nodi $V = \{1,2,3,4\}$. Si nota che in ogni

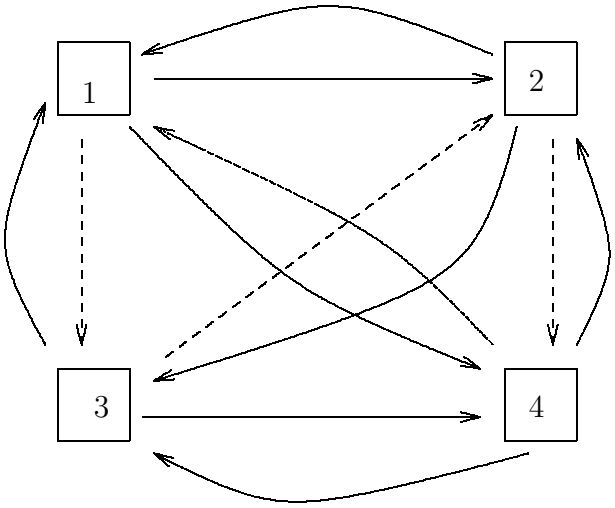


Figura 4.1: Un circuito hamiltoniano rappresentato dagli archi tratteggiati.

vertice c'e' esattamente un arco che entra nel vertice ed esattamente uno che esce dal vertice. Avremo quindi i seguenti due insiemi di vincoli:

1. Per ogni vertice $j \in V$:
$$\sum_{i \in V,\ i \neq j} x_{ij} = 1,$$
ovvero il circuito puó contenere uno ed un solo arco entrante in j.

2. Per ogni vertice $j \in V$:
$$\sum_{i \in V,\ i \neq j} x_{ji} = 1,$$
ovvero il circuito puó contenere uno ed un solo arco uscente da j.

Nel nostro esempio con 4 vertici avremo i seguenti 8 vincoli

$$x_{12} + x_{13} + x_{14} = 1 \quad x_{21} + x_{23} + x_{24} = 1$$

$$x_{31} + x_{32} + x_{34} = 1 \quad x_{41} + x_{42} + x_{43} = 1$$
$$x_{21} + x_{31} + x_{41} = 1 \quad x_{12} + x_{32} + x_{42} = 1$$
$$x_{13} + x_{23} + x_{43} = 1 \quad x_{14} + x_{24} + x_{34} = 1.$$

Tutti i circuiti hamiltoniani soddisfano questi vincoli. Ma é vero anche il viceversa? Cioé é vero che tutte le soluzioni che soddisfano tali vincoli rappresentano circuiti hamiltoniani? La risposta é no e lo si mostra attraverso il nostro esempio. Prendiamo la soluzione

$$x_{12} = x_{21} = x_{34} = x_{43} = 1$$
$$x_{13} = x_{31} = x_{14} = x_{41} = x_{23} = x_{32} = x_{24} = x_{42} = 0.$$

Questa non rappresenta un circuito hamiltoniano come si vede dalla Figura 4.2, ma si puó verificare che soddisfa i vincoli introdotti (in ogni vertice entra ed esce un solo arco). Ció si verifica per la presenza di sottocircuiti (nell'esempio

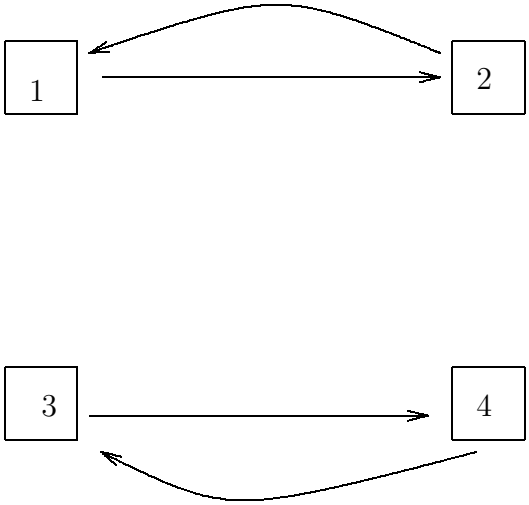


Figura 4.2: Una soluzione non ammissibile rappresentata da due sottocircuiti.

$1 \to 2 \to 1$ e $3 \to 4 \to 3$). Per avere un insieme di vincoli che rappresenta tutti e soli i circuiti hamiltoniani dobbiamo aggiungere ai vincoli giá introdoti altri vincoli. Tali nuovi vincoli sono i seguenti:

$$\forall\, U \subseteq V :\ 2 \leq \mid U \mid \leq \mid V \mid -2 \qquad \sum_{i \in U,\ j \in V \setminus U} x_{ij} \geq 1.$$

Tali vincoli richiedono che per ogni possibile partizione di V in due sottinsiemi (ciascuno di cardinalitá almeno pari a 2), deve esserci almeno un arco che va da un sottinsieme all'altro. Questo esclude i sottocircuiti come si puó constatare dal nostro esempio: tra $U = \{1, 2\}$ e $V \setminus U = \{3, 4\}$ non si ha alcun arco.

Vediamo ora di scrivere tali vincoli per il nostro esempio. Notiamo innanzitutto che, essendo $\mid V \mid = 4$, si avrá che gli insiemi U da considerare sono i sottinsiemi con cardinalitá che soddisfa $2 \leq \mid U \mid \leq 4 - 2 = 2$ e quindi sono i soli sottinsiemi con cardinalitá 2. Ognuno di questi dará origine ad un vincolo e quindi ai vincoli giá introdotti dovremo aggiungere i seguenti:

$$\begin{aligned}
U = \{1,2\} &\rightarrow \quad x_{13} + x_{14} + x_{23} + x_{24} \geq 1 \\
U = \{1,3\} &\rightarrow \quad x_{12} + x_{14} + x_{32} + x_{34} \geq 1 \\
U = \{1,4\} &\rightarrow \quad x_{12} + x_{13} + x_{42} + x_{43} \geq 1 \\
U = \{2,3\} &\rightarrow \quad x_{21} + x_{24} + x_{31} + x_{34} \geq 1 \\
U = \{2,4\} &\rightarrow \quad x_{21} + x_{23} + x_{41} + x_{43} \geq 1 \\
U = \{3,4\} &\rightarrow \quad x_{31} + x_{32} + x_{41} + x_{42} \geq 1
\end{aligned}$$

Vediamo ancora come la soluzione con sottocircuiti del nostro esempio é scartata in quanto:

$$x_{13} + x_{14} + x_{23} + x_{24} = 0 < 1,$$

e quindi abbiamo un vincolo violato.

Riassumendo, l'insieme di vincoli che definisce tutti e soli i circuiti hamiltoniani é il seguente:

$$\begin{array}{cc}
\sum_{i \in V,\ i \neq j} x_{ij} = 1 & \forall\ j \in V \\
\sum_{i \in V,\ i \neq j} x_{ji} = 1 & \forall\ j \in V \\
\sum_{i \in U,\ j \in V \setminus U} x_{ij} \geq 1 & \forall\ U \subseteq V :\ 2 \leq \mid U \mid \leq \mid V \mid -2 \\
x_{ij} \in \{0,1\} & \forall\ i,j \in \{1,\ldots,n\},\ i \neq j
\end{array} \tag{4.3}$$

Tra tutti i circuiti hamiltoniani noi cerchiamo quello con valore minimo e quindi l'obiettivo del problema sará il seguente:

$$\min \sum_{i,j \in V,\ i \neq j} v_{ij} x_{ij}.$$

Nel nostro esempio non abbiamo ancora fissato i valori degli archi. Assegniamo allora agli archi i seguenti valori:

$$v_{12} = 1 \quad v_{13} = 6 \quad v_{14} = 7 \quad v_{23} = 3 \quad v_{24} = 6 \quad v_{34} = 5$$

$$v_{21} = 1 \quad v_{31} = 2 \quad v_{41} = 5 \quad v_{32} = 4 \quad v_{42} = 3 \quad v_{43} = 6$$

L'obiettivo sará dunque il seguente:

$$\begin{aligned}
\min \quad & x_{12} + 6x_{13} + 7x_{14} + 3x_{23} + 6x_{24} + 5x_{34} + \\
& x_{21} + 2x_{31} + 5x_{41} + 4x_{32} + 3x_{42} + 6x_{43}.
\end{aligned}$$

4.3 Branch-and-bound

Daremo una descrizione di un generico algoritmo branch-and-bound e, di seguito, un esempio di tale algoritmo applicato al problema $KNAPSACK$ e uno applicato al problema TSP. Descriveremo l'algoritmo generico di branch-and-bound per problemi di massimo. Di seguito segnaleremo le minime variazioni che vanno introdotte per problemi di minimo. Cominceremo descrivendo le singole componenti di un algoritmo branch-and-bound.

Calcolo di un upper bound

Data la regione ammissibile S dell'istanza di un problema di ottimizzazione (qui e nel seguito, per semplicitá di notazione, l'indice I dell'istanza viene omesso), supponiamo di avere un sottinsieme $T \subseteq S$. Una limitazione superiore o *upper bound* per T é un valore $U(T)$ con la seguente proprietá

$$U(T) \geq f(x) \quad \forall\, x \in T.$$

Il valore $U(T)$ viene calcolato tramite una procedura che deve avere come proprietá quella di poter essere eseguita in tempi brevi (in particolare, il calcolo degli upper bound deve richiedere un tempo molto inferiore rispetto al tempo necessario per risolvere l'intero problema). Spesso la scelta di una procedura per il calcolo del'upper bound é fortemente legata al particolare problema che si sta risolvendo. Inoltre, non esiste un'unica procedura per un dato problema.

Un modo ampiamente utilizzato per determinare un upper bound $U(T)$ (o un lower bound $L(T)$ per problemi di minimo), é quello di determinare la soluzione di un suo rilassamento. Indichiamo con:

$$\alpha(f, T) = \max_{x \in T} f(x), \tag{4.4}$$

il valore ottimo della funzione f sull'insieme T. Si definisce rilassamento del problema (4.4), un problema:

$$\alpha(f', T') = \max_{x \in T'} f'(x) \tag{4.5}$$

dove:

$$T \subseteq T', \tag{4.6}$$

e

$$f'(x) \geq f(x) \quad \forall\, x \in T \tag{4.7}$$

(per problemi di minimo si deve invertire il verso della diseguaglianza). Vale la seguente osservazione che dimostra che il valore ottimo del rilassamento (4.5) sia un upper bound $U(T)$ per il valore ottimo del problema (4.4).

Osservazione 7 *Si ha che:*

$$\alpha(f', T') \geq \alpha(f, T).$$

Dimostrazione Sia $x^* \in T$ una soluzione ottima del problema (4.4), cioé:

$$f(x^*) = \alpha(f, T),$$

e sia $x' \in T'$ una soluzione ottima del problema (4.5), cioé:

$$f(x') = \alpha(f', T').$$

A causa di (4.6) si ha che $x^* \in T$ implica $x^* \in T'$. Inoltre, come conseguenza di (4.7), si ha:

$$f'(x^*) \geq f(x^*).$$

Infine, l'ottimalitá di x' implica $f'(x') \geq f'(x^*)$ e quindi:

$$\alpha(f', T') = f'(x') \geq f'(x^*) \geq f(x^*) = \alpha(f, T),$$

come si voleva dimostrare.

Come giá osservato, il calcolo dell'upper bound tramite la risoluzione del rilassamento (4.5) del problema (4.4) deve avere la proprietá di essere risolvibile in tempi molto piú rapidi rispetto al problema (4.4). Esistono molti possibili rilassamenti di un problema. Tra questi, uno che é giá stato incontrato é il *rilassamento lineare* per problemi di PLI. Dato il generico problema di PLI:

$$\begin{array}{lll} \max & cx & \\ & Ax \leq b & \\ & x \geq 0 \quad x \in Z^n, & \end{array} \tag{4.8}$$

sappiamo che questo é un particolare problema di ottimizzazione combinatoria con:

$$f(x) = cx \quad T = \{x \in Z^n \ : \ Ax \leq b, \ x \geq 0\}.$$

Il rilassamento lineare di tale problema é un problema della forma (4.5) con:

$$f'(x) \equiv f(x) \quad T' = \{x \in R^n \ : \ Ax \leq b, \ x \geq 0\},$$

dove si usa la stessa funzione obiettivo (il che rende banalmente vero (4.7)) ma nella regione ammissibile T' si accettano anche gli eventuali punti a coordinate non intere oltre a quelli in T e quindi la condizione (4.6) é soddisfatta. Quindi il rilassamento lineare coincide con il seguente problema di PL:

$$\begin{array}{lll} \max & cx & \\ & Ax \leq b & \\ & x \geq 0 & \end{array} \tag{4.9}$$

Si noti che, come richiesto, il rilassamento lineare, essendo un problema di PL, é risolvibile in tempi molto piú rapidi dell'originario problema (4.8) di PLI.

Un altro possibile rilassamento è il *rilassamento lagrangiano*. Supponiamo che il nostro problema sia formulato come problema di PLI:

$$\begin{array}{ll} \max & cx \\ & Ax \leq b \\ & Cx \leq d \\ & x \geq 0 \quad x \in Z^n. \end{array}$$

Quindi con:

$$f(x) = cx, \quad T = \{x \in Z^n \ : \ Ax \leq b, \ Cx \leq d, \ x \geq 0\}.$$

Supponiamo che i vincoli $Ax \leq b$ siano "facili" (ad esempio, A è TU e b è a coordinate tutte intere). Qunidi eliminando i vincoli "difficili" $Cx \leq d$ resta un problema di PLI facile da risolvere (basta risolverne il rilassamento lineare). Per eliminarli li spostiamo nell'obiettivo. Dato un vettore $\lambda \geq 0$, detto *vettore dei moltiplicatori di Lagrange*, delle stesse dimensioni di d, il rilassamento lagrangiano è il seguente:

$$\begin{array}{lll} u(\lambda) = \max & cx + \lambda(d - Cx) & \\ & Ax \leq b & \\ & x \geq 0 & x \in Z^n. \end{array}$$

con

$$f'(x) = cx + \lambda(d - Cx)$$

e

$$T' = \{x \in Z^n \ : \ Ax \leq b, \ x \geq 0\}.$$

Ovviamente, $T \subseteq T'$. Inoltre, per ogni $x \in T$ si ha che:

$$Cx \leq d \ \Rightarrow \ \forall \ \lambda \geq 0 : \ \lambda(d - Cx) \geq 0 \ \Rightarrow \ f'(x) \geq f(x).$$

Quindi sono soddisfatte le due condizioni che devono essere soddisfatte da un rilassamento. Notiamo infine che nel rilassamento lagrangiano rimangono solo i vincoli "facili" e quindi esso può essere risolto in tempo polinomiale, come viene richiesto per il calcolo di un upper bound. Notiamo anche che ad ogni $\lambda \geq 0$ distinto corisponde un diverso upper bound $u(\lambda)$. Per ottenere il miglior upper bound possibile (ovvero il più piccolo), possiamo risolvere questo ulteriore problema:

$$\min_{\lambda \geq 0} u(\lambda)$$

detto *duale lagrangiano*.

Un caso particolare di rilassamento lagrangiano si ha prendendo tutti i moltiplicatori di Lagrange nulli, cioè ponendo $\lambda = 0$. Questo coincide con il rilassamento ottenuto semplicemente omettendo dal problema i vincoli "difficili".

Osserviamo anche che in alcuni casi i vincoli "difficili" del problema sono vincoli di uguaglianza

$$Cx = d.$$

In tal caso, il rilassamento lagrangiano si definisce nello stesso modo ma i moltiplicatori di Lagrange relativi ai vincoli di uguaglianza non sono vincolati ad assumere solo valori non negativi ma possono assumere anche valori negativi.

Calcolo del lower bound

Vogliamo ora calcolare un limite inferiore o *lower bound* per il valore ottimo del nostro problema, ovvero un valore LB con la proprietá che

$$LB \leq f(x^*) = \max_{x \in S} f(x).$$

Si puó notare che se prendiamo un qualsiasi elemento $\overline{x} \in S$ e valutiamo in esso la funzione f, il valore $f(\overline{x})$ é giá un lower bound, dal momento che $f(\overline{x}) \leq f(x^*)$. Durante l'esecuzione di un algoritmo branch-and-bound la funzione f viene valutata per molti elementi $y_1, \ldots, y_h \in S$ e per ognuno di essi si ha

$$f(y_i) \leq f(x^*) \qquad i = 1, \ldots, h.$$

Quindi, come lower bound possiamo prendere il massimo dei valori f per tali elementi, cioé:

$$LB = \max\{f(y_i) \ : \ i = 1, \ldots, h\} \leq f(x^*).$$

Resta da chiarire da dove ricaviamo gli elementi di S in cui valutare la funzione f durante l'esecuzione dell'algoritmo. Se si ha a disposizione un'euristica (si veda il Capitolo 5)

é buona norma valutare f nel risultato di tale euristica. Inoltre, durante lo stesso calcolo degli upper bound si puó individuare uno o piú elementi di S e valutare in essi f. Ad esempio, se si calcola l'upper bound $U(T)$ tramite un rilassamento, nei casi in cui per la soluzione $x' \in T' \supseteq T$ valga anche $x' \in T$, allora si ha anche $x' \in S$ e si puó valutare f in x'. In altri casi non si ha $x' \in T$ ma con opportune operazioni (quali arrotondamenti o approssimazioni per eccesso/difetto di valori di variabili) si puó determinare partendo da $x' \notin T$ una soluzione $\overline{x}' \in T$ (un esempio di ció lo incontreremo nell'algoritmo branch-and-bound per il problema dello zaino).

Branching

L'operazione di branching consiste nel rimpiazzare un insieme $T \subseteq S$ con una sua partizione $T_1, \ldots, T_m$. Si ricordi che $T_1, \ldots, T_m$ formano una partizione di T se

$$T = \cup_{i=1}^{m} T_i \quad T_i \cap T_j = \emptyset \ \ \forall\, i \neq j.$$

La partizione puó essere rappresentata tramite una struttura ad albero come in Figura 4.3: l'insieme T é un nodo dell'albero da cui partono i rami (da qui il nome branching) verso i nodi della partizione, che vengono anche detti nodi successori o nodi figli del nodo T.

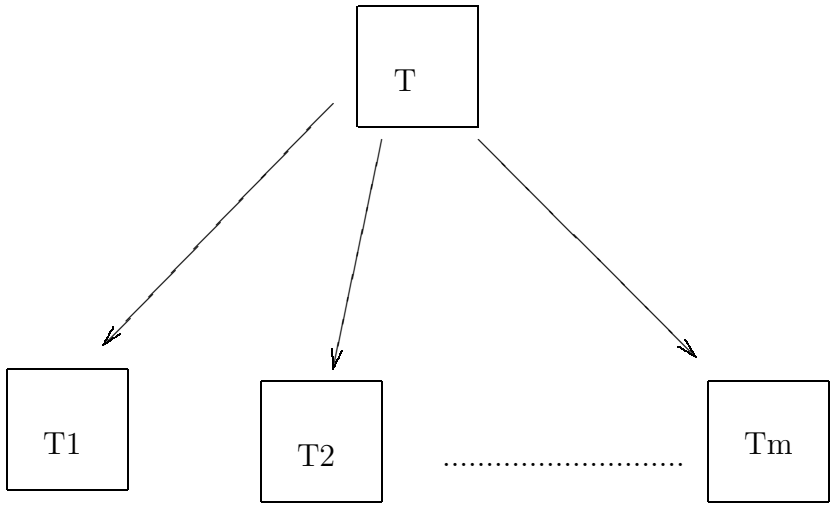


Figura 4.3: Il branching di un nodo T.

Cancellazione di sottinsiemi

Veniamo ora al punto chiave degli algoritmi di branch-and-bound, ovvero la cancellazione di sottinsiemi. Supponiamo che per un dato sottinsieme, T_2 ad esempio, si abbia

$$U(T_2) \leq LB.$$

Ma questo vuol dire che

$$\forall\ x \in T_2 \quad f(x) \leq U(T_2) \leq LB,$$

e cioé tra tutti gli elementi in T_2 non ne possiamo trovare alcuno con valore di f superiore a LB, ovvero al miglior valore di f osservato fino a questo momento. A questo punto posso *cancellare* il sottinsieme T_2 (si veda la Figura 4.4). In questo senso si parla di *enumerazione implicita*: il confronto tra upper bound $U(T_2)$ del sottinsieme e lower bound LB ci consente di scartare tutti gli elementi in T_2 *senza dover calcolare la funzione f in essi*.

L'algoritmo branch-and-bound

Siamo ora pronti per mettere insieme le componenti descritte sopra e formulare il generico algoritmo di branch-and-bound.

Passo 1 Si ponga $\mathcal{C} = \{S\}$ e $\mathcal{Q} = \emptyset$ (l'insieme $\mathcal{C}$ conterrá sempre i sottinsiemi ancora da tenere in considerazione e inizialmente contiene l'intero insieme S, mentre l'insieme $\mathcal{Q}$, inizialmente vuoto, conterrá tutti i sottinsiemi cancellati). Si ponga $k = 1$. Si calcoli $U(S)$ e si calcoli un valore per LB (eventualmente utilizzando anche i risultati di un'euristica, se disponibile).

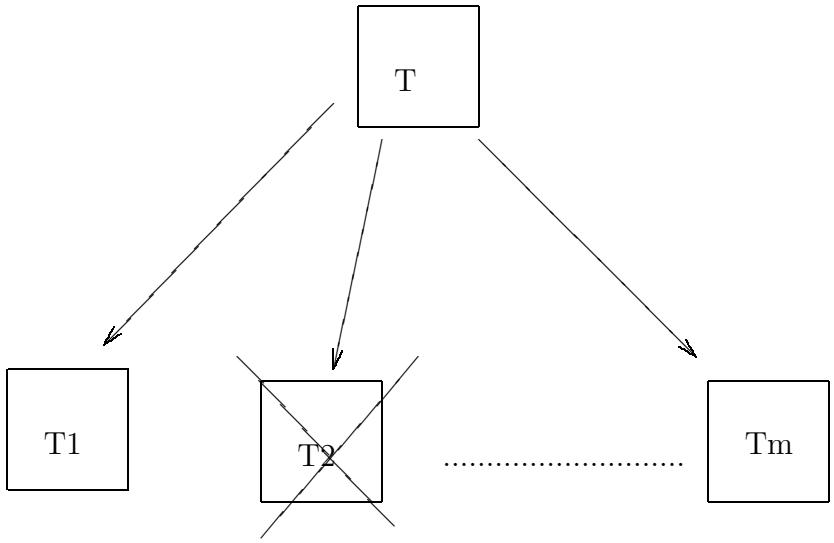


Figura 4.4: La cancellazione del nodo T_2.

Passo 2 (Selezione di un sottinsieme) Si selezioni un sottinsieme $T \in \mathcal{C}$. Tra le varie regole di selezione citiamo qui quella di selezionare il sottinsieme T in $\mathcal{C}$ con il valore di upper bound piú elevato, cioé

$$U(T) = \max_{Q \in \mathcal{C}} U(Q).$$

Passo 3 (Branching) Si sostituisca l'insieme T in $\mathcal{C}$ con la sua partizione in m_k sottinsiemi $T_1, \ldots, T_{m_k}$, ovvero

$$\mathcal{C} = \mathcal{C} \cup \{T_1, \ldots, T_{m_k}\} \setminus \{T\}.$$

Passo 4 (Upper bounding) Si calcoli un upper bound $U(T_i)$, $i = 1, \ldots, m_k$ per ogni sottinsieme della partizione.

Passo 5 (Lower bounding) Si aggiorni, eventualmente, il valore LB (si ricordi che il valore LB corrsiponde sempre al massimo dei valori di f osservati durante l'esecuzione dell'algoritmo).

Passo 6 (Cancellazione sottinsiemi) Si escludano da $\mathcal{C}$ tutti i sottinsiemi Q per cui $U(Q) \leq LB$, ovvero

$$\mathcal{C} = \mathcal{C} \setminus \{Q : \ U(Q) \leq LB\}.$$

e si trasferiscano tali sottinsiemi in $\mathcal{Q}$, cioé:

$$\mathcal{Q} = \mathcal{Q} \cup \{Q : \ U(Q) \leq LB\}.$$

Passo 7 Se $\mathcal{C} = \emptyset$: stop, il valore LB coincide con il valore ottimo $f(x^*)$. Altrimenti si ponga $k = k + 1$ e si ritorni al Passo 2.

Va fatto un breve commento circa il Passo 7. Questo dice che se $\mathcal{C} = \emptyset$ si ha che LB é il valore ottimo del nostro problema. Questa affermazione é una conseguenza del fatto che, nel momento in cui $\mathcal{C} = \emptyset$, tutti i sottinsiemi cancellati fino a quel momento, cioé la collezione $\mathcal{Q}$ di sottinsiemi, formano una *partizione dell'intero insieme* S. Quindi tra di essi ve ne é certamente uno, indicato con $T^* \in \mathcal{Q}$, che contiene x^*. Ma poiché T^* é stato cancellato si dovrá avere

$$f(x^*) \leq U(T^*) \leq LB \leq f(x^*),$$

da cui segue immediatamente che $LB = f(x^*)$.

Infine, dobbiamo brevemente commentare le modifiche da apportare per trattare problemi di minimo. In tali problemi si dovranno semplicemente invertire i ruoli di upper e lower bound: ad un sottinsieme $Q \subseteq S$ dovrá essere associato un valore di lower bound $L(Q)$; al posto del valore LB avremo un valore UB con la proprietá

$$UB \geq f(x^*) = \min_{x \in S} f(x).$$

Il valore UB sar'a il minimo tra i valori osservati della funzione obiettivo in punti della regione ammissibile S. Il sottinsieme Q viene cancellato se é vero che $L(Q) \geq UB$. Al Passo 2 della procedura di branch-and-bound si seleziona un nodo con lower bound pi'u piccolo, ovvero un nodo T tale che

$$L(T) = \max_{Q \in \mathcal{C}} L(Q).$$

4.3.1 Un algoritmo branch-and-bound per il problema $KNAPSACK$

Vediamo ora un esempio di algoritmo branch-and-bound per il problema $KNAPSACK$. Rispetto allo schema generale visto in precedenza dobbiamo solo specificare alcuni particolari come il calcolo dell'upper bound e la regola per effettuare il branching. Per prima cosa descriveremo una procedura per il calcolo di un upper bound $U(S)$ per il valore ottimo del problema su tutta la regione ammissibile S. Di seguito specializzeremo la procedura a sottinsiemi T della regione ammissibile con una particolare struttura.

Calcolo di un upper bound $U(S)$

L'upper bound $U(S)$ si calcola risolvendo il rilassamento lineare del problema originale, ovvero il problema originale senza i vincoli di interezza:

$$\begin{array}{lll} \max & \sum_{i=1}^{n} v_i x_i & \\ & \sum_{i=1}^{n} p_i x_i \leq b & \\ & 0 \leq x_i \leq 1 & \forall\ i \in \{1, \ldots, n\} \end{array}$$

Questo é un problema di PL, ma é di forma molto semplice e non abbiamo bisogno di scomodare l'algoritmo del simplesso per risolverlo. Si puó utilizzare la seguente procedura.

1. Supponiamo, senza perdita di generalitá, che gli oggetti siano ordinati in modo non crescente rispetto ai rapporti valore/peso $\frac{v_i}{p_i}$, cioé si abbia che:

$$\frac{v_1}{p_1} \geq \frac{v_2}{p_2} \geq \cdots \geq \frac{v_n}{p_n}.$$

2. Si calcolino i valori

$$\begin{aligned} &b - p_1 \\ &b - p_1 - p_2 \\ &b - p_1 - p_2 - p_3 \\ &\vdots \end{aligned}$$

   fino ad arrivare al primo valore negativo

$$b - \sum_{j=1}^{r+1} p_j$$

   se vi si arriva (se non vi si arriva vuol dire semplicemente che tutti gli oggetti possono essere messi nello zaino e la soluzione ottima del problema é proprio quella di mettere tutti gli oggetti nello zaino).

3. La soluzione ottima del problema lineare é la seguente

$$\begin{aligned} &x_1 = x_2 = \cdots = x_r = 1 \\ &x_{r+1} = \frac{b - \sum_{j=1}^{r} p_j}{p_{r+1}} \\ &x_{r+2} = x_{r+3} = \cdots = x_n = 0 \end{aligned} \tag{4.10}$$

   ed ha il valore ottimo

$$\sum_{j=1}^{r} v_j + v_{r+1} \frac{b - \sum_{j=1}^{r} p_j}{p_{r+1}}. \tag{4.11}$$

Quindi, come upper bound $U(S)$ del valore ottimo del problema originale si puó usare il valore (4.11).
Notiamo anche che la soluzione

$$\begin{aligned} &x_1 = x_2 = \cdots = x_r = 1 \\ &x_{r+1} = x_{r+2} = x_{r+3} = \cdots = x_n = 0 \end{aligned} \tag{4.12}$$

ottenuta approssimando per difetto il valore dell'unica variabile (la x_{r+1}) che puó avere coordinate non intere nella soluzione del rilassamento lineare, é appartenente a S (il peso dei primi r oggetti non supera la capacitá dello zaino). Quindi tale soluzione puó essere utilizzata per il calcolo del lower bound LB. Nel seguito vedremo di estendere la procedura per il calcolo dell'upper bound anche a particolari sottinsiemi della regione ammissibile S.

Calcolo dell'upper bound su sottinsiemi di S di forma particolare

Descriviamo ora una procedura per il calcolo dell'upper bound su sottinsiemi di S con forma speciale e cioé

$$\begin{aligned} S(I_0, I_1) = \quad \{N \in S : \quad & \text{l'oggetto } i \notin N \ \forall\ i \in I_0, \\ & \text{l'oggetto } i \in N \ \forall\ i \in I_1\} \end{aligned} \tag{4.13}$$

dove $I_0, I_1 \subseteq \{1, \ldots, n\}$ e $I_0 \cap I_1 = \emptyset$. In altre parole $S(I_0, I_1)$ contiene tutti gli elementi di S che non contengono gli oggetti in I_0 e contengono gli oggetti in I_1. Possono invece indifferentemente contenere o non contenere gli oggetti nell'insieme

$$I_f = \{i_1, \ldots, i_k\} = \{1, \ldots, n\} \setminus (I_0 \cup I_1),$$

con $i_1 < \cdots < i_k$. Come in precedenza, si suppone sempre che tutti gli oggetti siano stati ordinati in modo non crescente rispetto al rapporto valore/peso $\frac{v_i}{p_i}$. Vogliamo ora calcolare $U(S(I_0, I_1))$. Il nostro originario problema $KNAPSACK$ ristretto al sottinsieme $S(I_0, I_1)$ si presenta nella seguente forma:

$$\begin{aligned} \max \quad & \textstyle\sum_{i \in I_1} v_i + \sum_{i \in I_f} v_i x_i \\ & \textstyle\sum_{i \in I_f} p_i x_i \leq b - \sum_{i \in I_1} p_i \\ & x_i \in \{0, 1\} \qquad\qquad \forall\ i \in I_f \end{aligned} \tag{4.14}$$

Possiamo notare che si tratta ancora di un problema di tipo $KNAPSACK$ dove é presente una quantitá costante nell'obiettivo ($\sum_{i \in I_1} v_i$), dove lo zaino ha ora capacitá $b - \sum_{i \in I_1} p_i$ e dove l'insieme di oggetti in esame é ora ristretto ai soli oggetti in I_f. Trattandosi ancora di un problema dello zaino, possiamo applicare ad esso la stessa procedura che abbiamo adottato per trovare l'upper bound $U(S)$. Tale procedura verrá ora descritta nel seguito. Come output essa fornirá oltre all'upper bound $U(S(I_0, I_1))$, anche una soluzione appartenente a $S(I_0, I_1)$ utilizzabile per il calcolo del lower bound LB.

Calcolo upper bound $U(S(I_0, I_1))$

Passo 1 Se $b - \sum_{i \in I_1} p_i < 0$, il nodo non contiene soluzioni ammissibili (gli oggetti in I_1 hanno giá un peso superiore alla capacitá b dello zaino). In tal caso si pone

$$U(S(I_0, I_1)) = -\infty$$

Altrimenti, si sottraggano successivamente a $b - \sum_{i \in I_1} p_i$ i pesi degli oggetti in I_f

$$b - \sum_{i \in I_1} p_i - p_{i_1}$$

$$b - \sum_{i \in I_1} p_i - p_{i_1} - p_{i_2}$$

$$\vdots$$

arrestandoci se

Caso A si arriva ad un valore negativo, ovvero esiste $r \in \{1, \ldots, k-1\}$ tale che

$$b - \sum_{i \in I_1} p_i - p_{i_1} - \cdots - p_{i_r} \geq 0$$

ma

$$b - \sum_{i \in I_1} p_i - p_{i_1} - \cdots - p_{i_r} - p_{i_{r+1}} < 0.$$

Caso B Si sono sottratti i pesi di tutti gli oggetti in I_f senza mai arrivare ad un valore negativo.

Passo 2 Se ci si trova nel caso A si restituiscano come output:

- $$U(S(I_0, I_1)) = \sum_{i \in I_1} v_i + \sum_{h=1}^{r} v_{i_h} + v_{i_{r+1}} \times \frac{b - \sum_{i \in I_1} p_i - \sum_{h=1}^{r} p_{i_h}}{p_{i_{r+1}}}.$$

- l'elemento $N \in S$ definito come segue

  $$N = I_1 \cup \{i_1, \ldots, i_r\},$$

  con il relativo valore

  $$f(N) = \sum_{i \in I_1} v_i + \sum_{h=1}^{r} v_{i_h}$$

Nel caso B l'output sará:

- $$U(S(I_0, I_1)) = \sum_{i \in I_1} v_i + \sum_{h=1}^{k} v_{i_h}.$$

- l'elemento $N \in S$ definito come segue

$$N = I_1 \cup I_f,$$

con il relativo valore

$$f(N) = U(S(I_0, I_1)) = \sum_{i \in I_1} v_i + \sum_{h=1}^{k} v_{i_h}$$

Si noti che essendo $S = S(\emptyset, \emptyset)$, il calcolo di $U(S)$ é un caso particolare di applicazione della procedura appena vista in cui si ha $I_0 = I_1 = \emptyset$.

Branching

Vediamo ora di descrivere l'operazione di branching. Dapprima la descriviamo per l'insieme S e poi l'estendiamo agli altri sottinsiemi generati dall'algoritmo. Supponiamo di trovarci, al termine dell'esecuzione della procedura per il calcolo di $U(S)$, nel caso A (il caso B é un caso banale in cui tutti gli oggetti possono essere inseriti nello zaino). Avremo quindi un indice $r+1$ che é il primo oggetto per cui la sottrazione successiva dei pesi assume valore negativo. La regola di branching prescrive di suddividere S nei due sottinsiemi

$$S(\{r+1\}, \emptyset) \quad \text{e} \quad S(\emptyset, \{r+1\}),$$

ovvero in un sottinsieme della partizione si aggiunge l'oggetto $r+1$ all'insieme I_0, nell'altro lo si aggiunge all'insieme I_1. Quanto visto per l'insieme S puó essere esteso a tutti i sottinsiemi di forma (4.13): dato un sottinsieme $S(I_0, I_1)$ l'oggetto i_{r+1} che appare nel calcolo dell'upper bound nel caso A[1] viene aggiunto in I_0 in un sottinsieme della partizione di $S(I_0, I_1)$ e in I_1 nell'altro sottinsieme, ovvero la partizione di $S(I_0, I_1)$ sará data dai seguenti sottinsiemi

$$S(I_0 \cup \{i_{r+1}\}, I_1) \quad \text{e} \quad S(I_0, I_1 \cup \{i_{r+1}\}).$$

Si noti che con questa regola di branching tutti i sottinsiemi che appariranno nell'insieme $\mathcal{C}$ saranno del tipo $S(I_0, I_1)$ e quindi un upper bound per essi potrá sempre essere calcolato tramite la procedura vista. Per chiarire ulteriormente il funzionamento dell'algoritmo lo applichiamo ora all'esempio introdotto nella Sezione 4.1.

Esempio 19 *Si noti che gli oggetti sono giá ordinati in modo non crescente rispetto a* $\frac{v_i}{p_i}$.

$$\frac{v_1}{p_1} = \frac{8}{7} > \frac{v_2}{p_2} = \frac{6}{7} > \frac{v_3}{p_3} = \frac{10}{13} > \frac{v_4}{p_4} = \frac{1}{4}.$$

[1] Se ci si trova nel caso B si puó dimostrare che il sottinsieme viene certamente cancellato e quindi non dobbiamo preoccuparci di fare branching su di esso. Infatti nel caso B si ha che la soluzione $N \in S$ che viene restituita ha valore $f(N) = U(S(I_0, I_1))$ ma essendo anche $f(N) \leq LB$, il sottinsieme $S(I_0, I_1)$ viene immediatamente cancellato.

Calcoliamo $U(S)$ tramite la procedura vista con $I_0 = I_1 = \emptyset$ e $I_f = \{1, 2, 3, 4\}$. Si ha:

$$b - p_1 = 9 \quad b - p_1 - p_2 = 2 \quad b - p_1 - p_2 - p_3 = -11,$$

e quindi $i_{r+1} = 3$. La procedura restituisce

$$U(S) = 8 + 6 + 10 \times \frac{2}{13} = \frac{202}{13},$$

e l'elemento $N_1 = \{1, 2\} \in S$ con valore di f pari a 14. Non avendo osservato il valore di f in altri elementi della regione ammissibile S, possiamo porre $LB = 14$. Al momento si ha $\mathcal{C} = \{S\}$.

A questo punto partizioniamo S nei due sottinsiemi $S(\{3\}, \emptyset)$ e $S(\emptyset, \{3\})$ come illustrato in Figura 4.5. Si ha $\mathcal{C} = \{S(\{3\}, \emptyset), S(\emptyset, \{3\})\}$. Applichiamo ora

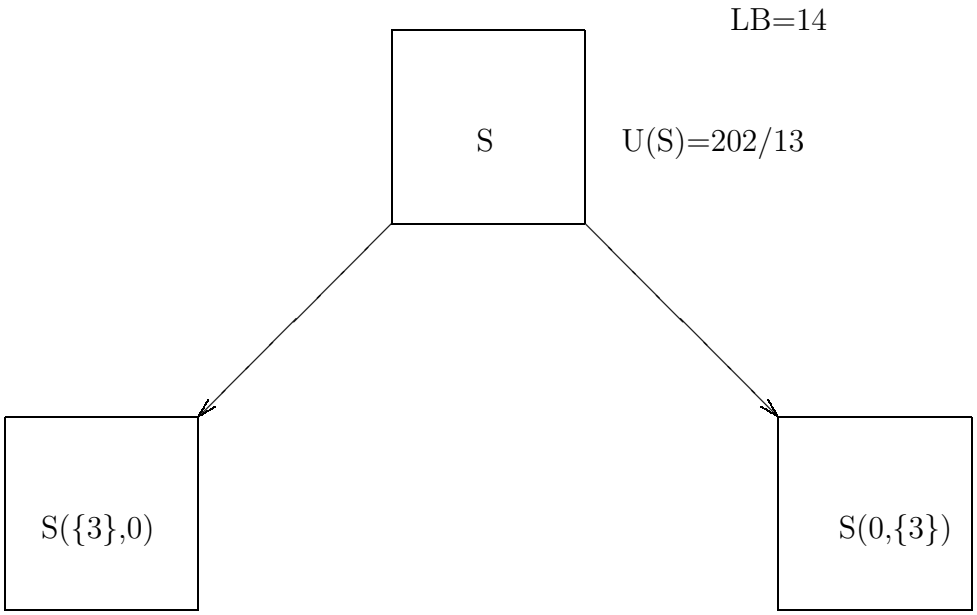


Figura 4.5: Albero di branch-and-bound dopo la prima iterazione.

la procedura per calcolare gli upper bound per $S(\{3\}, \emptyset)$ e $S(\emptyset, \{3\})$.
Per $S(\{3\}, \emptyset)$ avremo $I_0 = \{3\}$, $I_1 = \emptyset$ e $I_f = \{1, 2, 4\}$. Quindi:

$$b - p_1 = 9 \quad b - p_1 - p_2 = 2 \quad b - p_1 - p_2 - p_4 = -2,$$

con $i_{r+1} = 4$. La procedura restituisce quindi come output

$$U(S(\{3\}, \emptyset)) = 8 + 6 + 1 \times \frac{2}{4} = \frac{29}{2},$$

e l'elemento $N_1 = \{1, 2\}$ *per il quale é giá stata valutata la funzione* f *e che quindi non consente di migliorare il valore* LB.
Per $S(\emptyset, \{3\})$ *avremo* $I_0 = \emptyset$, $I_1 = \{3\}$ *e* $I_f = \{1, 2, 4\}$. *Quindi:*

$$b - p_3 - p_1 = -4,$$

con $i_{r+1} = 1$. *La procedura restituisce quindi come output*

$$U(S(\emptyset, \{3\})) = 10 + 8 \times \frac{3}{7} = \frac{94}{7},$$

e l'elemento $N_2 = \{3\}$ *con il relativo valore* $f(N_2) = 10$ *che non consente di migliorare il valore* LB.
Notiamo immediatamente che

$$U(S(\emptyset, \{3\})) = \frac{94}{7} < 14 = LB,$$

e quindi possiamo cancellare il sottinsieme $S(\emptyset, \{3\})$. *Quindi a questo punto si ha* $\mathcal{C} = \{S(\{3\}, \emptyset)\}$. *Ci resta dunque soltanto il sottinsieme* $S(\{3\}, \emptyset)$ *che partizioniamo nei due sottinsiemi* $S(\{3, 4\}, \emptyset)$ *e* $S(\{3\}, \{4\})$ *(si ricordi che nel calcolo di* $U(S(\{3\}, \emptyset))$ *si era trovato* $i_{r+1} = 4$*). A questo punto abbiamo* $\mathcal{C} = \{S(\{3, 4\}, \emptyset), S(\{3\}, \{4\})\}$. *La situazione é riassunta in Figura 4.6. Dobbiamo ora calcolare l'upper bound per i due nuovi sottinsiemi.*
Per $S(\{3, 4\}, \emptyset)$ *avremo* $I_0 = \{3, 4\}$, $I_1 = \emptyset$ *e* $I_f = \{1, 2\}$. *Quindi:*

$$b - p_1 = 9 \quad b - p_1 - p_2 = 2.$$

Ci troviamo quindi nel caso B e la procedura restituisce quindi come output

$$U(S(\{3, 4\}, \emptyset)) = 8 + 6 = 14,$$

e l'elemento $N_1 = \{1, 2\}$ *per il quale é giá stata valutata la funzione* f *e che quindi non consente di migliorare il valore* LB.
Per $S(\{3\}, \{4\})$ *avremo* $I_0 = \{3\}$, $I_1 = \{4\}$ *e* $I_f = \{1, 2\}$. *Quindi:*

$$b - p_4 - p_1 = 5 \quad b - p_4 - p_1 - p_2 = -2,$$

con $i_{r+1} = 2$. *La procedura restituisce quindi come output*

$$U(S(\{3\}, \{4\})) = 1 + 8 + 6 \times \frac{5}{7} = \frac{93}{7},$$

e l'elemento $N_3 = \{1, 4\}$ *con* $f(N_3) = 9$ *che non consente di migliorare il valore* LB.
Possiamo notare che

$$U(S(\{3, 4\}, \emptyset)) = 14 \leq 14 = LB,$$

e che

$$U(S(\{3\}, \{4\})) = \frac{93}{7} < 14 = LB.$$

Possiamo cancellare entrambi i sottinsiemi ed avremo quindi $\mathcal{C} = \emptyset$ (si veda anche la Figura 4.7, notando come i sottinsiemi cancellati formino una partizione della regione ammissibile S). Ció ci permette di concludere che il valore ottimo del nostro problema é $LB = 14$ e la soluzione ottima é l'elemento $N_1 = \{1, 2\}$.

Un modo semplice per migliorare gli upper bound dei nodi si basa sulla seguente osservazione. Se i valori degli oggetti sono tutti interi, il valore dell'obiettivo deve essere anch'esso intero. In tal caso possiamo sempre sostituire l'upper bound $U(S(I_0, I_1))$ con la sua parte intera

$$\lfloor U(S(I_0, I_1) \rfloor$$

Con questa modifica nell'esempio precedente avremmo avuto

$$\lfloor U(S(\{3\}, \emptyset)) \rfloor = \left\lfloor \frac{29}{2} \right\rfloor = 14$$

ed il nodo $S(\{3\}, \emptyset)$ sarebbe stato immediatamente cancellato.

4.3.2 Un algoritmo branch-and-bound per il problema TSP

In questa sezione presentiamo le componenti pricipali di un algoritmo branch-and-bound per il problema TSP. Trattandosi di un problema di minimo si rammenta che i ruoli di lower bound e upper bound vanno invertiti rispetto a quanto visto per il problema $KNAPSACK$ che é un problema di massimo.

Calcolo di un lower bound $L(S)$

Ricordiamo che il modello matematico per il problema TSP é il seguente.

$$\begin{array}{lll}
\min & \sum_{i \in V} \sum_{j \in V \,:\, j \neq i} v_{ij} x_{ij} & \\
& \sum_{i \in V,\ i \neq j} x_{ij} = 1 & \forall\ j \in V \\
& \sum_{i \in V,\ i \neq j} x_{ji} = 1 & \forall\ j \in V \\
& \sum_{i \in U,\ j \in V \setminus U} x_{ij} \geq 1 & \forall\ U \subseteq V :\ 2 \leq \mid U \mid \leq \mid V \mid -2 \\
& x_{ij} \in \{0, 1\} & \forall\ i, j \in \{1, \ldots, n\},\ i \neq j
\end{array}$$

Supponiamo di voler ottenere un lower bound $L(S)$ per il valore ottimo del problema. Per ottenere questo considereremo un rilassamento diverso da quello lineare. Il rilassamento che verrá preso in considerazione é quello ottenuto omettendo i vincoli

$$\sum_{i \in U,\ j \in V \setminus U} x_{ij} \geq 1$$

e cioé:

$$\begin{array}{lll}
\min & \sum_{i \in V} \sum_{j \in V \,:\, j \neq i} v_{ij} x_{ij} & \\
& \sum_{i \in V,\ i \neq j} x_{ij} = 1 & \forall\ j \in V \\
& \sum_{i \in V,\ i \neq j} x_{ji} = 1 & \forall\ j \in V \\
& x_{ij} \in \{0, 1\} & \forall\ i, j \in \{1, \ldots, n\},\ i \neq j
\end{array} \tag{4.15}$$

É chiaro che la regione ammissibile S' di questo problema contiene S (gli elementi di S' devono soddisfare meno vincoli rispetto a quelli di S). Inoltre la funzione obiettivo é la stessa per i due problemi. Quindi il problema (4.15) é effettivamente un rilassamento del problema TSP originario. Per poter peró calcolare il lower bound $L(S)$ dobbiamo essere in grado di risolvere abbastanza velocemente il problema (4.15). Come vedremo, la soluzione di questo problema si riduce a risolvere un problema di assegnamento. Associamo al nostro grafo originario $G = (V, A)$ un grafo bipartito $G' = (V_1 \cup V_2, A')$. I due insiemi V_1 e V_2 sono ottenuti sdoppiando i nodi in V, cioé per ogni nodo $i \in V$ se ne faranno due copie, il nodo $a_i \in V_1$ e il nodo $b_i \in V_2$ (ovviamente si avrá $\mid V_1 \mid = \mid V_2 \mid$). Inoltre, ad ogni arco $(i, j) \in A$ si associa l'arco $(a_i, b_j) \in A'$, a cui si associa il valore v_{ij}. A questo punto il grafo G' non é bipartito completo, ma lo si rende tale aggiungendo gli n archi (a_i, b_i), $i \in V$, a cui si associa un valore pari a $+\infty$. La costruzione del grafo G' associato al grafo G del nostro esempio é illustrata in Figura 4.8. Sul grafo G' possiamo riscrivere il problema (4.15) nel seguente modo:

$$\begin{array}{lll} \min & \sum_{i \in V_1} \sum_{j \in V_2} v_{ij} x_{ij} & \\ & \sum_{i \in V_1} x_{ij} = 1 & \forall\ j \in V_2 \\ & \sum_{i \in V_2} x_{ji} = 1 & \forall\ j \in V_1 \\ & x_{ij} \in \{0, 1\} & \forall\ i, j \in \{1, \ldots, n\} \end{array} \tag{4.16}$$

Si noti che nelle soluzioni ammissibili di questo problema si consente anche la presenza di coppie (a_i, b_i) che non corrispondono ad archi del problema TSP. Tuttavia, il fatto che a tali archi sia associato un valore $+\infty$ ci garantisce che nessuna soluzione ottima del problema (4.16) conterrá tali archi. Come si puó vedere il problema (4.16) é un problema di assegnamento dove i due insiemi da accoppiare sono V_1 e V_2. Questo ci consente di utilizzare l'algoritmo ungherese per il calcolo del lower bound $L(S)$. Una volta ottenuta una soluzione di (4.16) con insieme di archi:

$$(a_i^k, b_j^k) \quad k = 1, \ldots, n$$

questa corrisponde alla seguente soluzione del problema (4.15):

$$(i^k, j^k) \quad k = 1, \ldots, n.$$

Se tale soluzione forma un circuito hamiltoniano, cioé appartiene alla regione ammissibile S, essa rappresenta anche la soluzione ottima del problema TSP. Se non ci si trova in questa situazione fortunata, si puó dimostrare che la soluzione ottenuta é sempre una collezione di sottocircuiti. Vediamo ora di calcolare un lower bound per il nostro esempio.

Esempio 20 *Il problema di assegnamento da risolvere ha la seguente matrice*

associata con entrate per $i \neq j$ i valori v_{ij}:

	b_1	b_2	b_3	b_4
a_1	∞	1	6	7
a_2	1	∞	3	6
a_3	2	4	∞	5
a_4	5	3	6	∞

Si risolva come esercizio tale problema di assegnamento con l'algoritmo ungherese e si verifichi che l'algoritmo termina con la seguente matrice:

	b_1	b_2	b_3	b_4
a_1	∞	0	2	2
a_2	0	∞	0	2
a_3	0	3	∞	0
a_4	1	0	0	∞

(4.17)

e l'assegnamento ottimo $\{(a_1, b_2),\ (a_2, b_1),\ (a_3, b_4),\ (a_4, b_3)\}$ con valore pari a 13. Nel grafo originario l'assegnamento coincide con la seguente soluzione:

$$\{(1,2),\ (2,1),\ (3,4),\ (4,3)\}$$

che, come si vede anche dalla Figura 4.2, non é un circuito hamiltoniano ma é costituito da due sottocircuiti. Non siamo dunque nel caso fortunato in cui risolvendo il rilassamento otteniamo anche una soluzione del problema TSP ma vediamo come la soluzione del rilassamento é una collezione di due sottocircuiti.

Branching del nodo radice

Ci occuperemo ora di specificare come viene partizionata la regione ammissibile S in piú sottinsiemi. Abbiamo visto che se non siamo nel caso fortunato in cui la soluzione del rilassamento é un circuito hamiltoniano, tale soluzione é una collezione di sottocicuiti. Forniremo una semplice regola di suddivisione il cui scopo é quello di impedire il formarsi nei nodi figli di almeno uno dei sottocircuiti nella collezione. Prendiamo il sottocircuito della collezione con meno archi (con scelta arbitraria se ve ne é piú di uno). Indichiamo con $\{(i^1, j^1),\ (i^2, j^2), \ldots, (i^r, j^r)\}$ gli archi del sottocircuito. Il primo nodo figlio viene ottenuto imponendo che in esso non sia presente l'arco (i^1, j^1) (cioé si impone $x_{i^1,j^1} = 0$), il secondo nodo figlio viene ottenuto imponendo che sia presente l'arco (i^1, j^1) ma non sia presente l'arco (i^2, j^2) (cioé si impone $x_{i^1,j^1} = 1,\ x_{i^2,j^2} = 0$), e cosí via fino al r-esimo figlio in cui si impone che siano presenti gli archi (i^k, j^k), $k = 1, \ldots, r-1$, ma non sia presente l'arco (i^r, j^r) (cioé si impone $x_{i^k,j^k} = 1,\ k = 1, \ldots, r-1,\ x_{i^r,j^r} = 0$). Si veda la Figura 4.9. In altre parole si fa in modo che in ciascuno dei figli sia assente almeno un arco del sottocircuito il che esclude la presenza di tale sottocircuito in ciascuno dei nodi figli.

Esempio 21 *Nel nostro esempio abbiamo due sottocircuiti ciascuno con due archi. Selezioniamo arbitrariamente il sottocircuito $\{(1,2),\ (2,1)\}$ e l'operazione di branching é illustrata in Figura 4.10. Si noti anche che, non avendo ancora trovato alcun circuito il valore UB é per il momento fissato a $+\infty$.*

Calcolo del lower bound e branching in un nodo generico dell'albero

Dalla regola di branching illustrata per il nodo radice e che verrá iterata nei nodi successivi dell'albero, possiamo notare che in un nodo generico dell'albero di branch-and-bound ci troveremo ad avere a che fare con sottinsiemi della regione ammissibile S del problema TSP con la seguente forma: dati due sottinsiemi di archi $A_0, A_1 \subseteq A$ con $A_0 \cap A_1 = \emptyset$, i sottinsiemi di S che ci interessano sono:

$$S(A_0, A_1) = \{C = (V, A_C) \in S \ : \ \forall\, (i,j) \in A_1 \ : \ (i,j) \in A_C, \ \forall\, (i,j) \in A_0 \ : \ (i,j) \notin A_C\},$$

ovvero in $S(A_0, A_1)$ abbiamo tutti i circuiti hamiltoniani che contengono sicuramente gli archi in A_1 e che sicuramente non contengono gli archi in A_0.

Il calcolo del lower bound su tali sottinsiemi si effettua come quello del lower bound per S e cioé risolvendo un problema di assegnamento. Rispetto al calcolo del lower bound per S vanno presi i seguenti due accorgimenti:

- per ogni $(i,j) \in A_0$ si ponga $v_{ij} + \infty$: ció impedisce la formazione della coppia (a_i, b_j) e quindi l'introduzione dell'arco (i,j);
- per ogni $(i,j) \in A_1$ si escludano gli elementi a_i e b_j dal problema di assegnamento (essi sono giá accoppiati tra loro). Si riduce di uno la dimensione del problema di assegnamento.

Dopo aver effettuato queste modifiche, il valore ottimo del problema di assegnamento sommato ai valori v_{ij} degli archi $(i,j) \in A_1$ fornisce un lower bound per il sottinsieme $S(A_0, A_1)$. Piú precisamente, se indichiamo con:

$$(a_i^k, b_j^k) \quad k = 1, \ldots, \ell$$

la soluzione del problema di assegnamento, a questa corrisponde il seguente insieme di archi nel grafo originario:

$$A_s = \{(i^k, j^k),\ k = 1, \ldots, \ell\}.$$

Il lower bound per il sottinsieme $S(A_0, A_1)$ è pari a

$$L(S(A_0, A_1)) = \sum_{(i,j) \in A_1 \cup A_s} v_{ij}.$$

Se l'insieme di archi $A_s \cup A_1$ forma un circuito hamiltoniano, cioé appartiene alla regione ammissibile $S(A_0, A_1)$ e quindi anche a S, il suo valore coincide con il valore del lower bound trovato per il sottinsieme e possiamo utilizzare tale valore per aggiornare, eventualmente, l'upper bound UB. Altrimenti l'insieme di archi $A_s \cup A_1$ conterrá dei sottocircuiti.

Branching di $S(A_0, A_1)$

Per effetuare il branching di un sottinsieme $S(A_0, A_1)$ si ripete esattamente quanto giá visto per il nodo radice con una sola piccola differenza: tra i sottocircuiti formati dagli archi $A_s \cup A_1$ si prende non quello che contiene meno archi in assoluto ma quello che *contiene meno archi in* A_s (gli archi in

A_1 sono giá fissati e non possono essere rimossi). Quindi, se indichiamo con $\{(i^1, j^1),\ (i^2, j^2), \ldots, (i^t, j^t)\}$ gli archi del sottocircuito scelto che appartengono ad A_s, il primo nodo figlio viene ottenuto imponendo che in esso non sia presente l'arco (i^1, j^1) (cioé si impone $x_{i^1,j^1} = 0$), il secondo nodo figlio vine ottenuto imponendo che sia presente l'arco (i^1, j^1) ma non sia presente l'arco (i^2, j^2) (cioé si impone $x_{i^1,j^1} = 1,\ x_{i^2,j^2} = 0$), e cosí via fino al t-esimo figlio in cui si impone che siano presenti gli archi (i^k, j^k), $k = 1, \ldots, t-1$, ma non sia presente l'arco (i^t, j^t) (cioé si impone $x_{i^k,j^k} = 1,\ k = 1, \ldots, t-1,\ x_{i^t,j^t} = 0$).

Ma vediamo di ilustrare quanto visto calcolando i lower bound per i due nodi figli del nostro esempio.

Esempio 22 *Nel nodo $\mathcal{N}_1$ abbiamo $A_0 = \{(1, 2)\}$ e $A_1 = \emptyset$. Dobbiamo in tal caso limitarci a risolvere il problema di assegnamento fissando a $+\infty$ il valore dell'accoppiamento (a_1, b_2). Avremo quindi la seguente matrice di partenza:*

	b_1	b_2	b_3	b_4
a_1	∞	∞	6	7
a_2	1	∞	3	6
a_3	2	4	∞	5
a_4	5	3	6	∞

La risoluzione del problema di assegnamento conduce alla seguente soluzione:

$$\{(a_1, b_4),\ (a_2, b_3),\ (a_3, b_1),\ (a_4, b_2)\}$$

con valore pari a 15. La soluzione corrispondente sul grafo originario é la seguente:

$$\{(1, 4),\ (2, 3),\ (3, 1),\ (4, 2)\}$$

e coincide con il circuito hamiltoniano

$$1 \rightarrow\ 4 \rightarrow\ 2 \rightarrow\ 3 \rightarrow\ 1.$$

Questo ci consente di aggiornare il valore dell'upper bound UB e fissarlo pari al valore del lower bound del nodo, cioé $UB = 15$.
Accenniamo anche ad un modo equivalente ma più rapido di determinare un lower bound che consente di non risolvere ogni volta da zero il problema di assegnamento relativo ad un nodo. Gli stessi aggiornamenti fatti sulla matrice della distanze originarie si possono fare sulla matrice ottenuta al termine del calcolo del lower bound del nodo padre, in tal caso la matrice (4.17). In tale matrice si deve porre pari a ∞ l'elemento in posizione (a_1, b_2), ottenendo quindi la seguente matrice

	b_1	b_2	b_3	b_4
a_1	∞	∞	2	2
a_2	0	∞	0	2
a_3	0	3	∞	0
a_4	1	0	0	∞

A questo punto il lower bound per il nodo è dato dalla somma del lower bound del nodo padre (in questo caso 13) e dal valore ottimo del problema di assgnamento definito da quest'ultima tabella.

Passiamo ora all'altro nodo, il nodo $\mathcal{N}_2$. Per esso abbiamo $A_0 = \{(2,1)\}$ e $A_1 = \{(1,2)\}$. Dobbiamo in tal caso limitarci a risolvere il problema di assegnamento cancellando i nodi a_1 e b_2 (e gli archi incidenti su di essi) e fissando a $+\infty$ il valore dell'accoppiamento (a_2, b_1). Avremo quindi la seguente matrice di partenza:

	b_1	b_3	b_4
a_2	∞	3	6
a_3	2	∞	5
a_4	5	6	∞

La risoluzione del problema di assegnamento conduce alla seguente soluzione:

$$\{(a_2, b_3),\ (a_3, b_4),\ (a_4, b_1)\}.$$

Sul grafo originario, aggiungendo anche l'unico arco $(1,2) \in A_1$ si ottiene il seguente insieme di archi:

$$\{(1,2),\ (2,3),\ (3,4),\ (4,1)\}$$

e coincide con il circuito hamiltoniano

$$1 \rightarrow\ 2 \rightarrow\ 3 \rightarrow\ 4 \rightarrow\ 1$$

con valore pari a quello del lower bound, cioé 14. Questo ci consente di aggiornare il valore dell'upper bound UB. Esso infatti era pari a 15, quindi ad un valore superiore rispetto a 14.
Anche qui possiamo calcolare in modo più rapido il lower bound che consente di non risolvere ogni volta da zero il problema di assegnamento relativo ad un nod. Come prima, gli stessi aggiornamenti fatti sulla matrice della distanze originarie si possono fare sulla matrice ottenuta al termine del calcolo del lower bound del nodo padre, in tal caso la matrice (4.17). In tale matrice si deve porre pari a ∞ l'elemento in posizione (a_2, b_1) , cancellare la riga a_1 e la colonna b_2, ottenendo quindi la seguente matrice

	b_1	b_3	b_4
a_2	∞	0	2
a_3	0	∞	0
a_4	1	0	∞

A questo punto il lower bound per il nodo è dato dalla somma del lower bound del nodo padre (in questo caso 13) e dal valore ottimo del problema di assgnamento definito da quest'ultima tabella.

La situazione é ora riassunta in Figura 4.11 dove si nota che:

$$L(\mathcal{N}_1) = 15 > UB \qquad L(\mathcal{N}_2) = 14 \geq UB,$$

il che ci permette di cancellare entrambi i nodi $\mathcal{N}_1$ e $\mathcal{N}_2$ e concludere che la soluzione ottima é il circuito hamiltoniano

$$1 \rightarrow\ 2 \rightarrow\ 3 \rightarrow\ 4 \rightarrow\ 1$$

con valore $UB = 14$.

Di seguito riportiamo un ulteriore esempio di applicazione dell'algoritmo branch-and-bound per il problema TSP.

Esempio 23 *Si consideri il problema TSP sul grafo orientato $G = (V, A)$ con la seguente matrice di distanze:*

	1	2	3	4	5
1	–	4	6	1	3
2	3	–	4	–	5
3	3	0	–	4	–
4	–	1	4	–	0
5	2	3	–	1	–

Alcune distanze tra nodi (oltre, ovviamente, a quelle lungo la diagonale) non sono riportate e sono sostituite da trattini. Qusto indica che gli archi corrispondenti non sono presenti nel grafo, il quale non è completo. Per riportarci al caso di grafo completo possiamo aggiungere gli archi assenti attribuendo ad essi una distanza pari a ∞ in modo che tali archi non faranno sicuramente parte di un circuito hamiltoniano ottimo. La matrice delle distanze diventa quindi la seguente:

	1	2	3	4	5
1	–	4	6	1	3
2	3	–	4	∞	5
3	3	0	–	4	∞
4	∞	1	4	–	0
5	2	3	∞	1	–

Il primo passo dell'algoritmo branch-and-bound consiste nel calcolo di un lower bound per il nodo radice, ovvero per l'intera regione ammissibile S.

Nodo radice *S Per determinare un lower bound del nodo radice utilizziamo la tecnica già vista di risolvere un problema di assegnamento sul grafo bipartito completo $G' = (V_1 \cup V_2, A')$ ricavato a partire dal grafo G attraverso lo sdoppiamento dei nodi in V, cioé:*

$$\forall\ i \in V \ \textit{introduci i nodi}\ a_i \in V_1 \quad b_i \in V_2.$$

e con la tabella dei costi che coincide con la matrice delle distanze modificata con l'aggiunta dei valori ∞ lungo la diagonale (in modo da impedire gli

accoppiamenti (a_i, b_i)*). La tabella dei costi è quindi la seguente:*

	1	2	3	4	5
1	∞	4	6	1	3
2	3	∞	4	∞	5
3	3	0	∞	4	∞
4	∞	1	4	∞	0
5	2	3	∞	1	∞

Utilizzando l'algoritmo ungherese possiamo risolvere questo problema di assegnamento. Omettiamo i passaggi dell'algoritmo e riportiamo solo la sua tabella finale:

	b_1	b_2	b_3	b_4	b_5
a_1	∞	4	2	0	3
a_2	1	∞	0	∞	5
a_3	1	0	∞	3	∞
a_4	∞	1	0	∞	0
a_5	0	3	∞	0	∞

(4.18)

con valore ottimo dell'assegnamento pari a 7 ed assegnamento ottimo

$$(a_1, b_4) \quad (a_2, b_3) \quad (a_3, b_2) \quad (a_4, b_5) \quad (a_5, b_1)$$

Quindi avremo che il lower bound per il nodo radice è $L(S) = 7$ *mentre dall'assegnamento ottimo otteniamo i seguenti due sottocircuiti sul grafo* G

$$2 \to 3 \to 2 \quad 1 \to 4 \to 5 \to 1$$

Non essendo la soluzione ricavata dall'assegnamento ottimo un circuito hamiltoniano e non avendo ancora individuato alcun circuito hamiltoniano poniamo il valore dell'upper bound UB *pari a* ∞*.*

Ora dobbiamo procedere al branching del nodo radice. Il branching si ottiene prendendo un sottocircuito di lunghezza minima tra quelli ottenuti attraverso il calcolo del lower bound del nodo radice. Nel nostro caso si considera il sottocircuito $2 \to 3 \to 2$ *ed i nodi figli si ottengono imponendo che almeno un arco del sottocircuito non sia presente in ciascun nodo figlio. L'operazione di branching è illustrata in Figura 4.12. Nel nodo* S_1 *troveremo tutti i circuiti hamiltoniani che non contengono l'arco* $(2, 3)$ *(*$x_{23} = 0$*), mentre nel nodo* S_2 *si trovano tutti i circuiti hamiltoniani che contengono l'arco* $(2, 3)$ *ma non contengono l'arco* $(3, 2)$ *(*$x_{23} = 1, x_{32} = 0$*). Dobbiamo ora calcolare i lower bound per i nodi* S_1 *e* S_2*.*

Nodo S_1 *Il modo più rapido per ottenere un lower bound per* S_1 *consiste nel risolvere un problema di asegnamento ottenuto modificando opportunamente la tabella finale del nodo padre, il nodo radice* S*. Nel passare dal nodo padre* S *al nodo* S_1 *abbiamo imposto che in* S_1 *non sia presente l'arco* $(2, 3)$*. Per impedire*

la presenza di tale arco poniamo semplicemente pari a ∞ il valore della coppia (a_2, b_3) nella Tabella 4.18, da cui risulta:

	b_1	b_2	b_3	b_4	b_5
a_1	∞	4	2	0	3
a_2	1	∞	∞	∞	5
a_3	1	0	∞	3	∞
a_4	∞	1	0	∞	0
a_5	0	3	∞	0	∞

A questo punto si risolve con l'algoritmo ungherese il problema di assegnamento con questa tabella dei costi. Come al solito omettioamo i passagi dell'algoritmo e riportiamo direttamente la tabella finale:

	b_1	b_2	b_3	b_4	b_5
a_1	∞	2	0	0	1
a_2	0	∞	∞	∞	2
a_3	3	0	∞	5	∞
a_4	∞	1	0	∞	0
a_5	0	1	∞	0	∞

(4.19)

con valore ottimo dell'assegnamento pari a 3 e assegnamento ottimo

$$(a_1, b_3) \quad (a_2, b_1) \quad (a_3, b_2) \quad (a_4, b_5) \quad (a_5, b_4)$$

Il lower bound del nodo S_1 é dato dalla somma del lower bound del nodo padre, cioé 7, e del valore ottimo dell'assegnamento, cioè 3. Avremo quindi $L(S_1) = 10$. Dalla soluzione del problema di assegnamento otteniamo sul grafo G i seguenti sottocircuiti

$$1 \to 3 \to 2 \to 1 \quad 4 \to 5 \to 4$$

Anche in questo caso non abbiamo un circuito hamiltoniano e quindi il valore UB rimane pari a ∞.

Nodo S_2 *Anche qui il modo più rapido per ottenere un lower bound per S_2 consiste nel risolvere un problema di asegnamento ottenuto modificando opportunamente la tabella finale del nodo padre, il nodo radice S. Nel passare dal nodo padre S al nodo S_2 abbiamo imposto che in S_2 non sia presente l'arco $(3, 2)$ e sia invece presente l'arco $(2, 3)$. Per impedire la presenza dell'arco $(3, 2)$ poniamo semplicemente pari a ∞ il valore della coppia (a_3, b_2) nella Tabella 4.18 mentre imporre la presenza dell'arco $(2, 3)$ equibvale ad imporre l'accoppiamento (a_2, b_3) il che consente di omettere la riga a_2 e la colonna b_3. La tabella risultante è la seguente:*

	b_1	b_2	b_4	b_5
a_1	∞	4	0	3
a_3	1	∞	3	∞
a_4	∞	1	∞	0
a_5	0	3	0	∞

A questo punto si risolve con l'algoritmo ungherese il problema di assegnamento con questa tabella dei costi. Come al solito omettioamo i passagi dell'algoritmo e riportiamo direttamente la tabella finale:

	b_1	b_2	b_4	b_5
a_1	∞	1	0	1
a_3	0	∞	2	∞
a_4	∞	0	∞	0
a_5	0	0	0	∞

con valore ottimo dell'assegnamento pari a 4 e assegnamento ottimo

$$(a_1, b_4) \quad (a_3, b_1) \quad (a_4, b_5) \quad (a_5, b_2)$$

Il lower bound del nodo S_2 é dato dalla somma del lower bound del nodo padre, cioé 7, e del valore ottimo dell'assegnamento, cioè 4. Avremo quindi $L(S_2) = 11$. Dalla soluzione del problema di assegnamento e dagli archi in $\mathcal{A}^1_{S_2}$ (il solo arco $(2, 3)$) otteniamo sul grafo G il seguente circuito hamiltoniano

$$1 \to 4 \to 5 \to 2 \to 3 \to 1$$

con valore pari al lower bound, ovvero 11. Avendo finalmente un circuito hamiltoniano possiamo aggiornare il valore UB con il valore di tale circuito ed avremo quindi $UB = 11$.

Ora procediamo alla cancellazione di nodi. Non possiamo cancellare il nodo S_1 ($L(S_1) < UB$) ma possiamo cancellare il nodo S_2 ($L(S_2) \geq UB$). A questo punto tra i nodi foglia ci rimane il solo nodo S_1 e procediamo ad eseguire il branching su di esso. Il branching si esegue prendendo un sottocircuito di lunghezza minima tra quelli ottenuti dalla risoluzione del problema di assegnamento relativo al nodo S_2. Nel nostro caso il sottocircuito è $4 \to 5 \to 4$ ed i nodi figli si ottengono imponendo che in ciascuno di essi non sia presente almeno un arco del sottocircuito tra quelli che non appartengono ad $\mathcal{A}^1_{S_1}$ *(il nodo S_1 e tutti i suoi successori possono solo contenere circuiti hamiltoniani che contengono tutti gli archi in $\mathcal{A}^1_{S_1}$ e quindi non possiamo imporre in un nodo figlio di S_1 che non sia presente un arco in $\mathcal{A}^1_{S_1}$). Nel nostro caso $\mathcal{A}^1_{S_1}$ è vuoto e quindi non abbiamo problemi. Il branching avverrà creando due nodi figli: nel primo, S_3, si impone che non siano presenti circuiti hamiltoniani con l'arco $(4, 5)$ ($x_{45} = 0$), mentre nel secondo, S_4, si impone che vi siano solo circuiti hamiltoniani che contengono certamente l'arco $(4, 5)$ ma non contengono l'arco $(5, 4)$ ($x_{45} = 1, x_{54} = 0$). L'operazione di branching è illustrata in Figura 4.13. Ora dobbiamo calcolare i lower bound per i nodi S_3 e S_4.*

Nodo *S_3 Il modo più rapido per ottenere un lower bound per S_3 consiste nel risolvere un problema di assegnamento ottenuto modificando opportunamente la tabella finale del nodo padre, il nodo S_1. Nel passare dal nodo padre S_1 al nodo S_3 abbiamo imposto che in S_3 non sia presente l'arco $(4, 5)$. Per impedire la*

presenza di tale arco poniamo semplicemente pari a ∞ il valore della coppia (a_4, b_5) nella Tabella 4.19, da cui risulta:

	b_1	b_2	b_3	b_4	b_5
a_1	∞	2	0	0	1
a_2	0	∞	∞	∞	2
a_3	3	0	∞	5	∞
a_4	∞	1	0	∞	∞
a_5	0	1	∞	0	∞

A questo punto si risolve con l'algoritmo ungherese il problema di assegnamento con questa tabella dei costi. Come al solito omettiamo i passagi dell'algoritmo e riportiamo direttamente la tabella finale:

	b_1	b_2	b_3	b_4	b_5
a_1	∞	2	0	0	0
a_2	0	∞	∞	∞	1
a_3	3	0	∞	5	∞
a_4	∞	1	0	∞	∞
a_5	0	1	∞	0	∞

con valore ottimo dell'assegnamento pari a 1 e assegnamento ottimo

$$(a_1, b_5) \quad (a_2, b_1) \quad (a_3, b_2) \quad (a_4, b_3) \quad (a_5, b_4)$$

Il lower bound del nodo S_3 é dato dalla somma del lower bound del nodo padre S_1, cioé 10, e del valore ottimo dell'assegnamento, cioè 1. Avremo quindi $L(S_3) =$ 11. Dalla soluzione del problema di assegnamento otteniamo sul grafo G il seguente circuito hamiltoniano

$$1 \to 5 \to 4 \to 3 \to 2 \to 1$$

Abbiamo un circuito hamiltoniano ma non ha valore migliore rispetto all'upper bound UB e quindi non aggiorniamo UB.

Nodo S_4 *Il modo più rapido per ottenere un lower bound per S_4 consiste come al solito nel risolvere un problema di assegnamento ottenuto modificando opportunamente la tabella finale del nodo padre, il nodo S_1. Nel passare dal nodo padre S_1 al nodo S_4 abbiamo imposto che in S_4 non sia presente l'arco $(5, 4)$ e sia invece presente l'arco $(4, 5)$. Per impedire la presenza dell'arco $(5, 4)$ poniamo semplicemente pari a ∞ il valore della coppia (a_5, b_4) nella Tabella 4.19 mentre imporre la presenza dell'arco $(4, 5)$ equivale ad imporre l'accoppiamento (a_4, b_5) il che consente di omettere la riga a_4 e la colonna b_5. La tabella risultante è la seguente:*

	b_1	b_2	b_3	b_4
a_1	∞	2	0	0
a_2	0	∞	∞	∞
a_3	3	0	∞	5
a_5	0	1	∞	∞

A questo punto si risolve con l'algoritmo ungherese il problema di assegnamento con questa tabella dei costi. Come al solito omettioamo i passagi dell'algoritmo e riportiamo direttamente la tabella finale:

	b_1	b_2	b_3	b_4
a_1	∞	7	0	0
a_2	0	∞	∞	∞
a_3	4	0	∞	0
a_5	0	0	∞	∞

con valore ottimo dell'assegnamento pari a 6 e assegnamento ottimo

$$(a_1, b_3) \quad (a_2, b_1) \quad (a_3, b_4) \quad (a_5, b_2)$$

Il lower bound del nodo S_4 é dato dalla somma del lower bound del nodo padre S_1, cioé 10, e del valore ottimo dell'assegnamento, cioè 6. Avremo quindi $L(S_4) = 16$. Dalla soluzione del problema di assegnamento e dagli archi in $\mathcal{A}^1_{S_4}$ (il solo arco $(4, 5)$) otteniamo sul grafo G il seguente circuito hamiltoniano

$$1 \to 3 \to 4 \to 5 \to 2 \to 1$$

con valore pari al lower bound, ovvero 16. Abbiamo un circuito hamiltoniano ma con valore peggiore rispetto all'upper bound UB e quindi non aggiorniamo UB.

Ora posiamo cancellare sia S_3 che S_4 (per entrambi si ha $L(S_3), L(S_4) \geq UB$). Avendo cancellato tutti i nodi foglia (si veda la Figura 4.14) possiamo terminare restituendo come valore otimo del problema il valore $UB = 11$. Si sono inoltre ottenute due soluzioni ottime. Infatti, sia il circuito hamiltoniano ottenuto nel nodo S_2 che quello ottenuto nel nodo S_3 hanno valore pari al valore ottimo 11.

4.3.3 Un diverso lower bound per il caso simmetrico

Vogliamo ora presentare un diverso rilassamento e, quindi, un diverso modo di calcolare lower bound per il caso del TSP simmetrico, ovvero il TSP in cui

$$v_{ij} = v_{ji} \quad \forall\, i, j \in V.$$

Un rilassamento del problema TSP simmetrico

Supponiamo di avere un grafo $G = (V, A)$ non orientato e con distanze v_{ij} sugli archi $(i, j) \in A$. Vogliamo calcolare un nuovo lower bound per il problema TSP simmetrico su tale grafo. Per arrivare a questo introduciamo dapprima la definizione di *1-tree*.

Definizione 13 *Dato un grafo $G = (V, A)$ non orientato e un suo nodo $a \in V$, chiamiamo 1-tree un sottografo $Q = (V, A_Q)$ di G con le seguenti proprietà:*

- *in A_Q ci sono esattamente due archi incidenti sul nodo a;*

- *se escludo da Q il nodo a e i due archi incidenti su di esso, mi rimane un albero sull'insieme di nodi $V \setminus \{a\}$.*

In particolare, da questa definizione segue che $\mid A_Q \mid = \mid V \mid$.

Esempio 24 *Dato il grafo con* $V = \{a, b, c, d, e\}$ *e*

$$A = \{(a,b); (a,c); (b,c); (b,e); (c,d); (d,a); (d,e)\},$$

il sottografo con

$$A_Q = \{(a,b); (d,a); (b,c); (b,e); (d,e)\}$$

è un 1-tree.

Si può notare che ogni circuito hamiltoniano è un 1-tree. Infatti, in un circuito hamiltoniano su ogni nodo incidono esattamente due archi ed inoltre togliendo un nodo a qualsiasi e i due archi del circuito incidenti su di esso si ottiene un albero sull'insieme di nodi $V \setminus \{a\}$. Il viceversa non è vero (lo 1-tree dell'esempio *non* è un circuito hamiltoniano). Qunidi se indichiamo con S' l'insieme degli 1-tree su un grafo G, tale insieme contiene la regione ammissibile S del problema TSP. In altre parole, il problema

$$\min_{Q=(V,A_Q)\in S'} \sum_{(i,j)\in A_Q} v_{ij}$$

risulta essere un rilassamento per il problema TSP simmetrico e la sua risoluzione restituisce un lower bound per il valore ottimo del problema TSP.

Calcolo del lower bound per il problema originario

Si pone ora il problema di come risolvere il rilassamento. Possiamo utilizzare la seguente procedura.

Passo 1. Si risolva il problema MST sul grafo ottenuto scartando da $G = (V, A)$ il nodo a prescelto e tutti gli archi incidenti su di esso. Sia A_T l'insieme di archi della soluzione trovata;

Passo 2. Si aggiungano ad A_T i due archi (a, k) e (a, h) a distanza minima tra tutti quelli incidenti sul nodo a prescelto.

Passo 3. Si restituisca $Q = (V, A_Q)$ con $A_Q = A_T \cup \{(a, k); (a, h)\}$.

Si noti come i tempi di calcolo siano polinomiali dal momento che la risoluzione del problema MST si può fare in tempo polinomiale (ad esempio con l'algoritmo greedy) e lo stesso vale per il calcolo dei due valori minimi. Si noti anche che la scelta del nodo a è arbitraria. Al costo di un maggiore sforzo computazionale, si possono anche calcolare $\mid V \mid$ diversi lower bound scegliendo come nodo a tutti i nodi del grafo G e calcolando per ciascuno di essi il lower bound: come lower bound complessivo del problema originario si utilizza il migliore (ovvero il più grande) tra tutti i $\mid V \mid$ lower bound calcolati.

Calcolo del lower bound per sottoproblemi

Si ricorda che un sottoproblema $S(A_0, A_1)$ del problema TSP simmetrico originario è un problema TSP simmetrico sul grafo G originario in cui si cerca il circuito hamiltoniano a distanza minima tra tutti quelli che sicuramente non contengono gli archi in A_0 e sicuramente contengono gli archi in A_1. Per il calcolo del lower bound di un sottoproblema $S(A_0, A_1)$, si utilizza la stessa procedura imponendo però la presenza degli archi in A_1 ed escludendo quella degli archi in A_0 sia nella risoluzione del problema MST sia nell'individuazione dei due archi incidenti sul nodo a. In particolare, si può risolvere il problema MST sempre con l'algoritmo greedy ma:

- inizializzando l'insieme di archi A_T non con l'insieme vuoto ma con tutti gli archi in A_1 non incidenti sul nodo a;
- non considerando gli archi in A_0 durante l'esecuzione dell'algoritmo greedy.

Inoltre:

- se in A_1 non sono presenti archi incidenti sul nodo a, metteremo in A_Q i due archi a distanza minima tra tutti quelli incidenti sul nodo a e al di fuori di A_0;
- se in A_1 è già presente un arco incidente sul nodo a questo entrerà in A_Q insieme a quello a distanza minima tra tutti quelli incidenti sul nodo a e al di fuori di A_0 e A_1;
- se in A_1 sono già presenti due archi incidenti sul nodo a, solo questi entreranno in A_Q.

Esempio 25 *Supponiamo di avere il seguente problema del TSP simmetrico*

	1	2	3	4	5
1	–	5	8	3	5
2	5	–	4	6	2
3	8	4	–	10	3
4	3	6	10	–	1
5	5	2	3	1	–

Proviamo a calcolare il lower bound per il sottoproblema $S(A_0, A_1)$ *con* $A_0 = \{(1,3);(4,5)\}$ *e* $A_1 = \{(1,5);(2,4)\}$. *Utilizziamo come nodo a il nodo 1. Per prima cosa dobbiamo risolvere il problema MST sull'insieme di nodi* $V \setminus \{1\}$ *imponendo la presenza dell'arco* $(2,4)$ *che è in* A_1 *ed escludendo quella degli archi in* A_0. *Utilizzando l'algoritmo greedy con* A_T *inizializzato con gli archi in* A_1 *non incidenti sul nodo 1 (in questo caso il solo arco* $(2,4)$*) ed escludendo la possibilità di inserire gli archi in* A_0, *arriviamo al seguente albero su* $V \setminus \{1\}$

$$A_T = \{(2,4);(2,5);(3,5)\}.$$

Notiamo che in A_1 è presente l'arco $(1,5)$ incidente sul nodo 1. Ad A_T dobbiamo quindi aggiungere, oltre a questo arco $(1,5)$, l'arco a distanza minima tra tutti quelli incidenti sul nodo 1 e al di fuori di A_0 e A_1, ovvero $(1,4)$. Quindi lo 1-tree ottimo ha l'insieme di archi

$$A_Q = \{(2,4); (2,5); (3,5); (1,5); (1,4)\}$$

con valore ottimo (e quindi lower bound per il sottoproblema $S(A_0, A_1)$) pari a 19. Si noti anche come $Q = (V, A_Q)$ non sia un circuito hamiltoniano e quindi non possa essere utilizzato per aggiornare (eventualmente) il valore di upper bound.

4.3.4 Un rilassamento lagrangiano

Vediamo ora di reinterpretare in altro modo il rilassamento basato sugli 1-tree e di indicare poi una strada per eventualmente migliorare i lower bound. Per prima cosa diamo il modello matematico del problema 1-tree. In esso avremo:

- una variabile binaria x_{ij} per ogni arco (i,j) (con valore 1 se l' arco viene inserito nello 1-tree e 0 altrimenti);
- un vincolo che impone che ci siano esattamente due archi incidenti sul nodo a:
$$\sum_{i \in V,\ i \neq a} x_{ia} = 2;$$
- vincoli che impongano che gli archi incidenti sui soli nodi in $V \setminus \{a\}$ formino un albero.

Per imporre che gli archi selezionati formino un albero su $V \setminus \{a\}$ dobbiamo richiedere che il numero di tali archi sia pari a $\mid V \setminus \{a\} \mid -1$, ovvero pari a $\mid V \mid -2$:

$$\sum_{i,j \in V \setminus \{a\}} x_{ij} = \mid V \mid -2;$$

e che tali archi non formino cicli. Per l'eliminazione di cicli in $V \setminus \{a\}$ utilizzeremo i seguenti vincoli. Dato $U \subseteq V \setminus \{a\}$, sia

$$E(U) = \{(i,j) \ : \ i,j \in U\}$$

Osservando che un ciclo sui nodi in U dovrebbe contenere $\mid U \mid$ archi in $E(U)$, per eliminare cicli imporremo

$$\sum_{(i,j) \in E(U)} x_{ij} \leq \mid U \mid -1 \qquad \forall\ U \subseteq V \setminus \{a\}$$

Si noti che per $\mid U \mid \leq 2$ i vincoli risultanti sono banali e possono essere omessi. Riassumendo, il modello matematico del problema 1 -tree è il seguente:

$$\begin{array}{lll}
\min & \sum_{i,j\in V,\ i<j} v_{ij}x_{ij} & \\
& \sum_{i\in V,\ i\neq a} x_{ia} = 2 & \\
& \sum_{(i,j)\in E(U)} x_{ij} \leq \mid U \mid -1 & \forall\ U \subseteq V \setminus \{a\}\ : \mid U \mid \geq 3 \\
& \sum_{i,j\in V\setminus\{a\}} x_{ij} = \mid V \mid -2 & \\
& x_{ij} \in \{0,1\} & \forall\ i,j \in V,\ i<j
\end{array}$$

Esempio 26 *Problema TSP simmetrico con la seguente tabella di distanze:*

	1	2	3	4
1	–	12	9	14
2	12	–	8	9
3	9	8	–	1
4	14	9	1	–

Si fissi come nodo a il nodo 1. Il modello matematico risultante è il seguente:

$$\begin{array}{rc}
\min & 12x_{12} + 9x_{13} + 14x_{14} + 8x_{23} + 9x_{24} + x_{34} \\
& x_{12} + x_{13} + x_{14} = 2 \\
& x_{23} + x_{24} + x_{34} \leq 2 \\
& x_{23} + x_{24} + x_{34} = 2 \\
& x_{12}, x_{13}, x_{14}, x_{23}, x_{24}, x_{34} \in \{0,1\}
\end{array}$$

Si noti come in questo caso il vincolo $x_{23}+x_{24}+x_{34} \leq 2$ sia implicato dall'altro vincolo $x_{23} + x_{24} + x_{34} = 2$ e quindi può essere omesso.

Si può dimostrare che un modello valido per il problema TSP simmetrico è identico a quello visto per il problema 1-tree ma con l'aggiunta che su *tutti* i nodi in V incidano esattamente due archi, ovvero:

$$\begin{array}{lll}
\min & \sum_{i,j\in V,\ i<j} v_{ij}x_{ij} & \\
& \sum_{i\in V,\ i\neq j} x_{ij} = 2 & \forall\ j \in V \\
& \sum_{(i,j)\in E(U)} x_{ij} \leq \mid U \mid -1 & \forall\ U \subseteq V \setminus \{a\}\ : \mid U \mid \geq 3 \\
& \sum_{i,j\in V\setminus\{a\}} x_{ij} = \mid V \mid -2 & \\
& x_{ij} \in \{0,1\} & \forall\ i,j \in V,\ i<j
\end{array}$$

Esempio 27 *Il modello per il TSP simmetrico dell'esempio sarà:*

$$\begin{aligned}
\min \quad & 12x_{12} + 9x_{13} + 14x_{14} + 8x_{23} + 9x_{24} + x_{34} \\
& x_{12} + x_{13} + x_{14} = 2 \\
& x_{12} + x_{23} + x_{24} = 2 \\
& x_{13} + x_{23} + x_{34} = 2 \\
& x_{14} + x_{24} + x_{34} = 2 \\
& x_{23} + x_{24} + x_{34} = 2 \\
& x_{12}, x_{13}, x_{14}, x_{23}, x_{24}, x_{34} \in \{0, 1\}
\end{aligned}$$

In pratica possiamo vedere il problema 1-tree come il rilassamento del problema TSP simmetrico ottenuto omettendo da questo tutti i vincoli che richiedono che vi siano esattamente due archi incidenti su ogni nodo, tranne quello relativo al nodo a. Come già visto, l'omissione di vincoli può essere vista come un caso particolare di rilassamento lagrangiano in cui tutti i moltiplicatori di Lagrange sono fissati a 0 e viene allora naturale chiedersi cosa succede se prendiamo moltiplicatori di Lagrange diversi da 0. Quindi, introduciamo ora moltiplicatori di Lagrange $\lambda = (\lambda_k)_{k \in V \setminus \{a\}}$ per i vincoli che richiedono che esattamente due archi incidano sui nodi, con l'unica eccezione del nodo a selezionato. Il modello risultante del rilassamento lagrangiano è il seguente:

$$\begin{aligned}
u(\lambda) \quad = \min \quad & \textstyle\sum_{i,j \in V,\ i<j} v_{ij} x_{ij} + \sum_{k \in V \setminus \{a\}} \lambda_k (2 - \sum_{i \in V,\ i \neq k} x_{ik}) & \\
& \textstyle\sum_{i \in V,\ i \neq a} x_{ia} = 2 & \\
& \textstyle\sum_{(i,j) \in E(U)} x_{ij} \leq \mid U \mid -1 & \forall\ U \subseteq V \setminus \{a\}\ : \mid U \mid \geq 3 \\
& \textstyle\sum_{i,j \in V \setminus \{a\}} x_{ij} = \mid V \mid -2 & \\
& x_{ij} \in \{0, 1\} & \forall\ i, j \in V,\ i < j
\end{aligned}$$

Si rammenti che essendo i vincoli di uguaglianza, possiamo considerare valori dei moltiplicatori di Lagrange $\lambda_k,\ k \in V \setminus \{a\}$, anche negativi e non solo maggiori o uguali a zero. Si vede che il rilassamento visto prima basato sul calcolo dello 1-tree minimo è un caso particolare di questo rilassamento lagrangiano in cui $\lambda_k = 0$ per tutti i $k \in V \setminus \{a\}$. Per comodità di notazione si include nell'obiettivo anche un termine relativo al vincolo di incidenza di esattamente due archi sul nodo a con il relativo moltiplicatore di Lagrange λ_a, imponendo però che questo possa assumere il solo valore 0. In tal modo avremo a che fare con un vettore di moltiplicatori di Lagrange $\lambda = (\lambda_k)_{k \in V}$ le cui componenti possono assumere valori positivi, negativi o nulli con la sola eccezione della componente relativa al nodo a che può assumere solo valore nullo.

Il modello del rilassamento lagrangiano può essere riscritto nel seguente modo:

$$\begin{array}{lll} u(\lambda) = \min & \sum_{i,j\in V,\ i<j} v_{ij}x_{ij} + \sum_{k\in V} \lambda_k(2 - \sum_{i\in V,\ i\neq k} x_{ik}) & \\ & \sum_{i\in V,\ i\neq a} x_{ia} = 2 & \\ & \sum_{(i,j)\in E(U)} x_{ij} \leq \mid U \mid -1 & \forall\ U \subseteq V \setminus \{a\}\ : \mid U \mid \geq 3 \\ & \sum_{i,j\in V\setminus\{a\}} x_{ij} = \mid V \mid -2 & \\ & x_{ij} \in \{0,1\} \qquad \forall\ i,j \in V,\ i<j & \end{array}$$

Un'ulteriore elaborazione del modello ci porta a:

$$\begin{array}{lll} u(\lambda) = \min & \sum_{i,j\in V,\ i<j}(v_{ij} - \lambda_i - \lambda_j)x_{ij} + 2\sum_{k\in V} \lambda_k & \\ & \sum_{i\in V,\ i\neq a} x_{ia} = 2 & \\ & \sum_{(i,j)\in E(U)} x_{ij} \leq \mid U \mid -1 & \forall\ U \subseteq V \setminus \{a\}\ : \mid U \mid \geq 3 \\ & \sum_{i,j\in V\setminus\{a\}} x_{ij} = \mid V \mid -2 & \\ & x_{ij} \in \{0,1\} \qquad \forall\ i,j \in V,\ i<j & \end{array}$$

Ma come possiamo risolvere il rilassamento lagrangiano? Per valori fissati dei moltiplicatori di Lagrange λ_k, $k \in V$, il rilassamento lagrangiano è facilmente risolvibile con la procedura vista in precedenza per l'individuazione dello 1-tree minimo. Infatti, il problema consiste nell'individuare lo 1-tree minimo tenendo conto che le distanze degli archi sono ora definite come segue

$$v'_{ij} = v_{ij} - \lambda_i - \lambda_j.$$

Esempio 28 *Il rilassamento lagrangiano può essere scritto in questa forma:*

$$\begin{array}{lc} u(\lambda_1,\ldots,\lambda_4) = \min & 12x_{12} + 9x_{13} + 14x_{14} + 8x_{23} + 9x_{24} + x_{34} + \\ & \lambda_1(2 - x_{12} - x_{13} - x_{14}) + \lambda_2(2 - x_{12} - x_{23} - x_{24}) + \\ & +\lambda_3(2 - x_{13} - x_{23} - x_{34}) + \lambda_4(2 - x_{14} - x_{24} - x_{34}) \\ & \\ & x_{12} + x_{13} + x_{14} = 2 \\ & x_{23} + x_{24} + x_{34} = 2 \\ & x_{12}, x_{13}, x_{14}, x_{23}, x_{24}, x_{34} \in \{0,1\} \end{array}$$

Elaborando ulteriormente arriviamo a

$$\begin{array}{lc} u(\lambda_1,\ldots,\lambda_4) = \min & (12 - \lambda_1 - \lambda_2)x_{12} + (9 - \lambda_1 - \lambda_3)x_{13} + \\ & +(14 - \lambda_1 - \lambda_4)x_{14} + (8 - \lambda_2 - \lambda_3)x_{23} + \\ & +(9 - \lambda_2 - \lambda_4)x_{24} + (1 - \lambda_3 - \lambda_4)x_{34} + 2\sum_{i=1}^{4} \lambda_i \\ & \\ & x_{12} + x_{13} + x_{14} = 2 \\ & x_{23} + x_{24} + x_{34} = 2 \\ & x_{12}, x_{13}, x_{14}, x_{23}, x_{24}, x_{34} \in \{0,1\} \end{array}$$

Una volta visto il rilassamento lagrangiano, possiamo introdurre anche il duale lagrangiano. Questo consisterà nell'individuare i valori $\lambda^* = (\lambda_k^*)_{k \in V}$, con $\lambda_a^* = 0$, per cui la funzione $u(\lambda)$ abbia il valore più grande possibile. In altre parole si tratta di risolvere il seguente problema

$$\max_{\lambda:\ \lambda_a = 0} \ u(\lambda)$$

Esempio 29 *Risolvendo il rilassamento lagrangiano con $\lambda_1 = \lambda_2 = \lambda_3 = \lambda_4 = 0$ si ottiene lo 1-tree minimo*

$$(2,3) \quad (3,4) \quad (1,2) \quad (1,3)$$

con un lower bound pari a 30. Se ora però consideriamo i seguenti moltiplicatori di Lagrange:

$$\lambda_1 = 0 \quad \lambda_2 = 0 \quad \lambda_3 = -1 \quad \lambda_4 = 1,$$

arriviamo ad un problema con la seguente tabella di distanze

	1	2	3	4
1	–	12	10	13
2	12	–	9	8
3	10	9	–	1
4	13	8	1	–

Lo 1-tree minimo con queste distanze è

$$(3,4) \quad (2,4) \quad (1,2) \quad (1,3)$$

e il lower bound è pari a

$$2 \sum_{i \in V} \lambda_i + (\textit{valore 1-tree minimo}) = 0 + (1 + 8 + 12 + 10) = 31,$$

migliore rispetto al precedente. Nel caso specifico si osserva anche che quest'ultimo 1-tree minimo è anche un circuito hamiltoniano con distanza complessiva pari a 31 ed è quindi soluzione ottima del problema in questione.

Non ci addentreremo nelle tecniche di risoluzione del duale lagrangiano ma diamo una possibile strategia per migliorare quanto ottenuto con determinati valori $\tilde{\lambda}_i$ dei moltiplicatori di Lagrange (ad esempio, $\tilde{\lambda}_i = 0$ per ogni i).

Nella soluzione ottenuta con i moltiplicatori di Lagrange $\tilde{\lambda}_i$ dobbiamo diminuire il peso dei nodi con grado superiore a 2 nella soluzione e accrescere quello dei nodi con grado inferiore a 2. Per fare questo dobbiamo ridurre il valore dei moltiplicatori di Lagrange relativi ai nodi con grado superiore a 2 nella soluzione e accrescere quello dei nodi con grado inferiore a 2. Per questo possiamo aggiornare i moltiplicatori di Lagrange nel modo seguente:

$$\bar{\lambda}_i = \tilde{\lambda}_i + 2 - \text{grado di } i \text{ nello 1-tree minimo.}$$

Esempio 30 *Nell'esempio considerato i gradi dei nodi 1, 2, 3 e 4 nello 1-tree minimo ottenuto con tutti i moltiplicatori di Lagrange nulli sono rispettivamente 2, 2, 3 e 1 e la regola appena vista porta proprio ai moltiplicatori di Lagrange proposti precedentemente.*

4.4 Programmazione dinamica

La *programmazione dinamica* é un altro approccio che consente di risolvere problemi in modo esatto. Considereremo solo problemi di massimo ma piccole modifiche consentono di applicare tale approccio anche a problemi di minimo. La programmazione dinamica é applicabile a problemi con determinate proprietá che andremo ora ad elencare.

1. Il problema puó essere suddiviso in n *blocchi*.

2. In ogni blocco k, $k = 1, \ldots, n$ ci si trova in uno degli *stati* s_k appartenenti all'insieme di stati $Stati_k$. L'insieme $Stati_1$ del blocco 1 è costituito da un singolo stato s_1.

3. In ogni blocco k si deve prendere una decisione d_k appartenente ad un insieme di possibili decisioni D_k. L'insieme di possibili decisioni puó dipendere dallo stato s_k, ovvero $D_k = D_k(s_k)$.

4. Se al blocco k si prende la decisione d_k e ci si trova nello stato s_k, il blocco k fornisce un *contributo* alla funzione obiettivo f del problema pari a $u(d_k, s_k)$. La funzione obiettivo f sará pari alla somma dei contributi degli n blocchi (esistono anche varianti in cui i contributi non si sommano ma si moltiplicano tra loro ma qui ne omettiamo la trattazione).

5. Se al blocco k ci si trova nello stato s_k e si prende la decisione d_k, al blocco $k+1$ ci troveremo nello stato $s_{k+1} = t(d_k, s_k)$. La funzione t viene detta *funzione di transizione* (la Figura 4.15 illustra la transizione).

6. Infine, la proprietá essenziale é quella che viene chiamata *principio di ottimalitá*: se al blocco k mi trovo nello stato s_k, la sequenza di decisioni ottime da prendere nei blocchi $k, k+1, \ldots, n$ é *totalmente indipendente* da come sono giunto allo stato s_k, ovvero dalle decisioni ai blocchi $1, \ldots, k-1$ che mi hanno fatto arrivare allo stato s_k.

Poiché il principio di ottimalitá sancisce che la sequenza di decisioni ottime a partire da una stato s_k al blocco k fino al blocco n é indipendente da come sono arrivato allo stato s_k, posso definire una funzione $f_k^*(s_k)$ che restituisce il valore ottimo delle somme dei contributi dei blocchi $k, k+1, \ldots, n$ quando si parte dallo stao s_k al blocco k. Tipicamente é semplice calcolare $f_n^*(s_n)$ per ogni $s_n \in Stati_n$, cioé il valore ottimo del solo contributo del blocco n quando ci si trova nello stato s_n. La corrispondente decisione ottima verrá indicata con $d_n^*(s_n)$. A questo punto posso procedere a ritroso per calcolare $f_{n-1}^*(s_{n-1})$ per ogni $s_{n-1} \in Stati_{n-1}$. Infatti, dato un generico stato s_{n-1} in $Stati_{n-1}$, consideriamo una generica decisione $d_{n-1} \in D_{n-1}$. Il contributo di tale decisione al blocco $n-1$ é pari a $u(d_{n-1}, s_{n-1})$. Inoltre mi sposto nello stato $s_n = t(d_{n-1}, s_{n-1})$ al blocco n. Da qui posso procedere in modo ottimo e quindi con un contributo pari a $f_n^*(t(d_{n-1}, s_{n-1}))$. Quindi, se mi trovo nello

stato s_{n-1} e prendo la decisione d_{n-1} il contributo complessivo dei blocchi $n-1$ e n sará dato da

$$u(d_{n-1}, s_{n-1}) + f_n^*(t(d_{n-1}, s_{n-1})).$$

Tra tutte le possibili decisioni d_{n-1} la migliore sará quella che rende massimo il contributo complessivo. Tale decisione viene indicata con $d_{n-1}^*(s_{n-1})$ e si avrá

$$f_{n-1}^*(s_{n-1}) = u(d_{n-1}^*(s_{n-1}), s_{n-1}) + f_n^*(t(d_{n-1}^*(s_{n-1}), s_{n-1})) =$$
$$= \max_{d_{n-1} \in D_{n-1}} [u(d_{n-1}, s_{n-1}) + f_n^*(t(d_{n-1}, s_{n-1}))].$$

Una volta calcolati i valori $f_{n-1}^*(s_{n-1})$ per tutti gli stati $s_{n-1} \in Stati_{n-1}$, possiamo continuare a procedre a ritroso. Per il blocco k avremo

$$\forall\ s_k \in Stati_k : \quad f_k^*(s_k) = \max_{d_k \in D_k} [u(d_k, s_k) + f_{k+1}^*(t(d_k, s_k))],$$

con la corrispondente decisione ottima $d_k^*(s_k)$ (si noti che poiché si procede a ritroso i valori f_{k+1}^* sono giá stati calcolati). Arrivati al blocco 1 si ha che $Stati_1$ é formato da un unico stato s_1 e il valore $f_1^*(s_1)$ coincide con il valore ottimo del problema. Per ricostruire la soluzione ottima possiamo partire dal blocco 1. Al blocco 1 la decisione ottima é $d_1^*(s_1)$. Con tale decisione ci spostiamo allo stato $s_2^* = t(d_1^*(s_1), s_1)$ del blocco 2 e la decisione ottima per tale blocco sará $d_2^*(s_2^*)$. Con tale decisione ci spostiamo allo stato $s_3^* = t(d_2^*(s_2^*), s_2^*)$ del blocco 3 e la decisione ottima per tale blocco sará $d_3^*(s_3^*)$. Si procede in questo modo fino ad arrivare al blocco n. Riassumendo, avremo il seguente algoritmo.

Passo 1 Per ogni $s_n \in Stati_n$ si calcoli $f_n^*(s_n)$ e la corrispondente decisione ottima $d_n^*(s_n)$. Si ponga $k = n - 1$.

Passo 2 Per ogni $s_k \in Stati_k$ si calcoli

$$f_k^*(s_k) = \max_{d_k \in D_k} [u(d_k, s_k) + f_{k+1}^*(t(d_k, s_k))]$$

e la corrispondente decisione ottima $d_k^*(s_k)$.

Passo 3 Se $k = 1$ si ha che $f_1^*(s_1)$ é il valore ottimo del problema. Altrimenti si ponga $k = k - 1$ e si ritorni al Passo 2.

La soluzione ottima puó essere ricostruita attraverso la seguente procedura.

Passo 1 Si ponga $s_1^* = s_1$ e $k = 1$.

Passo 2 Per il blocco k la decisione ottima é $d_k^*(s_k^*)$. Si ponga

$$s_{k+1}^* = t(d_k^*(s_k^*), s_k^*).$$

Passo 3 Se $k = n$, stop: la soluzione ottima é stata ricostruita ed é rappresentata dalle decisioni

$$d_1^*(s_1^*), \ldots, d_n^*(s_n^*).$$

Altrimenti si ponga $k = k + 1$ e si ritorni al Passo 2.

Vedremo ora un esempio di algoritmo di programmazione dinamica applicato al problema dello zaino.

4.4.1 Un algoritmo di programmazione dinamica per il problema dello zaino

Prima di tutto dobbiamo definire i blocchi, gli stati, le decisioni possibili, i contributi di ogni blocco e la funzione di transizione.

1. I blocchi coincidono con gli oggetti. Avremo quindi un blocco per ogni oggetto.

2. Al blocco k i possibili stati s_k sono i valori $0, 1, \ldots, b$ che rappresentano la capacitá residua dello zaino per i blocchi da k fino a n. Quindi avremo

$$Stati_k = \{0, 1, \ldots, b\}$$

   per $k = 2, \ldots, n$, mentre $Stati_1$ contiene il solo stato b (la capacitá iniziale dello zaino).

3. Le decisioni al blocco k possono essere due se $p_k \leq s_k$ (ovvero il peso dell'oggetto k non supera la capacitá residua dello zaino) oppure una sola se $p_k > s_k$. Nel primo caso le due decisioni sono NO (non inserire l'oggetto k nello zaino) oppure SI (inserire l'oggetto k nello zaino), nel secondo caso la sola decisione possibile é NO.

4. Il contributo al blocco k sará dato dal valore dell'oggetto se la decisione é SI, mentre sará nullo se la decisione é NO. Quindi, per ogni s_k si avrá

$$u(NO, s_k) = 0 \quad u(SI, s_k) = v_k.$$

5. La funzione di transizione pone $s_{k+1} = s_k$ se la decisione é NO, e $s_{k+1} = s_k - p_k$ se la decisione é SI. Quindi

$$t(NO, s_k) = s_k \quad t(SI, s_k) = s_k - p_k.$$

6. Si noti che vale il principio di ottimalitá. Infatti, se ad un blocco k ho una capacitá residua dello zaino pari a s_k, le decisioni ottime circa gli oggeti $k, k+1, \ldots, n$ da inserire sono del tutto indipendenti da come sono giunto, attraverso le decisioni ai blocchi $1, \ldots, k-1$, ad avere la capacitá residua s_k.

7. I valori $f_n^*(0), \ldots, f_n^*(b)$ sono facilmente calcolabili. Infatti si ha

$$f_n^*(s_n) = v_n, \quad d_n^*(s_n) = SI \quad \forall\ s_n \geq p_n$$

$$f_n^*(s_n) = 0, \quad d_n^*(s_n) = NO \quad \forall\ s_n < p_n.$$

A questo punto siamo pronti ad applicare l'algoritmo all'Esempio 2. Costruiamo dapprima la Tabella 4.1 per il blocco 4. Con questa tabella possiamo costruire la Tabella 4.2 per il blocco 3. Con la Tabella 4.2 possiamo costruire la Tabella 4.3 per il blocco 2. Resta ora solo da calcolare $f_1^*(16)$ e la relativa decisone

Tabella 4.1: La funzione f_4^* con le relative decisioni ottime

s_4	f_4^*	d_4^*
0	0	NO
1	0	NO
2	0	NO
3	0	NO
4	1	SI
5	1	SI
6	1	SI
7	1	SI
8	1	SI
9	1	SI
10	1	SI
11	1	SI
12	1	SI
13	1	SI
14	1	SI
15	1	SI
16	1	SI

Tabella 4.2: La funzione f_3^* con le relative decisioni ottime

s_3	f_3^*	d_3^*
0	0	NO
1	0	NO
2	0	NO
3	0	NO
4	1	NO
5	1	NO
6	1	NO
7	1	NO
8	1	NO
9	1	NO
10	1	NO
11	1	NO
12	1	NO
13	$\max\{1, 10+0\} = 10$	SI
14	$\max\{1, 10+0\} = 10$	SI
15	$\max\{1, 10+0\} = 10$	SI
16	$\max\{1, 10+0\} = 10$	SI

Tabella 4.3: La funzione f_2^* con le relative decisioni ottime

s_2	f_2^*	d_2^*
0	0	NO
1	0	NO
2	0	NO
3	0	NO
4	1	NO
5	1	NO
6	1	NO
7	$\max\{1, 6+0\} = 6$	SI
8	$\max\{1, 6+0\} = 6$	SI
9	$\max\{1, 6+0\} = 6$	SI
10	$\max\{1, 6+0\} = 6$	SI
11	$\max\{1, 6+1\} = 7$	SI
12	$\max\{1, 6+1\} = 7$	SI
13	$\max\{10, 6+1\} = 10$	NO
14	$\max\{10, 6+1\} = 10$	NO
15	$\max\{10, 6+1\} = 10$	NO
16	$\max\{10, 6+1\} = 10$	NO

ottima. Si ha:

$$f_1^*(16) = \max\{10, 8+6\} = 14 \quad d_1^*(16) = SI.$$

Possiamo quindi concludere che il valore ottimo del nostro problema é 14 (come giá sapevamo dalla risoluzione tramite algoritmo branch-and-bound). Vediamo ora di ricostruire anche la soluzione ottima. Si ha:

$$\begin{aligned} d_1^*(16) &= SI \quad s_2^* = t(SI, 16) = 9 \\ d_2^*(s_2^*) &= SI \quad s_3^* = t(SI, 9) = 2 \\ d_3^*(s_3^*) &= NO \quad s_4^* = t(NO, 2) = 2 \\ d_4^*(s_4^*) &= NO \end{aligned}$$

Da cui si conclude che la soluzione ottima si ottiene inserendo nello zaino gli oggetti 1 e 2, ovvero $N^* = \{1, 2\}$.

Si puó dimostrare che la procedura di programmazione dinamica appena vista richiede un numero di operazioni pari a $O(nb)$. Tale numero di operazioni non rapppresenta una complessitá polinomiale (perché?).

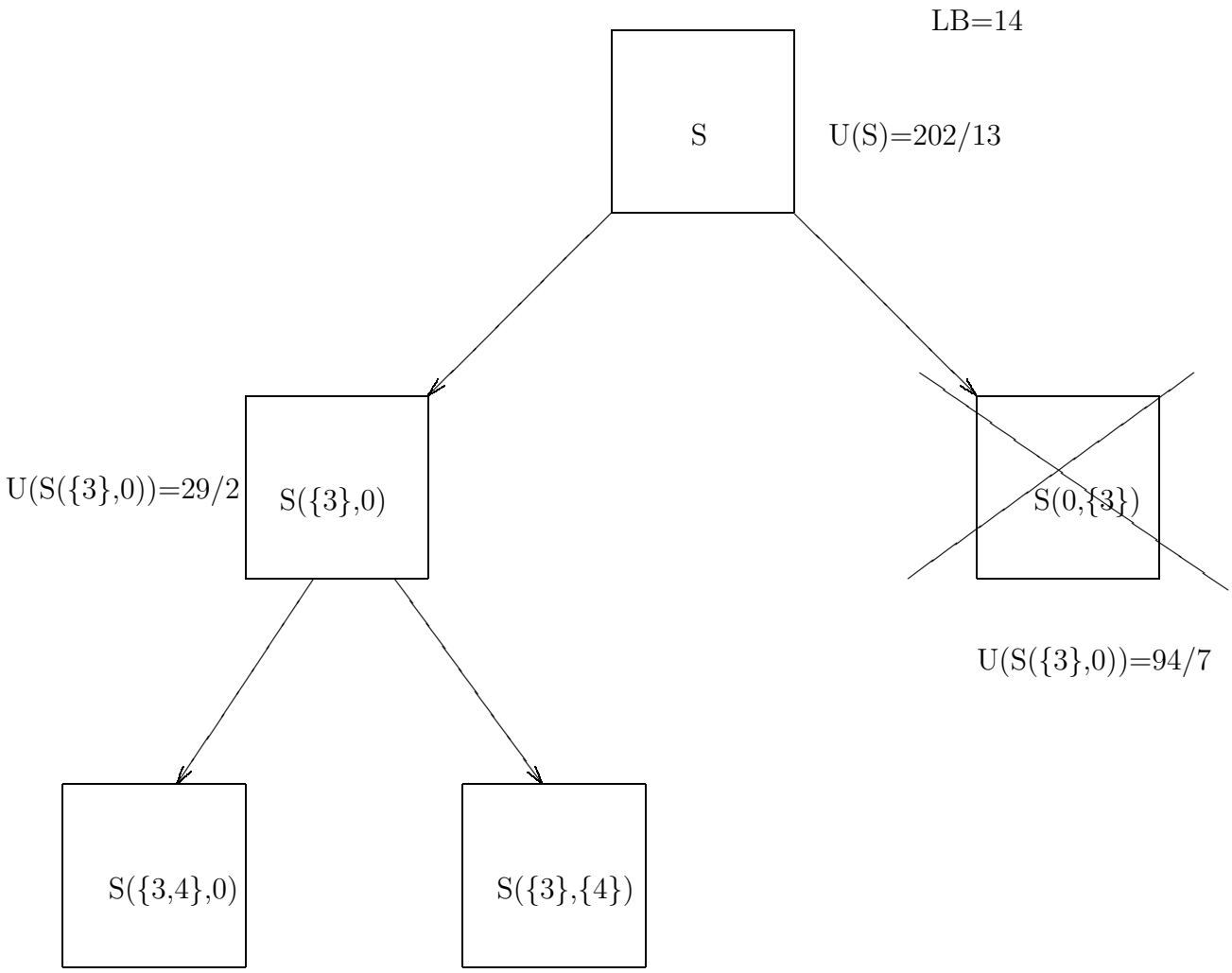


Figura 4.6: Albero di branch-and-bound dopo la seconda iterazione.

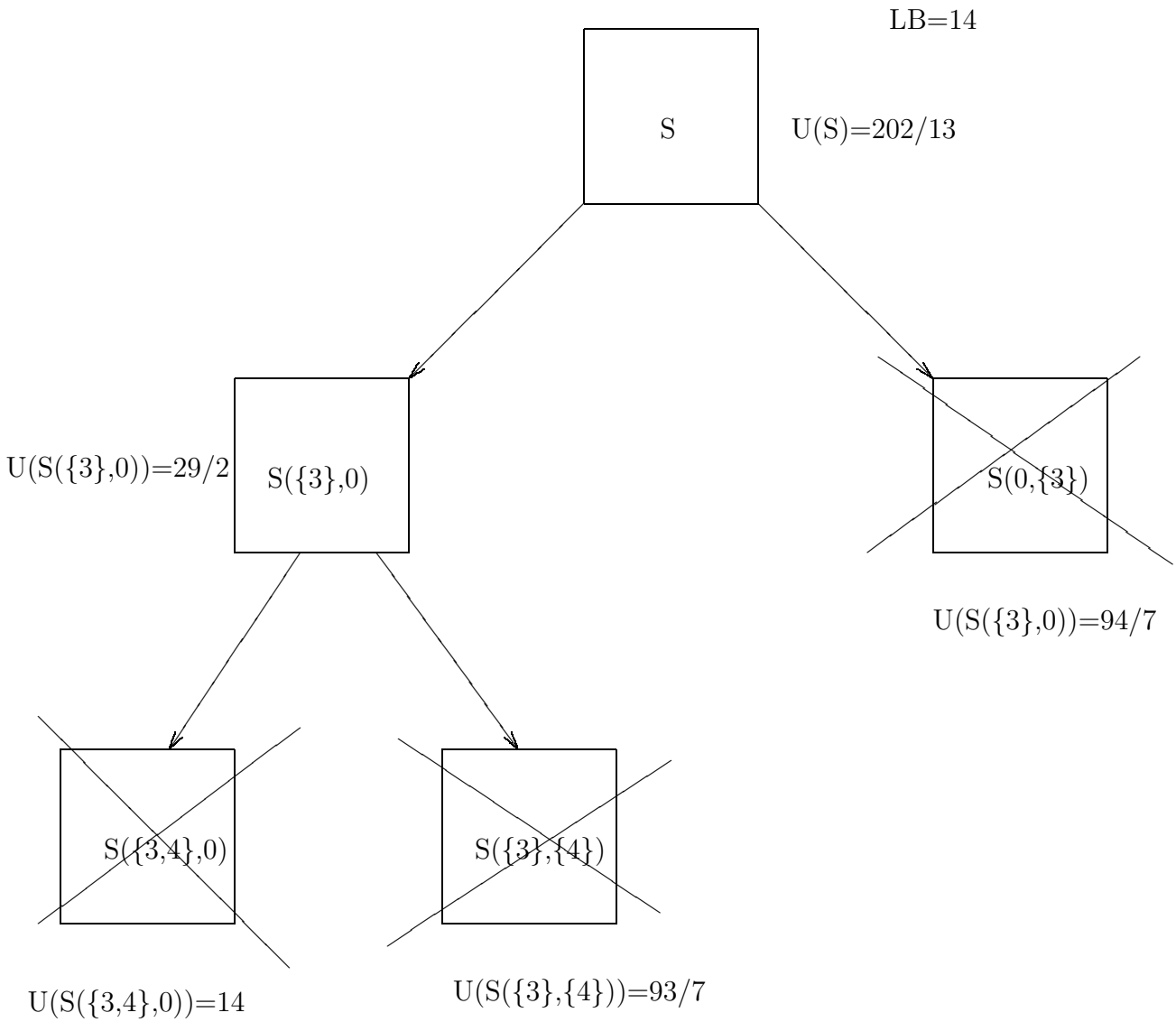


Figura 4.7: Albero di branch-and-bound dopo la terza iterazione.

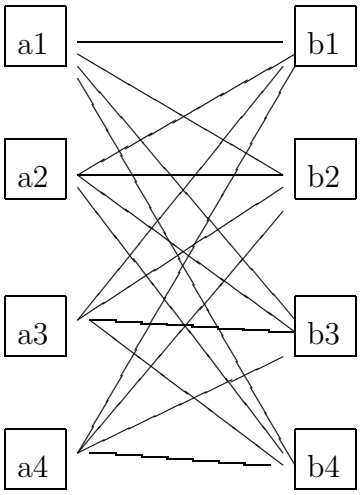


Figura 4.8: Il grafo bipartito G' associato al grafo G dell'esempio.

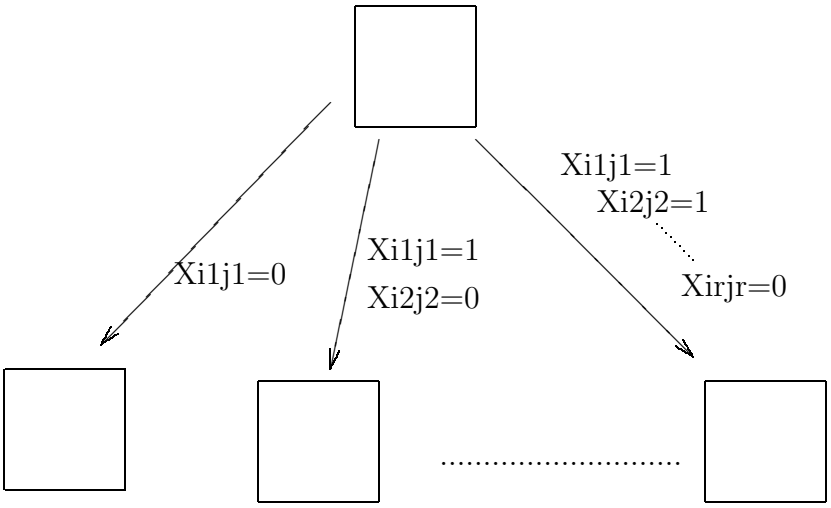


Figura 4.9: Branching per il problema TSP.

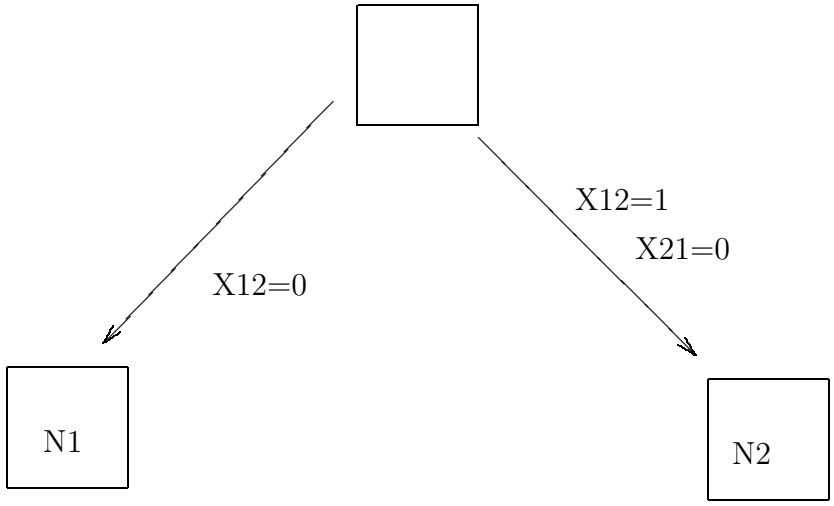


Figura 4.10: Branching del nodo radice nel problema di esempio.

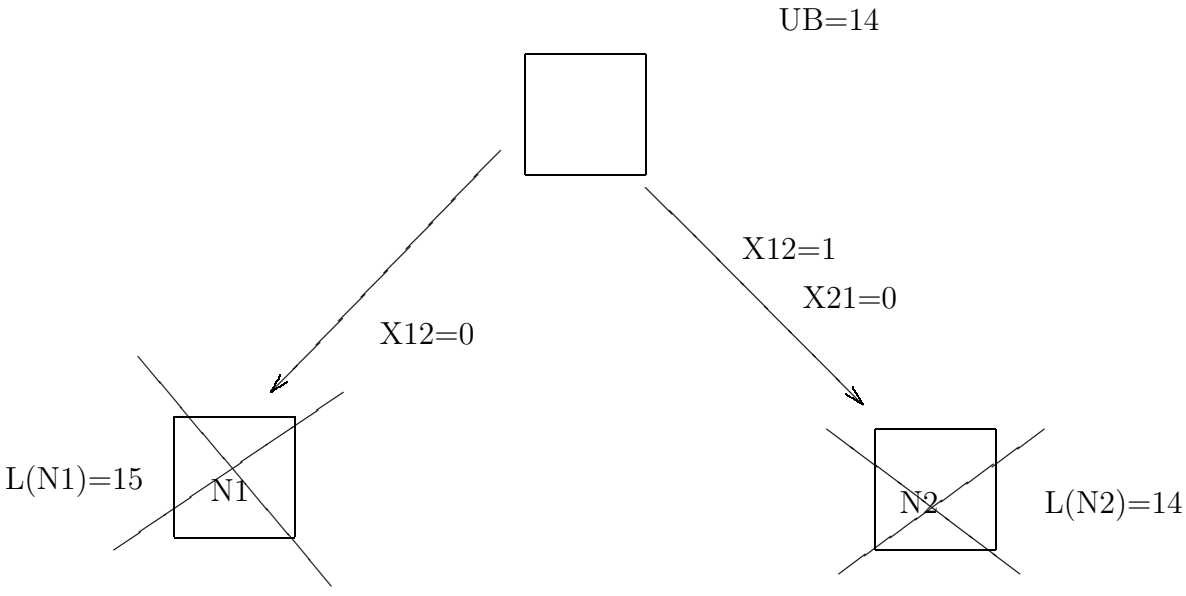


Figura 4.11: Albero di branch-and-bound nell'iterazione conclusiva.

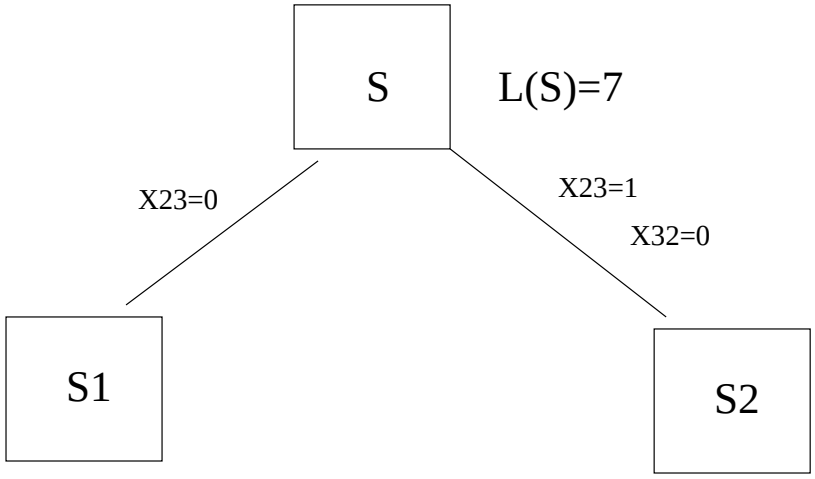


Figura 4.12:

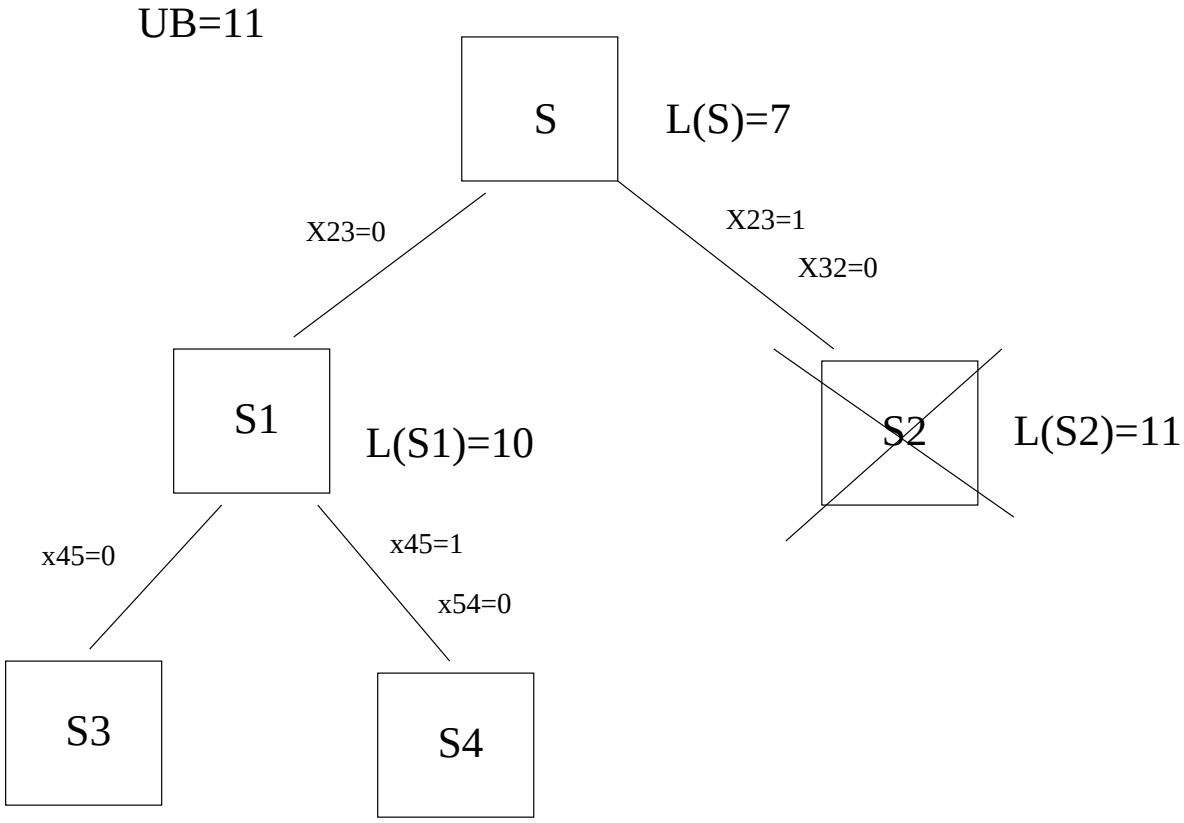


Figura 4.13:

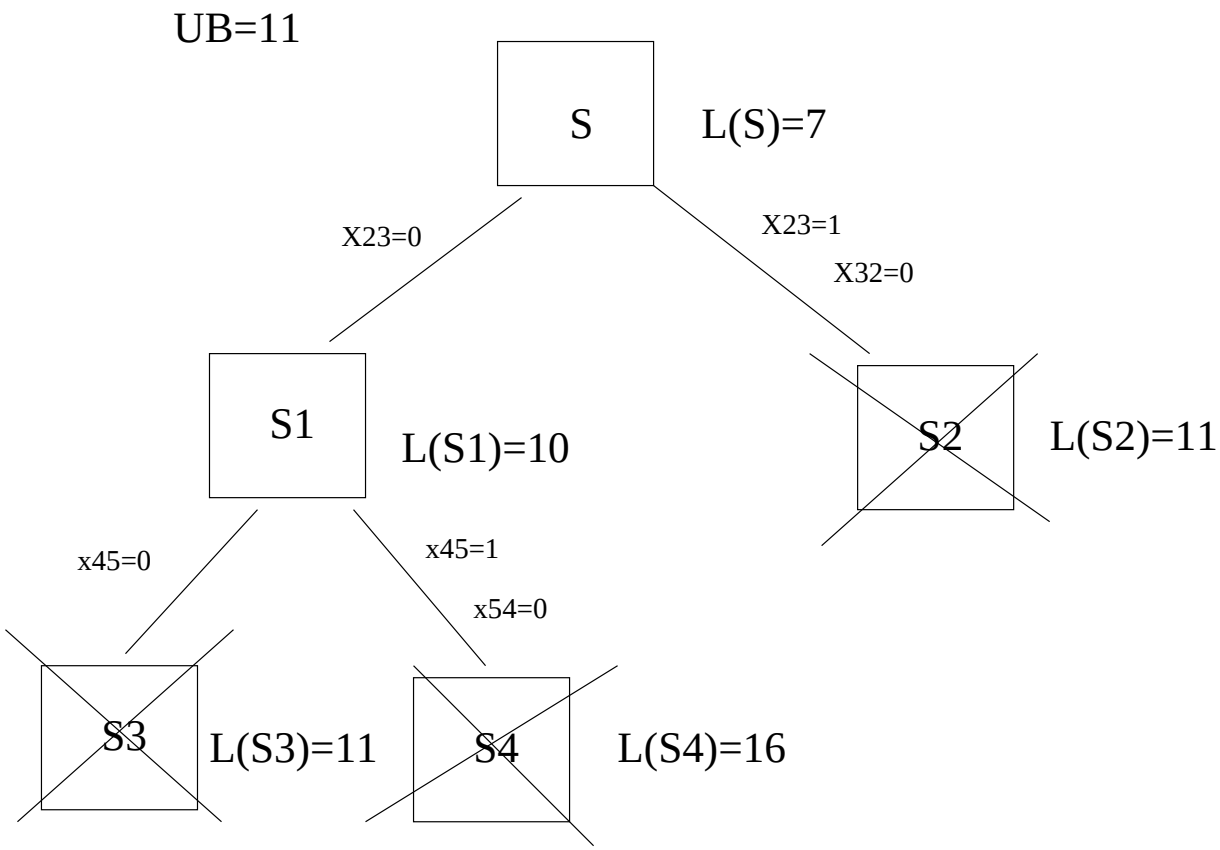


Figura 4.14:

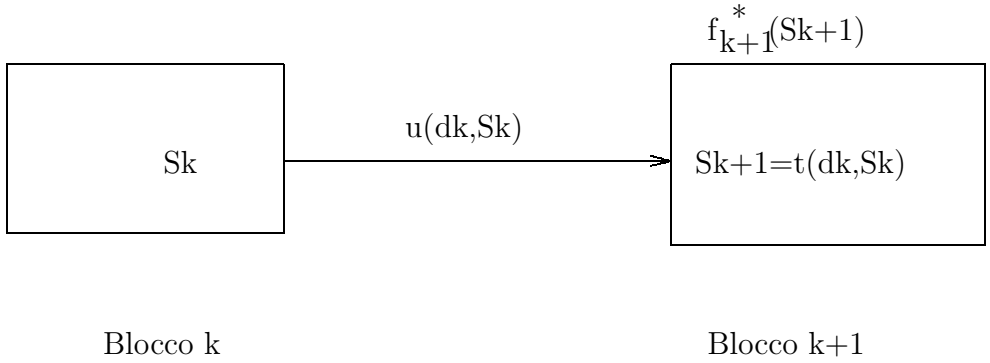


Figura 4.15: La transizione dallo stato s_k del blocco k allo stato s_{k+1} del blocco $k+1$ quando si prende la decisione d_k.

Capitolo 5

Algoritmi di approssimazione e tecniche euristiche

Nel capitolo 2 abbiamo introdotto le definizioni di algoritmi di approssimazione e di tecniche euristiche. In questo capitolo ne daremo degli esempi. In particolare, mostreremo dapprima un esempio di algoritmo di approssimazione applicato ad un caso speciale di problema TSP, il problema TSP metrico. In seguito vedremo alcuni esempi di tecniche euristiche.

5.1 Un algoritmo di approssimazione per il problema TSP metrico

Il problema TSP *metrico* é un problema TSP simmetrico in cui si introducono delle restrizioni sui possibili valori che possono assumere le distanze lungo gli archi di un grafo. Come vedremo le restrizioni modificano (in particolare, riducono) la difficoltá del problema. Il problema TSP metrico comprende i problemi TSP simmetrici in cui le distanze d_{ij} degli archi sono tutte non negative ed inoltre soddisfano la diseguaglianza triangolare:

$$\forall\ i, j, k\ :\quad d_{ij} + d_{jk} \geq d_{ik} \tag{5.1}$$

Consideriamo ora il seguente algoritmo, denominato algoritmo *Double Spanning Tree*, abbreviato con DST nel seguito.

Algoritmo DST

Passo 1 Dato il grafo $G = (V, A)$, si determini un albero di supporto a costo minimo di tale grafo e lo si indichi con $T = (V, A_T)$.

Passo 2 Si duplichi ogni arco in A_T assegnando ad ogni arco duplicato la stessa distanza dell'arco originale. Sul grafo risultante si determini un *ciclo euleriano*, ovvero un ciclo che partendo da un nodo attraversi tutti gli archi del grafo una ed una sola volta.

Passo 3 Dalla sequenza di nodi

$$i_1 \to i_2 \to \cdots \to i_{2n-2} \to i_{2n-1} = i_1$$

si eliminino tutte le ripetizioni di nodi a parte quella dell'ultimo nodo. La sequenza di nodi risultante é un circuito hamiltoniano.

Si puó dimostrare che l'algoritmo DST richiede tempo di esecuzione polinomiale rispetto alla dimensione dell'istanza. Si noti che un grafo ammette un ciclo euleriano se e solo se tutti i suoi nodi hanno grado pari. Nel nostro caso, avendo raddoppiato tutti gli archi dell'albero di supporto ottimo (e quindi i gradi dei nodi di tale albero), la condizione 'e ovviamente soddisfatta. Restya ancora da vedere come individuare un ciclo euleriano. Si puó utilizzare una procedura che, fissato un nodo radice dell'albero, consiste in pratica in una visita in profonditá dell'albero:

Algoritmo per determinare un ciclo euleriano

Passo 1 Dato l'albero $T = (V, A_T)$, si fissi un suo nodo radice v^* in modo arbitrario. Si ponga:

$$S = \emptyset, \quad i_1 = v^*, \quad k = 2, \quad w = v^*.$$

Passo 2 Se w ha nodi figli in $V \backslash S$, allora si selezioni un suo nodo figlio $z \in V \backslash S$, si ponga

$$w = z, \quad i_k = z, \quad k = k + 1$$

e si ripeta il Passo 2.
Altrimenti (cioé se w non ha nodi figli in $V \setminus S$): se $w \neq v^*$, si risalga al nodo padre y di w, si ponga

$$S = S \cup \{w\}, \quad w = y, \quad i_k = y, \quad k = k + 1$$

e si ripeta il Passo 2; altrimenti (se $w = v^*$) ci si arresti.

La procedura DST viene ora illustrata su un esempio.

Esempio 31 *Sia dato il grafo in Figura 5.1 con le distanze indicate su ciascun arco (si noti che ci troviamo nel caso simmetrico e quindi gli archi non sono orientati, il che significa che per ogni arco (i, j) la distanza per andare da i a j é uguale a quella per andare da j a i). Si puó verificare che gli archi del grafo soddisfano la diseguaglianza triangolare e quindi il problema é un'istanza di problema di TSP metrico. Nel Passo 1 si deve determinare un albero di supporto a costo minimo per tale grafo. Tale albero (ottenuto ad esempio con*

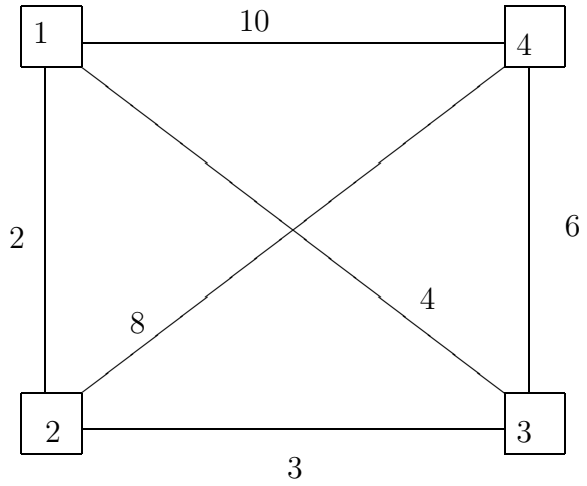


Figura 5.1:

la nota procedura greedy) é indicato in Figura 5.2. Al Passo 2 si richiede di duplicare gli archi dell'albero (si veda la Figura 5.3) e di determinare un ciclo euleriano sul grafo ottenuto in questo modo. Partendo dal nodo 1 possiamo ottenere il seguente ciclo euleriano:

$$1 \to 2 \to 3 \to 4 \to 3 \to 2 \to 1.$$

Nel Passo 3 dobbiamo eliminare le ripetizioni di nodi (tranne quella relativa all'ultimo nodo) ed otteniamo il seguente circuito hamiltoniano:

$$1 \to 2 \to 3 \to 4 \to 1.$$

Il valore di tale circuito é 21. Si noti che non é quello ottimo: il circuito

$$1 \to 2 \to 4 \to 3 \to 1$$

ha valore pari a 20.

Come visto, l'algoritmo non é esatto per il problema del TSP metrico ma la seguente osservazione evidenzia che si tratta di un algoritmo di approssimazione per questo problema.

Osservazione 8 *Per il TSP metrico l'algoritmo DST é un algoritmo di* 1*-approssimazione.*

Dimostrazione
Sia $T = (V, A_T)$ l'albero di supporto a costo minimo individuato al Passo 1 dell'algoritmo e sia

$$i_1 \to i_2 \to \cdots \to i_{2n-2} \to i_{2n-1} = i_1 \tag{5.2}$$

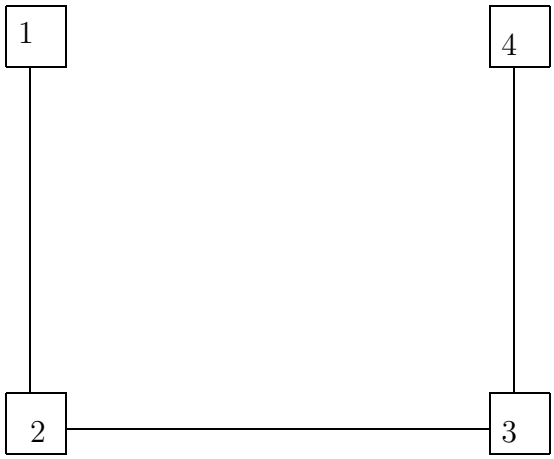


Figura 5.2:

il ciclo euleriano individuato al Passo 2 dell'algoritmo dopo aver raddoppiato gli archi dell'albero T. Indichiamo con

$$A_Q = \{(i_j, i_{j+1}) \ : \ j = 1, \ldots, 2n-2\}$$

l'insieme degli archi di tale ciclo. Si noti che

$$\sum_{e \in A_Q} d_e = 2 \sum_{e \in A_T} d_e. \tag{5.3}$$

Quando nel Passo 3. rimuoviamo un nodo i_j in quanto giá presente, sostituiamo nel cammino la coppia di archi

$$(i_{j-1}, i_j) \quad (i_j, i_{j+1})$$

con il singolo arco

$$(i_{j-1}, i_{j+1}).$$

Ma per la diseguaglianza triangolare (5.1) si ha

$$d_{i_{j-1}, i_j} + d_{i_j, i_{j+1}} \geq d_{i_{j-1}, i_{j+1}}.$$

Ripetendo questo ragionamento per ogni nodo rimosso, si ha che il circuito hamiltoniano $C = (V, A_C)$

$$i'_1 \to \ldots \to i'_n \to i'_1$$

ottenuto dal ciclo euleriano (5.2) con l'eliminazione dei nodi ripetuti, ha distanza complessiva certamente non superiore a quella del ciclo euleriano. In base a (5.3) avremo dunque

$$\sum_{e \in A_C} d_e \leq \sum_{e \in A_Q} d_e = 2 \sum_{e \in A_T} d_e. \tag{5.4}$$

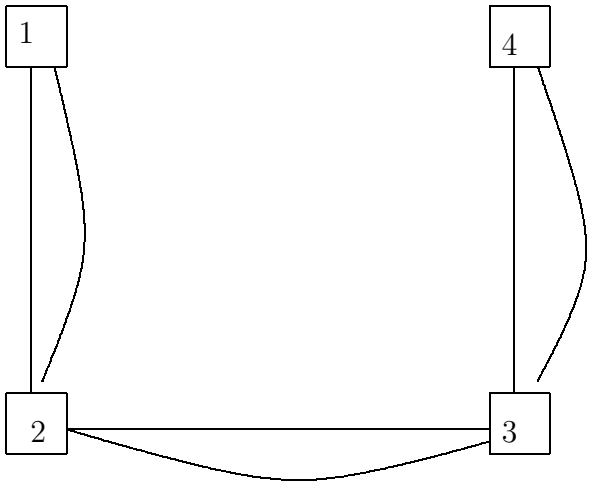


Figura 5.3:

Ma per la non negativitá degli archi si ha che il circuito hamiltoniano $C^* = (V, A_{C^*})$ sluzione ottima del problema di TSP metrico, ha valore complessivo non inferiore a quello dell'albero di supporto a costo minimo, cioé

$$\sum_{e \in A_{C^*}} d_e \geq \sum_{e \in A_T} d_e. \tag{5.5}$$

Infatti rimuovendo un qualsiasi arco del circuito si riduce il valore complessivo dell'insieme di archi (per la non negativitá del peso dell'arco rimosso) e si ottiene un grafo ancora connesso ed aciclico, ovvero un albero di supporto il cui valore non puó essere inferiore a quello dell'albero T che ha costo minimo. Da (5.4) e (5.5) si ricava che

$$\sum_{e \in A_C} d_e \leq 2 \sum_{e \in A_T} d_e \leq 2 \sum_{e \in A_{C^*}} d_e,$$

o, equivalentemente,

$$\frac{\sum_{e \in A_C} d_e}{\sum_{e \in A_{C^*}} d_e} \leq 2,$$

e ció dimostra che l'algoritmo DST é un algoritmo di 1-approssimazione per il problema TSP metrico.

Ci si puó chiedere se é possibile fare di meglio per il problema TSP metrico, ovvero se esiste un algoritmo di ε-approssimazione con valore di $\varepsilon < 1$. Un tale algoritmo esiste, anche se non lo vedremo, ed il corrispondente valore di ε é 0.5. Si puó anche peró dimostrare il seguente risultato negativo.

Osservazione 9 *Per il problema TSP metrico esiste un $\overline{\varepsilon} > 0$ tale che per ogni $\varepsilon \leq \overline{\varepsilon}$ il problema di ε-approssimazione associato al TSP metrico é $NP-$completo.*

Questa osservazione ci consente di collocare il problema TSP metrico nel Caso 3 tra quelli visti nella sezione 2.2.4 del capitolo 2.

5.2 Un FPTAS per il problema KNAPSACK

Vogliamo ora definire un FPTAS (Fully Polynomial Time Approximation Scheme, ovvero uno schema di approssimazione completamente polinomiale) per il problema KNAPSACK. Questo è una classe di algoritmi di ε-approssimazione che risultano essere polinomiali sia rispetto alla dimensione del problema KNAPSACK sia rispetto all'inverso $\frac{1}{\varepsilon}$ della precisione richiesta. Prima di arrivare a definire un FPTAS abbiamo bisogno di introdurre un nuovo metodo esatto (ovviamente non polinomiale) di risoluzione per il problema.

5.2.1 Un metodo di risoluzione esatto alternativo

In alternativa al metodo di programmazione dinamica visto in precedenza per risolvere il problema KNAPSACK, si può utilizzare anche questo altro metodo, sempre di programmazione dinamica.

Inizializzazione Si ponga $M_0 = \{(\emptyset, 0)\}$. Si ponga $j = 1$.

Passo 1 Si ponga $M_j = \emptyset$.

Passo 2 Per ogni terna $(N, p, v) \in M_{j-1}$, aggiungi in M_j l'elemento (N, p, v) e, se $p_j + p \leq b$, aggiungi in M_j anche $(N \cup \{j\}, p + p_j, v + v_j)$.

Passo 3 Per ogni coppia di elementi (S, p, v) e (S', p', v') in M_j, se

$$p' \geq p \quad \text{e} \quad v' = v,$$

allora scarta la terna (S', p', v').

Passo 4 Se $j = n$, allora restituisci come soluzione ottima del problema KNAPSACK la coppia in M_n con il massimo valore della seconda componente, altrimenti poni $j = j + 1$ e torna al Passo 1.

Si noti che ad ogni iterazione ogni coppia dell'insieme M_j ha come prima componente una soluzione ammissibile del problema KNAPSACK contenente *solo* i primi j oggetti, come seconda componente il relativo peso e come terza componente il relativo valore dell'obiettivo. Inoltre, il Passo 3 esclude quelle soluzioni ammissibili che sono dominate da altre, ovvero con lo stesso valore dell'obiettivo (uguale terza componente) ma con peso complessivo (seconda componente) superiore o uguale a quello di altre soluzioni. Si noti quindi che in M_n ci sono tutte le soluzioni ammissibili contenenti tutti gli n oggetti e quella con seconda componente massima è quindi anche soluzione ottima del problema KNAPSACK.

Volendo essere più precisi, la regola

$$p' \geq p \quad \text{e} \quad v' = v,$$

utilizzata per scartare la terna (S', p', v'), può essere rafforzata come segue

$$p' \geq p \quad \text{e} \quad v' \leq v.$$

Tuttavia, nel seguito, per semplificare l'analisi dell'algoritmo, utilizzeremo la regola più debole in cui si richiede l'uguaglianza tra i valori.

Calcoliamo ora il numero di operazioni eseguite dalla procedura. Sia v^* il valore ottimo del problema.

Osservazione 1 *La procedura di risoluzione sopra descritta richiede un numero di operazioni $O(nv^*)$.*

Dimostrazione In ogni insieme M_{j-1} abbiamo al più v^* elementi (ve ne è al più uno per ogni possibile valore della seconda componente e i possibili valori della seconda componente sono al più v^*). L'operazione di aggiornamento al Passo 2 deve essere fatta su al più v^* coppie per ognuna delle quali si devono fare al più due somme. Quindi, in tutto sono richieste al più $O(v^*)$ operazioni. Al Passo 3 si devono individuare eventuali coppie di soluzioni con lo stesso valore dell'obiettivo e scartare quella con peso maggiore (o una delle due se hanno lo stesso peso). Essendoci in M_j al più $2v^*$ soluzioni, tale operazione si può implementare in modo da dover eseguire $O(v^*)$ operazioni. Tenuto conto che i passi dell'algoritmo devono essere ripetuti n volte, abbiamo un totale di $O(nv^*)$ operazioni.

5.2.2 Descrizione del FPTAS

Dato un intero positivo t, supponiamo di modificare i profitti del nostro problema nel modo seguente

$$\bar{v}_i = \left\lfloor \frac{v_i}{10^t} \right\rfloor \times 10^t,$$

il che equivale ad azzerare le ultime t cifre del profitto. Ad esempio, per $t = 2$ e $v_i = 3453$ abbiamo $\bar{v}_i = 3400$. Ovviamente, possiamo risolvere il problema con i nuovi profitti $\bar{v}_i$, dividendoli tutti per 10^t e moltiplicando il valore ottimo alla fine per 10^t.

Osservazione 2 *Risolvere il problema modificato richiede un numero di operazioni pari al più a $O(n^2 v_{max} 10^{-t})$ dove v_{max} denota il massimo tra i profitti degli n oggetti.*

Dimostrazione Notando che il valore ottimo del problema ottenuto dividendo i profitti $\bar{v}_i$ per 10^t non può essere superiore a $10^{-t}v^*$, l'Osservazione 1 ci dice che risolvere il problema modificato con la procedura vista in precedenza richiede un numero di operazioni pari al più a $O(n10^{-t}v^*)$. Indicando con v_{max} il massimo tra i profitti degli oggetti, possiamo anche scrivere che il numero di operazioni è $O(n^2 v_{max} 10^{-t})$ (ovviamente si ha $v^* \leq nv_{max}$).

Sia ora N' la soluzione del problema modificato e N^* quella del problema originario. Introduciamo inoltre una terza soluzione

$$N'' = \begin{cases} N' & \text{se } \sum_{i \in N'} v_i \geq v_{max} \\ \{i_{max}\} & \text{altrimenti} \end{cases}$$

dove i_{max} è l'oggetto con il profitto massimo v_{max}. In pratica, N'' coincide con N' a meno che gli oggetti in N' abbiano un valore complessivo inferiore a quello dell'oggetto con profitto massimo, nel qual caso si utilizza al posto di N' la soluzione costituita dal solo oggetto di profitto massimo. Rispetto al calcolo di N', il calcolo di N'' richiede un'ulteriore somma di al più n addendi, ma questa non modifica l'ordine di grandezza del numero di operazioni individuato nell'Osservazione 2. Si dimostra il seguente risultato.

Osservazione 3 *Si ha che*

$$\frac{\sum_{i \in N^*} v_i}{\sum_{i \in N''} v_i} \leq 1 + \frac{n10^t}{v_{max}},$$

Dimostrazione Calcoliamo un bound dal di sopra per $\sum_{i \in N^*} v_i - \sum_{i \in N'} v_i$. Abbiamo:

$$\sum_{i \in N^*} v_i \geq \sum_{i \in N'} v_i \geq \sum_{i \in N'} \bar{v}_i \geq \sum_{i \in N^*} \bar{v}_i \geq \sum_{i \in N^*} (v_i - 10^t) \geq \sum_{i \in N^*} v_i - n10^t,$$

da cui

$$\sum_{i \in N^*} v_i - \sum_{i \in N'} v_i \leq n10^t.$$

Dal momento che $\sum_{i \in N''} v_i \geq \sum_{i \in N'} v_i$ abbiamo anche:

$$\sum_{i \in N^*} v_i - \sum_{i \in N''} v_i \leq n10^t.$$

Se si dividono entrambi i membri per $\sum_{i \in N''} v_i$ abbiamo:

$$\frac{\sum_{i \in N^*} v_i}{\sum_{i \in N''} v_i} \leq 1 + \frac{n10^t}{\sum_{i \in N''} v_i}.$$

da cui, tenendo conto che si ha anche $\sum_{i \in N''} v_i \geq v_{max}$, si arriva a:

$$\frac{\sum_{i \in N^*} v_i}{\sum_{i \in N''} v_i} \leq 1 + \frac{n10^t}{v_{max}}.$$

come si voleva dimostrare.

Si prenda ora

$$t = \left\lfloor \log_{10} \left(\frac{\varepsilon v_{max}}{n} \right) \right\rfloor.$$

Si noti che in tal caso, in base all'Osservazione 2, il tempo di esecuzione della procedura per risolvere il problema modificato è $O\left(\frac{n^3}{\varepsilon}\right)$, ovvero è polinomiale sia rispetto al numero di oggetti (e quindi rispetto alla dimensione del problema) sia rispetto all'inverso $\frac{1}{\varepsilon}$ della precisione richiesta. Inoltre, per la definizione di t si ha

$$\frac{n10^t}{v_{max}} \leq \varepsilon$$

e quindi, in base all'Osservazione 3, abbiamo anche

$$\frac{\sum_{i \in N^*} v_i}{\sum_{i \in N''} v_i} \leq 1 + \varepsilon.$$

Possiamo quindi concludere che la procedura che ci restituisce N'' rappresenta un FPTAS per il problema KNAPSACK.

5.3 Euristiche

Il fatto che un problema sia difficile e non ci possiamo aspettare di determinare in tempi ragionevoli per ogni istanza la soluzione ottima non vuole comunque dire che si debba rinunciare ad affrontare il problema. Pur con la consapevolezza di avere a che fare con problemi difficili, si puó cercare di determinare in tempi ragionevoli una buona (anche se non necessariamente ottima) soluzione. Per questo si utilizzano procedure *euristiche*. Un'euristica deve avere le due caratteristiche appena accennate e giá citate nel capitolo 2:

- avere tempi di esecuzione che non crescano troppo rapidamente rispetto alla dimensione del problema (polinomiali con esponente non troppo elevato);
- restituire soluzioni che siano almeno per molte istanze del problema ottime o comunque vicine a quelle ottime.

Un'euristica é quindi un compromesso tra due esigenze contrastanti: la qualitá della soluzione ottenuta (che dovrebbe essere il piú vicino possibile al valore ottimo) e la rapiditá con cui viene restituita una soluzione (non necessariamente ottima). Nelle euristiche si rinuncia alla garanzia di ottenere *sempre* una soluzione ottima (cosa che richiederebbe tempi troppo elevati) per poter ottenere una risposta in tempi accettabili. Di seguito vedremo alcuni esempi di euristiche.

5.3.1 Algoritmo greedy

In un algoritmo greedy la soluzione viene costruita un passo per volta facendo ad ogni passo una scelta greedy (golosa) ovvero una scelta che non é necessariamente la migliore in assoluto ma é la migliore in quel dato momento. Un esempio di tecnica greedy é giá stato visto nel capitolo 2 dove abbiamo introdotto un algoritmo greedy per il problema MST. In quel caso la scelta greedy

consisteva nel prendere in esame ad ogni iterazione l'arco con peso minimo tra tutti quelli non ancora analizzati. Per tale algoritmo greedy é improprio parlare di euristica. Sappiamo infatti che esso risolve in modo *esatto* il problema MST. Ma vediamo ora un altro esempio in cui siamo meno fortunati e l'algoritmo greedy non restituisce sempre una soluzione ottima. Sia dato un problema TSP che per semplicitá supporremo su di un grafo completo $G = (V, A)$. Un algoritmo greedy per tale problema é il seguente.

Passo 1 Si fissi un nodo di partenza i e lo si inserisca in un insieme W, ovvero $W = \{i\}$. Si ponga $r = i$.

Passo 2 Si consideri il nodo $s \in V \setminus W$ con "distanza" minima dal nodo r, cioé

$$d_{rs} = \min_{j \in V \setminus W} d_{rj},$$

dove d_{ij} denota la "distanza" da percorrere lungo l'arco (i, j). Si faccia seguire r da s nel circuito.

Passo 3 Si ponga $W = W \cup \{s\}$ e $r = s$. Se $W = V$ ci si arresta chiudendo il circuito andando da s al nodo iniziale i. Altrimenti si ritorni al Passo 2.

In pratica ad ogni iterazione ci si muove dall'ultimo nodo raggiunto, il nodo r, al nodo s che é il piú vicino (scelta greedy) a r tra tutti quelli non ancora visitati dal circuito. Si puó dimostrare che il numero di operazioni di tale algoritmo é dell'ordine di n^2, dove $n =\mid V \mid$. Si tratta quindi di un algoritmo con complessitá polinomiale. Vediamo ora di applicare l'algoritmo sull'esempio in Figura 5.4 dove $M > 0$. Partendo dal nodo 1 l'algoritmo si sposta verso il nodo 3. Da

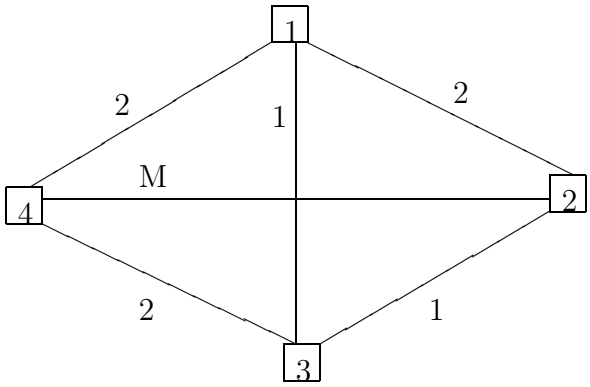


Figura 5.4: Un grafo G su cui applicare l'algoritmo greedy per determinare il circuito hamiltoniano a "distanza" minima.

questo si muove verso il nodo 2, dal nodo 2 si va verso il nodo 4 ed infine dal nodo 4 si chiude il circuito ritornando al nodo 1. L'algoritmo restituisce quindi il circuito C_1

$$1 \to 3 \to 2 \to 4 \to 1$$

con $f(C_1) = 1 + 1 + M + 2 = 4 + M$. Questa é la soluzione ottima solo per $M \leq 3$ ma per $M > 3$ la soulzione ottima é il circuito C_2

$$1 \to 2 \to 3 \to 4 \to 1$$

con $f(C_2) = 2 + 1 + 2 + 2 = 7$. Si puó anche notare che per $M > 3$

$$\frac{f(C_1)}{f(C_2)} = \frac{4 + M}{7}.$$

Al crescere di M il rapporto cresce all'infinito, mostrando quindi come giá tra queste piccole istanze sia sempre possibile trovarne alcune per cui questa euristica non é in grado di risolvere il problema di r-approssimazione per ogni $r > 0$. Questo non é altro che una conferma di quanto giá visto in precedenza: per il problema TSP é improbabile che esista una procedura di risoluzione polinomiale che risolva il problema di r-approssimazione per *ogni* istanza di tale problema e per ogni $r > 0$.

5.3.2 Ricerca locale

Per poter definire un algoritmo di ricerca locale dobbiamo introdurre il concetto di vicinanza per un elemento della regione ammissibile S.

Definizione 14 *Una* vicinanza N *nella regione ammissibile* S *é definita come una funzione*

$$N : S \to 2^S$$

che ad ogni elemento $x \in S$ *associa un sottinsieme di punti di* S *detti* vicini *di* x*.*

Dato un problema di ottimizzazione combinatoria non esiste necessariamente un'unica vicinanza per esso.

Esempio 32 *Si consideri il problema* TSP*. Dato un circuito hamiltoniano* C *la vicinanza* $N_k(C)$ *con* $k \leq n$ *é costituita da tutti i circuiti hamiltoniani ottenibili rimuovendo* k *archi da* C *ed aggiungendo altri* k *archi (non necessariamente diversi dai precedenti). Per ogni* $k = 2, 3, \ldots, n$ *si ha una diversa vicinanza. Nell'esempio di Figura 5.5 se considero il circuito* C

$$1 \to 2 \to 3 \to 4 \to 5 \to 1$$

la vicinanza $N_2(C)$ *é costituita dai 6 circuiti hamiltoniani*

$$1 \to 2 \to 3 \to 4 \to 5 \to 1$$
$$1 \to 3 \to 2 \to 4 \to 5 \to 1$$
$$1 \to 4 \to 3 \to 2 \to 5 \to 1$$
$$1 \to 2 \to 4 \to 3 \to 5 \to 1$$
$$1 \to 2 \to 5 \to 4 \to 3 \to 1$$
$$1 \to 2 \to 3 \to 5 \to 4 \to 1$$

Per $k = 5$ *si avrá invece che* $N_5(C)$ *comprende tutti i 12 circuiti hamiltoniani.*

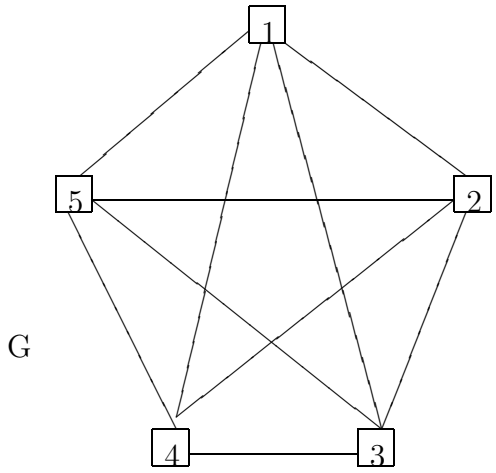


Figura 5.5: Un grafo completo G con 5 nodi.

Una volta fissata una vicinanza N é possibile introdurre la definizione di ottimo locale (limitata ad un problema di minimo ma facilmente estendibile a problemi di massimo).

Definizione 15 *Un elemento $\overline{x} \in S$ si definisce ottimo locale (rispetto alla vicinanza N) se*

$$\forall\ y \in N(\overline{x}) \quad f(y) \geq f(\overline{x}),$$

cioé un punto é un ottimo (minimo) locale se ha valore della funzione obiettivo f non peggiore (non superiore) rispetto a tutti i suoi vicini.

Si noti che un punto di ottimo globale é sempre anche un ottimo locale indipendentemente dalla vicinanza. Non é invece sempre vero il viceversa. Una vicinanza in cui ogni ottimo locale é anche ottimo globale viene detta *vicinanza esatta*. Nel problema TSP la vicinanza N_n (in cui ogni circuito hamiltoniano ha come vicini tutti i circuiti hamiltoniani) é esatta, mentre la vicinanza N_2 non é esatta (vi sono ottimi locali che non sono ottimi globali).

Esempio 33 *Come ulteriore esempio esaminiamo il problema illustrato in Figura 5.6 dove S é la griglia di punti*

$$\{(i,j):\ i=1,\ldots,5\ \ j=1,\ldots,5\}$$

e la funzione obiettivo (i cui valori sono riporati all'interno dei nodi della griglia) é data da

$$f(i,j) = 25(i-2)^2(i-4)^2 + i + 25(j-2)^2(j-4)^2 + j. \tag{5.6}$$

La struttura di vicinanza é definita nel modo seguente:

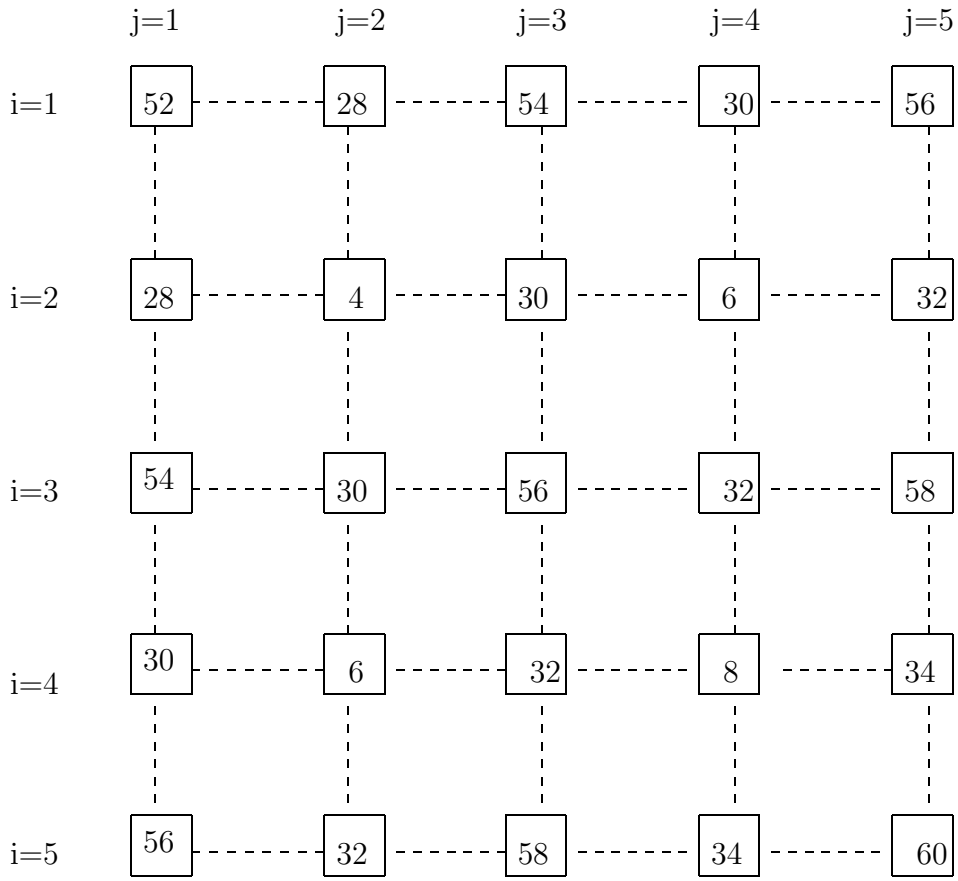


Figura 5.6: Un problema di ottimizzazione combinatoria con una possibile struttura di vicinanza.

$$\begin{aligned} N(i,j) &= \{(i,h) \in S:\ h = j-1, j, j+1\} \cup \\ &\quad \cup \{(h,j) \in S:\ h = i-1, i, i+1\} \end{aligned}$$

(nella figura ogni punto (i,j) *é collegato ai suoi vicini tramite un arco). Si noti che l'ottimo globale é il punto* $(2,2)$ *ma vi sono anche tre ottimi locali non globali:* $(2,4), (4,2)$ *e* $(4,4)$*. La vicinanza quindi non é esatta. Se definiamo un'altra vicinanza*

$$\begin{aligned} N'(i,j) &= \{(i,h) \in S:\ h = j-2, j-1, j, j+1, j+2\} \cup \\ &\quad \cup \{(h,j) \in S:\ h = i-2, i-1, i, i+1, i+2\} \end{aligned}$$

(si noti che $N'(i,j) \supset N(i,j)$ *per ogni coppia* (i,j)*), abbiamo invece un unico ottimo locale e globale, il punto* $(2,2)$*, e quindi la vicinanza é esatta.*

Dato un problema di ottimizzazione combinatoria (f, S) ed una vicinanza N per esso, possiamo introdurre un algoritmo di ricerca locale.

Passo 1 Sia $x_0 \in S$ e si ponga $k = 0$.

Passo 2 Se per ogni $y \in N(x_k)$ si ha $f(y) \geq f(x_k)$, ovvero se x_k é un ottimo locale, allora ci arrestiamo restituendo x_k. Altrimenti andiamo al Passo 3.

Passo 3 Si selezioni $\overline{y} \in N(x_k)$ tale che $f(\overline{y}) < f(x_k)$ e si ponga $x_{k+1} = \overline{y}$.

Passo 4 Si ponga $k = k + 1$ e si ritorni al Passo 2.

Per esempio nel problema di Figura 5.6 posso partire con $x_0 = (5, 5)$. Da qui posso spostarmi in $x_1 = (4, 5)$ e da qui in $x_2 = (4, 4)$. Essendo x_2 un ottimo locale mi arresto e resituisco il punto x_2. Se la vicinanza é esatta, l'algoritmo di ricerca locale mi restituisce l'ottimo globale. Se non é esatta puó succedere che mi venga restituito un ottimo locale ma non globale. Nel nostro esempio é esattamente ció che accade: ci viene restituito il punto $x_2 = (4, 4)$ che é un ottimo locale ma non globale. Da quanto detto sembrerebbe auspicabile utilizzare sempre vicinanze esatte. Ad esempio, nel problema TSP sembrerebbe auspicabile utilizzare la vicinanza esatta N_n piuttosto che quella non esatta N_2. Ma non dobbiamo dimenticare che a noi non interessa soltanto il risultato di un algoritmo ma anche il tempo necessario per la sua esecuzione. Nel caso degli algoritmi di ricerca locale il tempo di esecuzione é strettamente legato ai tempi necessari per esplorare i vicini del punto corrente ai Passi 2 e 3 dell'algoritmo. Nei problemi piú difficili le vicinanze esatte hanno anche dimensioni molto elevate e la loro esplorazione richiede tempi molto elevati. Ad esempio, la vicinanza N_n del problema TSP é tale che ogni circuito abbia come vicini tutti i circuiti hamiltoniani e quindi l'esplorazione dei vicini di un circuito si riduce all'enumerazione completa di tutti i circuiti che, come giá visto, é eseguibile in tempi ragionevoli solo per istanze molto piccole. Molto minore é il numero di vicini se si usa la vicinanza N_2 (tale numero é dell'ordine di n^2; più in generale tale numero è $O(n^k)$ per la vicinanza N_k). D'altra parte usando la vicinanza N_2 non si ha alcuna garanzia che la ricerca locale restituisca un ottimo globale. La scelta di una vicinanza dovrá basarsi su un compromesso tra tempi di esecuzione e qualitá della soluzione restituita dalla ricerca locale. Studi sperimentali indicano che la vicinanza N_3 fornisce risultati molto migliori rispetto alla N_2, seppure in tempi piú elevati, mentre i risultati ottenuti con la vicinanza N_4 non sono di molto migliori rispetto a quelli ottenibili con la N_3 nonostante i piú elevati tempi di esecuzione.

5.3.3 Simulated Annealing

Gli algoritmi Simulated Annealing (SA nel seguito) sono basati su un'analogia con un fenomeno fisico: mentre a temperature elevate le molecole in un liquido tendono a muoversi liberamente, se la temperatura viene decrementata in modo sufficientemente lento, la mobilitá termica delle molecole viene persa e tendono a formare un cristallo puro che corrisponde anche ad uno stato di minima energia. Se la temperatura viene decrementata troppo velocemente si parla di *quenching*

e lo stato finale é uno stato policristallino od amorfo con un'energia piú elevata di quella del cristallo puro. Stabilendo un'analogia tra configurazioni di un sistema fisico ed elementi della regione ammissibile da un lato e tra energia del sistema e funzione obiettivo dall'altro, ovvero trattando il problema di ottimizzazione come un particolare sistema fisico, si é arrivati a formulare gli algoritmi SA per problemi di ottimizzazione combinatoria attraverso la simulazione del processo fisico. Come nel processo fisico una lenta decrescita della temperatura conduce allo stato di minima energia, allo stesso modo nel problema di ottimizzazione combinatoria vogliamo arrivare in un punto (l'ottimo globale) con valore minimo della funzione obiettivo. La struttura degli algoritmi SA é la seguente.

Passo 1 Sia $x_0 \in S$ e si fissi $k = 0$.

Passo 2 Si generi in modo casuale un punto $y_{k+1} \in N(x_k)$.

Passo 3 Si ponga $x_{k+1} = y_{k+1}$ con probabilitá pari a

$$\min\left\{\exp\left\{\frac{f(x_k) - f(y_{k+1})}{T_k}\right\}, 1\right\}, \tag{5.7}$$

dove T_k é un parametro non negativo che, per analogia con il fenomeno fisico, viene chiamato temperatura. Altrimenti si ponga $x_{k+1} = x_k$. Nel primo caso si dice che il punto y_{k+1} viene *accettato* come nuovo punto, nel secondo che viene *rigettato*.

Passo 4 (Cooling Schedule) Si aggiorni il valore T_{k+1} della temperatura.

Passo 5 Si verifichi un criterio di arresto (ad esempio ci si arresti se il numero k di iterazioni é superiore ad un numero prefissato MAX_ITER di iterazioni). Se il criterio é soddisfatto ci si arresti e si restituisca il miglior elemento di S osservato durante l'esecuzione dell'algoritmo. Se invece non é soddisfatto si ponga $k = k + 1$ e si ritorni al Passo 2.

La probabilitá (5.7) viene detta funzione di accettazione di Metropolis. In essa si puó notare che ci si sposta sempre in un punto y_{k+1} se questo ha un valore di f inferiore rispetto al punto corrente x_k, esattamente come avviene in una ricerca locale. Ma la novitá rispetto alla ricerca locale é che ci si puó spostare anche in punti peggiori con una probabilitá che é tanto piú bassa quanto peggiore (piú elevato) é il valore di f nel punto y_{k+1} rispetto al valore di f nel punto corrente x_k. Si noti inoltre che tale probabilitá é controllata dal parametro temperatura: per valori elevati della temperatura é molto probabile accettare anche punti di molto peggiori, ma man mano che si decresce la temperatura la probabilitá di accettare un punto peggiore diventa sempre minore. La ragione per cui si accettano anche punti con valori della funzione obiettivo peggiori é che questo é il solo modo di sfuggire ad ottimi locali non globali. Nella ricerca locale applicata all'esempio in Figura 5.6 abbiamo visto come una volta giunti nell'ottimo locale x_2 non siamo piú in grado di procedere e siamo costretti ad arrestarci (siamo intrappolati nell'ottimo locale). In un algoritmo SA possiamo

invece sfuggire da un ottimo locale accettando anche punti peggiori rispetto a quello corrente. Per questa ragione tali algoritmi vengono detti *hill-climbing*, scavalcano le "colline" che separano tra loro gli ottimi locali.
Una componente essenziale degli algoritmi SA é il Passo 4 detto di *cooling schedule*, cioé il passo in cui si aggiorna la temperatura. Tipicamente si inizia con una temperatura elevata e di seguito la temperatura viene diminuita nel corso delle iterazioni dell'algoritmo. Se potessimo eseguire l'algoritmo per un tempo infinito allora si puó dimostrare che con una probabiltá pari a 1 esso sarebbe in grado di individuare l'ottimo globale, *a patto di far decrescere la temperatura in modo sufficientemente lento*, proprio come avviene nel fenomeno fisico. In particolare si deve avere ad ogni iterazione k che

$$T_k \geq \frac{M}{\log(k)},$$

dove M é una costante dipendente dal problema. Una decrescita piú rapida (il quenching del fenomeno fisico) puó fare in modo che si rimanga intrappolati in un ottimo locale ma non globale. In realtá sappiamo bene che non si ha mai a disposizione un tempo infinito e neppure molto elevato. Quindi ad una scelta che garantirebbe l'individuazione dell'ottimo globale ma in tempi troppo elevati, si preferisce spesso una scelta di temperature che decrescono piú rapidamente. Tale scelta non esclude di rimanere intrappolati in ottimi locali, ma consente di giungere in tempi piú brevi ad una buona soluzione. Non esploreremo oltre la questione delle temperature, precisando peró che scelte come quale *temperatura iniziale* utilizzare, *quando* ridurre la temperatura e *di quanto* ridurla, sono molto delicate per il buon funzionamento dell'algoritmo e sono tipicamente legate al problema da risolvere.

5.3.4 Algoritmi genetici

Diamo ora una breve descrizione di un altro approccio euristico, gli algoritmi genetici. Anche questi algoritmi nascono da un'analogia, l'analogia con il concetto darwiniano di *selezione naturale*. Ad una data iterazione k tali algoritmi lavorano su una "popolazione" P_k di elementi della regione ammissibile S. Da tale popolazione vengono "generati" nuovi elementi di S attraverso i processi di *mutazione*, che genera nuovi elementi di S modificando singoli elementi della popolazione P_k, e di *ricombinazione*, che forma nuovi elementi di S mettendo assieme "pezzi" di coppie di elementi di P_k (piú in generale la ricombinazione puó riguardare non solo coppie di elementi di P_k ma anche piú di due elementi della popolazione; qui ci restringiamo al caso, molto frequente, di ricombinazione di due elementi). Indicando con O_k i nuovi elementi di S generati tramite mutazione e ricombinazione, abbiamo una popolazione allargata $P_k \cup O_k$. A questo punto interviene il processo di *selezione*: all'iterazione successiva arriverá un sottinsieme P_{k+1} di $P_k \cup O_k$, ovvero solo alcuni degli elementi in $P_k \cup O_k$ "sopravviveranno" e faranno parte della popolazione P_{k+1} all'iterazione $k+1$. Nella scelta delle coppie da ricombinare ed anche nella selezione si tendono a favorire elementi della popolazione con un elevato valore di *fitness* (adattamento). Come

ovvia (ma non necessariamente unica) misura di fitness si puó pensare al valore della funzione obiettivo f: tanto piú piccolo é il valore di f per un elemento in un problema di minimo (tanto piú grande in un problema di massimo), qunato maggiore é il valore di fitness di tale elemento. Siamo ora pronti a fornire lo schema di un generico algoritmo genetico.

Passo 1 Si generi una popolazione $P_0 \subset S$ e si ponga $k = 0$.

Passo 2 (mutazione) Si scelgano alcuni elementi in P_k e si generino tramite mutazione nuovi elementi di S. Si inseriscano tali nuovi elementi in un insieme O_k.

Passo 3 (ricombinazione) Si scelgano coppie di elementi in P_k (tipicamente la scelta avviene in modo casuale ma favorendo elementi con un miglior valore di fitness) e si ricombinino tali coppie generando nuovi elementi di S. Si aggiungano tali nuovi elementi in O_k.

Passo 4 (selezione) Si determini P_{k+1} selezionando alcuni degli elementi in $P_k \cup O_k$ (tipicamente in P_{k+1} vengono conservati, tra gli altri, anche un certo numero di elementi con il miglior valore di fitness tra tutti quelli in $P_k \cup O_k$).

Passo 5 Si verifichi un criterio di arresto. Se é soddisfatto ci si arresti restituendo la miglior soluzione trovata durante l'esecuzione dell'algoritmo. Altrimenti si ponga $k = k + 1$ e si ritorni al Passo 2.

Per illustrare brevemente i processi di mutazione e ricombinazione, possiamo considerare il problema illustrato in Figura 5.6. Per tale problema il processo di mutazione puó essere considerato semplicemente come la generazione di un vicino dell'elemento su cui agisce la mutazione. Quindi una mutazione che agisce sull'elemento $(4, 5)$ puó generare, ad esempio, l'elemento $(4, 4)$. Piú significativa é la ricombinazione. Supponiamo di avere una coppia di elementi (i, j) e (h, k). Il processo di ricombinazione puó agire generando a partire da tale coppia di elementi, due nuovi elementi: uno con la prima componente del primo elemento e la seconda del secondo, quindi (i, k) e l'altro con la prima componente del secondo elemento e la seconda del primo, quindi (h, j). Per esempio, se la coppia di elementi é $(2, 5)$ e $(5, 2)$, i due nuovi elementi saranno $(2, 2)$ e $(5, 5)$. É importante notare che mentre i due elementi $(2, 5)$ e $(5, 2)$ sono lontani dall'ottimo globale, uno dei loro "figli", é l'ottimo globale stesso $(2, 2)$. Quindi, mentre nelle ricerche locali, negli algoritmi SA e nella mutazione stessa degli algoritmi genetici ci possiamo solo spostare in punti vicini, con la ricombinazione possiamo anche balzare in un solo colpo in una zona molto lontana da entrambi gli elementi della coppia. In certi casi il processo di ricombinazione fornisce dei grossi vantaggi. Questo succede in particolare quando il problema gode di una certa separabilitá. Non si vuole qui approfondire cosa si intenda per separabilitá ma ne possiamo dare una nozione intuitiva attraverso il problema di Figura 5.6. Se ne osserviamo la funzione obiettivo (5.6), si nota che essa é formata da due termini, $(i - 2)^2(i - 4)^2 + i$, dove compare la sola prima

componente degli elementi (i, j) di S, e $(j-2)^2(j-4)^2 + j$, dove compare la sola seconda componente. In questo caso quindi intendiamo per separabilitá il fatto che la funzione obiettivo puó essere scomposta in due parti dove le componenti che formano gli elementi di S compaiono separatamente. Se osserviamo i due elementi $(2, 5)$ e $(5, 2)$ si puó notare che il primo elemento ha la prima componente ottima rispetto al primo termine $(i-2)^2(i-4)^2+i$ ma non rispetto al secondo termine $(j-2)^2(j-4)^2 + j$, mentre per il secondo elemento vale esattamente il viceversa. A questo punto la ricombinazione, mettendo assieme il pezzo buono del primo elemento (la prima componente) ed il pezzo buono del secondo (la seconda componente) é in grado di generare un nuovo elemento migliore di entrambi (in questo caso addirittura l'ottimo globale).

Capitolo 6

Ottimizzazione continua

Per prima cosa ricordiamo che un'istanza di problema di ottimizzazione continua si presenta nella seguente forma

$$\min_{x \in S} f(x)$$

dove la regione ammissibile $S \subseteq R^n$ contiene un'infinitá non numerabile di punti. Va sottolineato che il fatto di restringere l'attenzione a soli problemi di minimo non è una restrizione reale in quanto l'uguaglianza

$$\max_{x \in S} f(x) = -\min_{x \in S} -f(x)$$

consente di ricondurre sempre un problema di massimo ad uno di minimo. I casi in cui $S = R^n$ vengono chiamati problemi di *ottimizzazione non vincolata*. In altri casi la regione ammissibile S è definita attraverso equazioni e/o disequazioni, ovvero

$$S = \{x \in R^n \ : \ g_i(x) \leq 0, \ i \in I_1, \ g_i(x) = 0, \ i \in I_2\}.$$

Questi problemi vengono definiti di *ottimizzazione vincolata*. Si osservi che quest'ultima classe di problemi comprende come sottocaso i problemi di PL dove la funzione f è una funzione lineare e le funzioni g_i sono funzioni affini. Una prima questione da porre è la seguente: esiste sempre una soluzione del problema? Nei problemi di ottimizzazione combinatoria, almeno quelli (la maggior parte) con regione ammissibile formata da un numero finito di punti, la risposta era sempre affermativa ma qui (come il caso di illimitatezza nella PL giá ci dimostra) la risposta puó essere negativa. Semplici esempi mostrano casi in cui il problema non ammette soluzioni.

- $f(x) = x^3$ Il minimo di questa funzione su $S = R$ non esiste in quanto la funzione puó essre fatta decrementare a piacere spostandosi verso $-\infty$.
- $f(x) = e^{-x}$ In questo caso la funzione non puó essere decrementata a piacere su $S = R$ (essa non scende mai sotto il valore 0) ma nessun punto in R ha un valore non superiore rispetto a tutti gli altri.

Nel caso di ottimizzazione vincolata ricordiamo che se f è continua e S è un insieme compatto (chiuso e limitato), allora il teorema di Weierstrass ci garantisce l'esistenza di una soluzione ottima.

Un'altra importante nozione è quella di *ottimo locale*. Il nostro obiettivo principale è quello di determinare un ottimo globale della funzione, ovvero un punto $x^* \in S$ tale che

$$f(x^*) \leq f(x) \quad \forall\ x \in S.$$

In alcuni casi peró questo puó essere estremamente difficile, mentre è piú semplice determinare un punto di ottimo locale che possiamo definire in modo informale come un punto che non è il migliore in assoluto ma lo è tra quelli in un suo intorno. Diamo ora una definizione formale di ottimo locale.

Definizione 16 *Un punto x' si definisce ottimo locale della funzione f se esiste un $\varepsilon > 0$ e un intorno di x'*

$$N(x', \varepsilon) = \{y \in R^n \ : \ \|y - x'\| \leq \varepsilon\},$$

tale che

$$\forall\ y \in N(x', \varepsilon) \cap S \ : \ f(x') \leq f(y). \tag{6.1}$$

In Figura 6.1 il punto x' è un ottimo locale mentre il punto x^* è sia un ottimo locale che un ottimo globale. In generale ogni ottimo globale (se esiste) è anche un ottimo locale mentre non è vero il viceversa. Si noti che in alcuni casi esistono ottimi locali ma non ottimi globali. Ad esempio la funzione $f(x) = x^3 - 3x$ ha un ottimo locale $x' = 1$ ma non ha ottimi globali. Ci sono peró anche casi piú semplici in cui tutti gli ottimi locali sono anche ottimi globali. È questo ad esempio il caso dei *problemi convessi* in cui f è una funzione convessa (si veda la definizione successiva) e S è un insieme convesso (si veda la Definizione 9 nel Capitolo 3).

Definizione 17 *Una funzione f si dice* convessa *su un insieme convesso S se $\forall\ x_1, x_2 \in S$ e $\forall\ \lambda \in (0,1)$:*

$$f(\lambda x_1 + (1-\lambda)x_2) \leq \lambda f(x_1) + (1-\lambda) f(x_2).$$

Si dice strettamente convessa *se $\forall\ x_1, x_2 \in S$ e $\forall\ \lambda \in (0,1)$:*

$$f(\lambda x_1 + (1-\lambda)x_2) < \lambda f(x_1) + (1-\lambda) f(x_2).$$

Vale il seguente teorema.

Teorema 5 *Se f è una funzione convessa e S è un insieme convesso, allora ogni ottimo locale x' di f su S (se ne esistono) è anche un ottimo globale. Se f è strettamente convessa puó esistere un solo ottimo globale.*

Dimostrazione Per dimostrare che ogni ottimo locale x' è anche un ottimo globale ragioniamo per assurdo e supponiamo che x' non sia un ottimo globale. Quindi esiste $y \in S$ tale che $f(y) < f(x')$. Si prenda ora il segmento

$$\lambda x' + (1-\lambda)y \quad \lambda \in [0,1].$$

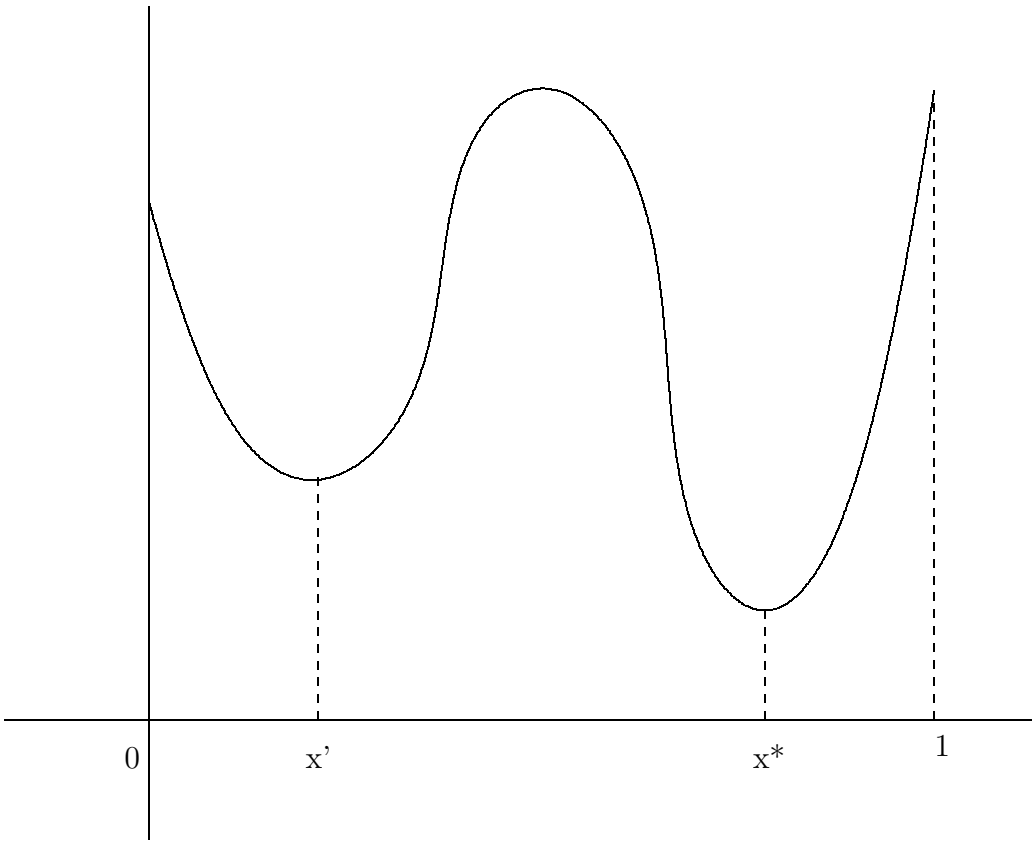


Figura 6.1: Un problema di ottimizzazione con un ottimo locale non globale e un ottimo globale e locale.

Per la convessitá di S il segmento è interamente contenuto in S. Inoltre la convessitá di f implica che $\forall\ \lambda \in (0,1)$:

$$f(\lambda x' + (1-\lambda)y) \leq \lambda f(x') + (1-\lambda)f(y) < \lambda f(x') + (1-\lambda)f(x') = f(x').$$

Quindi ogni punto sul segmento ha valore della funzione strettamente minore di $f(x')$. Dal momento che per ogni $\varepsilon > 0$ l'intorno $N(x', \varepsilon)$ di x' contiene almeno un punto del segmento diverso da x', questo contraddice il fatto che x' sia un ottimo locale (si veda (6.1)).
Nel caso in cui f sia strettamente convessa si supponga ancora per assurdo che esistano due distinti ottimi globali $x_1^*, x_2^* \in S$ con, ovviamente, $f(x_1^*) = f(x_2^*)$. Allora, in base alla definizione di funzione strettamente convessa si ha che $\forall\ \lambda \in (0,1)$:

$$f(\lambda x_1^* + (1-\lambda)x_2^*) < \lambda f(x_1^*) + (1-\lambda)f(x_2^*) = \lambda f(x_1^*) + (1-\lambda)f(x_1^*) = f(x_1^*).$$

Quindi tutti i punti del segmento diversi da x_1^* e x_2^* (tutti appartenenti a S per la convessitá di S) hanno valore della funzione minore di $f(x_1^*)$ e $f(x_2^*)$ il che contraddice il fatto che x_1^* e x_2^* siano ottimi globali.

Prendiamo ora in esame una particolare classe di funzioni, le funzioni quadratiche, ovvero con la seguente forma:

$$f(x) = bx + \frac{1}{2}xAx,$$

dove b è un vettore in R^n e A è una matrice quadrata simmetrica in $R^{n\times n}$. La sola analisi della matrice A ci consente di individuare quali tra le funzioni quadratiche sono convesse e strettamente convesse. Introduciamo la seguente definizione.

Definizione 18 *Una matrice simmetrica M si dice* semidefinita positiva *se*

$$\forall\ x \in R^n\ :\ xMx \geq 0.$$

Equivalentemente, M è semidefinita positiva se tutti i suoi autovalori sono non negativi. Una matrice simmetrica M si dice definita positiva *se*

$$\forall\ x \in R^n \setminus \{0\}\ :\ xMx > 0.$$

Equivalentemente, M è definita positiva se tutti i suoi autovalori sono strettamente positivi.

Introduciamo anche la seguente osservazione (senza dimostrazione) che ci verrá utile nel seguito.

Osservazione 10 *Se M è definita positiva è anche invertibile e la sua inversa M^{-1} è anch'essa definita positiva.*

Leghiamo ora tali definizioni al problema della determinazione di funzioni quadratiche convesse. Vale la seguente osservazione.

Osservazione 11 *Data la funzione quadratica*

$$f(x) = \frac{1}{2}xAx + bx,$$

essa è convessa se e solo se A è semidefinita positiva. È strettamente convessa se e solo se A è definita positiva.

6.1 Ottimizzazione non vincolata

Anche se esiste una vasta letteratura sull'ottimizzazione vincolata, qui restringeremo l'attenzione ai problemi di ottimizzazione non vincolata. Dapprima parleremo di condizioni di ottimalitá, ovvero condizioni necessarie o sufficienti a garantire che un punto sia un ottimo locale del problema. In seguito vedremo anche alcuni metodi di risoluzione. Si noti che d'ora in poi restringeremo l'attenzione ai soli casi in cui la funzione f é differenziabile in modo continuo almeno fino al secondo ordine, ovvero la funzione ammette derivate sia prime che seconde continue.

6.1.1 Condizioni di ottimalitá

Le condizioni di ottimalitá possono essere derivate dall'espansione in forma di Taylor della funzione in un punto. Sotto opportune condizioni di differenziabilitá della funzione f (f differenziabile in modo continuo almeno fino al secondo ordine) si ha che in un intorno di un punto x' la funzione $f(x)$ è pari a:

$$f(x) = f(x') + g(x')(x - x') + \frac{1}{2}(x - x')H(x')(x - x') + o(\|x - x'\|^2), \quad (6.2)$$

dove

- $g(x')$ è il *gradiente* della funzione f in x', ovvero un vettore in R^n la cui j-esima componente è:
$$\left[\frac{\partial f(x)}{\partial x_j}\right]_{x=x'}.$$
- $H(x')$ è l'hessiano di f in x', ovvero una matrice in $R^{n\times n}$ che in corrispondenza della riga i e della colonna j ha la derivata seconda:
$$\left[\frac{\partial^2 f(x)}{\partial x_i \partial x_j}\right]_{x=x'}.$$
- $o(\|x - x'\|^2)$ è una quantitá che al tendere di x a x' è piccola rispetto a $\|x - x'\|^2$ (più precisamente converge a 0 piú rapidamente di $\|x - x'\|^2$ al tendere di x a x').

Il primo risultato che dimostriamo è il seguente.

Teorema 6 *Condizione necessaria perchè un punto x' sia un ottimo locale di f è che*

$$g(x') = 0.$$

Dimostrazione Per assurdo sia $g(x') \neq 0$ e si considerino i punti

$$x = x' - \lambda g(x') \quad \lambda > 0. \quad (6.3)$$

Sostituendo in (6.2) x con i punti indicati in (6.3) si ottiene:

$$f(x) = f(x') - \lambda\|g(x')\|^2 + \frac{1}{2}\lambda^2[g(x')H(x')g(x')] + o(\lambda^2). \quad (6.4)$$

Se $g(x') \neq 0$, allora $\|g(x')\|^2 \neq 0$ e per $\lambda > 0$ e sufficientemente piccolo si ha che in (6.4) il termine dominante è $-\lambda\|g(x')\|^2$, cioè per ogni $\lambda > 0$ sufficientemente piccolo si ha:

$$f(x) \approx f(x') - \lambda\|g(x')\|^2 < f(x'),$$

il che contraddice il fatto che x' sia un ottimo locale.

Si noti che la condizione è necessaria ma non sufficiente, cioè esistono punti

che soddisfano la condizione ma non sono ottimi locali. Infatti, se si considera $f(x) = x^3$ si ha $g(x) = 3x^2$ e $g(0) = 0$ ma $x' = 0$ non è un ottimo locale. Punti x' per i quali $g(x') = 0$ vengono detti *punti stazionari*. Il teorema appena visto ci dice che se un punto è un ottimo locale, allora è anche un punto stazionario, mentre l'esempio successivo mostra che non è vero il viceversa, cioè un punto puó essere un punto stazionario senza essere un ottimo locale.
Introduciamo ora un'ulteriore condizione necessaria.

Teorema 7 *Condizione necessaria perchè un punto x' sia un ottimo locale di f è che*

- $g(x') = 0$.
- $H(x')$ *sia semidefinita positiva.*

Dimostrazione Abbiamo giá dimostrato che deve essere $g(x') = 0$. Per assurdo sia $H(x')$ non semidefinita positiva, cioè esiste $d \in R^n$ tale che

$$dH(x')d < 0.$$

Si considerino ora i punti

$$x = x' + \lambda d \quad \lambda > 0.$$

Sostituendo in (6.2) e ricordando che $g(x') = 0$ si ottiene:

$$f(x) = f(x') + \frac{1}{2}\lambda^2[dH(x')d] + o(\lambda^2).$$

Poiché $dH(x')d < 0$, per ogni $\lambda > 0$ sufficientemente piccolo si avrá

$$f(x) \approx f(x') + \frac{1}{2}\lambda^2[dH(x')d] < f(x'),$$

il che contraddice il fatto che x' sia un ottimo locale.

Si noti che anche questa condizione è necessaria ma non sufficiente, cioè esistono punti che soddisfano la condizione ma non sono ottimi locali. Infatti, se si considera di nuovo $f(x) = x^3$ si ha $g(x) = 3x^2,\ H(x) = 6x$ e $g(0) = 0,\ H(0) = 0$ ma $x' = 0$ non è un ottimo locale.
Per essere sicuri che un punto sia un ottimo locale dobbiamo richiedere qualcosa di piú forte della semidefinita positivitá di $H(x')$, come indica il seguente teorema (senza dimostrazione).

Teorema 8 *Condizione sufficiente perchè un punto x' sia un ottimo locale di f è che*

- $g(x') = 0$.
- $H(x')$ *sia definita positiva.*

In tal caso la condizione è sufficiente ma non necessaria, cioè esistono ottimi locali che non la soddisfano.Infatti, se si considera $f(x) = x^4$ si ha $g(x) = 4x^3$, $H(x) = 12x^2$ e $g(0) = 0$, $H(0) = 0$, quindi $H(0)$ é semidefinita positiva ma non definita positiva, pur essendo $x' = 0$ un ottimo locale. Se si considera invece la funzione

$$f(x, y) = x^2 + y^2$$

si ha che

$$g(x, y) = \begin{pmatrix} 2x \\ 2y \end{pmatrix} \quad H(x, y) = \begin{pmatrix} 2 & 0 \\ 0 & 2 \end{pmatrix}.$$

Si nota che per il punto $(x, y) = (0, 0)$ si ha che $g(0, 0) = (0, 0)$ mentre l'hessiano è indipendente da (x, y) ed è sempre definito positivo (essendo una matrice diagonale i suoi due autovalori coincidono con gli elementi lungo la diagonale e sono entrambi uguali a 2, quindi strettamente positivi). Quindi, il punto $(0, 0)$ è un ottimo locale.

Facciamo infine un breve accenno al caso convesso. Nel caso in cui la funzione f sia convessa le cose si semplificano. Si puó infatti dimostrare il seguente teorema.

Teorema 9 *Se f é una funzione convessa, condizione necessaria e sufficiente perché il punto x' sia un ottimo locale é che esso sia un punto stazionario, cioé $g(x') = 0$.*

Procedendo oltre, nel caso di funzioni quadratiche

$$f(x) = bx + \frac{1}{2}xAx,$$

si puó dimostrare che esse possono avere almeno un ottimo locale (e globale) solo nel caso in cui siano convesse (e quindi, equivalentemente, se A è semidefinita positiva, si veda l'Osservazione 11). Inoltre, hanno un unico ottimo locale (e globale) se e solo se sono strettamente convesse (e quindi, equivalentemente, se A è definita positiva, si veda ancora l'Osservazione 11). La seguente osservazione ci dice anche qual é l'unico ottimo locale (e globale) di una funzione quadratica strettamente convessa.

Osservazione 12 *Data una funzione quadratica strettamente convessa il suo unico ottimo locale (e globale) é il punto*

$$x' = -A^{-1}b.$$

Dimostrazione Per il Teorema 9 nel caso convesso gli ottimi locali coincidono con i punti stazionari. Il gradiente di f è il seguente:

$$g(x) = b + Ax$$

e quindi se vogliamo trovare l'ottimo locale basta azzerare il gradiente, cioé l'ottimo locale é la soluzione del seguente sistema:

$$Ax = -b,$$

ovvero il punto $x' = -A^{-1}b$, come si voleva dimostrare.

6.2 Metodi di ricerca locale

Parleremo ora di alcuni metodi di risoluzione per problemi di ottimizzazione continua non vincolata. Essi sono per lo piú *iterativi*. Ció vuol dire che generano una sequenza $\{x_k\}$ di punti. Ci aspetteremmo che tale sequenza converga ad punto x', cioé che

$$x_k \to x' \quad k \to +\infty,$$

e che x' sia un ottimo locale della funzione f. In realtá questo non si puó sempre garantire. Si puó invece tipicamente garantire che x' sia un punto stazionario, che, come abbiamo visto, non vuol dire necessariamente che x' sia un ottimo locale (a parte i casi di funzioni convesse dove le nozioni di punto stazionario e ottimo locale sono equivalenti). Tuttavia, il fatto che in questi metodi si abbia tipicamente che la corrispondente sequenza $\{f(x_k)\}$ dei valori della funzione nei punti della sequenza sia decrescente fa in modo che, a parte casi particolarmente sfortunati, il punto stazionario x' a cui si converge sia anche un ottimo locale.

Vi sono due rilevanti proprietá di convergenza di questi metodi dette rispettivamente di *convergenza globale* e di *convergenza locale*.

Convergenza globale Un metodo converge globalmente se per qualsiasi punto iniziale $x_0 \in R^n$ della sequenza, esso converge ad un qualche punto stazionario della funzione.

Convergenza locale La proprietá di convergenza locale riguarda la rapiditá con cui il metodo converge se il punto iniziale x_0 della sequenza si trova giá molto vicino ad un ottimo locale x'. Se indichiamo con ε la piccola distanza tra x_0 e x', si dirá che il metodo ha *convergenza quadratica* se la distanza tra x_1 e x' è dell'ordine di ε^2, quella tra x_2 e x' è dell'ordine di ε^4 e cosí via. Si parlerá invece di *convergenza lineare con coefficiente* $p < 1$ se la distanza di x_1 da x' è $p\varepsilon$, quella di x_2 da x' è $p^2\varepsilon$ e cosí via. Chiaramente, tra i due tipi di convergenza quella quadratica è migliore in quanto la distanza dei punti della sequenza da x' decresce molto piú rapidamente rispetto alla convergenza lineare. In particolare, la convergenza lineare è tanto piú lenta, quanto piú il coefficiente p è vicino a 1.

Descriveremo ora una delle due principali categorie di metodi di ricerca locale, i metodi basati su *ricerche lineari*, mentre ci limiteremo qui a citare l'altra categoria di metodi, quelli *trust region*.

6.2.1 Metodi basati su ricerche lineari

Questa categoria di metodi si basa su un modello $M_k(x)$ che approssima la funzione f in un intorno del punto corrente x_k. Il modello è tipicamente ricavato dall'espansione in forma di Taylor (6.2). Prima di descrivere il funzionamento di questi metodi diamo la definizione di *direzione di discesa* rispetto ad un punto x_k.

Definizione 19 *Una direzione d_k si definisce di discesa per f rispetto ad un punto x_k se*

$$g(x_k)d_k < 0.$$

Dall'espansione in forma di Taylor (6.2) con x_k al posto di x' e $x_k + \lambda d_k$ al posto di x, si ottiene:

$$f(x_k + \lambda d_k) = f(x_k) + \lambda g(x_k)d_k + \frac{1}{2}\lambda^2 d_k H(x_k)d_k + o(\lambda^2).$$

Il fatto che d_k sia una direzione di discesa rispetto a x_k garantisce che per λ sufficientemente piccolo si abbia:

$$f(x_k + \lambda d_k) \approx f(x_k) + \lambda g(x_k)d_k < f(x_k),$$

cioé è possibile ridurre il valore della funzione spostandosi lungo la direzione d_k, da qui il nome di direzione di discesa. Siamo ora pronti a descrivere i metodi basati su ricerche lineari.

Metodi basati su ricerche lineari

0. Inizializzazione Si consideri un punto $x_0 \in R^n$ e si ponga $k = 0$.

1. Individuazione direzione di discesa Si individui una direzione di discesa d_k rispetto al punto corrente x_k.

2. Ricerca lineare Si effettui una ricerca lineare lungo la direzione d_k, cioé si individui un valore $\lambda_k > 0$ tale che $f(x_k + \lambda_k d_k)$ sia "sufficientemente" piú piccolo di $f(x_k)$.

3. Aggiornamento Si ponga

$$x_{k+1} = x_k + \lambda_k d_k$$

e $k = k + 1$ e si ritorni al Passo 1.

Mentre il metodo descritto sopra genera una sequenza infinita di punti x_k, nelle applicazioni reali l'algoritmo possiede anche un criterio di arresto. Ad esempio, ci si arresta quando $\|g(x_k)\|$ è al di sotto di una piccola soglia δ (in tal caso si puó pensare di essere giá sufficientemente vicini ad un punto stazionario).
Riguardo il Passo 2, la ricerca lineare, se possibile il valore λ_k si prende come il minimo della seguente funzione monodimensionale

$$q(\lambda) = f(x_k + \lambda d_k) \quad \lambda \geq 0,$$

cioé:

$$q(\lambda_k) = min_{\lambda \geq 0}\ q(\lambda).$$

Se non è possibile individuare rapidamente tale minimo ci si accontenta di trovare un λ_k tale che $x_k + \lambda_k d_k$ garantisca una "sufficiente" decrescita rispetto a $f(x_k)$. Quindi, non basta che

$$f(x_k + \lambda_k d_k) < f(x_k),$$

ma si deve avere anche che $f(x_k+\lambda_k d_k)$ sia sufficientemente piú piccolo di $f(x_k)$. Cosa significhi "sufficientemente" inferiore è stabilito attraverso opportune formule. A puro titolo informativo diamo un esempio di tali formule. Un punto $x_k + \lambda_k d_k$, con $\lambda_k > 0$, garantisce una "sufficiente" decrescita ripsetto a x_k se soddisfa le seguenti due condizioni:

$$f(x_k + \lambda_k d_k) \leq f(x_k) + \lambda_k \rho g(x_k) d_k \tag{6.5}$$

$$f(x_k + \lambda_k d_k) \geq f(x_k) + \lambda_k (1-\rho) g(x_k) d_k, \tag{6.6}$$

dove ρ è una costante tale che $0 < \rho < \frac{1}{2}$. Si noti che la condizione (6.6) esclude valori di λ_k troppo vicini a 0. Infatti, dall'espansione in forma di Taylor da cui si tralasciano i termini del secondo ordine si ha, per valori di λ_k vicini a 0

$$f(x_k + \lambda_k d_k) \approx f(x_k) + \underbrace{\lambda_k g(x_k) d_k}_{<0} < f(x_k) + \underbrace{(1-\rho)}_{<1} \underbrace{\lambda_k g(x_k) d_k}_{<0}$$

e quindi valori piccoli di λ_k non soddisfano la condizione (6.6). Contemporanemante, la condizione (6.5) garantisce la "sufficiente" decrescita di $f(x_k + \lambda_k d_k)$ rispetto a $f(x_k)$. Infatti tale condizione impone che $f(x_k + \lambda_k d_k)$ sia inferiore rispetto a $f(x_k)$ almeno del valore $\mid \lambda_k \rho g(x_k) d_k \mid$. Esistono opportuni algoritmi di ricerca lineare che sono in grado di restituire valori λ_k che soddisfano contemporaneamente (6.5) e (6.6) ma non ci addentreremo nella descrizione di tali algoritmi. Ci concentreremo invece sull'altro importante passo dei metodi basati su ricerche lineari, il Passo 1 di individuazione di una direzione di discesa. Infatti, fino a questo punto non abbiamo specificato come si puó individuare una direzione di discesa. Vedremo ora due diversi metodi che si differenziano proprio per tale scelta.

6.2.2 Il metodo dell'antigradiente

Si consideri l'espansione in forma di Taylor (6.2) con x_k al posto di x' e $x_k + d$ al posto di x. Si ottiene:

$$f(x_k + d) = f(x_k) + g(x_k) d + \frac{1}{2} d H(x_k) d + o(\|d\|^2). \tag{6.7}$$

Nelle vicinanze di x_k e quindi per vettori d con una norma sufficientemente piccola, possiamo utilizzare il seguente modello lineare di f, ottenuto eliminando da (6.7) i termini di grado superiore al primo:

$$M_k(x_k + d) = f(x_k) + g(x_k) d. \tag{6.8}$$

Basandoci su tale modello, lungo quale direzione d possiamo far decrescere piú rapidamente la funzione? Il prodotto tra vettori ci dice che, a paritá di norma dei vettori d, quello per cui $g(x_k)d$ é piú piccolo é quello che forma un angolo di 180° con $g(x_k)$, ovvero quello con direzione $-g(x_k)$, detta anche *direzione dell'antigradiente* che dá il nome al metodo. Quindi, basandosi sul modello

lineare (6.8) della funzione f in un intorno di x_k, il metodo dell'antigradiente sceglie come direzione di ricerca ad ogni iterazione l'antigradiente $-g(x_k)$. Si noti che:

$$d_k = -g(x_k) \quad \Rightarrow \quad g(x_k)d_k = -\|g(x_k)\|^2 < 0,$$

e quindi l'antigradiente é una direzione di discesa rispetto a x_k.
Vediamo ora di commentare le proprietá di convergenza di questo metodo. Si puó dimostrare che il metodo converge globalmente. Tuttavia, le sue proprietá di convergenza locale possono essere molto scarse. Esso infatti ha convergenza locale di tipo lineare con un coefficiente p che puó essere arbitrariamente vicino a 1, il che vuol dire che localmente il metodo puó convergere in modo molto lento. Mostreremo questo fatto con un esempio.

Esempio 34 *Si consideri la funzione*

$$f(x, y) = Mx^2 + y^2,$$

con $M > 0$. Si puó facilmente dimostrare che il punto di ottimo locale e globale di questa funzione é l'origine $(0, 0)$. Prendiamo ora il punto

$$x_0 = \alpha(1/M, 1).$$

Il gradiente é $g(x, y) = (2Mx, 2y)$ e quindi nel punto x_0 la direzione di ricerca é l'antigradiente $-\alpha(2, 2)$ e il nuovo punto viene cercato lungo la semiretta:

$$\alpha(1/M, 1) - \lambda\alpha(2, 2) \quad \lambda \geq 0.$$

Possiamo facilmente trovare il minimo della funzione:

$$q(\lambda) = f(\alpha(1/M - 2\lambda), \alpha(1 - 2\lambda)) = M\alpha^2(1/M - 2\lambda)^2 + \alpha^2(1 - 2\lambda)^2 \quad \lambda \geq 0.$$

Il minimo é il valore $\lambda_0 = \frac{1}{M+1}$, indipendente da α e il nuovo punto x_1 é il seguente:

$$x_1 = \alpha\frac{M-1}{M+1}(1/M, 1).$$

Notando che

$$x_1 = \frac{M-1}{M+1}x_0 = \alpha\frac{M-1}{M+1}(1/M, 1),$$

si puó ripetere quanto visto sopra e ottenere:

$$x_2 = \frac{M-1}{M+1}x_1 = \alpha\left(\frac{M-1}{M+1}\right)^2(1/M, 1),$$

$$x_3 = \frac{M-1}{M+1}x_2 = \alpha\left(\frac{M-1}{M+1}\right)^3(1/M, 1),$$

e cosí via. Questo ci dice che ad ogni iterazione la distanza dall'ottimo (l'origine $(0, 0)$) viene ridotta del fattore

$$\frac{M-1}{M+1}$$

che al crescere di M tende a 1, confermando quindi che il metodo dell'antigradiente ha convergenza locale lineare con un coefficiente p che puó essere reso arbitrariamente vicino a 1.

6.2.3 Il metodo di Newton e i metodi quasi-newtoniani

Anche nel metodo di Newton si considera l'espansione in forma di Taylor (6.7) ma questa volta troncando i termini di ordine superiore al secondo. Quindi, il modello M_k della funzione nell'intorno di x_k é la seguente funzione quadratica:

$$M_k(x_k + d) = f(x_k) + g(x_k)d + \frac{1}{2}dH(x_k)d. \tag{6.9}$$

Nel metodo di Newton si sceglie come direzione di ricerca il vettore d_k (se esiste) che é l'unico minimo locale della funzione quadratica (6.9). Solo se M_k ha un unico minimo locale il metodo di Newton é ben definito ma non é assolutamente detto che ció accada per un qualsiasi punto x_k. In base a quanto visto sulle funzioni quadratiche si puó dimostrare che ció é possibile se e solo se M_k é una funzione strettamente convessa, il che, in base all'Osservazione 11 equivale a dire che l'hessiano $H(x_k)$ é una matrice definita positiva. L'Osservazione 12 ci dice che l'unico minimo locale coincide con l'unico punto stazionario della funzione, ovvero il punto:

$$d_k = -H(x_k)^{-1}g(x_k). \tag{6.10}$$

Tale punto coincide con la direzione di ricerca nel metodo di Newton. Si noti che quando il metodo é definito, cioé $H(x_k)$ é definita positiva, si ha

$$d_k = -H(x_k)^{-1}g(x_k) \quad \Rightarrow \quad -g(x_k)H(x_k)^{-1}g(x_k) < 0,$$

e quindi la direzione é di discesa (si ricordi che, per l'Osservazione 10, se $H(x_k)$ è definita positiva, lo è anche $H(x_k)^{-1}$). Se invece $H(x_k)$ non é definita positiva il metodo non é applicabile. Quindi il metodo di Newton non é applicabile in generale. Ció che interessa maggiormente di tale metodo sono le sue proprietá di convergenza locale. Si puó infatti dimostrare la seguente osservazione.

Osservazione 13 *Sia dato un ottimo locale x^* e un punto x_0 nelle vicinanze di x^*. Sotto opportune ipotesi, tra le quali ci limitiamo a citare che $H(x^*)$ deve essere definita positiva, si dimostra che il metodo di Newton a partire dal punto x_0 ha convergenza locale di tipo quadratico.*

Quindi, una volta vicini ad un ottimo locale il metodo di Newton ha una convergenza molto rapida. Queste ottime proprietá di convergenza locale rendono il metodo di Newton un modello a cui devono tendere gli altri metodi quando si avvicinano ad un ottimo locale. In pratica, i metodi che, non appena si trovano sufficientemente vicini ad un ottimo locale, convergono rapidamente ad esso, sono solo quelli che tendono a scegliere una direzione di ricerca che, al crescere del numero di iterazioni, é sempre piú vicina a quella individuata dal metodo di Newton.
Tra questi metodi con una rapida convergenza locale citiamo i metodi quasi-newtoniani che sono descritti di seguito

Metodi quasi-newtoniani

0. Inizializzazione Si consideri un punto $x_0 \in R^n$ e una matrice definita positiva G_0. Si ponga $k = 0$.

1. Individuazione direzione di discesa Si calcoli la direzione di discesa rispetto al punto corrente x_k come soluzione unica del sistema:

$$G_k d = -g(x_k),$$

ovvero la direzione:

$$d_k = -G_k^{-1} g(x_k). \tag{6.11}$$

2. Ricerca lineare Si effettui una ricerca lineare lungo la direzione d_k, cioé si individui un valore λ_k tale che $f(x_k + \lambda_k d_k)$ sia "sufficientemente" piú piccolo di $f(x_k)$.

3. Aggiornamento Si ponga

$$x_{k+1} = x_k + \lambda_k d_k$$

e si aggiorni la matrice G_k tramite una nuova matrice definita positiva G_{k+1}. Si ponga $k = k + 1$ e si ritorni al Passo 1.

Si noti che il fatto che le matrici G_k (e quindi anche le relative inverse G_k^{-1} per l'Osservazione 10) siano tutte definite positive garantisce che le direzioni d_k siano tutte direzioni di discesa. Si ha infatti:

$$d_k = -G_k^{-1} g(x_k) \quad \Rightarrow \quad -g(x_k) G_k^{-1} g(x_k) < 0.$$

Quindi, a differenza del metodo di Newton, i metodi quasi-newtoniani sono sempre definiti. Essenziale é l'aggiornamento di G_k in G_{k+1} che viene fatto ad ogni iterazione. Tale aggiornamento, di cui non entreremo nei dettagli, viene fatto in modo che ad ogni iterazione k la matrice G_k sia un'approssimazione della matrice $H(x_k)$ sempre piú buona man mano che ci si avvicina ad un ottimo locale. In questo modo avvicinandosi ad un ottimo locale la scelta della direzione di ricerca in (6.11) diventa sempre piú vicina alla scelta (6.10) effettuata dal metodo di Newton, consentendo a questi metodi di avere proprietá di convergenza locale molto buone.