Nota Queste dispense sono riprese integralmente da quelle dei corsi di Ricerca Operativa tenuti presso la sede di Roma, in particolare da quelle di Francisco Facchinei e Carlo Mannino per i Cap. 1,2,4,5,6,7,8, e da quelle di Stefano Lucidi e Massimo Roma per il Cap. 3.

Capitolo 4

Problemi di Programmazione Lineare Intera

La Programmazione Lineare Intera (PLI) tratta il problema della massimizzazione (minimizzazione) di una funzione di più variabili, soggetta a vincoli di uguaglianza e disuguaglianza ed alla restrizione che una o più variabili possano assumere soltanto valori interi.

Grazie alla generalità del modello, un grandissimo numero di problemi reali possono essere rappresentati da modelli di Programmazione Lineare Intera. In generale, i modelli di Programmazione Intera sono adatti alle applicazioni caratterizzate dall'indivisibilità delle risorse e dalla necessità di scegliere tra un numero finito di alternative. Queste applicazioni includono *problemi operativi* quali la distribuzione di beni ed il sequenziamento delle attività produttive; *problemi di pianificazione* quali la gestione ottima del portafoglio titoli e la localizzazione degli impianti ed infine *problemi di progettazione* quali il progetto di circuiti VLSI ed il progetto di sistemi automatici di produzione (robotica). Recenti applicazioni dell'ottimizzazione combinatoria ad altri settori riguardano problemi in biologia molecolare, fisica delle alte energie e cristallografia a raggi X.

Le questoni teoriche poste da tali problemi e le tecniche usate per la loro soluzione hanno caratteristiche molto diverse da quelle relative ai problemi di ottimizzazione continua studiati nei precedenti capitoli. Lo scopo di questo capitolo è quindi quello di dare alcune indicazioni sulle peculiarietà dei problemi di Programmazione Lineare Intera (PLI) e studiare una classe di algoritmi per la loro risoluzione.

4.1 Formulazioni Classiche di Problemi Lineari Interi

In questo paragrafo vengono presentati esempi classici di problemi che possono essere formulati come problemi di PLI. Lo scopo di questi esempi è quello di mostrare come le variabili intere e le disequazioni che le collegano possano essere usate come *linguaggio formale* per esprimere una serie di relazioni tra eventi.

4.1.1 Knapsack binario

Il primo uso delle variabili intere (binarie) che esamineremo è anche il più naturale. Si supponga di dover modellare il fatto che un dato evento possa verificarsi oppure no. La natura binaria del problema suggerisce immediatamente l'idea di modellare questa dicotomia per mezzo di una variabile x che può assumere solo valori $0, 1$. In particolare, si porrà $x = 1$ se l'evento si verifica e $x = 0$ altrimenti.

Supponiamo di avere n oggetti. Il j-esimo oggetto, $j = 1, \ldots, n$, ha un valore pari a c_j e un peso pari a p_j. Supponiamo di avere una bisaccia ("knapsack" in inglese) in cui vogliamo mettere alcuni degli oggetti. La bisaccia può portare al massimo un peso b. Il problema di scegliere un sottoinsieme degli

oggetti allo scopo di massimizzare la somma dei valori senza superare il limite imposto dal peso è il cosiddetto problema di *knapsack binario*:

$$\begin{array}{ll} \max & \sum_{j=1}^{n} c_j x_j \\ & \sum_{j=1}^{n} p_j x_j \leq b \\ & x_j \in \{0,1\} \quad j = 1, \ldots, n. \end{array}$$

In questo caso l'evento è costituito dalla scelta (o meno) del singolo oggetto. In generale, questi problemi di scelta tra progetti possono avere più vincoli (si pensi al caso in cui la bisaccia abbia un limite dovuto al peso e uno dovuto al volume degli oggetti); in tal caso il problema viene detto di *knapsack multidimensionale*.

4.1.2 Assegnamento

Un altro problema classico di pianificazione riguarda l'assegnamento di lavori a persone. Supponiamo che n persone debbano svolgere n lavori. Ciascun lavoro deve essere svolto esattamente da una persona; inoltre, ciascuna persona può svolgere al più un lavoro. Il costo della persona j assegnata al lavoro i è c_{ij}. Il problema è quello di assegnare i lavori alle persone minimizzando il costo totale di realizzazione di tutti i lavori. Per formulare questo problema, che è noto come problema di *assegnamento*, introduciamo le variabili binarie x_{ij}, $i = 1, \ldots, n$, $j = 1, \ldots, n$ corrispondenti all'evento ij definite come segue

$$x_{ij} = \begin{cases} 1 & \text{se la persona } j \text{ è assegnata al lavoro } i \\ 0 & \text{altrimenti} \end{cases}$$

Poiché esattamente una persona deve essere assegnata al lavoro i, avremo i vincoli:

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

Inoltre, poiché ciascuna persona non può svolgere più di un lavoro, avremo i vincoli:

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

È facile verificare che un vettore $x \in \{0,1\}^{m \times n}$ che soddisfa tutti i vincoli appena descritti individua un assegnamento ammissibile di persone ai lavori. La funzione obiettivo, ovviamente da minimizzare, può essere scritta come $\sum_{i=1}^{m} \sum_{j=1}^{n} c_{ij} x_{ij}$.

4.1.3 Problema del costo fisso.

Nei modelli di Programmazione Lineare le variabili di decisione rappresentano usualmente i livelli ai quali le varie attività vengono svolte e la funzione obiettivo da minimizzare è una funzione lineare di tali variabili. In molti problemi pratici, tuttavia, tale ipotesi non è giustificata in quanto il costo di una attività, in funzione del livello cui essa viene svolta, può avere un andamento come quello riportato in Figura 4.1.

In particolare, il costo dell'attività j è zero se $x_j = 0$ (cioè se l'attività non è avviata) ed è invece uguale a $f_j + c_j x_j$ se $x_j > 0$ con f_j positivo. La funzione relativa all'attività j è quindi

$$z_j(x) = \begin{cases} 0 & \text{se } x_j = 0 \\ f_j + c_j x_j & \text{se } x_j > 0 \end{cases}$$

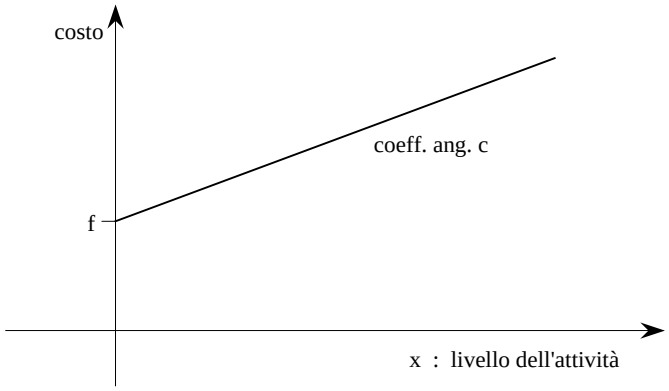


Figura 4.1: Costo dell'attività j.

Il valore f_j è detto *costo fisso* dell'attività j e deve essere pagato solamente se l'attività j viene svolta ad un livello non nullo. Per esempio, supponiamo che la variabile x_j rappresenti la quantità di petrolio greggio che deve essere trasportata da un pozzo A ad una raffineria B. In questo caso, se il pozzo A non rifornisce la raffineria B il costo complessivo di trasporto è ovviamente nullo. Se, al contrario, si decide di inviare una quantità non nulla x_j di greggio da A a B, al costo di trasporto $c_j x_j$ (proporzionale alla quantità trasferita) dovrà essere sommato il costo fisso di costruzione dell'oleodotto f_j.

Se indichiamo che J_f le attività che prevedono un costo fisso, il problema che dobbiamo risolvere può essere scritto

$$\begin{aligned} \min \quad & z(x) = \sum_{j \notin J_f} c_j x_j + \sum_{j \in J_f} z_j(x) \\ & Ax = b \\ & x \geq 0_n \end{aligned}$$

Chiaramente la funzione $z(x)$ è discontinua nell'origine e quindi il problema non è di Programmazione Lineare. Una possibile formulazione alternativa del Problema di Costo Fisso come problema di PLI si ottiene introducendo, per ciascuna attività, una variabile y_j che valga 1 quando $x_j > 0$ e 0 quando $x_j = 0$, cioè

$$y_j = \begin{cases} 0 & \text{se } x_j = 0 \\ 1 & \text{altrimenti} \end{cases}$$

In questo modo la funzione obiettivo può essere scritta

$$z(x) = \sum_{j=1}^{n} c_j x_j + \sum_{j \in J_f} y_j f_j = c^T x + f^T y,$$

dove $y \in \{0,1\}^{|J_f|}$ è il vettore le cui componente sono y_j e $z(x)$ è una funzione lineare.

Si tratta ora di imporre un vincolo che consenta di modellare la condizioni logiche

$$\begin{aligned} x_j > 0 \Rightarrow y_j = 1 \\ x_j = 0 \Rightarrow y_j = 0 \end{aligned}$$

Se supponiamo di conoscere un limite superiore per la variabile x_j, cioè un valore α positivo maggiore del più grande valore che può assumere la x_j, il vincolo

$$x_j - \alpha y_j \leq 0$$

forza la variabile y_j ad assumere valore 1 se $x_j > 0$[1].

Per quanto riguarda la seconda condizione $x_j = 0 \Rightarrow y_j = 0$, osserviamo che se $x_j = 0$ la variabile y_j il processo di minimizzazione farà in modo che y_i sia nulla all'ottimo poichè $f_j \geq 0$.

Il problema di costo fisso può essere quindi formulato, assumendo che tutti i costi fissi f_j siano positivi, nel modo seguente:

$$
\begin{array}{lllll}
(FC2) & \min\ z(x) & = & c^T x + f^T y & \\
 & & & Ax = b & \\
 & & & x_j \leq \alpha y_j, & j \in J_f \\
 & & & x \geq 0_n & \\
 & & & y_j \in \{0,1\}, & j \in J_f
\end{array}
$$

dove α è un numero positivo maggiore del più grande valore che può essere assunto da ciascuna delle variabili x_j in una soluzione ottima. Se $x_j > 0$ la variabile y_j sarà forzata ad assumere il valore 1 ed il suo costo fisso si aggiungerà al valore della funzione obiettivo. È quindi evidente che una soluzione ottima del problema FC1 è anche ottima per FC2 e viceversa.

4.1.4 Capital Budgeting

Questo esempio illustra l'applicazione della PLI al problema della pianificazione degli investimenti ("Capital Budgeting" in inglese). Si tratta di uno degli esempi più significativi di applicazione della PLI alle problematiche della pianificazione finanziaria.

Un' azienda genera continuamente, nello svolgimento delle sue attività, proposte di investimento e spesa (*progetti* nel seguito). Alcuni di questi progetti richiedono l'impiego di risorse finanziarie per consentire lo sviluppo dei prodotti e della produzione; altre proposte possono invece riguardare il miglioramento delle strutture produttive dell'azienda. Usualmente la decisione di attivare o meno un progetto condiziona la possibilità di attivare altri progetti, sia per la presenza di limitazioni sulla disponibilità dei capitali (*Razionamento dei Capitali*), sia per la presenza di vincoli sulla disponibilità di personale, macchine etc.

Un' azienda che si appresti a operare delle scelte individua solitamente un orizzonte temporale T entro il quale intende limitare l'analisi. Per esempio si decide di considerare gli investimenti e le loro conseguenze limitando l'analisi ai prossimi tre anni, $T = 3$ anni. L'orizzonte temporale viene poi suddiviso in *periodi* $1, 2, \ldots t$. Per esempio, se $T = 3$ anni, e i periodi sono i trimestri abbiamo dodici periodi, cioè $t = 12$. Il progetto i può essere caratterizato dal vettore $a_i = (a_{i1}, a_{i2}, \ldots, a_{it})$ del *Flusso di Cassa*. Il valore a_{ij} rappresenta il flusso di cassa (positivo o negativo) generato dal progetto i nel periodo j. Assumiamo che un flusso di cassa positivo corrisponda a una spesa, mentre uno negativo a un guadagno. Quindi se il vettore di flusso di cassa relativo a un certo progetto su un orizzonte temporale diviso in 5 periodi è $(5, 2, -1, -3, -8)$, vorrà dire che l'attivazione del progetto richiede una spesa di 5 nel primo periodo, di 2 nel secondo, e dà quindi un guadagno di 1, 3, 8 nel terzo, quarto e quinto periodo rispettivamente. Notiamo per inciso che questa è una struttura tipica (anche se esistono certamente delle eccezioni) dei flussi di cassa. Nei primi periodi l'attivazione di un progetto richiede degli investimenti, e quindi delle spese, una volta superata questa fase, si inziano ad avere guadagni. L'esborso totale generato dal progetto è dato dalla somma dei flussi di cassa nei vari periodi. In questo caso, l'esborso totale è $5+2-1-3-8 = -5$. Ricordiamo che con la nostra convenzione sui segni un numero negativo rappresenta un guadagno[2] In

[1] Osserviamo che la condizione $x_j > 0 \Rightarrow y_j = 1$ è equivalente a $y_j = 0 \Rightarrow x_j = 0$. Inoltre, volendo modellare la condizione $y_j = 1 \Rightarrow x_j = 0$, si può imporre il vincolo $x_j - \alpha(y_j - 1) \leq 0$.

[2] E' evidente che un guadagno (o un costo) all'istante j ha un "valore" diverso da un guadagno (o un costo) di pari importo ottenuto all'istante $k > j$. Per esempio, chiunque, dovendo scegliere tra l'avere un milione oggi o tra 2 anni, preferirebbe avere un milione oggi. Infatti, un milione oggi può essere investito in modo sicuro, (per esempio in buoni del tesoro) in modo di avere, tra due anni, più di un milione. Nel valutare una proposta

una situazione reale, per ogni periodo $j = 1, 2, \ldots, t$ abbiamo un "budget" cioè un limite agli esborsi (flussi di cassa positivi) che possiamo fare. Tali limiti derivano, ad esempio, da scelte aziendali o dalla limitata capacità dell'azienda di accedere al credito. Indichiamo con b_j il limite di budget nel periodo j-esimo. Cerchiamo di formulare adesso il problema di scegliere un sottoinsieme di progetti da attivare con il vincolo che in ogni periodo il vincolo di budget sia rispettato e in modo tale da rendere massimo il guadagno totale. A questo fine possiamo introdurre una variabile binaria x_i per ognuno degli n progetti possibili. Le variabili sono definite nel seguente modo:

$$x_i = \begin{cases} 1 & \text{se il progetto } i \text{ è attivato} \\ 0 & \text{altrimenti} \end{cases}$$

Possiamo a questo punto formulare facilmente il problema come problema di PLI:

$$\begin{array}{ll} \max & \sum_{i=1}^{n} x_i \left(-\sum_{j=1}^{t} a_{ij} \right) \\ & \sum_{i=1}^{n} a_{ij} x_i \leq b_j, \quad j = 1, \ldots, t \\ & x_i \in \{0, 1\} \quad i = 1, \ldots, n. \end{array}$$

Per capire la funzione obiettivo, bisogna ricordare che con la nostra convenzione di segni i guadagni sono numeri negativi. Quindi il termine $\left(-\sum_{j=1}^{t} a_{ij} \right)$ rappresenta, per ogni i, il guadagno derivante dall'attivazione del progetto i. Il cambio di segno serve a far diventare i guadagni numeri positivi e le perdite numeri negativi. I vincoli sono t, uno per ciascun periodo in cui è diviso l'orizzonte temporale e impongono che in ogni periodo non si superi il budget disponibile.

La formulazione del problema come problema di PLI permette di tenere facilmente conto di *vincoli logici* tra le attività. Facciamo alcuni esempi.

1. Se vogliamo imporre che il progetto 1 venga sicuramente attivato, basta aggiungere alla formulazione precedente il vincolo $x_1 = 1$.

2. Se vogliamo dire che almeno uno tra i progetti 2, 5, 89 deve essere attivato, basta aggiungere alla formulazione precedente il vincolo
$$x_2 + x_5 + x_{89} \geq 1.$$

3. Se vogliamo dire che uno e uno solo tra i progetti 2, 5, 89 deve essere attivato, basta aggiungere alla formulazione precedente il vincolo
$$x_2 + x_5 + x_{89} = 1.$$

4. Se vogliamo dire che al più uno tra i progetti 2, 5, 89 deve essere attivato, basta aggiungere alla formulazione precedente il vincolo
$$x_2 + x_5 + x_{89} \leq 1.$$
In questo, e nel precedente caso, i progetti di dicono *alternativi*.

5. Se vogliamo modellare il fatto che il progetto 7 può essere attivato solo se è stato attivato il progetto 9, basta aggiungere il vincolo
$$x_7 \leq x_9.$$
Se il progetto 7 può essere attivato solo se sono stati attivati i progetti 9 e 10, si può aggiungere il vincolo
$$x_7 \leq \frac{1}{2}(x_9 + x_{10}).$$

di investimento è fondamentale riportare i flussi di cassa nei vari periodi a una base comune. Questo processo è detto *attualizzazione*. Senza approfondire l'argomento, noi supponiamo che i vari a_{ij} siano stati attualizzati e siano quindi tra loro pienamente confrontabili. La procedure di attualizzazione è una procedure standard, molto semplice che non riportiamo qui solo perchè non aggiunge nulla agli argomenti cui siamo interessati.

Questi esempi possono essere facilmente generalizzati, e rappresentano il prototipo di come le variabili intere (e in particolare le variabili binarie) possano essere usate per modellare relazioni logiche tra eventi (in questo caso l'evento consiste nell'attivazione o meno di un progetto).

Facciamo un esempio di quanto visto finora. Supponiamo che il nostro orizzonte temporale di un anno sia diviso in 6 bimestri (T = un anno, t = 6), e che all'inizio dell'anno si debba decidere quali tra 5 progetti attivare. Nella tabella seguente si riportano i dati di interesse. La riga relativa ad ogni progetto riporta, per ogni periodo, il flusso di cassa generato dal progetto, mentre nell'ultima riga è riportato, per ogni periodo, il budget disponibile (le cifre sono tutte in milioni di euro e sono state attualizzate).

	Periodo 1	Periodo 2	Periodo 3	Periodo 4	Periodo 5	Periodo 6
Progetto 1	2	1	0	-2	-3	-1
Progetto 2	3	3	1	-4	-3	-4
Progetto 3	1	3	-1	2	-3	-4
Progetto 4	0	4	0	-5	-1	0
Progetto 5	4	-1	-3	-2	-1	-1
budget	8	8	5	2	2	1

Indichiamo con a_i la somma dei flussi di cassa generati dal progetto nei sei periodi i: $a_i = \sum_{j=1}^{6} a_{ij}$. Abbiamo

$$a_1 = -3, \quad a_2 = -4, \quad a_3 = -2, \quad a_4 = -2, \quad a_5 = -4.$$

Notiamo che gli a_i sono tutti negativi, e questo corrisponde al fatto che i progetti considerati, al termine dell'orizzonte temporale, danno dei guadagni. A questo punto possiamo scrivere facilmente il problema di PLI che ci permette di determinare la scelta ottima. Il probelma ha cinque variabili, una per ogni progetto, e sei vincoli di budget, uno per ogni periodo.

$$\begin{array}{lll} \max & 3x_1 + 4x_2 + 2x_3 + 2x_4 + 4x_5 & \\ & 2x_1 + 3x_2 + x_3 + 4x_5 & \leq \quad 8 \\ & x_1 + 3x_2 + 3x_3 + 4x_4 - x_5 & \leq \quad 8 \\ & x_2 - x_3 - 3x_5 & \leq \quad 5 \\ & -2x_1 - 4x_2 + 2x_3 - 5x_4 - 2x_5 & \leq \quad 2 \\ & -3x_1 - 3x_2 - 3x_3 - x_4 - x_5 & \leq \quad 2 \\ & -x_1 - 4x_2 - 4x_3 - x_5 & \leq \quad 1 \\ & x_i \in \{0, 1\} \quad i = 1, \ldots, 5. & \end{array}$$

Supponiamo ora che il progetto 4 copra solo 4 periodi dei 6 considerati. Questo è congruente col fatto che nel primo e nel sesto periodo quel progetto genera un flusso di cassa nullo.[3] Più precisamente, possiamo supporre che il progetto 4 copra il secondo, terzo, quarto e quinto periodo. Ci possiamo allora porre il problema di "posizionare" temporalmente il progetto. In particolare ci possiamo domandare se è conveniente far partire il progetto 4 e, nel caso, se sia più conveniente farlo partire nel primo nel secondo

[3]Attenzione, però, non è detto che un flusso di cassa nullo debba necessariamente corrispondere al fatto che un progetto non copra tutti i periodi. Un flusso di cassa zero può benissimo indicare semplicemente il fatto che in quel periodo le spese e i guadagni si equilibrano (vedi per esempio il progetto 1 nel periodo 3, o lo stesso progetto 4 sempre nel terzo periodo). In altre parole, l'informazione sulla durata effettiva dei progetti non è interamente ricavabile da una tabella come quella riportata. Per esempio, sarebbe stato congruente con la tabella data sapere che il progetto 4 copre 5 periodi (dal secondo al seto), ma che nell'ultimo periodo non genera né guadagni né perdite. Al contrario, dalla tabella si può dire con certezza che il progetto 5 copre tutti e sei i periodi considerati, perché esso genera un flusso di cassa non nullo in ogni periodo.

o nel terzo periodo. Notiamo che nella versione attuale il progetto 4 viene fatto iniziare nel secondo periodo. Per analizzare questo caso possiamo introdurre, al posto del progetto 4, tre progetti, che non sono altro che copie del progetto 4 posizionate temporalmente in modo diverso. La tabella con i dati del problema diventa allora la seguente.

	Periodo 1	Periodo 2	Periodo 3	Periodo 4	Periodo 5	Periodo 6
Progetto 1	2	1	0	-2	-3	-1
Progetto 2	3	3	1	-4	-3	-4
Progetto 3	1	3	-1	2	-3	-4
Progetto 4a	4	0	-5	-1	0	0
Progetto 4b	0	4	0	-5	-1	0
Progetto 4c	0	0	4	0	-5	-1
Progetto 5	4	-1	-3	-2	-1	-1
budget	8	8	5	2	2	1

In corrispondenza il probelma di PLI diventa:

$$
\begin{aligned}
\max \quad & 3x_1 + 4x_2 + 2x_3 + 2(x_{4a} + x_{4b} + x_{4c}) + 4x_5 & & \\
& 2x_1 + 3x_2 + x_3 + 4x_{4a} + 4x_5 & \leq & \ 8 \\
& x_1 + 3x_2 + 3x_3 + 4x_{4b} - x_5 & \leq & \ 8 \\
& x_2 - x_3 - 5x_{4a} + 4x_{4c} - 3x_5 & \leq & \ 5 \\
& -2x_1 - 4x_2 + 2x_3 - x_{4a} - 5x_{4b} - 2x_5 & \leq & \ 2 \\
& -3x_1 - 3x_2 - 3x_3 - x_{4b} - 5x_{4c} - x_5 & \leq & \ 2 \\
& -x_1 - 4x_2 - 4x_3 - x_{4c} - x_5 & \leq & \ 1 \\
& x_{4a} + x_{4b} + x_{4c} & \leq & \ 1 \\
& x_i \in \{0,1\}^7. &&
\end{aligned}
$$

Notiamo che, oltre all'ovvio incremento di variabili, abbiamo aggiunto il vincolo

$$x_{4a} + x_{4b} + x_{4c} \leq 1,$$

che ci dice che al più una delle "copie" del progetto 4 può essere attivata, non avendo ovviamente senso dire che bisogna far partire lo stesso progetto in due periodi differenti.

4.1.5 Localizzazione

I modelli di localizzazione sono uno dei principali strumenti per la pianificazione territoriale di *reti di servizio*. L'obiettivo generico è quello di decidere dove localizzare dei *centri di servizio*, quali impianti di produzione, depositi per la distribuzione, sportelli bancari, ospedali, allo scopo di soddisfare una domanda distribuita sul territorio, minimizzando un'opportuna funzione di costo.

Si consideri l'esempio di un'amministrazione cittadina che debba decidere dove costruire un numero prefissato di centri di pronto soccorso per servire i quartieri della città. Per ogni possibile sito di localizzazione sono noti i tempi medi di percorrenza da ciascun quartiere. Per fissare le idee, supponiamo che il numero di centri da costruire sia p, il numero di quartieri sia m e il numero di possibili siti per la localizzazione degli impianti sia n. Inoltre, indicheremo con c_{ij} il tempo medio di percorrenza dal quartiere $i \in \{1, \ldots, m\}$ al sito $j = \{1, \ldots, n\}$. L'obiettivo degli amministratori e quello di non sfavorire (troppo) nessuno dei potenziali utenti. In altri termini, a) tutti i quartieri devono essere serviti da un pronto soccorso e b) si deve minimizzare il tempo di percorrenza necessario all'utente più sfavorito (e cioè quello che impiega più tempo di tutti) a raggiungere il pronto soccorso.

Per poter modellare il problema di localizzazione come problema di programmazione lineare intera dobbiamo innanzitutto decidere le associazioni fra variabili ed eventi da rappresentare. In primo luogo si deve stabilire quali siti vengono scelti per localizzare i centri: associamo quindi una variabile booleana con ogni possibile sito, e cioè introduciamo una variabile booleana $x_j \in \{0,1\}$ per $j = 1, \ldots, n$, con la convenzione che:

$$x_j = \begin{cases} 1 & \text{se il sito } j \text{ è scelto} \\ 0 & \text{altrimenti.} \end{cases}$$

La seconda informazione necessaria è a quale centro viene assegnato ciascun quartiere. Introduciamo quindi una nuova variabile booleana y_{ij} per ogni quartiere $i \in \{1, \ldots, m\}$ e per ogni possibile sito $j \in \{1, \ldots, n\}$, ove

$$y_{ij} = \begin{cases} 1 & \text{se il quartiere } i \text{ è servito da un centro localizzato nel sito } j \\ 0 & \text{altrimenti.} \end{cases}$$

Poichè verranno costruiti esattamente p centri di pronto soccorso, sarà:

$$x_1 + x_2 + \ldots + x_n = p \tag{4.1}$$

Inoltre, poichè ogni quartiere deve essere assegnato ad (esattamente) un sito, si deve avere

$$y_{i1} + y_{i2} + \ldots + y_{in} = 1 \qquad \text{per } i = 1, \ldots, m. \tag{4.2}$$

I vincoli (4.2) assicurano che ogni quartiere sia servito da un sito. Ovviamente, affinchè un quartiere possa essere servito da un determinato sito, occorre che in tale sito sia effettivamente localizzato un centro di pronto soccorso. Questo fatto può essere espresso dai seguenti vincoli lineari:

$$y_{ij} \leq x_j \qquad \text{per ogni } i = 1, \ldots, m, j = 1, \ldots, n. \tag{4.3}$$

Si osservi infatti che, qualora si abbia $x_j = 0$ per qualche $j \in \{1, \ldots, n\}$ - e cioè nel sito j-esimo non è localizzato un centro di pronto soccorso, il vincolo (4.3) comporta $y_{ij} = 0$ per $i = 1, \ldots, n$ e quindi nessun quartiere può essere assegnato a un sito non selezionato per la localizzazione di un centro.

I vincoli finora descritti assicurano che ogni quartiere sia servito da qualche sito in cui sia localizzato un centro di pronto soccorso. Per descrivere la funzione obiettivo, abbiamo bisogno di qualche considerazione (e di una variabile) aggiuntiva. In particolare, se $\bar{x}, \bar{y}$ è una soluzione che soddisfa i vincoli (4.1),(4.2) e (4.3), qual è il tempo di percorrenza $t(i, \bar{y})$ dall'i-esimo quartiere al sito assegnatogli? Si osservi che se $\bar{y}_{ij} = 1$, il quartiere i-esimo è servito dal centro localizzato nel sito j-esimo, e quindi il tempo di percorrenza sarà c_{ij}. Quindi, grazie al vincolo (4.2), il generico tempo di percorrenza $t(i, y)$ può essere espresso come

$$t(i, y) = c_{i1}y_{i1} + c_{i2}y_{i2} + \ldots + c_{in}y_{in} \qquad \text{per } i = 1, \ldots, m.$$

L'obiettivo è quello di trovare la soluzione (x, y) che minimizza il più grande $t(i, y)$ per $i = 1, \ldots, m$. Introduciamo quindi una nuova variabile $z \in I\!R_+$ che rappresenta (un limite superiore per) il massimo tempo di percorrenza associato alla soluzione (x, y). Poichè z deve essere (maggiore o) uguale al massimo tempo di percorrenza, sarà maggiore o uguale di ogni tempo di percorrenza $t(i, y)$ per $i = 1, \ldots, m$. Quindi introduciamo i seguenti m vincoli lineari:

$$z \geq c_{i1}y_{i1} + c_{i2}y_{i2} + \ldots + c_{in}y_{in} \qquad \text{per } i = 1, \ldots, m. \tag{4.4}$$

Per minimizzare il massimo tempo di percorrenza è sufficiente minimizzare il suo limite superiore z e la funzione obiettivo si scriverà semplicemente

$$\min z$$

Riepilogando, il modello di localizzazione può essere scritto come segue:

$$
\begin{aligned}
&\min z \\
&\textstyle\sum_{j=1}^{n} x_j = p \\
&\textstyle\sum_{j=1}^{n} y_{ij} = 1 && i = 1, \ldots, m \\
&z \geq \textstyle\sum_{j=1}^{n} c_{ij} y_{ij} && i = 1, \ldots, m \\
&x \in \{0,1\}^n, \qquad y \in \{0,1\}^{m \times n} && .
\end{aligned}
$$

Esempio. Siano dati 3 siti candidati e 4 quartieri e supponiamo si vogliano localizzare 2 centri di pronto soccorso. I tempi di trasporto da quartiere a sito sono espressi nella seguente tabella

	quart. 1	**quart. 2**	**quart. 3**	**quart. 4**
Sito 1	7	6	7	8
Sito 2	10	10	1	1
Sito 3	9	5	4	1

Scriviamo innazitutto il vincolo (4.1):

$$x_1 + x_2 + x_3 = 2$$

Scriviamo ora i vincoli (4.2)

$$
\begin{cases}
y_{11} + y_{12} + y_{13} = 1 \\
y_{21} + y_{22} + y_{23} = 1 \\
y_{31} + y_{32} + y_{33} = 1 \\
y_{41} + y_{42} + y_{43} = 1
\end{cases}
$$

Scriviamo ora i vincoli (4.3)

$$
\begin{cases}
y_{11} \leq x_1 \\
y_{12} \leq x_2 \\
y_{13} \leq x_3 \\
y_{21} \leq x_1 \\
y_{22} \leq x_2 \\
y_{23} \leq x_3 \\
y_{31} \leq x_1 \\
y_{32} \leq x_2 \\
y_{33} \leq x_3 \\
y_{41} \leq x_1 \\
y_{42} \leq x_2 \\
y_{43} \leq x_3
\end{cases}
$$

Infine, i vincoli (4.4)

$$
\begin{cases}
z \geq 7y_{11} + 10y_{12} + 9y_{13} \\
z \geq 6y_{21} + 10y_{22} + 5y_{23} \\
z \geq 7y_{31} + 1y_{32} + 4y_{33} \\
z \geq 8y_{41} + 1y_{42} + 1y_{43}
\end{cases}
$$

Si osservi che per un problema così piccolo è facile calcolare la soluzione ottima enumerando tutte le soluzioni. A tal scopo, notiamo innanzitutto che, una volta scelti i due siti ove costruire i centri, la soluzione ottima è ottenuta assegnando ciascun cliente al sito più vicino. Ad esempio, se scegliamo il sito 1 e il sito 2 ($x_1 = 1, x_2 = 1, x_3 = 0$, ci converrà assegnare il quartiere 1 e 2 al sito 1 , mentre i quartieri 3 e 4 al sito 2 ($y_{11} = y_{21} = y_{32} = y_{42} = 1$). Il quartiere più svantaggiato è il quartiere 1, con un tempo di percorrenza pari a 7. Se scegliamo il sito 1 e il sito 3, ci converrà assegnare il quartiere 1 al sito 3 ,

mentre i quartieri 2, 3 e 4 al sito 2. Anche in questo caso il quartiere più svantaggiato è il quartiere 1 con tempo medio di percorrenza pari a 9. Infine, se scegliamo il sito 2 e il sito 3, ci converrà assegnare i 1 e 2 al sito 3 , mentre i quartieri 3 e 4 al sito 2. Anche in questo caso il quartiere più svantaggiato è il quartiere 1 con tempo medio di percorrenza pari a 9. Quindi, la soluzione che minimizza il tempo di percorrenza del quartiere più svantaggiato è la prima, in corrispondenza alla scelta dei siti 1 e 2.

Per comprendere il significato dell'introduzione della variabile z, si consideri ancora l'esempio descritto. Si è visto che la soluzione ottima corrisponde alle seguenti assegnazioni ottime per il vettore (x,y): $x_1 = 1, x_2 = 1, x_3 = 0$, $y_{11} = y_{21} = y_{32} = y_{42} = 1$, $y_{ij} = 0$ altrimenti. Sostituendo nei vincoli (4.4), otteniamo:

$$\begin{cases} z \geq 7 \\ z \geq 6 \\ z \geq 1 \\ z \geq 1 \end{cases}$$

Quindi si deve avere $z \geq 7$ che corrisponde al tempo di percorrenza del quartiere più svantaggiato. Naturalmente, siccome si sta cercando il minimo valore di z, e non ci sono ulteriori vincoli sulla variabile z, all'ottimo avremo $z = 7$.

Questa particolare classe di problemi di programmazione lineare in cui si vuole minimizzare il massimo di una famiglia di funzioni lineari, viene detta *problema di min-max*: come visto, problemi di questo tipo possono essere risolti mediante l'introduzione di una variabile artificiale. Generalizzando leggermente, quello che abbiamo appena mostrato è che, se abbiamo un problema di min-max del tipo

$$\begin{array}{lrcl} \min & (\max_i c_i^T x) & & \\ & Ax & \leq & b \\ & (x & & \text{intero}), \end{array}$$

lo possiamo riscrivere come problema lineare introducendo una nuova variabile z:

$$\begin{array}{lrcll} \min & z & & & \\ & c_i^T x & \leq & z, & \forall i \\ & Ax & \leq & b & \\ & (x & & \text{intero}). & \end{array}$$

Se (z^*, x^*) è una soluzione di questo problema, allora x^* è una soluzione del problema di min-max.

Un'ultima osservazione riguarda una naturale e più realistica estensione del problema di localizzazione. Infatti, molto spesso si devono aggiungere nuovi centri a un insieme di centri già attivi e localizzati sul territorio. Supponiamo ad esempio che k centri di pronto soccorso siano già localizzati nella città e che se ne vogliano attivare altri q. Allora, è possibile ancora una volta risolvere un problema di localizzazione in cui si vogliano attivare $p = k+q$ centri; nel modello, tuttavia, si porrà $x_j = 1$ per ogni sito j corrispondente a un centro già attivato.

4.1.6 Scheduling (Sequenziamento)

Concludiamo questa sezione sulle formulazioni con un esempio di utilizzazione delle variabili binarie per modellare *vincoli disgiuntivi*. Nell'usuale definizione di un problema di ottimizzazione si assume che tutti i vincoli debbano essere soddisfatti da una soluzione ammissibile. Tuttavia, in molte applicazioni capita che solo un sottoinsieme dei vincoli debba essere soddisfatto e che tale sottoinsieme sia specificato dal valore assunto da una opportuna variabile di decisione. In questo caso si dice che i vincoli sono *disgiuntivi*. Un esempio di applicazione di tali vincoli disgiuntivi è fornito dal problema di *Scheduling*.

Vincoli di tipo disgiuntivo sorgono abbastanza naturalmente in problemi di sequenziamento. In tali problemi si ha l'obiettivo di decidere l'ordine di processamento di un certo numero di lavori su una macchina a capacità unitaria. I vincoli disgiuntivi appaiono in quanto due lavori i e j non possono essere processati contemporaneamente sulla macchina e quindi *uno solo* dei vincoli (i precede j) o (j precede i) deve essere soddisfatto.

Supponiamo che debbano essere sequenziati n lavori su una macchina. Sia p_i il tempo di processamento del lavoro i sulla macchina. Poichè la macchina ha capacità unitaria, essa dovrà completare un lavoro prima di iniziare il lavoro successivo. Sia t_i l'istante in cui la macchina effettivamente inizia la lavorazione del lavoro i. Formulare il problema di scheduling consiste nel determinare vincoli sulle variabili t_i in modo tale che esse rappresentino sequenze effettivamente realizzabili sulla macchina.

Se il lavoro i è iniziato sulla macchina prima del lavoro j, dobbiamo avere $t_j \geq t_i + p_i$. D'altra parte, se il lavoro j inizia prima del lavoro i, dobbiamo avere $t_i \geq t_j + p_j$. Sia α un numero positivo molto grande e sia $y_{ij} = 1$ se i precede j e $y_{ij} = 0$ se j precede i. Considera il seguente sistema di vincoli:

$$\begin{aligned} \alpha y_{ij} + t_i - t_j &\geq p_j, \qquad 1 \leq i < j \leq n \\ \alpha(1 - y_{ij}) + t_j - t_i &\geq p_i, \qquad 1 \leq i < j \leq n \end{aligned}$$

Osserviamo che se $y_{ij} = 1$ (cioè se i precede j) allora il primo vincolo è sempre soddisfatto poichè $\alpha >> p_j + t_j - t_i$, mentre il secondo vincolo esprime la condizione che la lavorazione di j può iniziare solo dopo il completamento di i. Una situazione analoga si avrà quando $y_{ij} = 0$.

Se un vettore (t, y) con $t \in I\!R^n$ ed $y \in \{0,1\}^{n \times n}$ soddisfa questo sistema allora, per quanto detto, ciascuna componente del vettore t rappresenta un istante ammissibile di inizio processamento per il corrispondente lavoro. Viceversa, per ogni vettore ammissibile t esiste sicuramente un vettore y (che rappresenta l'ordine di processamento dei lavori sulla macchina) tale che il vettore (t, y) è ammissibile per il precedente sistema di vincoli. Vincoli di precedenza o altre restrizioni temporali possono essere facilmente inseriti nel modello aggiungendo vincoli lineari sulle variabili t ed y.

4.2 Tecniche di soluzione per problemi di PLI

In forma del tutto generale un problema di PLI può essere scritto come

$$\begin{array}{ll} \max & c^T x \\ & Ax \leq b \\ & 0 \leq x \leq U \\ & x \quad \text{intera} \end{array} \qquad (PLI)$$

dove U è un vettore a n componenti che limita superiormente il valori che possono essere assunti dalle variabili. La presenza di questo vincolo implica che *la regione ammissibile del problema PLI è limitata.* Questa assunzione semplificherà le considerazioni svolte in questo paragrafo. Essa è, peraltro, non limitativa, in quanto in ogni problema reale il valore assunto dalle componenti delle soluzioni ammissibili è sempre limitato superiormente da qualche costante.

In questo paragrafo consideriamo alcune tecniche risolutive per problemi di PLI. In particolare, dopo alcune brevi considerazione sulla soluzione per enumerazione totale e sulla soluzione approssimata per arrotondamento di problemi di PLI, descriveremo una delle più note famiglie di algoritmi per la soluzione esatta di problemi di PLI: gli algoritmi di *Branch and Bound.*

Prima di procedere vogliamo introdurre un concetto molto importante, quello di *problema rilassato*, che verrà ampiamento ripreso e utilizzato nel seguito. Dato il problema PLI si definisce suo *rilassamento lineare* (o *problema rilassato associato*) il seguente problema.

$$\begin{array}{ll} \max & c^T x \\ & Ax \leq b \\ & 0 \leq x \leq U. \end{array} \qquad (PR)$$

Si tratta cioè di un problema di PL ottenuto eliminando dal problema PLI il vincolo di interezza. La regione di un Problema di PLI è ovviamente contenuta in quella del suo rilassamento PR (che è ottenuto, lo ripetiamo, *eliminando* un vincolo da PLI) e quindi ne segue che il valore ottimo di PR è sicuramente maggiore o uguale di quello di PLI. In particolare, se la soluzione ottima di PR è intera allora questa è anche la soluzione ottima di PLI.

Per chiarire ulteriormente quanto detto facciamo un semplice esempio informale. Supponiamo di voler trovare lo studente più alto della facoltà di economia e commercio. Si tratta di trovare lo studente ("punto ammissibile") che massimizza la funzione obiettivo che associa ad ogni studente la sua altezza. Consideriamo poi il problema di trovare lo studente più alto di tutta l'università ("problema rilassato"). Ovviamente l'insieme ammissibile di questo nuovo problema è più ampio di quello del problema precedente, comprendendo tutti gli studenti di economia e commercio più gli studenti di tutte le altre facoltà. Se noi risolviamo il "problema rilassato" il valore ottimo che troviamo (altezza dello studente più alto di tutta l'università) è ovviamente maggiore o tutt'al più uguale al valore ottimo del problema di trovare lo studente più alto della facoltà di economia e commercio. Se poi risulta che lo studente più alto di tutta l'università è proprio uno studente della facoltà di economia e commercio, allora è ovvio che questo studente è anche "la soluzione ottima" del primo problema, è cioè lo studente più alto della facoltà di economia e commercio.

4.2.1 Soluzione per enumerazione totale

Nelle ipotesi fatte la regione ammissibile di PLI è costituita da un insieme finito di punti: l'insieme di punti a cordinate intere contenuti nell'insieme costituito dai punti che soddisfano i vincoli espressi dalle disequazioni lineari di PLI (che coincide con l'insieme ammissibile di PR) é limitato per ipotesi. In linea di principio è quindi sempre possibile risolvere PLI calcolando il valore della funzione obiettivo in ogni punto ammissibile e scegliendo quello che la massimizza. Definiremo questo metodo **Enumerazione Totale**. Se la cardinalità dell'insieme ammissibile di PLI è piccola allore l'enumerazione totale non solo è possibile, ma è certamente il modo migliore di risolvere un problema di PLI. Se, viceversa, la cardinalità dell'insieme delle soluzioni ammissibili è molto grande, l'enumerazione totale diviene non proponibile.

Osserviamo che se abbiamo un problema di PLI con dieci variabili e supponiamo che ognuna di queste variabili possa assumere dieci valori diversi, il numero di possibili punti ammissibili è di 10^{10}, cioè 10 miliardi. Questo semplice esempio mostra che l'enumerazione totale è raramente utile nei casi pratici, dove spesso il numero di variabili intere è dell'ordine delle centinaia se non delle migliaia.

4.2.2 Soluzione approssimata per arrotondamento

Abbiamo già avuto modo di osservare, nell'introduzione di questo paragrafo, che se la soluzione ottima del rilassamento lineare di PLI, $\bar{x}$, è intera allora $\bar{x}$ è la soluzione ottima di PLI. Se, invece, anche una sola delle componenti di $\bar{x}$ non è intera $\bar{x}$ non è nemmeno ammisibile per PLI.

Si può allora essere tentati di concludere che la soluzione ottima di PLI può essere ottenuta *arrotondando $\bar{x}$ a un punto a coordinate intere "vicino" a $\bar{x}$.*

Questo metodo di arrotondamento può essere un metodo pratico di grande utilità per la soluzione di PLI, se tutte variabili del problema sono variabili intere che rappresentano il numero di oggetti indivisibili usati o prodotti, ed è ragionevole aspettarsi che le variabili stesse abbiano valori abbastanza grandi all'ottimo. Inoltre i vincoli lineari dovrebbero essere tali da poter facilmente decidere se l'ammissibilità è preservata arrotondando una variabile non intera al più vicino valore intero (o, per esempio, al più vicino valore intero inferiore). Per esempio, se x_1 rappresenta il numero di automobili di un certo modello che devono essere assemblate durante un mese e se esiste una limitazione inferiore su x_1, è ragionevole aspettarsi che un valore $\bar{x}_1 = 1000.1$ possa essere arrotondato a 1001 senza che venga violata l'ammissibilità. Se tutte le variabili del problema sono di questo tipo e se esiste un metodo semplice di passare dalla soluzione non intera $\bar{x}$ di PR a una soluzione "vicina", intera e ammisibile per PLI, allora

un ragionevole approccio pratico alla soluzione di PLI è quello di risolvere il suo rilassamento lineare ed arrotondare quindi la soluzione.

Tuttavia, se il valore delle variabili della soluzione ottima del rilassamento lineare è presumibilmente piccolo, questa tecnica non è molto sensata. Se, per esempio, x_1 rappresenta il numero di portaaeri che devono essere costruite dall'esercito italiano, è intuitivo capire che è molto difficile decidere se si debba arrotondare $\bar{x}_1 = 2.5$ a 2 o a 3, e che le due possibilità portano a risultati completamente diversi. La situazione diventa ancora più drammatica quando le variabili intere sono, in realtà, variabili binarie. Abbiamo visto nel primo paragrafo di questo capitolo che questa è una situazione abbastanza frequente. In questi problemi le variabili 0-1 sono variabili che indicano quale di due possibili scelte alternative deve essere attuata. In questo caso l'arrotondamento è totalmente privo di senso logico. Se $\bar{x} \in I\!R^n$, possono esistere 2^n vettori interi ottenuti arrotondando le componenti non intere di $\bar{x}$ al valore intero superiore o inferiore più prossimo, e non si vede con quale criterio si possa sceglierne uno quale soluzione di PLI. Inoltre molti (se non tutti) di questi 2^n vettori possono non essere ammissibili per PLI, e il problema di determinare un vettore a componenti intere, ammissibile per PLI, e "vicino" a $\bar{x}$ può essere, in generale, un problema di difficoltà paragonabile alla soluzione di PLI.

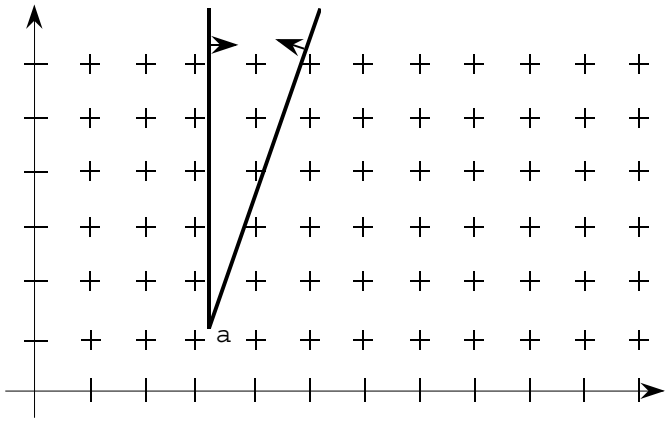


Figura 4.2: Soluzione per arrotondamento: primo esempio.

Consideriamo l'esempio di figura 4.2. Supponiamo che la soluzione del rilassamento lineare sia a ($\bar{x} = a$). Il punto a ha tutte le componenti non intere e tutti i punti a componenti intere ottenuti arrotondando le sue componenti all'intero immediatamente superiore o inferiore non sono ammisibili per PLI. Da questo esempio dovrebbe risultare chiaro, inoltre, che è possibile costruire problemi PLI in cui la soluzione ottima del relativo rilassamento lineare è lontana "quanto si vuole" dal più vicino punto ammisibile di PLI.

Anche nel caso in cui possa essere trovato un punto ammissibile "vicino" a $\bar{x}$, questi può essere molto lontano dalla soluzione ottima di PLI.

Un esempio al riguardo è dato nella figura 4.3, dove la soluzione ottima del rilassamento lineare è a, b è la soluzione ottenuta per arrotondamento, mentre c è la vera soluzione di PLI.
Tutte le considerazioni svolte finora mostrano chiaramente la necessità di algoritmi esatti ed efficienti per la soluzione di PLI. Un possibile algoritmo di questo genere viene illustrato nella sezione seguente.

4.2.3 La Tecnica del Branch and Bound

Abbiamo già visto che una possibile, semplice tecnica di soluzione per PLI, purtroppo quasi mai utilizzabile in pratica, è l' enumerazione totale.

Il "Branch and Bound" (BB) fornisce, allora, una metodologia di ricerca della soluzione ottima che effettua una esplorazione *parziale* dell'insieme delle soluzioni ammisibile. In particolare la funzione obiet-

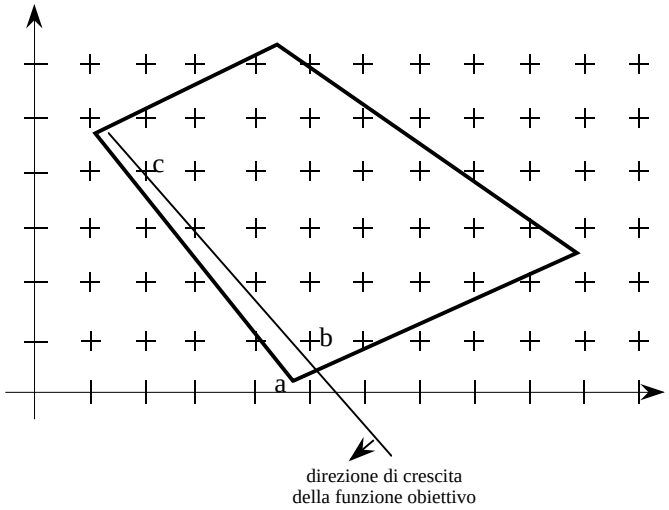


Figura 4.3: Soluzione per arrotondamento: secondo esempio.

tivo viene calcolata per una sottoinsieme (piccolo) delle soluzioni ammissibili con la proprietà di contenere almeno una soluzione ottima. Descriviamo ora in maggior dettaglio i vari aspetti della procedura. Sia S_0 l'insieme delle soluzioni ammissibili PLI. Una *partizione* di S_0 è una famiglia di sottoinsiemi $(S_1, \ldots, S_r)$, $r \geq 2$ tale che:

$$S_i \cap S_j = \emptyset \qquad \text{per ogni coppia} \qquad 1 \leq i < j \leq r$$

e

$$\bigcup_{i=1}^{r} S_i = S_0$$

Evidentemente, la soluzione ottima del problema originario è data dal massimo tra i valori $z_1^* = c^T x_1^*, \ldots, z_r^* = c^T x_r^*$ dove x_i^* è la soluzione ottima del sottoproblema PLI_i definito dal problema di massimizzare la funzione obiettivo del problema PLI sulla regione ammissibile S_i (nel seguito, con un piccolo abuso di notazione, indicheremo spesso con S_i sia la regione ammissibile sia il sottoproblema PLI_i). Se un certo sottoproblema PLI_i risulta, a sua volta, di difficile soluzione si partiziona ulteriormente l'insieme S_i producendo nuovi sottoproblemi ed iterando la procedura fino a che il problema originario non risulti decomposto in problemi elementari di facile soluzione.

Cerchiamo di chiarire la procedura con un esempio informale. Supponiamo di dover trovare la persona più alta dell'universita. Risolvere questo problema confrontando le altezze di tutti gli studenti fra loro (enumerazione totale) può risultare in un processo troppo lungo. Si può allora chiedere ai presidi di ogni facoltà qual è lo studente più alto delle loro facolta e scegliere il più alto tra questi studenti. Abbiamo decomposto il problema originario in più sottoproblemi: la determinazione dello studente più alto di ogni facoltà. In alcuni casi la risoluzione di questi sottoproblemi può essere agevole. Per esempio il preside della facoltà di Noia Applicata non ha difficoltà alcuna ad individuare lo studente più alto tra i suoi *tre* iscritti. Anche il preside della facoltà di Bel Turpiloquio non ha difficoltà ad individuare il suo studente più alto, avendo appena dovuto rompere le porte di tutte le aule del primo anno per permettere (come da regolamento) il passaggio senza chinare la testa della matricola Amilcare Basso. In altri casi il preside può non essere in grado di fornire facilmente il nome dello studente più alto. In questo caso si può scomporre ulteriormente il problema. Per esempio, il preside della facoltà di Dilapidazione e Latrocinio chiede a tutti i direttori dei suoi istituti di fornirgli il nome dello studente più alto che frequenta l'istituto stesso. Da questi egli potrà così risalire al nome dello studente più alto della facoltà. E così via.

È evidente che la procedura appena descritta non compie una totale enumerazione dell'insieme S ed è computazionalmente efficiente solo se il numero dei sottoproblemi generati si mantiene estremamente limitato e quindi solo se la strategia di soluzione dei sottoproblemi è sufficientemente efficace.

In generale però, risolvere un sottoproblema può essere altrettanto gravoso che risolvere il problema originario. È per questo che, in luogo della soluzione esatta del problema S_i si preferisce calcolare un **"upper bound"** di z_i^* e cioè un valore $U_i \geq z_i^*$. Tale valore viene poi confrontato con il miglior valore $\tilde{z}$ (ad un certo passo della procedura) della funzione obiettivo calcolata in un punto ammissibile (**ottimo corrente**). Se il valore approssimato risulta non superiore a quello dell'ottimo corrente, ovvero se

$$\tilde{z} \geq U_i \geq z_i^*$$

si deduce che non esiste nell'insieme S_i un punto in cui la funzione obiettivo abbia un valore migliore di $\tilde{z}$. Un tale riultato ci permette di sospendere l'esame dell'insieme S_i e di eliminarlo dalla lista dei sottoproblemi da risolvere.

Per tornare all'esempio informale precedente, supponiamo che il preside della facoltà di Bel Torpiloquio abbia comunicato che Amilcare Basso è alto 2 metri e 10 centimetri (ottimo corrente). Supponiamo anche che il preside della facoltà di Occultismo, pur senza sapere esattamente qual è l'altezza esatta del suo studente più alto, sia riuscito ad avere la certezza, con mezzi solo a lui noti, che ogni studente della sua facoltà non è alto più di 2 metri (upper bound). È evidente che è inutile fare lo sforzo di cercare di trovare l'altezza esatta del più alto studente della facoltà di Occultismo, tanto Amilcare Basso è comunque più alto.

Evidentemente la soluzione del problema originario sarà tanto più efficiente quanto migliori saranno i valori degli "upper bound" ed a loro volta tali valori approssimeranno tanto meglio il valore ottimo del sottoproblema quanto più efficace sarà stata la decomposizione del problema originario. Di conseguenza l'efficienza della tecnica del BB dipende essenzialmente dalla qualità delle due strategie che ne caratterizzano la struttura:

(a) Strategia di Soluzione, ovvero la strategia per il calcolo di un valore che approssimi per eccesso (in un problema di massimo) il valore ottimo di un sottoproblema.

(b) Strategia di Separazione, ovvero la strategia per la scelta della partizione dell'insieme delle soluzioni ammissibili di un sottoproblema.

È anche evidente che a seconda della strategia di soluzione e di quella di separazione adottate, lo schema generale appena descritto si concretizzerà in un algoritmo differente.

Nel seguito di questo paragrafo descriveremo un possibile schema (semplificato) di BB che corrisponde a uno degli schemi di BB più utilizzati nella pratica dando al contempo maggiore sistematicità al metodo presentato.

Ricordiamo che vogliamo risolvere un problema di PLI della seguente forma

$$\begin{array}{ll} \max & c^T x \\ & Ax \leq b \\ & 0 \leq x \leq U \\ & x \quad \text{intera} \end{array} \qquad (PLI)$$

Questo problema, nel seguito sarà indicato anche come problema S_0. Nel corso del metodo considereremo dei sottoproblemi che hanno una struttura simile. Più precisamente considereremo sottoproblemi S_i con la seguente struttura.

$$\begin{array}{ll} \max & c^T x \\ & Ax \leq b \\ & l_i \leq x \leq u_i \\ & x \quad \text{intera,} \end{array} \qquad (S_i)$$

dove i vettori n dimensionali l_i e u_i sono calcolati dall'algoritmo. Nel seguito indicheremo, inoltre, con x_i^* e z_i^* rispettivamente una soluzione ottima e il valore ottimo di S_i.

La strategia che proponiamo per calcolare un upper bound U_i di z_i^* è la seguente:

dato il problema S_i, risolvere il suo rilassamento lineare.

Indichiamo con y^i la soluzione ottima del rilassamento lineare di S_i e con U_i il corrispondente valore ottimo. Per quanto visto precedentemente il valore ottimo U_i così ottenuto costituisce un upper bound di z_i^*, inoltre il rilassamento lineare è facilmente risolubile con il metodo del simplesso.

Dobbiamo ora descrivere un metodo per la separazione di un generico S_i. Supponiamo di aver risolto il rilassamento lineare di S_i e sia y^i la sua soluzione ottima e, come già detto, con U_i il corrispondente valore ottimo. Se y^i ha tutte componenti intere allora $y^i = x_i^*$ e il problema non va separato. Se U_i è minore o uguale all'*ottimo corrente*, cioè al valore massimo della funzione obiettivo finora trovato, il problema non può dar origine ad una soluzione intera migliore di quella corrente e non è necessario separarlo per trovare la sua soluzione ottima intera. Supponiamo quindi che nessuno di questi due casi si sia verificato, e vediamo come seperare questo sottoproblema. Sia y_j^i una componente non intera del vettore y^i (almeno una ne deve esistere), α_j la sua parte intera inferiore (ossia il più grande intero minore o uguale a $(x_i^*)_j$) e β_j la sua parte intera superiore (ossia il più piccolo intero maggiore o uguale a y_j^i). Ovviamente valgono le seguenti relazioni:

$$\begin{aligned} &\alpha_j < \beta_j \\ &\beta_j - \alpha_j = 1. \end{aligned}$$

Separiamo il problema S_i nei seguenti due problemi

$$\begin{aligned} \max \quad & c^T x \\ & Ax \leq b \\ & l_i^1 \leq x \leq u_i^1 \\ & x \quad \text{intera,} \end{aligned}$$

dove

$$\begin{aligned} &(l_i^1)_j = (l_i)_j \ \text{ se } \ i \neq j \quad \text{e} \quad (l_i^1)_j = \beta_j \ \text{ se } \ i = j \\ &u_i^1 = u_i \end{aligned}$$

e

$$\begin{aligned} \max \quad & c^T x \\ & Ax \leq b \\ & {l_i}^2 \leq x \leq {u_i}^2 \\ & x \quad \text{intera,} \end{aligned} \tag{S_i}$$

dove

$$\begin{aligned} &(u_i^2)_j = (u_i)_j \ \text{ se } \ i \neq j \quad \text{e} \quad (u_i^2)_j = \alpha_j \ \text{ se } \ i = j \\ &l_i^2 = l_i \end{aligned}$$

Notiamo che è facile convincersi che l'unione delle regioni ammissibili di questi due problemi dà la regione ammissibile del problema S_i, di modo tale che questa è in effetti una separazione del problema S_i del tipo prima considerato.

Per concludere vogliamo osservare che dobbiamo considerare un altro caso che finora non abbiamo menzionato. Nel risolvere il rilassamento del problema S_i può accadere che la regione ammissibile del rilassamento risulti vuota. Siccome la regione ammissibile del problema S_i è contenuta in quella del suo rilassamento, questo vuol dire che il problema S_i ha regione ammissibile vuota, e quindi non c'è bisogno di separarlo.

Abbiamo ora tutti gli elementi per descrivere il metodo di BB proposto.

1. Inizializzazione

Inizialmente il problema $S_0 = PLI$ è l'unico problema candidato. La procedura inizia con l'applicazione della strategia di soluzione ad S_0 ottenendo un primo "upper bound" U_0.

Inoltre, viene calcolato un **"lower bound"** $\tilde{z}$ per il problema S_0, cioè un valore sicuramente non superiore a quello z^* della soluzione ottima. Tale valore viene usualmente ottenuto in corrispondenza ad una soluzione ammissibile $\tilde{x}$ facile da individuare (ad es. mediante una semplice euristica). I valori $\tilde{x}$ e $\tilde{z}$ vengono successivamente aggiornati nel corso della procedura non appena vengono individuate soluzioni ammissibili per il problema originario con un valore più alto della funzione obiettivo. Se non è possibile individuare facilmente una soluzione ammissibile si può porre $\tilde{z} = -\infty$, mentre $\tilde{x}$ viene, per il momento lasciato indefinito.

Se l'ottimo del problema rilassato y^0 appartiene all'insieme S_0 il problema è risolto. Altrimenti è necessario applicare una strategia di separazione al problema S_0 ottenendo due nuovi problemi candidati S_1 ed S_2.

2. Passo generico dell'algoritmo.

Ad un generico passo dell'algoritmo supponiamo di avere la lista $\mathcal{L} = \{S_1, \ldots, S_q\}$ dei sottoproblemi che non sono ancora stati esaminati (**problemi terminali**). Illustriamo ora brevemente le possibili azioni dell'algoritmo in un passo generico.

2.1. Scelta del problema aperto da esaminare

Si estrae un problema dalla lista $\mathcal{L}$, diciamo S_i, che diventa il *problema corrente.* Se la lista $\mathcal{L}$ è vuota, l'algoritmo termina e $\tilde{x}$, la soluzione ottima corrente, è la soluzione ottima del problema. Se la lista $\mathcal{L}$ non è vuota possono essere adottate diverse strategie per la scelta del problema S_i da esaminare. Diamo qui due esempi:

(1) *Scelta con criterio di priorità FIFO (First In First Out).* In questo caso i sottoproblemi terminali sono gestiti dalla procedura secondo lo schema a *coda.* In particolare, il problema scelto è il primo problema generato nella fase di separazione (tra quelli presenti nella lista).

(2) *Scelta con criterio di priorità LIFO (Last In First Out).* In questo caso i sottoproblemi terminali sono gestiti dalla procedura secondo lo schema a *stack.* In particolare, il problema scelto è l'ultimo problema generato nella fase di separazione.

Si risolve il rilassamento lineare di S_i. Se S_i è vuoto si elimina e il passo dell'algoritmo è terminato. Si può quindi passare ad esaminare un nuovo problema della lista $\mathcal{L}$.

Se S_i non è vuoto si calcolano y^i e U_i, rispettivamente la soluzione ottima e il valore ottimo del problema rilassato (si noti che siccome stiamo supponendo che la regione ammissibile del problema rilassato è limitata, e avendo appena verificato che non è vuota, per il teorema fondamentale della Programmazione Lineare, il rilassamento del problema in esame deve avere una soluzione ottima y^i).

2.1. Confronto con l'ottimo corrente

Se $U_i \leq \tilde{z}$, detto z_i^* il valore ottimo della funzione obiettivo per il problema S_i, abbiamo che:

$$z_i^* \leq U_i \leq \tilde{z}$$

e quindi che nessuna soluzione *migliore di* $\tilde{z}$ può essere contenuta in S_i. In tal caso il problema S_i può essere eliminato della lista dei problemi candidati e viene definito **chiuso** e il passo dell'algoritmo è terminato. Si può quindi passare ad esaminare un nuovo problema della lista $\mathcal{L}$.

2.3. Aggiornamenti

Arrivati a questo punto sono possibili due casi, che esaminiamo separatamente. Ricordiamo che y^i è l'ottimo del problema rilassato, cioè il punto che soddisfa $U_i = c^T y^i$.

2.3.1. y^i è intero

Se l'ottimo del problema rilassato y^i ha componenti intere abbiamo che esso è ottimo anche per il sottoproblema S_i. Inoltre poiché $U_i > \tilde{z}$, y^i è una soluzione ammissibile del problema originario con valore della funzione obiettivo ($c^T y^i = U_i$) migliore di $\tilde{z}$. Di conseguenza, possiamo porre $\tilde{x} = y^i$ e $\tilde{z} = U_i$. (**aggiornamento dell'ottimo corrente**). Il problema S_i viene chiuso.

2.3.2. y^i non è intero

In tal caso si deve applicare la strategia di separazione precedentemente illustrata, produrre i due nuovi sottoproblemi ed aggiungerli alla lista $\mathcal{L}$ dei problemi terminali.

In tutt'e due i casi il passo dell'algoritmo è terminato e si può passare ad esaminare un nuovo problema della lista $\mathcal{L}$. Si riporta di seguito lo schema a blocchi dell'algoritmo appena descritto. □

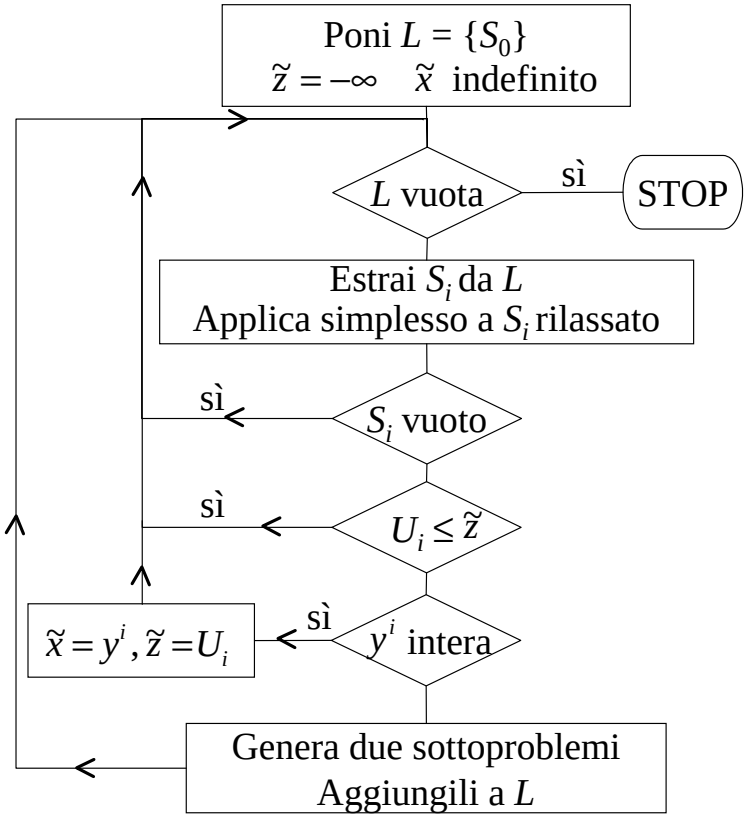


Figura 4.4: Schema di branch and bound

4.3 Esempi

Completiamo il capitolo con degli esempi di applicazione del Branch and Bound.

Esempio1. Sia dato il seguente problema di programmazione lineare a numeri interi[4]:

$$
\begin{array}{lll}
(S_0) & \max & -x_1 + 2x_2 \\
 & & -4x_1 + 6x_2 \leq 9 \\
 & & x_1 + x_2 \leq 4 \\
 & & x \geq 0 \\
 & & x_1, x_2 \in \mathcal{Z}
\end{array}
$$

Il problema è un problema di massimizzazione, quindi le soluzioni dei problemi rilassati forniranno dei limiti superiori ("upper bound") al valore della soluzione ottima intera. Inoltre, il valore della funzione obiettivo calcolato in corrispondenza ad una soluzione ammissibile fornirà un limite inferiore ("lower bound"). Si noti che, benché non siano esplicitamente presenti degli upper bound sulle variabili (del tipo $x \leq U$), il problema ha ovviamente una regione ammissibili limitata (infatti le variabili sono non negative e la loro somma deve essere minore o uguale a quattro, si veda il secondo vincolo). È facile convincersi quindi, che nulla cambia nell'applicazione dell'algoritmo e nella sua validità.

Poniamo inizialmente $\tilde{z} = -\infty$, l'ottimo corrente è per il momento non definito. Poniamo il problema S_0 nella lista $\mathcal{L}$.

Viene selezionato un problema in $\mathcal{L} = \{S_0\}$. In particolare, viene estratto dalla lista il problema S_0, unico problema nella lista.

La soluzione ottima y^0 del rilassamento lineare di S_0 è $y^0 = (3/2, 5/2)^T$ (si veda la figura Figura 4.5) e fornisce un "upper bound" pari a $U_0 = c^T y^0 = 7/2$.

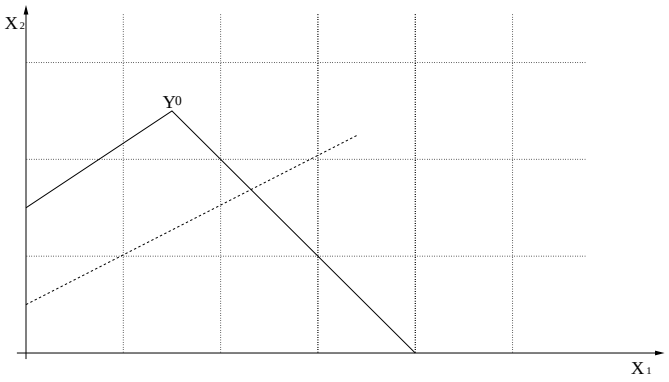


Figura 4.5: Problema S_0.

Non è possibile chiudere il problema S_0 in quanto la soluzione y^0 non è intera e l' "upper bound" è strettamente maggiore del "lower bound" che vale al momento $-\infty$. Siccome la soluzione ottima del rilassamento non è intera, si procede alla separazione di S_0. Possiamo separare rispetto ad una qualsiasi delle variabili non intere, per semplicità decidiamo di separare rispetto alla variabile con indice più piccolo (x_1). A tale scopo, aggiungiamo ai vincoli della formulazione di S_0, i due vincoli $x_1 \leq 1$ ($x_1 \leq \lfloor y_1^0 \rfloor$[5]) e

[4]Ricordiamo che con $\mathcal{Z}$ si indica l'insieme dei numeri interi

[5]Ricordiamo che $\lfloor y_1^0 \rfloor$ indica il più grande numero intero minore o uguale a y_1^0

$x_1 \geq 2$ ($x_1 \geq \lceil y_1^0 \rceil$[6]), ottenendo i due problemi seguenti :

$$
\begin{array}{lll}
(S_1) & \max & -x_1 + 2x_2 \\
 & & -4x_1 + 6x_2 \leq 9 \\
 & & x_1 + x_2 \leq 4 \\
 & & x_1 \leq 1 \\
 & & x \geq 0 \\
 & & x_1, x_2 \in \mathcal{Z}
\end{array}
$$

$$
\begin{array}{lll}
(S_2) & \max & -x_1 + 2x_2 \\
 & & -4x_1 + 6x_2 \leq 9 \\
 & & x_1 + x_2 \leq 4 \\
 & & x_1 \geq 2 \\
 & & x \geq 0 \\
 & & x_1, x_2 \in \mathcal{Z}
\end{array}
$$

che vengono messi in $\mathcal{L}$: $\mathcal{L} = \{S_1, S_2\}$. Estraiamo un problema da $\mathcal{L}$, per esempio, S_2. Risolviamo il rilassamento di S_2 (si veda la figura Figura 4.6), e otteniamo $y^2 = (2, 2)^T$ e un corrispondente $U_2 =$

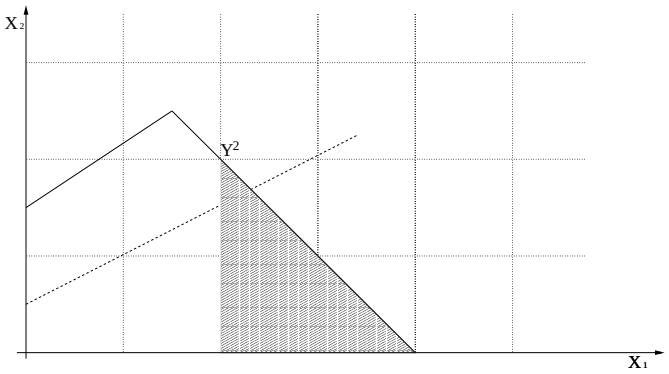


Figura 4.6: Problema S_2.

$c^T y^2 = 2$. Siccome la soluzione del problema rilassato è intera e il valore ottimo corrispondente è migliore dell'ottimo corrente $\tilde{z}$, aggiorniamo e poniamo: $\tilde{z} = 2$ e $\tilde{x} = (2, 2)^T$. Il problema S_2 è chiuso e ora $\mathcal{L} = \{S_1\}$.

Estraiamo un problema da $\mathcal{L}$, siccome c'è solo S_1 esaminiamo S_1. Risolviamo il rilassamento di S_1(si veda la figura Figura 4.7) e otteniamo $y^1 = (1, 13/6)^T$ e un corrispondente $U_1 = c^T y^1 = 10/3$. Siccome $U_1 > \tilde{z} = 2$ e y^1 non è intera, generiamo due sottoproblemi e li mettiamo in $\mathcal{L}$. Poiché solo x_2 è frazionaria , possiamo fare il branching solo su x_2. Otteniamo così i due problemi:

$$
\begin{array}{lll}
(S_3) & \max & -x_1 + 2x_2 \\
 & & -4x_1 + 6x_2 \leq 9 \\
 & & x_1 + x_2 \leq 4
\end{array}
$$

[6]Ricordiamo che $\lceil y_1^0 \rceil$ indica il più piccolo numero intero maggiore o uguale a y_1^0

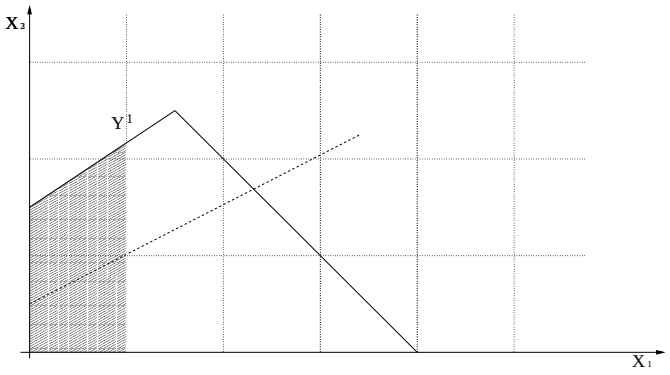


Figura 4.7: Problema S_1.

$$
\begin{aligned}
& x_1 \leq 1 \\
& x_2 \leq 2 \\
& x \geq 0 \\
& x_1, x_2 \in \mathcal{Z}
\end{aligned}
$$

$$
\begin{aligned}
(S_4) \quad \max \quad & -x_1 + 2x_2 \\
& -4x_1 + 6x_2 \leq 9 \\
& x_1 + x_2 \leq 4 \\
& x_1 \leq 1 \\
& x_2 \geq 3 \\
& x \geq 0 \\
& x_1, x_2 \in \mathcal{Z}
\end{aligned}
$$

A questo punto $\mathcal{L} = \{S_3, S_4\}$. Esaminiamo il problema S_4. Trovo che il suo rilassamento lineare è vuoto. Chido il problema. Estraggo dalla lista $\mathcal{L}$ l'unico problema presente, S_3. Il suo rilassamento lineare ha per soluzione $y^3 = (3/4, 2)^T$ (si veda la figura Figura 4.8), con un corrispondente valore $U_3 = c^T y^3 = 13/4$.

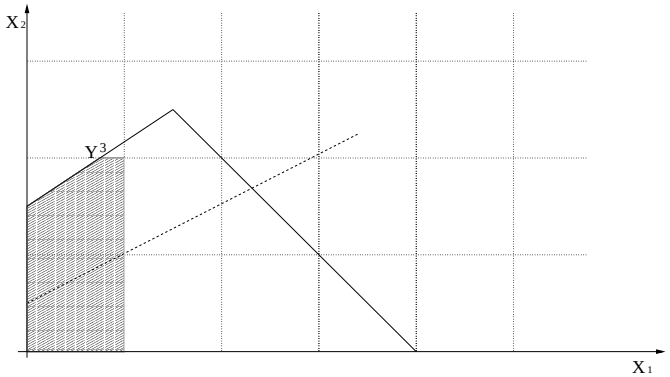


Figura 4.8: Problema S_3.

Siccome $U_3 > \tilde{z}$ e y^3 non è intero, genero due sottoproblemi a partire da S_3 facendo branching sull'unica variabile frazionaria, x_1:

$$
\begin{array}{lll}
(S_5) & \max & -x_1 + 2x_2 \\
 & & -4x_1 + 6x_2 \leq 9 \\
 & & x_1 + x_2 \leq 4 \\
 & & x_1 \leq 1 \\
 & & x_1 \leq 0 \\
 & & x_2 \leq 2 \\
 & & x \geq 0 \\
 & & x_1, x_2 \in \mathcal{Z}
\end{array}
$$

$$
\begin{array}{lll}
(S_6) & \max & -x_1 + 2x_2 \\
 & & -4x_1 + 6x_2 \leq 9 \\
 & & x_1 + x_2 \leq 4 \\
 & & x_1 \leq 1 \\
 & & x_1 \geq 1 \\
 & & x_2 \leq 2 \\
 & & x \geq 0 \\
 & & x_1, x_2 \in \mathcal{Z}
\end{array}
$$

Notiamo che in S_5 i due vincoli $x_1 \leq 1$ è reso superfluo dal nuovo vincolo $x_1 \leq 0$. Abbiamo lasciato il vincolo superfluo per rendere chiaro che ogni volta che si compie un operazione di branching per generare due nuovi sottoproblemi che eridatano tutti i vincoli del problema che li genera e a cui vengono aggiunti i vincoli specifici del branching. Analogamente, per S_6, i due vincoli $x_1 \leq 1$ e $x_1 \geq 1$ dicono che $x_1 = 1$, ma abbiamo preferito lasciare i due vincoli distinti.

A questo punto $\mathcal{L} = \{S_5, S_6\}$. Esaminiamo il problema S_6. Il suo rilassamento lineare ha per soluzione $y^6 = (1, 2)^T$(si veda la figura Figura 4.9), con un corrispondente valore $U_6 = c^T y^6 = 3$. Siccome

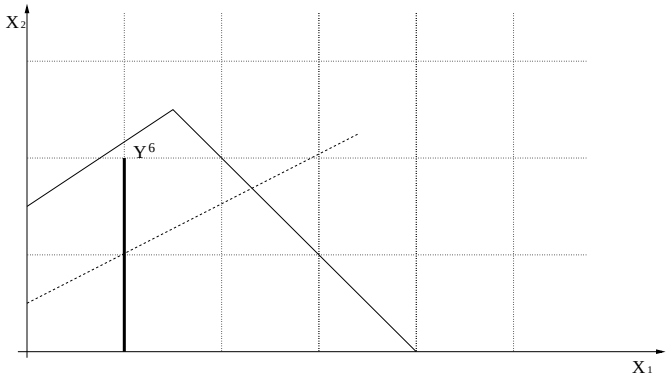


Figura 4.9: Problema S_6.

$U_6 > \tilde{z} = 2$ e y^6 è intero, aggiorniamo e poniamo: $\tilde{z} = 3$ e $\tilde{x} = (1, 2)^T$. Il problema S_6 è chiuso e ora $\mathcal{L} = \{S_5\}$.

Estraggo il problema S_5 dalla lista dei problemi aperti e ne risolvo il rilassamento. Ottengo $y^5 = (0, 3/2)$ e $U_5 = c^T y^5 = 3$ (si veda la figura Figura 4.10). Siccome $U_5 \leq \tilde{z} = 3$, chiudo il problema.

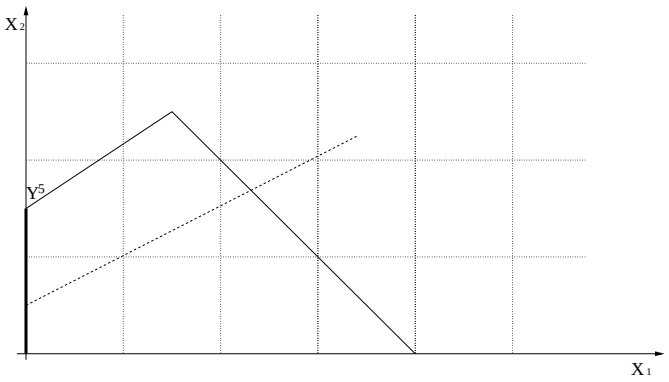


Figura 4.10: Problema S_5.

Siccome la lista è ora vuota, il punto $(1, 2)^T$, ottimo corrente, è la soluzione del problema.

I vari passi della procedura di soluzione possono essere schematizzati utilizzando il cosiddetto *Albero di Enumerazione* riportato in figura 4.11. nodi indicano i problemi mentre i rami stabiliscono le relazioni padre-figlio (generatore-generato) nel processo di generazione.

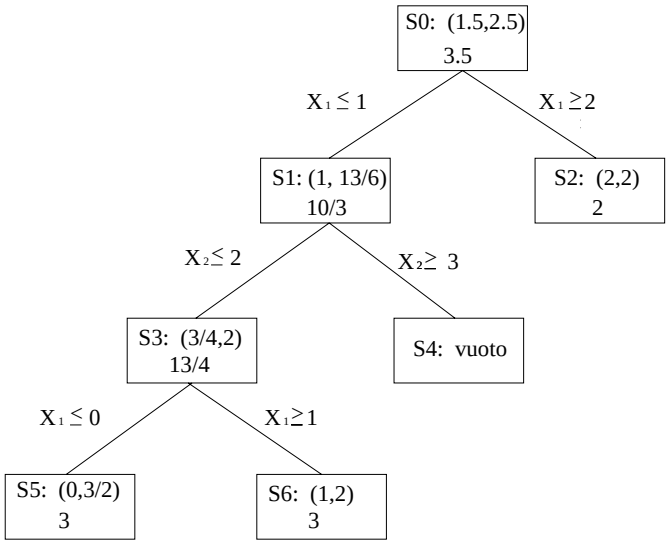


Figura 4.11: Albero di enumerazione.

Notiamo che la strategia adottato in questo caso, nello scegliere i problemi nella lista è stata quella LIFO.

Esempio 2. Passiamo ora ad esaminare, più sinteticamente, il secondo esempio. Sia dato il seguente problema di programmazione lineare a numeri interi.

$$\begin{array}{ll} \max & x_1 + x_2 \\ & 2x_1 + 5x_2 \leq 30 \\ & 4x_1 - 3x_2 \leq 6 \\ & 0 \leq x_1 \\ & 0 \leq x_2 \\ & x_1, x_2 \in \mathcal{Z} \end{array} \qquad (S_0)$$

Notiamo che in anche in questo problema non sono esplicitamente presenti gli "upper bound" sule variabili. Essendo comunque la regione ammissibile del problema rilassato limitato, non è necessario inserirli esplicitamente.

Poniamo inizialmente $\tilde{z} = -\infty$ e risolviamo il rilassamento lineare di S_0. Si ottiene

$$U_0 = \frac{114}{13} \qquad y_1^0 = \frac{60}{13} \qquad y_2^0 = \frac{54}{13}.$$

Si elimina S_0 dalla lista $\mathcal{L}$, e vi si aggiungono i due nuovi problemi S_1 e S_2 individuati dai seguenti vincoli aggiuntivi.

$$\begin{array}{ll} S_1: & x_1 \leq 4 \\ S_2: & x_1 \geq 5. \end{array}$$

Si avrà pertanto $\mathcal{L} = \{S_1, S_2\}$.

Si sceglie S_1 da L e lo si risolve il suo rilassamento. Si ottiene

$$U_1 = \frac{42}{5} \qquad y_1^1 = 4 \qquad y_2^1 = \frac{22}{5}$$

Essendo $U_1 > \tilde{z}$ si elimina S_1 da $\mathcal{L}$ e vi si aggiungono, invece altri due problemi, S_3 e S_4, individuati dai seguenti vincoli aggiuntivi.

$$\begin{array}{lll} S_3: & x_1 \leq 4 & x_2 \leq 4 \\ S_4: & x_1 \leq 4 & x_2 \geq 5. \end{array}$$

Risulta pertanto $\mathcal{L} = \{S_2, S_3, S_4\}$.

Si sceglie S_3 da L e si risolve il suo rilassamento. Si ottiene

$$U_3 = 8 \qquad y_1^3 = 4 \qquad y_2^3 = 4$$

Essendo $U_3 > \tilde{z}$ ed essendo intera la soluzione trovata, si pone

$$\tilde{z} = U_3 = 8 \qquad \tilde{x}_1 = y_1^3 = 4 \qquad \tilde{x}_2 = y_2^3 = 4,$$

e si elimina il problema S_3. Risulta a questo punto $\mathcal{L} = \{S_2, S_4\}$.

Si sceglie da $\mathcal{L}$ S_4 e si risolve il suo rilassamento. Si ottiene

$$U_4 = \frac{15}{2} \qquad y_1^4 = \frac{5}{2} \qquad y_2^4 = 5$$

Essendo $U_4 < \tilde{z}$ si chiude S_4 e la lista diventa $\mathcal{L} = \{S_2\}$.

Si sceglie S_2 dalla lista e si risolve il suo rilassamento. Si vede che non esistono soluzioni ammisibili. Si chide S_2 e la lista L è vuota. La soluzione corrente è la soluzione ottima.

$$z^* = 8 \qquad x_1^* = 4 \qquad x_2^* = 4.$$

Capitolo 5

Grafi: nozioni fondamentali

In questo capitolo introduciamo la nozione di grafo ed alcune definizioni ad essa collegate. I grafi sono una struttura matematica molto usata nelle applicazioni e si prestano a rappresentare problemi apparentemente molto diversi tra di loro con un linguaggio semplice ed unificato. Questo spiega la loro importanza nella matematica applicata e, in particolare, nella Ricerca Operativa.

5.1 Definizioni fondamentali

Un *grafo non orientato* $G = (V, E)$ è definito da un insieme finito $V(G) = \{v_1, \ldots, v_n\}$ di elementi detti *nodi* o *vertici* e da un insieme $E(G) = \{e_1, \ldots, e_m\} \subseteq V \times V$ di coppie non ordinate di nodi dette *archi* o *spigoli*.

Dato l'arco $e = (v, w) = (w, v)$, i nodi v e w sono detti *estremi* di e, e si dice che l'arco e *incide* su u e v. Una comoda rappresentazione del grafo viene mostrata in figura 5.1a. I nodi sono rappresentati da cerchi,

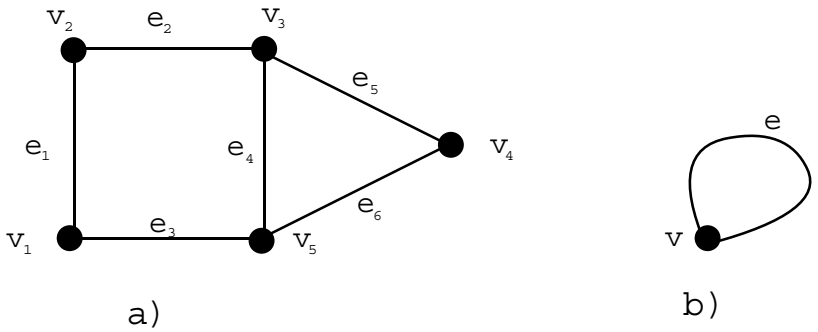


Figura 5.1: a) Grafo non-orientato b) Loop

metre gli archi sono tratti di curva congiungenti i due estremi. Per il grafo $G = (V, E)$ rappresentato in figura si ha

$V = \{v_1, v_2, v_3, v_4, v_5\}$

$E = \{e_1, e_2, e_3, e_4, e_5, e_6\} = \{(v_1, v_2), (v_2, v_3), (v_1, v_5), (v_3, v_5), (v_3, v_4), (v_4, v_5)\}$.

Due nodi u, v sono detti *adiacenti* (reciprocamente) se l'arco (u, v) appartiene ad E. Nella figura sono adiacenti, per esempio, i nodi v_1 e v_2. Due archi sono detti *adiacenti* se hanno un estremo in comune, come gli archi e_2 ed e_4 in figura.

Si definisce *intorno* di un nodo v in G, indicato con $N(v)$, l'insieme dei nodi adiacenti a v. Nella figura $N(v_5) = \{v_1, v_3, v_4\}$. Un nodo v si dice *isolato* se $N(v) = \emptyset$. Si definisce *stella* di v in G, indicata con $\delta(v)$, l'insieme degli archi incidenti su v. Nella figura $\delta(v_5) = \{e_3, e_4, e_6\}$.

Si definisce *sottografo* di $G = (V, E)$ un grafo $H = (W, F)$ tale che $W \subseteq V$ e $F \subseteq E$. Si definisce *sottografo indotto* da $W \subseteq V$ in $G = (V, E)$ il grafo $H = (W, F)$, ove l'insieme degli archi F è tale che l'arco (u, v) appartiene a F se e solo se: (a) u e v appartengono a W e (b) $(u, v) \in E$. Informalmente possiamo dire che il sottografo H eredita tutti gli archi di G i cui estremi sono entrambi contenuti nel sottoinsieme W. In figura 5.2 vengono mostrati due sottografi del grafo di figura 5.1. Il primo non è un sottografo indotto ("manca" l'arco e_4), mentre il secondo è indotto in G dall'insieme di nodi $\{v_2, v_3, v_4, v_5\}$.

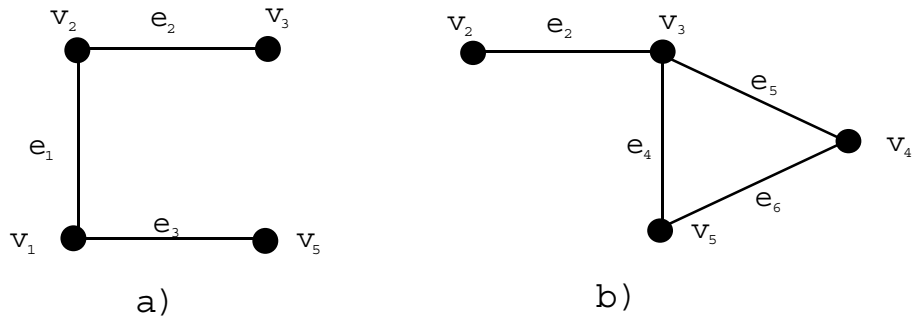


Figura 5.2: a) Sottografo b) Sottografo indotto

Un *grafo orientato* $G = (V, E)$ è definito da un insieme finito $V(G) = \{v_1, \ldots, v_n\}$ di elementi detti *nodi* e da un insieme $E(G) = \{e_1, \ldots, e_m\} \subseteq V \times V$ di coppie **ordinate** di nodi dette *archi*. Dato l'arco $e = (v, u)$, il primo nodo nella coppia (v) è detto *coda*, e si dice che l'arco e è *uscente* da v. Il secondo nodo (u) è detto *testa*, e si dice che l'arco e è *entrante* in u. L'arco si dice *orientato* dal nodo v al nodo u. I nodi v e u sono detti *estremi* di e, e si dice che l'arco e *incide* su v e u. Notiamo che a differenza del caso non orientato, l'arco (v, u) è distinto dall'arco (u, v). Una rappresentazione del grafo orientato viene mostrata in figura 5.3. L'unica differenza con la rappresentazione del grafo non orientato sta nella presenza sull'arco di una freccia che indica l'orientamento dell'arco stesso.

Analogamente al caso non-orientato, si definisce *stella* di v in G, indicata con $\omega(v)$, l'insieme degli archi incidenti su v. In figura 5.3, $\omega(v_1) = \{e_1, e_2, e_5\}$. La stella $\omega(v)$ può essere partizionata in *stella entrante* $\omega^-(v)$, cioè l'insieme degli archi entranti in v, e in *stella uscente* $\omega^+(v)$, l'insieme degli archi uscenti da v. Nell'esempio, $\omega^-(v_1) = \{e_2\}$, mentre $\omega^+(v_1) = \{e_1, e_5\}$. Sottografi e sottografi indotti sono definiti come per il caso non orientato.

Cammini, Cammini Orientati, Cicli. Dato un grafo $G = (V, E)$ (indifferentemente orientato o non orientato), ove $V = \{v_1, \ldots, v_n\}$ e $E = \{e_1, \ldots, e_m\}$ diremo *cammino* in (di) G, un insieme ordinato $P = \{v_{j_0}, e_{h_1}, v_{j_1}, e_{h_2}, \ldots, v_{j_{p-1}}, e_{h_p}, v_{j_p}\}$ con: $j_k \in \{1, \ldots, n\}$, $h_i \in \{1, \ldots, m\}$, per $k = 0, \ldots, p$ e $i = 1, \ldots, p$, e con l'arco e_{h_i} incidente sui nodi $v_{j_{i-1}}$ e v_{j_i}.[1] Notiamo che nel caso del grafo orientato, l'orientamento degli archi del cammino è indifferente. Si richiede semplicemente che l'arco e_{h_i} incida sui

[1]Spesso, quando non c'è possibilità di ambiguità, un cammino viene indicato indicando la sequenza dei soli nodi o quella dei soli archi

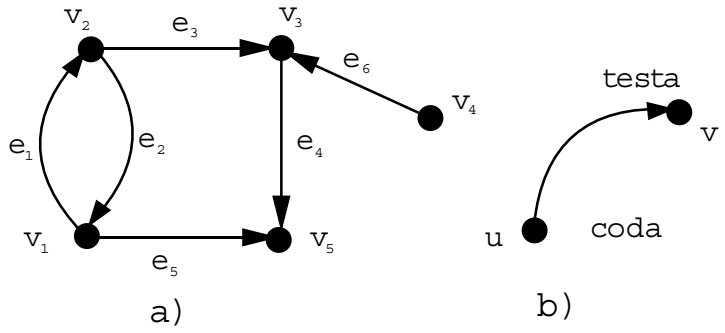


Figura 5.3: a) Grafo orientato b) Arco orientato

nodi $v_{j_{i-1}}$ e v_{j_i}, senza specificare quale dei due nodi sia la testa e quale la coda di e_{h_i}. I nodi v_{j_0} e v_{j_p} sono detti *estremi* del cammino P. Un cammino è detto *semplice* se gli archi e i nodi che lo definiscono sono tutti distinti.

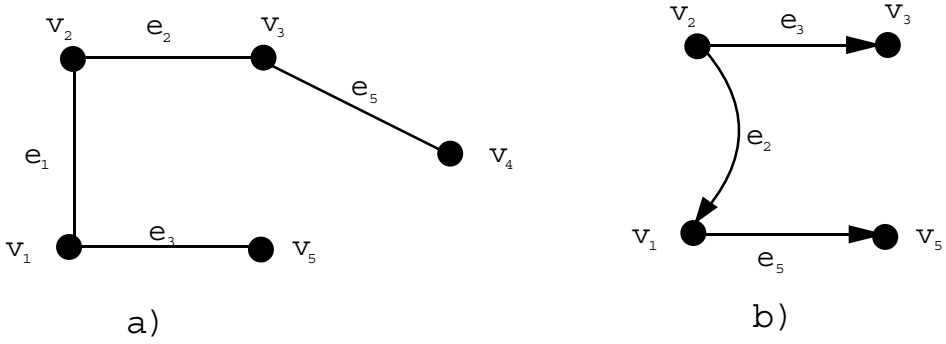


Figura 5.4: Cammini semplici

In figura 5.4a è mostrato un cammino semplice del grafo non orientato di figura 5.1, $P = \{v_5, e_3, v_1, e_1, v_2, e_2, v_3, e_5, v_4\}$, mentre in figura 5.4b è mostrato il cammino semplice del grafo orientato di figura 5.3, $P = \{v_5, e_5, v_1, e_2, v_2, e_3, v_3\}$. In figura 5.5a è mostrato il cammino (non semplice) del grafo non orientato di figura 5.1, $P = \{v_1, e_1, v_2, e_2, v_3, e_5, v_4, e_6, v_5, e_4, v_3, e_5, v_4\}$. Notiamo che se in un grafo esiste un cammino fra i nodi u e v, esisterà un cammino **semplice** fra u e v.

In un grafo orientato, chiameremo *cammino orientato* un cammino $P = \{v_{j_0}, e_{h_1}, v_{j_1}, \ldots, e_{h_p}, v_{j_p}\}$ tale che $e_{h_i} = (v_{j_{i-1}}, v_{j_i})$. Nel camino orientato è dunque importante anche l'orientamento degli archi: il nodo che precede l'arco nel cammino deve esserne la coda, mentre il nodo che succede all'arco deve esserne la testa. In figura 5.5b è mostrato il cammino orientato del grafo orientato di figura 5.3, $P = \{v_1, e_1, v_2, e_3, v_3, e_4, v_5\}$. Se u e v sono, rispettivamente, il primo e l'ultimo nodo di un cammino orientato P, si dirà che P *va* da u a v[2].

[2]Spesso, quando si tratta di grafi orientati e non ci sia possibilità di ambiguità, si indica col nome di cammino quello che più propriamente dovrebbe essere chiamato cammino orientato

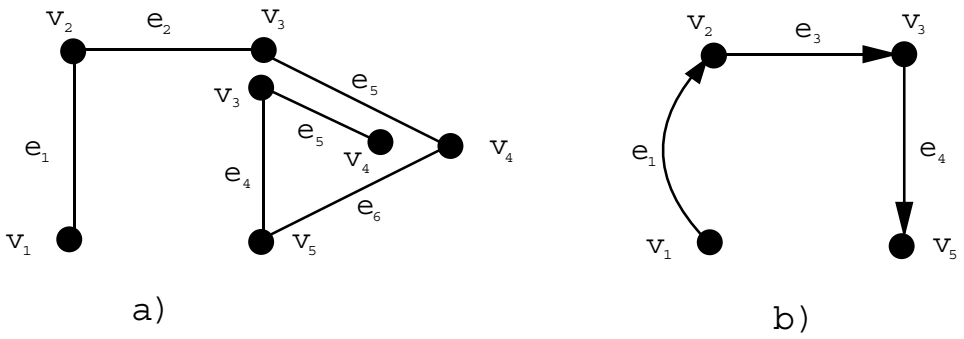


Figura 5.5: a) Cammino (non semplice) b) Cammino orientato

Un cammino è detto *chiuso* se i suoi estremi coincidono. Definiremo *ciclo* un cammino chiuso $C = \{v_{j_0}, e_{h_1}, v_{j_1}, \ldots, v_{h_{p-1}}, e_{h_p}, v_{j_0}\}$, tale che $\{v_{j_0}, e_{h_1}, v_{j_1}, \ldots, e_{h_{p-1}}, v_{h_{p-1}}\}$ è un cammino semplice. Il cammino chiuso $P = \{v_1, e_1, v_2, e_2, v_3, e_3, v_4, e_4, v_2, e_5, v_5, e_6, v_1\}$ di figura 5.6a non è un ciclo perchè il cammino $P = \{v_1, e_1, v_2, e_2, v_3, e_3, v_4, e_4, v_2, e_5, v_5\}$ non è semplice (il nodo v_2 è contenuto due volte). In figura 5.6b è mostrato un ciclo. Notiamo che ancora una volta l'orientamento degli archi nei grafi orientati è indifferente. Per i grafi orientati si definisce *ciclo orientato* un camino orientato chiuso $C = \{v_{j_0}, e_{h_1}, v_{j_1}, \ldots, e_{h_{p-1}}, v_{h_{p-1}}, e_{h_p}, v_{j_0}\}$, tale che $\{v_{j_0}, e_{h_1}, v_{j_1}, \ldots, e_{h_{p-1}}, v_{h_{p-1}}\}$ è un cammino orientato semplice (Fig. 5.6c)

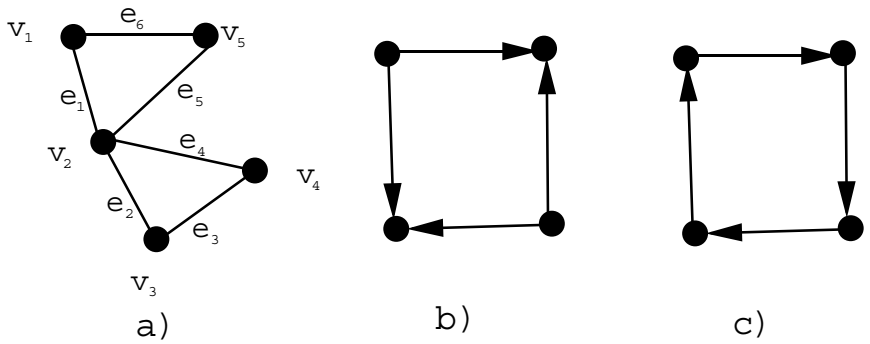


Figura 5.6: a) Cammino chiuso b) Ciclo c) Ciclo orientato

Componenti Connesse. Dato un grafo $G = (V, E)$ (orientato o non orientato), un nodo $v \in V$ si dice connesso a un nodo $u \in V$ se esiste un cammino (non necessariamente orientato) tra v e u. La relazione di connessione è una relazione binaria, ed è facile vedere che gode delle seguenti proprietà:

1. v è connesso a v (RIFLESSIVITÀ).
2. Se v è connesso a u, allora u è connesso a v (SIMMETRIA).
3. Se v è connesso a u, u è connesso a z, allora v è connesso a z (TRANSITIVITÀ).

Quindi la relazione *essere connesso a* è una relazione di equivalenza, che definisce una partizione dell'insieme dei nodi V in classi di equivalenza $C_1, \ldots, C_q$. Si ha dunque $V = C_1 \cup C_2 \cup \ldots \cup C_q$, e $C_i \cap C_j = \emptyset$, per $i \neq j$. Il grafo indotto in G dai nodi apparteneti a una classe d'equivalenza C_i è detto *componente connessa* di G. Un grafo è detto *connesso* se possiede una sola componente connessa. In figura 5.7 è mostrato un grafo non orientato formato da due componenti connesse (notare che $|V| = 6$)[3].

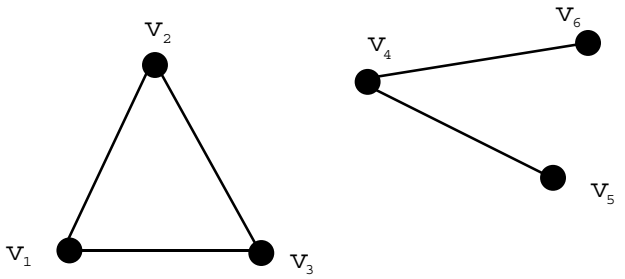


Figura 5.7: Componenti connesse

Dato un grafo orientato $G = (V, E)$, un nodo $v \in V$ è *fortemente connesso* a un nodo $u \in V$ se esistono due cammini orientati in G, uno da v ad u e l'altro da u ad v. Anche in questo caso è facile vedere che la relazione di forte connessione è una relazione d'equivalenza che definisce le classi $C_1, \ldots, C_q$ nell'insieme dei nodi V. Il grafo indotto in G dai nodi appartenenti a una classe C_i è detto *componente fortemente connessa* di G. Un grafo si dice *fortemente connesso* se possiede una sola componente fortemente connessa. In figura 5.8a è mostrato un grafo orientato e in figura 5.8b le sue tre componenti fortemente connesse.

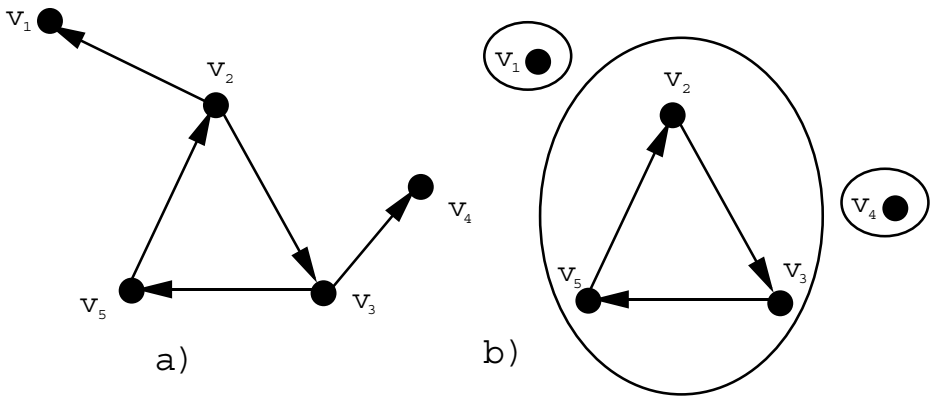


Figura 5.8: a) Grafo orientato b) Componenti fortemente connesse

Alberi. Un grafo (orientato o non orientato) è detto *aciclico* se non contiene cicli (non necessariamente orientati) come sottografi. Si dice *albero* un grafo aciclico e connesso. Ogni grafo aciclico è l'unione di

[3]Con il simbolo $|I|$, dove I è un insieme finito di elementi, si indica *la cardinalità* dell'insieme I, cioè il numero di elementi che compongono l' insieme stesso.

uno o più alberi e viene detto *foresta*. In figura 5.9a è mostrato un albero, in figura 5.9b una foresta.

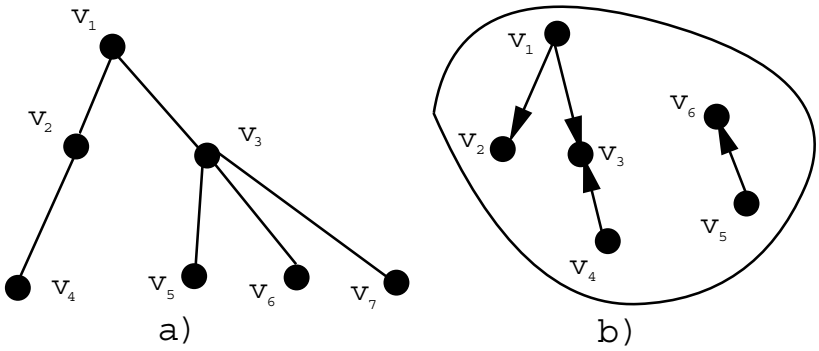


Figura 5.9: a) Albero b) Foresta

Dato un albero $T = (V, E)$, con $|V| \geq 2$, diremo *foglia* un nodo $v \in V$ tale che $|\delta(v)| = 1$ ($|\omega(v)| = 1$). Ad esempio, in figura 5.9a sono foglie v_4, v_5, v_6, v_7.

Valgono i due seguenti risultati, che diamo senza dimostrazione. Notiamo, in particolare, che il secondo lemma fornisce una serie di definizioni alternative di albero.

Proposizione 5.1.1 *Le seguenti asserzioni sono equivalenti:*

1. *$G = (V, E)$ è un albero.*
2. *Ogni coppia di nodi è connessa da un unico cammino.*

Dato un grafo connesso $G = (V, E)$ diremo *albero ricoprente* di G un albero $T = (W, F)$ con $W = V$ e $F \subseteq E$.

Grafi bipartiti Un grafo $G = (V, E)$ non orientato è detto *bipartito* se è possibile trovare una partizione V_1, V_2 dei suoi nodi [4] tale che gli estremi di ogni arco appartengono uno a V_1 e uno a V_2. Nella figura 5.10 è riportato un esempio di grafo bipartito.

Si puo dimostrare che vale la seguente caratterizzazione dei grafi bipartiti

Proposizione 5.1.2 *Un grafo è bipartito se e solo se non contiene cicli di lunghezza dispari.*

Spesso un grafo bipartito viene indicato, mettendo in evidenza la partizione dell'insieme dei nodi, come $G = (V_1, V_2, E)$.

5.2 Rappresentazioni di un grafo

Un grafo G(V,E) pu essere rappresentato in molti modi diversi. Nelle pagine precedenti abbiamo gia incontrato due modi diversi: la rappresentazione estensiva e quella grafica.

Nella rappresentazione estensiva vengono elencati tutti gli elementi dell'insieme V e tutti quelli dell'insieme E, nella rappresentazione grafica, invece, il grafo viene rappresentato mediante un disegno in cui ai nodi si fanno corrispondere piccoli cerchi e agli archi tratti di curva congiungenti i cerchi. Già da

[4] Deve risultare, cioè, $V_1 \cup V_2 = V$ e $V_1 \cap V_2 = \emptyset$

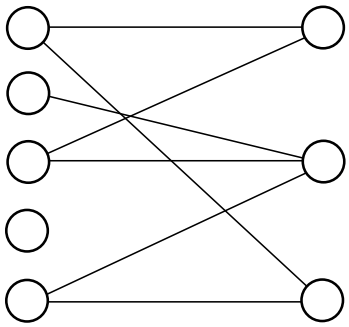


Figura 5.10: Grafo bipartito

questi due esempi si capisce che, benché concettualmente equivalenti, questi modi diversi di rappresentare un grafo possano avere caratteristiche molto differenti. Per esempio, la rappresentazione grafica è molto immediata e ci dà subito una visione intuitiva delle caratteristiche del grafo, e molta della terminologia introdotta finora è ispirata proprio dalla rappresentazione grafica (si pensi, ad esempio, alla definizione di albero e foresta). Tuttavia, se, come spesso accade nella pratica, il grafo ha moltissimi nodi (per esempio migliaia) si capisce subito che la rappresentazione grafica non è più di pratica utilità.

Si capisce allora come sia utile avere a disposizione diversi modi di rappresentare un grafo. La decisione di scegliere una rappresentazione piuttosto che un'altra andrà poi fatta di volta in volta in base al problema che si vuole risolvere, alle dimensioni del grafo, alle sue caratteristiche etc.

Passiamo allora a descrivere tre possibili modi di rappresentare un grafo.

1. matrice di adiacenze;
2. matrice di incidenza;
3. lista di adiacenze.

La prima e la terza modalità sono le più diffuse per la memorizzazione di grafi.

La prima è preferibile nel caso di grafi con un elevato numero di archi rispetto al numero di nodi (per esempio quando $m \approx n^2$) in quanto, pur essendo in questo caso particolare la prima e la terza modalità equivalenti dal punto di vista dell'efficienza computazionale, la rappresentazione in termini di matrice di adiacenze è più immediata e fornisce una rappresentazione del grafo più efficace.

La terza è preferibile nel caso di grafi "sparsi", in cui cioè il numero di archi è piccolo rispetto al numero di nodi (per esempio quando $m \approx n$, come negli alberi).

La seconda, pur se meno efficiente delle altre dal punto di vista computazionale, è preferibile in alcune applicazioni della Ricerca Operativa per la sua corrispondenza ad alcune formulazioni standard di modelli di ottimizzazione e la conseguente maggior facilità di analisi del modello.

Matrice di adiacenze. Questa modalità di rappresentazione è basata su una matrice quadrata $n \times n$. Il generico elemento (i, j) della matrice sarà pari a 1 se l'arco (i, j) del grafo esiste, sarà pari a 0 se l'arco (i, j) non esiste.

In questo modo si possono memorizzare grafi sia orientati che non orientati. Nel caso di grafi non orientati la matrice di adiacenze è simmetrica.

Matrice di incidenza. Questa modalità di rappresentazione è basata su una matrice $n \times m$. La matrice ha quindi un numero di righe pari al numero di nodi e un numero di colonne pari al numero di archi.

Nel caso di grafi non orientati, il generico elemento (i, j) della matrice sarà pari a 1 se il j-esimo arco del grafo incide sul nodo i, sarà pari a 0 se l'arco j-esimo non incide sul nodo i.

Nel caso di grafi orientati, il generico elemento (i, j) della matrice sarà pari a -1 se l'arco j-esimo esce dal nodo i, sarà pari a 1 se l'arco j-esimo entra nel nodo i, sarà pari a 0 se l'arco j-esimo non incide sul nodo i.

Nella matrice di incidenza ogni colonna ha esattamente due elementi diversi da zero; essi sono in corrispondenza delle due righe della matrice relative ai due nodi estremi dell'arco.

Lista di adiacenze. Questa modalità di rappresentazione è basata su una lista di adiacenze del grafo, in cui per ogni nodo del grafo vengono elencati i nodi adiacenti.

Questa rappresentazione è in genere la più efficiente dal punto di vista computazionale, specialmente per grafi con pochi archi.

Completiamo questa sezione con un esempio. Consideriamo il grafo $G(V, E)$ definito nel seguente modo:

$V = \{v_1, v_2, v_3, v_4, v_5, v_6\}$

$E = \{e_1 = (v_1, v_2), e_2 = (v_1, v_3), e_3 = (v_3, v_2), e_4 = (v_2, v_4), e_5 = (v_3, v_5), e_6 = (v_4, v_5), e_7 = (v_4, v_6)\}$

Spesso, nel riportare la rappresentazione estensiva di un grafo, si identificano, per comodità, i vertici con i primi n numeri naturali. In questo caso la rappresentazione estensiva del grafo diventa

$V = \{1, 2, 3, 4, 5, 6\}$

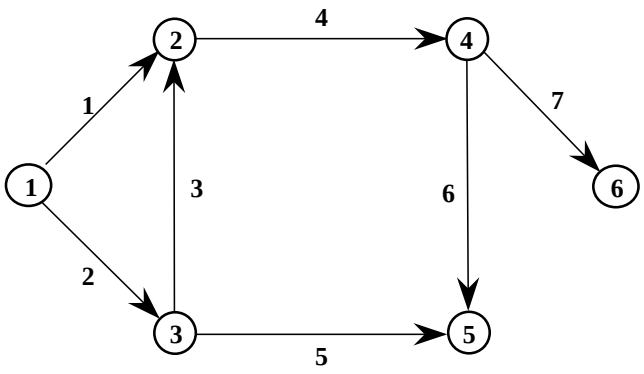
$E = \{(1, 2), (1, 3), (3, 2), (2, 4), (3, 5), (4, 5), (4, 6)\}$,

e spesso ci si limita a fornire il valore di n, cioè il numero di nodi e la lista degli archi. In figura 5.11 sono riportate le rappresentazioni di questo grafo con le varie tecniche appena descritte.

5.3 Alcuni esempi

In questa ultima sezione verranno formulati alcuni problemi utilizzando il formalismo dei grafi. Scopo di questi esempi non è quello di dare una panoramica completa di quei problemi la cui formulazione (e risoluzione) è basata sui grafi (non sarebbe sufficiente un intero corso dedicato a questo tema!). Piuttosto vogliamo mostrare come problemi molto diversi tra loro possano essere formulati (e risolti) usando una rappresentazione basata sui grafi, che spesso contribuisce a rendere chiari problemi apperentemente molto intricati, dando così, inoltre, una prova dell'importanza della scelta di un buon modello nella risoluzione di un problema e della necessità di disporre di un ampio spettro di modelli per poter adeguatamente modellare problemi pratici.

Scambio dei cavalli. Sia data una scacchiera 3×3 in cui le caselle siano numerate con i numeri 1,2,3,4,5,6,7,8,9 come in figura 5.12, due cavalli bianchi nelle caselle 1 e 3, due cavalli neri nelle caselle 7 e 9. Il problema è quello di spostare i due cavalli bianchi al posto di quelli neri e viceversa, spostando un cavallo per volta secondo le modalita delle mosse di scacchi (per esempio il cavallo nella casella 1 in una mossa potrà andare solo nelle caselle 6 e 8, i cavallo nella casella 7 potrà andare nelle caselle 2 e 6, ecc.) e facendo in modo che due cavalli non occupino mai contemporaneamente la stessa casella. Questo problema, apparentemente intricato, diventa semplice non appena viene formulato come problema su grafi. Associamo ad ogni casella della scacchiera un nodo di un grafo. Due nodi sono collegati da un arco se (e solo se) è possibile passare con un cavallo dall'una all'altra delle corrispondenti caselle. Il grafo risultante è quello disegnato in figura 5.12 (da cui ovviamente si osserva come non sia mai possibile passare attraverso la casella 5). Osservando il grafo si individua immediatamente la strategia da seguire per ottenere la soluzione desiderata: basta far circolare i cavalli sul grafo (in questo caso formato da un unico ciclo) in modo da non sovrapporli mai, fino a quando ogni cavallo non ha raggiunto il posto desiderato. Si osservi che il minimo numero di mosse necessario per raggiungere l'obiettivo è pari a 24.



Rappresentazione grafica

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

Matrice di connessione

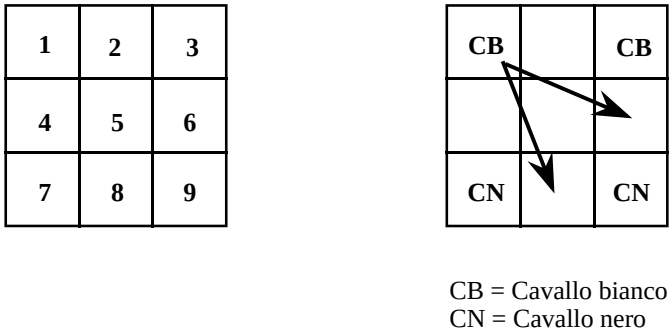
$$\begin{bmatrix} -1 & -1 & 0 & 0 & 0 & 0 & 0 \\ 1 & 0 & 1 & -1 & 0 & 0 & 0 \\ 0 & 1 & -1 & 0 & -1 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & -1 & -1 \\ 0 & 0 & 0 & 0 & 1 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 \end{bmatrix}$$

Matrice di incidenza

1 : 2 , 3	archi da 1 a 2 e da 1 a 3
2 : 4 .	arco da 2 a 4
3 : 2 , 5	archi da 3 a 2 e da 3 a 5
4 : 5 , 6	archi da 4 a 5 e da 4 a 6
5 :	nessun arco
6 :	nessun arco

Lista di adiacenze

Figura 5.11: Rappresentazioni di un grafo



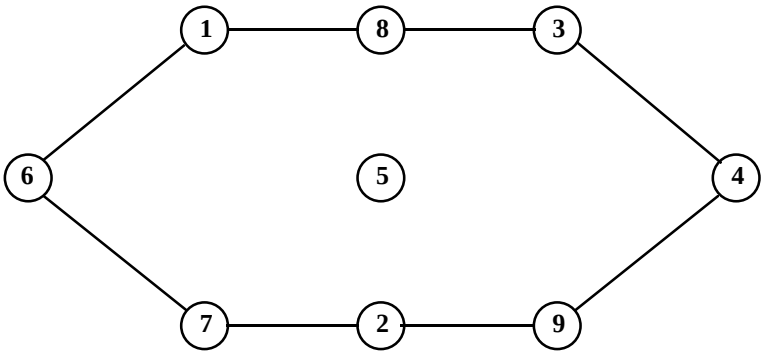


Figura 5.12: Scambio di cavalli

Assegnazione delle aule. In una facoltà universitaria vi sono nello stesso orario n corsi e m aule disponibili. Per ragioni di capienza delle aule e di attrezzature disponibili, ogni corso pu essere tenuto solo in alcune delle aule. Il problema è quello di stabilire quale è il numero massimo di corsi che è possibile tenere nell'orario considerato.

Il problema può essere scritto come problema su grafi nel seguente modo: sia $G(V_1, V_2, E)$ un grafo bipartito in cui l'insieme di nodi V_1 corrisponde ai corsi, l'insieme di nodi V_2 corrisponde alle aule ed esiste un arco fra due nodi i e j (con i appartenente ad V_1 e j appartenente a V_2) se (e solo se) il corso i può essere tenuto nell'aula j. Il problema di stabilire qual è il numero massimo di corsi che è possibile tenere nell'orario considerato può ora essere formulato come il problema di scegliere sul grafo il massimo numero di archi (corrispondenti ad assegnazioni di corsi ad aule) tali che due archi scelti non siano mai adiacenti. Infatti se due archi fossero adiacenti in un nodo dell'insieme V_1, allora vorrebbe dire che uno stesso corso si tiene contemporaneamente in due aule diverse. Se due archi fossero adiacenti in un nodo dell'insieme V_2, allora vorrebbe dire che in una stessa aula si tengono contemporaneamente due corsi. In entrambi i casi le assegnazioni di aule non sono ovviamente ammissibili. Questo problema è detto di "accoppiamento bipartito" e verrà risolto nel capitolo 8.

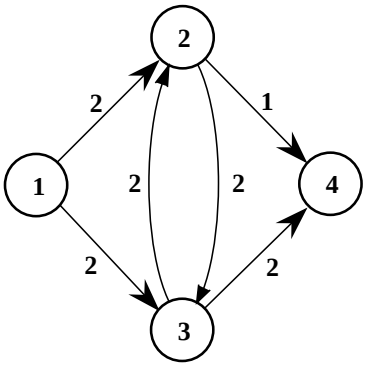
Rete di trasporto con ritardo Sia data una rete di trasporto con 4 nodi e 5 archi, pesata sugli archi [5] con pesi corrispondenti alla quantità trasportabile sull'arco in un singolo viaggio (per esempio 5 tonnellate). Su ogni arco un viaggio richiede una unità di tempo (per esempio un'ora) e in ogni unità di tempo può essere effettuato al più un viaggio su ogni arco. In ogni nodo può essere inoltre immagazzinata merce senza limiti di tempo e di quantità.

La rete è riportata come grafo $G = (V, E)$ nella figura 5.13. Il problema è quello di stabilire se è possibile trasferire una assegnata quantità di merce dal nodo 1 della rete al nodo 4 entro 3 unità di tempo. Per formulare il problema in modo da individuare più facilmente la soluzione è opportuno introdurre un altro grafo (orientato) $H = (V', E')$ (vedi figura 5.13). Nel grafo H, ogni nodo corrisponde ad un nodo del grafo G ad un dato istante di tempo t. Il tempo è in questo caso assunto discreto, per cui t può essere uguale a 0,1, 2 oppure 3. Ogni nodo di H può quindi essere indicato con due indici: uno relativo al nodo di G considerato, l'altro relativo al tempo. Nel grafo H, esiste un arco tra due nodi (per esempio (i, j) e (k, h)) se è possibile inviare in una unità di tempo del materiale da un nodo all'altro. Quindi, nel nostro esempio, perche esista un arco, dovrà essere$h = j + 1$ e (i, k) dovrà appartenere all'insieme di archi A. La capacità di ogni arco di H sarà pari alla capacità del corrispondente arco di G. Per esempio la capacità dell'arco da (i, j) a (k, h) di H sarà pari alla capacità dell'arco (i, k) di G. Inoltre, poiche si suppone che la merce possa restare quanto si vuole in ogni nodo senza limiti di capienza, è necessario introdurre alcuni archi aggiuntivi, di capacità infinita, tra tutte le coppie di nodi (i, j) e (k, h) tali che $i = k$ e$h = j + 1$. Il problema, formulato sul nuovo grafo H, è ora quello di decidere se è possibile inviare un flusso pari alla quantità di merce da trasportare dal nodo $(1, 0)$ al nodo $(4, 3)$. Questo problema può essere facilmente risolto con l'algoritmo del massimo flusso che verrà trattato nel capitolo 8.

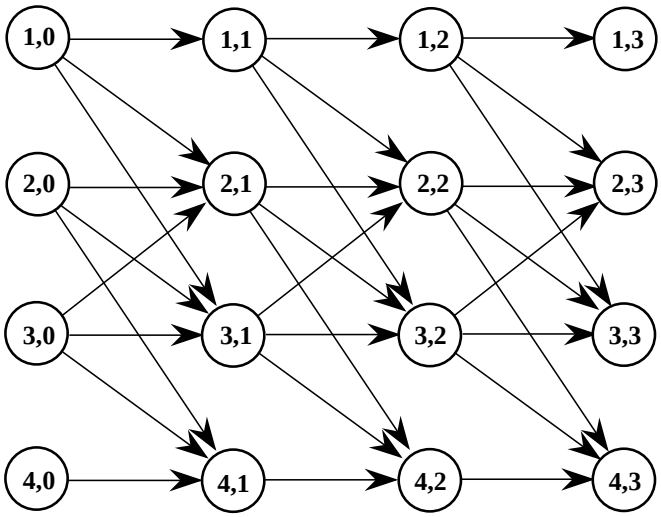
Colorazione di una carta geografica. Il problema è quello di stabilire il numero minimo di colori necessario per colorare una carta geografica, assegnando colori alle varie nazioni rappresentate sulla carta, in modo che due nazioni adiacenti (che quindi abbiano un tratto di frontiera in comune) siano sempre colorate in modo diverso. Si suppone che ogni nazione sia formata da una unica regione connessa e che nazioni adiacenti abbiano una linea di confine in comune di lunghezza maggiore di zero (e non quindi solo un singolo punto in comune). Assegnando ad ogni nazione un nodo di un grafo e collegando due nodi se e solo se le due nazioni corrispondenti sono adiacenti, si ottiene un grafo (che viene detto planare).

La famosa "congettura dei quattro colori", ormai da chiamare "teorema dei quattro colori", afferma che ogni carta geografica è colorabile utilizzando al più quattro colori. Questa congettura era già nota nel 1840. Nel corso di oltre un secolo sono state fornite dimostrazioni per molti casi particolari del problema

[5] questo vuol dire che ad ogni arco è associato un "peso", cioè un numero



Grafo G



Grafo H

Figura 5.13: Rete di trasporto con ritardo

(oltre a molte dimostrazioni della congettura, poi rivelatesi sbagliate o valide per particolari classi di grafi). Solo nel 1976 la congettura è stata dimostrata in tutta la sua generalità usando gli strumenti della teoria dei grafi.

Capitolo 6

Cammini minimi

Come abbiamo accennato nel capitolo precedente, il linguaggio dei grafi permette di rappresentare in modo semplice la struttura di molti problemi applicativi, consentendo, in molti casi di grande importanza, di costruire metodi razionali di soluzione dei problemi stessi.

Fra i problemi più importanti, più semplici e più antichi, per la cui soluzione sono utilizzate rappresentazioni basate su grafi, vi sono i problemi di ricerca di cammini minimi, di cui ci occupiamo in questo capitolo.

I grafi considerati in questo capitolo sono, salvo diversa specificazione, grafi orientati. Nel caso della ricerca di cammini infatti è sempre possibile ricondursi con poca fatica a questo caso. Per fare questo è sufficiente sostituire ogni arco non orientato (e quindi percorribile in entrambi i versi) con due archi diretti in direzione opposta. Se esiste un cammino con le caratteristiche richieste nel grafo originario, allora esiste anche nel grafo trasformato e viceversa.

Per brevità, infine, trattando solo di grafi orientati, *indichiamo con grafo aciclico un grafo che non contenga cicli orientati.*

6.1 Il problema del cammino minimo e alcuni esempi di applicazioni

Dato un grafo orientato $G = (V, E)$, associamo a ciascun arco $e = (u, v) \in E$ un *peso* $p(u, v) \in \mathbb{R}$. Per ogni cammino **orientato** $P = \{v_1, e_1, \ldots, e_{p-1}, v_p\}$, definiamo peso $p(P)$ del cammino P la somma dei pesi degli archi che appartengono a P, e cioè: $p(P) = \sum_{(u,v)\in P} p(u, v)$. Il problema del cammino minimo può essere enunciato nel modo seguente:

dati due nodi $s \in V$ e $t \in V$, trovare un cammino orientato
P^* in G da s a t che abbia peso minimo.

Notiamo che:

- Se non esiste un cammino orientato che vada da s a t in G, il problema non ha soluzioni ammissibili.
- Se esiste un ciclo orientato C in G, tale che $p(C) < 0$ (peso negativo), il problema è illimitato inferiormente.

In genere, dato un nodo s gli algoritmi per il calcolo di cammini minimi determinano non il (o uno dei) cammini minimi da s a un fissato nodo t, ma il cosidetto *albero dei cammini minimi*, cioè un sottografo di G che è un albero i cui nodi includono s e tutti i nodi da esso raggiungibili con un cammino orientato e tale che l'unico (vedi teorema 5.1.1) cammino da s a ogni altro nodo t dell'albero sia un cammino minimo da s a t. A prima vista questa può sembrare una complicazione non necessaria, ma in effetti, per come

sono organizzati gli algoritmi, vedremo che non è così. Quindi, in pratica, gli algoritmi che studieremo risolvono il seguente problema:

> dato un nodo $s \in V$, trovare un albero dei cammini minimi
> da s a ogni nodo in V raggiungibile da s.

Nella letteratura esistono moltissimi algoritmi per il calcolo dei cammini minimi. Ognuno di essi ha le sue peculiarità e le sue limitazioni. Per esempio alcuni algoritmi funzionano solo su determinate classi di grafi (grafi aciclici, grafi con pesi non negativi, etc.), inoltre ci possono essere notevoli differenze nella efficienza degli algoritmi. In generale un algoritmo che funziona su una classe ampia di grafi sarà più complesso di un algoritmo studiato per una classe ristretta di grafi, tenendo conto delle peculiarità di quella classe.

In questo capitolo consideremo due algoritmi: il primo calcola i cammini minimi su grafi aciclici, il secondo calcola i cammini minimi su grafi con pesi non negativi.

Nella parte rimanente di questa sezione illustriamo alcuni semplici esempi di calcolo dei cammini minimi. Alla fine del capitolo considereremo due casi più complessi e interessanti.

6.1.1 Percorso di tempo minimo su una rete stradale

Dato un grafo pesato che rappresenta la rete stradale italiana in cui i pesi degli archi indicano il (valore atteso del) tempo di percorrenza dell'arco, il problema è quello di trovare il cammino che congiunge due particolari nodi del grafo (nodo di partenza e nodo di arrivo) con tempo di percorrenza minimo. Si noti che possono esistere piu cammini con la caratteristica di essere minimi.

Questo esempio, apparentemente semplice, solleva complessi problemi di modellistica. Un primo problema è relativo al livello di dettaglio necessario nella rappresentazione. Se, per esempio, il percorso da effettuare parte da Milano e arriva a Brindisi, il grafo dovrà contenere solo le autostrade e le principali strade di collegamento fra città diverse e un'intera città potrà essere rappresentata con un nodo del grafo. Un eccesso di dettaglio appesantirebbe inutilmente la rappresentazione, rendendo ogni algoritmo di soluzione lento e inefficace. Se, d'altra parte, il percorso da effettuare parte da piazza San Pietro a Roma e arriva a un indirizzo di Frascati (a circa 20 km. da Roma), la rappresentazione davrà essere completamente diversa, non solo perché il grafo sarà diverso, ma anche perché il peso degli archi, per essere significativo, dovrà prendere in considerazione i problemi del traffico urbano e quindi essere funzione dell'ora ed eventualmente del giorno. Infatti, effettuare tale percorso alle 10 di mattina di un giorno feriale non sarà ovviamente la stessa cosa che effettuarlo alle 4 del mattino (magari di un giorno festivo).

Un secondo problema è relativo al tipo di obiettivi che ci si propone. Infatti, una rappresentazione può essere o può non essere adeguata, a seconda del motivo per cui si vuole conoscere il cammino di tempo minimo. Per esempio, per alcune applicazioni critiche (autombulanze, vigili del fuoco, polizia), non basta l'informazione sul valore atteso del tempo di transito, ma serve anche valutare la varianza di tale tempo, ossia le possibili variazioni rispetto alla media. È meglio infatti utilizzare un percorso un po più lungo ma con un tempo di percorrenza prevedibile con relativa certezza, piuttosto che un percorso mediamente piu breve ma per cui vi sia il rischio di restare imbottigliati nel traffico.

Un terzo problema è relativo alla quantità di informazioni che è necessario inserire nell'elaboratore per affrontare il problema. Per calcolare il cammino da Milano a Brindisi è necessario inserire l'intera carta stradale italiana, o basta una porzione? Da un lato, più informazioni vengono inserite maggiore è il tempo necessario per inserirle e il costo dell'operazione; d'altro canto se vengono calcolati spesso percorsi di tempo minimo sulla rete stradale italiana forse conviene memorizzare tutto in modo organico una volta per tutte.

6.1.2 Costruzione di una autostrada

Il problema considerato è quello di costruire al costo minimo una autostrada fra le citta A e B. Nel grafo riportato in figura 6.1 i nodi rappresentano i punti per cui l'autostrada puo passare e gli archi i

possibili collegamenti fra punti. Il peso degli archi rappresenta il costo di costruzione della relativa tratta di autostrada.

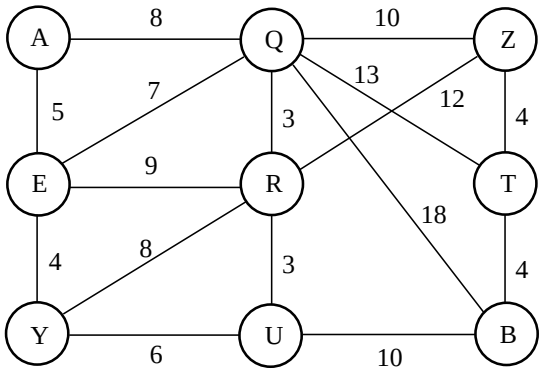


Figura 6.1: Cammino minimo tra A e B

Si noti che in questo caso il grafo è non orientato in quanto ogni arco puo essere percorso in entrambi i sensi; per ricondursi al caso orientato basta sostituire ogni arco con due archi orientati in senso opposto, a ognuno dei quali viene attribuito un peso pari al peso dell'arco eliminato.

La scelta di una autostrada tra A e B con costo complessivo di costruzione minimo, corrisponde alla scelta del cammino di minimo costo dal nodo A al nodo B sul grafo. Nella figura 6.1 tale cammino è indicato in grassetto e il costo complessivo è pari a 24.

6.2 Cammini minimi e massimi su grafi aciclici

Una classe di grafi orientati di particolare interesse in campo applicativo (vedi gli esempi alla fine del capitolo) è la classe di grafi aciclici. La ricerca di cammini minimi o massimi su tali grafi è uno strumento di progetto di notevole importanza. Come si vedrà nel seguito, i due problemi di minimo e di massimo sono, in questo caso particolare, risolti da due algoritmi identici eccetto che per la sostituzione di un massimo a un minimo nella formula ricorsiva alla base del procedimento. Nel seguito verrà prima considerato il caso di un problema di cammino minimo. Per poter procedere con la descrizione dell'algoritmo, è necessario studiare prima una particolare tecnica di numerazione dei nodi di un grafo aciclico.

6.2.1 Numerazione topologica dei nodi di un grafo

Una caratteristica peculiare dei grafi aciclici (sia $G = (V, E)$ il grafo, con$|V| = n$ e $|E| = m$) consiste nella possibilità di numerare i nodi del grafo con i numeri 1,2,3,....,n-1,n in modo tale che:

$$\text{se esiste un arco dal nodo } i \text{ al nodo } j \text{ allora } j > i$$

Tale numerazione viene detta numerazione topologica dei nodi del grafo e non è in generale unica.

Non tutti i grafi possono essere numerati topologicamente. In effetti l'esistenza di una numerazione topologica dei nodi di un grafo caratterizza esattamente la classe dei grafi aciclici. Vale infatti il seguente teorema.

Teorema 6.2.1 *Un grafo è aciclico se e solo se esiste una numerazione topologica dei suoi nodi.*

Dimostrazione.
Sufficienza. Supponiamo che esista una numerazione topologica dei nodi e facciamo vedere che l' esistenza di un ciclo porterebbe ad una contraddizione. Possiamo assumere che i nodi siano numerati topologicamente, indicheremo l'i-esimo nodo di questa particolare numerazione, come v_i. Se esiste un ciclo (orientato) vuol dire che esiste una successione di nodi $(v_i, v_j, v_k, \ldots, v_r, v_s)$ tali che

- esistono gli archi (v_i, v_j), (v_j, v_k), ..., (v_r, v_s);

- $v_i = v_s$.

Ma allora, da una parte, siccome la numerazione è topologica abbiamo $i < j < k < \ldots < r < s$, cioè $i < s$, mentre dall'altra, poiché $v_i = v_s$ abbiamo $i = s$. Questa è una contraddizione e così il grafo deve essere aciclico.
Necessità. Supponiamo che il grafo sia aciclico e mostriamo che deve esistere almeno una numerazione topologica. La dimostrazione è costruttiva, faremo cioè vedere che esiste una numerazione topologica costruendone una.

Come primo passo osserviamo che se il grafo è aciclico, deve esistere almeno un nodo che non abbia archi entranti. Infatti, se ciò non fosse vero, potremmo ragionare nel modo seguente. Prendiamo un nodo qualunque, chiamiamolo v_1. Siccome tutti i nodi hanno archi entranti esiste un nodo predecessore di v_1, indichiamolo con v_2; notiamo che per come abbiamo scelto v_2 esiste l'arco (v_2, v_1) (attenzione, questa è una numerazione non topologica). Possiamo ripetere il ragionamento con v_2 e trovare un nodo v_3 tale che esista l' arco (v_3, v_2). Siccome stiamo supponendo, per assurdo, che tutti gli archi abbiano degli archi entranti, possiamo ripetere il ragionamento quante volte vogliamo. Notiamo che ogni nodo generato deve essere diverso dai precedenti, altrimenti avremmo trovato un ciclo, contraddicendo l'aciclicità del grafo. D'altra parte, arrivati a v_n i nodi del grafo sono "finiti" e quindi il predecessore di v_n che stiamo supponendo esitente per assurdo, deve per forza essere uno dei nodi già esaminati. Così si viene a formare un ciclo.

Quindi dato un grafo aciclico deve per forza esistere almeno un nodo che non ha archi entranti. Prendiamo uno di questi nodi e numeriamolo con il numero 1. Eliminiamo dal grafo il nodo 1 e tutti gli archi uscenti da esso. Il nuovo grafo che otteniamo è ovviamente ancora un grafo aciclico. Quindi per lo stesso ragionamento fatto prima deve esistere almeno un nodo che non ha archi entranti. Prendiamo uno di questi nodi e numeriamolo con il numero 2. Notiamo che ovviamete se consideriamo il grafo originario il nodo 2 può avere archi entranti, ma solo provenienti dal nodo 1 e quindi la condizione $i < j$ è rispettata. Possiamo ora ripetere il procedimento n volte (quanti sono i nodi) ed ottenere così una numerazione topologica del grafo. □

Il precedente teorema è importante, anche perché nella dimostrazione della necessità è sostanzialmente dato un algoritmo per numerare topologicamente i nodi di un grafo.

Riesponiamo qui l'algoritmo per chiarezza.

- Siccome il grafo è aciclico, deve esistere almeno un nodo con solo archi uscenti;

- individuiamo uno di questi nodi e attribuiamogli il numero 1;

- cancelliamo il nodo numerato e tutti gli archi adiacenti, nel nuovo grafo ridotto individuiamo un nodo con soli archi uscenti e attribuiamogli il numero 2, e così via fino ad aver numerato tutti i nodi

La correttezza e validità di questa procedura è stata provata nella dimostrazione della necessità del Teorema 6.2.1.

Notiamo che se ad un certo punto dell'applicazione della procedura non possiamo procedere, se succede cioè che ad un determinato passo non riusciamo a trovare un nodo senza archi entranti, questo vuol dire che il grafo considerato contiene un ciclo.

Quindi la procedura per la numerazione topologica di un grafo può anche essere utilizzata per determinare se un grafo è aciclico o meno.

Come esempio consideriamo il grafo di figura 6.2.

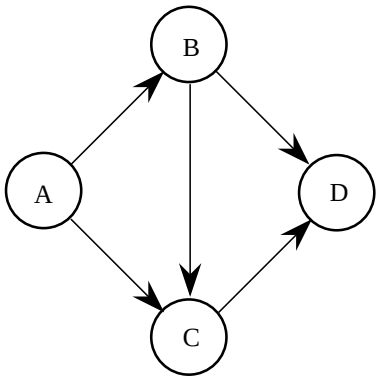


Figura 6.2: Numerazione dei nodi di un grafo

Si tratta di decidere se il grafo è aciclico e, in caso, numerare i nodi topologicamente. I vari passi dell'algoritmo sono riportati qui di seguito

Passo	Nodo senza archi entranti	Nodi non ancora numerati	Numerazione
1	A	B,C,D	A=1
2	B	C,D	B=2
3	C	D	C=3
4	D	$\emptyset$	D=4

Il procedimento è terminato con la numerazione di tutti i nodi. Il grafo è dunque aciclico e la numerazione trovata è topologica.

Supponiamo ora che nel precedente grafo l'arco (B, D) sia orientato da D a B. Ovviamente si verrebbe a creare un ciclo. Se proviamo ad applicare la procedura per la determinazione di una numerazione topologica possiamo iniziare numerando il nodo A come nodo 1. Ma dopo non possiamo più procedere perché eliminato il nodo A e gli archi da esso uscenti ((A,B) e (A,C)) non è più possibile individuare un nodo che non abbia archi entranti.

6.2.2 Un algoritmo per il cammino minimo su grafi aciclici

La numerazione dei nodi di un grafo aciclico descritta nella sezione precedente consente di costruire un algoritmo di soluzione per il problema di cammino minimo particolarmente semplice. Infatti, nella ricerca di un cammino tra una qualsiasi coppia di nodi i e j del grafo, a causa della numerazione attribuita ai nodi, si può affermare che:

se $j < i$ allora non esistono cammini da i a j;

se $j > i$ allora gli unici nodi che è necessario considerare nella ricerca del cammino da i a j sono i nodi con indice k tale che $i < k < j$.

Infatti, se il cammino passasse per un nodo $h > j$, allora non potrebbe tornare su j, a causa della mancanza di archi che collegano nodi con indice maggiore a nodi con indice minore; se passasse per un nodo $h < i$, allora dovrebbe esistere un cammino da h a i, il che comporterebbe l'esistenza di archi che collegano nodi con indice maggiore a nodi con indice minore.

Sulla base di queste considerazioni, è possibile impostare un algoritmo per il calcolo dell'albero dei cammini minimi tra un nodo del grafo (per esempio il nodo 1) e tutti i nodi con indice superiore (per quelli con indice inferiore non esiste sicuramente un cammino; ovviamente, se il nodo di partenza è quello contrassegnato con l'indice 1, allora si tratta di calcolare l'albero dei cammini minimi tra il nodo 1 e tutti gli altri). L'algoritmo per il calcolo dell'albero dei cammini minimi dal nodo 1 a tutti gli altri nodi si basa sul calcolo in sequenza dei cammini minimi dal nodo 1 al nodo 2, dal nodo 1 al nodo 3, dal nodo 1 al nodo 4, e cosí via. Indichiamo con:

- $p(i,j)$ il peso dell'arco (i,j) che parte dal nodo i e arriva al nodo j;
- $f(i)$ il valore del cammino minimo dal nodo 1 al nodo i;
- $J(i)$ il nodo che precede i su tale cammino (nel caso il cammino minimo non sia unico, allora se ne sceglie uno qualsiasi fra quelli minimi).

Si noti che dai valori $J(i)$ è possibile ricostruire in modo immediato l'albero (o uno dei possibili alberi) dei cammini minimi.

Possiamo allora illustrare l'algoritmo per il calcolo dei percorsi minimi.

- $f(1) := 0$; $J(1) := 1$;
- per $j = 2, 3, 4, \ldots, n-1, n$ ripeti la seguente serie di operazioni

  $f(j) := \min_{(i,j)\in\omega^-(j)}\{f(i) + p(i,j)\}$;
  $J(j) :=$ valore di i per cui si è verificato il minimo;

Si noti che una volta assegnato un peso f(i) a un nodo (cioè un valore del cammino minimo dal nodo 1 al nodo considerato), tale peso non viene più modificato nel corso dell'algoritmo, ma indica in modo definitivo il valore del cammino. Questo è dovuto al fatto, già citato, che tutti i nodi successivi non devono essere considerati per il calcolo del percorso dal nodo 1 al nodo i.

Come esempio consideriamo il grafo di figura 6.3.

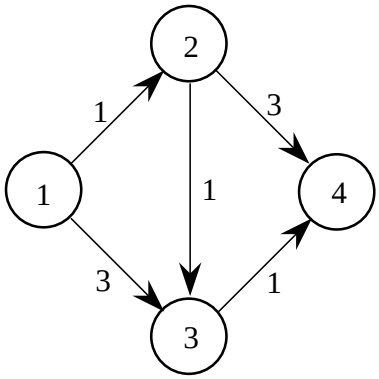


Figura 6.3: Percorso minimo su grafi aciclici

Si tratta di determinare l'albero dei cammini minimi tra il nodo 1 e tutti gli altri nodi. I vari passi dell'algoritmo (corrispondenti alla successione di nodi visitati, si osservi che in questo caso particolare l'indice del passo coincide con l'indice del nodo visitato) portano alla seguente successione di valori $f(i)$ e $J(i)$

Passo	Valore di f(i)	Valore di J(i)
1	$f(1) = 0$	$J(1) = 1$
2	$f(2) = 1$	$J(2) = 1$
3	$f(3) = \min\{3, 1+1\} = 2$	$J(3) = 2$
4	$f(4) = \min\{1+3, 2+1\} = 3$	$J(4) = 3$

Il corrispondente albero dei cammini minimi è riportato in figura 6.4

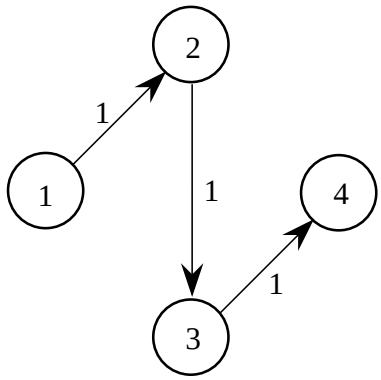


Figura 6.4: Albero dei cammini minimi

Esempio 6.2.2 *Sia dato il grafo di Figura 6.5. Determinare l'albero dei cammini minimi, utilizzando*

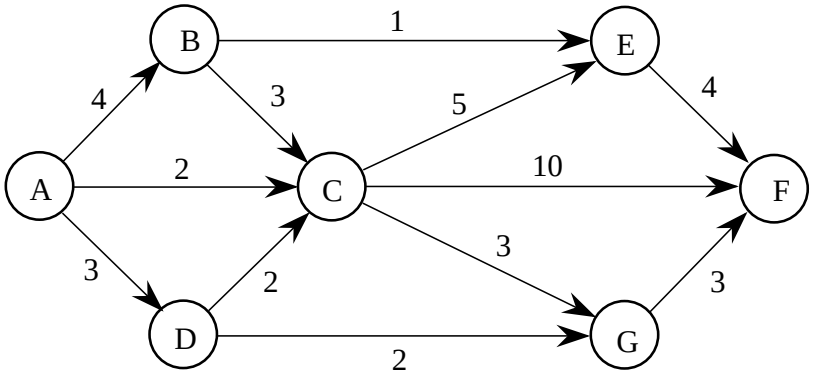


Figura 6.5: Grafo Esercizio 6.2.2

l'algoritmo per grafi aciclici.

Soluzione. Si deve prima numerare topologicamente il grafo. I passi sono riportati nella seguente tabella ed il grafo risultante in Figura 6.6

Passo	Nodo senza archi entranti	Nodi non ancora numerati	Numerazione
1	A	B,C,D,E,F,G	A=1
2	B o D	B,C,E,F,G	D =2
3	B	C,E,F,G	B=3
4	C	E,F,G	C=4
5	E o G	F,G	E=5
6	G	F	G=6
7	F	$\emptyset$	F=7

Osserviamo che la numerazione topologica in questo esempio non é unica; infatti ai passi 2 e 5 potevamo scegliere tra due nodi. Apllichiamo ora l'algortimo. I passi sono riportati nella seguente tabella

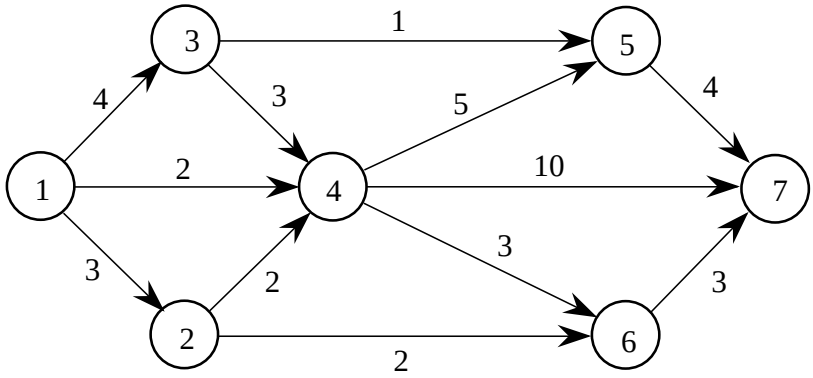


Figura 6.6: Numerazione topologica del grafo di Figura 6.5

Passo	Valore di f(i)	Valore di J(i)
1	$f(1) = 0$	$J(1) = 1$
2	$f(2) = 3$	$J(2) = 1$
3	$f(3) = 4$	$J(3) = 1$
4	$f(4) = \min\{0+2, 3+2, 4+3\} = 2$	$J(4) = 1$
5	$f(5) = \min\{4+1, 2+5\} = 5$	$J(5) = 3$
4	$f(6) = \min\{3+2, 2+3\} = 5$	$J(6) = 4$
4	$f(7) = \min\{2+10, 5+4, 5+3\} = 8$	$J(7) = 6$

Il corrispondente albero dei cammini minimi è riportato in figura 6.7.

6.2.3 Un algoritmo per il cammino massimo su grafi aciclici

Se il problema di ottimo è quello della determinazione del cammino di peso massimo sul grafo, allora è facile convincersi che basta sostituire nella formula ricorsiva al min un max e tutte le considerazioni fatta continuano, in questo caso particolare di grafi aciclici, a essere valide. Oltre che da considerazioni dirette questo risultato può essere dedotto coniderando che il problema di trovare un cammino di peso massimo su un grafo aciclico è equivalente a quello di trovare un cammino di peso minimo sullo stesso grafo dove i pesi, però, sono stati cambiati di segno.

L'algoritmo per il calcolo dei cammini massimi su grafici aciclici è allora il seguente

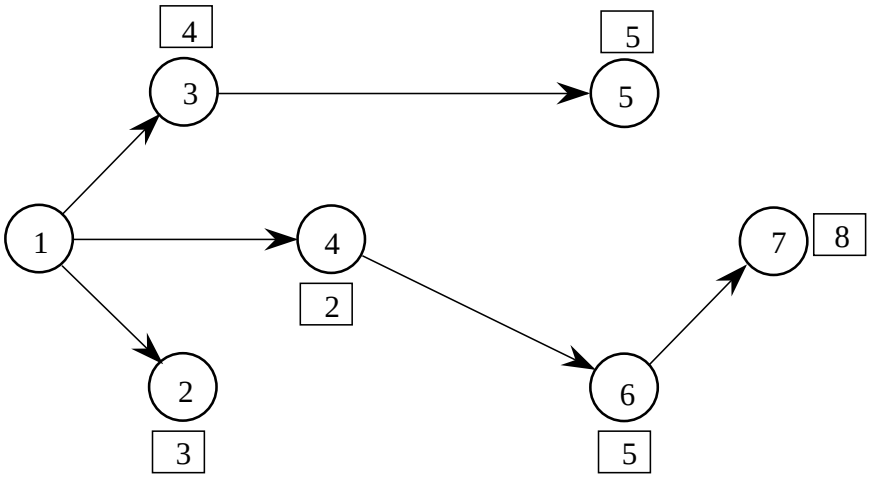


Figura 6.7: Albero dei cammini minimi

- $f(1) := 0$; $J(1) := 1$;
- per $j = 2, 3, 4, \ldots, n-1, n$ ripeti la seguente serie di operazioni

  $f(j) := \max_{(i,j)\in\omega^-(j)}\{f(i) + p(i,j)\}$;

  $J(i) :=$ valore di i per cui si è verificato il massimo;

Come esempio consideriamo sempre il grafo di figura 6.3 Si tratta di determinare l'albero dei cammini massimi tra il nodo 1 e tutti gli altri nodi. I vari passi dell'algoritmo portano alla seguente successione di valori $f(i)$ e $J(i)$

Passo	Valore di f(i)	Valore di J(i)
1	$f(1) = 0$	$J(1) = 1$
2	$f(2) = 1$	$J(2) = 1$
3	$f(3) = \max\{3, 1+1\} = 3$	$J(3) = 1$
4	$f(4) = \max\{1+3, 3+1\} = 4$	$J(4) = 2$ (oppure $J(4) = 3$)

6.3 Cammini minimi su grafi con pesi positivi: algoritmo di Dijkstra

L'algoritmo di Dijkstra permette di risolvere il problema del cammino minimo fra due nodi qualora tutti i pesi degli archi siano **non negativi**. Più precisamente, l'algoritmo calcola il peso del cammino minimo da un nodo s a tutti gli altri nodi del grafo, costruendo conteporaneamente l'albero dei cammini minimi. Siccome in questo caso la numerazione non gioca nessun ruolo, in questo paragrafo supponiamo, senza perdita di generalità, che s sia sempre uguale a 1.

Notiamo la differenza con il caso esaminato nel paragrafo precedente: nel caso precedente non c'era nessuna restrizione sui pesi, ma c'era una restrizione sulla *topologia* del grafo, che non doveva contenere cicli orientati. Nel caso esaminato in questo paragrafo, non c'è nessuna restrizione sulla topologia del grafo (che può essere qualunque e contenere, dunque, cicli orientati), ma c'è una restrizione sui pesi, che devono essere non negativi.

È evidente che il caso analizzato in questo paragrafo è di interesse in quanto, per esempio, in tutti i casi in cui la ricerca di cammini minimi corrisponde alla ricerca di un reale percorso in, per esempio, una città, esisteranno cicli, ma i pesi, che corrispondono a distanze fisiche, sono ovviamente positivi.

Un'altra differenza che vogliamo segnalare subito è che in questo caso, non è possibile dare una semplice variante dell'algoritmo che calcoli i cammini di peso massimo in quanto, se facciamo diventare il problema di minimo un problema di massimo cambiando i segni dei pesi, il grafo che otteniamo ha i pesi tutti *non positivi*, e quindi l'algoritmo non è più utilizzabile.

L'algoritmo per il calcolo dei cammini minimi su grafi aciclici si basa fortemente sul fatto che i nodi del grafo siano numerati topologicamente. Tenendo conto del fatto che il cammini minimo tra il nodo i e il nodo j (con $j > i$), se esiste, può passare solo per i nodi k, con k compreso tra i e j, abbiamo sviluppato una semplice procedure iterativa. Nel caso di grafi con pesi non negativi vogliamo, in qualche modo, ancora cercare di sviluppare un algoritmo che abbia le stesse caratteristiche. Ovviamente, non disponendo più di una numerazione topologica dobbiamo ragionare in maniera diversa. Vediamo su un esempio come possiamo ragionare.

Consideriamo il grafo in figura 6.8 e proponiamoci di trovare i cammini minimi dal nodo 1 a tutti i nodi da esso raggiungibili. Analogamente al caso dei grafi aciclici, poniamo $f(1) = 0$ e $J(1) = 1$. Vogliamo anche in questo caso arrivare ad associare ad ogni nodo i del grafo due etichette, $f(i)$ e $J(i)$ che diano rispettivamente, la distanza dal nodo 1 e il predecessore di i su un cammino minimo che va da 1 a i.

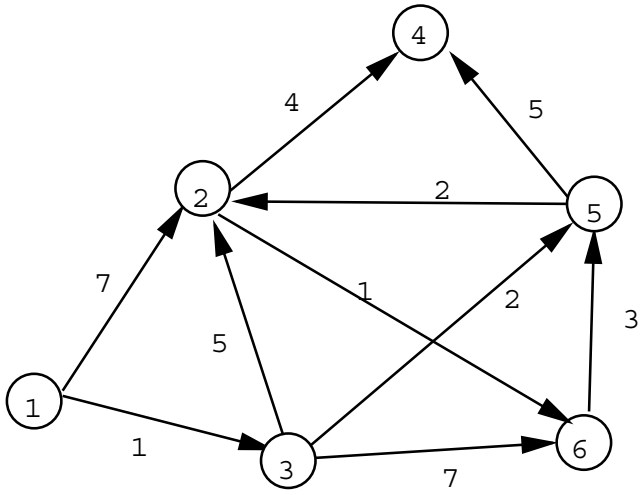


Figura 6.8: Grafo con pesi non negativi

Consideriamo ora i nodi raggiungibili da 1 con un solo arco: sono 2 e 3. Notiamo che il peso dell'arco $(1, 3)$, che è uguale a 1, è minore del peso dell'arco $(1, 2)$, che è uguale a 7. Possiamo allora porre $f(3) = 1$ e $J(3) = 1$. Infatti, supponiamo per assurdo che esista un cammino minimo per andare da 1 a 3 diverso da quello fornito dall'arco (1,3) e con un peso più piccolo. Questo cammino dovrebbe prima "uscire" da 1, passando quindi per un arco di peso 1 o uno di peso 7, quindi dovrebbe "tornare" a 3. Ma siccome gli archi hanno tutti pesi non negativi, il peso di questa seconda parte di cammino si va ad aggiungere a quello dell'arco usato per "uscire" da 1. E' quindi ovvio che, essendo il peso dell'arco (1,3) il più piccolo tra i pesi degli archi uscenti da 1, il peso di questo altro ipotetico cammino, deve essere almeno di 1. Quindi le etichette assegnate al nodo 3 sono corrette.

Possiamo ora ripeter qeusto ragionamento. Consideriamo i nodi raggiungibili da 1 e 3 (i nodi già

etichettati). Sono 2, 5, 6. Le distanze minime da 1, con la restrizione di passare solo per 1 e 3, sono:

- per il nodo 2: 6 (distanza data dal cammino che passa per i nodi 1, 3, 2);
- per il nodo 5: 3 (distanza data dal cammino che passa per i nodi 1, 3, 5);
- per il nodo 6: 8 (distanza data dal cammino che passa per i nodi 1,3,6).

Notiamo che per andare al 2 due passando solo per i nodi 1 e 3 (già etichettati) esiste anche dato dall'arco (1,2), ma questi ha un peso 7 superiore a quello che si ottiene passando per 3. Osserviamo anche che il nodo scelto *è, tra i nodi ancora non etichettati, quello che ha la distanza distanza minima da 1 se ci limitiamo a considerare solo i cammini che passano per i nodi già etichettati.* A questo punto possiamo ragionare in modo simile a quello adottato prima, anche se la situazione è leggermente più complessa. Consideriamo il nodo 5 (quello che ha la distanza più piccola, tra quelli raggiungibili dai nodi 1 e 3), e poniamo $f(5) = 3$ e $J(5) = 3$. Queste sono etichette corrette. Supponiamo infatti che esista un altro cammino, C, da 1 a 5, diverso da quello trovato (e dato dai nodi 1,3 e 5) e con un peso minore di 3. Questo cammino C deve passare per almeno un nodo diverso da 1 e 3 e 5 per l'osservazione in corsivo fatta poche righe sopra. Ora, sia j il primo nodo diverso da 1, 3 e 5 nel cammino C. Il peso del cammino da 1 a j deve essere di almeno 3, sempre per l'osservazione in corsivo fatta prima. Siccome la parte del cammino che va da j a 5 ha un valore non negativo (per l'ipotesi che i pesi siano tutti non negativi) abbiamo di nuovo un assurdo.

A questo punto la tecnica da adottare dovrebbe essere chiara. A un generico passo dell'algoritmo possiamo supporre di avere un insieme di nodi, diciamo S, che hanno già le loro etichette f e J correttamente assegnate. Si sceglie il nuovo nodo da mettere in S come il nodo (o uno dei nodi) che ha la distanza minima da 1 con il vincolo di passare solo per nodi di S. Ad ogni passo aggiungiamo un nodo in S e quindi in un numero di passi uguale al numero dei nodi, l'algoritmo termina con le etichette correttamente assegnate.

Qui di seguito diamo una descrizione più dettagliata (e più vicina a una possibile implementazione sul calcolatore) dell'algoritmo delineato. In questa descrizione indichiamo con S i nodi a cui sono state assegnate le etichette corrette f e J, e con T tutti gli altri nodi. A differenza di quanto visto finora, però, noi diamo delle *etichette provvisorie,* che denominiamo sempre, per brevità, f e J, anche ai nodi in T. Se i appartiene a T, $f(i)$ rappresenta le distanze minima del nodo i dal nodo 1, con il vincolo di passare solo per nodi di S, e $J(i)$ è il corrispondente predecessore lungo il cammino minimo. Se non esiste nessun cammino da 1 a i che passa solo per nodi in S, poniamo $f(i) = +\infty$ e $J(i)$ è lasciato indefinito. Queste etichette hanno l'unico scopo di facilitare, ad ogni passo, la scelta del nodo in T da mettere in S. Infatti è chiaro che basterà scegliere, di volta in volta, il nodo in T con il valore di f più piccolo. Ovviamente, ad ogni iterazione, cambiando i nodi in S, le etichette provvisori dovranno essere aggiornate. Questo viene fatto nel punto (c) dell'algoritmo, la cui logica dovrebbe essere chiara, e che verrà ulteriormente chiarito dall'esempio che faremo subito dopo la descrizione dell'algoritmo. L'algoritmo termina quando o tutti i nodi del grafo sono in S o quando tutti i nodi in T hanno il valore di f uguale a $+\infty$, fatto che ovviamente indica che i nodi in T non sono raggiungibili da 1.

Agoritmo di Dijkstra

(a) *Inizializzazione.*

Poni $S \leftarrow \{1\}$, $T \leftarrow \{2, \ldots, n\}$. $f(1) = 0$, $J(1) = 1$.

Poni $f(i) = p(1, i)$, $J(i) = 1$, per $(1, i) \in \omega^+(1)$.

Poni $f(i) = +\infty$, per $(1, i) \notin \omega^+(1)$.

(b) *Assegnazione etichetta permanente*

Trova $j \in T$ tale che $f(j) = \min_{i \in T} f(i)$.

Poni $T = T - \{j\}$, $S = S \cup \{j\}$.

Se $T = \emptyset$ o $f(i) = +\infty, \forall i \in T$ **STOP** (*terminazione dell'algoritmo*).

(c) *Assegnazione etichetta provvisoria*

Per ogni $(j, i) \in T \cap \omega^+(j)$ tale che $f(i) > f(j) + p(j, i)$ *Poni*:

c.1 $f(i) = f(j) + p(j, i)$

c.2 $J(i) = j$

Vai al passo (b).

Vediamo, nel caso del grafo di figura 6.8, come l'algoritmo proceda (per semplicità, l'evoluzione di T è omessa essendo $T = V - S$).

Iterazione 0 Inizializzazione.

(a) $S = \{1\}$. $d(1) = 0$. $d(2) = 7$. $d(3) = 1$. $d(4) = +\infty$. $d(5) = +\infty$. $d(6) = +\infty$.
$J(2) = 1$. $J(3) = 1$. $J(4) = 1$. $J(5) = 1$. $J(6) = 1$.

Iterazione 1.

(b) $j = 3$. $S = \{1, 3\}$.

(c.1) $(\omega^+(j) \cap T = \{(3, 2), (3, 5), (3, 6)\}$. E' facile vedere che per ognuno dei nodi $\{2, 5, 6\}$ è verificata la condizione $d(i) > d(j) + p(j, i)$, e quindi le etichette vanno tutte aggiornate. $d(2) = d(3) + 1 = 6$. $d(5) = 2 + d(3) = 3$. $d(6) = 7 + d(3) = 8$.

(c.2) $J(2) = 3$. $J(5) = 3$. $J(6) = 3$.

Iterazione 2.

(b) $j = 5$. $S = \{1, 3, 5\}$.

(c.1) $\omega^+(j) \cap T = \{(5, 2), (5, 4)\}$. $d(2) = d(5) + 2 = 5$. $d(4) = d(5) + 5 = 8$.

(c.2) $J(2) = 5$. $J(4) = 5$.

Iterazione 3

(b) $j = 2$. $S = \{1, 2, 3, 5\}$.

(c.1) $\omega^+(j) \cap T = \{(2, 4), (2, 6)\}$. L'etichetta del nodo 4 non soddisfa la condizione al passo (c), per cui va' aggiornata solo l'etichetta (e il predecessore) del nodo 6. $d(6) = d(2) + 1 = 6$.

(c.2) $J(6) = 2$.

Iterazione 4.

(b) $j = 6$. $S = \{1, 2, 3, 5, 6\}$.

(c) $\omega^+(j) \cap T = \emptyset$.

Iterazione 5.

(b) $j = 4$. $S = \{1, 2, 3, 4, 5, 6\}$. **STOP**.

I pesi dei cammini minimi saranno quindi: $d(1) = 0$. $d(2) = 5$. $d(3) = 1$. $d(4) = 8$. $d(5) = 3$. $d(6) = 6$.

Una comoda rappresentazione dell'evolvere dell'algoritmo è la seguente forma tabellare ove le righe rappresentano iterazioni mentre le colonne rappresentano i nodi selezionati ad ogni iterazione. Per ciascun nodo j ci sono due colonne che riportano il valore della variabile $d(j)$ e della $J(j)$ all'iterazione i-esima. L'elemento selezionato all'iterazione i-esima è rappresentato in grassetto e, nelle iterazioni successive, il valore della variabile corrispondente non viene più riportato. La colonna corrispondente al nodo 1 è omessa.

	nodo									
	2		3		4		5		6	
	d	J	d	J	d	J	d	J	d	J
It. 0	7	1	1	1	$+\infty$	1(fitt.)	$+\infty$	1(fitt.)	$+\infty$	1(fitt.)
It. 1	6	3	**1**	**1**	$+\infty$	1(fitt.)	3	3	8	3
It. 2	5	5			8	5	**3**	**3**	8	3
It. 3	**5**	**5**			8	5			6	2
It. 4					8	5			**6**	**2**
It. 5					**8**	**5**				

6.4 Due esempi

In questo paragrafo consideriamo più in dettaglio due esempi di applicazioni non banali di quanto visto in questo capitolo.

6.4.1 Tecniche reticolari di programmazione delle attività

I progetti di grandi dimensioni sono costituiti da più attività, che devono essere tutte completate affinché il progetto di cui fanno parte sia completato, ma che possono essere iniziate e svolte indipendentemente l'una dall'altra, purché sia rispettata una data sequenza. Queste condizioni sono caratteristiche di molti progetti di sviluppo e produzione, ad esempio nel settore aeronautico ed aereospaziale, o di costruzione, ad esempio nell'ingegneria civile, o di manutenzione di grossi sistemi; tutti progetti in cui il numero di attività costituenti può essere dell'ordine delle migliaia. La gestione di un progetto consiste nel coordinamento dell'esecuzione delle varie attività, unitamente al controllo dei tempi e dei costi di esecuzione. Poiché questo è evidentemente un problema di rilevante importanza economica, e a volte strategica, per esso sono state sviluppate, a partire dal 1958, alcune tecniche particolarmente efficaci, tra cui hanno assunto un ruolo importante il PERT (Program Evaluation and Review Technique) e il CPM (Critical Path Method). Il PERT è stato sviluppato inizialmente per pianificare le operazioni di ricerca e sviluppo connesse al progetto del missile Polaris, e l'applicazione di questa tecnica ha consentito di concludere il progetto con due anni di anticipo sui cinque anni inizialmente preventivati. Questo successo iniziale ha portato ad una sua rapida diffusione. Lo scopo principale del PERT è quello di pianificare e controllare i tempi di completamento delle attività di un progetto, e del progetto nel suo insieme, tenendo conto del fatto che i tempi di esecuzione delle varie attività non sono a priori noti con certezza, e possono variare in dipendenza di molteplici fattori aleatori (PERT-Time); successivamente sono state fatte estensioni alla pianificazione e controllo dei costi di esecuzione, soggetti anch'essi ad aleatorietà (PERT-Cost). PERT-Time e PERT-Cost costituiscono applicazioni software molto diffuse. Il CPM si applica invece quando al tempo di esecuzione di ogni attività può essere attribuito un valore certo, più o meno lungo a seconda di quanto si decide di spendere per l'esecuzione dell'attività stessa; è esperienza comune che se si riduce il tempo di esecuzione di un'attività il suo costo aumenta e viceversa. Il CPM ha come scopo principale quello di pianificare e controllare i tempi di esecuzione di un progetto, rendendo minima la spesa complessiva, e trova larga applicazione in programmi riguardanti la manutenzione periodica di grossi impianti industriali, e lavori di produzione e costruzione per cui esiste una consolidata esperienza, cosicché

si possono ritenere note con esattezza le relazioni costo-tempo di esecuzione. Base comune del PERT e del CPM è la rappresentazione del progetto mediante un grafo orientato, secondo opportune norme. Poiché in questo contesto, al grafo rappresentativo del progetto viene dato il nome di diagramma reticolare , queste tecniche vengono chiamate tecniche reticolari di programmazione delle attività . Dall'analisi del diagramma reticolare si possono ricavare molte infomazioni significative sul progetto. Vogliamo qui illustrare alcune considerazioni, che possono essere fatte nell'analizzare i progetti, in cui giocano un ruolo importanti i cammini minimi.

Iniziamo la nostra analisi spiegando come sia possibile associare ad un progetto un *diagramma reticolare* (cioè un grafo orientato).

La costruzione del diagramma

Il diagramma reticolare rappresenta la successione temporale e la reciproca dipendenza delle varie attività che concorrono all'esecuzione del progetto, attività che devono essere completate prima che il progetto possa considerarsi eseguito. Il primo passo nella costruzione del diagramma reticolare consiste nell'individuazione e nell'elencazione di tutte le attività coinvolte nell'esecuzione del progetto, con un livello di disaggregazione tale per cui le si possa considerare ciascuna distinta da tutte le altre. Segue una fase di rappresentazione grafica, che dà luogo al disegno di un grafo orientato in cui ogni attività è rappresentata da un arco o ramo i cui nodi estremi rappresentano, secondo la direzione del ramo, l'inizio e il termine dell'attività in questione. Pertanto nei diagrammi reticolari un'attivita A è rappresentata come in figura 6.9, ove i nodi i e j rappresentano rispettivamente l'inizio e il termine dell'attività.

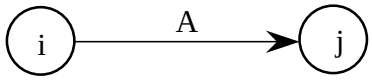


Figura 6.9: Rappresentazione grafica dell' attività A

Naturalmente tra le varie attività esistono delle precedenze, per cui, per ciascuna attività, esisteranno altre attività che devono essere completate prima che quella in questione possa avere inizio. Il caso più semplice di precedenza quello indicato in figura 6.10a, ove l'attività A precede l'attività B, e il nodo j rappresenta il termine dell'attività A e l'inizio dell'attività B. In figura 6.10b è rappresentato il caso in cui l'attività A precede l'attività B che a sua volta precede l'attività C. Può però anche avvenire che due attività, la A e la B precedano una terza, la C, senza che tra A e B esista una relazione di precedenza: questo caso, in cui le attività A e B possono essere svolte in parallelo, è rappresentato in figura 6.10c. In figura 6.10d abbiamo il caso in cui le due attività B e C, tra cui non sussistono precedenze, sono entrambe precedute dall'attività A. Per esprimere il fatto che l'attività A precede l'attività B utilizziamo la notazione A $<$ B , per esprimere il fatto che l'attività B è preceduta dall'attività A, utilizziamo la notazione B $>$ A .

Esempio 1. In figura 6.11b) è rappresentato il caso di un progetto il cui completamento richiede l'esecuzione di 9 attività, tra cui sussistono le relazioni di precedenza:

$$\text{A} < \text{B,C}; \quad \text{B} < \text{D, E}; \quad \text{C} < \text{F}; \quad \text{D} < \text{G}; \quad \text{E,F} < \text{H}; \quad \text{G,H} < \text{I}.$$

Il progetto rappresentato in 6.11b verrà più volte riutilizzato a scopo esemplificativo; ad esso faremo pertanto riferimento con il nome di progetto P1.

Nel disegnare il diagramma reticolare si utilizzano le seguenti regole fondamentali, alcune delle quali già implicitamente enunciate:

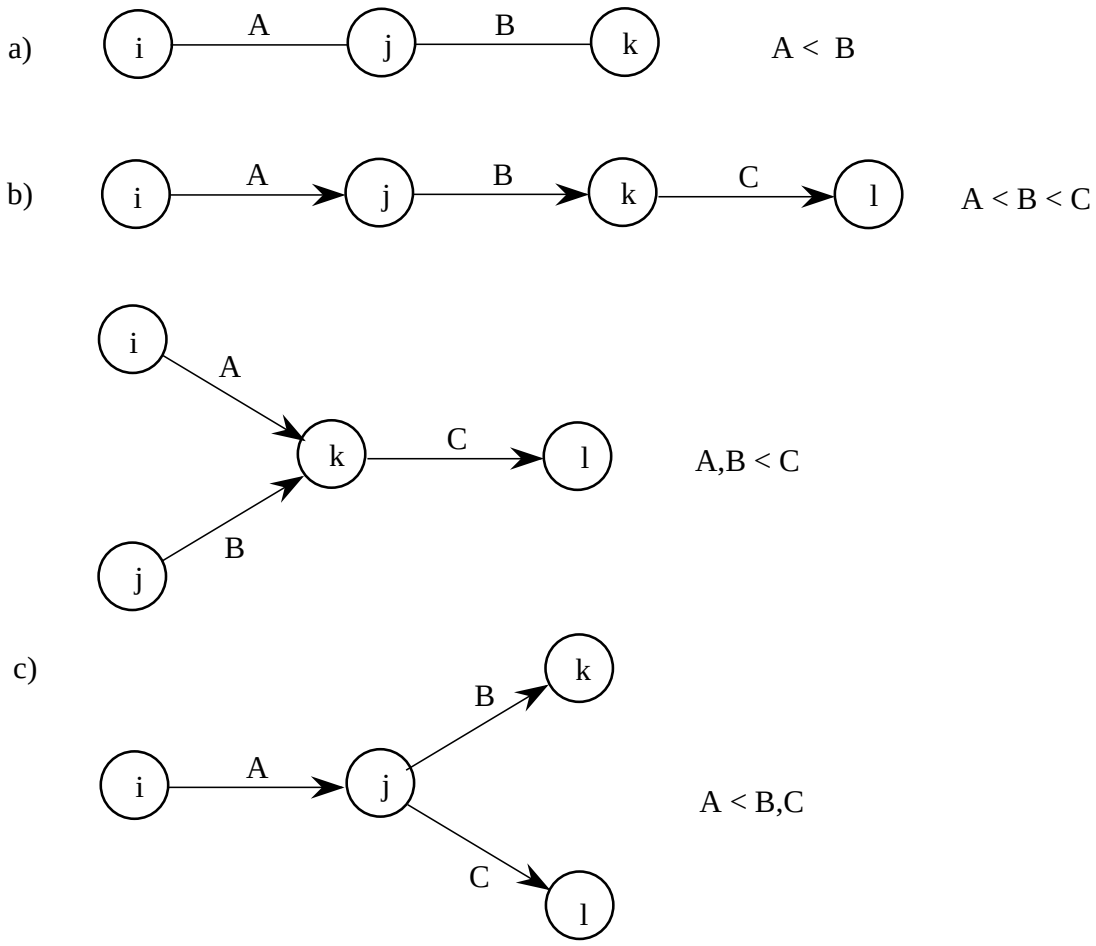


Figura 6.10: Rappresentazione grafica delle regole di precedenza

1. Le attività sono rappresentate dai rami del grafo.
2. L'inizio di un'attività è subordinato al completamento di tutte quelle che la precedono: in termini di diagramma reticolare ciò significa che rami diretti verso un nodo rappresentano attività da completare prima che abbiano inizio le attività rappresentate da rami aventi origine nel nodo stesso.
3. La lunghezza dei rami o la loro forma non hanno significato.
4. Due nodi non possono essere collegati da più di un ramo.
5. L'inizio del progetto è rappresentato da un nodo contrassegnato con zero.
6. Tutti i nodi sono numerati in modo che, se esiste un ramo diretto dal nodo i al nodo j, risulta $i < j$.
7. Il grafo può avere un solo nodo iniziale e un solo nodo finale.

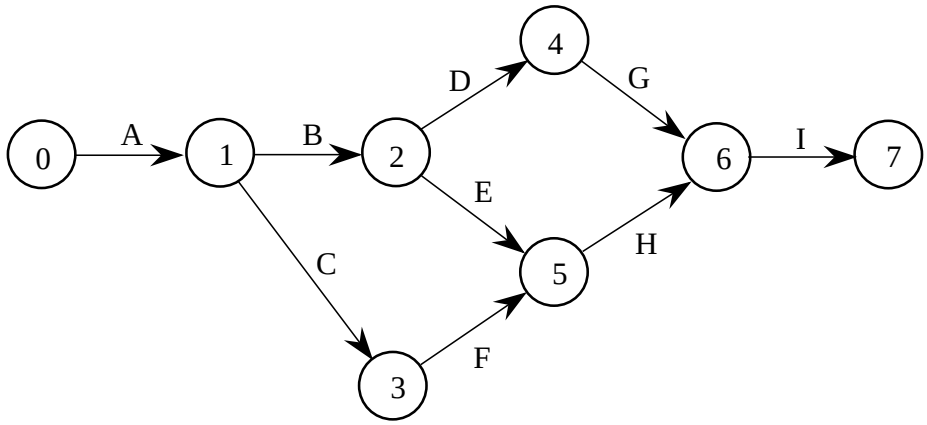


Figura 6.11: Diagramma reticolare del progetto P1

Delle suddette regole, le prime tre tengono conto della logica interna del grafo; le altre quattro sono richieste dai metodi di analisi del grafo, metodi che utilizzano tabulazioni ed uso di calcolatori.

Per quanto riguarda la regola 6, ricordiamo che una siffatta numerazione è detta numerazione topologica e che nel paragrafo precedente abbiamo visto che è possibile dare una numerazione topologica ai nodi di un grafo orientato se e solo se il grafo è aciclico. Bisogna quindi chiedersi se il diagramma reticolare di un progetto è un grafo aciclico. La risposta è ovviamente positiva. Se infatti esistesse un ciclo (orientato) nel diagramma reticolare di un progetto, questo vorrebbe dire, per come abbiamo costruito il diagramma reticolare stesso, che esistono delle attività che non possono iniziare prima di essere state concluse, e questo e ovviamente assurdo.

La regola 4, infine, ha lo scopo di rendere univoca la corrispondenza tra coppie di nodi ed attività, corrispondenza che potrebbe venire meno quando alcune attività possono essere svolte in parallelo, come accade nel seguente esempio.

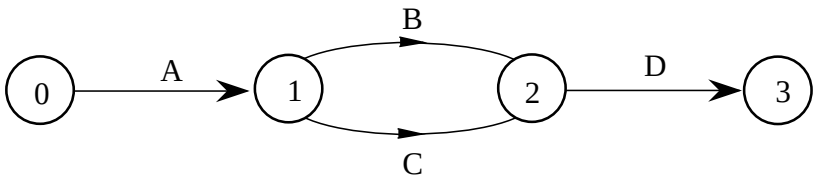


Figura 6.12: Una coppia di nodi che non individua univocamente un' attività

Esempio 2. Consideriamo le attività A, B, C, D, con le relazioni di precedenza A < B, C; B, C < D. Il grafo costruito ignorando la regola 4 è quello di 6.12, in cui alla coppia di nodi (1,2) non è associata in maniera univoca un'attività.

Quando la regola 4 non è esplicitamente soddisfatta dal progetto, come accade nell' Esempio 2, occorre ricorrere ad un artificio che consiste nell'introdurre un' *attività fittizia,* cui va associato un tempo di esecuzione nullo: nel caso dell'Esempio 2, introducendo l'attività fittizia X si ottiene il grafo della 6.13,

che rispetta la regola 4. Con l' introduzione di attività fittizie è quindi possibile individuare ogni attività

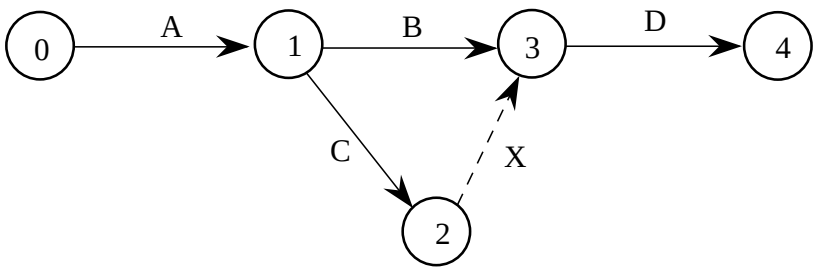


Figura 6.13: Introduzione di un' attività fittizia

mediante la coppia ordinata dei nodi estremi.

Esercizio 1. Un progetto comporta l'esecuzione delle 7 attività A, B, C, D, E, F, G, tra cui sussistono le relazioni di precedenza: A < B,C; C < D,E; D,E < F; B,F < G . Si costruisca il diagramma reticolare del progetto.

Il progetto considerato nell'esercizio precedente (il cui diagramma reticolare è dato in fondo al capitolo) sarà in seguito riutilizzato a scopo di esercizio; ad esso faremo riferimento con il nome di progetto P2.

Un altro caso in cui è richiesta l'introduzione di un'attività fittizia si verifica quando due attività precedono entrambe una terza attivià, e una sola delle due ne precede una quarta. In questo caso è solo l'introduzione di un'attività fittizia che rende possibile la costruzione del grafo, come si vede nel prossimo esempio.

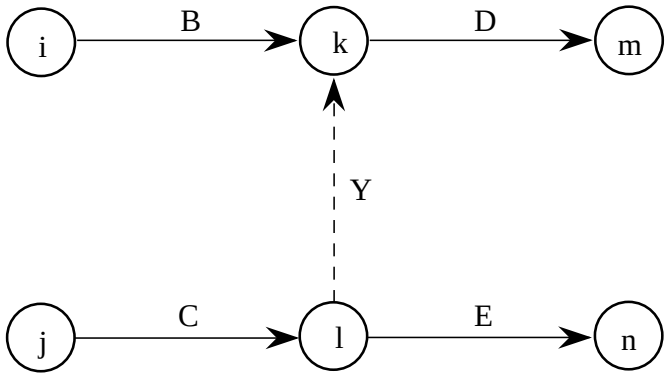


Figura 6.14: Introduzione dell' attività fittizia Y

Esempio 3. Un progetto prevede, tra la altre, le attività B, C, D, E, che devono essere svolte rispettando le precedenze: B,C < D ; C < E . Dal diagramma di figura 6.14 si rileva come solo l'introduzione dell'attività fittizia Y rende possibile la rappresentazione di questa parte del progetto.

Nel diagramma reticolare ogni nodo (ad eccezione del primo e dell'ultimo) rappresenta il termine di alcune attività e l'inizio di altre. Pertanto, in questo contesto, i nodi vengono anche chiamati eventi.

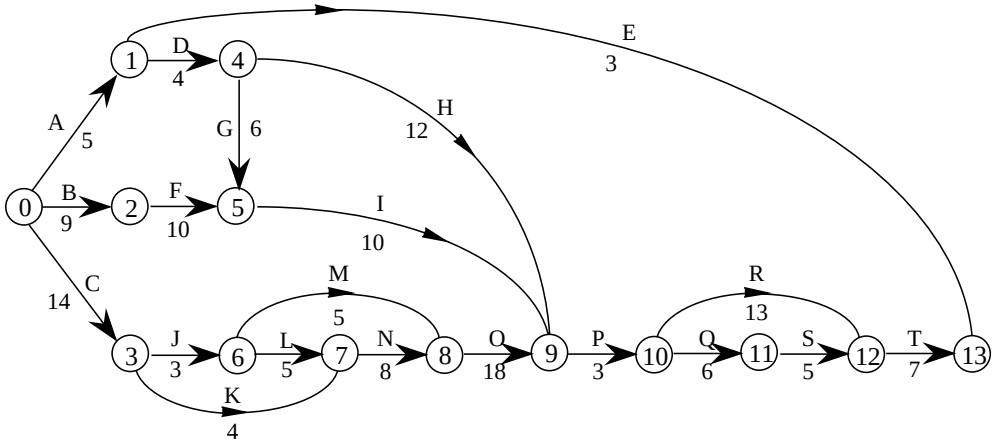


Figura 6.15: Diagramma reticolare del progetto P3

Esempio 4 Un progetto consiste nell'esecuzione di una serie di attività indicate con A, B, ..., T con le seguenti relazioni di precedenza: A, B, C possono iniziare immediatamente; D, E > A; F > B; G, H > D; I > F, G; J,K > C; M,L > J; N > K,L; O > M,N; P > H,I,O; R,Q> P; S > Q; T > R,S . Costruiamo il diagramma reticolare del progetto, numerando i nodi in modo che se il ramo (i, j) rappresenta un'attività, risulta $i < j$. Utilizzando le regole prima elencate, otteniamo il grafo di figura 6.15.

Anche il progetto considerato nell'Esempio 4 verrà riutilizzato nel seguito; ad esso faremo riferimento con il nome di progetto P3.

Domanda 1. Sai descrivere a cosa corrisponde il verificarsi dell'evento 9 nel diagramma reticolare del progetto P3 rappresentato in figura 6.15?

Domanda 2. Nel costruire il diagramma reticolare del progetto P3 è stato necessario introdurre attività fittizie ?

Esercizio 2. Supponiamo che alle relazioni di precedenza del progetto P3 debba essere aggiunta la C < F . Come si modifica il grafo di figura 6.15?

Esercizio 3. Supponi che il progetto P3 sia ampliato con l'introduzione dell'attività U, per cui sussistono le relazioni di precedenza U > M,N ; U < Q . Come si modifica il grafo di figura 6.15?

Il percorso critico

Abbiamo finora visto come sia possibile costruire un grafo che rappresenti l'esecuzione di un progetto, dopo che il progetto stesso è stato decomposto in attività, o fasi, di cui si siano analizzate le relazioni di precedenza. Non abbiamo però finora tenuto conto del tempo richiesto per l'esecuzione delle varie attività che compongono il progetto, e che ovviamente condizionano il tempo di esecuzione complessivo. Poiché i metodi reticolari di programmazione hanno, tra gli altri, lo scopo di controllare i tempi di esecuzione delle attività al fine di ottenere il rispetto del tempo di completamento del progetto, occorre aggiungere alla analisi qualitativa delle precedenze già effettuata anche un'analisi quantitativa che determini i valori

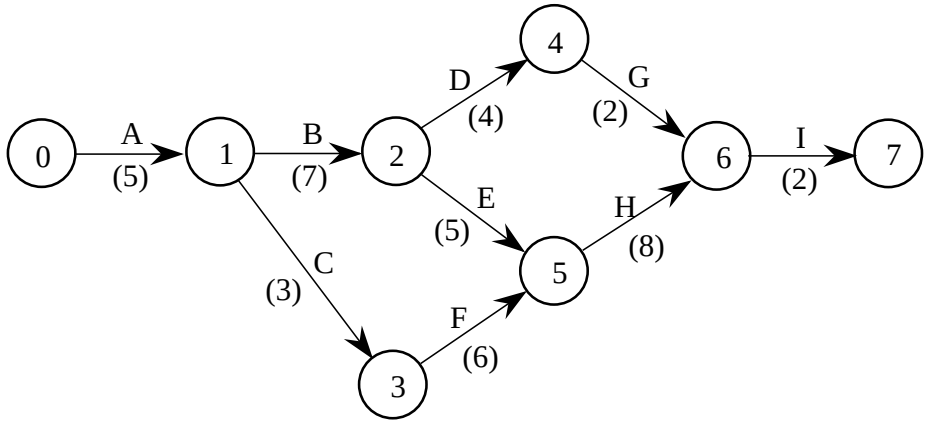


Figura 6.16: Tempi di esecuzione delle attività del progetto P1

temporali corrispondenti agli eventi descritti dal grafo, e individui i limiti entro cui tali valori temporali possono variare senza pregiudicare il valore del tempo complessivo di completamento. Per effettuare quest'analisi associamo ad ogni attività (i, j) un tempo di esecuzione t_{ij}. Il tempo di esecuzione t_{ij} può essere assunto come variabile certa, il che avviene nel CPM, o come variabile aleatoria, il che avviene nel PERT. In entrambi i casi il metodo di analisi è fondamentalmente lo stesso, per cui in questo paragrafo facciamo riferimento alla situazione in cui i tempi di esecuzione sono noti con certezza. In ogni progetto esiste un certo insieme di attività che sono di particolare importanza ai fini della determinazione del tempo di completamento dell'intero progetto, nel senso che se si verifica un ritardo nel completamento di una di queste attività, si verifica un ritardo anche nel completamento del progetto. Altre attività invece sono meno importanti, nel senso che possono anche subire un ritardo, entro certi limiti, senza che l'intero progetto ne risenta. È evidente l' importanza di distinguere tra questi due tipi di attività, così come quella di determinare il tempo minimo entro il quale certe attività intermedie possono essere completate. Quanto esposto in questo paragrafo serve proprio a consentire questa analisi. Supponiamo dunque che ad ogni attività (i, j) sia associato il tempo di esecuzione t_{ij}; per le attività fittizie il tempo di esecuzione è ovviamente nullo.

Definizione 6.4.1 *Si definisce tempo minimo di raggiungimento del nodo i, e si indica con t_i , il minimo tempo entro cui possono essere terminate tutte le attività afferenti al nodo i.*

Data la definizione precedente viene del tutto naturale definire il tempo di completamento minimo del progetto nel seguente modo.

Definizione 6.4.2 *Si definisce tempo minimo di completamento dell'intero progetto, e si indica con T, il tempo minimo di raggiungimento del nodo finale $T = t_f$.*

In base alle regole di costruzione del diagramma reticolare di un progetto è facile convincersi che il tempo minimo di raggiungimento di un nodo i coincide con il peso del cammino massimo dal nodo iniziale al nodo i, dove i pesi degli archi sono dati dalle durate t_{ij} delle attività che essi rappresentano. Poiché il diagramma reticolare è aciclico e i nodi sono già numerati in modo topologico, è immediato applicare l' algoritmo per i cammini massimi su grafi aciclici al fine di calcolare i tempi di raggiungimento minimi.

Esempio 3. Consideriamo il diagramma reticolare del progetto P1, e associamo alle attività A, B, ..., I i seguenti tempi di esecuzione, espressi in giorni lavorativi :

tempo di esecuzione di A: $t_{01} = 5$
tempo di esecuzione di B: $t_{12} = 7$
tempo di esecuzione di C: $t_{13} = 3$
tempo di esecuzione di D: $t_{24} = 4$
tempo di esecuzione di E: $t_{25} = 5$
tempo di esecuzione di F: $t_{25} = 6$
tempo di esecuzione di G: $t_{46} = 2$
tempo di esecuzione di H: $t_{56} = 8$
tempo di esecuzione di I : $t_{67} = 2$.

Nella figura 6.4.1 i tempi di esecuzione delle attività sono stati associati ai rami del diagramma reticolare del progetto. Posto $t_0 = 0$, possiamo calcolare per i successivi nodi i tempi minimi di raggiungimento, espressi in giorni, utilizzando l' algoritmo dei cammini massimi.

per il nodo 1, $t_1 = t_0 + t_{01} = 0 + 5 = 5$
per il nodo 2, $t_2 = t_1 + t_{12} = 5 + 7 = 12$
per il nodo 3, $t_3 = t_1 + t_{13} = 5 + 3 = 8$
per il nodo 4, $t_4 = t_2 + t_{24} = 12 + 4 = 16$
per il nodo 5, $t_5 = \max(t_2 + t_{25}, t_3 + t_{35}) = \max(12 + 5, 8 + 6) = 17$
per il nodo 6, $t_6 = \max(t_4 + t_{46}, t_5 + t_{56}) = \max(16 + 2, 17 + 8) = 25$
per il nodo 7, $t_7 = t_6 + t_{67} = 25 + 2 + 27$;

avremo inoltre per il tempo minimo di completamento del progetto, $T = t_7 = 27$ giorni.

Esercizio 4. Consideriamo nuovamente il progetto P2 e associamo alle attività i seguenti tempi di esecuzione, espressi in settimane: A, 3 settimane; B, 2; C, 1; D, 4 ; E, 1; F, 2; G, . Determinare il tempo minimo di completamento del progetto.

Domanda 3 Con riferimento al progetto P1, supponiamo che il tempo di esecuzione dell'attività H passi da 8 a 10 giorni. Come varia il tempo minimo di completamento del progetto?

Domanda 4 Sempre con riferimento al progetto P1, supponiamo ora che il tempo di esecuzione dell'attività G passi da 2 a 6 giorni. Come varia il tempo minimo di completamento del progetto?

Oltre al tempo minimo di completamento dell'intero progetto, è utile introdurre anche la nozione di tempo minimo di completamento per ogni attività. Ovviamente un'attività (i, j) può avere inizio, al più presto, dopo un tempo t_i dall'inizio dell'esecuzione del progetto, in quanto perché l'attività possa avere inizio deve essere stato raggiunto il nodo i; di conseguenza se l'attività richiede un tempo di esecuzione pari a t_{ij}, non potrà essere completata prima di un tempo pari a $t_i + t_{ij}$. Possiamo quindi dare la definizione seguente:

Definizione 6.4.3 *Si definisce tempo minimo di completamento dell'attività (i, j), e si indica con C_{ij}, il valore $C_{ij} = t_i + t_{ij}$.*

Esempio 6 Consideriamo ancora il progetto P1. Per le attività del progetto, tenendo conto dei tempi di raggiungimento dei nodi calcolati nell'Esempio 5, abbiamo i seguenti tempi minimi di completamento espressi in giorni:

per l'attività A: $C_{01} = t0 + t_{01} = 5$
per l'attività B: $C_{12} = t_1 + t_{12} = 5 + 7 = 12$
per l'attività C: $C_{13} = t_1 + t_{13} = 5 + 3 = 8$
per l'attività D: $C_{24} = t_2 + t_{24} = 12 + 4 = 16$
per l'attività E: $C_{25} = t_2 + t_{25} = 12 + 5 = 17$
per l'attività F: $C_{35} = t_3 + t_{35} = 8 + 6 = 14$

per l'attività G: $C_{46} = t_4 + t_{46} = 16 + 2 = 18$
per l'attività H: $C_{56} = t_5 + t_{56} = 17 + 8 = 25$
per l'attività I: $C_{67} = t_6 + t_{67} = 25 + 2 = 27$.
Nelle tecniche reticolari di programmazione sono di fondamentale importanza le cosidette attività critica di cui diamo la definizione.

Definizione 6.4.4 *Sia T il tempo minimo di completamento di un progetto, corrispondente a un insieme $\{t_{ij}\}$ di valori prefissati dei tempi di esecuzione delle singole attività. Un'attività (h, k) viene detta attività critica se un variazione positiva comunque piccola ma non nulla del suo tempo di esecuzione comporta una variazione della stessa entità nel tempo minimo di completamento del progetto; e cioè, un'attività (h, k) è critica se, sostituito t_{hk} con $t_{hk} + \Delta t$, con $\Delta t \neq 0$ il tempo minimo di completamento del progetto diventa $T + \Delta t$, per qualunque valore positivo di Δt .*

Esempio 7 Nel progetto P1 l'attività E è critica: infatti se si pone $t_{25} = 5 + \Delta t$, si ottiene per i tempi di raggiungi mento dei nodi 5, 6, 7 :
per il nodo 5, $t_5 = \max(t_2 + t_{25}, t_3 + t_{35}) = \max(12 + 5 + \Delta t, 8 + 6) = 17 + \Delta t$
per il nodo 6, $t_6 = \ max(t_4 + t_{46}, t_5 + t_{56}) = \max(16 + 2, 17 + \Delta t + 8) = 25 + \Delta t$
per il nodo 7, $t_7 = t_6 + t_{67} = 25 + \Delta t + 2 = 27 + \Delta t$;
invece l'attività F non critica; infatti posto $t_{35} = 6 + \Delta t$, si ottiene per il tempo di raggiungimento del nodo 5 :

$$t_5 = \max(t_2 + t_{25}, t_3 + t_{35}) = \ max(12 + 5, 8 + 6 + \Delta t) = 17$$

e cioè lo stesso valore di prima, almeno fintanto cheΔt non supera i 3 giorni; e ovviamente se t_5 non varia, non variano neanche i tempi minimi di raggiungimento dei nodi successivi.

La determinazione dei percorsi critici è evidentemente di fondamentale importanza nelle tecniche reticolari di programmazione. Infatti le attività critiche sono quelle su cui più stretto deve essere il controllo di chi gestisce l'esecuzione del progetto, nei casi in cui un ritardo dell'esecuzione complessiva comporta una penalità, che può essere sia economica, sia di immagine. Da quanto abbiamo visto finora, l'individuazione delle attività critiche è molto semplice: è evidente che le attività critiche sono tutte e sole le attività che appartengono ad almeno un cammino massimo dal nodo iniziale al nodo finale.

Per quanto riguarda le attività non critiche è anche possibile stimare di quanto esse possano essere ritardate senza aumentare il tempo di completamento del progetto. Un'attività non critica, per esempio l'attività F nel progetto P1, non appartiene a un cammino massimo dal nodo iniziale al nodo finale, ma appartiene comunque a cammini che vanno dal nodo iniziale al nodo finale (nel caso specifico considerato uno solo, in generale più di uno). Sia allora $L_{\max}$ la lunghezza massima di un percorso dal nodo iniziale al nodo finale *che passa per F*. È chiaro che l'attività F è l'unica a subire un ritardo, il ritardo massimo che è possibile tollerare senza che aumenti la durata del progetto è $T - L_{\max}$. Questo tempo è chiamato tempo di slittamento dell'attività. In generale:

Definizione 6.4.5 *Si definisce tempo di slittamento, o margine di tempo dell'attività (i, j) il valore che indica di quanto tempo può essere ritardato il completamento dell'attività (i, j) senza che si determini un aumento del tempo minimo di completamento dell'intero progetto.*

Ovviamente le attività critiche hanno tempo di slittamento nullo, le attività non critiche hanno un tempo di slittamento positivo. Segnaliamo che è possibile determinare in maniera molto semplice e efficiente i tempi di slittamento di tutte le attività di un progetto, noi non approfondiamo qui ulteriormente la questione per mancanza di tempo.

6.4.2 Gestione delle scorte.

Il problema di cui ci occuperemo in questo capitolo è il cosiddetto problema di Gestione delle Scorte. Si tratta di un problema di programmazione della produzione industriale che consiste nel decidere, dato un

particolare prodotto, le quantità da produrre o da immagazzinare in modo da soddisfare una domanda presente o futura. Tipicamente per la pianificazione si considera un numero ristretto di periodi di tempo, ad esempio i prossimi dodici mesi o le prossime 20 settimane. In generale indicheremo con $\{1, \ldots, T\}$ l'orizzonte temporale, che è composto di T periodi detti *periodi di controllo*. Per ogni periodo è nota una domanda del bene d_t. Ad esempio, se il bene consiste in tondini metallici, per ogni giorno (o settimana o mese) del nostro orizzonte di pianificazione sono note le tonnellate richieste dal mercato. Ora, la domanda giornaliera di tondini deve essere integralmente soddisfatta. Io posso scegliere se produrre quotidianamente la quantitá richiesta, oppure produrre solo in alcuni giorni (ad esempio una volta a settimana) una quantità sufficiente a soddisfare la domanda di più giorni successivi, mettendo in magazzino ciò che invierò al mercato nei giorni futuri. Ovviamente, sia produrre che immagazzinare costa, e il costo può variare da periodo a periodo. Di seguito indicheremo con c_t il costo unitario di produzione nel periodo t, mentre indicheremo con h_t il costo unitario di immagazzinamento, per $t = 1 \ldots T$. Infine, iniziare un lotto di produzione presenta i cosiddetti costi di *setup*, ovvero costi amministrativi e di avviamento. Questi costi sono fissi, non dipendono cioè dalle quantità prodotte, e vengono effettivamente sostenuti solo se si produce qualcosa. Chiameremo con f_t i costi fissi di set-up relativi al periodo t, per $t = 1 \ldots, T$.

Il problema è quello di scegliere quando (in che giorni) e quanto produrre in modo da minimizzare i costi di produzione (fissi e variabili) e i costi di immagazzinamento.

Una formulazione di PL01.

Introduciamo innanzitutto una variabile reale non-negativa p_t che indica il livello di produzione e una variabile reale non-negativa s_t che indica il livello delle scorte di magazzino per ogni periodo $t = 1, \ldots, T$. Nel primo periodo il magazzino é vuoto (per ipotesi): quindi la domanda d_1 deve essere soddsfatta dalla produzione p_1. Se $p_1 > d_1$, cioé produciamo piú di quanto domandato, la parte residua finirá nel magazzino per servire la domanda dei periodi successivi. In particolare, avendo indicato con s_1 il livello delle scorte accumulate nel primo periodo, si avrá:

$$p_1 = d_1 + s_1 \tag{6.1}$$

Per i periodi successivi al primo, la quantità prodotta p_t nel periodo t si va ad aggiungere alle scorte accumulate nel periodo precedente s_{t-1}. Quindi, nel periodo t avremo a disposizione una quantità di bene pari a $p_t + s_{t-1}$. Una parte di questa quantità servirà a soddisfare la domanda d_t nel periodo t, mentre il resto rimarrà in magazzino e cioè sarà il livello di scorte s_t nel periodo t (utilizzabili nel periodo successivo). Quindi possiamo scrivere

$$p_t + s_{t-1} = d_t + s_t \qquad t = 1, \ldots, T \tag{6.2}$$

Per tenere conto del costo fisso di produzione f_t che va pagato solo nei periodi per cui si abbia $p_t > 0$, dobbiamo aggiungere una variabile booleana x_t per $t = 1, \ldots, T$ con il seguente significato

$$x_t = \begin{cases} 1 & \text{se si produce nel periodo } t \ (p_t > 0) \\ 0 & \text{altrimenti.} \end{cases}$$

Per esprimere con un vincolo che $x_t = 1$ quando $p_t > 0$, o, equivalentemente, che possiamo produrre nel periodo t solo se $x_t = 1$ introduciamo il seguente vincolo:

$$p_t \leq M \cdot x_t \qquad t = 1, \ldots, T \tag{6.3}$$

dove M è una costante sufficientemente grande. Il ruolo del vincolo 6.3 è il seguente: se $x_t = 0$, allora non è possibile produrre ($p_t \leq 0 \rightarrow p_t = 0$). Se $x_t = 1$ allora $p_t \leq M$ e noi possiamo produrre una quantità compresa fra 0 e M. Come si è detto M deve essere scelto opportunamente in modo da rendere il vincolo sempre soddisfatto quando $x_t = 1$. Ad esempio, poichè in ogni periodo non ha senso produrre più di quanto viene richiesto durante tutto l'orizzonte temporale, un valore ammissibile per M può essere $M = \sum_{t=1}^{T} d_t$.

Siamo ora in grado di scrivere la funzione obiettivo, che si compone di tre termini:

- Il costo variabile di produzione, pari a $\sum_{t=1}^{T} c_t p_t$.
- Il costo fisso di produzione, pari a $\sum_{t=1}^{T} f_t x_t$.
- Il costo (variabile) di immagazzinamento, pari a $\sum_{t=1}^{T} h_t s_t$.

La funzione obiettivo risulta quindi essere:

$$\min \sum_{t=1}^{T} c_t p_t + f_t x_t + h_t s_t \tag{6.4}$$

Uno sguardo alla "struttura" del problema

Nelle righe precedenti abbiamo formulato il problema di gestione delle scorte come problema di programmazione mista. Per la soluzione del modello potremmo utilizzare algoritmi generali quali il branch-and-bound. Tuttavia, studiando più a fondo la struttura del problema, faremo vedere come sia possibile applicare algoritmi molto più efficienti per identificare una soluzione ottima.

Una prima osservarvazione riguarda la struttura delle soluzioni ottime. In particolare, si può facilmente dimostrare che esiste sempre una soluzione ottima per cui sia $p_t \cdot s_{t-1} = 0$ per $t = 1, \ldots, T$. In altri termini, se decidiamo nel periodo t di soddisfare la domanda d_t con scorte provenienti dal periodo precedente non produrremo niente; di converso, se produciamo qualcosa ($p_t > 0$) allora il livello delle scorte precedenti s_{t-1} deve essere 0 e tutta la domanda sarà soddisfatta dalla produzione attuale. Quindi, possiamo distinguere i periodi di controllo in periodi in cui si produce (per cui $p_t > 0$ e $s_{t-1} = 0$) e periodi in cui non si produce (per cui $s_{t-1} > 0$ e $p_t = 0$). La domanda di ciascun periodo produttivo è soddisfatta dalla produzione, mentre nei periodi non produttivi la domanda è soddisfatta dalle scorte. Se i è un periodo produttivo e j è il prossimo periodo produttivo, la domanda relativa ai periodi compresi fra i e $j-1$ è servita dalla produzione realizzata in i. Quindi, in i io dovrò produrre $d_i+d_{i+1}+d_{i+2}\ldots+d_{j-1} = \sum_{t=i}^{j-1} d_t$. Calcoliamo ora il costo complessivo $C(i,j)$ di produrre in i per soddisfare la domanda fino a j. Poichè la produzione in i sarà $p_i = \sum_{t=i}^{j-1} d_t$, il suo costo ammonterà a $c_i * \sum_{t=i}^{j-1} d_t$; a questo costo va' aggiunto il costo fisso di produzione del periodo i, f_i; infine, bisogna calcolare periodo per periodo il livello delle scorte. Ricordiamo che le scorte nel periodo k sono determinate dall'equazione $s_k = p_k + s_{k-1} - d_k$ (e cioè le scorte che accumulo in un certo periodo sono pari alle scorte del periodo precedente, più la produzione nel periodo, meno ciò che se ne va per soddisfare la domanda del periodo). Quindi, nel periodo i le scorte saranno pari a $s_i = p_i + s_{i-1} - d_i$; essendo $p_i = d_i + d_{i+1} + \ldots + d_{j-1} = \sum_{t=i}^{j-1} d_t$, ed avendo inoltre $s_{i-1} = 0$, sarà: $s_i = \sum_{t=i}^{j-1} d_t - d_i = \sum_{t=i+1}^{j-1} d_t$ e il suo costo sarà pari quindi $h_i * \sum_{t=i+1}^{j-1} d_t$. Nel periodo successivo, $i+1$ (che supponiamo momentaneamente non essere un periodo produttivo), il livello delle scorte sarà $s_{i+1} = p_{i+1}+s_i-d_{i+1}$, ed essendo $p_{i+1} = 0$, $s_i = \sum_{t=i+1}^{j-1} d_t$ avremo $s_{i+1} = \sum_{t=i+1}^{j-1} d_t - d_{i+1} = \sum_{t=i+2}^{j-1} d_t$, ed il costo sarà $h_{i+1} * \sum_{t=i+2}^{j-1} d_t$. Quindi, il costo complessivo delle scorte mantenute dal periodo i al periodo $j-1$ sarà pari a $h_i * \sum_{t=i+1}^{j-1} d_t + h_{i+1} * \sum_{t=i+2}^{j-1} d_t + \ldots + h_{j-2} * d_{j-1}$. Riassumendo, il costo complessivo di produrre in i per soddisfare la domanda fino a $j-1$ sarà pari a $C(i,j) = c_i * \sum_{t=i}^{j-1} d_t + f_i + h_i * \sum_{t=i+1}^{j-1} d_t + h_{i+1} * \sum_{t=i+2}^{j-1} d_t + \ldots + h_{j-2} * d_{j-1}$.

Ora, una soluzione (livello di produzione e livello di scorte in ogni periodo) è completamente determinata una volta stabilito l'insieme dei periodi produttivi (periodi in cui la produzione è positiva). Infatti, se $I = \{1, i, j, \ldots, r, q\}$ è l'insieme ordinato dei periodi produttivi, allora vuol dire che si produce in 1 per soddisfare la domanda fino a $i-1$, quindi si produce in i per soddisfare la domanda fino a $j-1$, etc. etc., quindi si produce in r per soddisfare la domanda fino a $q-1$ e si produce in q per soddisfare la domanda fino a T. Il costo della soluzione sarà $C(I) = C(1,i) + C(i,j) + C(j,k) + C(k,.) + \ldots + C(.,r) + C(r,q)$.

Quindi, il problema di gestione delle scorte può essere riformulato come il problema di trovare l'insieme di periodi produttivi I^* che minimizzi il costo $C(I^*)$.

Rappresentazione del problema di gestione delle scorte come problema di cammino minimo su grafo

Vediamo ora come il problema di trovare un insieme di istanti produttivi ottimale può essere agevolmente ricondotto a un problema di cammino minimo su grafi aciclici. A tal scopo definisco un grafo orientato $G = (V, E)$. L'insieme dei nodi è l'insieme dei periodi di controllo più un nodo fittizio che denominiamo $T + 1$, e cioè $V = \{1, \ldots, T, T + 1\}$. L'insieme degli archi è l'insieme di tutti gli archi "in avanti", e cioè $E = \{(i, j) : i \in V, j \in V, i < j\}$. E' facile che con questa definizione degli archi, l'insieme dei nodi risulta ordinato topologicamente, e cioè gli archi vanno sempre da nodi con indice più piccolo a nodi con indice più grande. Quindi, il grafo G è privo di cicli orientati (vedi il teorema 6.2.1). Associamo inoltre a ogni arco (i, j) il peso $p(i, j) = C(i, j)$.

Ora, a ogni cammino $P = \{1, i, j, k, \ldots, r, q, T + 1\}$ dal nodo 1 al nodo $T + 1$ associamo l'insieme di periodi produttivi $I(P) = \{1, i, j, k \ldots, r, q\}$. Per quanto visto sopra, il costo $C(I) = C(1, i) + C(i, j) + C(j, k) + C(k, .) + \ldots + C(., r) + C(r, q)$; ma, per come sono stati definiti i pesi sugli archi di G, è facile vedere che $C(I) = p(P)$. Quindi, il peso del cammino minimo da 1 a $T + 1$ in G corrisponde al costo di un insieme di istanti produttivi di costo minimo.

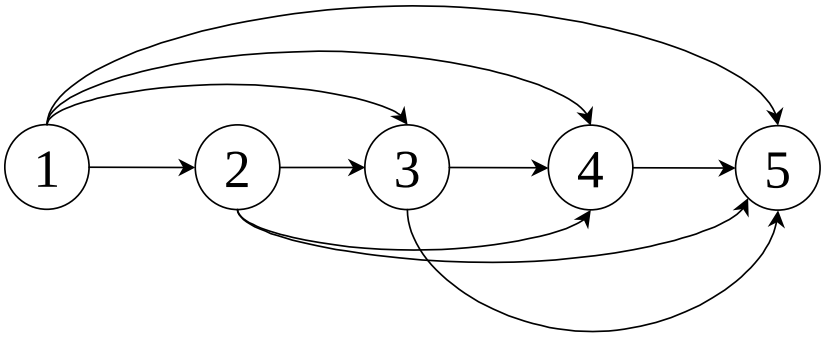
Esempio

Si consideri l'esempio riportato in tabella.

Periodo	Domanda	f	c	h
1	20	30	3	2
2	30	40	3	2
3	40	30	4	1
4	30	50	4	1

Per prima cosa calcoliamo costruiamo il grafo associato all'istanza del problema. Si tratta di un nodo con 5 nodi, uno in più dei periodi di controllo: quindi $V = \{1, 2, 3, 4, 5\}$ e tutti gli archi "in avanti".

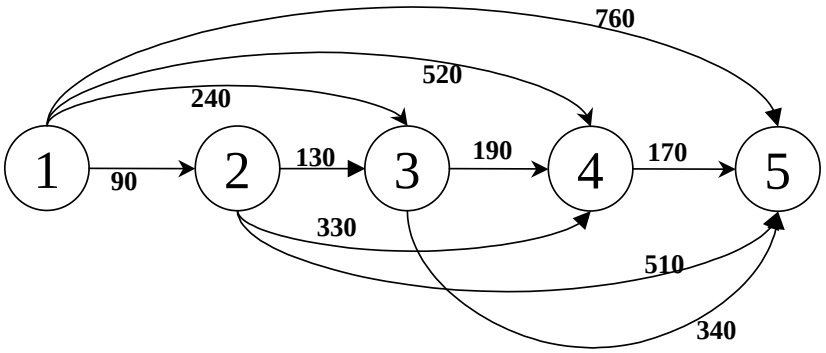
Figura 6.17: Il grafo dei periodi



Per prima cosa calcoliamo i pesi degli archi. L'arco $(1, 2)$ ha peso pari a $C(1, 2)$, cioè il costo di produrre nel periodo 1 per soddisfare la domanda fino al periodo $2 - 1 = 1$. Poichè la domanda è

pari a $d_1 = 20$, il costo variabile di produzione sarà $c_1 d_1 = 3 \times 20 = 60$; a questo va' aggiunto il costo fisso di produzione nel periodo 1, $f_1 = 30$. Non ci sono scorte residue e quindi il costo complessivo $C(1,2) = 60+30 = 90$. Calcoliamo ora il costo $C(1,3)$, cioè il costo di produrre nel periodo 1 per soddisfare la domanda fino al periodo $3-1 = 2$. La domanda complessiva sarà $d_1+d_2 = 50$. Quindi il costo variabile di produzione sarà pari a $c_1(d_1+d_2) = 3 \times 50 = 150$; a questo va' aggiunto il costo fisso di produzione nel periodo 1, $f_1 = 30$. Inoltre, una parte della produzione (d_2) dovrà essere immagazzinata per un periodo (da 1 fino a 2), e quindi pagheremo un costo d'immagazzinamento pari a $s_1 * d_2 = 2 \times 30 = 60$. Quindi, il costo complessivo $C(1,3) = 150 + 30 + 60 = 230$. Calcoliamo ora il costo $C(1,4)$, cioè il costo di produrre nel periodo 1 per soddisfare la domanda fino al periodo $4-1 = 3$. La domanda complessiva sarà $d_1 + d_2 + d_3 = 90$. Quindi il costo variabile di produzione sarà pari a $c_1(d_1 + d_2 + d_3) = 3 \times 90 = 270$; a questo va' aggiunto il costo fisso di produzione nel periodo 1, $f_1 = 30$. Inoltre, una parte della produzione $(d_2 + d_3)$ dovrà essere immagazzinata nel periodo 1 fino al periodo 2, e un'altra parte d_3 dovrà essere immagazzinata nel periodo 2 fino al periodo 3 e quindi pagheremo un costo d'immagazzinamento pari a $s_1(d_2 + d_3) + s_2 d_3 = 2 \times 70 + 2 \times 40 = 220$. Quindi, il costo complessivo $C(1,4) = 270 + 30 + 220 = 520$. A titolo esemplificativo, calcoliamo un ultimo peso, il peso dell'arco $(3,5)$ corrispondente a $C(3,5)$.

Figura 6.18: Calcolo dei pesi sugli archi

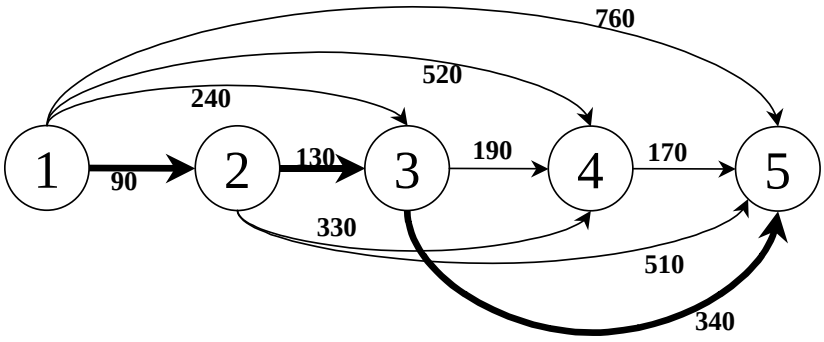


A questo punto, applichiamo l'algoritmo per il calcolo di un cammino minimo su grafi aciclici per calcolare un cammino di peso minimo dal nodo 1 al nodo 5.

- $f(1) = 0$;
  $J(1) = 1$;
- $f(2) = f(1) + p(1,2) = 90$;
  $J[2] = 1$.
- $f(3) = \min\{f(1) + p(1,3), f(2) + p(2,3)\} = \min\{240, 220\} = 220$;
  $J(3) = 2$.
- $f(4) = \min\{f(1) + p(1,4), f(2) + p(2,4), f(3) + p(3,4)\} = \min\{520, 420, 410\} = 410$;
  $J(4) = 4$.
- $f(5) = \min\{f(1) + p(1,5), f(2) + p(2,5), f(3) + p(3,5), f(4) + p(4,5)\} = \min\{760, 600, 560, 580\} = 560$;
  $J(5) = 3$.

A questo punto è possibile ricostruire il cammino minimo da 1 a 5 utilizzando il vettore dei predecessori. Infatti, il predecessore di 5 è il nodo 3, il predecessore di 3 è il nodo 2 e il predecessore di 3 è il nodo 1.

Figura 6.19: Cammino minimo da 1 a 5



Capitolo 7

Massimo flusso

In questo capitolo esaminiamo un importante problema di PL su grafi, il problema del *flusso massimo* in un grafo; dopo aver presentato il problema e alcune delle sue caratteristiche fondamentali, studieremo un algoritmo per la sua soluzione e considereremo inoltre alcune sue generalizzazioni ed estensioni.

In maniera molto discorsiva, possiamo immaginare che il grafo sia una rete idrica con il quale si tenta di portare dell'acqua da un nodo sorgente s ad un utilizzatore rappresentato da un nodo t. Per far questo si usa una rete di tubature rappresentata dagli archi. Le tubature hanno una certa capacita' c_{uv} (cioè possono trasportare al più c_{uv} litri di acqua al secondo). Le tubature si intersecano nei nodi, dove può essere deciso, come l' acqua che arriva nel nodo deva essere distribuita ai tubi "in uscita". Si tratta di determinare come incanalare l'acqua in modo da massimizzare il flusso da s a t; da qui il nome del problema di massimo flusso. Ovviamente, l'interpretazione idrica è solo una delle molte possibili. In fatti si potrebbe trattare di una rete di trasporto di energia elettrica (e allora i rami rappresenterebbero cavi elettrici), di persone (e allora i rami rappresenterebbero tratte servite da aerei, treni, macchine etc.), di informazioni etc.

7.1 Il problema del massimo flusso.

Sia dato un grafo orientato $G = (V, E)$ con $V = \{s, t, 3, \ldots, n\}$ ed $|E| = m$ e sia data una capacità $c_{uv} \in I\!R^+$ associata a ciascun arco $(u, v) \in E$. Supponiamo che esistano

- un nodo s detto *sorgente* con solo archi uscenti ($\omega^-(s) = \{\emptyset\}$);
- un nodo t detto *pozzo* con solo archi entranti ($\omega^+(t) = \{\emptyset\}$).

I nodi distinti da s e da t sono detti *nodi intermedi*.

Un vettore $\bar{x} \in I\!R^m$ (notare che $\bar{x}$ ha un numero di componenti pari alla cardinalità dell'insieme di archi di G) viene detto *flusso s-t ammissibile* o, per brevità *flusso ammissibile*, se soddisfa i seguenti vincoli:

$$0 \leq \bar{x}_{uv} \leq c_{uv} \qquad \text{per ogni } (u, v) \in E. \tag{7.1}$$

$$\sum_{(u,v)\in\omega^-(v)} \bar{x}_{uv} - \sum_{(v,k)\in\omega^+(v)} \bar{x}_{vk} = 0, \quad \forall v \in V - \{s, t\} \tag{7.2}$$

I vincoli (7.1) indicano che il flusso su ogni arco è sempre non-negativo e non eccede la capacità dell'arco stesso. I vincoli (7.2) indicano che per ogni nodo intermedio il flusso totale entrante nel nodo eguagli il flusso totale uscente dal nodo stesso. Per questo motivo, le $n - 2$ equazioni (7.2) sono dette *equazioni di conservazione* (del flusso).

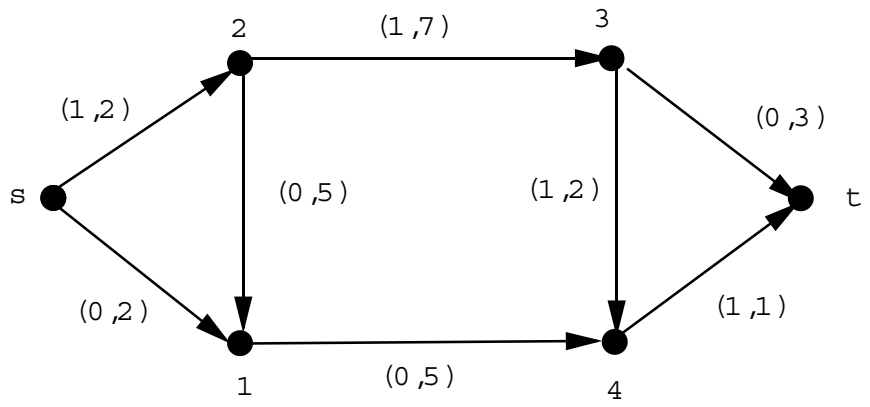


Figura 7.1: Flusso ammissibile

Consideriamo l'esempio di figura 7.1. Nella figura, fra parentesi tonde, si è indicato per ogni arco e il valore del flusso $\bar{x}_e$ e la capacità c_e. Ad esempio, nell'arco $(s, 1)$, il flusso vale 0 mentre la capacità 2. Si può facilmente verificare che il flusso indicato è ammissibile.

Dato un flusso s-t, diremo *valore* del flusso, il flusso uscente dal nodo s, e cioè la quantità:

$$\bar{f} = \sum_{(s,u)\in\omega^+(s)} \bar{x}_{su} \tag{7.3}$$

Nel grafo di figura 7.1, il valore del flusso $\bar{f}$ è uguale a 1.

Il problema del **massimo flusso** si può ora enunciare come segue:

Trovare un flusso ammissibile nel grafo G in modo che il valore del flusso $\bar{f}$ sia massimo.

Questo problema può essere formulato come un problema di PL. Associamo una variabile x_{uv} ad ogni arco $(u, v) \in E$. Tale variabile rappresenta il flusso inviato sull'arco (u, v). Inoltre associamo una variabile f alla quantità totale di flusso inviata da s a t. Da quanto sopra detto, il problema (MF) verrà formulato nel seguente modo:

$$\max f \tag{7.4}$$

$$-\sum_{(s,k)\in\omega^+(s)} x_{sk} + f = 0 \tag{7.5}$$

$$\sum_{(u,v)\in\omega^-(v)} x_{uv} - \sum_{(v,k)\in\omega^+(v)} x_{vk} = 0, \quad \forall v \in V - \{s, t\} \tag{7.6}$$

$$x_{uv} \leq c_{uv} \qquad \text{per ogni } (u, v) \in E. \tag{7.7}$$

$$x_{uv} \geq 0 \qquad \text{per ogni } (u, v) \in E. \tag{7.8}$$

Il primo vincolo (7.5) indica che la somma dei flussi sugli archi uscenti da s è uguale al flusso totale f.

Si ha quindi un vincolo per ogni nodo intermedio (7.6), che rappresenta il fatto che il flusso entrante nel nodo è uguale al flusso uscente (conservazione del flusso). Si può verificare facilmente che i vincoli (7.5) e (7.6) implicano che la somma dei flussi sugli archi entranti in t è uguale al flusso totale f.

Quindi abbiamo i vincoli (7.7) che rappresentano il fatto che il flusso su ogni arco non può eccedere la capacità dell'arco stesso.

Infine ci sono i vincoli di non negatività delle variabili (7.8).

Notiamo che la regione ammissibile di questo problema di PL è non vuota (il vettore di flusso $\bar{x} = 0$ è ovviamente ammissibile) e limitata (per la presenza dei vincoli (7.7) e (7.8)). Quindi, per il teorema fondamentale della programmazione lineare ammette una soluzione ottima e un valore ottimo della funzione obiettivo, che verrà indicato nel seguito con con f^*.

Molte altre considerazioni, sia teoriche sia algoritmiche, potrebbero essere fatte sul problema del massimo flusso a partire dalla sua formulazione come problema di PL. Intanto, per fare un esempio, è evidente che il problema del massimo flusso potrebbe essere risolto con il metodo del Simplesso studiato nella prima parte del corso. Questo non è tuttavia l'algoritmo migliore applicabile in questo caso (notiamo che il problema ha una "struttura" particolare che può e deve essere sfruttata).

In questo capitolo, per semplicità e brevità, noi studieremo le proprietà del problema di massimo flusso e un algoritmo per la sua soluzione a partire dalla sua formulazione come problema su grafi. Questo approccio, se da una parte non mette pienamente in luce i rapporti con la teoria della programmazione lineare, ha il vantaggio di essere più semplice e di permettere una più immediata interpretazione intuitiva dei risultati che via via esporremo.

7.2 Alcuni risultati preliminari

Nel problema del massimo flusso un ruolo centrale è svolto dal concetto di taglio s-t.

Definizione 7.2.1 *Dato un grafo orientato $G = (V, E)$, con $s, t \in V$, e $s \neq t$, si definisce taglio s-t una partizione dei nodi $(W, \bar{W})$ tale che $s \in W$ e $t \in \bar{W}$.*

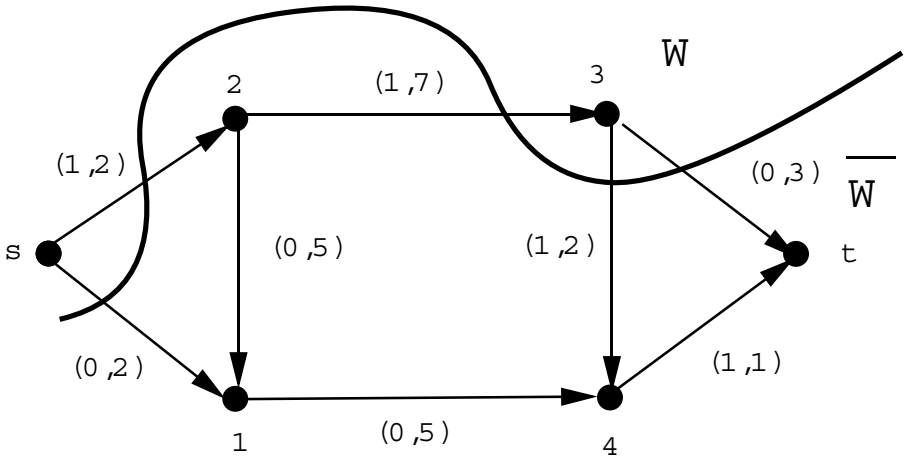


Figura 7.2: Taglio s-t con $W = \{s, 3\}$ e $\bar{W} = \{1, 2, 4, t\}$

Nella figura 7.2 è illustrato un taglio s-t, con $W = \{s, 3\}$ e $\bar{W} = \{1, 2, 4, t\}$. È importante notare come una qualsiasi partizione dei nodi in due classi, tale che s e t appartengano a classi diverse, definisce

un taglio s-t. Ad ogni taglio s-t $(W, \bar{W})$ associamo una *capacità del taglio* $C(W, \bar{W})$ che è pari alla somma delle capacità degli archi che hanno il primo estremo in W e il secondo estremo in $\bar{W}$, e cioè

$$C(W, \bar{W}) = \sum_{u \in W, v \in \bar{W}} c_{uv} \tag{7.9}$$

Notiamo che gli archi che attraversano il taglio in senso opposto, ovvero che abbiano il primo estremo in $\bar{W}$ e il secondo estremo in W, non contribuiscono alla capacità del taglio. Ad esempio, dato il taglio di figura 7.2, gli archi che hanno il primo estremo in W e il secondo estremo in $\bar{W}$ sono gli archi $(s,1),(s,2),(3,4),(3,t)$, e dunque la capacità del taglio $C(W, \bar{W}) = 2+2+2+3 = 9$. Notiamo inoltre che l'arco $(2,3)$ attraversa il taglio in direzione opposta, e cioè va da $\bar{W}$ a W, e dunque la sua capacità non contribuisce alla capacità complessiva del taglio.

Dato un flusso s-t $\bar{x}$, è possibile associare ad ogni taglio s-t $(W, \bar{W})$ un *flusso netto di* $(W, \bar{W})$, che è pari alla somma dei flussi sugli archi uscenti da W ed entranti in $\bar{W}$, meno la somma dei flussi sugli archi uscenti da $\bar{W}$ ed entranti in W:

$$F(W, \bar{W}) = \sum_{u \in W, v \in \bar{W}} \bar{x}_{uv} - \sum_{u \in W, v \in \bar{W}} \bar{x}_{vu}. \tag{7.10}$$

Per il taglio di figura 7.2 è facile verificare che $F(W, \bar{W}) = 1$.

Esiste una semplice relazione tra il flusso netto di un taglio e il valore del flusso, tale relazione è data nel seguente teorema.

Teorema 7.2.2 *Dato un flusso ammissibile $\bar{x}$, il flusso netto $F(W, \bar{W})$ in ogni taglio $(W, \bar{W})$ è pari al valore del flusso $\bar{f}$, cioè*

$$F(W, \bar{W}) = \bar{f}.$$

Dimostrazione. Sommando le equazioni (7.2) di conservazione del flusso relative ai soli nodi dell'insieme $W - \{s\}$ e l'equazione (7.3) $\bar{f} = \sum_{(s,u) \in \omega^+(s)} \bar{x}_{su}$, otteniamo:

$$\sum_{v \in W - \{s\}} \Big(\sum_{(u,v) \in \omega^-(v)} \bar{x}_{uv} - \sum_{(v,k) \in \omega^+(v)} \bar{x}_{vk} \Big) - \sum_{(s,u) \in \omega^+(s)} \bar{x}_{su} = -\bar{f}$$

ovvero, inserendo il termine relativo al nodo s nella doppia sommatoria:

$$\sum_{v \in W} \Big(\sum_{(u,v) \in \omega^-(v)} \bar{x}_{uv} - \sum_{(v,k) \in \omega^+(v)} \bar{x}_{vk} \Big) = -\bar{f}.$$

Notiamo il flusso relativo ad un arco (u,v) con entrambi gli estremi in W compare nella sommatoria per $\omega^+(u)$ con segno negativo e in $\omega^-(v)$ con segno positivo. Pertanto il suo contributo complessivo alla sommatoria è nullo. Dunque gli archi (u,v) che contribuiscono alla sommatoria avranno necessariamente un estremo in W e l'altro estremo in $\bar{W}$, ovvero saranno tali che $u \in W$ e $v \in \bar{W}$, oppure $v \in W$ e $u \in \bar{W}$. Pertanto

$$\sum_{u \in \bar{W}, v \in W} \bar{x}_{uv} - \sum_{u \in W, v \in \bar{W}} \bar{x}_{vu} = -\bar{f},$$

oppure, equivalentemente:

$$F(W, \bar{W}) = \bar{f} \qquad \square$$

Se $W = V - \{t\}$ e $\bar{W} = \{t\}$ abbiamo:

$$F(W, \bar{W}) = \sum_{u \in W, v \in \bar{W}} \bar{x}_{uv} = \sum_{(u,t) \in \omega^-(t)} \bar{x}_{ut} = \bar{f}.$$

Quindi il flusso uscente dalla sorgente s è uguale al flusso netto entrante nel pozzo t.

Passiamo ora ed esaminare le relazioni tra il valore del flusso e le capacità di un taglio. È estremamente semplice provare i seguenti due risultati.

Teorema 7.2.3 *Sia dato un flusso ammissibile $\bar{x}$ in G. Il flusso netto di un taglio qualunque è minore o uguale alla capacità dello stesso taglio. Cioè, se $(W, \bar{W})$ è un taglio, risulta*

$$F(W, \bar{W}) \leq C(W, \bar{W}).$$

Dimostrazione. Siccome il flusso è ammissibile soddisfa, in particolare, le (7.1). Ma allora il teorema segue immediatamente dal confronto delle definizioni di capacità flusso netto di un taglio, cioè dal confronto delle (7.9) e (7.10). □

Notiamo, per inciso, che il flusso netto di un taglio dipende dal flusso $\bar{x}$ che stiamo considerando, mentre la capacità dello stesso taglio è indipendente dal flusso $\bar{x}$ e dipende invece solamente dalle capacità degli archi.

Il flusso massimo f^* è ottenuto in corrispondenza ad un particolare flusso ammissibile, che indichiamo con $\bar{x}^*$; quindi, come caso particolare del precedente teorema otteniamo il seguente risultato.

Teorema 7.2.4 *Il valore del massimo flusso f^* è minore o uguale alla capacità di un qualunque taglio s-t, cioè*

$$f^* \leq C(W, \bar{W})$$

per ogni taglio $(W, \bar{W})$.

Dimostrazione. Per il Teorema 7.2.2 abbiamo, per qualunque taglio $(W, \bar{W})$

$$f^* = F(W, \bar{W}). \tag{7.11}$$

Ma abbiamo anche, per il teorema precedente

$$F(W, \bar{W}) \leq C(W, \bar{W}) \tag{7.12}$$

Dalle (7.11) e (7.12) segue immediatamente

$$f^* \leq C(W, \bar{W}),$$

che è quanto volevamo dimostrare. □

7.3 Cammini aumentanti

Un secondo elemento che gioca un ruolo fondamentale nello studio di problemi di massimo flusso è il concetto di *cammino aumentante*, che è l'oggetto di questa sezione.

Per introdurre la definizione di cammino aumentante abbiamo bisogno di una definizione preliminare

Definizione 7.3.1 *Dato un cammino semplice $P = \{u_0, e_1, u_1 \ldots, e_p, u_p\}$, chiamiamo* arco diretto *un arco di tipo $e_i = (u_{i-1}, u_i)$, mentre chiamiamo* arco inverso *un arco di tipo $e_i = (u_i, u_{i-1})$*[1].

Definizione 7.3.2 *Data una soluzione ammissibile $\bar{x}$ del problema di massimo flusso, definiamo* cammino aumentante *da u_0 a u_p un cammino semplice $P = \{u_0, e_1, \ldots, e_p, u_p\}$ tale che:*

- *per ogni arco diretto $e_i = (u_{i-1}, u_i)$, si ha $c_{u_{i-1}, u_i} > \bar{x}_{u_{i-1}, u_i}$*
- *per ogni arco inverso $e_i = (u_i, u_{i-1})$, si ha $\bar{x}_{u_i, u_{i-1}} > 0$.*

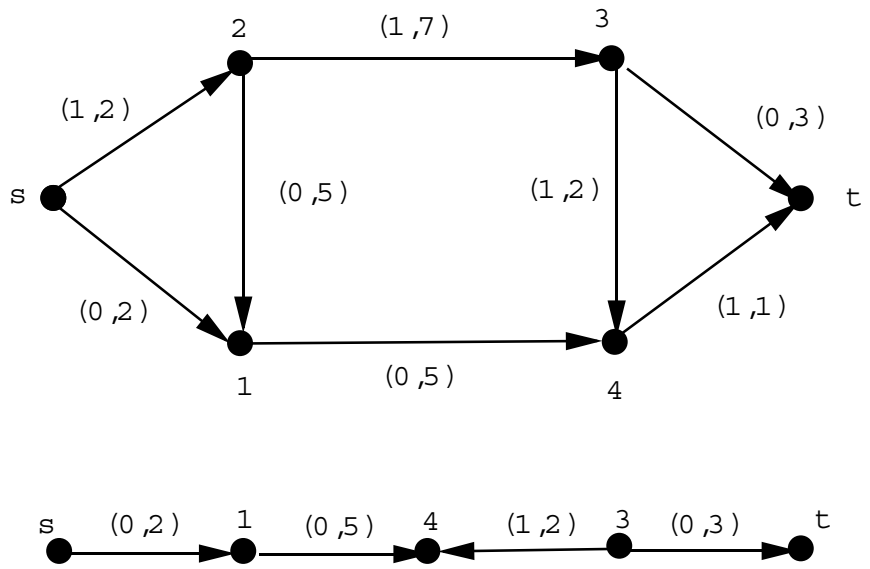


Figura 7.3: Cammino aumentante da s a t

In un cammino aumentante, dunque, gli archi diretti sono *non saturi*, mentre quelli inversi sono *non vuoti*. Consideriamo l'esempio di figura 7.3. Il cammino aumentante $\{s, 1, 4, 3, t\}$ è mostrato in basso in figura. Per ogni arco diretto (nell'esempio gli archi $(s, 1), (1, 4), (3, t)$) , si ha che la capacità è strettamente maggiore del flusso. Per ogni arco inverso (nell'esempio l'arco $(3, 4)$) il flusso è strettamente positivo.

La ragione del nome di cammino aumentante deriva dal fatto che, dato un flusso ammissibile $\bar{x}$, una volta individuato e un cammino aumentante è facilmente definibile un nuovo flusso ammissibile $\tilde{x}$ con un valore di flusso maggiore di quello associato ad $\bar{x}$. In altre parole, sfruttando un cammino aumentante è possibile aumentare il flusso inviato da s a t.

Mostriamo come ciò sia possibile. Supponiamo quindi di avere un flusso ammissibile $\bar{x}$ e un cammino aumentante P. Indichiamo con P_+ l'insieme degli archi diretti del cammino P e con P_- l'insieme di quelli inversi. In modo formale

$$P_+ = \{e_i : e_i \in P, \text{con } e_i \text{ arco diretto}\}$$

e

$$P_- = \{e_i : e_i \in P, \text{ con } e_i \text{ arco inverso}\}.$$

Se consideriamo l'esempio di figura 7.3 abbiamo $P_+ = \{(s1), (12), (3t)\}$ e $P_- = \{(34)\}$.

Sia $\delta_+ = \min_{(u,v)\in P_+}(c_{uv} - \bar{x}_{uv})$. È facile vedere che $\delta_+ > 0$, in quanto per ogni arco diretto si ha $c_{uv} > \bar{x}_{uv}$. Sia $\delta_- = \min_{(u,v)\in P_-} \bar{x}_{uv}$. Anche in questo caso è facile vedere che $\delta_- > 0$, in quanto per ogni arco inverso si ha $\bar{x}_{uv} > 0$. Sia $\delta = \min(\delta_+, \delta_-) > 0$. Definiamo un flusso modificato $\tilde{x}$ nel seguente modo.

$$\tilde{x} = \begin{cases} \tilde{x}_{uv} = \bar{x}_{uv} & \text{se } (u, v) \notin P \\ \tilde{x}_{uv} = \bar{x}_{uv} + \delta & \text{se } (u, v) \in P^+ \\ \tilde{x}_{uv} = \bar{x}_{uv} - \delta & \text{se } (u, v) \in P^- \end{cases}$$

[1]Con linguaggio meno preciso possiamo dire che un arco diretto è un arco il cui verso è concorde con quello di percorrenza del cammino P da u_0 a u_p, mentre un arco inverso ha direzione opposta.

Notiamo che, rispetto a $\bar{x}$ vengono modificati solo i flussi degli archi appartenenti al cammino aumentante P. Vogliamo mostrare che, come preannunciato, $\tilde{x}$ è ammissibile e ha un valore di flusso $\tilde{f}$ maggiore di $\bar{f}$.

Mostriamo dapprima che $\tilde{x}$ è una soluzione ammissibile. Le variabili corrispondenti ad archi non appartenenti a P non vengono modificate rispetto alla soluzione ammissibile $\bar{x}$ e dunque non violano i vincoli di capacità (7.7) e di non negatività (7.8). Le variabili corrispondenti agli archi di P^+ vengono aumentate di δ, quindi i vincoli di non negatività (7.8) non vengono violati. Inoltre, poichè $\delta \leq \delta^+ \leq \min(c_{uv} - \bar{x}_{uv})$, i vincoli di capacità (7.7) non sono violati. Le variabili corrispondenti agli archi di P^- vengono diminuite di δ, quindi i vincoli di capacità (7.7) non sono violati. Inoltre, poichè $\delta \leq \delta^- \leq \min(\bar{x}_{uv})$, i vincoli non negatività (7.8) non vengono violati. Dobbiamo ora mostrare che i vincoli di conservazione ai nodi (7.6) non sono violati. Poichè solo le variabili che corrispondono ad archi di P vengono modificate, i vincoli che potrebbero essere violati sono quelli che corrispondono a nodi di P. Per ogni nodo $u_i \in P$, con $u_i \neq s, t$ ci sono esattamente due archi incidenti in u_i appartenenti a P, l'arco e_{i-1} e l'arco e_i. Sono possibili i seguenti quattro casi:

1. e_{i-1} è diretto ed e_i è inverso su P. In questo caso $\tilde{x}_{e_{i-1}} = \bar{x}_{e_{i-1}} + \delta$, mentre $\tilde{x}_{e_i} = \bar{x}_{e_i} - \delta$. Poichè

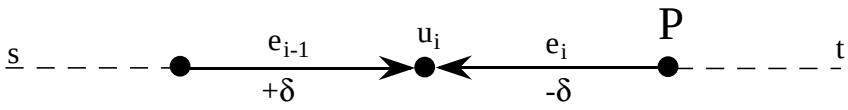


Figura 7.4: e_{i-1} diretto, e_i inverso

   ambedue gli archi sono entranti in u_i, il flusso uscente da u_i resterà invariato, mentre il flusso entrante in u_i varierà di $+\delta - \delta = 0$.

2. e_{i-1} ed e_i sono ambedue diretti su P. In questo caso $\tilde{x}_{e_{i-1}} = \bar{x}_{e_{i-1}} + \delta$, mentre $\tilde{x}_{e_i} = \bar{x}_{e_i} + \delta$.

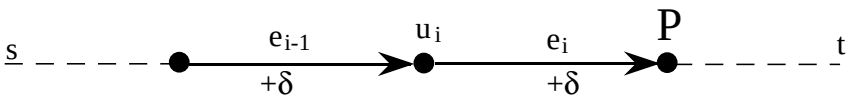


Figura 7.5: e_{i-1} diretto, e_i diretto

   Poichè e_{i-1} è entrante in u_i, mentre e_i è uscente, il flusso entrante aumenta di δ e il flusso uscente aumenta di δ: l'uguaglianza fra flusso entrante e flusso uscente è preservata.

3. e_{i-1} ed e_i sono ambedue inversi su P. In questo caso $\tilde{x}_{e_{i-1}} = \bar{x}_{e_{i-1}} - \delta$, mentre $\tilde{x}_{e_i} = \bar{x}_{e_i} - \delta$.

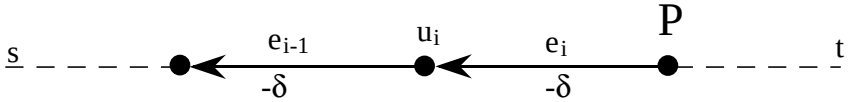


Figura 7.6: e_{i-1} inverso, e_i inverso

Poichè e_{i-1} è uscente da u_i, mentre e_i è entrante, il flusso entrante diminuisce di δ e il flusso uscente diminuisce di δ: l'uguaglianza fra flusso entrante e flusso uscente è preservata.

4. e_{i-1} è inverso ed e_i è diretto su P. In questo caso $\tilde{x}_{e_{i-1}} = \bar{x}_{e_{i-1}} + \delta$, mentre $\tilde{x}_{e_i} = \bar{x}_{e_i} - \delta$. Poichè

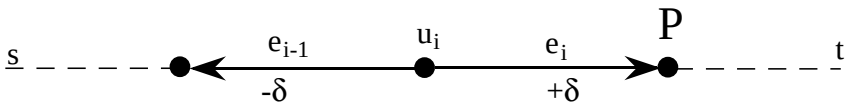


Figura 7.7: e_{i-1} inverso, e_i diretto

ambedue gli archi sono uscenti in u_i, il flusso entrante in u_i resterà invariato, mentre il flusso uscente da u_i varierà di $-\delta + \delta = 0$.

Dunque $\tilde{x}$ è ammissibile. Inoltre, poichè il primo arco di P è sempre diretto ed uscente dal nodo s, il flusso totale uscente dal nodo s viene aumentato di δ e risulta quindi

$$\tilde{f} = \bar{f} + \delta.$$

7.4 Condizioni di ottimalità

In questa sezione ci proponiamo di trovare delle condizioni che, dato un flusso ammissibile $\bar{x}$, ci permettano di capire se $\bar{x}$ sia una soluzione ottima del problema di massimo flusso o meno. La prima condizione che consideriamo è legata all'esistenza di cammini aumentanti ed ha implicazioni algoritmiche immediate. La seconda, equivalente, coinvolge la capacità dei tagli ed è interessante perché mostra come risolvendo un problema di massimo flusso possiamo contemporaneamente risolvere un altro problema apparentemente molto diverso: il "problema del minimo taglio".

Passiamo ora a considerare la prima delle due condizioni preannunciate.

Teorema 7.4.1 *Un flusso s-t $\bar{x}$ di valore $\bar{f}$ è massimo se e solo se non esiste nessun cammino aumentante $P = \{s = u_0, e_1, \ldots, u_p = t\}$ da s a t in G rispetto a $\bar{x}$.*

Dimostrazione.
Necessità. Se $\bar{x}$ è ottimo non possono esistere cammini aumentanti da s a t in G rispetto a $\bar{x}$. Infatti, se ne esistesse uno sappiamo (vedi sezione precedente) che è possibile trovare un nuovo flusso $\tilde{x}$ con un valore maggiore di $\bar{f}$. Ciò è assurdo perché $\bar{x}$ è ottimo.
Sufficienza. Consideriamo il seguente taglio:

$$W = \{u : \text{esiste un cammino aumentante da } s \text{ a } u \text{ rispetto ad } \bar{x}\} \cup \{s\}$$
$$\bar{W} = V - W$$

Notiamo che deve risultare $t \notin W$. Infatti, se risultasse $t \in W$ esisterebbe, per definizione di W, un cammino aumentante da s a t. Ma questo implicherebbe, per quanto visto nella sezione precedente, che sarebbe possibile trovare un nuovo flusso $\tilde{x}$ con un valore di flusso $\tilde{f}$ maggiore di $\bar{f}$. Siccome questo è assurdo perchśtiamo supponendo che $\bar{f}$ sia massimo, deve essere, come affermato $t \notin W$.

Quindi $(W, \bar{W})$ è un taglio s-t. Consideriamo adesso il flusso sugli archi che attraversano il taglio da W a $\bar{W}$ e sugli archi che attraversano il taglio da $\bar{W}$ a W.

- Per ogni $(u, v) \in E$ tale che $u \in W$, e $v \in \bar{W}$ si ha che il flusso $\bar{x}_{uv} = c_{uv}$. Infatti, supponiamo per assurdo che $\bar{x}_{uv} < c_{uv}$. Essendo $u \in W$ esiste un cammino aumentante P' da s ad u. Se $\bar{x}_{uv} < c_{uv}$, allora il cammino $P = P' \cup (u, v)$ è un cammino aumentante da s a v. Infatti lungo l'arco diretto (u, v) si verifica la condizione $\bar{x}_{uv} < c_{uv}$. Ma allora v dovrebbe appartenere a W contraddicendo l'ipotesi che $v \in \bar{W}$.
- Per ogni $(v, u) \in E$ tale che $u \in W$, e $v \in \bar{W}$ si ha che il flusso $\bar{x}_{vu} = 0$. Infatti, supponiamo per assurdo che $\bar{x}_{vu} > 0$. Essendo $u \in W$ esiste un cammino aumentante P' da s ad u. Se $\bar{x}_{vu} > 0$, allora il cammino $P = P' \cup (u, v)$ è un cammino aumentante da s a v. Infatti lungo l'arco inverso (v, u) si verifica la condizione $\bar{x}_{vu} > 0$. Ma allora v dovrebbe appartenere a W contraddicendo l'ipotesi che $v \in \bar{W}$.

Per il Teorema (7.2.2), il flusso totale $\bar{f}$ da s a t è pari al flusso attraverso il taglio $(W, \bar{W})$ che vale, per quanto appena visto,

$$\bar{f} = \sum_{u \in W, v \in \bar{W}} \bar{x}_{uv} = \sum_{u \in W, v \in \bar{W}} c_{uv} = C(W, \bar{W}).$$

Possiamo allora scrivere

$$\bar{f} = C(W, \bar{W}).$$

Ma per il Teorema 7.2.4 sappiamo che una qualunque distribuzione di flusso ammissibile non può generare un flusso totale maggiore di $C(W, \bar{W})$ e quindi il flusso $\bar{f}$ è massimo. □

Nel paragrafo successivo vedremo come questo teorema sia alla base di un metodo per il calcolo del massimo flusso. Prima di occuparci di questo, però, vediamo come il teorema appena visto possa essere riformulato in un modo che metta in luce le connessioni tra il problema di trovare il massimo flusso e il problema di trovare il tagli a capacità minima.

Per introdurre questo risultato facciamo delle considerazioni preliminari. Nella seconda sezione di questo capitolo abbiamo associato ad ogni taglio $(W, \bar{W})$ un ben definito numero: la sua capacità $C(W, \bar{W})$. Siccome il numero di tagli possibile è finito (anche se, eventualmente, molto grande) possiamo considerare tra tutti i tagli quello che ha capacità minore di tutti gli altri (ovviamente possono esistere più tagli diversi che forniscono la stessa capacità minima). Indichiamo con C_{min} il valore di questa capacità minima. Il Teorema 7.2.4 è valido per qualunque taglio, in particolare per il taglio di capacità minima. Quindi possiamo affermare che

$$f^* \leq C_{min}$$

La capacità del taglio minimo costituisce quindi un limite superiore per il valore del flusso massimo, e se troviamo un flusso $\bar{x}$ per cui $\bar{f} = C_{min}$, $\bar{x}$ è ovviamente un flusso ottimo[2].

In generale però, il valore del flusso massimo f^* potrebbe risultare inferiore o uguale a C_{min}. Il teorema del Massimo flusso - Minimo taglio (più noto, con terminologia inglese, come Max flow - Min cut theorem) stabilisce invece che un flusso $\bar{x}$ è ottimo se e solo se il valore del flusso è esattamente C_{min}

Teorema 7.4.2 (Massimo flusso - Minimo taglio) *Una distribuzione di flusso è ottima se e solo se risulta*

$$\bar{f} = C_{min}$$

dove $\bar{f}$ è il valore di flusso di $\bar{x}$.

[2]Cambiando "punto di vista" possiamo anche dire che il flusso massimo è un limite inferiore per la capacità di un taglio. Quindi, se per qualche taglio $(W, \bar{W})$ risulta $C(W, \bar{W}) = f^*$, $(W, \bar{W})$ è un taglio minimo e $C(W, \bar{W}) = C_{min}$. Questo fatto verrà usato spesso nel seguito.

Dimostrazione.
Sufficienza. Mostriamo che se $\bar{f} = C_{min}$ allora $\bar{x}$ è ottimo. Procediamo per assurdo. Supponiamo che risulti $\bar{f} = C_{min}$, ma $\bar{x}$ non sia ottimo. Allora esiste un altro flusso, $\tilde{x}$ il cui valore di flusso è maggiore di $\bar{f}$. Possiamo quindi scrivere

$$\tilde{f} > \bar{f} = C_{min}$$

che è ovviamente assurdo perchè contraddice il Teorema 7.2.4.
Necessità. Supponiamo che $\bar{x}$ sia ottimo e mostriamo che deve risultare $\bar{f} = C_{min}$ Procediamo per assurdo e supponiamo che sia $\bar{f} \neq C_{min}$. Più in particolare, siccome sappiamo che non può essere $\bar{f} > C_{min}$ per il Teorema 7.2.4, deve risultare

$$\bar{f} < C_{min}. \tag{7.13}$$

Definiamo ora lo stesso taglio considerato nel teorema precedente. Poniamo cioè $W = \{u$: esiste un cammino aumentante da s a u rispetto ad $\bar{x}\} \cup \{s\}$ e $\bar{W} = V - W$.

Ragionando come nella prova del teorema precedente, abbiamo che $(W, \bar{W})$ è un taglio s-te che

$$\bar{f} = C(W, \bar{W}). \tag{7.14}$$

Per le (7.13) e (7.14) possiamo allora scrivere

$$C(W, \bar{W}) \geq C_{min} > \bar{f} = C(W, \bar{W})$$

che è ovviamente assurda e così la prova del teorema è completa. □

Nella dimostrazione della necessità del teorema appena dimostrato, abbiamo anche implicitamente trovato un modo per determinare un taglio di capacità minima. Infatti, se x^* è un flusso ottimo, il taglio definito da

$$W = \{u : \text{esiste un cammino aumentante da } s \text{ a } u \text{ rispetto ad } \bar{x}\} \cup \{s\}$$
$$\bar{W} = V - W$$

è tale che $f^* = C(W, \bar{W})$ (equazione (7.14)) ed è quindi minimo per il Teorema (7.2.2). Quindi, se sappiamo risolvere un problema di massimo flusso, sappiamo anche risolvere il problema di trovare un taglio di capacità minima!

Il fatto che il massimo flusso sia minore o uguale alla capacità di ogni taglio e quindi anche alla capacità del taglio minimo è abbastanza ovvio. Meno ovvio è che all' ottimo valga sempre l'uguaglianza. Questa proprietà permette, concettualmente, di verificare se un dato flusso (ammissibile) è ottimo. Infatti per fornire la prova di ottimalità basta esibire un taglio con capacità pari al flusso inviato da s a t. Nel caso tale taglio non esista, sappiamo che deve esistere un flusso che permette di inviare un flusso maggiore da s a t.

Il teorema del Massimo flusso - Minimo taglio mette quindi in relazione un problema sostanzialmente continuo (trovare il massimo flusso inviabile da s a t) con un problema intrinsecamente combinatorio (trovare il taglio di minima capacit che separa s a t). I due problemi sono di natura diversa. In particolare, il primo cerca una soluzione in un insieme infinito di possibili soluzioni. Il secondo cerca una soluzione in un insieme finito di possibili soluzioni. Questo fatto potrebbe portare all'impressione (errata) che il primo problema sia più difficile del secondo, in quanto nel secondo è sempre possibile enumerare tutte le possibili soluzioni e scegliere la migliore. Questa impressione è, però, come ormai ben sappiamo, errata, in quanto l'insieme di tutte le soluzioni è sì finito, ma talmente grande, anche per grafi di piccole dimensioni, che non è pensabile esaminarlo tutto. Per esempio, un grafo con 300 nodi (piccolo per le applicazioni normalmente affrontate con questa metodologia) porta a 2^{288} possibili tagli separatori di due nodi assegnati. Alcuni ricercatori stimano che il numero di atomi dell'universo osservabile sia inferiore a 2^{263}.

Prima di chiudere questa sezione, vogliamo dare una formulazione equivalente del teorema del Massimo flusso - Minimo taglio che può aiutare a chiarire ulteriormente le relazioni tra i vari concetti introdotti.

7.5 L'algoritmo di Ford e Fulkerson.

I precedenti risultati ci permettono di definire un algoritmo per il calcolo del flusso massimo da s a t in un grafo orientato. Sia dato un flusso $\bar{x}$ ammissibile. Per il Teorema 7.4.1, se non esiste in G un cammino aumentante da s a t rispetto a $\bar{x}$, allora $\bar{x}$ è ottimo. Supponiamo invece che esista un cammino aumentante $P = \{s = u_0, e_1, \ldots, u_p = t\}$. Il flusso $\bar{x}$ non è ottimo, e quindi può essere "aumentato", nel senso che è possibile modificare $\bar{x}$ in modo da aumentare il flusso totale inviato da s a t come visto nella sezione "Cammini aumentanti".

Viene allora spontaneo definire un algoritmo per il calcolo del massimo flusso nel seguente modo.

Algoritmo di Ford e Fulkerson

Inizializzazione Poni $x^0 := 0$; $i := 0$.

Passo i-esimo Cerca un cammino aumentante da s a t rispetto al flusso x^i.

Se non esiste: STOP. x^i è la soluzione ottima.

Altrimenti poni:

$$x^{i+1} = \begin{cases} x^{i+1}_{uv} = x^i_{uv} & \text{se } (u,v) \notin P \\ x^{i+1}_{uv} = x^i_{uv} + \delta & \text{se } (u,v) \in P^+ \\ x^{i+1}_{uv} = x^i_{uv} - \delta & \text{se } (u,v) \in P^- \end{cases}$$

Poni $i := i + 1$; vai al passo i.

Notiamo subito che in questo algoritmo c'è una parte non completamente specificata. Infatti all'inizio del Passo i viene detto "Cerca un cammino aumentante...", ma non viene detto né come trovare un tale cammino aumentante né quale scegliere nel caso ne esista più di uno. Queste questioni verranno affrontate nella prossima sezione. Per quel che ci interessa qui, supponiamo di essere in grado di trovare i cammini aumentanti e, nel caso ne esista più di uno, di sceglierne uno a caso.

Per chiarire meglio il procedimento illustriamolo subito con un esempio. Consideriamo di nuovo il grafo di figura 7.2. Al passo 0 avremo $x^0_{s1} = 0, x^0_{s2} = 0, x^0_{14} = 0$, $x^0_{21} = 0, x^0_{23} = 0, x^0_{34} = 0, x^0_{3t} = 0, x^0_{4t} = 0$. Un possibile cammino aumentante sarà: $P = \{s, (s,2), 2, (2,3), 3, (3,4), 4, (4,t), t\}$. $P_+ = \{(s,2), (2,3), (3,4), (4,t)\}$, mentre $P_- = \emptyset$. Quindi il primo cammino aumentante possiede solo archi diretti. $\delta = \delta_+ = \min(2-0, 7-0, 2-0, 1-0) = 1$. Quindi tutte le variabili relative agli archi del cammino aumentante vengono aumentate di una unità. Le altre resteranno invariate. Si avrà: $x^1_{s1} = 0, x^1_{s2} = 1, x^1_{14} = 0$, $x^1_{21} = 0, x^1_{23} = 1, x^1_{34} = 1, x^1_{3t} = 0, x^1_{4t} = 1$. Il flusso è aumentato da 0 a 1.
Un nuovo cammino aumentante sarà $P = \{s, (s,1), 1, (1,4), 4, (3,4), 3, (3,t), t\}$. Risulta quindi $P_+ = \{(s,1), (1,4), (3,4), (3,t)\}$ e $P_- = \{(3,4)\}$. Quindi $\delta_+ = \min(2-0, 5-0, 3-0) = 2$, mentre $\delta_- = \min(1) = 1$ e $\delta = \min(\delta_+, \delta_-) = 1$.

Le variabili corrispondenti ad archi diretti vengono quindi aumentate di una unità, mentre le variabili corrispondenti ad archi inversi (in questo caso la variabile x_{34}) vengono diminuite di un'unità. $x^2_{s1} = 1, x^2_{s2} = 1, x^2_{14} = 1$, $x^2_{21} = 0, x^2_{23} = 1, x^2_{34} = 0, x^2_{3t} = 1, x^2_{4t} = 1$. Il flusso è aumentato da 1 a 2.

Infine troviamo il cammino aumentante: $P = \{s, (s,2), 2, (2,3), 3, (3,t), t\}$. Sono tutti archi diretti e per essi vale $\delta = 1$. Quindi si avrà:

$x^3_{s1} = 1, x^3_{s2} = 2, x^3_{14} = 1$, $x^3_{21} = 0, x^3_{23} = 2, x^3_{34} = 0, x^3_{3t} = 2, x^3_{4t} = 1$. Il flusso finale vale 3, cioè la somma dei flussi uscenti da s. Per il teorema 7.4.2, per dimostrare l'ottimalità del flusso basta mostrare un taglio s-$t$la cui capacità è pari al valore del flusso. Bisogna cioè trovare il taglio a capacità minima.

Nell'esempio il taglio a capacità minima sarà $W = \{s, 1, 4\}$ e $\bar{W} = \{2, 3, t\}$, ed è illustrato in figura 7.8. Gli archi che hanno la coda in W e la testa in $\bar{W}$ sono gli archi $(s,2)$ e $(4,t)$, e la somma delle loro capacità vale 3. Per trovare questo taglio basta usare la costruzione suggerita nella dimostrazione della sufficienza del teorema 7.4.1, ovvero basta mettere in W il nodo s e tutti i nodi raggiungibilli da s con un cammino aumentante rispetto alla soluzione ottima $\bar{x}$. Nell'esempio appena visto l'algoritmo di Ford e Fulkerson

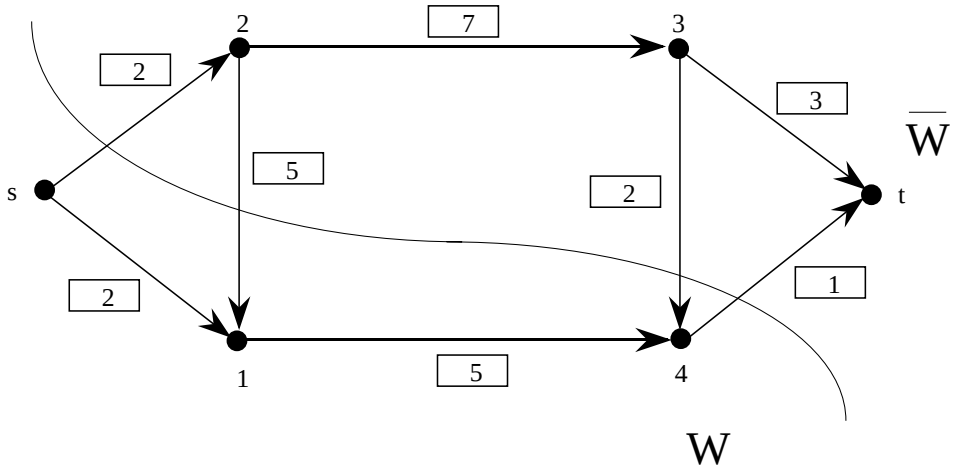


Figura 7.8: Taglio a capacità minima

ha trovato la soluzione ottima in un numero finito di iterazioni. Ma quale garanzia abbiamo che ciò accada sempre? Più in particolare, se non possiamo trovare un cammino aumentante sappiamo di essere all'ottimo, mentre in caso contrario sappiamo che l'algoritmo aumenta il valore del flusso. Ma, per esempio, possiamo pensare di poter trovare cammini aumentanti con i quali è possibile aumentare il valore del flusso di quantità δ sempre più piccole cosicché se, poniamo, il flusso ottimo vale 10, i nostri flussi abbiano valori di 9, 9.9, 9.99, 9.999 ... e non si arrivi mai al flusso ottimo.

In proposito vale il seguente risultato.

Teorema 7.5.1 *Supponiamo che tutte le capacità siano numeri interi. Allora l' algoritmo di Ford e Fulkerson trova l'ottimo in un numero finito di passi.*

Dimostrazione. Sappiamo che il flusso ottimo esiste sempre e ha quindi una valore (finito) f^*. Il flusso da cui parte l' algoritmo è quello nullo, che ha un valore di 0. Siccome le capacità sono tutte intere è facile convincersi che ad ogni passo il flusso viene aumentato di un numero intero (δ è un numero intero). Quindi ad ogni iterazione il flusso viene aumentato di almeno un'unità (δ è maggiore o uguale a 1, il più piccolo intero positivo). Questo vuol dire che in un numero finito di passi (pari, nel caso peggiore, al più piccolo intero superiore a f^*) l'algoritmo trova (= "raggiunge") l'ottimo. □

Con ragionamenti molto simili si può dimostrare che anche se le capacità sono numeri razionali[3] l'algoritmo termina in un numero finito di passi. Invece se alcune delle capacità sono numeri irrazionali è possibile fornire degli esempi (piuttosto artificiosi in verità) in cui l'algoritmo non riesce a trrovare la soluzione ottima in un numero finito di passi.

È importante sapere che esistono delle semplici varianti dell' algoritmo di Ford e Fulkerson che garantiscono la convergenza in un numero finito di passi qualunque siano i valori delle capacità

7.6 Calcolo e scelta dei cammini aumentanti

In questa sezione affrontiamo la questione che era stata lasciata in sospeso nella descrizione dell'algoritmo di Ford e Fulkerson: come possiamo, in maniera sistematica, trovare un cammino aumentante o, nel caso

[3]Ricordiamo che i numeri razionali sono quelli esprimibili come rapporto di due numeri interi. Esempi di numeri irrazionali sono invece π e $\sqrt{2}$.

non ne esista uno, trovare un taglio minimo? A questo fine introduciamo un semplice algoritmo noto come *algoritmo di etichettatura*. L'algoritmo assegna ad ogni nodo un'etichetta a partire da s fino a raggiungere, se possibile, t. Se riesce effettivamente a raggiungere t è poi possibile ricostruire un cammino aumentante a partire dalle etichette; se invece non riesce a raggiungere t allora non esistono cammini aumentanti e l'insieme dei nodi etichettati e quello dei nodi non etichettati costituiscono un taglio di capacità minima.

Ogni etichetta è costituita da due numeri; la generica etichetta del nodo i verrà indicata con $(E_1(i), E_2(i))$, dove $E_1(i)$ è il primo numero dell'etichette e $E_2(i)$ il secondo.

Sia allora $\bar{x}$ un flusso s-t ammissibile. L' algoritmo procede nel seguente modo.

Assegna alla sorgente l'etichetta $(0, \infty)$. Poi seguita ad etichettare i nodi, finché possibile, secondo le seguenti regole.

(a) Se

- i è un nodo etichettato,
- j è un nodo non etichettato,
- esiste l'arco (i, j),
- risulta $\bar{x}_{ij} < c_{ij}$,

assegna a j l'etichetta $(i, \min\{c_{ij} - \bar{x}_{ij}, E_2(i)\})$.

(b) Se

- i è un nodo etichettato,
- j è un nodo non etichettato,
- esiste l'arco (j, i),
- risulta $\bar{x}_{ji} > 0$,

assegna a j l'etichetta $(i, \min\{x_{ji}, E_2(i)\})$.

Commenti. A un certo stadio dell'algoritmo ci possono essere più modi in cui un nodo non etichettato può essere etichettato seguendo le regole (a) e (b). In questo caso basta scegliere uno qualsiasi dei modi possibili e proseguire. Per esempio consideriamo la parte di un grafo parzialmente etichettato, mostrata in figura 7.9a). Il nodo 9 può essere etichettato indifferentemente a partire dal nodo 5, 7.9b), o dal nodo 11, 7.9c).

È importante tener presente che una volta attribuita un' etichetta a un nodo questa non va più cambiata.

La procedura termina quando o viene etichettato il pozzo t o non è più possibile etichettare nessun nodo.

Significato delle etichette Le etichette indicano un possibile cammino aumentante dalla sorgente s al nodo etichettato. Per ricostruire il cammino basta usare le $E_1(i)$ e leggerle all'indietro fino ad arrivare alla sorgente s. È infatti facile convincersi, tenendo presente le regole (a) e (b) adottate nel processo di etichettatura, che il cammino dal nodo s al nodo i costruito in questo modo è in effetti un cammino aumentante e che il flusso aggiuntivo che è possibile inviare da s a i è esattamente pari a $E_2(i)$.

Quindi se etichettiamo il nodo t abbiamo trovato un cammino aumentante e possiamo procedere con l'algoritmo di Ford e Fulkerson.

Consideriamo invece il caso in cui l'algoritmo di etichettatura si fermi, senza aver etichettato il pozzo t, perché non è più possibile etichettare nessun nodo. Definiamo W come l'insieme dei nodi che sono stati etichettati e indichiamo con $\bar{W}$ i rimanenti nodo del grafo. Risulta ovviamente $s \in W$, per costruzione, e $t \in \bar{W}$ perché abbiamo supposto che il pozzo non sia stato etichettato. $(W, \bar{W})$ costituisce quindi un taglio s-t.

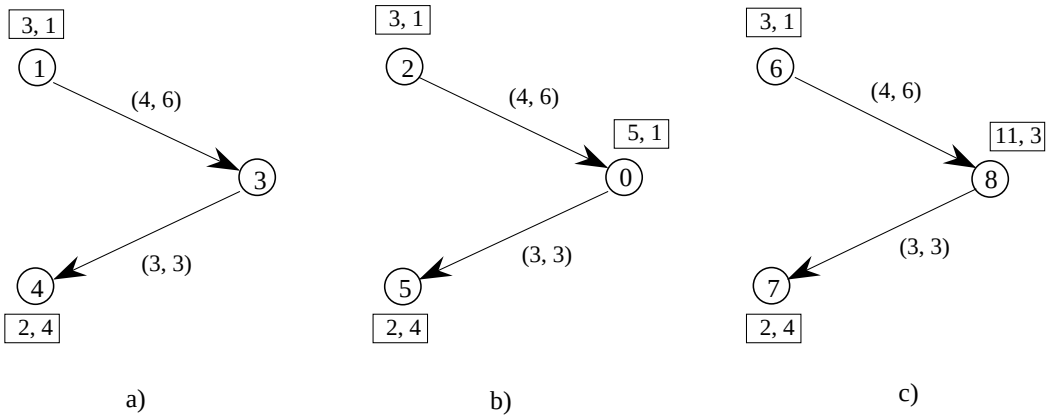


Figura 7.9: Esempio di etichettatura

Calcoliamo la capacità di questo taglio. Poiché l'algoritmo di etichettatura si è fermato senza essere in grado di etichettare nessun nodo in $\bar{W}$ abbiamo, per le regole (a) e (b), che tutti gli archi (i, j) che attraversano il taglio sono o saturi ($x_{ij} = c_{ij}$), se $i \in W$ e $j \in \bar{W}$, o vuoti ($x_{ij} = 0$) se $i \in \bar{W}$ e $j \in W$. Ma allora, tenendo presente il Teorema 7.2.2 e le definizioni di capacità e flusso netto di un taglio, abbiamo

$$\bar{f} = F(W, \bar{W}) = C(W, \bar{W}).$$

Quindi $(W, \bar{W})$ è un taglio minimo e $\bar{x}$ è ottimo.

Notiamo che l'algoritmo di etichettatura proposto trova, se ne esiste almeno uno, un cammino aumentante qualunque. Ovviamente il comportamento dell'algoritmo di Ford e Fulkerson sarà diverso se si scelgono cammini aumentanti diversi. In particolare la scelta del cammino aumentante da usare ad ogni iterazione può avere una influenza critica sul numero di iterazioni necessario per raggiungere l'ottimo. A questo scopo può essere istruttivo considerare il semplice esempio riportato nella figura 7.10 dove i numeri sugli archi indicano le capacità. Applichiamo l'algoritmo di Ford e Fulkerson. Se scegliamo i cammini

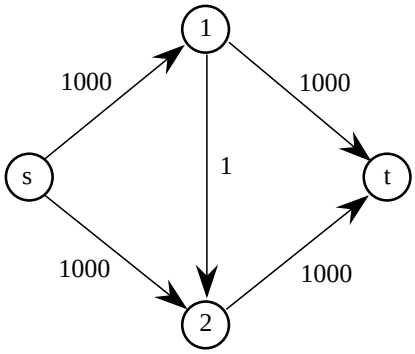


Figura 7.10: Importanza della scelta dei cammini aumentanti

aumentanti $((s, 1), (1, t))$ e $((s, 2), (1, t))$ si vede facilmente che l'algoritmo trova l'ottimo in due sole iterazioni. Se invece scegliamo come cammini aumentanti i cammini $((s, 1), (1, 2), (2, t))$, $((s, 2), (2, 1), (1, t))$,

$((s,1),(1,2),(2,t))$, $((s,2),(2,1),(1,t))$, ..., si vede che sono necessarie 2000 iterazioni, perchè attraverso ogni cammino aumentato indicato si può aumentare il valore del flusso di una sola unità alla volta.

Esistono dei modi "intelligenti" di scegliere i cammini aumentanti che assicurano che situazione disastrose come quella evidenziata nell'esempio appena visto non possono accadere. Non abbiamo qui lo spazio per approfondire l' argomento. Diciamo solo che due "buone" scelte sono quelle di scegliere ad ogni passo, tra tutti i cammini aumentanti possibili, un cammino composto dal minore numero possibili di archi o un cammino che assicuri di poter aumentare il valore del flusso della più grande quantità possibile. Notiamo che usando uno qualunque di questi due criteri nell' esempio della figura 7.10 porta alla scelta dei cammini $((s,1),(1,t))$ e $((s,2),(1,t))$ e fa quindi terminare l'algoritmo di Ford e Fulkerson in 2 sole iterazioni.

Esistono dei modi efficienti e esistematici, diversi dalla procedura di etichettatura illustrata in questa sezione, per calcolare cammini aumentanti che soddisfino uno dei due criteri appena indicati. Qui, per ragioni di spazio, diciamo solo che sono basati sulle tecniche di calcolo dei cammini minimi del tipo illustrato nel capitolo precedente.

7.7 Esempi

Dato il grafo di figura 7.11, trovare il valore del massimo flusso inviabile da s a t (la capacità associata a ciascun arco è mostrata in figura in prossimità dell'arco stesso).

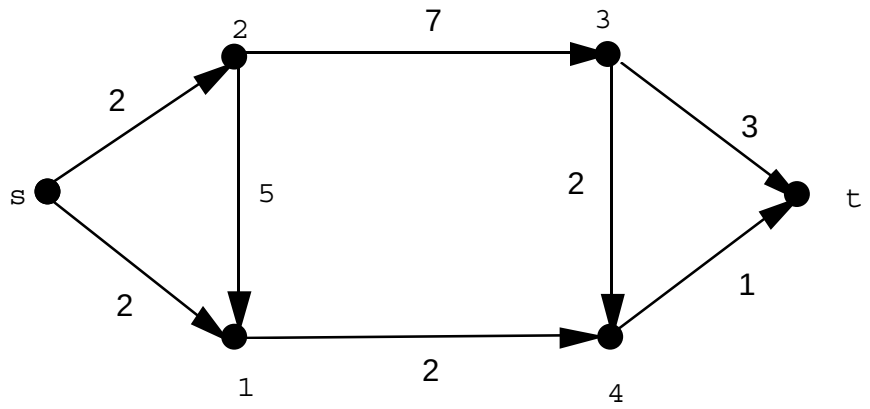


Figura 7.11:

Soluzione.
Per calcolare il flusso utilizzeremo l'algoritmo di Ford e Fulkerson, con flusso iniziale nullo. Ricordiamo infatti che l'algoritmo di Ford e Fulkerson richiede in ingresso un flusso ammissibile, e che il flusso nullo è un flusso ammissibile per il nostro problema.

- *Iterazione 0.*

  Al passo 0 avremo $x^0_{s1} = 0$, $x^0_{s2} = 0$, $x^0_{14} = 0$, $x^0_{21} = 0$, $x^0_{23} = 0$, $x^0_{34} = 0$, $x^0_{3t} = 0, x^0_{4t} = 0$. Un possibile cammino aumentante da s a t sarà: $P = \{s, (s,2), 2, (2,3), 3, (3,4), 4, (4,t), t\}$. $P_+ = \{(s,2), (2,3), (3,4), (4,t)\}$, mentre $P_- = \emptyset$. Ci sono solo archi diretti e dunque $\delta = \delta_+ = \min(2-0, 7-0, 2-0, 1-0) = 1$. Quindi tutte le variabili relative agli archi del cammino aumentante vengono aumentate di un'unità. Le altre resteranno invariate. Si avrà: $x^1_{s1} = 0, x^1_{s2} = 1, x^1_{14} = 0$, $x^1_{21} = 0, x^1_{23} = 1$, $x^1_{34} = 1, x^1_{3t} = 0, x^1_{4t} = 1$. Il flusso è aumentato da 0 a 1.

- *Iterazione 1.*

  Un nuovo cammino aumentante sarà: $P = \{s, (s,1), 1, (1,4), 4, (3,4), 3, (3,t), t\}$.

  Risulta $P_+ = \{(s,1), (1,4), (3,t)\}$, mentre $P_- = \{(3,4)\}$. Inoltre $\delta_+ = \min(2-0, 2-0, 3-0) = 2$, mentre $\delta_- = \min(1) = 1$. Quindi $\delta = \min(\delta_+, \delta_-) = 1$.

  Le variabili corrispondenti ad archi diretti vengono quindi aumentate di un'unità, mentre le variabili corrispondenti ad archi inversi (in questo caso la variabile x_{34}) vengono diminuite di un'unità. Sarà dunque: $x^2_{s1} = 1, x^2_{s2} = 1, x^2_{14} = 1$, $x^2_{21} = 0, x^2_{23} = 1, x^2_{34} = 0, x^2_{3t} = 1, x^2_{4t} = 1$. Il flusso è aumentato da 1 a 2.

- *Iterazione 2.*

  Infine troviamo il cammino aumentante: $P = \{s, (s,2), 2, (2,3), 3, (3,t), t\}$. Sono tutti archi diretti e per essi vale $\delta = 1$. Quindi si avrà:

  $x^3_{s1} = 1, x^3_{s2} = 2, x^3_{14} = 1$, $x^3_{21} = 0, x^3_{23} = 2, x^3_{34} = 0, x^3_{3t} = 2, x^3_{4t} = 1$.
  Il flusso finale vale 3.

- *Iterazione 3.*

  Non è possibile trovare un cammino aumentante rispetto al flusso x^3, che è dunque ottimo. □

Ovviamente, se è noto un flusso iniziale ammissibile, diverso da 0, possiamo tranquillamente applicare l'algoritmo a partire dal flusso dato. Consideriamo anche un'altro esempio.

Dato il grafo e il flusso ammissibile rappresentati in Fig. 7.12, trovare il flusso massimo iniviabile da s a t, certificando l'ottimalità della soluzione ottenuta con l'esibizione di un taglio di capacità uguale al flusso massimo trovato.

Soluzione.

Per trovare il flusso massimo utilizzeremo l'algoritmo di Ford e Fulkerson, con flusso iniziale uguale a quello in figura. Il flusso iniziale vale 10.

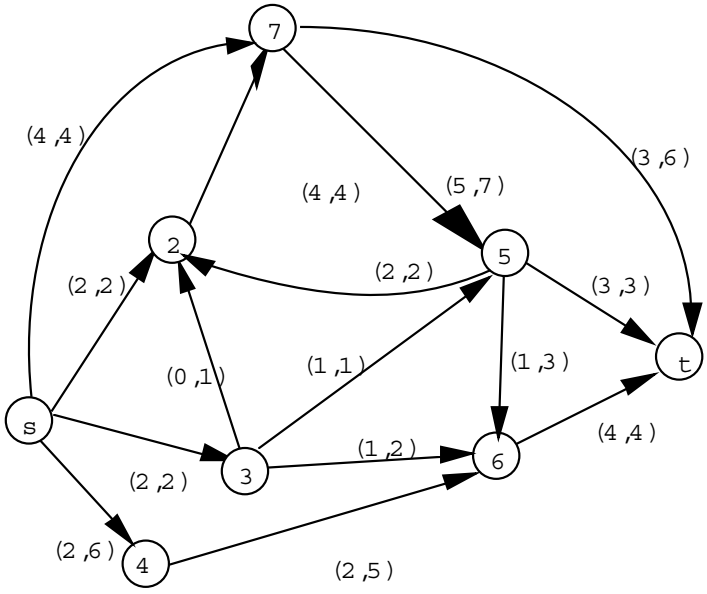


Figura 7.12:

- *Iterazione 0.*

  Al passo 0 avremo $x^0_{s2} = 2$, $x^0_{s3} = 2$, $x^0_{s4} = 2$, $x^0_{s7} = 4$, $x^0_{27} = 4$, $x^0_{32} = 0$, $x^0_{35} = 1$, $x^0_{36} = 1$, $x^0_{46} = 2$ $x^0_{52} = 2$ $x^0_{56} = 1$ $x^0_{5t} = 3$ $x^0_{6t} = 4$ $x^0_{75} = 5$ $x^0_{7t} = 3$. Un possibile cammino aumentante da s a t sarà: $P = \{s, (s,4), 4, (4,6), 6, (5,6), 5, (7,5), 7, (7,t), t\}$. $P_+ = \{(s,4), (4,6), (7,t)\}$, mentre $P_- = \{(5,6), (7,5)\}$. $\delta_+ = \min(4,3,3) = 3$, mentre $\delta_- = \min(1,3) = 1$. Quindi $\delta = \min(\delta_+, \delta_-) = 1$. Si avrà quindi (scriviamo solo le variabili da aggiornare: $x^1_{s4} = 3, x^1_{46} = 3, x^1_{56} = 0$, $x^1_{75} = 4, x^1_{7t} = 4$. Il flusso è aumentato da 10 a 11.

- *Iterazione 1.*

  Un nuovo cammino aumentante sarà: $P = \{s, (s,4), 4, (4,6), 6, (3,6), 3, (3,2), 2, (5,2), 5, (7,5), 7, (7,t), t\}$. $P_+ = \{(s,4), (4,6), (3,2), (7,t)\}$, mentre $P_- = \{(3,6), (5,2), (7,5)\}$. $\delta_+ = \min(3,2,1,2) = 1$, mentre $\delta_- = \min(1,2,2) = 1$. Quindi $\delta = \min(\delta_+, \delta_-) = 1$. Si avrà: $x^2_{s4} = 4$, $x^2_{46} = 4$, $x^2_{36} = 0$, $x^2_{32} = 1$, $x^2_{52} = 1$, $x^1_{75} = 5, x^1_{7t} = 5$. Il flusso è aumentato da 11 a 12.

- *Iterazione 2.*

  Non è possibile trovare un cammino aumentante rispetto a x^2, che è dunque ottimo.

Un modo per rappresentare graficamente le iterazioni dell'algoritmo è illustrato in Figura 7.13. Ogni volta che il valore del flusso su un arco viene aggiornato, si sbarra il precedente valore del flusso sovrapponendogli il nuovo valore.

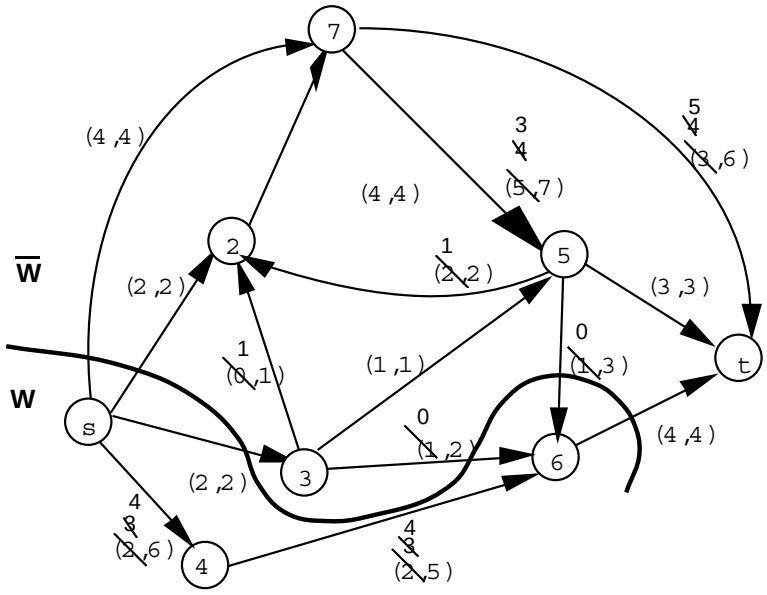


Figura 7.13:

Nell'esempio un taglio a capacità minima sarà $W = \{s, 4, 6\}$ e $\bar{W} = \{2, 3, 5, 7, t\}$, ed è mostrato in figura 7.13 con una linea che separa i due insiemi. Per calcolare la capacità del taglio bisogna sommare le capacità degli archi che hanno la coda in W e la testa in $\bar{W}$, e cioè gli archi $(s, 2)$, $(s, 3)$, $(s, 7)$, e $(6, t)$, e la somma delle loro capacità vale 12. Per trovare questo taglio si può usare la costruzione suggerita dall'enunciato del Teorema **??**, ovvero basta mettere in W il nodo s e tutti i nodi raggiungibili da s con un cammino aumentante rispetto alla soluzione ottima x^2. In questo esempio, i nodi raggiungibili sono appunto il nodo 4 (cammino aumentante $\{s, (s, 4), 4\}$ e il nodo 6 (cammino aumentante $\{s, (s, 4), 4, (4, 6), 6\}$).

7.8 Accoppiamento bipartito

Sia dato un grafo bipartito $G(S, D, E)$ (con $|S| = l$,$|D| = r$,$l+r = n$, $|E| = m$) non orientato. Il problema che si vuole affrontare è quello di determinare un sottoinsieme A di archi di G di cardinalità massima, tale che due archi di A non abbiano mai un nodo in comune. In altre parole, si tratta di assegnare il massimo numero di nodi dell'insieme S a nodi dell'insieme D (o viceversa), in modo che ogni nodo sia di S sia di D sia assegnato al più ad un solo altro nodo, con il vincolo che una assegnazione di un nodo i a un nodo j possa avvenire solo se nel grafo G esiste l'arco (i, j).

Questo problema, noto come problema di accoppiamento di cardinalità massima, ha moltissime applicazione, e si presenta tutte le volte in cui bisogna assegnare in maniera esclusiva i membri di un insieme

(nodi S) a quelli di un altro insieme (nodi D). Per esempio, supponiamo che i nodi S rappresentino i dipendenti di un'azienda e i nodi D i lavori che devono essere svolti. Supponiamo inoltre che un arco colleghi un nodo di S a un nodo di D se e solo se la persona rappresentata dal nodo di S è in grado di svolgere il lavoro rappresentato dal nodo di D. in questo caso allora, risolvere il problema di accoppiamento di cardinalità massima vuol dire assegnare ai dipendenti i lavori in modo da massimizzare il numero di lavori svolti.

In questa sezione presentiamo un algoritmo per la risoluzione del problema di accoppiamento di cardinalità massima basato sulla riduzione del problema a un problema di massimo flusso, che può quindi essere risolto con l'algoritmo di Ford e Fulkerson. Più precisamente mostreremo come sia possibile formulare il problema come problema di massimo flusso; ossia mostreremo come sia possibile formulare un problema di massimo flusso la cui soluzione ottima fornisca immediatamente la soluzione ottima del problema di accoppiamento bipartito.

Per raggiungere questo scopo, costruiamo il seguente grafo "stratificato" H con $(n+2)$ nodi, $(m+n)$ archi e 4 "strati" (vedi la figura 7.14).

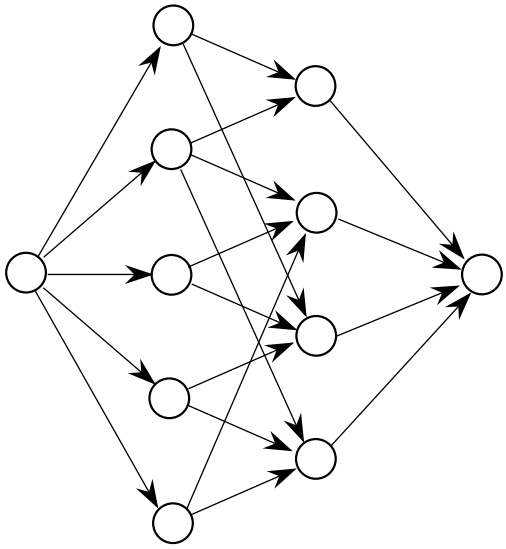


Figura 7.14: Grafo H

- Il primo strato è formato da un solo nodo (detto sorgente e indicato con s).
- Il secondo strato è formato da l nodi (corrispondenti ai nodi dell'insieme S).
- Il terzo strato è formato da r nodi (corrispondenti ai nodi dell'insieme D).
- Il quarto strato è formato da un solo nodo (detto pozzo e indicato con p).

Il nodo s è collegato a tutti i nodi del secondo strato con archi, di capacit pari a 1, orientati da s verso gli altri nodi. I nodi del secondo strato sono collegati ai nodi del terzo strato con archi, di capacit pari a 1, orientati dai nodi del secondo strato verso i nodi del terzo. Questi archi corrispondono agli archi dell'insieme E (ossia, sia i un nodo del secondo strato e dell'insieme S, j un nodo del terzo strato e dell'insieme D, esiste un arco nel grafo H tra i e j se e solo se (i, j) appartiene all'insieme E).

Tutti i nodi del terzo strato sono collegati con il nodo p con archi, di capacit pari a 1, orientati dai nodi del terzo strato verso p.

Si noti che s ha solo archi uscenti, i nodi del secondo strato hanno tutti un solo arco entrante, i nodo del terzo strato hanno tutti un solo arco uscente, p ha solo archi entranti.

Risolvendo il problema del massimo flusso dal nodo s al nodo p sul grafo H, viene anche risolto il problema di accoppiamento massimo sul grafo G.

In effetti, per la particolare struttura del grafo, è facile convincesi che una distribuzione di flusso che massimizza il flusso inviato da s a p inviando complessivamente un flusso pari a f^*, (e ottenuta con l' algoritmo di Ford e Fulkerson) è formata da un insieme di f^* cammini in cui scorre un flusso unitario, tutti con origine nel nodo s, arrivo nel nodo p, formati da tre archi (uno tra gli strati 1 e 2, uno tra gli strati 2 e 3, uno tra gli strati 3 e 4) e senza altri nodi comuni oltre a s e p. Accoppiando i nodi associati agli archi con flusso pari a 1, si ottiene un accoppiamento ammissibile di cardinalitá pari a f^*. D'altra parte, se, per assurdo, esistesse un accoppiamento di cardinalitá maggiore di f^*, allora sarebbe anche possibile, per la struttura del problema, costruire una distribuzione di flussi maggiore di f^* (e pari alla cardinalità dell'accoppiamento). Infatti basterebbe inviare un flusso pari a 1: da s a tutti i nodi del secondo strato accoppiati, fra i nodi accoppiati, e dai nodi del terzo strato accoppiati al nodo p; questo è ovviamente assurdo. Quindi l'accoppiamento di cardinalità f^*, trovato in base alla risoluzione del problema di massimo flusso, è massimo e risolve il problema di accoppiamento bipartito posto all'inizio.

7.9 Il problema di distribuzione di flusso a costo minimo

In questo ultimo pragrafo vogliamo mostrare come tutti i problemi su grafi considerati fin qui (cammini minimi, flusso massimo, accoppiamento bipartito), siano in verità casi particolari di un unico problema: Il problema della distribuzione di flusso a costo minimo.

Descriviamo dapprima questo problema. Problemi di distribuzione di flusso a costo minimo nascono naturalmente quando si voglia determinare il modo più economico di trasportare merci da alcune origini a delle destinazioni attraverso una *rete fisica*, e in effetti modelli di questo tipo sono stati gia incontrati quando abbiamo presentato modelli di PL. Consideriamo una grafo orientato $G = (V, E)$ con associato ad ogni arco $(i, j) \in E$ un costo c_{ij} (negativo, positivo o nullo), una capacità $u_{ij} \geq 0$ ed un limite inferiore l_{ij}, con $0 \leq l_{ij} \leq u_{ij}$. Supponiamo inoltre che a ciascun nodo i di V sia associato un numero intero $b(i)$, con la convenzione che:

- $b(i) > 0$ indica la presenza di offerta (il nodo i viene detto *origine*);
- $b(i) < 0$ indica la presenza di domanda (il nodo i viene detto *destinazione*);
- $b(i) = 0$ indica l'assenza di offerta e di domanda (il nodo i viene detto *di trasferimento*).

Nel seguito supporremo valida la seguente ipotesi:

Ipotesi di ammissibilità. L'offerta totale uguaglia la domanda totale; ovvero, formalmente:

$$\sum_{i \in V} b(i) = 0. \tag{7.15}$$

Definizione 7.9.1 *Un* flusso ammissibile *in una rete G è un vettore non negativo $x \in \mathbb{R}^{|E|}$ che soddisfa i seguenti vincoli:*
– vincoli di capacità: per ogni arco $(ij) \in E$:

$$l_{ij} \leq x_{ij} \leq u_{ij}.$$

– vincoli di bilanciamento della massa: per ogni nodo $k \in V$

$$\sum_{(jk) \in \omega^-(k)} x_{jk} - \sum_{(ki) \in \omega^+(k)} x_{ki} = b(k).$$

Il *costo di un flusso* è definito come la somma

$$\sum_{(ij)\in E} c_{ij}x_{ij}.$$

Il problema del flusso a costo minimo, si può enunciare come segue:

trovare una distribuzione ammissibile di flusso sulla rete G tale in modo da minimizzare il costo totale.

Se si indica con A_G la matrice d'incidenza della rete, con $b = (b(1), \ldots, b(n))^T$ il vettore della domanda/offerta, con $c \in \mathbb{R}^m$ il vettore dei costi e con $l \in \mathbb{R}^m_+$ ed $u \in \mathbb{R}^m_+$ i vettori le cui componenti rappresentano il limite inferiore e la capacità dell'arco corrispondente, il problema di flusso a costo minimo può essere scritto nel seguente modo:

$$\begin{array}{l} \min c^T x \\ A_G x = b \\ l \leq x \leq u \end{array} \qquad (P_C)$$

Nel caso di rete non capacitata, cioè quando si abbia $l_{ij} = 0$ e $u_{ij} = \infty$, allora il problema di flusso a costo minimo si scrive:

$$\begin{array}{l} \min c^T x \\ A_G x = b \\ x \geq 0 \end{array} \qquad (Q).$$

In questo caso il problema è anche noto in letteratura come *problema di trasferimento* (transhipment problem).

L'ipotesi di ammissibilità consente di assicurare l'esistenza di una soluzione ammissibile per il problema di trasferimento $(\tilde{P})$. Si ha infatti, il seguente teorema:

Teorema 7.9.2 (Ipotesi di ammissibilità) *Condizione necessaria e sufficiente per l'esistenza di una soluzione ammissibile per problema di trasferimento (Q) è cha valga (7.15).*

Una delle proprietà più interessanti della classe di problemi che stiamo analizzando è espressa dal seguente teorema.

Teorema 7.9.3 (Proprietà di interezza) *Dato un problema di flusso a costo minimo che soddisfa l'ipotesi di ammissibilità, se i vettori l, u e b sono interi non negativi, se il problema ha una soluzione ottima, allora ha una soluzione intera dello stesso valore.*

In particolare se si utilizzasse il metodo del simplesso (cosa che in genere, bisogna dire, non viene fatta, poichè si possono sviluppare sue specializzazioni che risultano più efficienti), si può mostrare che la soluzione ottima trovata, se sono soddisfatte le condizioni del teorema precedente, sono intere.

Riconsideriamo ora i problemi su grafi già studiati e mostriamo che si tratta di casi particolari del problema di flusso a costo minimo.

Il problema del massimo flusso

Un problema di massimo flusso può essere ricondotto a un problema di flusso a costo minimo nel seguente modo:

- si aggiunga un arco dal nodo t al nodo s di limite inferiore $l_{ts} = 0$, capacità u_{ts} infinita e costo $c_{ts} = -1$;
- per tutti gli altri archi $ij \in E$, $(ij \neq ts)$, si ponga $c_{ij} = 0$, $l_{ij} = 0$, lasciando la capacità invariata;

- per ogni nodo $i \in V$, si ponga $b(i) = 0$ (vincoli di conservazione del flusso).

È facile vedere che minimizzare il costo di un flusso su una rete così definita, corrisponde a massimizzare la quantità di flusso sull'arco fittizio ts, e quindi a massimizzare la quantità di flusso entrante in t.

Il problema del cammino minimo

Per capire come un problema di commino minimo possa essere visto, in effetti, come un problema di flusso a costo minimo, dobbiamo prima considerare una riformulazione del problema del cammino minimo come problema di PLI. A tal fine ricordiamo dapprima la definizione del problema.

Dato un grafo orientato $G = (N, A)$, associamo a ciascun arco $e = (u, v) \in A$ un numero reale p_e, detto *peso* dell'arco. Per ogni cammino *orientato* $C = \{v_1, e_1, \ldots, e_p, v_p\}$, denotiamo con $p(C)$ il *peso* di C, ossia la somma dei pesi degli archi di PC. Il problema del *cammino minimo* è:

dati due nodi $s \in V$ e $t \in V$, trovare un cammino orientato P^* in G da s a t di peso minimo.

Si osservi che (i) se non esiste un cammino orientato da s a t in G, il problema non ha soluzioni ammissibili; (ii) se esiste un ciclo orientato in G di peso negativo la soluzione del problema è illimitata (poiché conterrà tale ciclo ripetuto un numero illimitato di volte).

Una formulazione del problema che utilizza variabili binarie ed è descritta di seguito.

– *Variabili.* Si definiscono le variabili booleane

$$x_{ij} = \begin{cases} 1 & \text{se } (i,j) \in P \\ 0 & \text{altrimenti} \end{cases}$$

Il vettore $x \in R^m$ si chiama vettore di incidenza del cammino P.

– *Funzione obiettivo.* La funzione obiettivo è il peso del cammino:

$$\sum_{(i,j)\in E} c_{ij} x_{ij}.$$

– *Vincoli.* I vincoli devono imporre che la soluzione corrisponda al vettore d'incidenza di un cammino semplice da s a t, cioè:

- in ciascun nodo k che non sia s o t incidono esattamente due archi del cammino, uno entrante e l'altro uscente. Questo si esprime con il vincolo:

$$\sum_{(ik)\in\omega^-(k)} x_{ik} = 1 \qquad \forall k \in V - \{s,t\}$$

$$\sum_{(kj)\in\omega^+(k)} x_{kj} = 1 \qquad \forall k \in V - \{s,t\}$$

- dalla sorgente s esce esattamente un arco e nel pozzo t entra esattamente un arco; cioè:

$$\sum_{(it)\in\omega^-(t)} x_{it} = 1$$

$$\sum_{(sj)\in\omega^+(s)} x_{sj} = 1$$

Se poniamo $b = (1\ -1\ 0\ \ldots\ 0)^T$, e consideramo la matrice di incidenza A della rete G, allora questi vincoli si possono esprimere in forma matriciale come:

$$Ax = b.$$

Se si eccettua la richiesta che le variabili siano binarie, si tratta ovviamente di un problema di flusso a costa minimo in cui

- la rete è non-capacitata;
- per ogni nodo $i \in V - \{s, t\}$, si pone $b(i) = 0$;
- si pone $b(s) = 1$ e $b(t) = -1$.

È facile vedere che ogni flusso ammissibile corrisponde a inviare un flusso unitario dal nodo s al nodo t; se si suppone che il flusso ottimo sia intero, un flusso unitario corrisponde a un cammino diretto da s a t (formato dagli archi il cui flusso corrispondente è uguale a uno). Dunque, il flusso di costo minimo corrisponderà al cammino di costo minimo da s a t. Per il Teorema 7.9.3 sappiamo che in effetti, almeno una soluzione di questo problema problema è intera. Quindi se utilizziamo un algoritmo che ci garantisce di trovare una soluzione intera (per esempio il simplesso), nella formulazione precedente possiamo trascurare il vincolo (difficile da trattare) che impone che le variabili siano 0-1 e ridurci a un puro problema di flusso a costo minimo.

I problemi di flusso a costo minimo costituiscono una delle più importanti classi di problemi su grafi, e anche se non possiamo approfondire i metodi di soluzione per questa classe di problemi, abbiamo voluto almeno introdurre il problema e alcune sue importanti proprietà, quali l'esistenza di soluzioni intere.

L'aver fatto vedere che il problema dei cammini minimi e quello del massimo flusso sono casi particolari del problema di flusso a costo minimo non vuole assolutamente suggerire che i metodi di risoluzione per quest'ultima classe di problemi dovrebbero essere usati per la risoluzione di problemi di cammino minimo o di massimo flusso. È piuttosto vero il contrario: gli algoritmi che abbiamo studiato per risolvere i problemi di massimo flusso e di cammino minimo contengono in nuce le idee di base e, in alcuni casi, possono essere direttamente utilizzati come "mattoncini" per sviluppare algoritmi efficienti per la risoluzione dei problemi di flusso a costo minimo.

Capitolo 8

Euristiche per la soluzione di problemi di ottimizzazione combinatoria

Il termine *Euristica*, dal greco *euristikein* = scoprire, indica un metodo (algoritmico) per la ricerca di soluzioni ammissibili (non necessariamente ottime) di un problema di ottimizzazione. Se i vincoli o la funzione obiettivo del problema hanno una qualche particolare struttura si è talvolta in grado di sviluppare algoritmi efficienti che trovano la soluzione ottima in tempi ragionevoli anche per problemi di grande dimensione. Esempi di siffatti problemi sono il problema del cammino minimo in un grafo e il problema del masiimo flusso. In molti casi, tuttavia, non sono noti metodi efficienti, e ci si deve accontantare di applicare metodi generali quali ad esempio il branch-and-bound. Purtroppo, quando la dimensione del problema cresce, il metodo del branch-and-bound potrebbe richiedere la generazione di un numero troppo elevato di sottoproblemi, e quindi non convergere in tempo ragionevole.

Per problemi di questo tipo, all'aumentare delle dimensioni ci si deve spesso accontentare di metodi che non assicurano l'ottimalità delle soluzioni trovate - le euristiche, appunto - ma che producono soluzioni di "qualità" ragionevole. In generale, se Z^* è il valore della soluzione ottima del problema dato (supposto di minimizzazione, per fissare le idee) e Z^E è il valore della soluzione trovata dalla nostra euristica, sarà $Z^E \geq Z^*$. Il punto cruciale è che la soluzione euristica *potrebbe* essere ottima (ovvero $Z^E = Z^*$), ma noi non siamo in grado di dimostrarne l'ottimalità. Questa è la differenza sostanziale fra metodi euristici e metodi esatti (come il branch-and-bound): se il metodo esatto ha il tempo sufficiente a terminare le sue iterazioni, la soluzione finale *è* una soluzione ottima.

Alcuni tipi di euristiche forniscono un risultato *garantito* nel senso che è possibile dimostrare che la soluzione prodotta ha una valore che è al più α volte la soluzione ottima (con $\alpha > 1$), ovvero $Z^E \leq \alpha Z^*$.

Più spesso il risultato delle euristiche non è garantito, e la soluzione prodotta potrebbe avere un valore molto maggiore della soluzione ottima. Per poter valutare la qualità di un'euristica per un problema dato, i ricercatori hanno messo a punto un metodo detto di *benchmarking*. Si tratta semplicemente di utilizzare delle librerie di istanze del problema considerato per le quali sono noti i risultati ottenuti dalle altre euristiche proposte nella letteratura, e verificare come si posizionano rispetto ad essi i risultati prodotti dalla nuova euristica.

Di seguito ci occuperemo di euristiche per una classe particolare di problemi di ottimizzazione, introdotta nel prossimo paragrafo, e che comprende numerosissimi esempi di problemi reali.

8.1 L'ottimizzazione combinatoria

Una particolare classe di problemi di ottimizzazione è rappresentata dai cosiddetti *problemi di ottimizzazione combinatoria*. A questa classe appartengono, fra l'altro, i problemi di PL01, il problema dell'accoppiamento e il problema del cammino minimo su grafi visti nei capitoli precedenti.

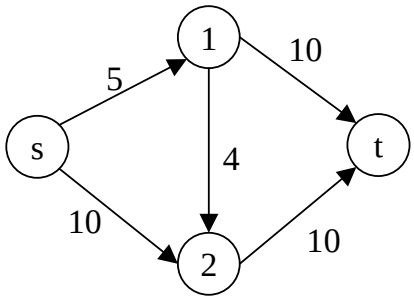


Figura 8.1: Esempio di cammino minimo

Un generico problema di ottimizzazione combinatoria può essere definito come segue. Sia $B = \{b_1, \ldots, b_n\}$ un qualsiasi insieme finito detto *insieme base* ("*ground set*") e sia $\mathcal{S} = \{S_1, \ldots, S_m\}$ una famiglia di sottoinsiemi di B ("*Subset System*"). Infine, sia $w : \mathcal{S} \rightarrow I\!R$ una funzione obiettivo che associa a ciascun insieme $S \in \mathcal{S}$ un numero reale $w(S)$. Allora, diremo *Problema di Ottimizzazione Combinatoria* il seguente problema:

$$\min_{S \in \mathcal{S}} w(S)$$

.

Si osservi la differenza fra la definizione di problema di ottimizzazione combinatoria e la piú generale definizione di problema di ottimizzazione introdotta nel Capitolo 1. Nella definizione generale, l'insieme di soluzioni ammissibili $\mathcal{S}$ puó assumere qualunque forma, mentre nei problemi di ottimizzazione $\mathcal{S}$ é una famiglia (finita) di sottoinsiemi di un insieme finito di elementi.

Ad esempio, il problema del cammino minimo da s a t in un grafo orientato $G = (V, E)$, con un peso p_e associato a ogni arco $e \in E$, visto nel capitolo precedente, è un problema di ottimizzazione combinatoria. Per questo problema, l'insieme di base è l'insieme degli archi. Per l'istanza mostrata in figura 8.1, l'insieme di base sarà $B = \{(s,1),(s,2),(1,2),(1,t),(2,t)\}$. La famiglia di soluzioni è l'insieme di tutti i sottoinsiemi di archi che corrispondono a cammini orientati da s a t. In particolare, avremo 3 possibili cammini distinti da s a t e quindi $\mathcal{S} = \{S_1 = \{(s,1),(1,t)\}, S_2 = \{(s,1),(1,2),(2,t)\}, S_3 = \{(s,2),(2,t)\}\}$. La funzione obiettivo è semplicemente la somma dei pesi associati agli archi: $w(S) = \sum_{e \in S} p_e$. Quindi $w(S_1) = 15$, $w(S_2) = 19$ e $w(S_3) = 20$.

Nella maggioranza dei casi trattati, a ogni elemento dell'insieme di base é associato un peso e la funzione obiettivo é semplicemente la somma dei pesi degli elementi contenuti nella soluzione. Tuttavia la funzione obiettivo puó assumere in generale qualunque forma. Un esempio di diversa funzione obiettivo verrá mostrato in seguito.

Nei prossimi paragrafi illustreremo alcuni esempi importanti di problemi di ottimizzazione combinatoria.

8.1.1 Il problema dell'accoppiamento massimo su grafi bipartiti

Sia dato un grafo $G = (V, E)$ non orientato e bipartito, ovvero i) $V = U \cup W$ e $W \cap U = \emptyset$ e ogni arco appartenente ad E ha per estremi un nodo $u \in U$ e un nodo $v \in W$ (cioè non esistono archi con ambo gli estremi in U o ambo gli estremi in W). Inoltre, con ogni arco $(u, v) \in E$ è associato un peso w_{uv}, che rappresenta il "vantaggio" di assegnare il nodo u al nodo v. Un esempio di grafo bipartito (pesato) è mostrato in figura 8.2.a. Ora, a ogni nodo dell'insieme W vogliamo assegnare al più un nodo dell'insieme U e viceversa, in modo da massimizzare la somma dei vantaggi. Si osservi che assegnare un nodo $u \in U$ a un nodo $v \in W$ è equivalente a selezionare l'arco (u, v) (di peso w_{uv}). Inoltre, se è stato selezionato l'arco (u, v), e quindi se è stato assegnato il nodo u a v (e viceversa), nessun altro arco incidente in u può essere selezionato (perchè ciò equivarrebbe ad assegnare u a più di un nodo); analogamente, nessun altro arco incidente in v può essere selezionato. Un sottoinsieme $M \subseteq E$ degli archi di G che gode della proprietà che, comunque presi due archi appartenenti a M, essi non hanno estremi in comune, è detto *accoppiamento* (in inglese "*matching*"). Un esempio di matching (di valore 19) è mostrato in figura 8.2.b.

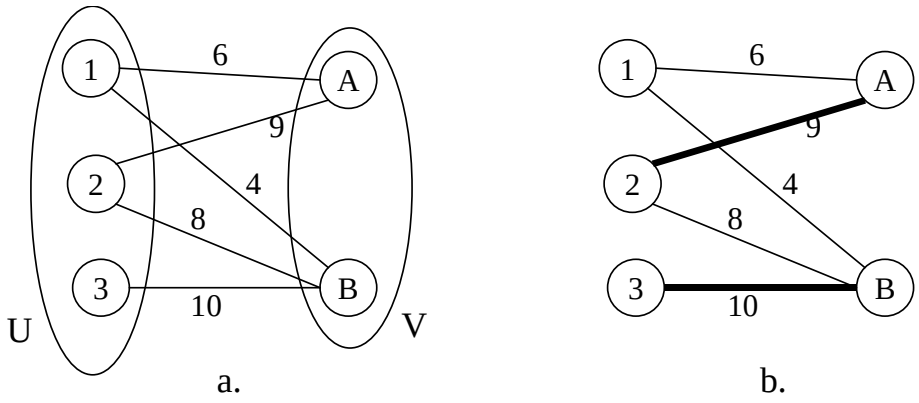


Figura 8.2: Esempio di accoppiamento bipartito

Dunque, il problema dell'accoppiamento massimo può essere così ridefinito: selezionare un sottoinsieme M degli archi di G tale che i) M sia un matching, e ii) la somma dei pesi degli archi in M sia massima.

Così formulato il problema dell'accoppiamento massimo su grafo bipartito è chiaramente un problema di ottimizzazione combintatoria, ove:

- l'insieme di base coincide con l'insieme degli archi E.
- la famiglia di sottoinsiemi di archi coincide con la famiglia di tutti i possibili matching di G
- la funzione obiettivo associa a ogni matching M la somma dei pesi degli archi in M.

8.1.2 Il Problema del Commesso Viaggiatore.

Un commesso viaggiatore deve visitare un certo numero di città, partendo dalla sua città di residenza e ritornandoci alla fine del giro. Questo problema può essere utilmente rappresentato su un grafo $G = (V, E)$, ove ogni nodo $u \in V$ rappresenta una città da visitare, mentre gli archi rappresentano strade (o voli) che collegano coppie di città. Per semplificare la trattazione supporremo che il grafo G sia non orientato: l'estensione al caso orientato di quanto verrà detto è immediata. A ogni arco $(u, v) \in E$ si associa una distanza $d_{uv} \in \mathbb{R}_+$ che rappresenta la lunghezza del percorso fra la città u e la città v. Si osservi che d_{uv} potrebbe anche rappresentare il tempo di percorrenza, o il costo del trasferimento.

Dunque, il commesso viaggiatore, partendo dalla città in cui abita (sia per esempio la città 1), vuole effettuare tutte le visite e ritornare alla fine alla città di partenza, percorrendo una strada di lunghezza complessiva minima. Si osservi intanto che il "giro" del commesso viaggiatore corrisponde a un ciclo del grafo G che attraversi ogni nodo una e una sola volta. Un ciclo siffatto è detto *ciclo hamiltoniano*. Quindi, il *Problema del Commesso Viaggiatore* (indicato anche con l'acronimo TSP, dall'inglese *Travelling Salesman Problem*) si definisce come il problema di selezionare un particolare ciclo hamiltoniano, quello (o uno di quelli) di costo (lunghezza, peso) minimo.

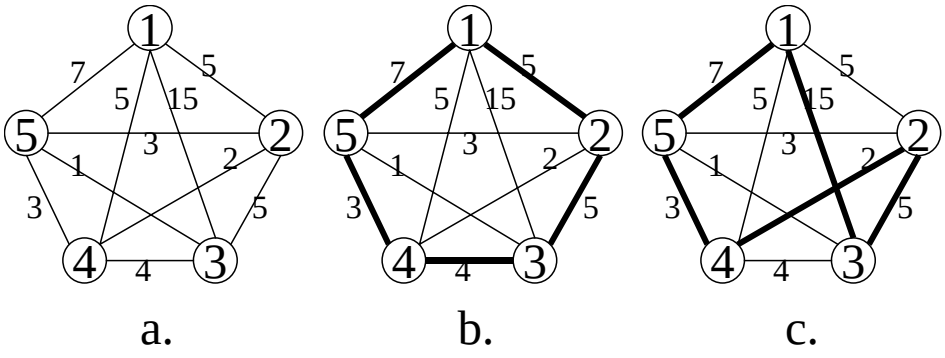


Figura 8.3: Esempi di cicli hamiltoniani

Si consideri ad esempio il grafo in Fig. 8.3. Un possibile ciclo hamiltoniano è, ad esempio, l'insieme di archi $H_1 = \{(1,2),(2,3),(3,4),(4,5),(5,1)\}$ (Fig. 8.3.b), e il suo costo (somma dei costi degli archi) è pari a 24. Un altro ciclo hamiltoniano è $H_2 = \{(1,3),(3,2),(2,4),(4,5),(5,1)\}$ (Fig. 8.3.c) di costo pari a 32.

Un ciclo hamiltoniano può dunque essere rappresentato come insieme di archi. Quindi, l'insieme di base del nostro problema di ottimizzazione combinatoria è l'insieme E degli archi di G. L'insieme delle soluzioni è una famiglia $\mathcal{H} = \{H_1, \ldots, H_m\}$ di sottoinsiemi di E, ove H_i è l'insieme di archi di un ciclo hamiltoniano per $i = 1, \ldots, m$. La funzione obiettivo è semplicemente, per ogni $H \in \mathcal{H}$, $w(H) = \sum_{(u,v)\in H} d_{uv}$.

8.1.3 Il problema del partizionamento di un grafo.

Problemi di questo tipo appaiono ogni qual volta si debbia scegliere una partizione di un insieme di elementi in un certo numero di classi in modo da minimizzare (o massimizzare) una funzione di distanza fra gli elementi appartenenti a una stessa classe. Sia quindi dato un grafo non orientato $G = (V, E)$, e sia associato un peso w_{uv} con ogni arco $uv \in E$. Si vuole trovare la partizione di V in k classi (o *cluster*), con k prefissato, che minimizzi la somma dei pesi degli archi incidenti in nodi appartenenti a una stessa classe. Si ricordi che se $\pi = \{C_1, C_2, \ldots, C_k\}$ è una k partizione di V si ha $\cup_{i=1}^{n} C_i = V$ e $C_i \cap C_j = \emptyset$ per $i, j \in \{1, \ldots, k\}, i \neq j$. Il costo $w(C_i)$ di una classe di nodi C è definito come $\sum_{u\in C, v\in C} w_{uv}$. Il costo di una partizione $w(\pi) = w(\{C_1, C_2, \ldots, C_k\})$ è definito come $\sum_{i=1}^{k} w(C_i)$. Il problema di Partizionamento di un Grafo consiste nel scegliere la k-partizione π^* che minimizza $w(\pi^*)$.

Anche il problema di partizionamento di un grafo può essere inserito nel *framework* dei problemi di ottimizzazione, anche se l'associazione non è così immediata. Esistono in realtà (almeno) due diverse rappresentazioni del problema come problema di ottimizzazione combinatoria. La prima consiste nel definire l'insieme di base come l'insieme degli archi E e le soluzioni ammissibili come quei sottoinsiemi di archi che sono archi completamente contenuti in qualche classe di una k-partizione di G. Cioè, $S \subseteq E$ è una soluzione ammissibile se e solo se esiste una k-partizione $\pi = \{C_1, \ldots, C_k\}$ tale che, per ogni arco $uv \in S$, il nodo u e il nodo v sono entrambi contenuti in una stessa classe della partizione π. In questo

caso, la funzione di costo $w(S)$ è semplicemente la somma degli archi contenuti in S.

Una seconda rappresentazione è in qualche senso più naturale, perchè discende direttamente dalla definizione di k-partizione. Infatti, una k-partizione può essere definita come una funzione che associa a ogni nodo v un intero appartenente all'intervallo $[1, \ldots, k]$, e cioè (l'indice del) la classe di appertenza. Quindi, una soluzione è completamente definita da un insieme di n coppie $(v_1, i_1), (v_2, i_2), \ldots, (v_n, i_n)$, ove con i_j si è indicato la classe di appartenenza del nodo v_j. Quindi, l'insieme di base è l'insieme $B = \{(v, i) : v \in V, i = 1 \ldots, k\}$, mentre una soluzione $S \subseteq B$ è un insieme di coppie nodo/classe (v, i) con la proprietà che ogni nodo di V è contenuto in esattamente una coppia di S. La funzione obiettivo, in questo caso, non è una funzione lineare degli elementi di S: in effetti, a ciascun elemento dell'insieme di base non è associato alcun peso. Tuttavia, il costo della soluzione S dipende dalle coppie apaprtenenti ad S nel senso che queste ultime determinano gli archi che appartengono a una medesima classe della partizione e quindi il costo di S. La funzione di costo sarà quindi $w(S) = \sum_{i=1}^{k} \sum_{(u,i)\in S,(v,i)\in S} w_{uv}$. Questo é un esempio di funzione obiettivo "atipica" anticipato nella parte iniziale di questo capitolo.

Una importante famiglia di problemi di partizionamento pone una restrizione sulla dimensione delle classi. In particolare, con ogni classe viene associata una cardinalità massima, e il problema diventa: trovare la k-partizione di costo minimo, col vincolo che la cardinalità della j-esima classe non ecceda la cardinalità massima r_j, per $j = 1, \ldots, k$.

8.1.4 PL01

Le soluzioni ammissibili di un problema di programmazione lineare (0,1) sono vettori booleani $x \in \{0, 1\}^n$. In genere, ogni componente di un vettore booleano rappresenta il fatto che un certo elemento di un insieme di base $B = \{b_1, \ldots, b_n\}$ è scelto oppure no. Tipicamente, se la componente i-esima $x_i = 1$ allora l'elemento b_i è scelto, se $x_i = 0$ allora b_i non è scelto. Ad esempio, nel problema di scelta dei progetti più remunerativi, l'insieme di base è l'insieme dei possibili progetti attivabili e $x_i = 1$ significa che il progetto i-esimo è attivato, $x_i = 0$ significa che il progetto i-esimo non è attivato. In altri termini, un vettore booleano identifica tutti gli elementi dell'insieme di base che sono stati scelti, e cioè un particolare sottoinsieme $S \subseteq B$. Al contrario, dato un qualunque sottoinsieme $S \subseteq B$, possiamo associare a S un vettore booleano x^S detto *vettore d'incidenza di S*, tale che la componente i-esima $x_i^S = 1$ se e solo se $b_i \in S$.

In definitiva, un problema di programmazione lineare (0,1) $\min\{w^T x : Ax \geq b, x \in \{0, 1\}^n\}$ può sempre essere interpretato nel seguente modo:

- Le variabili $x_1, \ldots, x_n$ sono in corrispondenza agli elementi di un insieme di base $B = \{b_1, b_2, \ldots, b_n\}$.
- I vincoli del problema definiscono la regione ammissibile e quindi ci dicono quali vettori booleani sono ammissibili e quali no. Poichè ogni vettore booleano è in corrispondenza con un sottoinsieme dell'insieme di base, i vincoli definiscono quali sottoinsiemi dell'insieme di base sono soluzioni ammissibili del nostro problema e quali no.
- La funzione obiettivo (lineare) associa un costo w_i a ogni variabile x_i per $i = 1, \ldots, n$ e quindi a ciascun elemento b_i dell'insieme di base B. E' facile vedere che il costo di una soluzione x^S, e cioè di un sottoinsieme S di B, è semplicemente la somma dei costi degli elementi appartenenti a S.

8.2 Un sistema multitaxi per il servizio Areoporto Fiumicino - Roma Centro

Descriveremo di seguito un'applicazione reale e recente che ha richiesto l'utilizzazione di diverse tecniche dell'ottimizzazione combinatoria (algortimi greedy e ricerca locale) e dell'ottimizzazione su reti (cammini minimi e assegnamento). In effetti, gli algoritmi di soluzione descritti nei prossimi paragrafi sono stati scelti come "esempi" di algoritmi proprio perché necessari alla soluzione dell'applicazione descritta. Altre applicazioni avrebbero ovviamente richiesto algoritmi diversi. Si osservi inoltre che la maggior parte dei

problemi di ottimizzazione presenti in "natura" non sono puri e la scelta dei modelli da utilizzare non è univoca dipendendo dalla dimensione delle istanze d'interesse, dalla disponibilità di algoritmi efficienti, etc.

Nel 1999 l'Ente Nazionale per le Energie Alternative (ENEA) è stato incaricato dal ministero dell'ambiente di sviluppare sistemi alternativi per il trasporto urbano. Investigando una serie di strategie per la politica dei trasporti, l'ENEA si è resa conto dell'esistenza di alcuni problemi di ottimizzazione connessi alle proposte allo studio. Si tratta di un buon esempio di "sinergia" fra diverse competenze, e soprattutto si tratta di un buon esempio di utilizzazione delle metodologie dell'ottimizzazione intese non come solutori universali, ma come "mattoncini" (in inglese "building block") per sviluppare sistemi complessi.

Fra le varie strategie individuate dall'ENEA, una consisteva nello sviluppo di un sistema integrato di *taxi collettivo* (detto anche *multitaxi*) per il trasporto di persone dall'areoporto di Fiumicino alle loro destinazioni cittadine e viceversa.

Il taxi collettivo è un'interessante alternativa ai tradizionali sistemi di trasporto pubblico. La principale caratteristica di questo tipo di servizio risiede nel fatto che i percorsi dei veicoli non sono prefissati (come nel caso delle linee degli autobus), ma sono calcolati di volta in volta in base alla domanda; questa flessibilità è propria del servizio taxi "classico", ma a differenza di quest'ultimo, il servizio multitaxi non è individuale. Un sistema di trasporto multitaxi è composto da:

- Una flotta di vetture, non necessariamente uguali tra loro, il cui compito è prelevare dei clienti dai loro punti di partenza e trasportarli ciascuno nelle rispettive destinazioni.
- Uno o più depositi in cui gli automezzi ritornano alla fine del turno di servizio.
- Una centrale di gestione che raccoglie le richieste di servizio degli utenti e comunica alle vetture i percorsi da seguire e i clienti da servire.

Il servizio può essere di tipo immediato o a prenotazione. Nel primo caso le richieste dei clienti sono soddisfatte appena possibile, a partire dal momento della loro formulazione, oppure sono subito rifiutate se non è possibile (o conveniente) soddisfarle. Nel secondo caso le domande devono pervenire alla centrale di gestione con un certo anticipo per essere servite in un secondo momento.

Le richieste degli utenti sono espresse in termini di località di prelievo, località di destinazione e, nel caso di servizio a prenotazione, orario desiderato (di partenza o di arrivo).

Il servizio, inoltre, può essere dei seguenti tipi:

- Molti a molti. In questo modo si indica l'esistenza di molteplici (e distinti) punti di raccolta e di consegna.Questo implica che le vetture, durante i loro viaggi, effettuano sia prelievi che consegne di clienti.
- Uno a molti. In questo caso, invece, si indica l'esistenza di un unico punto di raccolta e di molteplici (e distinti) punti di consegna; di conseguenza le vetture effettuano un solo prelievo di clienti, all'inizio del viaggio, dopodichè si limitano a portare a destinazione tutti i passeggeri a bordo.
- Molti a uno. Le vetture effettuano una serie di prelievi e concludono il loro viaggio nell'unico punto di discesa per i clienti.

Un sistema di trasporto multitaxi è particolarmente adatto in situazioni di utenza debole (anziani e disabili) e nel caso di domanda scarsa (trasporto urbano notturno, servizio in aree poco abitate).

Nel caso specifico del servizio multitaxi dall'areoporto di Fiumicino a Roma centro, si tratta di un servizio immediato di tipo uno a molti. Coloro che intendono avvalersi di questo servizio si presentano ad uno sportello di accettazione dove vengono registrati i loro dati (nominativo, ora di arrivo allo sportello, indirizzo di destinazione,...). Terminata la fase di registrazione, i clienti restano in attesa fino al momento in cui viene loro comunicato di salire su una vettura per partire.

Per semplicità supporremo che tutte le autovetture abbiano la stessa capacità (ad esempio, possono trasportare al massimo 5 passeggeri). Il sistema automatico, ad intervalli regolari (ad esempio ogni

minuto), suddivide i clienti in attesa in gruppi composti al massimo da cinque persone, assegna ciascun gruppo a un taxi specifico, decide quali gruppi debbano partire e, infine, calcola per ogni taxi l'itinerario da seguire, ovvero la sequenza delle consegne. I clienti che compongono i gruppi selezionati per la partenza vengono fatti salire sulle vetture e queste ultime iniziano il loro viaggio, seguendo le indicazioni di percorso fornite dalla centrale di gestione. I dati dei clienti partiti vengono quindi rimossi dal sistema. Infine, le vetture devono tornare alll'areoporto per accogliere nuovi passeggeri. Bisogna quindi decidere a) come si compongono i gruppi (cioè gli equipaggi dei singoli taxi) e b) quale sia, per ogni vettura, il percorso migliore da seguire. Benchè gli obiettivi possono essere molteplici, in prima analisi potremmo assumere che il gestore debba soddisfare le richieste dei clienti cercando di minimizzare il tempo medio (o le distanze) percorse dai suoi taxi. Si tratta evidentemente di un problema di ottimizzazione. Esiste una naturale decomposizione del problema in due fasi: 1. fase di clustering, o di partizionamento, in cui i clienti attualmente in attesa vengono suddivisi in un numero di gruppi di dimensione al più pari alla capacità dei veicoli. Tipicamente si preferisce aggegare clienti con destinizioni il più possibile vicine fra loro. 2. fase di istradamento (in inglese "*routing*"). In questa fase si decidono i percorsi che i taxi devono seguire per scaricare i passeggeri. I percorsi devono essere tali da minimizzare, ad esempio, le distanze percorse o i tempi di viaggio.

Per poter modellare quanto sopra descritto in termini di problema di ottimizzazione è conveniente definire una struttura matematica adeguata per rappresentare strade, incroci e destinazioni.

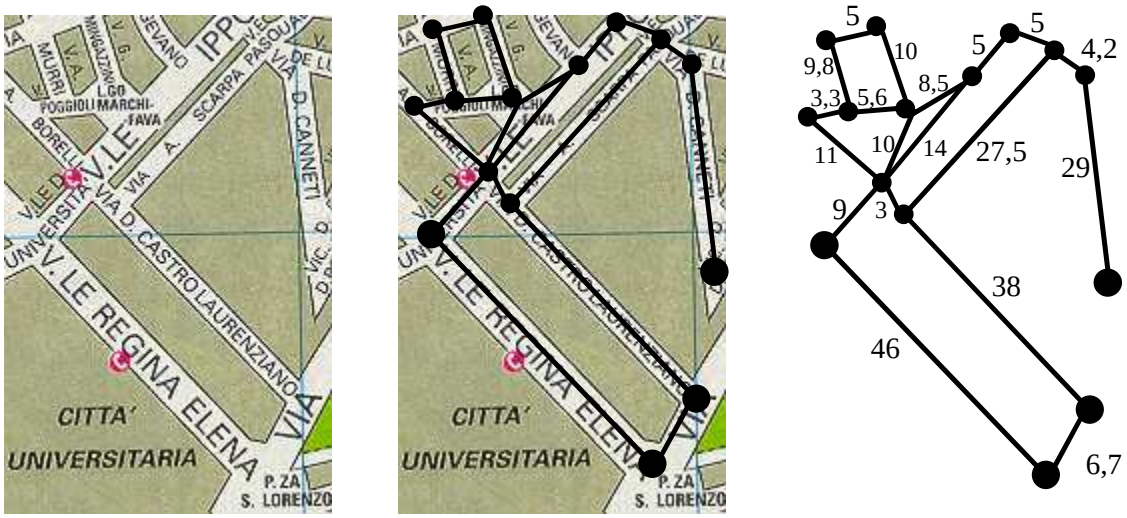


Figura 8.4: Dettaglio del grafo di Roma

Il grafo di Roma Tutte le possibili destinazioni di clienti sono rappresentate dal grafo di Roma. Si tratta di un grafo orientanto $R = (W, A)$ dove l'insieme dei nodi W rappresenta gli incroci (o piazze) della città mentre gli archi sono i tratti di strada compresi fra coppie di incroci (vedi Fig. 8.4). La lunghezza c_{uv} di ogni arco $(u, v) \in A$ rappresenta la lunghezza effettiva del tratto di strada corrispondente.

Per semplificare la trattazione successiva, supporremo sempre che la destinazione dei clienti coincida con una piazza o un incrocio, cioè con un nodo del grafo di Roma. Qualora questa ipotesi sia troppo restrittiva, in corrispondenza cioè a tratti di strada molto lunghi fra due incroci adiacenti, si può sempre aggiungere un incrocio *fittizio* suddividendo il tratto in due tratti consecutivi separati (vedi figura 8.5).

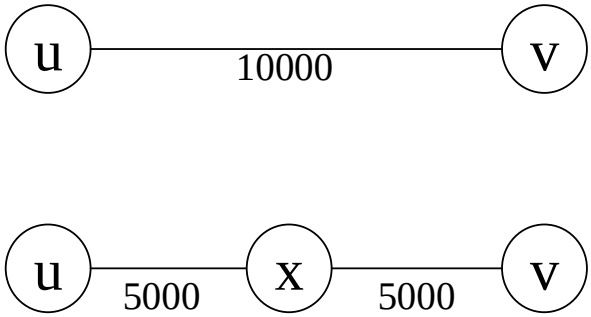


Figura 8.5: Aggiunta di nodi fittizi

Poichè ogni cliente ha una destinazione e ogni destinazione è un nodo del grafo di Roma, i clienti attualmente in attesa sono in corrispondenza con un sottoinsieme di nodi del grafo di Roma. Quindi, il problema di partizionare i clienti in modo che le destinazioni di clienti appartenenti a una stessa classe siano fra loro il più vicino possibile può essere riformulato come un problema di clustering (partizionamento dei nodi) nel sottografo del grafo di Roma indotto dai clienti in attesa.

Una volta che i clienti siano stati assegnati ai veicoli si devono ottimizzare i percorsi dei taxi. E' facile vedere che il problema di minimizzare la lunghezza del percorso di un taxi che deve scaricare i suoi passeggeri in alcuni nodi del grafo di Roma e quindi tornare al (nodo di) Fiumicino è esattamente il problema di calcolare un ciclo hamiltoniano di lunghezza minima nel grafo indotto dai nodi di destinazione dei passeggeri e dal nodo di Fiumicino.

Quindi, il problema di gestione di un servizio multitaxi può essere affrontato risolvendo in sequenza due problemi di ottimizzazione su grafi: il problema di clustering (sul grafo indotto da tutti i clienti in attesa) e il problema del commesso viaggiatore (TSP) sul grafo indotto dai nodi destinazione dei clienti di un taxi. Chiaramente, questo secondo problema andrà risolto per ogni vettura in partenza.

Un'altra assunzione che faremo di seguito per semplificare le descrizioni degli algoritmi è l'ipotesi di *grafo completo*.

Definizione 8.2.1 *Un grafo $R = (W, A)$ si dice* completo *se, per ogni coppia di nodi distinti $u, v \in W$, l'arco (u, v) appartiene ad A.*

Chiaramente, il grafo di Roma non soddisfa l'ipotesi di completezza, a causa della presenza di moltissime coppie di piazze o incroci (la maggior parte) non collegati direttamente da archi. Tuttavia, per i nostri scopi è possibile completare il grafo, e cioè aggiungere gli archi mancanti, sfruttando le seguenti considerazioni. Quando un taxi si deve spostare da un generico nodo u a un nodo v distinto e non collegato direttamente a u, dovrebbe preferibilmente scegliere il percorso più breve che da u porti a v, e cioè il cammino orientato di lunghezza minima da u a v. Nell'ipotesi di taxi "intelligente", quindi, possiamo rimpiazzare l'arco mancante con un arco fittizio di lunghezza pari alla lunghezza del camminimo minimo da u a v. In altri termini, prima di applicare gli algoritmi di ottimizzazione, il grafo di Roma subirà una fase di *pre-processamento* nella quale:

1. per ogni coppia (orientata) di nodi distinti u e v viene calcolato (e memorizzato in opportune strutture dati) il cammino minimo da u a v: sia c^*_{uv} la lunghezza di tale cammino. A tal scopo può essere adoperato l'algoritmo di Dijkstra descritto nel Capitolo **??**.

2. per ogni coppia (orientata) di nodi distinti u e v si aggiunge l'arco (u, v) all'insieme di archi A: la lunghezza associata sarà posta uguale a c^*_{uv}.

8.3 Euristiche di tipo "Greedy"

Una prima classe di euristiche per la soluzione di problemi di ottimizzazione combinatoria va sotto il nome di *euristiche di tipo "greedy"*. Il termine inglese "greedy" può essere tradotto come *avido* ed è stato scelto per rappresentare una caratteristica fondamentale dell'algoritmo greedy, appunto la cosiddetta *scelta greedy*. Per costruire la soluzione finale, che ricordiamo essere un sottoinsieme S dell'insieme di base B, l'algoritmo parte in genere da una soluzione parziale che estende aggiungendo un elemento per volta fino a costruire la soluzione finale. L'elemento scelto di volta in volta è quello che minimizza il costo della funzione obiettivo (da cui "avido"). Inoltre, una volta scelto un elemento, questo non viene mai rimosso dalla soluzione (altra caratteristica dell'avidità). Di seguito, denoteremo con $B = \{b_1, \ldots, b_n\}$ l'insieme di base e con $\mathcal{S} = \{S_1, \ldots, S_m\}$ l'insieme delle soluzioni ammissibili del problema di ottimizzazione combinatoria. Inoltre, supponiamo che un costo sia a ssociato a *qualsiasi* sottoinsieme di B (non solo alle soluzioni ammissibili). E cioè, indicando con $\mathcal{T}$ la famiglia di tutti i sottoinsiemi di B, supporremo che sia definita una funzione $w : \mathcal{T} \to I\!R$ per ogni sottoinsieme $T \in B$. L'obiettivo è sempre quello di trovare una soluzione ammissibile $S \in \mathcal{S} \subseteq \mathcal{T}$ tale che $w(S)$ sia minimo.

Per introdurre formalmente l'algoritmo greedy, abbiamo bisogno della seguente definizione:

Definizione 8.3.1 *Un sottoinsieme T dell'insieme di base B è detto soluzione parziale (o anche sottosoluzione), se esiste una soluzione $S \in \mathcal{S}$ tale che $T \subseteq S$.*

Si osservi che:

a. Se S é una soluzione, allora S é anche una soluzione parziale. Infatti $S \subseteq S$.

b. Se T é una soluzione parziale, allora é sempre possibile costruire una soluzione S tale che $T \subseteq S$ semplicemente inserendo elementi a T.

L'algoritmo greedy costruisce una sequenza di soluzioni parziali $T_0, T_1, \ldots, T_q$ con la proprietà che T_0 è l'insieme vuoto, T_q è una soluzione ammissibile del problema e $T_i = T_{i-1} \cup \{e\}$, ove $e \in B - T_{i-1}$ è l'elemento tale che $T_{i-1} \cup \{e\}$ sia ancora una soluzione parziale e abbia costo minimo (fra tutte le soluzioni parziali ottenibili aggiungendo un elemento a T_{i-1}. Il seguente schema fornisce una versione semplificata dell'algoritmo greedy:

Algoritmo Greedy

a. *Inizializzazione.* Poni $T_0 = \emptyset$. Poni $i = 1$.

b. *Iterazione i-esima.* Scegli $e \in B - T_{i-1}$ tale che $T_{i-1} \cup \{e\}$ sia una soluzione parziale e $w(T_{i-1} \cup \{e\})$ sia minimo.

c. Poni $T_i = T_{i-1} \cup \{e\}$.

d. *Terminazione* Se T_i è una soluzione ammissibile: STOP.

e. Altrimenti Poni $i = i + 1$. Va al passo b.

Si osservi che il passo [b.] possiede un certo grado di indeterminatezza. In particolare, potrebbero esistere più elemtenti che minimizzano il costo della nuova soluzione parziale. In questo caso, è necessaria una regola per "dirimire i pareggi" (in inglese "tie breaking rule"). A volte è sufficiente scegliere a caso (*random choice*); altre volte è più opportuno definire una nuova funzione di costo. Vedremo di seguito esempi delle due alternative.

Per concludere questa sezione, si osservi che ogni qual volta si voglia applicare l'algoritmo greedy a un qualunque problema di ottimizzazione combinatoria, bisgonerà innanzitutto definire:

- l'insieme di base
- la famiglia delle soluzioni
- la famiglia delle soluzioni parziali
- la funzione obiettivo

Poichè il nostro esempio prevede la soluzione in cascata di un problema di partizionamento di grafi e di un problema di commesso viaggiatore, considereremo appunto le applicazioni dell'algoritmo greedy a tali problemi. Consideriamo innanzitutto l'applicazione dell'algoritmo greedy al problema del TSP.

8.3.1 Applicazione dell'algoritmo greedy generico al problema del Commesso Viaggiatore.

Consideriamo il problema del Commesso Viaggiatore (TSP). Come visto nel paragrafo 8.1.2, una soluzione ammissibile per il TSP è un insieme di archi che definisce un ciclo hamiltoniano. Si consideri ad esempio il grafo in Fig. 8.6. L'insieme delle soluzioni è rappresentato da tutti i possibili cicli hamiltoniani del grafo. Un possibile ciclo hamiltoniano è, ad esempio $S_1 = \{(1,2),(2,3),(3,4),(4,5),(5,1)\}$, e il suo costo (somma dei costi degli archi) è pari a 24. Un altro ciclo hamiltoniano è $S_2 = \{(1,3),(3,2),(2,4),(4,5),(5,1)\}$ di costo pari a 32. E' facile vedere che i cicli hamiltoniani in un grafo completo sono in corrispondenza ai possibili ordinamenti circolari dei nodi del grafo. Ad esempio, la soluzione S_1 è in corrispondenza dell'ordinamento $o_1 = (1,2,3,4,5)$, mentre S_2 è in corrispondenza a $o_2 = (1,3,2,4,5)$ Quindi, se i nodi sono 5, abbiamo $4! = 24$ cicli hamiltoniani distinti.

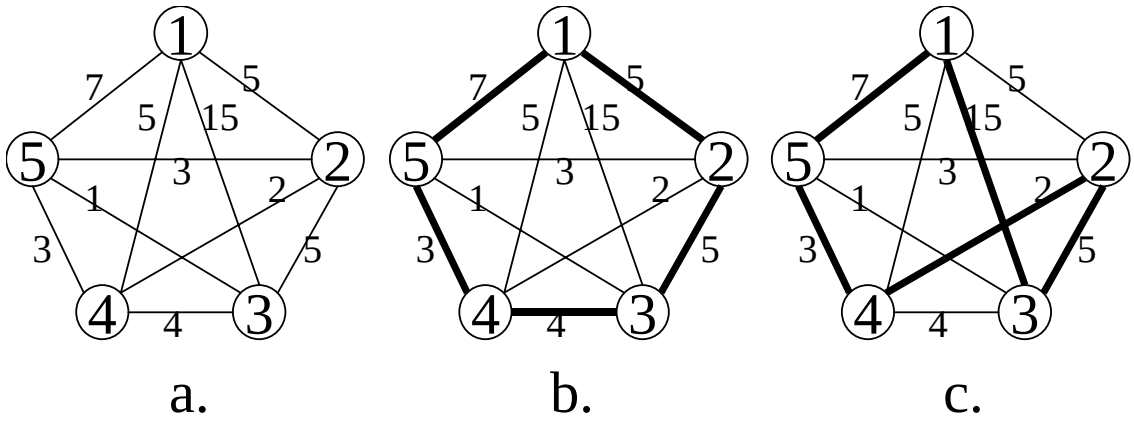


Figura 8.6: Esempi di cicli hamiltoniani

Osservazione 8.3.2 *Il numero di cicli hamiltoniani in un grafo completo con n nodi è pari a* $(n-1)!$.

La precedente osservazione permette di rilevare che il numero di cicli hamiltoniani di un grafo completo cresce *esponenzialmente* col numero di nodi. In altri termini, quando il numero di nodi è elevato, l'esplorazione completa dell'insieme delle soluzioni diventa impraticabile ed è cruciale sviluppare delle buone euristiche per la soluzione del problema.

Vediamo come l'algoritmo greedy si applica al TSP. Innanzitutto dobbiamo rispondere alla domanda: chi sono le soluzioni parziali del problema del Commesso Viaggiatore? Per vedere ciò consideriamo innanzitutto le proprietà di cui deve godere un insieme di archi s perchè esso sia un ciclo hamiltoniano.

Osservazione 8.3.3 *Dato un grafo $G = (V, A)$ e un insieme $S \in A$, S è l'insieme di archi di un ciclo hamiltoniano se e solo se:*

1. *in ogni nodo di G incidono esattamente due archi di S*
2. *S non contiene cicli di cardinalità inferiore a $|V|$.*

Per capire la natura della condizione 2., si osservi la figura 8.7. In grassetto è evidenziato un insieme S di archi che soddisfa la condizione 1. ma non la condizione 2. (l'insieme contiene infatti cicli di cardinalità inferiore a 6). In effetti, l'insieme S non rappresenta in questo caso un ciclo hamiltoniano.

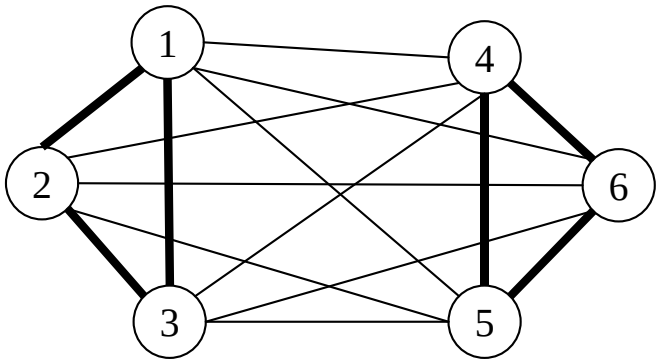


Figura 8.7: Esempi di cicli hamiltoniani

E' facile a questo punto vedere che T è una soluzione parziale per il TSP se e solo se

1. in ogni nodo di G incidono al più due archi di T.
2. T non contiene cicli di cardinalità inferiore a $|V|$.

Un altro elemento da determinare è la funzione di costo. Si è detto che il costo di un ciclo hamiltoniano è pari alla somma dei costi (o lunghezze) degli archi che lo compongono. L'immediata estensione è la seguente: il costo di un sottoinsieme di archi T è pari alla somma dei costi degli archi nel sottoinsieme, e cioè $w(T) = \sum_{e \in T} w_e$.

Siamo ora in grado di applicare l'algoritmo greedy al grafo di Fig. 8.6.

Inizializzazione.

a. $T_0 = \emptyset$ $(w(T_0) = 0)$. $i = 1$.

Iterazione 1.

b. Poichè ogni arco del grafo può essere aggiunto a T_0 soddisfacendo le condizioni di soluzione parziale, scegliamo l'arco che costa di meno, cioè l'arco $(3, 5)$ di costo $w_{3,5} = 1$.

c. Poni $T_1 = T_0 \cup \{(3, 5)\}$ $(w(T_1) = 1)$.

d. Poichè T_1 non è un ciclo hamiltoniano non possiamo fermarci.

e. Poni $i = 2$.

Iterazione 2.

b. Anche in questo caso ogni arco (in $A - \{(3,5)\}$) può essere aggiunto a T_1 soddisfacendo le condizioni di soluzione parziale; scegliamo l'arco che costa di meno, cioè l'arco $(2,4)$ di costo $w_{2,4} = 2$.

c. Poni $T_2 = T_1 \cup \{(2,4)\} = \{(3,5),(2,4)\}$ $(w(T_2) = 3)$.

d. T_2 non è un ciclo hamiltoniano.

e. Poni $i = 3$.

Iterazione 3.

b. Tutti gli archi residui possono essere aggiunti e quindi scegliamo l'arco meno costoso in $A - T_2$: l'arco $(2,5)$ di costo $w_{2,5} = 3$. Si osservi che anche l'arco $(4,5)$ ha costo $w_{4,5} = 3$. La scelta fra due archi equivalenti può essere fatta casualmente. Tuttavia, in altri casi si può decidere un criterio di *risoluzione dei pareggi* ("*tie breaking rule*"), legato ovviamente al tipo di problema.

c. Poni $T_3 = T_2 \cup \{(2,5)\} = \{(3,5),(2,4),(2,5)\}$ $(w(T_3) = 6)$.

d. T_3 non è un ciclo hamiltoniano.

e. Poni $i = 4$.

Iterazione 4.

b. Non tutti gli archi in $A - T_3$ possono essere aggiunti a T_3 soddisfacendo le condizioni di soluzione parziale; infatti, se aggiungiamo l'arco $(3,4)$ si crea il ciclo su quattro nodi $(2,3),(3,4),(4,5),(5,2)$, violando così la condizione (2) di soluzione parziale. Se aggiungiamo l'arco $(1,2)$, invece, avremo tre archi incidenti nel nodo 2. Ragionamenti analoghi valgono per gli archi $(1,5),(2,3),(4,5)$. Quindi, gli unici archi che aggiunti non violano le condizioni per le soluzioni parziali, sono l'arco $(1,4)$ e l'arco $(1,3)$. Fra i due, scegliamo $(1,4)$ che ha costo minimo.

c. Poni $T_4 = T_3 \cup \{(1,4)\} = \{(3,5),(2,4),(2,5),(1,4)\}$ $(w(T_4) = 11)$.

d. T_4 non è un ciclo hamiltoniano.

e. Poni $i = 5$.

Iterazione 5.

b. L'unico arco selezionabile è l'arco $(1,3)$.

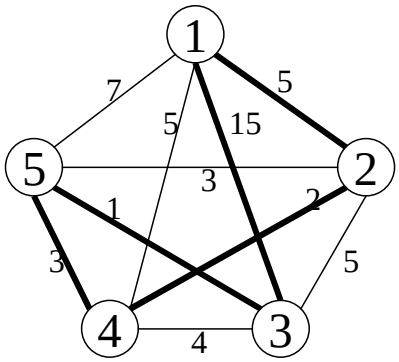
c. Poni $T_5 = T_4 \cup \{(1,3)\} = \{(3,5),(2,4),(2,5),(1,4),(1,3)\}$ $(w(T_5) = 26)$

d. T_5 è un ciclo hamiltoniano: STOP.

8.3.2 Applicazione dell'algoritmo greedy generico al problema del Partizionamento di Grafi (clustering).

Si è visto come il problema di clustering consiste nel partizionare i nodi di un grafo $G = (V, A)$ in k classi $\mathcal{C} = \{C_1, C_2, \ldots, C_k\}$ in modo da minimizzare la somma dei costi degli archi appartenenti alla stessa classe. Una soluzione S (detta *k-partizione*) può essere rappresentata come un insieme di coppie (v, c), ove $v \in V$ è un nodo del grafo e $c \in \{C_1, \ldots, C_k\}$ è un possibile cluster. Quindi, l'insieme di base B è l'insieme di tutte le coppie $(nodo, cluster)$ e cioè $B = \{(v, c) : v \in V, c \in \mathcal{C}\}$. Una soluzione $S \subseteq B$ è quindi un sottoinsieme di B con la proprietà che, per ogni nodo $v \in V$ esiste una e una sola

Figura 8.8: Soluzione greedy



coppia $(u, c) \in S$ tale che $u = v$. Una conseguenza di ciò è che S conterrà esattamente $|V|$ coppie (cioè $|S| = |V|$). Inoltre, per la nostra applicazione (ogni classe corrisponde a una vettura), le classi hanno una cardinalità massima prefissata. Le soluzioni parziali sono tutti i sottoinsiemi T di B tali che, per ogni nodo $v \in V$, esisterà al più una coppia $(u, c) \in T$ tale che $u = v$. In altri termini, le soluzioni parziali sono k-partizioni parziali, cioè k-partizioni di un sottoinsieme dei nodi di G. Il costo di una soluzione parziale (più in generale, di un sottoinsieme di B) è calcolato ancora una una volta come la somma dei costi degli archi appartenenti a una stessa classe.

Supponiamo adesso di avere 5 persone in attesa e due taxi disponibili. Inoltre, supponiamo che ogni taxi possa ospitare al massimo 3 persone. Supponiamo infine che il grafo dei clienti sia quello di Figura 8.6. Vediamo adesso come sia possibile calcolare una 2-partizione (partizione in due classi, bi-partizione) del grafo in Figura 8.6.

Poichè $V = \{1, \ldots, 5\}$ e $\mathcal{C} = \{C_1, C_2\}$, l'insieme di base sarà $B = \{(1, C_1), (1, C_2), (2, C_1), (2, C_2), (3, C_1), (3, C_2), (4, C_1), (4, C_2), (5, C_1), (5, C_2)\}$. E' importante osservare che in questo caso i pareggi vengono risolti individuando una funzione di costo ad hoc per motivi che verranno discussi di seguito. Infine, poichè scegliere una coppia di B corrisponde ad assegnare un nodo a un cluster, di seguito adotteremo una terminologia (semplificata) e a fianco della locuzione "scegliere la coppia (v, C_j)", diremo anche "assegnare il nodo v al cluster C_j, e via di seguito.

Inizializzazione.

a. $T_0 = \emptyset$ ($w(T_0) = 0$). $i = 1$.

   (*Equivalentemente: crea* 2 *classi vuote* C_1, C_2*).*

Iterazione 1.

b. Poichè ogni coppia può essere aggiunta a T_0 senza incrementare il costo di T_0 (ovvero, ogni nodo può essere assegnato a una qualunque classe a costo nullo), ho bisogno di una regola per decidere quale nodo assegnare a quale classe. Nell'esempio del TSP si era optato per una scelta di tipo "random". In questo esempio si preferisce una regola che tenga in conto della particolare struttura del problema. Una considerazione euristica è la seguente: conviene innanzitutto "sistemare" in qualche classe della partizione quei nodi su cui incidono gli archi più costosi in modo da poter scegliere per essi classi abbastanza (se non del tutto) vuote. Quindi la regola di tie breaking sarà: scegli il nodo per cui la somma degli archi incidenti è massima. Di seguito, indicheremo con $g(v)$ la somma dei costi degli archi incidenti nel nodo v. E' facile verificare che $g(1) = 32$, $g(2) = 17$, $g(3) = 25$, $g(4) = 14$, $g(5) = 14$. Scegliamo quindi il nodo 1. Poichè le coppie $(1, 1)$ e $(1, 2)$ sono equivalenti dal punto di vista del costo delle partizioni corrispondenti (e cioè è indifferente

assegnare il nodo 1 alla classe C_1 o alla classe C_2, possiamo scegliere in modo casuale. Scegliamo quindi la coppia $(1, C_1)$ (assegnamo cioè il nodo 1 alla classe C_1).

c. Poni $T_1 = T_0 \cup \{(1, C_1)\}$ $(w(T_1) = 0)$.

Equivalentemente $C_1 = \{1\}$, $C_2 = \emptyset$.

d. Poichè T_1 non è una partizione (completa) dei nodi di G, si continua.

e. Poni $i = 2$.

Iterazione 2.

b. Ogni coppia in B che non contiene il nodo 1 può essere aggiunta a T_1 soddisfacendo le condizioni di soluzione parziale. Tuttavia, ogni coppia del tipo (v, C_1), che corrisponde ad assegnare il nodo v alla classe C_1, produce un incremento di costo pari al costo dell'arco $(1, v)$ (infatti il nodo 1 appartiene alla classe 1); al contrario, ogni nodo può essere assegnato alla classe 2 senza incrementi di costo (infatti la classe C_2 è ancora vuota). Infine, poichè ogni coppia di tipo (v, C_2) produce un incremento di costo nullo, dobbiamo usare ancora una volta la regola di tie breaking e scegliere quindi il nodo di costo massimo, ovvero il nodo 3 $(g(3) = 25)$. Scegliamo quindi la coppia $(3, 2)$, assegnando in questo modo il nodo 3 alla classe 2.

c. Poni $T_2 = T_1 \cup \{(3, C_2)\} = \{(1, C_1), (3, C_2)\}$ $(w(T_2) = 0)$.

$C_1 = \{1\}$, $C_2 = \{3\}$.

d. T_2 non è una partizione di V.

e. Poni $i = 3$.

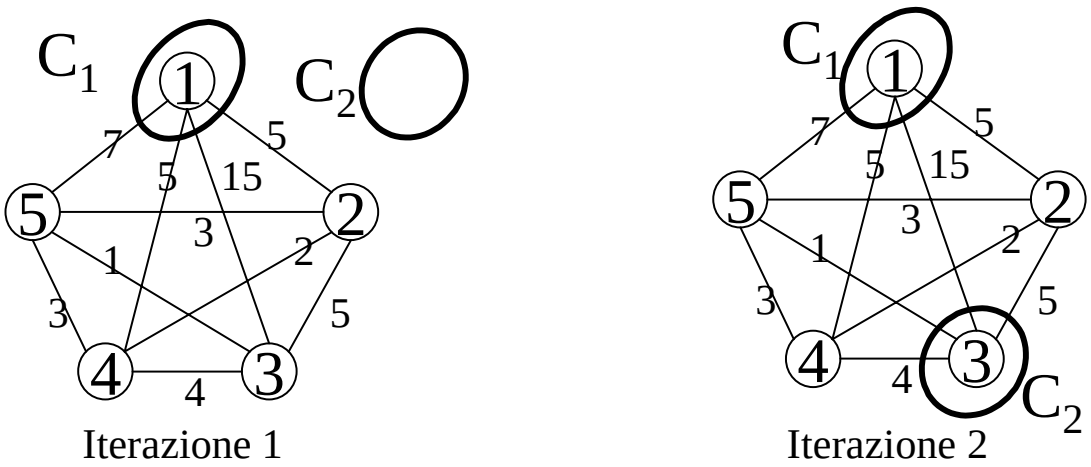


Figura 8.9: Evoluzione delle classi I

Iterazione 3.

b. In questo caso, ogni coppia aggiungibile a T_2, e cioè tutte quelle coppie che non contengono il nodo 1 e il nodo 3, producono un incremento di costo positivo. Ad esempio, se il nodo 2 viene assegnato alla classe 1, l'incremento di costo è pari al costo dell'arco (1,2), ovvero 5. E' faicle vedere che il minimo incremento di costo si ottiene scegliendo la coppia $(5, C_2)$, ovvero assegnando il nodo 5 alla classe C_2. Non ci sono pareggi.

c. Poni $T_3 = T_2 \cup \{(5, C_2)\} = \{(1, C_1), (3, C_2), (5, C_2)\}$ $(w(T_3) = 5)$.

$C_1 = \{1\}$, $C_2 = \{3, 5\}$.

d. T_3 non è una partizione.

e. Poni $i = 4$.

Iterazione 4.

b. Tutte le coppie contenenti i nodi $1, 3$ e 5 non possono essere aggiunte. L'incremento di costo associato alla coppia $(2, C_1)$ è pari a 5, per la coppia $(2, C_2)$ è pari a $3+5 = 8$, per la coppia $(4, C_1)$ è pari a 5, per la coppia $(4, C_2)$ è pari a $3+4 = 7$. Fra la coppia $(2, C_1)$ e la coppia $(4, C_1)$ si sceglie $(2, C_1)$ perchè $g(2) = 17 > g(4) = 14$.

c. Poni $T_4 = T_3 \cup \{(2, C_1)\} = \{(1, C_1), (3, C_2), (5, C_2), (2, C_1)\}$ $(w(T_3) = 10)$.

$C_1 = \{1, 2\}$, $C_2 = \{3, 5\}$.

d. T_4 non è una partizione.

e. Poni $i = 5$.

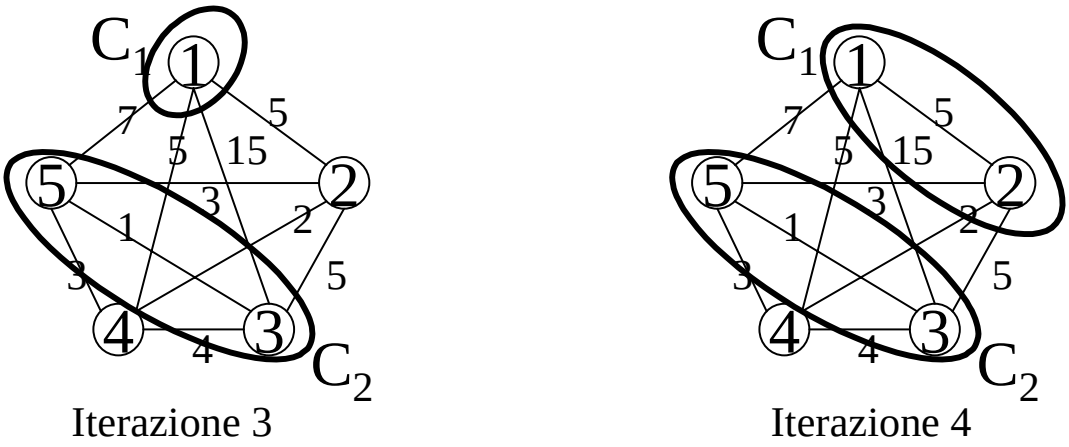


Figura 8.10: Evoluzione delle classi II

Iterazione 5.

b. Le uniche coppie selezionabili sono $(4, C_1)$ e $(4, C_2)$. L'incremento di costo corrispondente a $(4, C_1)$ è pari a 7 ed è uguale all'incremento prodotto dalla coppia $(4, C_2)$. Possiamo scegliere indifferentemente.

c. Poni $T_4 = T_3 \cup \{(2, C_1)\} = \{(1, C_1), (3, C_2), (5, C_2), (2, C_1), (4, C_2)\}$ $(w(T_5) = 17)$

$C_1 = \{1, 2\}$, $C_2 = \{3, 4, 5\}$.

d. T_5 è una partizione: STOP.

Un'ultima osservazione riguarda la forma con cui vengono generalmente rappresentati gli algoritmi greedy. Infatti, raramente si fa esplicito riferimento all'insieme di base (o alle soluzioni ammissibili come sottoinsiemi dell'insieme di base). Normalmente gli algoritmi vengono presentati in una forma molto

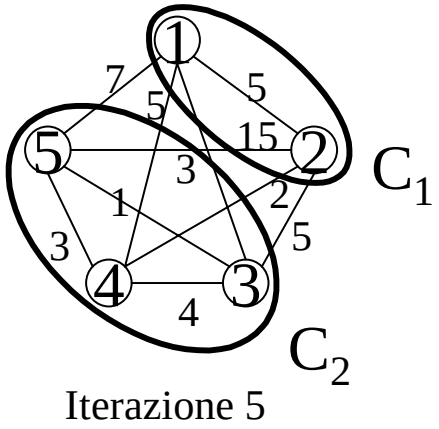


Figura 8.11: Evoluzione delle classi III

più vicina alla descrizione "naturale" del problema. Come visto nell'esempio del problema di clustering, ogni coppia dell'insieme di base rappresenta in realtà l'assegnazione di un nodo a un cluster. Una forma alternativa e più leggibile dell'algoritmo greedy generico applicato al problema di clustering è la seguente.

Problema. Sia dato un grafo $G = (V, A)$, con costi $c_a \in I\!R$ associati a ciascun arco $a \in A$. Trovare la partizione dell'insieme dei nodi V in k classi $C_1, \ldots, C_k$ che minimizza il costo $\sum_{i=1}^{k} \sum_{u \in C_i, v \in C_i} c_{uv}$.

Algoritmo Greedy particolarizzato per il problema di clustering
Note: l'insieme W utilizzato nell'algoritmo rappresenta l'insieme dei nodi già assegnati (nelle iterazioni precedenti a quella corrente) a qualche cluster.

a. *Inizializzazione.* Poni $C_i = \emptyset$ per $i = 1, \ldots, k$. Poni $W = \emptyset$. Poni $i = 1$.

b. *Iterazione i-esima.* Scegli un nodo $u \in V - W$ e assegna u a un cluster C_j in modo da minimizzare l'incremento di costo della partizione parziale, e cioè la quantità $\sum_{v \in C_j} c_{uv}$. In caso di pareggi, scegli il nodo u che massimizza $g(u)$.

c. Poni $C_j = C_j \cup \{u\}$.

d. *Terminazione* Se $W = V$ ogni nodo è stato assegnato a un cluster: STOP.

e. Altrimenti Poni $i = i + 1$. Va al passo b.

E' in questa forma che verranno di seguito descritti gli algoritmi di ricerca locale per l'ottimizzazione combinatoria.

8.3.3 Una diversa forma del generico algoritmo greedy.

Gli esempi di applicazione dell'algoritmo greedy sopra elencati hanno una specifica struttura delle soluzioni e delle soluzioni parziali. In particolare, tutte le soluzioni parziali generate dal greedy - tranne l'ultima - non sono soluzioni ammissibili per il problema. Inoltre la funzione obiettivo peggiora a ogni iterazione, ma noi siamo comunque obbligati a continuare i passi finchè la sottosoluzione diventa una soluzione del problema.

Per altri problemi di ottimizzazione, come il problema dell'accoppiamento, le soluzioni parziali sono anche soluzioni del problema originario. Infatti, se $M = \{e_1, \ldots, e_q\}$ è un matching, allora ogni sottoinsieme di M è ancora un matching (e cioè una soluzione ammissibile). In questo caso si tende in generale

a estendere le soluzioni parziali perchè tipicamente la funzione obiettivo migliora all'aumentare del numero di elementi contenuti nella soluzione. Per problemi di questo tipo, il generico algoritmo greedy (per problemi di minimizzazione) viene riscritto nel modo seguente:

Algoritmo Greedy II

a. *Inizializzazione.* Poni $T_0 = \emptyset$. Poni $i = 1$.

b. *Iterazione i-esima.* Scegli $e \in B - T_{i-1}$ tale che $T_{i-1} \cup \{e\}$ sia una soluzione parziale e $w(T_{i-1} \cup \{e\})$ sia minimo.

c. Se per ogni $e \in B - T_{i-1}$, $T_{i-1} \cup \{e\}$ non é una soluzione ammissibile: STOP. T_{i-1} è la soluzione greedy.

d. Altrimenti se $w(T_{i-1}) < w(T_{i-1} \cup \{e\})$: STOP. T_{i-1} è la soluzione greedy.

e. Altrimenti Poni $T_i = T_{i-1} \cup \{e\}$. Poni $i = i + 1$. Vai al passo b.

Il passo [c.] serve ad assicurare che l'algoritmo termini qualora ogni elemento residuo non può essere aggiunto alla soluzione parziale corrente senza violare il vincolo di essere soluzione parziale. Il passo [d.] assicura invece che se la prossima soluzione parziale è peggiore di quella attuale essa non venga generata e l'algoritmo termini. Si osservi infine che in questa forma l'algoritmo greedy puó essere applicato anche a problemi in cui le sottosoluzioni via via generate non sono soluzioni ammissibili. Si consideri ad esempio il problema del Commesso Viaggiatore. A ogni iterazione generiamo sottosoluzioni che non sono cicli hamiltoniani finché, all'iterazione n-esima, genereremo il nostro ciclo hamiltoniano. Secondo lo schema qui sopra riportato, all'iterazione $n + 1$ l'algoritmo terminerá in quanto l'aggiunta di un qualunque arco a un ciclo hamiltoniano non produce una sottosoluzione: quindi sará verificato il test di terminazione al passo [c.].

E' facile vedere che per adattare il precedente algoritmo a problemi di massimizzazione è sufficiente cambiare il passo [b.] scegliendo l'elemento e tale che $w(T_{i-1} \cup \{e\})$ sia massimo e invertendo il senso della disequazione nel test al passo [d.] che va' riscritto come:

[d.] Altrimenti se $w(T_{i-1}) > w(T_{i-1} \cup \{e\})$: STOP. T_{i-1} è la soluzione greedy.

Si consideri l'esempio di Fig. 8.12, ove si voglia calcolare l'accoppiamento di peso massimo.

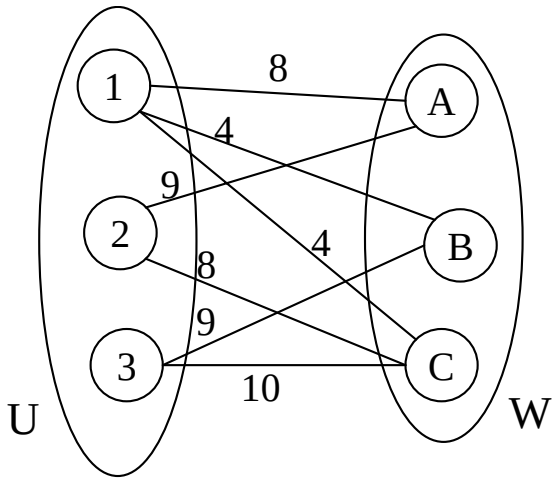


Figura 8.12: Un problema di accoppiamento massimo

Inizializzazione.

a. $T_0 = \emptyset$ ($w(T_0) = 0$). $i = 1$.

Iterazione 1.

b. Poichè ogni arco può essere aggiunto a T_0 mantenendo la proprietà di essere una soluzione parziale, scegliamo l'arco di peso massimo $(3, C)$.

c. L'arco esiste, il test di terminazione non è soddisfatto.

d. $10 = w(T_0 \cup \{(3, C)\}) > w(T_0)$ e il secondo test di terminazione non è soddisfatto (si ricordi che si tratta di un problema di massimizzazione).

e. Poni $T_1 = T_0 \cup \{(3, C)\}$ ($w(T_1) = 10$). Poni $i = 2$.

Iterazione 2.

b. Gli archi $(1, C)$ e $(2, C)$ non possono essere aggiunti a T_1 in quanto incidenti in C (esiste già un arco in T_1 incidente in C). Analogamente, l'arco $(3, B)$ non può essere aggiunto perchè incidente in 3. Fra gli archi residui, quello di peso massimo è l'arco $(2, A)$, di peso 9.

c. L'arco esiste, il test di terminazione non è soddisfatto.

d. $18 = w(T_1 \cup \{(2, A)\}) > w(T_1)$ e il secondo test di terminazione non è soddisfatto.

e. Poni $T_2 = T_1 \cup \{(2, A)\}$ ($w(T_2) = 19$). Poni $i = 3$.

Iterazione 3.

b. L'unico arco che può essere aggiunto a T_2 è l'arco $(1, B)$, di peso 4.

c. L'arco esiste, il test di terminazione non è soddisfatto.

d. $23 = w(T_2 \cup \{(1, B)\}) > w(T_2)$ e il secondo test di terminazione non è soddisfatto.

e. Poni $T_3 = T_2 \cup \{(1, B)\}$ ($w(T_3) = 23$). Poni $i = 4$.

Iterazione 4.

b. Nessun arco può essere aggiunto a T_3.

c. L'arco non esiste, il test di terminazione è soddisfatto. T_3 è la soluzione greedy.

La soluzione trovata dall'algoritmo greedy per il grafo di Fig. 8.12 è mostrata in Fig. 8.13.

8.4 Ricerca Locale

Si è osservato come uno degli elementi caratterizzanti dell'algoritmo greedy sia la natura irrevocabile della scelta greedy. Tuttavia, assai spesso apportando "piccole" modifiche alla soluzione greedy si possono avere miglioramenti nel valore della funzione obiettivo. Si consideri l'esempio del ciclo hamiltoniano di Fig. 8.14.a, che è quello prodotto dall'algoritmo greedy (si confronti con la Fig. 8.6). Il costo del ciclo è pari a 26. Per comodità gli archi non appartenenti al ciclo non sono stati rappresentati nella figura. Se rimuoviamo gli archi $(1, 3)$ e $(4, 5)$ otteniamo la soluzione parziale di Fig. 8.14.b. Esistono due soli modi di completare questo insieme di archi per renderlo un ciclo hamiltoniano. Il primo è re-inserire gli archi $(1, 3)$ e $(4, 5)$, riottenendo così il ciclo di partenza. Il secondo è invece scegliere gli archi $(3, 4)$ e $(1, 5)$, ottenendo così il ciclo hamiltoniano di Fig. 8.14.c.

Si osservi che questo ciclo ha costo 19, cioè un costo inferiore al ciclo di partenza. Quindi, con una piccola modifica della soluzione greedy (cambiando solo due archi) si è ottenuto un sostanziale

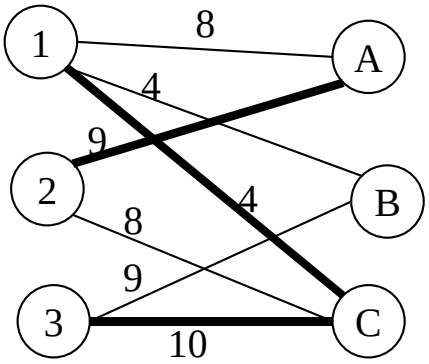


Figura 8.13: Matching massimo

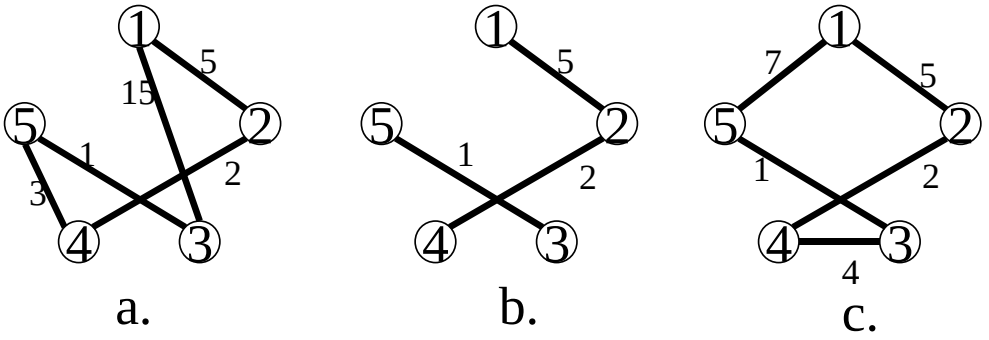


Figura 8.14: Trasformazione di un ciclo hamiltoniano

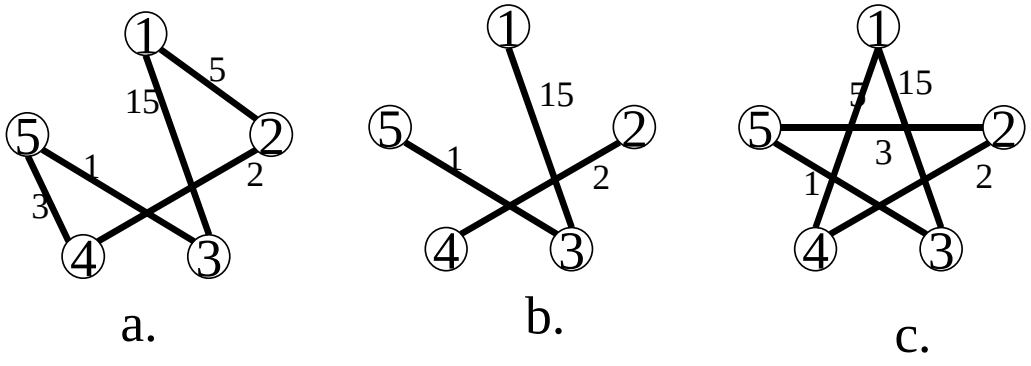


Figura 8.15: Un'altra possibile trasformazione della soluzione greedy

miglioramento della funzione obiettivo. Proviamo ora, sempre partendo dal grafo di Fig.8.14 a sostituire la coppia $(1,2)$ e $(4,5)$. In questo caso, l'unico modo per ricostruire un ciclo hamiltoniano diverso da quello di partenza è aggiungere gli archi $(1,4)$, $(2,5)$ (si veda la Fig. 8.15).

Questo nuovo ciclo tuttavia ha lo stesso costo di quello di partenza. Cerchiamo di generalizzare ciò che è stato fatto. Innanzitutto osserviamo che, se rimuoviamo dal ciclo di partenza due archi adiacenti (ovvero incidenti in uno stesso nodo), esiste un solo modo per completare la soluzione parziale così ottenuta consistente nel re-inserire gli archi appena eliminati. Al contrario, se rimuoviamo due archi non adiacenti (cioè due archi che non hanno nodi in comune), la soluzione parziale può essere completata in due modi distinti, uno dei quali produce un nuovo ciclo. In generale, se indichiamo con $H_0 = \{(u_1, u_2), (u_2, u_3), \ldots, (u_{q-1}, u_q), (u_q, u_1)\}$ un generico ciclo hamiltoniano, allora il meccanismo di generazione di un nuovo ciclo hamiltoniano può essere descritto come segue (si veda la Fig. **??**):

1. Scegli una coppia di archi non adiacenti (u_i, u_{i+1}) e (u_j, u_{j+1}).
2. Rimuovi la coppia di archi dal ciclo
3. Aggiungi i due nuovi archi (u_i, u_j) e (u_{i+1}, u_{j+1}).

Poichè l'operazione coinvolge lo scambio di due archi con altri due archi, verrà chiamata 2-*scambio* (in inglese "*2-exchange*").

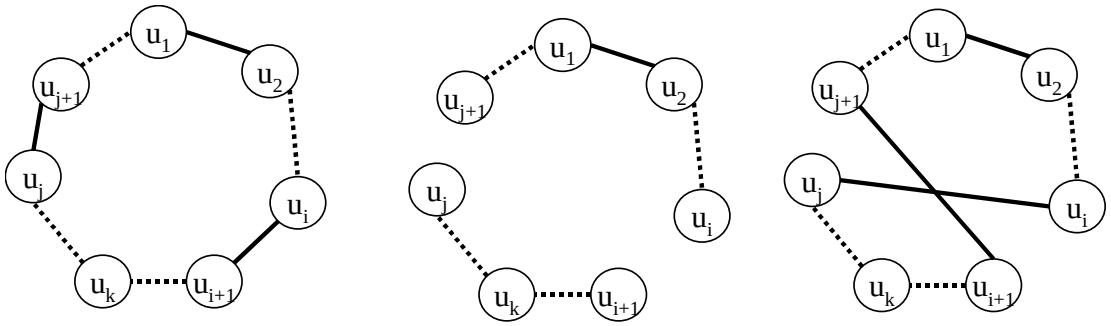


Figura 8.16: Scambio di archi generalizzato

Negli esempi delle figure 8.14 e 8.15 ci siamo limitati a provare la sostituzione di due coppie di archi del ciclo originario H_0. Un'ovvia generalizzazione consiste nel provare tutti i possibili 2-scambi (in corrispondenza a tutte le possibili coppie di archi non adiacenti di H_0) e scegliere alla fine il 2-scambio che produce il ciclo hamiltoniano di costo minimo. Formalmente, possiamo definire la seguente procedura:

1. Per ogni coppia di archi non adiacenti di H_0 calcola la lunghezza del ciclo hamiltoniano corrispondente
2. Scegli il migliore dei cicli così prodotti.

Si osservi che il numero di cicli hamiltoniani esaminati non è molto grande. Infatti, il ciclo iniziale H_0 contiene esattamente n archi (ove $n = |V|$). Quante sono le coppie di archi non adiacenti? Per ogni arco (u, v), ci sono esattamente due archi adiacenti (un arco incidente in u e un arco incidente in v). Quindi, il numero di archi non adiacenti è $n-3$. Quindi, per ognuno degli n archi del ciclo, esistono esattamente $n-3$ 2-scambi distinti. Il numero di cicli generati per 2-scambio a partire da H_0 è quindi minore o uguale a $n(n-3)$ (esattamente è pari a $n(n-3)/2$).

Ora, se indichiamo con H_1 il migliore dei cicli hamiltoniani ottenibili per 2-scambio da H_0 saranno possibili due casi:

a. $w(H_1) < w(H_0)$, cioè il ciclo trovato è migliore di quello di partenza.

b. $w(H_1) \geq w(H_0)$, cioè il ciclo trovato non è migliore di quello di partenza.

Supponiamo adesso di trovarci nel caso a., e cioè H_1 è migliore di H_0. Nulla ci impedisce di ricominciare da capo, nella speranza di migliorare ulteriormente il ciclo hamiltoniano. Quindi possiamo riapplicare ad H_1 la procedura di ricerca del miglior 2-scambio, eventualmente identificando un nuovo ciclo H_2 migliore di H_1 (e quindi di H_0); infine, la procedura può essere re-iterata finchè si riesce a identificare un ciclo migliore del precedente.

Siamo in grado di descrivere adesso una delle euristiche più efficienti per la ricerca di un ciclo hamiltoniano di costo (lunghezza) minimo, la cosiddetta euristica $2 - Opt$. Di seguito, denotiamo con $\mathcal{T}$ la famiglia dei cicli hamiltoniani di G.

Algoritmo 2-opt per il TSP

Inizializzazione

a. Scegli un ciclo hamiltoniano iniziale $H_0 \in \mathcal{T}$ (ad esempio mediante l'euristica greedy). Poni $i = 1$.

Iterazione i-esima

b. Sia H_i il più corto ciclo hamiltoniano ottenibile da H_{i-1} per 2-scambio.

c. Se $w(H_i) \geq w(H_{i-1})$ allora STOP. H_{i-1} è il miglior ciclo trovato fino a questo punto.

d. Altrimenti poni $i = i + 1$ a va al passo b.

Fine iterazione i-esima

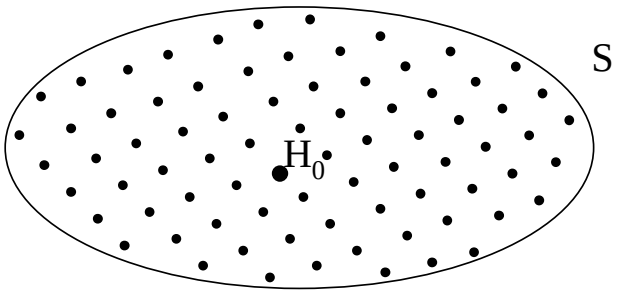


Figura 8.17: Insieme delle soluzioni ammissibili

Quanto abbiamo visto finora applicato al caso del TSP può essere generalizzato a ogni problema di ottimizzazione combinatoria. Esaminiamo più nel dettaglio ciò che è stato fatto. Siamo partiti da un ciclo hamiltoniano iniziale H_0 (*soluzione iniziale*) e abbiamo esaminato un certo numero di cicli hamiltoniani "non troppo diversi" da H_0. In particolare, abbiamo esaminato tutti quei cicli che hanno esattamente $n - 2$ archi in comune con H_0. Chiaramente, questo è solo un piccolo sottoinsieme dell'insieme di tutti i cicli hamiltoniani del grafo (che, come si è detto, ha dimensione $n!$). I cicli hamiltoniani appartenenti a questo sottoinsieme hanno la caratteristica di trovarsi, in qualche senso, "vicino" a H_0. Graficamente,

possiamo rappresentare tutti i cicli hamiltoniani come dei punti in uno spazio di soluzioni S (vedi Fig. 8.17).

Il ciclo H_0 sarà un punto appartenente a questo sottoinsieme. L'insieme di tutti i cicli "vicini" ad H_0 è evidenziato in figura 8.18 da un ovale centrato in H_0 e contenuto in S. Questo insieme è detto in generale *l'intorno* di H_0, ed è indicato come $N(H_0)$. In questo caso stiamo parlando di un particolare intorno, quello definito dal 2-scambio. Nulla ci impedisce di definire intorni più grandi (ad esempio, l'insieme di tutti i cicli hamiltoniani ottenibili sostituendo 3 archi) o più piccoli. In genere, è bene che la dimensione dell'intorno non cresca troppo per poter effettuare efficientemente la visita (e cioè la generazione) dei cicli hamiltoniani in esso contenuti, il calcolo della funzione obiettivo e quindi la scelta della migliore soluzione appartenente all'intorno.

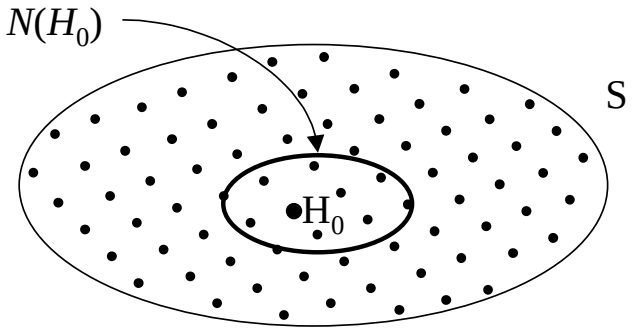


Figura 8.18: Intorno di una soluzione

Continuando con questo esempio, è semplice interpretare l'euristica 2-opt come traiettoria di punti all'interno di questo spazio di soluzioni. Infatti, il metodo può essere interpretato come segue: parto da H_0, visito il suo intorno, mi sposto nella migliore soluzione trovata H_1, visito l'intorno di H_1, mi sposto nella migliore soluzione trovata H_2, etc. La traiettoria termina quando l'ultima soluzione visitata è migliore (non peggiore) di tutte quelle nel suo intorno.

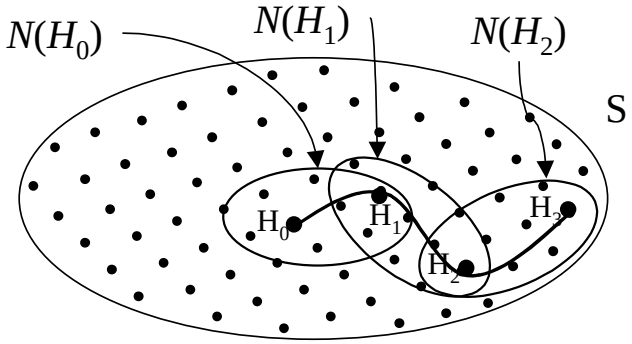


Figura 8.19: Traiettoria nello spazio delle soluzioni

La Figura 8.19 rappresenta appunto sia la traiettoria di soluzioni che la sequenza di intorni visitata. Si capisce a questo punto il significato del termine "Ricerca Locale". La ricerca (nello spazio delle

soluzioni) avviene localmente, negli intorni delle soluzioni sulla traiettoria seguita. L'ultima soluzione è detta *minimo locale*, a sottolineare il fatto che questa soluzione non è l'ottimo globale (su tutto l'insieme delle soluzioni) ma solo l'ottimo locale, relativo alla famiglia di intorni definita.

8.4.1 Algoritmo generico di Ricerca Locale

.

L'euristica 2-Opt per il TSP è un esempio di euristica di *Ricerca Locale*. Il primo passo per costruire un'euristica consiste nel definire un opportuno intorno delle soluzioni. Chiamando con $S \in \mathcal{S}$ una soluzione ammissibile per il problema di ottimizzazione, si tratta di stabilire come costruire a partire da S un insieme di soluzioni $N(S)$. La definizione dell'intorno avviene tipicamente mediante la specificazione di una particolare operazione, detta *mossa*, da applicare alla soluzione corrente per ottenere le nuove soluzioni. Per esempio, nell'euristica 2-Opt per il Commesso Viaggiatore, la mossa era il cosiddetto 2-scambio (sostituzione di due archi non adiacenti con due nuovi archi). Consideriamo come nuovo esempio il problema di k-partizionamento. Per semplificare la trattazione, considereremo il caso senza vincoli di cardinalità sulle partizioni. Le soluzioni del problema di k-clustering sono dunque tutte le k-partizioni dell'insieme di nodi V del grafo. Quindi, una soluzione è una specifica k-partizione, che possiamo indicare come $S = \{C_1, \ldots, C_k\}$. Un modo per definire l'intorno di una partizione è il seguente: l'intorno $N(S)$ di una partizione S è l'insieme di tutte le partizioni ottenibili da S spostando uno e un solo nodo dalla sua classe di appartenenza a una nuova classe. La mossa in questo caso consiste in:

- scegli un nodo $v_j \in V$. Sia C_l la classe di appartenenza di v_j.
- genera una nuova partizione $S' = \{C'_1, C'_2, \ldots, C'_l, \ldots, C'_q\}$ nel seguente modo:
  - Scegli un indice r con $r \in \{1, \ldots, q\}$ e $r \neq l$.
  - Poni $C'_i = C_i$ per $i = 1, \ldots, q$ e $i \neq r$, $i \neq l$.
  - Poni $C'_l = C_l - \{v_j\}$, $C'_r = C_r \cup \{v_j\}$.

Se consideriamo ancora l'esempio di Fig. 8.11, la soluzione prodotta dall'algoritmo greedy è: $S_0 = \{C_1 = \{1, 2\}, C_2 = \{3, 4, 5\}\}$. L'intorno di S_0 è formato da cinque soluzione $N(S_0) = \{S_1, S_2, S_3, S_4, S_5\}$, ove $S_1 = \{C_1 = \{1, 2, 3\}, C_2 = \{4, 5\}\}$, $S_2 = \{C_1 = \{1, 2, 4\}, C_2 = \{3, 5\}\}$, $S_3 = \{C_1 = \{1, 2, 5\}, C_2 = \{3, 4\}\}$, $S_4 = \{C_1 = \{2\}, C_2 = \{1, 3, 4, 5\}\}$, $S_5 = \{C_1 = \{1\}, C_2 = \{2, 3, 4, 5\}\}$. E' facile anche verificare che $w(S_0) = 13$, $w(S_1) = 18$, $w(S_2) = 13$, $w(S_3) = 21$, $w(S_4) = 35$. $w(S_5) = 16$.

Introdotta la definizione di intorno di una soluzione, siamo finalmente in grado di descrivere formalmente il generico algoritmo di ricerca locale.

Algoritmo di Ricerca Locale

Inizializzazione

a. Scegli una soluzione iniziale $S_0 \in \mathcal{S}$ (ad esempio mediante l'euristica greedy). Poni $i = 1$.

Iterazione i-esima

b. Sia $S_i \in N(S_{i-1})$ la migliore soluzione nell'intorno di S_{i-1}.

c. Se $w(S_i) \geq w(S_{i-1})$ allora STOP. S_{i-1} è il minimo locale.

d. Altrimenti poni $i = i + 1$ a va al passo b.

Fine iterazione i-esima

Ovviamente é sempre possibile interrompere la ricerca locale prima di aver raggiunto un ottimo locale, ad esempio dopo aver eseguito un prefissato massimo numero di iterazioni.

8.5 Estensione del modello al caso Roma Centro - Fiumicino Areoporto

Finora abbiamo trattato il caso del servizio da Fiunmicino Areoporto al centro città. In questo contesto le vetture, una volta lasciati i passeggeri alle loro destinazioni, tornano all'areoporto per prelevare nuovi clienti. Tuttavia nulla impedisce loro di trasportare passeggeri da Roma centro all'areoporto di Fiumicino effettuando il servizio anche nel viaggio di ritorno. In effetti, un servizio del genere è allo studio. Il modo più semplice per affrontare il problema consiste nel considerare le due tratte (Fiumicino-Roma Centro e Roma Centro - Fiumicino) come due problemi distinti.

Il caso Roma Centro - Fiumicino presenta qualche elemento di complessità addizionale. Infatti, in questo caso il servizio viene effettuato su prenotazione e i clienti vanno raccolti rispettando il più possibile gli orari stabiliti. In ogni caso, ancora una volta di stratta di un problema di clustering (assegnazione dei clienti alle vetture) e di un problema di Comesso Viaggiatore (scelta del percorso ottimo) risolti in sequenza. Quindi, alla fine del processo di ottimizzazione avremo un certo numero di cicli hamiltoniani $H = \{H_1, H_2, \ldots, H_q\}$ ciascuno rappresentante i clienti da prelevare e la sequenza da seguire per ciascun taxi. A ogni ciclo avremo inoltre associato un tempo d'inizio raccolta t_i per $i = 1, \ldots, q$, ovvero il primo cliente di ogni ciclo H_i deve essere prelevato al tempo t_i. Si tratta ora di assegnare ciascun ciclo a un taxi proveniente dall'areoporto e che abbia già concluso il suo giro di consegne. Quale vettura conviene assegnare al ciclo H_j (per $j = 1, \ldots, q$)? Sicuramente un taxi che abbia finito le consegne in tempo per prelevare il primo cliente al tempo t_j (ma non troppo prima, altrimenti il taxi dovrà aspettare a lungo). Inoltre, ci converrà assegnare un taxi che consegni il suo ulimo passeggero in un nodo del grafo di Roma abbastanza vicino al nodo corrispondente al primo cliente del ciclo di ritorno. Allora, se chiamiamo con $A = \{A_1, A_2, \ldots, A_r\}$ l'insieme dei cicli hamiltoniani che corrispondono a taxi provenienti dall'areoporto di fiumicino che possono essere assegnati ai cicli di ritorno $\{H_1, H_2, \ldots, H_q\}$, si tratterà innanzitutto di definire un insieme di pesi p_{ij} che rappresentino il vantaggio di assegnare il taxi che serve A_i al percorso di ritorno H_j. Ad esempio, se d_{ij} è la distanza fra l'ultimo passeggero di A_i e il primo passeggero di H_j, potremmo porre $p_{ij} = -d_{ij}$. Potremmo aggiungere nel peso anche un fattore relativo al tempo di attesa - minori tempi di attesa per il conducente sono da preferire). A questo punto vogliamo scegliere come assegnare ogni vettura che arriva dall'areoporto a un ciclo verso l'areoporto. Costruiamo quindi un grafo bipartito $G = (V, E)$ ove l'insieme dei nodi sarà $V = H \cup A$, ed esiste un arco fra in nodo $H_j \in H$ e il nodo $A_i \in A$ se e solo se l'ultimo cliente di A_i viene lasciato in tempo utile per prelevare il primo cliente di H_j. Inoltre, associamo a ogni arco $(i, j) \in E$ il peso p_{ij} che rappresenta il vantaggio di assegnare il taxi del ciclo A_j al ciclo di ritorno H_i. E' facile convincersi che cercare l'assegnamento migliore corrisponde a risolvere un problema di accoppiamento massimo nel grafo G.