Grafi e reti di flusso

1 Modelli su grafi e reti di flusso

Una vasta classe di problemi di notevole rilevanza pratica può venire modellata tramite grafi o reti di flusso. In questo capitolo svilupperemo modelli con l'ausilio di questi strumenti, ma analizzeremo anche gli algoritmi per la soluzione dei vari problemi che incontreremo. Un grafo G viene generalmente definito da un insieme di nodi N e un insieme di archi A. Un arco è definito da una coppia di nodi. Vediamo una breve rassegna di problemi che ammettono una facile rappresentazione tramite grafi.

Progetto di una rete di telecomunicazione

Il centro di calcolo di ateneo viene decentrato in vari edifici del campus; in ogni centro sarà installato un computer. La messa in opera di una linea di trasmissione dati tra due centri ha un costo dipendente dalla distanza da coprire. Dovendo assicurare il passaggio dei dati tra le varie macchine, i sistemisti devono progettare la rete di comunicazione cercando di contenere le spese quanto più possibile.

Assegnamento di esercizi a studenti

Per l'esame di ricerca operativa sono stati preparati tanti esercizi quanti sono i candidati. Per ogni esercizio si conosce il grado di difficoltà percepito da ogni candidato. Si vuole attribuire un esercizio a ogni candidato in modo che non ci siano due candidati a cui è stato assegnato lo stesso esercizio e il grado di difficoltà massimo sia minimo (o che il grado di difficoltà minimo sia massimo!).

Il problema del postino

Un postino deve consegnare la corrispondenza agli abitanti delle case distribuite sui lati di un dato insieme di strade. Il postino, partendo dall'ufficio postale, per effettuare le consegne deve percorrere tutte le strade su entrambi i lati almeno una volta e, al termine, deve fare ritorno al punto di partenza. Il problema consiste nel trovare la sequenza di strade (con eventuali ripetizioni) da far percorrere al postino in modo che l'itinerario complessivo sia più breve possibile.

Itinerario del Giro d'Italia

I sindaci di varie città si sono messi d'accordo su dove fissare gli arrivi di tappa del Giro d'Italia. In particolare hanno stabilito che sia la partenza che l'arrivo devono aver luogo a Milano e che per motivi di ordine pubblico i corridori non possono passare più volte per una stessa città sede di tappa. I vari paesi d'Italia mettono in palio dei premi se il Giro li attraversa. L'organizzatore del Giro deve fissare l'itinerario in modo che ogni singola tappa non sia più lunga di 200 Km, e sia massimizzata l'entità complessiva dei premi.

Rinnovo di automobile

Un commesso viaggiatore deve pianificare l'acquisto e/o l'eventuale sostituzione dell'automobile di cui necessita per svolgere il proprio lavoro nei prossimi 5 anni. In pratica all'inizio di ogni anno deve decidere se sostituire l'auto con una nuova o affrontare l'anno e le spese di manutenzione con la vecchia.

2 Notazione e definizioni

Un grafo G viene generalmente definito da un insieme di nodi N e un insieme di archi A. Un arco è definito da una coppia di nodi, detti *estremi* dell'arco.

Se le coppie di nodi che definiscono gli archi sono non orientate, il grafo è detto *non orientato*. In fig. 1 viene riportata la rappresentazione del grafo non orientato $G=(N,A)$, $N=\{1,2,3,4,5\}$, $A=\{\{1,2\}, \{1,4\}, \{1,5\}, \{2,3\}, \{2,4\}, \{3,4\}, \{4,5\}\}$.

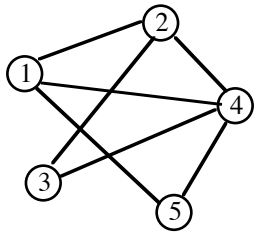

fig. 1: un grafo non orientato

Al contrario, se le coppie di nodi che definiscono gli archi sono orientate, il grafo è detto *orientato*. In fig. 2 viene riportata la rappresentazione del grafo orientato $G=(N,A)$, $N=\{1,2,3,4,5\}$, $A=\{(1,2), (1,4), (1,5), (2,3), (2,4), (4,2), (4,3), (4,5), (5,4)\}$.

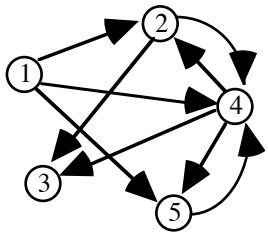

fig. 2: un grafo orientato

Si consideri un arco orientato $a=(i,j)$, il nodo i è chiamato *coda dell'arco a*, mentre il nodo j è detto *testa dell'arco a*.

Un grafo è *semplice* se non ha archi multipli (archi con stessa testa e stessa coda) e anelli (archi con la testa uguale alla coda); in particolare un grafo orientato $G=(N,A)$ è semplice se per ogni coppia di archi di A, $a=(i,j)$ e $a'=(h,k)$, $i \neq h$ oppure $j \neq k$; inoltre $i \neq j$, e $h \neq k$.

Introduciamo ora alcune definizioni relative ai grafi orientati che si possono comunque estendere al caso di grafi non orientati ovunque l'orientamento non sia rilevante.

Si consideri un grafo orientato $G=(N,A)$:

Insieme dei successori:

$$FN(i) = \{j \in N: (i,j) \in A\}.$$

Insieme dei predecessori:

$$PN(j) = \{i \in N: (i,j) \in A\}.$$

esempio: in fig. 2 $FN(1)=\{2,4,5\}$, $FN(2)=\{3,4\}$, $FN(3)=\emptyset$, $FN(4)=\{2,3,5\}$, $FN(5)=\{4\}$; $PN(1)=\emptyset$, $PN(2)=\{1,4\}$, $PN(3)=\{2,4\}$, $PN(4)=\{1,2,5\}$, $PN(5)=\{1,4\}$.
Insieme degli adiacenti:

$$AD(i) = FN(i) \cup PN(i).$$

Stella uscente:

$$FS(i) = \{(i,j) \in A\}.$$

Stella entrante:

$$BS(j) = \{(i,j) \in A\}.$$

esempio: in fig. 2 $FS(1)=\{(1,2),(1,4),(1,5)\}$, $FS(2)=\{(2,3),(2,4)\}$, $FS(3)=\emptyset$, $FS(4)=\{(4,2),(4,3),(4,5)\}$, $FS(5)=\{(5,4)\}$; $BS(1)=\emptyset$, $BS(2)=\{(1,2),(4,2)\}$, $BS(3)=\{(2,3),(4,3)\}$, $BS(4)=\{(1,4),(2,4),(5,4)\}$, $BS(5)=\{(1,5),(4,5)\}$.
Stella:

$$S(i) = FS(i) \cup BS(i).$$

Il *grado uscente* di un nodo i è dato dalla cardinalità della sua stella uscente (o equivalentemente dell'insieme di successori), mentre il *grado entrante* è dato dalla cardinalità della stella entrante (o dell'insieme dei predecessori). Il *grado* è dato dalla somma del grado uscente e del grado entrante.
Il *sottografo di G indotto* da $V \subseteq N$ è il grafo $G' = (V,A')$, dove l'insieme degli archi $A'=\{(i,j) \in A: i, j \in V\}$. In fig. 3a viene illustrato il sottografo di G indotto da $V=\{1,3,4,5\}$. Il *grafo parziale di G definito da* $B \subseteq A$ è il grafo $G' = (N,B)$. In fig. 3b viene illustrato il grafo parziale definito da $B=\{(1,2),(1,5),(2,4),(4,2),(4,5),(5,4)\}$. Le due definizioni possono essere combinate. In fig. 3c viene mostrato il sottografo parziale di G indotto da V definito da B.

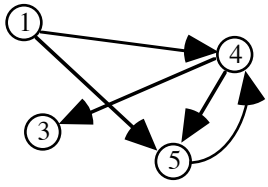

fig. 3a

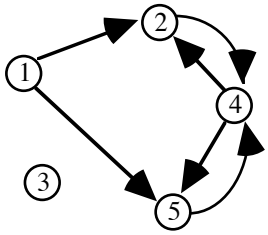

fig. 3b

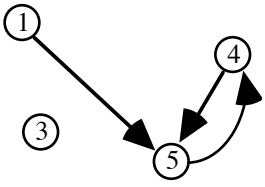

fig. 3c

Definiamo ora grafi con particolari strutture e introduciamo alcuni concetti di basilare importanza.

Catena: Una catena in un grafo orientato $G=(N,A)$ è un sottografo parziale di G definito da $B=\{a_1,a_2,a_{r-1}\}$ rappresentato come una sequenza finita di nodi e archi adiacenti, i_1, a_1, i_2,…, i_{r-1}, a_{r-1}, i_r tale che o $a_k=(i_k,i_{k+1})$ oppure $a_k=(i_{k+1},i_k)$, per $k=1,\ldots,r-1$. Stabilendo un verso di percorrenza da i_1 a i_r, gli archi del tipo (i_k,i_{k+1}) sono detti *concordi con il verso di percorrenza* e gli archi del tipo (i_{k+1},i_k) sono detti *discordi con il verso di percorrenza*. Per semplicità nel seguito indicheremo una catena come una sequenza di archi o di nodi.

Catena chiusa: Si tratta di una catena in cui $i_1=i_r$. Una catena chiusa è detta anche *circuito*.

esempio: per il grafo di fig. 2 una catena è data da (1,2),(4,2),(4,5),(1,5),(1,2). Una catena chiusa è data da (1,2),(4,2),(1,4).

Cammino: Un cammino in un grafo orientato $G=(N,A)$ è un grafo parziale di G definito da $B=\{a_1,a_2,a_{r-1}\}$ rappresentato come una sequenza di nodi e archi adiacenti, i_1, a_1, i_2,…, i_{r-1}, a_{r-1}, i_r tale che $a_k=(i_k,i_{k+1})$ per $k=1,\ldots,r-1$. Per semplicità nel seguito indicheremo un cammino come una sequenza di nodi o come una sequenza di archi. *Cammino chiuso*: Si tratta di un cammino in cui $i_1=i_r$.

esempio: per il grafo di fig. 2 un cammino è dato da 2,4,5,4,3. Un cammino chiuso è dato da 2,4,5,4,2.

Cammino semplice: Un cammino semplice è costituito da un cammino senza archi ripetuti.

Cammino elementare: Un cammino elementare è costituito da un cammino senza nodi ripetuti.

esempio: per il grafo di fig. 2 un cammino semplice è dato da 2,4,5,4,3. Un cammino elementare è dato da 1,2,4,5.

Ciclo: Si tratta di un cammino chiuso in cui l'unico nodo ripetuto è il primo. In altri termini un ciclo è costituito da un cammino elementare da i_1 a i_r e dall'arco (i_r, i_1). Un grafo che non contiene cicli è detto *aciclico*.

esempio: per il grafo di fig. 2 un ciclo è dato da 2,4,2.

Ciclo Hamiltoniano: Un ciclo Hamiltoniano di un grafo orientato $G=(N,A)$ è un ciclo i cui nodi sono tutti quelli in N, quindi contiene esattamente $|N|$ archi. Si noti che non sempre esiste un ciclo Hamiltoniano in un grafo, ad esempio nel grafo di fig. 2 non è possibile individuare alcun ciclo Hamiltoniano.

Ciclo Euleriano: Il ciclo Euleriano di un grafo orientato $G=(N,A)$ è un cammino chiuso semplice con esattamente $|A|$ archi. Come nel caso del ciclo Hamiltoniano, non tutti i grafi ammettono un ciclo Euleriano. Una condizione necessaria e sufficiente per l'esistenza di un ciclo Euleriano è data da: $|FS(i)|=|BS(i)|$, $\forall i \in N$.

Grafo connesso: Un grafo orientato $G=(N,A)$ è connesso se per ogni coppia di nodi i e $j \in N$, esiste una catena da i a j.

Grafo fortemente connesso: Un grafo orientato $G=(N,A)$ è fortemente connesso se per ogni coppia di nodi i e $j \in N$, esiste un cammino da i a j.
esempio: il grafo di fig. 2 è connesso ma non è fortemente connesso: ad esempio, è facile convincersi che non esiste alcun cammino che parte dal nodo 3 (infatti $FS(3)=\emptyset$), inoltre non esiste alcun cammino che arriva al nodo 1 (infatti $BS(1)=\emptyset$).
Componente connessa: Una componente connessa di un grafo orientato $G=(N,A)$ è un sottografo $G'=(N',A')$ connesso massimale, cioè tale che per ogni $i \in N \setminus N'$, non esiste né l'arco (i,j) né l'arco (j,i) con $j \in N'$. Analogamente si definisce una *componente fortemente connessa*.
esempio: in fig. 3c si individuano 2 componenti connesse {3} e {1,4,5}. In fig. 2 ci sono 3 componenti fortemente connesse: {1}, {3} e {2,4,5}.
Taglio: Un taglio di un grafo è definito da una partizione dell'insieme dei nodi N in due sottoinsiemi N' e N'' con $N=N' \cup N''$ e $N' \cap N''=\emptyset$. Gli *archi del taglio* (N',N'') sono definiti dall'insieme di archi con la coda in N' e la testa in N'' (archi concordi con il taglio) e dall'insieme degli archi con la coda in N'' e la testa in N' (archi discordi). Rimuovendo gli archi del taglio, un grafo inizialmente connesso viene ripartito in almeno due componenti connesse.
esempio: un taglio del grafo di fig. 2 è dato da $N'=\{1,4\}$ e $N''=\{2,3,5\}$; gli archi del taglio sono (1,2), (1,5), (2,4), (4,2), (4,3), (4,5), (5,4).
Grafo bipartito: Un grafo è bipartito se l'insieme dei nodi N può essere partizionato in due sottoinsiemi N_1 e N_2 tali che per ogni arco (i,j) di A o $i \in N_1$ e $j \in N_2$ oppure $i \in N_2$ e $j \in N_1$.
Albero: Un albero è un grafo connesso che non contiene catene chiuse.
Albero di copertura di un grafo: Un albero T è di copertura per il grafo G se tutti i nodi di G sono testa o coda di archi di T.
esempio: un albero di copertura per il grafo di fig. 2 è dato da {(1,4), (2,4), (4,3),(5,4)}.
Accoppiamento: Un accoppiamento è un grafo parziale tale che non vi sono due archi incidenti in uno stesso nodo. Se ogni nodo ha un arco incidente si parla di *accoppiamento perfetto*.

Riconsideriamo rapidamente i problemi elencati sopra e cerchiamo di interpretarli in termini di grafi e di individuare quali caratteristiche devono avere le soluzioni.

Nel problema del progetto di rete di telecomunicazione possiamo utilizzare un grafo in cui i nodi rappresentano i centri da collegare e gli archi rappresentano le linee di comunicazione che possono venire attivate. Il grafo non è orientato e una soluzione, dovendo garantire la connessione tra tutti i centri, è data da un albero di copertura, in particolare si cercherà quello di costo minimo. Se però nel progetto si vuole considerare anche la robustezza della rete nel caso di guasti, ci rendiamo conto che un albero ha un difetto: in caso di rottura di una linea di comunicazione la rete risulta non connessa. Un possibile modo di ovviare a questo inconveniente può essere quello di richiedere la

biconnessione della rete, quindi tra ogni coppia di nodi devono esistere due catene disgiunte su nodi e archi. È facile rendersi conto che con questi requisiti quello che si cerca è un anello che connette tutti i nodi, quindi un circuito Hamiltoniano.

Nel caso di assegnamento di esercizi a studenti possiamo ricorrere a un grafo bipartito in cui i nodi sono partizionati in due sottoinsiemi: un sottoinsieme rappresenta gli studenti, e l'altro gli esercizi. Gli archi vanno da nodi del sottoinsieme studenti a quelli del sottoinsieme esercizi e il peso loro associato è quello della difficoltà percepita. Il problema consiste nel trovare un accoppiamento perfetto in cui l'arco di peso massimo è minimo (funzione obiettivo bottleneck).

Nel caso del postino possiamo utilizzare un grafo in cui i nodi rappresentano gli incroci delle strade e gli archi le strade stesse. Per semplicità assumiamo che il postino consegni contemporaneamente le lettere sul lato pari e sul lato dispari della strada, quindi il grafo è non orientato. Se il grafo così ottenuto è Euleriano c'è ben poco da decidere, infatti è possibile percorrere tutti gli archi una volta sola e tornare al punto di partenza. Lasciamo come esercizio l'individuazione del circuito Euleriano. Se invece il grafo non è Euleriano, quindi esistono nodi di grado dispari (e il numero di tali nodi è pari, perché?), è necessario determinare quali strade il postino deve percorrere più volte per riuscire a coprire il servizio, ovviamente minimizzando la lunghezza del percorso. Vedremo come si può risolvere questo problema combinando vari algoritmi su grafi.

Nel problema della determinazione dell'itinerario del Giro d'Italia, ammesso che vengano utilizzati criteri di ottimizzazione, possiamo rappresentare la rete stradale con un grafo: i nodi sono i paesi candidati a essere traguardi volanti o sede di tappa e gli archi le strade che collegano le varie località idonee al passaggio della carovana del Giro. A differenza del problema del postino, in questo caso il grafo è opportuno che sia orientato per tener conto della percorribilità delle strade in salita o in discesa. Dovendo partire e arrivare in uno stesso luogo dobbiamo cercare sul grafo un ciclo, non necessariamente Hamiltoniano, con però dei vincoli aggiuntivi che riguardano la lunghezza delle tappe, i metri di dislivello, etc.. La funzione obiettivo invece deve tener conto dei premi che si totalizzano nel toccare le varie località. Dato che spesso i vincoli aggiuntivi non sono imposizioni tassative e che tenerne conto può risultare complicato, si può pensare di considerarli come obiettivi secondari e includerli in una unica funzione di utilità.

Nel caso del rinnovo dell'automobile potrebbe sembrare strano ricorrere ad un grafo per determinare la soluzione ottima. Possiamo però utilizzare un grafo in cui i nodi rappresentano l'inizio dei vari anni di pianificazione, quando bisogna prendere la decisione riguardante il rinnovo dell'auto. Gli archi rappresentano invece le decisioni: avremo quindi un arco orientato che collega direttamente il

nodo i con il nodo $i+k$ se si decide di acquistare l'auto l'anno i e rivenderla dopo k anni. Questa scelta comporta un costo ben quantificabile: il costo di acquisto nell'anno i, il costo della manutenzione nei k anni successivi che potrebbe crescere con l'invecchiamento dell'auto, a cui sottraiamo il ricavato dalla vendita dell'auto usata dopo k anni. Per risolvere il problema dobbiamo quindi individuare una strategia di rinnovo dell'auto che garantisca di averne sempre esattamente a disposizione una e che minimizzi i costi nell'arco temporale della pianificazione. Vedremo che la soluzione ottima di questo problema corrisponde a un cammino minimo dal nodo che rappresenta l'inizio della pianificazione a quello che rappresenta la fine delle attività del commesso viaggiatore.

2.1 Rappresentazione di grafi

Per rappresentare i grafi, a parte la rappresentazione grafica di nodi ed archi, si possono adottare varie tecniche a seconda dell'utilizzo. Alcune sono più adatte ad un uso di tipo matematico e altre più adatte ad un uso algoritmico.

Matrice di adiacenza

Si utilizza una matrice quadrata AD con una riga/colonna in corrispondenza a ciascun nodo del grafo $G=(N,A)$. Il coefficiente ad_{ij} della matrice è uguale a 1 se e solo se l'arco $(i,j)\in A$. Se il grafo è non orientato la matrice AD è simmetrica, oppure potrebbe essere memorizzata solo nella parte triangolare superiore o inferiore.

Per il grafo orientato di fig. 2 abbiamo la seguente matrice:

0	1	0	1	1
0	0	1	1	0
0	0	0	0	0
0	1	1	0	1
0	0	0	1	0

Si noti che gli elementi non zero di ciascuna riga individuano la corrispondente stella uscente, mentre quelli sulle colonne corrispondono alle stelle entranti.

Matrice di incidenza nodi-archi

Dato un grafo orientato $G=(N,A)$, la matrice di incidenza nodi-archi E ha una riga per ogni nodo e una colonna per ogni arco del grafo e viene costruita nel modo seguente:

$$e_{ia} = \begin{cases} -1 & \text{se } a=(i,j)\in A \\ 1 & \text{se } a=(j,i) \\ 0 & \text{altrimenti} \end{cases}$$

La matrice di incidenza è particolarmente utile nelle formulazioni matematiche.

La matrice di incidenza del grafo di fig. 2 è data da:

	(1,2)	(1,4)	(1,5)	(2,3)	(2,4)	(4,2)	(4,3)	(4,5)	(5,4)
1	-1	-1	-1	0	0	0	0	0	0
2	1	0	0	-1	-1	1	0	0	0
3	0	0	0	1	0	0	1	0	0
4	0	1	0	0	1	-1	-1	-1	1
5	0	0	1	0	0	0	0	1	-1

Liste

Le rappresentazioni matriciali sono poco adatte alle implementazioni efficienti di algoritmi specie se i grafi sono sparsi. In questi casi si ricorre a una rappresentazione di stelle entranti e stelle uscenti tramite liste. Quindi ogni arco è rappresentato da un record con i seguenti campi: puntatore agli attributi dell'arco (costo, capacità, peso, flusso, etc.), puntatore al nodo testa, puntatore al nodo coda, puntatore all'arco successivo nella stella uscente del nodo coda, puntatore al nodo successivo nella stella entrante del nodo testa. Se si prevede di manipolare il grado con aggiunte o cancellazioni di archi, potrebbe essere vantaggioso prevedere delle liste doppie, quindi mettere anche i puntatori al predecessore nella stella entrante e nella stella uscente.

3 Grafi non orientati: albero di copertura e circuito Hamiltoniano

Consideriamo in dettaglio il problema di progetto di una rete di telecomunicazione che, come abbiamo visto prima può venire rappresentato per mezzo di un grafo non orientato pesato $G=(N,A)$, in cui i nodi corrispondono ai centri da collegare e gli archi corrispondono alle linee di comunicazione che è possibile installare. Il peso $w_e > 0$ associato a ciascun arco $e=(i,j)$ è il costo necessario alla installazione della linea tra i centri agli estremi dell'arco. In fig. 4 è riportato un esempio di un grafo con 7 nodi e 12 archi.

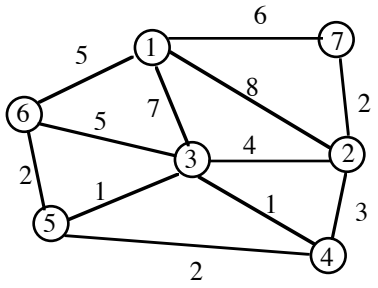

fig. 4: grafo non orientato pesato

3.1 Albero di copertura: formulazione e un algoritmo di soluzione

Ricordiamo che, nel caso dell'albero di copertura, i requisiti di una soluzione richiedono che tra ogni coppia di nodi deve esistere una (unica) catena. Prima di procedere con la descrizione di un possibile algoritmo di soluzione proviamo a formulare il problema in termini di variabili, funzione obiettivo e vincoli. Per arrivare a una rappresentazione di una soluzione T, notiamo che un albero è

composto da archi, pertanto possiamo introdurre delle variabili logiche associate agli archi con il seguente significato:

$$x_e = \begin{cases} 1 & \text{se } e \in T \\ 0 & \text{altrimenti} \end{cases}$$

per ogni arco $e \in A$. La funzione obiettivo è semplice da esprimere:

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

Esprimere matematicamente i vincoli di connessione è assai meno banale di quanto sia possibile fare con le parole. Tali vincoli si basano sulla seguente osservazione: in un albero di copertura, e comunque in qualsiasi sottografo connesso, considerato un sottoinsieme proprio di nodi $S \subset N$, contenente almeno un nodo, deve essere possibile connettersi al resto del grafo con almeno un arco. I vincoli quindi risultano:

$$\sum_{e \in \delta(S)} x_e \geq 1 \qquad \forall S \subset N, |S| \geq 1,$$

dove con $\delta(S)$ indichiamo l'insieme degli archi con un estremo in S e l'altro in $N \setminus S$: $\delta(S) = \{(i,j) \in A : i \in S, j \in N \setminus S\}$. Riassumendo, quindi, la formulazione del problema dell'albero di copertura è:

$$\begin{aligned} \min \sum_{e \in A} & w_e x_e \\ & \sum_{e \in \delta(S)} x_e \geq 1 \qquad \forall S \subset N, |S| \geq 1, \\ & x_e \in \{0,1\} \qquad \forall\, e \in A. \end{aligned}$$

Si noti come la formulazione sopra riportata non contenga un vincolo esplicito riguardante l'unicità della catena tra ogni coppia di nodi. Tale vincolo infatti è implicato dalla funzione obiettivo e dal fatto che i pesi degli archi sono positivi. Una ulteriore osservazione riguarda la dimensione del modello: il numero di variabili è pari al numero di archi, ma il numero di vincoli è pari al numero di possibili modi di scegliere un sottoinsieme di nodi, quindi esponenziale nel numero di nodi. Con gli strumenti visti fino ad ora, tale formulazione risulta pertanto abbastanza inutile. A dispetto di questo però, è possibile descrivere un algoritmo estremamente efficiente per determinare la soluzione ottima del problema.

Esercizio
Il modello per l'albero di copertura descritto sopra non fornisce una soluzione ammissibile in caso di pesi non ristretti a valori positivi. Perché? Come si può completare il modello in modo da renderlo corretto anche in questo caso più generale?

L'algoritmo che consideriamo costruisce una soluzione iterativamente a partire da quella vuota, e aggiunge ad ogni passo un nuovo elemento, quindi un nuovo arco all'insieme di quelli già scelti.

Due criteri devono essere stabiliti per garantire un funzionamento corretto dell'algoritmo: i) l'ordine con il quale si selezionano gli archi da inserire nella soluzione e ii) il criterio con il quale si decide di aggiungere un arco a quelli già presenti.

Per quel che riguarda l'ordine, ci facciamo guidare dal peso degli archi. Inizieremo quindi a considerare gli archi di peso più piccolo per terminare con quelli più pesanti. Un arco viene aggiunto alla soluzione corrente solo se l'insieme degli archi così aumentato può far parte di una soluzione ammissibile. Nel nostro caso questo significa che l'arco aggiunto alla soluzione non deve formare cicli con gli archi già scelti. In caso contrario arriveremmo ad avere una soluzione ridondante. In definitiva l'algoritmo, noto in letteratura come l'algoritmo di Kruskal, è il seguente.

Procedure Kruskal (G=(N,A), T):
  begin
    $T := \emptyset$; $S := A$;
    repeat
      e := argmin(w_a, $a \in S$); $S := S \setminus \{e\}$;
      if $T \cup \{e\}$ non contiene cicli **then** $T := T \cup \{e\}$
    until $S = \emptyset$
  end.

Durante l'esecuzione dell'algoritmo S contiene l'insieme degli archi candidati ad entrare in soluzione e non ancora esaminati; al termine T contiene la soluzione. La funzione **argmin**(w_a, $a \in S$) restituisce l'elemento di peso minimo tra quelli in S. Nella seguente tabella descriviamo l'evoluzione dell'algoritmo applicato al grafo di fig. 4.

e	***S***
(3,4)	{(3,4)}
(3,5)	{(3,4), (3,5)}
(4,5)	{(3,4), (3,5)}
(2,7)	{(3,4), (3,5), (2,7)}
(5,6)	{(3,4), (3,5), (2,7), (5,6)}
(2,4)	{(3,4), (3,5), (2,7), (5,6), (2,4)}
(2,3)	{(3,4), (3,5), (2,7), (5,6), (2,4)}
(1,6)	{(3,4), (3,5), (2,7), (5,6), (2,4), (1,6)}
(3,6)	{(3,4), (3,5), (2,7), (5,6), (2,4), (1,6)}
(1,7)	{(3,4), (3,5), (2,7), (5,6), (2,4), (1,6)}
(1,3)	{(3,4), (3,5), (2,7), (5,6), (2,4), (1,6)}
(1,2)	{(3,4), (3,5), (2,7), (5,6), (2,4), (1,6)}

Tabella 1: evoluzione dell'algoritmo di Kruskal applicato al grafo di fig. 4

Si noti che l'algoritmo potrebbe arrestarsi prima di considerare tutti gli archi. Nel nostro esempio infatti le ultime quattro iterazioni sono inutili in quanto abbiamo già completato l'albero di copertura con l'aggiunta dell'arco (1,6). L'algoritmo può venire velocizzato modificando la condizione di arresto: se infatti abbiamo un grafo G connesso con n nodi, sappiamo che un albero di copertura contiene esattamente n-1 archi. La condizione di arresto diventa quindi:

until $S = \emptyset$ or $|T|=|N|-1$

Teorema

L'algoritmo di Kruskal fornisce la soluzione ottima del problema dell'albero di copertura.

Dim.

La dimostrazione è per assurdo. Sia T^* l'albero di copertura di peso minimo e supponiamo che l'algoritmo di Kruskal fornisca un albero T diverso da T^* e di peso complessivo maggiore di quello ottimo. Questo significa che esiste un arco e del grafo che appartiene a T^* ma non a T. Se aggiungiamo l'arco e all'albero T si viene a formare un ciclo C. Per come funziona l'algoritmo di Kruskal $w_e \geq w_{e'}$ per ogni $e' \in C$. Se ora rimuoviamo l'arco e da T^* otteniamo due componenti connesse, e almeno un arco $e' \in C$, non appartenente a T^*, appartiene al taglio venutosi a formare. Se ora sostituiamo l'arco e con l'arco e' nella soluzione T^* otteniamo un albero di copertura di peso complessivo minore o uguale a quello di T^* contraddicendo l'ipotesi.

♦

Esercizio

Individuare una struttura dati opportuna e degli accorgimenti che permettano di implementare efficientemente l'algoritmo di Kruskal. Particolare attenzione merita il metodo con il quale vengono individuati i cicli.

Esercizio

Si consideri il problema di memorizzare in forma compatta un insieme di record, siano essi fotogrammi di un film o codici genetici. Invece di memorizzare per intero ogni singolo record è possibile registrare solamente le differenze tra un record e l'altro. Formulare il problema in termini di albero di copertura di costo minimo.

3.2 Circuito Hamiltoniano di costo minimo

Il problema del circuito Hamiltoniano di costo minimo è uno dei problemi di ottimizzazione combinatoria più studiati e non a caso ha molteplici applicazioni. In letteratura è noto anche come problema del commesso viaggiatore (Traveling Salesperson Problem - TSP). Il problema è specificato da un grafo non orientato $G=(N,A)$ pesato come nel caso dell'albero di copertura. Spesso si fa anche l'ipotesi che il grafo sia *completo*, cioè che tra ogni coppia di nodi esista l'arco che li connette. Ci si può sempre ricondurre a questo caso aggiungendo al grafo gli archi mancanti e attribuendo loro un peso molto elevato. Una semplice applicazione è quella alla quale abbiamo accennato nel contesto del progetto di una rete di telecomunicazione: volendo rendere più robusta rispetto ai guasti una soluzione, imponiamo che esistano sempre due catene disgiunte tra ogni coppia di nodi. La soluzione più semplice con questa caratteristica è un circuito Hamiltoniano. Si noti come, rispetto all'albero di copertura, il circuito Hamiltoniano ha un solo arco in più. Una possibile formulazione prende spunto proprio dalla formulazione del problema dell'albero di copertura. Pertanto detta C una soluzione ammissibile (circuito Hamiltoniano) introduciamo le seguenti variabili:

$$x_e = \begin{cases} 1 & \text{se } e \in C \\ 0 & \text{altrimenti} \end{cases}$$

Notiamo che in un circuito Hamiltoniano vi sono esattamente due archi incidenti in ciascun nodo. Quindi un primo vincolo potrebbe essere:

$$\sum_{e \in S(i)} x_e = 2 \qquad \forall\, i \in N.$$

Questi vincoli non sono sufficienti dato che, oltre ai circuiti Hamiltoniani, li soddisfano anche tutte le soluzioni che toccano tutti i nodi del grafo con esattamente un ciclo, non necessariamente Hamiltoniano (dette anche *coperture per cicli*). Una soluzione di questo tipo nel grafo di fig. 4 per esempio è: C_1={(1,2),(2,7),(1,7)}, C_2={(3,6),(3,4),(4,5),(5,6)}. È necessario quindi aggiungere dei vincoli per restringere l'insieme delle soluzioni ammissibili alle coperture per cicli formate da un unico ciclo. Questo si ottiene facendo ricorso ai vincoli visti per l'albero di copertura che imponevano la connessione dell'insieme di archi ottenuti.

$$\sum_{e \in \delta(S)} x_e \geq 1 \qquad \forall S \subset N,\ |S| \geq 1,$$

Una possibile formulazione del problema del commesso viaggiatore è quindi:

$$\begin{aligned} \min \sum_{e \in A} & w_e x_e \\ & \sum_{e \in S(i)} x_e = 2 && \forall\, i \in N \\ & \sum_{e \in \delta(S)} x_e \geq 1 && \forall S \subset N,\ |S| \geq 1, \\ & x_e \in \{0,1\} && \forall\, e \in A. \end{aligned}$$

Una formulazione alternativa del problema considera un altro modo di evitare cicli che non contengono tutti i nodi. Possiamo imporre un vincolo che, considerato un qualsiasi sottoinsieme proprio di N contenente almeno due nodi, eviti l'esistenza di un *sottocircuit*. Questo significa che il numero degli archi che connettono i nodi del sottoinsieme deve essere minore o uguale alla cardinalità del sottoinsieme meno 1. La formulazione quindi è:

$$\begin{aligned} \min \sum_{e \in A} & w_e x_e \\ & \sum_{e \in S(i)} x_e = 2 && \forall\, i \in N \\ & \sum_{e=(i,j):\, i,j \in S} x_e \leq |S| - 1 && \forall S \subset N,\ |S| \geq 2, \\ & x_e \in \{0,1\} && \forall\, e \in A. \end{aligned}$$

Come nel caso del problema dell'albero di copertura le formulazioni che abbiamo descritto contengono un numero esponenziale di vincoli. Purtroppo, al contrario del caso del problema precedente, il problema del commesso viaggiatore non è altrettanto facile da risolvere, come vedremo più approfonditamente nel seguito.

4 Connessione di un grafo: algoritmo di visita

Si consideri un grafo orientato $G=(N,A)$ con s e $t \in N$ e ci si ponga il problema di dire se esiste un cammino che collega s con t. Ovviamente, se un tale cammino esiste, è facile fornire una "giustificazione sintetica" (*certificato*) dell'esistenza: infatti basta descrivere il cammino elementare che collega s e t. Al contrario, se tale cammino non esiste, non è possibile giustificare la risposta in modo sintetico rimanendo nello spazio dei cammini, infatti si potrebbero enumerare tutti i cammini uscenti da s, ma il loro numero potrebbe risultare esponenziale. Il seguente teorema ci viene in aiuto e mette in evidenza la relazione che c'è tra lo spazio dei cammini e lo spazio dei tagli di un grafo.

Proprietà: cammino o taglio

Sia dato un grafo orientato $G=(N,A)$ con s e $t \in N$. Una sola delle seguenti affermazioni vale:

i) esiste un cammino orientato da s a t;

ii) esiste un taglio (N_s, N_t) tale che $s \in N_s$ e $t \in N_t$, e non esistono archi concordi al taglio, ovvero tutti gli archi $(i,j) \in A$ che attraversano il taglio hanno $i \in N_t$ e $j \in N_s$.

Dim.

La dimostrazione è di tipo costruttivo e si basa sulla possibilità di visitare il grafo a partire da s. Procediamo a visitare il grafo a partire da s percorrendo gli archi in base al loro verso. Ci si presentano due casi mutuamente esclusivi:

i) durante la ricerca di sequenze di archi percorribili, si riesce a raggiungere il nodo t ricadendo nel caso i) dell'enunciato: il cammino da s a t è ricostruibile risalendo i predecessori.

ii) la visita del grafo a partire da s si arresta senza aver definito il predecessore del nodo t. Chiamiamo N_t tutti i nodi che hanno il predecessore nullo, mentre sia N_s l'insieme di tutti i nodi con predecessore diverso da zero. Ovviamente $N=N_s \cup N_t$, e $N_s \cap N_t=\emptyset$. Per come funziona l'algoritmo non possono esistere archi (i,j) con $i \in N_s$ e $j \in N_t$, altrimenti j apparterebbe a N_s. ♦

Un algoritmo semplice ma di fondamentale importanza è quello che effettua la visita del grafo orientato G a partire da un nodo radice s. L'algoritmo restituisce un vettore di predecessori. Il predecessore $P[i]$ di un nodo i fornisce il nodo immediatamente precedente a i nel cammino orientato da s a i. Se al termine dell'algoritmo un nodo ha predecessore uguale a zero, significa che il nodo non è raggiungibile da r.

```
Procedure Visita (G=(N,A),s,P):
    begin
      for i := 1 to n do P[i] :=0; P[s] := s; Q := {s};   {inizializzazione}
      repeat
        select i from Q; Q:=Q\{i};       {selezione e rimozione di un nodo da Q}
        for each (i,j) ∈ FS(i)  do
          if P[j] = 0 then begin P[j] := i;  Q := Q ∪ {j} end;
      until Q = ø
    end.
```

Analizziamo la complessità dell'algoritmo: denotando con n il numero di nodi del grafo e con m il numero degli archi, notiamo che la procedura di visita esamina ogni nodo al più una volta, e per ogni nodo viene esaminata la stella uscente che può contenere al più n archi. Da questa prima analisi risulta che l'algoritmo effettua al più n selezioni dall'insieme Q e al più $O(n^2)$ controlli e aggiornamenti del predecessore. Se però andiamo più nel dettaglio e valutiamo complessivamente quanto viene svolto dal ciclo, notiamo che ogni arco del grafo viene esaminato al più una volta. Infatti un arco appartiene ad una sola stella uscente. Questo significa che l'algoritmo effettua al più n selezioni da Q e al più m controlli e aggiornamenti dei predecessori. Assumendo una implementazione dell'insieme Q con una lista non ordinata o altra struttura di semplice gestione, la complessità dell'algoritmo di visita è $O(m)$.

5 Cammini minimi

Sia dato un grafo orientato pesato $G=(N,A)$ con pesi c_{ij} associati agli archi $(i,j)\in A$. I pesi c_{ij} vengono anche chiamati lunghezze degli archi. La lunghezza di un cammino è data dalla somma delle lunghezze degli archi che lo compongono. Poniamoci ora il problema di cercare il cammino più breve da un nodo s a un nodo t. Assumiamo che nel grafo non siano presenti cicli di lunghezza negativa. Infatti l'esistenza di un ciclo negativo renderebbe vantaggioso scegliere un cammino che contiene il ciclo e lo percorre un numero infinito di volte, producendo una sequenza infinita di archi e di lunghezza complessiva illimitata inferiormente. Per evitare questo inconveniente bisognerebbe richiedere che il cammino cercato fosse elementare cioè che non passasse più volte per uno stesso nodo, cosa tutt'altro che semplice. Vedremo più avanti come fare ad individuare i cicli di lunghezza negativa in un grafo. Per formulare il problema utilizziamo delle variabili logiche legate agli archi, componenti elementari di un cammino p:

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

I vincoli del problema sono:

i) il cammino deve partire dal nodo s:

$$\sum_{(s,j)\in FS(s)} x_{sj} = 1$$

ii) il cammino deve arrivare nel nodo t:

$$\sum_{(j,t)\in BS(t)} x_{jt} = 1$$

ii) se il cammino passa per un nodo intermedio i, diverso da s e t, non può fermarsi:

$$\sum_{(j,i)\in BS(i)} x_{ji} - \sum_{(i,j)\in FS(i)} x_{ij} = 0, \ \forall i \neq s, t.$$

Si noti come quest'ultima classe di vincoli vale per tutti i nodi diversi da s e da t. Infatti, se un nodo i non fa parte del cammino, la somma degli archi entranti è uguale a zero e deve essere uguale alla somma degli archi uscenti. Facendo l'ipotesi di assenza di cicli negativi, non occorre restringere le variabili x_{ij} ad assumere valori 0 o 1, ma basta imporre la non negatività; la formulazione del problema può venire riassunta come segue:

$$\begin{aligned} \min \quad & \sum_{(i,j)\in A} c_{ij}\, x_{ij} \\ & \sum_{(j,i)\in BS(i)} x_{ji} - \sum_{(i,j)\in FS(i)} x_{ij} = b_i \qquad \forall i \in N \\ & x_{ij} \in \{0,1\} \qquad\qquad\qquad\qquad \forall\, (i,j) \in A, \end{aligned}$$

dove $b_i = -1$ se $i=s$, $b_i = 1$ se $i=t$, e $b_i = 0$ in tutti gli altri casi. Notiamo che in questa formulazione abbiamo un vincolo per ogni nodo e una variabile per ogni arco.

5.1 Il problema dello "spago"

Il problema della ricerca di un cammino minimo da un nodo s a un nodo t può venire rappresentato e risolto anche facendo uso di uno spago non elastico. Per ogni arco (i,j) del grafo si taglia un pezzo di spago di lunghezza c_{ij}. I pezzi di spago vengono annodati secondo la definizione delle stelle dei nodi del grafo, in modo che i "nodi" del groviglio corrispondano ai nodi del grafo. Per determinare il cammino minimo da un nodo s a un nodo t è sufficiente prendere i "nodi" corrispondenti nel groviglio e cercare di allontanarli tra loro quanto più possibile senza spezzare i fili. I pezzi di spago tesi corrispondono agli archi nel cammino minimo. Il problema può essere descritto anche matematicamente, usando delle variabili π_i associate ai nodi (dette variabili potenziale) che ne rappresentano la posizione, o altezza da terra:

$$\begin{aligned} \max \quad & \pi_t - \pi_s \\ & -\pi_i + \pi_j \le c_{ij} \qquad\qquad \forall\, (i,j) \in A. \end{aligned}$$

Il problema quindi equivale a massimizzare la differenza di potenziale tra il nodo s e il nodo t in modo che la differenza di potenziale tra gli estremi di ogni arco non superino la lunghezza dell'arco stesso (che equivale a imporre che i fili non si spezzino nel groviglio).

Si noti che se un arco (i,j) appartiene al cammino minimo, la differenza di potenziale tra i e j deve essere uguale a c_{ij}. Inoltre, se attribuiamo potenziale 0 al nodo s, il potenziale ottimo dei singoli nodi appartenenti al cammino da s a t, sarà dato dalla lunghezza del cammino minimo che li collega con s.

Esercizio

Per il grafo in fig. 5 formulare con il linguaggio di modellazione visto nel capitolo 1 il problema della ricerca del cammino minimo da 1 a 7 e determinare la soluzione ottima. Formulare anche il corrispondente problema dello spago e verificare che la sua soluzione ottima è equivalente a quella trovata con la prima formulazione. Osservare le relazioni tra gli archi scelti dalla soluzione e la differenza tra i potenziali dei nodi estremi degli archi. Si osservi che per i due problemi il file di input è il medesimo.

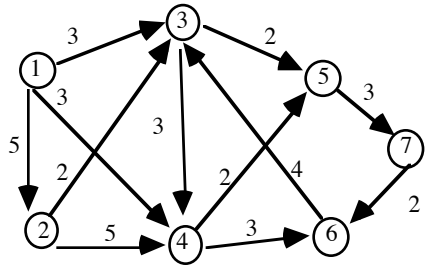

fig. 5: problema del cammino minimo da 1 a 7

5.2 Applicazione: pianificazione di acquisti

Una impresa di costruzioni usa travi di acciaio con stessa sezione, ma lunghezze diverse $L_1<L_2<\ldots<L_n$. Il fabbisogno per ogni tipo di trave i è dato da D_i. Una trave di lunghezza L_i se non disponibile può essere ottenuta tagliando una trave di lunghezza maggiore (gli scarti non vengono riutilizzati). Il costo per la predisposizione del magazzino ad ospitare travi di lunghezza L_i è dato da K_i, ed è indipendente dal numero delle travi; il costo per l'acquisto di una trave di lunghezza L_i è di C_i. Il problema di acquistare le travi necessarie e predisporre il magazzino per ospitarle in modo da minimizzare la spesa complessiva può essere formulato come un problema di ricerca di un cammino di lunghezza minima su un opportuno grafo $G=(N,A)$. L'insieme dei nodi N è dato da $\{0,1,\ldots,n\}$, mentre l'insieme degli archi $A=\{(i,j): i<j, i,j\in N\}$. Il generico arco $(i,j)\in A$ rappresenta la scelta di acquistare e immagazzinare travi di lunghezza L_j per ottenere travi di lunghezza $L_{i+1},\ldots, L_j$. La lunghezza dell'arco (i,j) pertanto è data da $c_{ij}= K_j + C_j(\sum_{h=i+1}^{j} D_h)$. Un cammino dal nodo 0 al nodo n rappresenta una soluzione, quindi il cammino minimo fornisce la soluzione ottima.

5.3 Applicazione: acquisto di una automobile

A un giovane in cerca di occupazione viene offerto un contratto di 5 anni come agente di commercio. Tale incarico prevede di essere dotato di auto, quindi prima di essere assunto il giovane deve acquistare un'auto e deve mantenerla in buona efficienza o eventualmente sostituirla durante la durata del contratto. Il modello al quale il giovane è interessato costa 12.000 €. Il valore dell'auto sul mercato dell'usato se rivenduta è di 7.000 € dopo un anno, 6.000 € dopo 2 anni, 2.000 dopo 3,

1.000 dopo 4 e 0 dopo 5. Il costo di un anno di manutenzione è di 2.000 € se l'auto è nuova, 4.000 se ha un anno di anzianità, 5.000 se ne ha 2, 9.000 se ne ha 3 e 11.000 se ne ha 4. Il giovane deve stabilire una strategia di acquisto dell'auto in modo da minimizzare la spesa complessiva (acquisto più manutenzione). Il problema può venire formulato in termini di ricerca di un cammino su un grafo. Il grafo ha 6 nodi, ogni nodo i rappresenta l'inizio dell'anno i-esimo. Tra ogni coppia di nodi i e j (con $i < j$) esiste un arco che rappresenta l'acquisto dell'auto all'inizio dell'anno i e la sua vendita al termine dell'anno j-1. La lunghezza dell'arco è fissata uguale al costo dell'operazione. Qualsiasi cammino da 1 a 6 rappresenta una strategia ammissibile di acquisto e manutenzione. Il cammino di lunghezza minima corrisponde alla strategia di spesa minima. Si noti come anche in questo caso il grafo non contenga cicli.

5.4 Albero dei cammini minimi

Con una piccola modifica al modello per la ricerca del cammino minimo tra due nodi, si può formulare il problema della ricerca dell'insieme dei cammini minimi da un nodo r (*radice*) a tutti gli altri nodi del grafo. Questo insieme di cammini minimi viene chiamato *albero dei cammini minimi di radice r*. È detto albero perché l'insieme dei cammini minimi non contiene cicli. Infatti si noti che se il cammino minimo tra r e un nodo s passa per q, il sotto cammino da r a q è minimo (altrimenti nemmeno il cammino tra r e s sarebbe minimo). La formulazione matematica si ottiene sovrapponendo e sommando tra loro i vincoli dei problemi della ricerca del cammino minimo da r a ogni altro nodo, quindi risulta:

$$\begin{aligned} \min \quad & \sum_{(i,j)\in A} c_{ij}\, x_{ij} \\ & \sum_{(j,i)\in BS(i)} x_{ji} - \sum_{(i,j)\in FS(i)} x_{ij} = b_i \qquad \forall i \in N \\ & x_{ij} \geq 0 \qquad\qquad\qquad\qquad \forall\, (i,j)\in A, \end{aligned}$$

dove $b_i = -(n\text{-}1)$ se $i=r$, $b_i = 1$ in tutti gli altri casi.

Anche nel caso dell'albero dei cammini minimi possiamo formulare un corrispondente problema dello "spago", con variabili associate ai nodi e vincoli associati agli archi del grafo.

$$\begin{aligned} \max \quad & \sum_{i\in N\setminus\{r\}} \pi_i - (n\text{-}1)\,\pi_r \\ & -\pi_i + \pi_j \leq c_{ij} \qquad \forall\, (i,j)\in A. \end{aligned}$$

Siccome valgono le stesse proprietà viste nel caso del cammino minimo da s a t, anche in questo caso i potenziali π ottimi, avendo fissato π_r=0, corrispondono alle lunghezze dei cammini minimi dalla radice r a ogni altro nodo.

Esercizio

Formulare e risolvere il problema dell'albero dei cammini minimi di radice 1 per il grafo di fig. 5 con entrambe le formulazioni viste.

5.5 Algoritmo per l'albero dei cammini minimi su grafi aciclici

Sia dato un grafo $G=(N,A)$ orientato, aciclico e pesato. Si noti che entrambi gli esempi di applicazione visti in precedenza danno luogo a un grafo di questo tipo. Per verificare se un grafo è aciclico basta vedere se è possibile rinumerare i nodi del grafo in modo tale che se $(i,j)\in A$ allora $i<j$. Quando i nodi soddisfano questa proprietà si dice che il grafo è *ben numerato*. Per rinumerare un grafo aciclico secondo una buona numerazione possiamo applicare il seguente semplice algoritmo ricorsivo che restituisce nel vettore φ la nuova numerazione dei nodi:

```
Procedure Buona_numerazione(N,A,φ,x)
    begin
        if ∃ i∈N: BS(i)=ø then φ(i)=x; Buona_numerazione(N\{i}, A\FS(i),x+1)
        else if N≠ø then return("Grafo non aciclico")
    end.
```

L'algoritmo attribuisce il primo numero disponibile ad un nodo che non ha archi entranti, se esiste. In seguito passa a rinumerare il resto del grafo a partire dal numero successivo. Si noti come l'algoritmo può venire utilizzato anche per verificare se un grafo è aciclico.

Esercizio
Valutare la complessità dell'algoritmo per la buona numerazione.

D'ora in avanti assumiamo che G sia ben numerato. Volendo trovare l'albero dei cammini minimi di radice 1 possiamo procedere per induzione. Associamo ad ogni nodo i una etichetta $d[i]$ che fornisce la lunghezza del cammino minimo da 1 a i. Un possibile algoritmo procede secondo la seguente regola induttiva:

$$d[1] = 0;$$
$$d[k] = \min \{d[i] + c_{ik},\ i=1,\ldots,k\text{-}1\},\ k=2,\ldots,n.$$

In pratica ad ogni iterazione andiamo a visitare la stella entrante di ogni nodo. Considerando i nodi in ordine crescente abbiamo la garanzia che le etichette dei nodi che precedono il nodo corrente sono già state determinate.

Esercizio
Formalizzare l'algoritmo basato sulla regola induttiva di cui sopra e valutarne la complessità.

Un altro metodo equivalente visita le stelle uscenti di ogni nodo. Le etichette dei nodi, in questo caso, vengono aggiornate man mano che si trova un percorso migliore. All'inizio, ad esclusione del nodo 1 la cui etichetta viene fissata definitivamente al valore 0, le etichette vengono poste uguali ad un valore molto elevato, tale per cui un qualsiasi cammino da 1 al nodo in questione risulti più corto di tale valore. L'algoritmo è il seguente:

```
Procedure SPT_Acyclic (P,d)
   begin
      P[1]:=1; d[1]:=0; for i:=2 to n do begin P[i]:=1; d[i]:=M end;        {inizializzazione}
      for i:=1 to n-1 do
        for each (i,j)∈FS(i) do
          if d[i]+c_ij < d[j] then begin P[j]:=i; d[j]:=d[i]+c_ij end
   end.
```

Oltre all'etichetta l'algoritmo tiene traccia anche di come sono fatti i cammini. Questo viene fatto per mezzo del predecessore associato a ciascun nodo, come abbiamo visto nel caso dell'algoritmo di visita di un grafo. Gli archi del grafo vengono esaminati una sola volta, e per ciascun arco il numero di operazioni effettuate è costante. Questo significa che, come nel caso della visita, la complessità dell'algoritmo è $O(m)$, dove m è la cardinalità dell'insieme degli archi.

5.6 Grafi aciclici e programmazione dinamica

La determinazione di cammini minimi o massimi su un grafo aciclico sta alla base della soluzione di problemi di ottimizzazione con la tecnica della *programmazione dinamica*. Questa tecnica decompone la determinazione di una soluzione ammissibile in una sequenza di decisioni. Inoltre il costo della soluzione viene calcolato come la somma dei costi delle singole decisioni. Si costruisce quindi un cosiddetto *grafo degli stati* in cui i nodi rappresentano un riassunto delle decisioni prese negli stati precedenti. Gli archi invece rappresentano le possibili transizioni da uno stato all'altro. Per una trattazione più approfondita della programmazione dinamica e delle sue applicazioni si veda il libro di Bertsekas[1].

Vediamo un esempio di applicazione della programmazione dinamica alla soluzione di un problema di zaino la cui generalizzazione può essere utilizzata per risolvere altri problemi trattati nel corso. Si consideri una istanza del problema dello zaino:

$$\max \sum_{i=1}^{n} c_i x_i$$
$$\sum_{i=1}^{n} a_i x_i \le b$$
$$x_i \in \{0,1\},\ i=1,\ldots,n.$$

Per semplicità, facciamo l'ipotesi che la capacità b dello zaino e tutti i coefficienti a_i siano interi non negativi. Le decisioni in cui può venire decomposta una soluzione riguardano i singoli oggetti che possono venire introdotti o meno nello zaino e che consideriamo secondo la sequenza dettata dalla loro numerazione. I nodi del grafo degli stati sono caratterizzati da una coppia di informazioni (i,K) dove i è l'indice dell'ultimo oggetto considerato dalla sequenza delle decisioni mentre K è il peso

---

[1] D. Bertsekas, *Dynamic Programming and Optimal Control*, Athena Scientifica, Belmont MA, 1995.

complessivo presente nello zaino. È ben evidente che volendo considerare unicamente gli stati ammissibili avremo solo i nodi (i,K) con $i=1,\ldots, n$ e $K=0,\ldots, b$. Aggiungiamo inoltre gli stati (0,0) e $(n+1,b)$ che rappresentano l'inizio e la fine del nostro processo decisionale. Gli archi corrispondenti alle transizioni ammissibili sono di due tipi: dal nodo $(i-1,K)$ al nodo (i,K), corrispondente alla decisione di non scegliere l'oggetto i, e dal nodo $(i-1,K)$ al nodo $(i,K+a_i)$, con $K+a_i \leq b$, corrispondente alla decisione di introdurre l'oggetto i nello zaino. Alle transizioni del primo tipo possiamo attribuire un valore 0 mentre a quelle del secondo tipo attribuiamo valore c_i. Al grafo degli stati aggiungiamo anche le transizioni dai nodi (n,K) al nodo $(n+1,b)$ con valore 0. Il grafo costruito è evidentemente aciclico. Notiamo che ogni nodo rappresenta un possibile stato dello zaino, ma a uno stato possono essere associate più soluzioni parziali: è solo andando a determinare uno dei percorsi possibili dal nodo (0,0) al nodo in questione che troviamo una soluzione. Qualsiasi percorso nel grafo degli stati che collega il nodo iniziale (0,0) con il nodo finale $(n+1,b)$ corrisponde a una soluzione ammissibile. Tra tutte le soluzioni vogliamo determinare quella di valore massimo che ovviamente corrisponde al cammino di lunghezza massima. La determinazione del cammino massimo su un grafo aciclico può venire fatta adattando l'algoritmo SPT_Acyclic.

Con la programmazione dinamica possono venire affrontati efficientemente molti problemi. Ora possiamo notare che nei due esempi della pianificazione degli acquisti e dell'acquisto di una automobile i grafi che fanno da supporto al modello non sono che dei casi particolari di grafo degli stati.

Esercizio
Modificare l'algoritmo SPT_Acyclic in modo da determinare l'albero dei cammini massimi. Applicarlo all'esempio dello zaino visto nel capitolo precedente.

5.7 Pianificazione di progetti e metodo del cammino critico

Il calcolo di cammini su grafo ci viene in aiuto anche nella pianificazione delle attività di un progetto. Un progetto complesso può venire decomposto in varie attività, ognuna caratterizzata da un tempo necessario al suo espletamento. Le attività sono legate tra di loro da vincoli di precedenza e tali vincoli sono quelli che di fatto regolano lo svolgimento del progetto. Analizziamo un piccolo esempio che riguarda il montaggio di una bicicletta. Le attività sono riassunte in Tabella 2, in cui vengono riportati anche i tempi di esecuzione in minuti.

Attività	Descrizione	Tempo
A	Assemblaggio di mozzi, raggi, cerchi e centratura ruote	120
B	Sterzo e forcella	60
C	Manubrio e sellino	15
D	Freni	45
E	Cambio e deragliatore	30
F	Movimento centrale, pedivelle e pedali	60
G	Parafanghi e ruote	40
H	Montaggio catena e registrazione cambio	30
I	Montaggio di comandi e cavi	40
J	Fanali	20

Tabella 2: attività del progetto e tempi in minuti

Disponendo di sufficiente personale, le attività possono venire svolte in parallelo purché siano rispettati i vincoli di precedenza. Nel nostro esempio i vincoli di precedenza sono i seguenti: (A→G), (B→C, D), (B, G→I, J), (E, I, G→H), (F→E), (D→I). Il problema della ottimizzazione della pianificazione del progetto consiste nello stabilire un ordinamento delle varie attività e stabilirne una collocazione temporale con l'obiettivo di minimizzare il tempo che intercorre tra l'inizio della prima attività e la fine dell'ultima. Per semplificare e uniformare la formulazione si introducono due attività fittizie: l'attività "inizio" che precede tutte le attività che possono essere svolte per prime, quindi che non compaiono mai a destra nei vincoli di precedenza, e l'attività "fine" che segue tutte le attività che non compaiono mai a sinistra nei vincoli di precedenza. Nel caso del nostro esempio avremo queste nuove precedenze: ("inizio"→A, B, F) e (C,H, J→"fine").
Per formulare il problema introduciamo delle variabili legate alle attività: per ogni attività *i* la variabile π_i fornisce l'istante di inizio della attività. La funzione obiettivo è la seguente:

$$\min \pi_{fine} - \pi_{inizio}.$$

Nel fissare i tempi di inizio delle varie attività dobbiamo solamente guardare ai vincoli di precedenza e fare in modo che se una attività *i* precede un'altra attività *j* il tempo di fine di *i* sia minore o al più uguale al tempo di inizio di *j*. Denotando con p_i il tempo di esecuzione della attività *i*, i vincoli quindi sono:

$$\pi_i + p_i \le \pi_j \qquad \text{per ogni precedenza } (i \rightarrow j).$$

Se riscriviamo i vincoli nel seguente modo

$$-\pi_i + \pi_j \ge p_i$$

possiamo osservare le analogie con i vincoli e la formulazione del problema dello "spago". Possiamo quindi provare a costruire un grafo *G* in cui i nodi corrispondono alle attività e gli archi ai vincoli di precedenza tra coppie di attività. Le lunghezze degli archi possono venir poste uguali alla durata dell'attività rappresentata dal nodo coda dell'arco. Secondo la formulazione data sopra, in questo grafo il nostro scopo è quello di tentare di avvicinare il più possibile il nodo "fine" al nodo

"inizio", contrariamente a quanto facevamo nel caso del problema dello "spago". Anche i vincoli sono rovesciati, infatti, nel problema della pianificazione, nell'avvicinare i nodi dobbiamo rispettare le distanze minime date dalla lunghezza degli archi (durata delle attività).
Prendendo spunto da queste considerazioni possiamo cercare di capire se anche la soluzione del problema di pianificazione del progetto può trovare una interpretazione in termini di cammino sul grafo. È abbastanza facile convincersi che la soluzione ottima del problema di pianificazione corrisponde al cammino di lunghezza massima dal nodo "inizio" al nodo "fine" nel grafo G. Tale cammino è detto *cammino critico* poiché un ritardo di una qualsiasi attività appartenente al cammino critico porterebbe a un identico ritardo di tutto il progetto, cosa che non è necessariamente vero per altre attività non appartenenti a tale cammino.

Esercizio
Formulare il problema della ricerca del cammino massimo da "inizio" a "fine" e osservare differenze e analogie con il problema del cammino minimo.

Si osservi che il grafo G risulta essere aciclico, infatti se esistesse un ciclo significherebbe che i vincoli di precedenza sono ciclici, cosa che renderebbe impossibile l'esecuzione del progetto. Questo implica che possiamo calcolare molto efficientemente il cammino massimo, adattando l'algoritmo SPT_Acyclic al caso massimo, e determinare la soluzione del problema di pianificazione del progetto.

Esercizio
Costruire il grafo relativo al progetto del montaggio di una bicicletta e calcolare il cammino critico.

Esercizio
Dato il cammino critico, come ricostruire i tempi di inizio delle attività? Per le attività non appartenenti al cammino critico il tempo di inizio può assumere valori diversi: come calcolare l'intervallo di ammissibilità di tali valori? Applicarlo al caso del montaggio della bicicletta.

5.8 Algoritmo di Dijkstra

Osserviamo alcune proprietà dell'algoritmo SPT_Acyclic discusso in precedenza per la ricerca di un cammino su grafo aciclico. Durante l'esecuzione dell'algoritmo abbiamo che:

i) il valore $d[i]$ è maggiore o uguale alla lunghezza del cammino minimo da r a i;
ii) a ogni iterazione viene trovato il valore ottimo $d[i]$ (definitivo) per almeno un nodo;
iii) quando viene esaminata la $FS(i)$ il valore $d[i]$ è ottimo.

In particolare quest'ultima proprietà è quella che permette all'algoritmo SPT_Acyclic di essere estremamente efficiente.
Presentiamo ora un algoritmo che sotto opportune ipotesi si comporta come SPT_Acyclic anche su grafi non aciclici. L'algoritmo utilizza un insieme Q dove vengono introdotti i nodi la cui stella uscente deve venire esplorata. Inizialmente Q contiene solo il nodo radice. Ogni volta che viene aggiornata l'etichetta di un nodo, questo viene introdotto in Q, se non già presente. L'istruzione chiave dell'algoritmo è l'estrazione da Q del nodo dal quale continuare la visita del grafo. Nel caso

dell'algoritmo SPT_Acyclic i nodi venivano esaminati secondo l'ordinamento topologico dato dalla buona numerazione. Per poter avere la garanzia di un comportamento analogo, in particolare essere certi che un nodo esaminato non verrà mai più introdotto in Q, l'algoritmo esamina di volta in volta il nodo i con etichetta $d[i]$ minima.

```
Procedure SPT_Dijkstra (r,P,d)
   begin
      for i:=1 to n do begin P[i]:=r; d[i]:=M end; d[r]:=0; Q:={r};          {inizializzazione}
      repeat
        select i from Q such that d[i]=min{d[h]: h∈Q};          {estratto il nodo di etichetta minima}
        Q:=Q\{i};
        for each (i,j)∈FS(i) do
          if d[i]+cij < d[j] then
          begin P[j]:=i; d[j]:=d[i]+cij; if j∉Q then Q:=Q∪{j} end
      until Q=∅
   end.
```

Facendo l'ipotesi che le lunghezze degli archi siano tutte maggiori o uguali a zero ($c_{ij} \geq 0$), quando un nodo viene estratto da Q non ha più modo di venirvi introdotto poiché non è possibile riuscire a trovare un cammino più breve che lo colleghi alla radice. Questo fa sì che valgano le proprietà i), ii) e iii) viste per l'algoritmo SPT_Acyclic.

Cerchiamo di analizzare ora la complessità dell'algoritmo, indicando con n il numero dei nodi e con m il numero di archi del grafo, come di consueto. Nella fase di inizializzazione vengono effettuate $O(n)$ operazioni elementari; per la proprietà vista prima verranno eseguite al più n selezioni dall'insieme Q, e al più m controlli sugli archi, aggiornamenti delle etichette e inserzioni in Q. Quindi tutto dipende dalle strutture dati utilizzate per implementare l'insieme Q. Se viene utilizzata una lista non ordinata, la selezione del nodo di etichetta minima comporta una scansione dell'intera lista, quindi la sua complessità è $O(n)$, mentre l'inserzione viene fatta in $O(1)$. Pertanto la complessità globale è $O(n+n^2+m) = O(n^2)$.

Se invece utilizziamo una coda di priorità come per esempio uno *heap binario*, la complessità della selezione è $O(1)$, mentre la rimozione, inserzione e aggiornamento sono $O(\log n)$. Complessivamente avremo $O(n+n\log n+m\log n) = O(m\log n)$.

L'implementazione con coda di priorità conviene quando $m\log n < n^2$, cioè quando il numero di archi è minore di $n^2/\log n$.

Esempio di esecuzione dell'algoritmo di Dijkstra

Consideriamo il seguente grafo orientato e pesato e applichiamo l'algoritmo SPT_Dijkstra a partire dal nodo radice 1. Prima di iniziare l'esecuzione osserviamo che il grafo non è aciclico (a parte il nodo 1, in ogni nodo arrivano e partono archi, quindi non è possibile dare una buona numerazione). Inoltre tutte le lunghezze degli archi sono positive.

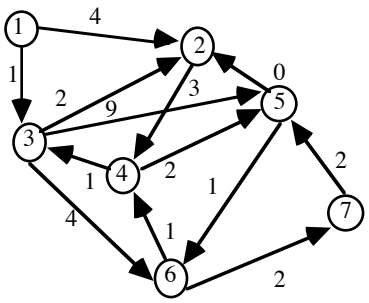

fig. 6: grafo a cui applicare SPT_Dijkstra con r=1

Inizialmente Q={1}, pertanto estraiamo 1 da Q e visitiamo la sua stella uscente. Le etichette dei nodi 2 e 3 valgono rispettivamente 1 e 4, ed entrambi i nodi vengono introdotti nell'insieme Q (Q={2,3}. Poiché l'etichetta minima è 1, estraiamo 3 da Q e facciamo diventare definitiva la sua etichetta (fig. 7).

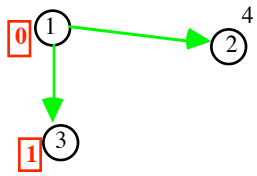

fig.7: etichette dei nodi visitati all'inizio della seconda iterazione. In evidenza le etichette definitive.

Visitando la stella uscente del nodo 3 possiamo migliorare le etichette dei nodi 2 (il cui valore non era ancora stato fissato definitivamente), 5 e 6; il nodo 2 è già presente in Q, mentre 5 e 6 vengono introdotti per la prima volta, pertanto Q={2,5,6} e il nodo di etichetta minima risulta essere 2 che viene estratto da Q e la cui etichetta viene fissata (fig. 8).

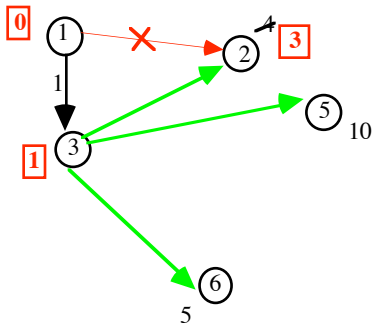

fig. 8: etichette all'inizio della terza iterazione. Variazione nel predecessore del nodo 2

Visitando la stella uscente di 2 riusciamo a migliorare solo l'etichetta del nodo 4 (che viene introdotto in Q), infatti il nodo 6, pur raggiungibile da 2 ha una etichetta minore di 3+5. Q={4,5,6)} e il nodo di etichetta minima è 6, che viene successivamente estratto da Q.

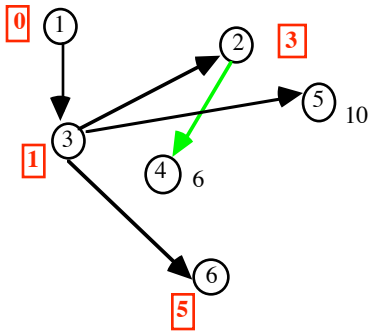

fig. 9: etichette all'inizio della quarta iterazione

Visitando la stella uscente di 6, miglioriamo l'etichetta di 7, ma non quella di 4 che rimane invariata. Q={4,5,7}. Il nodo di etichetta minima è 4 che viene eliminato da Q.

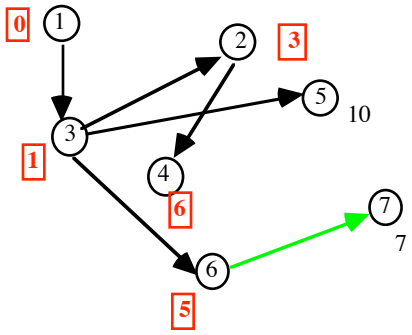

fig. 10: etichette all'inizio della quinta iterazione

Visitando la stella uscente di 4 riusciamo a migliorare l'etichetta di 5, quindi ne aggiorniamo il puntatore. Q={5,7} e il nodo di etichetta minima è 7. Visitando la sua stella uscente non miglioriamo alcuna etichetta così come per il nodo 5 alla successiva iterazione. La situazione finale è raffigurata in fig. 11.

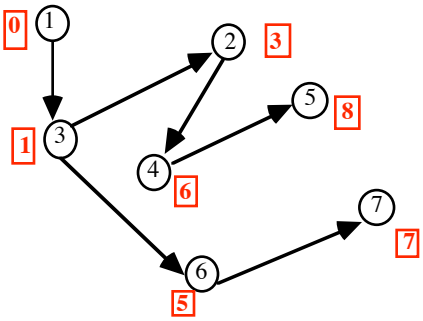

fig. 11: albero dei cammini minimi di radice 1

5.9 Algoritmo SPT_L (label correcting)

Se il grafo non è aciclico e alcune delle lunghezze associate agli archi sono negative, le proprietà viste prima non valgono più. In particolare, non è necessariamente vero che un nodo una volta estratto da *Q* non possa più rientrarvi. Si può dimostrare che l'algoritmo di Dijkstra in taluni casi

patologici ha un comportamento particolarmente inefficiente e la sua complessità risulta di $O(2^n)$, quindi esponenziale.

Per tentare di fornire un algoritmo efficiente, manteniamo ugualmente uno schema algoritmico analogo a quelli visti in precedenza. In particolare, utilizziamo delle etichette $d[i]$ per i nodi tali che a ogni passo dell'algoritmo siano una stima superiore del valore del potenziale ottimo ($d[i] \geq \pi_i$). Riprendendo la formulazione del problema dello "spago", condizioni necessarie di ottimalità sono (*condizioni di Bellman*):

$$d[j] \leq d[i] + c_{ij}.$$

Se consideriamo un qualsiasi cammino da r a j, $P(r,j)$ abbiamo che per le etichette ottime risulta

$$d[j] \leq \sum_{(i,h)\in P(r,j)} c_{ih},$$

e quindi $d[j]$ è proprio la lunghezza del cammino minimo da r a j.

Pertanto un possibile algoritmo non fa altro che controllare arco per arco che valgano le condizioni di Bellman, e in caso contrario aggiorna l'etichetta del nodo testa dell'arco in modo da far valere le condizioni con l'uguaglianza. Lo schema generale dell'algoritmo è riportato qui di seguito:

```
Procedure SPT_L (r,P,d)
   begin
      for i:=1 to n do begin P[i]:=r; d[i]:=M end; d[r]:=0;          {inizializzazione}
      while ∃(i,j)∈A such that d[i]+c_ij < d[j] do
      begin                          {aggiornamento dell'etichetta che non soddisfa le condizioni di Bellman)
        P[j]:=i; d[j]:=d[i]+c_ij;
      end
   end.
```

Se non si adottano particolari criteri per la selezione degli archi da controllare l'algoritmo può richiedere un numero esponenziale di iterazioni.

Complessità

Per valutare la complessità dell'algoritmo visto sopra, in cui non adottiamo alcun particolare criterio nella scelta dell'arco da controllare, è sufficiente considerare il valore delle etichette. Assumiamo che le lunghezze del grafo siano intere. Definendo $C = \max\{|c_{ij}|: (i,j)\in A\}$, possiamo affermare che per ogni nodo i, $-(n-1)C \leq d[i] \leq (n-1)C$. Dato che a ogni iterazione si ha una diminuzione di almeno una unità per un nodo, e che per determinare un arco che viola le condizioni di Bellman, nel caso pessimo, è necessaria la scansione dell'intera lista dei nodi, la complessità dell'algoritmo è $O(2(n-1)Cnm)$, quindi è $O(n^2mC)$.

Esercizio

Costruire un grafo e individuare la sequenza nella quale controllare gli archi tale per cui l'algoritmo SPT_L richiede $O(2^n)$ iterazioni.

Variante polinomiale

Scegliendo opportunamente un criterio da adottare nella ricerca degli archi su cui effettuare il test delle condizioni di Bellman, l'algoritmo SPT_L può diventare polinomiale. Consideriamo gli archi in un ordine fissato qualsiasi. Il criterio con il quale cerchiamo un arco che viola le condizioni di Bellman scandisce la lista ordinata degli archi ciclicamente; quando si trova un arco che non soddisfa le condizioni, si corregge l'etichetta del nodo testa e si prosegue la scansione dall'arco successivo nella lista. Chiamiamo *fase* il completamento di una scansione completa. La misura della complessità si basa sulla stima di quante fasi ci possono essere al massimo prima che tutti gli archi soddisfino le condizioni di Bellman.

Le fasi possono essere al massimo (n-1); la dimostrazione si basa sul significato delle etichette $d[i]$. Al termine della fase k, l'etichetta $d[i]$ fornisce la *lunghezza del cammino minimo dalla radice r a i che utilizza al più k archi*. La dimostrazione è per induzione.

Per k=1 le etichette vengono corrette solamente nelle teste degli archi in FS(r), mentre per tutti gli altri nodi le etichette rimangono invariate a M, dato che non sono raggiungibili dalla radice. Quindi quanto affermato vale.

Supponiamo che valga per $k \geq 1$, vogliamo dimostrare che al termine della fase (k+1) l'etichetta $d[j]$ di ogni nodo ha per valore la lunghezza del cammino minimo dalla radice r a j che utilizza al più (k+1) archi. Consideriamo un nodo j collegato a r da un cammino (r - i_1 - i_2 - … - i_k - j), che risulta essere il più corto cammino di k+1 archi tra r e j, e non ne esistono di più corti che usano meno archi. Si noti che il cammino (r - i_1 - i_2 - … - i_k) risulta essere il più corto tra r e i_k e per l'ipotesi induttiva $d[i_k]$ al termine della fase k deve essere uguale alla lunghezza di questo cammino. Di conseguenza quando, nella fase k+1 si esamina l'arco (i_k, j) fissiamo $d[j]$ uguale alla lunghezza del cammino (r - i_1 - i_2 - … - i_k - j), confermando la tesi.

Come conseguenza, dato che un cammino semplice può contenere al massimo (n-1) archi, si ha che la complessità dell'algoritmo è $O(nm)$.

Aspetti implementativi

Si noti che quando $d[j]$ viene diminuito, potenzialmente tutti gli archi (j,k) in $FS[j]$ possono violare le condizioni di Bellman, infatti $d[j]+c_{jk}$ potrebbe essere diventato minore di $d[k]$. Tenendo conto di questa osservazione, possiamo guidare la ricerca degli archi che violano le condizioni di Bellman memorizzando in un insieme Q i nodi a cui abbiamo corretto le etichette e che potenzialmente hanno archi con le condizioni violate nella propria stella uscente. L'algoritmo diventa:

```
Procedure SPT_L (r,P,d)
  begin
      for i:=1 to n do begin P[i]:=r; d[i]:=M end; d[r]:=0;      {inizializzazione}
    repeat
      select i from Q; Q:=Q\{i};
      for each (i,j)∈FS(i) do
        if d[i]+c_ij < d[j] then
        begin
          P[j]:=i; d[j]:=d[i]+c_ij; Q:=Q∪{j}
        end
    until Q=∅
  end.
```

Quando le inserzioni e le estrazioni da Q vengono effettuate secondo una politica FIFO (quindi Q è una fila), la complessità globale dell'algoritmo è $O(nm)$, dato che si implementa la variante polinomiale della scansione degli archi, come se venissero ordinati nella lista mantenendo le stelle uscenti consecutive. La fila Q permette di "saltare" nella scansione le stelle uscenti che sicuramente non sono interessate dalla modifica di etichette, producendo dei miglioramenti nei tempi di esecuzione dell'algoritmo, sebbene questi miglioramenti non abbiano implicazioni dirette sulla complessità nel caso pessimo.

Esempio: Esecuzione dell'algoritmo SPT_L con politica FIFO

Consideriamo il grafo in fig. 12 e applichiamo l'algoritmo SPT_L a partire dalla radice 1.

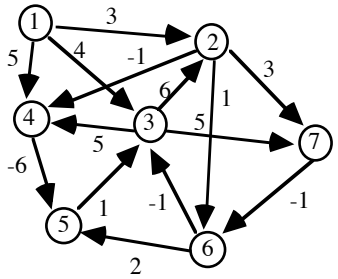

fig. 12: grafo a cui applicare SPT_L con r=1

Osserviamo che il grafo non è aciclico e ha archi di lunghezza negativa, quindi non è appropriato applicare SPT_Acyclic e nemmeno SPT_Dijkstra. Inizialmente Q={1}, estraiamo il nodo 1 dall'insieme Q e visitiamo la sua stella uscente. Poiché tutti i nodi hanno inizialmente etichetta M, riusciamo a correggere l'etichetta di 2, 3 e 4 (fig. 13).

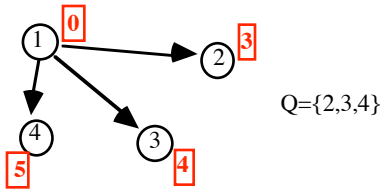

fig. 13: albero e insieme Q al termine della prima iterazione

All'iterazione successiva viene estratto da Q il nodo 2, in quanto introdotto prima degli altri, e viene visitata la sua stella uscente. L'operazione porta alla correzione delle etichette dei nodi 4 (già presente in Q), 6 e 7 (fig. 14).

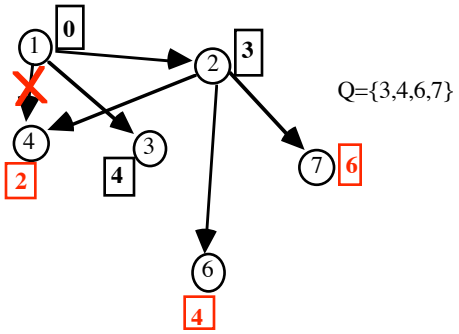

fig. 14: albero e insieme Q al termine della terza iterazione

Visitando la stella uscente del nodo 3 si migliora l'etichetta del nodo 6 (fig. 15).

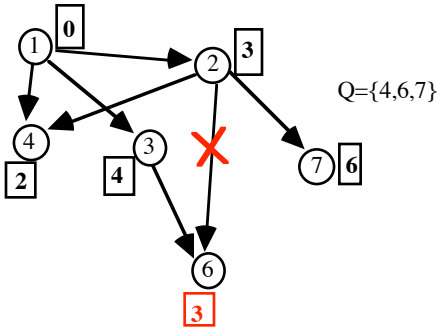

fig. 15: albero e insieme Q al termine della quarta iterazione

Visitando la stella uscente del nodo 4 si migliora l'etichetta del nodo 5 che viene introdotto in Q (fig. 16).

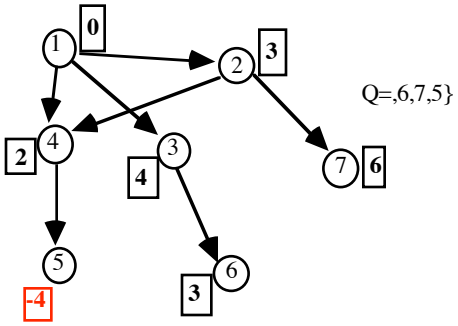

fig. 16: albero e insieme Q al termine della quinta iterazione

Successivamente viene estratto il nodo 6 dal quale non è possibile migliorare alcuna etichetta. Altrettando avviene per il nodo 7. Quindi estraiamo il nodo 5 e visitiamo la sua stella uscente migliorando l'etichetta del nodo 3 che viene introdotto in Q. Si noti che è la seconda volta che questo

avviene. Si noti anche che ora le etichette di tutti i nodi che discendono da 3 (cioè 6) hanno etichette migliorabili, pur non comportando necessariamente una modifica dei predecessori (fig. 17).

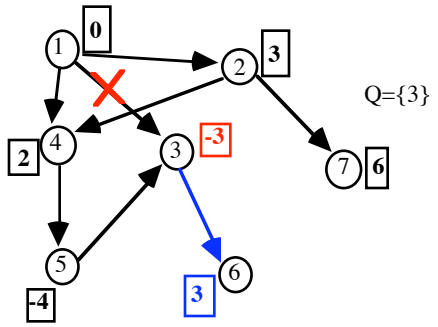

fig. 17: albero e insieme Q al termine della ottava iterazione

La visita della stella uscente del nodo 3 comporta il miglioramento delle etichette dei nodi 6, 7 che vengono introdotti in Q (fig. 18). Le successive iterazioni non portano ad altre modifiche.

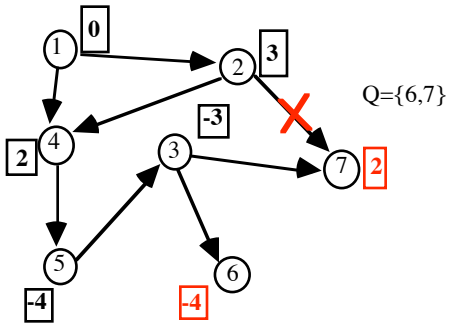

fig. 18: albero e insieme Q al termine della nona iterazione

5.10 Individuazione di cicli di lunghezza negativa

Esistono vari metodi per riconoscere i cicli di lunghezza negativa. Se esistono tali cicli tutti gli algoritmi visti in precedenza non si arrestano e ciclano all'infinito.

Sia C il massimo valore assoluto delle lunghezze degli archi; è evidente che se non esistono cicli di lunghezza negativa non potremo mai avere etichette $d[j]<-(n-1)C$. Non appena nelle iterazioni dell'algoritmo si verifica una tale eventualità possiamo fermarci. Il ciclo di lunghezza negativa si ricostruisce risalendo i predecessori P a partire da j, fino a che non si raggiunge nuovamente j.

I cicli di lunghezza negativa possono venire trovati anche in modo più efficiente. Se utilizziamo l'algoritmo SPT_L implementato con una fila, sappiamo che un nodo può venire introdotto in coda non più di n-1 volte, se non esistono cicli negativi. Pertanto non appena un nodo viene introdotto in Q per l'n-esima volta siamo in presenza di un ciclo negativo. Quindi con una complessità di $O(nm)$ individuiamo i cicli di lunghezza negativa. I nodi che sono ancora nella fila Q dopo n-1 iterazioni dell'algoritmo fanno parte di cicli negativi. Per ricostruire i cicli è sufficiente risalire i predecessori dei nodi presenti in Q.

Esercizio
Si determini se nel grafo di fig. 12 in cui l'arco (3,2) ha lunghezza 5 invece di 6, esistono cicli di lunghezza negativa, applicando il procedimento illustrato in precedenza.

**5.11 Cambiamento di radice dell'albero dei cammini minimi*

Supponiamo di dover calcolare gli alberi dei cammini minimi per più radici $(r_1, r_2, \ldots, r_p)$. Bisognerebbe quindi applicare l'algoritmo di ricerca dei cammini minimi per ognuna delle radici. Ad ogni modo, dopo aver calcolato il primo albero relativo alla radice r_1 possiamo cercare di trarre vantaggio dalle informazioni relative alla soluzione trovata per velocizzare il calcolo degli altri alberi. Un modo possibile tiene conto dei *costi ridotti* degli archi rispetto alla soluzione trovata. Siano $d[i]$ le etichette ottime relative all'albero dei cammini minimi di radice r_1, il costo ridotto dell'arco (i,j) è dato da:

$$\bar{c}_{ij} = c_{ij} + d[i] - d[j].$$

Confrontiamo costo e costo ridotto di un cammino $P_{hk} = (h, i_1, i_2, \ldots, i_s, k)$:

$$C(P_{hk}) = \sum_{(i,j)\in P_{hk}} c_{ij}$$

$$\bar{C}(P_{hk}) = \sum_{(i,j)\in P_{hk}} (c_{ij} + d[i] - d[j]) = \sum_{(i,j)\in P_{hk}} c_{ij} + d[h] - d[k].$$

Il costo ridotto di un cammino risulta quindi uguale al costo del cammino a meno di una costante che dipende dal nodo iniziale e dal nodo finale, ma è indipendente dai nodi intermedi. Questo significa che se un cammino è minimo rispetto ai costi ridotti, lo è anche rispetto ai costi originali. Il vantaggio derivante dall'utilizzo dei costi ridotti consiste nel fatto che i costi ridotti hanno valore maggiore o uguale a zero (perché?), pertanto è possibile e più conveniente applicare l'algoritmo di Dijkstra. La complessità di calcolare gli alberi dei cammini minimi per le radici $(r_1, r_2, \ldots, r_p)$ su di un grafo con costi qualsiasi è dato da $O(mn)$ per il calcolo del primo albero più p volte l'applicazione della versione più efficiente dell'algoritmo di Dijkstra.

Esercizio
A partire dall'albero dei cammini minimi del grafo in fig. 12 rappresentato in fig. 18, si determini l'albero dei cammini minimi di radice 3.

**5.12 Algoritmo di Floyd-Warshall: cammino minimo tra ogni coppia di nodi*

Dovendo calcolare il cammino minimo tra ogni coppia di nodi si può ricorrere al procedimento descritto brevemente nel paragrafo precedente che ha complessità nel caso pessimo di $O(m\,n + D(n,m))$ dove $D(n,m)$ è la complessità dell'algoritmo di Dijkstra utilizzato che varia in base alla struttura dati utilizzata. In alternativa si può ricorrere all'algoritmo di Floyd-Warshall che, senza l'utilizzo di particolari strutture dati (sono sufficienti due matrici $n \times n$) calcola con complessità costante i cammini richiesti. L'idea dell'algoritmo si basa su una semplice regola ricorsiva. Se

indichiamo con $d^k[i,j]$ la lunghezza del cammino da i a j che usa come nodi intermedi $\{1,\ldots, k\text{-}1\}$ vale quanto segue:

$$d^{k+1}[i,j] = \min\{d^k[i,j], d^k[i,k]+d^k[k,j]\}.$$

Il cammino minimo da i a j che usa come nodi intermedi $\{1,\ldots,k\}$ è dato dal più corto tra il cammino minimo da i a j che usa come nodi intermedi $\{1,\ldots,k\text{-}1\}$ e quello che si ottiene componendo il cammino più corto tra i e k e quello più corto tra k e j che utilizzano $\{1,\ldots,k\text{-}1\}$ come nodi intermedi. La regola di ricorsione è completata dal valore iniziale definito solo se tra i e j esiste il collegamento diretto:

$$d^1[i,j] = \begin{cases} c_{ij} & \text{se } (i,j)\in A \\ \infty & \text{se } (i,j)\notin A \end{cases} .$$

Per tenere traccia dei cammini ricorriamo ai predecessori come nel caso dell'albero, solo che in questo caso abbiamo bisogno di una matrice invece che di un vettore. L'algoritmo è sintetizzato nel seguito.

```
Procedure Floyd_Warshall(c,p,d);
  begin
    for i:=1 to n
      for j:=1 to n
        if i=j then begin d[i,i]:=0; p[i,i]:=ø end
        else if (i,j)∈A then begin d[i,j]:=c_ij; p[i,j]:=i end
          else d[i,j]:=M;
    for k:=1 to n
      for i:=1 to n
        for j:=1 to n
          if d[i,j]>d[i,k]+d[k,j] then
            begin
              d[i,j]:=d[i,k]+d[k,j]; p[i,j]:=p[k,j];
            end
  end.
```

È banale analizzare la complessità computazionale dell'algoritmo: dato che l'operazione dominante è costituita da tre cicli innestati che scandiscono tutti i nodi, la complessità è $O(n^3)$. Anche il riconoscimento dei cicli negativi è immediato: se al termine dell'algoritmo $d[i,i]$ risulta essere minore di 0 significa che il nodo i appartiene ad un ciclo negativo.

6 Flusso massimo

Dato un grafo $G=(N,A)$ orientato con capacità u_{ij} associate agli archi, il problema del flusso massimo consiste nel trovare la massima quantità di flusso che può venire inviata da un nodo $s\in N$ (detto *sorgente*) a un nodo $t\in N$ (detto *pozzo*), rispettando le capacità degli archi e il fatto che il flusso non va disperso nei nodi intermedi. Le variabili del problema sono le quantità di flusso che

vengono inviate sui singoli archi: x_{ij}, per ogni $(i,j) \in A$. Il problema può venire formulato nel seguente modo:

$$\max \sum_{(s,j)} x_{sj}$$
$$0 \le x_{ij} \le u_{ij} \qquad \forall (i,j) \in A$$
$$-\sum_{(i,j) \in FS(i)} x_{ij} + \sum_{(j,i) \in BS(i)} x_{ji} = 0 \quad \forall i \in N \setminus \{s,t\}.$$

Il primo gruppo di vincoli, associati agli archi, sono detti *vincoli di capacità*. Il secondo gruppo di vincoli, associato a tutti i nodi del grafo con eccezione di s e t, sono i cosiddetti *vincoli di conservazione del flusso* e stabiliscono che in ciascun nodo intermedio il flusso entrante deve essere uguale al flusso uscente, garantendo quindi l'assenza di dispersioni. La funzione obiettivo calcola il flusso complessivo uscente dal nodo s. Si noti che potevamo indifferentemente considerare in alternativa la somma del flusso entrante in t, per via della conservazione del flusso.
Volendo scrivere il problema in modo più uniforme possiamo introdurre un arco fittizio di ritorno (t,s) con capacità $u_{ts}=M$, con M valore opportunamente grande tale che M sia maggiore del valore della soluzione ottima. Il problema del flusso massimo può venire riscritto equivalentemente come segue:

$$v = \max x_{ts}$$
$$0 \le x_{ij} \le u_{ij} \qquad \forall (i,j) \in A'$$
$$-\sum_{(i,j) \in FS(i)} x_{ij} + \sum_{(j,i) \in BS(i)} x_{ji} = 0 \quad \forall i \in N.$$

dove $A'=A \cup \{(t,S)\}$.

6.1 Esempio di problema di flusso massimo

Si consideri una officina meccanica con M operai (con identiche capacità lavorative) e un insieme $J=\{1,\ldots,n\}$ macchine da riparare. Ogni macchina da riparare j è caratterizzata da una tripla (p_j,r_j,d_j), dove p_j è il tempo necessario per ripararla, r_j è l'ora alla quale il proprietario la consegna all'officina, e d_j l'ora alla quale passa a ritirarla. Le riparazioni possono venire interrotte e riprese (prelazione) senza perdite di tempo, ma devono essere eseguite da un operaio alla volta. Ci poniamo il problema di vedere se è possibile trovare un assegnamento delle riparazioni agli operai in modo da riuscire a consegnare le macchine entro l'orario stabilito. Il problema può essere risolto per mezzo della ricerca di un flusso massimo su un opportuno grafo. Prendiamo in considerazione un esempio in cui $M=3$ e i dati relativi ai lavori in J sono riportati dalla seguente tabella:

J	1	2	3	4
p_j	1.5	1.25	2.1	3.6
r_j	3	1	3	5
d_j	5	4	7	9

Tabella 3: tempi di esecuzione, di arrivo e di partenza dei lavori

Per costruire il grafo, ordiniamo gli "eventi" (consegne o ritiri di macchine) in ordine crescente eliminando le ripetizioni:

1, 3, 4, 5, 7, 9.

Tra due "eventi" consecutivi abbiamo intervalli di tempo di lavoro continuato (T_{kl}):

T_{13}(2 ore), T_{34}(1 ora), T_{45}(1 ora), T_{57}(2 ore), T_{79}(2 ore).

Durante l'intervallo T_{kl} possono venir riparate le macchine che sono arrivate prima di k ($r_j \le k$) e vengono ritirate dopo l ($d_j \ge l$). Il grafo viene costruito come segue:

- l'insieme dei nodi comprende una sorgente s, un pozzo t, un nodo per ogni macchina $j \in J$ e uno per ogni intervallo T_{kl};
- ci sono archi che collegano s a ogni nodo $j \in J$, questi archi hanno capacità uguale al tempo di riparazione p_j; ogni nodo $j \in J$ è a sua volta collegato a tutti gli intervalli T_{kl} nei quali la macchina può essere riparata, la capacità di tali archi è data da $(l-k)$; infine abbiamo gli archi che collegano i nodi T_{kl} al pozzo t, la capacità di tali nodi è data dalla capacità lavorativa dell'officina in tale intervallo cioè $M(l-k)$.

Il flusso sugli archi del grafo così costruito rappresenta la quantità di lavoro che viene svolto.
Se il flusso massimo che si può spedire da s a t è uguale alla somma dei tempi di riparazione, allora è possibile riparare tutte le macchine entro i tempi stabiliti.

6.2 Proprietà di flussi e tagli

Una partizione dei nodi in due sottoinsiemi (N_s,N_t) tale che $s \in N_s$, e $t \in N_t$, si chiama *taglio s-t*. Gli archi che attraversano il taglio (che hanno un estremo in N_s e l'altro in N_t) vengono a loro volta distinti in due sottoinsiemi: l'insieme degli *archi diretti* $A^+(N_s,N_t)=\{(i,j) \in A: i \in N_s, j \in N_t\}$, e l'insieme degli *archi inversi* $A^-(N_s,N_t)=\{(i,j) \in A: i \in N_t, j \in N_s\}$.
Dato un taglio (N_s,N_t) la *quantità di flusso che attraversa il taglio* è dato dalla quantità di flusso che percorre gli archi diretti meno la quantità di flusso che percorre gli archi inversi:

$$x(N_s,N_t) = \sum_{(i,j) \in A^+(Ns,Nt)} x_{ij} - \sum_{(i,j) \in A^-(Ns,Nt)} x_{ij}.$$

La *capacità di un taglio* (N_s,N_t) è data dalla somma delle capacità degli archi diretti:

$$U(N_s,N_t) = \sum_{(i,j)\in A^+(Ns,Nt)} u_{ij}.$$

Ovviamente per ogni flusso ammissibile e per ogni taglio s- t si ha che il flusso che attraversa il taglio è minore o uguale della capacità del taglio stesso. Dato un flusso ammissibile $\bar{x}$ di valore $\bar{v}$, è facile verificare anche che, per la conservazione del flusso, per ogni taglio (N_s,N_t)

$$\bar{x}(N_s,N_t) = \bar{v}.$$

Intuitivamente è chiaro che il taglio che limita maggiormente il passaggio del flusso da s a t è quello di capacità minima. Possiamo anche affermare che se un flusso x è tale che esiste un taglio di capacità pari alla quantità di flusso che lo attraversa, tale flusso è ottimo. Ora risulta abbastanza spontaneo chiedersi se il valore della capacità del taglio minimo non sia proprio uguale al valore del flusso massimo. Possiamo dimostrarlo costruttivamente, pensando a un possibile algoritmo per la soluzione del problema di flusso massimo.

6.3 Algoritmo dei cammini aumentanti

Cerchiamo di risolvere il problema incrementalmente: dato un flusso ammissibile proviamo a vedere se è migliorabile, ovvero se esiste modo di fare arrivare altro flusso da s a t. Per scoprire se tale aumento di flusso è attuabile, introduciamo il *grafo residuale di un flusso* $\bar{x}$. Dato un grafo $G=(N,A)$ con capacità sugli archi u, e un flusso ammissibile $\bar{x}$ definiamo il *grafo residuale* $G_R(\bar{x}) = (N,A(\bar{x}))$ dove gli archi sono definiti come segue:

$$A(\bar{x}) = A^+(\bar{x}) \cup A^-(\bar{x}),$$
$$A^+(\bar{x}) = \{(i,j): (i,j)\in A, \bar{x}_{ij}<u_{ij}\},$$
$$A^-(\bar{x}) = \{(i,j): (j,i)\in A, \bar{x}_{ji}>0\}.$$

Esempio: grafo residuale

Consideriamo il grafo in fig. 19 con indicato anche un flusso ammissibile.

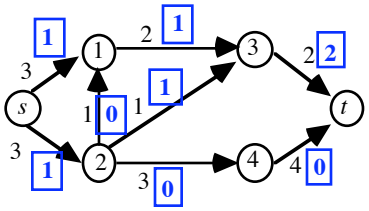

fig. 19: rete di flusso: nei quadrati sono indicati i valori di flusso sui vari archi

Possiamo costruire il grafo residuale andando a vedere, arco per arco, se il flusso può essere aumentato (introducendo un arco di $A^+(\bar{x})$) e/o diminuito (introducendo un arco di $A^-(\bar{x})$) (fig. 20).

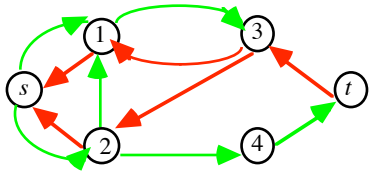

fig. 20: grafo residuale in cui sono distinti gli archi di $A^+(\bar{x})$ e quelli di $A^-(\bar{x})$

Si noti che gli archi del grafo residuale corrispondono agli archi del grafo originale sui quali è possibile effettuare una variazione di flusso. La variazione di flusso corrispondente ad un arco (i,j) di $A^+(\bar{x})$ è in positivo, cioè è possibile aumentare il flusso $\bar{x}_{ij}$; la *capacità residua* dell'arco $(i,j)\in A$ rispetto a questa variazione è data da $(u_{ij}-\bar{x}_{ij})$, non potendo superare la capacità massima. La variazione di flusso corrispondente ad un arco (i,j) di $A^-(\bar{x})$ è in negativo, cioè è possibile diminuire il flusso $\bar{x}_{ji}$; la capacità residua dell'arco $(j,i)\in A$ rispetto a questa variazione è data da $\bar{x}_{ij}$, non potendo ottenere un flusso negativo.

Dato un flusso ammissibile $\bar{x}$, si consideri il grafo residuale $G_R(\bar{x})$. Cercando un cammino da s a t, corrispondente ad un possibile aumento del flusso, possono verificarsi due casi esclusivi:

i) esiste un cammino da s a t in $G_R(\bar{x})$;

ii) esiste un taglio (N_s,N_t) in $G_R(\bar{x})$ tale che $A^-(\bar{x})(N_s,N_t) = \varnothing$.

Nel primo caso il flusso $\bar{x}$ non è ottimo e può venire incrementato di una quantità strettamente positiva; nel secondo caso invece $\bar{x}$ è ottimo.

Per cercare il cammino da s a t applichiamo l'algoritmo di visita del grafo residuale $G_R(\bar{x})$ a partire dal nodo s. L'algoritmo di visita termina o marcando t (caso i), oppure lasciando t ed eventuali altri nodi non marcati (caso ii).

Nel primo caso abbiamo trovato un *cammino aumentante* (su cui si può variare il flusso senza violare i vincoli di capacità). Sia P_{st} tale cammino. Definiamo ora la quantità massima di flusso θ che possiamo spedire sul cammino trovato, definita dal minimo delle capacità residue degli archi corrispondenti nel grafo originale:

$$\theta = \min\{\{u_{ij}-\bar{x}_{ij}: (i,j)\in A^+(\bar{x})\cap P_{st}\}, \{\bar{x}_{ji}: (i,j)\in A^-(\bar{x})\cap P_{st}\}\}.$$

Si noti che, per come è stato costruito il grafo residuale, $\theta>0$. La variazione del flusso, detta operazione di *aumento del flusso*, viene effettuata secondo le seguenti regole:

$$x'_{ij} = \begin{cases} \bar{x}_{ij} + \theta & \text{se } (i,j)\in A^+(\bar{x})\cap P_{st}\} \\ \bar{x}_{ij} - \theta & \text{se } (j,i)\in A^-(\bar{x})\cap P_{st}\} \\ \bar{x}_{ij} & \text{altrimenti} \end{cases}$$

Il valore del nuovo flusso x' è pari al valore di $\bar{x}$ incrementato di θ, pertanto il flusso da cui siamo partiti non era ottimo.

Nel secondo caso invece possiamo costruire un taglio mettendo in N_s l'insieme dei nodi marcati nella visita di $G_R(\bar{x})$ e in N_t tutti gli altri. Consideriamo gli archi del taglio nel grafo originale G. Per costruzione risulta che se $(i,j) \in A^+(N_s,N_t)$, $\bar{x}_{ij} = u_{ij}$, viceversa se $(i,j) \in A^-(N_s,N_t)$ allora $\bar{x}_{ij} = 0$. In caso contrario esisterebbe in $G_R(\bar{x})$ un arco (i,j) e quindi j sarebbe stato marcato e non si troverebbe nell'insieme N_t. Pertanto risulta che

$$\bar{x}(N_s,N_t) \sum_{(i,j)\in A^+(N_s,N_t)} \bar{x}_{ij} - \sum_{(i,j)\in A^+(N_s,N_t)} \bar{x}_{ij.} = \sum_{(i,j)\in A^+(N_s,N_t)} u_{ij.} = U(N_s,N_t).$$

Questo dimostra l'equivalenza tra valore del flusso massimo e capacità del taglio minimo. Risulta ora abbastanza semplice fornire un algoritmo per risolvere il problema del flusso massimo, in cui partendo da un flusso ammissibile iniziale, per esempio il flusso nullo, si va alla ricerca di un cammino aumentante utilizzando il grafo residuale che viene aggiornato ad ogni variazione di flusso. L'algoritmo in questa forma è noto come algoritmo dei cammini aumentanti o algoritmo di Ford-Fulkerson.

```
Procedure Cammini_aumentanti (G,x);
  begin
    x:=0;
    repeat
      Gr:=Grafo_residuale(x);
      Visita (Gr,s);
      if P[t]≠0 then Aumenta_flusso(P_st,x)
    until P[t]=0;
  end.
```

La procedura Aumenta_flusso spedisce la massima quantità θ di flusso sul cammino sa s a t trovato dalla visita, secondo la regola enunciata sopra.

Esempio: esecuzione dell'algoritmo dei cammini aumentanti

Continuando l'esempio di fig. 19-20, la visita del grafo a partire dal nodo s individua un primo cammino s, 2, 4, t. θ=min{3-1, 3-0, 4-0}=2 poiché è composto da soli archi di $A^+(\bar{x})$. Il nuovo flusso si ottiene sommando θ=2 al flusso preesistente sugli archi del cammino. Quindi $\bar{x}_{s2}$=3, $\bar{x}_{24}$=$\bar{x}_{4t}$=2. Il nuovo grafo residuale è quello in fig. 21.

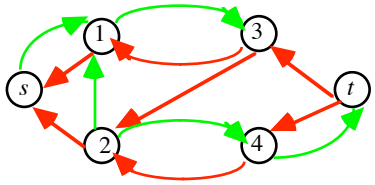

fig. 21: grafo residuale dopo il primo aggiornamento del flusso

Applicando nuovamente la visita a partire dal nodo s troviamo un secondo cammino: s, 1, 3, 2, 4, t. Si noti che in questo caso utilizziamo un arco di $A^-(\bar{x})$ (l'arco (3,2)). θ=min{3-1, 2-1, 1, 3-2,4-2}=1.

L'aggiornamento del flusso comporta un aumento di 1 unità di flusso sugli archi (s,1), (1,3), (2,4), (4,t) mentre una diminuzione di 1 unità di flusso sull'arco (2,3). Il nuovo grafo residuale è quello riportato in fig. 22; su tale grafo non è più possibile trovare un cammino da s a t, infatti vi è un taglio $(N_s, N_t)=(\{s, 1\}, \{2, 3, 4, t\})$.

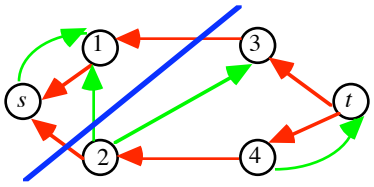

fig. 22: grafo residuale dopo il secondo aggiornamento del flusso e taglio (N_s, N_t)

Il flusso aggiornato dopo le due iterazioni fatte è rappresentato in fig. 23. Si noti che nel taglio individuato il flusso sugli archi di $A^+(N_s,N_t)=\{(s,1), (1,3)\}$ è alla capacità superiore, mentre sugli archi di $A^-(N_s,N_t)=\{(2,1)\}$ è a zero, inoltre il flusso che attraversa il taglio è pari alla capacità del taglio stesso.

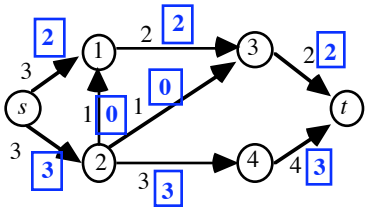

fig. 23: flusso aggiornato dopo due iterazioni di aumento di flusso

*Algoritmo di Edmonds e Karp

Se il cammino aumentante non viene scelto opportunamente si possono verificare dei casi patologici che fanno crescere a dismisura il numero delle iterazioni necessarie a determinare la soluzione ottima.

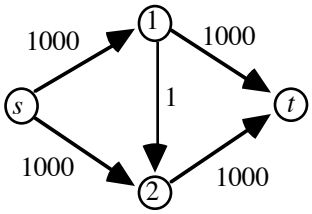

fig. 24: esempio di caso patologico

Nell'esempio in fig. 24 è evidente che il flusso massimo è 2000 e tale soluzione può venire determinata con sole due iterazioni dell'algoritmo dei cammini aumentanti. Se a ogni passo l'algoritmo dei cammini aumentanti considera alternativamente gli archi (1,2) e (2,1) del grafo residuale, il flusso massimo viene determinato effettuando 2000 iterazioni, ognuna in grado di aumentare il flusso di una sola unità. Per evitare questi casi bisogna scegliere opportunamente i cammini su cui aumentare il flusso.

L'algoritmo di Edmonds e Karp è di fatto una particolarizzazione dell'algoritmo dei cammini aumentanti che garantisce la polinomialità del numero di iterazioni. L'unico accorgimento adottato è nella scelta del cammino aumentante. A ogni iterazione viene scelto il cammino aumentante di lunghezza minima cioè che utilizza il minor numero di archi. La ricerca di tale cammino può venire fatta realizzando la visita a ventaglio (Breadth First Search), quindi implementando l'insieme Q dell'algoritmo di visita con una fila. Questo accorgimento non appesantisce la complessità dell'algoritmo di visita che rimane $O(m)$.

La scelta del cammino aumentante con il minimo numero di archi ha come effetto quello di considerare cammini di lunghezza crescente con il numero di iterazioni dell'algoritmo. Di iterazione in iterazione, la configurazione del grafo residuale sul quale cerchiamo i cammini aumentanti, si evolve poiché nell'operazione di aumento flusso almeno un arco del cammino raggiunge la capacità massima o zero, quindi, ad ogni iterazione, almeno un arco del grafo residuale sparisce. Questa è una conseguenza diretta della scelta della quantità di flusso θ da inviare lungo il cammino. Siccome i cammini da s a t nel grafo residuale possono avere al massimo (n-1) archi, il numero di iterazioni è limitato. Adesso l'unico punto delicato da risolvere è la valutazione di quante volte un arco (i,j) può sparire dal grafo residuale o, in altri termini, quante volte un arco del grafo originale si satura o si annulla. Consideriamo una data iterazione dell'algoritmo in cui viene utilizzato l'arco (i,j) del grafo residuale (fig. 25) come facente parte del cammino da s a t e per cui θ è uguale alla capacità residua dell'arco. Indichiamo con $d[i]$ e $d[j]$ il numero di archi del cammino tra s e i e tra s e j rispettivamente. Poiché (i,j) fa parte del cammino deve essere:

$$d[j]=d[i]+1.$$

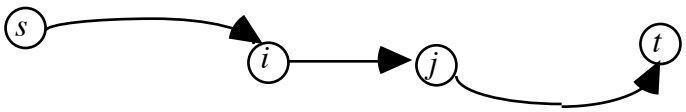

fig. 25: primo cammino aumentante sul grafo residuale che utilizza (i,j)

Una volta che abbiamo aggiornato il flusso, l'arco (i,j) sparisce dal grafo residuale dato che abbiamo esaurito la sua capacità residua. Comparirà, se non già presente, l'arco (j,i). Per poter rimettere in gioco l'arco (i,j) deve succedere che in una qualche iterazione dell'algoritmo viene trovato un cammino aumentante (di lunghezza maggiore o uguale a quella dei cammini trovati in precedenza) che utilizza l'arco (j,i) (fig. 26).

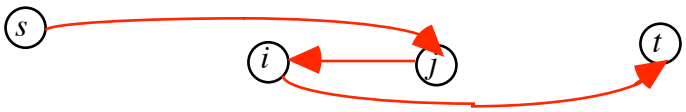

fig. 26: cammino aumentante che utilizza (j,i)

Quindi, considerando il nuovo cammino e indicando con $d'[i]$ e $d'[j]$ il numero di archi del cammino da s a i e a j, si ha che:

$$d'[i] = d'[j]+1;$$

con $d'[j] \geq d[j]$. Una volta modificato il flusso lungo il cammino trovato, nel nuovo grafo residuale compare nuovamente l'arco (i,j). Tale arco potrà venire utilizzato (ed eventualmente saturato) solo quando troveremo un nuovo cammino aumentante tra s e t che utilizza (i,j) (fig. 27), quindi quando:

$$d''[j] = d''[i]+1;$$

con $d''[i] \geq d'[i]$.

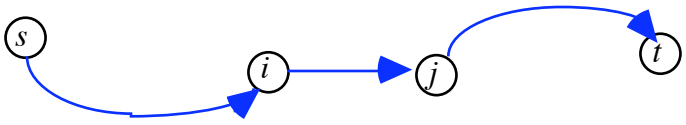

fig. 27: secondo cammino aumentante sul grafo residuale che utilizza (i,j)

Valutiamo di quanto si è allungato il cammino da s ad j rispetto alla prima volta che l'arco (i,j) è stato utilizzato. Sappiamo che

$$d''[j] = d''[i]+1 \geq d'[i]+1 = d'[j]+1 + 1 \geq d[j] + 2.$$

Questo significa che la lunghezza del cammino che utilizza l'arco (i,j) si è allungata di almeno 2, pertanto, dato che i cammini possono essere lunghi al più $(n-1)$ archi, un arco può sparire dal grafo residuale al più $(n-1)/2$ volte. Ripetendo questo ragionamento per tutti gli m archi, abbiamo che il numero massimo di iterazioni dell'algoritmo di Edmonds e Karp è $O(nm)$. Poiché l'operazione computazionalmente più pesante di ogni iterazione è la visita del grafo residuale di complessità $O(m)$, la complessità dell'algoritmo risulta $O(m^2n)$.

*Algoritmo di Capacity Scaling

Anche l'algoritmo di Capacity Scaling è un particolare algoritmo di cammini aumentanti. L'idea che è alla base dell'algoritmo di Capacity Scaling per il problema del flusso massimo è quella di aumentare il flusso lungo cammini con capacità residua elevata. Individuare il cammino da s a t di capacità massima, pur essendo relativamente semplice, porterebbe pochi benefici dal punto di vista della complessità rispetto ad altri algoritmi. Infatti la maggiore efficienza del cammino trovato verrebbe pagata in maggiore tempo di calcolo necessario per trovare tale cammino. Essendo comunque l'idea interessante si ricorre ad una approssimazione che mantenga la semplicità di un algoritmo di visita nella ricerca di un cammino di capacità elevata da s a t. Si fissa un parametro Δ per selezionare gli archi con sufficiente capacità. Dato un flusso ammissibile $\bar{x}$, chiamiamo $G_R(\bar{x},\Delta)$ il sottografo di $G_R(\bar{x})$ che contiene solo gli archi di capacità maggiore o uguale a Δ.

Assumendo capacità intere, si ha che $G_R(\bar{x}) = G_R(\bar{x},1)$. Indicando con U la capacità massima degli archi di G, l'algoritmo può essere riassunto come segue:

```
Procedure Capacity Scaling (G,x);
  begin
    x:=0;
    Δ:=2⌊log U⌋;
    while Δ≥1 do
    begin
      Cammini_aumentanti (Gr(x,Δ),x)
      Δ:=Δ/2;
    end
  end.
```

Considerando capacità intere, è abbastanza semplice convincersi che l'algoritmo termina con la soluzione ottima. Infatti il parametro Δ viene dimezzato a ogni iterazione esterna e all'ultima iterazione si ha che $G_R(\bar{x}) = G_R(\bar{x},1)$. Quindi se il flusso trovato dall'algoritmo di Capacity Scaling non ammette cammini aumentanti possiamo a ragione affermare che tale flusso è ottimo. Per quel che riguarda l'analisi della complessità dell'algoritmo la questione è un po' più delicata. Si può dimostrare che ad ogni iterazione del ciclo **while** la procedura **Cammini_aumentanti** effettua al massimo $2m$ iterazioni (ognuna delle quali costa $O(m)$ essendo una visita del grafo residuale). Le iterazioni del ciclo sono $O(\log U)$ dato che a ogni iterazione il valore del parametro Δ viene dimezzato. Ne consegue che la complessità dell'algoritmo è $O(m^2 \log U)$.

Esercizio
Descrivere un algoritmo per il calcolo del cammino di capacità residua massima da s a t nel grafo residuale. Analizzarne la complessità.

6.4 Caso particolare: accoppiamento di cardinalità massima

Si consideri un grafo bipartito $G=(S,T,A)$ con l'insieme degli archi $A \subseteq S \times T$. Il problema dell'accoppiamento di cardinalità massima consiste nel determinare un sottoinsieme M di archi di massima cardinalità tali che M non contenga più di un arco incidente su uno stesso nodo. Il problema può venire formulato matematicamente nel modo seguente:

$$\max \sum_{(i,j)\in A} x_{ij}$$
$$\sum_{(i,j)\in FS(i)} x_{ij} \le 1 \qquad \forall i \in S$$
$$\sum_{(i,j)\in BS(j)} x_{ij} \le 1 \qquad \forall j \in T$$
$$x_{ij} \in \{0,1\} \qquad \forall (i,j) \in A,$$

dove la variabile logica x_{ij} vale uno se e solo se l'arco $(i,j) \in M$. Lo stesso problema può venire formulato in termini di flusso massimo su un opportuno grafo di cui riportiamo un esempio in fig. 28.

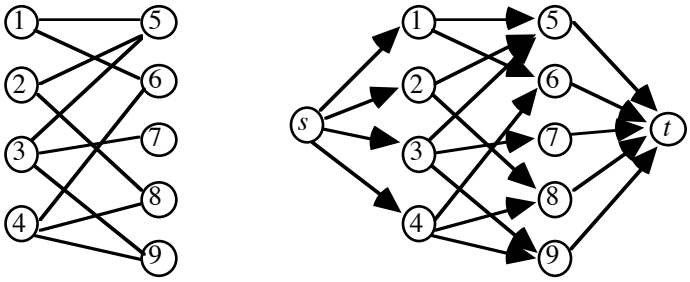

fig. 28: grafo bipartito e formulazione in termini di flusso massimo

Al grafo bipartito in cui i nodi degli insiemi S e T sono disposti sulle due colonne centrali, sono stati aggiunti i nodi sorgente s e pozzo t. Agli archi, orientati da S a T, sono stati aggiunti gli archi che collegano s con tutti i nodi di S e quelli che collegano tutti i nodi di T a t. Se le capacità degli archi vengono poste a 1 e si cerca il flusso massimo da s a t, è evidente che per ciascun nodo di S e ciascun nodo di T può transitare al più una unità di flusso. Questo rende di fatto equivalente il problema di flusso massimo a quello dell'assegnamento di cardinalità massima su grafo bipartito, in cui vi è al più un arco uscente da ciascun nodo di S e al più un arco entrante in ciascun nodo di T. L'unica differenza che si può ravvisare tra le due formulazioni è che, mentre il problema di flusso massimo tra le varie soluzioni ne ammette anche di quelle con valori dei flussi frazionari, il problema dell'accoppiamento di cardinalità massima, essendo un problema con variabili di tipo logico, ammette solo soluzioni in cui le variabili sono intere. Però, in un problema di flusso massimo in cui le capacità degli archi sono intere, data una soluzione ottima con valori dei flussi frazionari, è sempre possibile trovare una soluzione ottima equivalente in cui tutti i flussi sono interi. Nel caso particolare del problema di accoppiamento di cardinalità massima possiamo quindi trovare una soluzione in cui i flussi assumono valori 0 o 1.

L'algoritmo dei cammini aumentanti trae particolare vantaggio dal fatto che nel grafo vi siano capacità unitarie. A ogni iterazione infatti il flusso aumenta esattamente di una unità. Siccome l'accoppiamento di cardinalità massima ha valore al più uguale al minimo tra la cardinalità di S e la cardinalità di T, il problema può venire risolto in tempo $O(m \min\{|S|,|T|\})$.

Esercizio

Applicare l'algoritmo descritto all'istanza di fig. 7. A ogni iterazione i cammini aumentanti alternano archi di $A^+(\bar{x})$ e archi di $A^-(\bar{x})$ iniziando e terminando sempre con un arco di $A^+(\bar{x})$. Dimostrare che questo avviene sempre per qualsiasi problema di accoppiamento.

7 Flusso di costo minimo

Il problema di flusso di costo minimo è un problema di ottimizzazione su reti assai generale: sia il problema di flusso massimo che il problema dell'albero dei cammini minimi possono essere visti come casi particolari del problema di flusso di costo minimo. Una rete di flusso è specificata da un grafo orientato $G=(N,A)$, a ogni nodo $i \in N$ è associato un valore b_i (detto *bilancio* del nodo), a ogni arco (i,j) sono associati il *costo* unitario c_{ij} e una *capacità* u_{ij} che limita la quantità massima di

flusso che può transitare. I bilanci ai nodi regolano il flusso sulla rete. Un nodo con bilancio $b_i = 0$ è detto nodo di *transito*, poiché non richiede né offre nessun flusso; un nodo con $b_i < 0$ è detto nodo *origine* (o *sorgente*) poiché mette a disposizione del flusso; un nodo con $b_i > 0$ è detto nodo *destinazione* (o *pozzo*) poiché richiede del flusso. Nel seguito assumeremo che il bilancio complessivo della rete sia nullo, cioè che la somma dei b_i sia uguale a zero. Con opportuni accorgimenti ci si può sempre ricondurre a un caso di rete a bilancio complessivo nullo.

Il problema di *flusso di costo minimo* consiste nel determinare il flusso sugli archi della rete in modo che dalle sorgenti parta tutto il flusso a disposizione, alle origini arrivi tutto il flusso richiesto, le capacità degli archi non vengano superate e il costo complessivo del flusso sugli archi sia minimo. Utilizzando le consuete variabili di flusso x_{ij} sugli archi, la formulazione del problema è la seguente:

$$\begin{aligned} \min \quad & \sum_{(i,j)\in A} c_{ij}x_{ij} \\ & -\sum_{(i,j)\in FS(i)} x_{ij} + \sum_{(j,i)\in BS(i)} x_{ji} = b_i \qquad \forall i \in N \\ & 0 \le x_{ij} \le u_{ij} \qquad \forall (i,j) \in A \end{aligned}$$

Se nella formulazione di un problema di flusso di costo minimo risulta che $b_i = 0$ per ciascun nodo i, abbiamo un problema di *circolazione*. In questa classe di problemi la quantità di flusso che circola nella rete non è dettata dai bilanci ma viene indotta dai costi sugli archi. Si noti che per indurre un flusso diverso da zero, il costo di qualche arco dovrà essere negativo.

Osserviamo che il problema dell'albero dei cammini minimi può essere visto come un problema di flusso di costo minimo in cui i costi degli archi sono dati dalle lunghezze, le capacità degli archi sono illimitate e i bilanci ai nodi sono: per la radice dell'albero r abbiamo una offerta di n-1 unità di flusso, quindi $b_r = -(n-1)$, mentre per tutti gli altri nodi i abbiamo una richiesta di una unità di flusso, quindi $b_i = 1$.

Il problema di flusso massimo nella seconda e più generale formulazione proposta precedentemente è un problema di circolazione, infatti tutti i bilanci ai nodi sono zero. Ciò che rende conveniente far circolare flusso rispetto ad una soluzione nulla è la funzione obiettivo. Nel caso del flusso massimo possiamo fissare il costo di tutti gli archi a 0 tranne che per l'arco fittizio di ritorno (t,s) per cui fissiamo il costo a -1.

Esercizio

Nel caso di un problema di flusso di costo minimo in cui la somma dei bilanci sia diversa da 0 la formulazione deve venire modificata. Facendo l'ipotesi di avere un eccesso di offerta (somma dei bilanci negativa) i vincoli di conservazione del flusso dei nodi sorgente devono venire modificati in vincoli di maggiore o uguale, infatti non tutto il flusso offerto riuscirà a partire dalle sorgenti. Volendo ricondursi a un problema in cui tutti i vincoli sono di uguaglianza dovremmo introdurre opportune variabili di scarto. Dare una interpretazione in termini di flusso delle variabili di scarto tenendo presente che nei problemi di flusso le variabili sono legate agli archi.

7.1 Applicazioni: costruzione di una strada e progetto di una rete di comunicazione

Esempio: costruzione di una strada

Consideriamo il problema di ottimizzazione delle operazioni di sbancamento di un terreno ondulato in vista della costruzione di una strada. Una volta fissato l'itinerario è necessario sbancare il terreno in modo da eliminare i dossi e pareggiare gli avvallamenti. L'entità dello spostamento di materiale viene quantificato in numero di camion.

Dopo aver partizionato la strada in porzioni opportune, possiamo rappresentare il problema tramite una rete di flusso. A ogni sezione della strada associamo un nodo i, il cui bilancio è dato dal numero di camion di materiale da spostare. Ovviamente se una sezione è in corrispondenza ad un dosso saremo in presenza di una sorgente (b_i negativo), se invece si tratta di un avvallamento avremo un pozzo (b_i positivo). Ogni nodo è collegato con i nodi relativi alle sezioni adiacenti. Visto che possiamo assumere di non avere vincoli sul numero di camion da spostare da un nodo all'altro, le capacità degli archi sono tutte uguali a infinito. Il costo del trasporto tra due nodi adiacenti viene specificato dal costo c_{ij}. In questo caso il flusso che circola su ogni arco rappresenta il numero di camion che si spostano da un luogo all'altro.

Esempio: progetto di una rete di comunicazione

Un insieme di n terminali devono venire collegati al computer centrale. Per ogni terminale vi sono due possibilità: costruire una linea diretta verso il computer centrale oppure collegare il terminale tramite un concentratore. Sono a disposizione p concentratori; ad ogni concentratore possono venire collegati al più k terminali. La linea che connette il concentratore con il computer centrale è già presente. Il costo di costruzione delle linee per ogni terminale i è dato da c_{i0} per la linea diretta dal terminale al computer centrale, e da c_{ih} per la linea che collega il terminale al concentratore h. Il problema di progettare la rete di comunicazione di costo minimo può essere formulato come un problema di flusso su una opportuna rete. Nella rete abbiamo un nodo 0 che rappresenta il computer centrale, i nodi 1,…, p che rappresentano i concentratori, e i nodi p+1,…, p+n che rappresentano i terminali. I nodi p+1,…, p+n hanno bilancio -1 (offrono una unità di flusso ognuno), i nodi 1,…, p sono nodi di transito, il nodo 0 è un nodo pozzo e richiede esattamente n unità di flusso. Per ogni nodo-terminale i vi sono p+1 archi uscenti: uno diretto verso il nodo 0, di costo c_{i0}, e gli altri diretti verso i concentratori, di costo c_{ih}, h=1,…, p. Abbiamo inoltre un arco che collega ogni concentratore h con il nodo 0, la cui capacità u_{h0} è uguale a k. Le capacità degli altri archi sono poste a infinito. Una soluzione ottima "intera" (ovvero in cui i flussi assumono tutti valore 0 o 1) corrisponde al progetto ottimo della rete di comunicazione. In questo caso il flusso sui singoli archi rappresenta una variabile decisionale: se è uguale a 1 si costruisce la linea corrispondente, se è uguale a 0 no. Si noti che se esiste una soluzione ottima non intera, è sempre possibile trovarne una equivalente intera.

7.2 Algoritmo di eliminazione dei cicli negativi

Come visto per i due problemi precedenti, per dare un algoritmo è di grande aiuto caratterizzare le soluzioni ottime. Consideriamo un flusso ammissibile $\bar{x}$ per il problema. Come abbiamo fatto per il problema di flusso massimo possiamo definire il grafo residuale $G_R(\bar{x})$ a esso associato. Il grafo residuale, esattamente come per il caso precedente, fornisce le informazioni necessarie a valutare le possibili variazioni di flusso. Siccome nel caso di flusso di costo minimo la componente costo è l'elemento essenziale per decidere se una variazione di flusso è vantaggiosa, ad ogni arco del grafo residuale possiamo associare un *costo residuale* g_{ij} che quantifica l'effetto sulla funzione obiettivo di una variazione unitaria del flusso corrispondente all'arco del grafo residuale:

$$g_{ij} = \begin{cases} c_{ij} & \text{se } (i,j) \in A^+(\bar{x}) \\ -c_{ji} & \text{se } (i,j) \in A^-(\bar{x}) \end{cases}$$

Un percorso sul grafo residuale corrisponde a una possibile variazione del flusso $\bar{x}$, secondo le stesse regole viste per il flusso massimo, ovvero una variazione in positivo per le variabili di flusso corrispondenti ad archi di $A^+(\bar{x})$, e una variazione in negativo per le variabili di flusso corrispondenti ad archi di $A^-(\bar{x})$.

Esempio: grafo residuale

Si consideri il problema di flusso di costo minimo rappresentato in fig. 29, in cui abbiamo indicato anche una soluzione ammissibile.

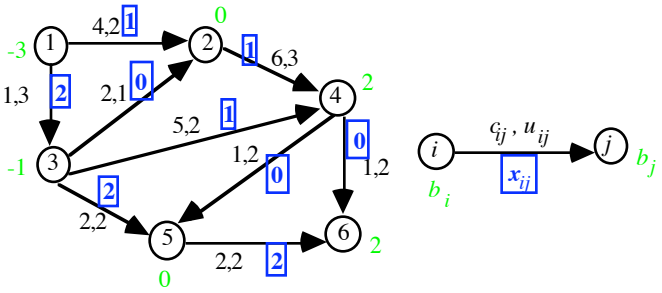

fig. 29: problema di flusso di costo minimo con flusso ammissibile

Il grafo residuale relativo al flusso dato è quello rappresentato in fig. 30.

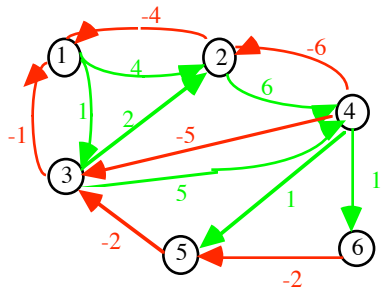

fig. 30: grafo residuale

L'unico modo di modificare la soluzione senza violare i vincoli di conservazione del flusso è quello di variare il flusso su un ciclo (o su un insieme di cicli) di una stessa quantità θ. Infatti una variazione di flusso su un percorso che non si chiude su se stesso porterebbe a una immediata violazione dei vincoli di conservazione di flusso che erano soddisfatti dal flusso originale $\bar{x}$.
Considerato un ciclo C, la massima variazione di flusso attuabile è data da:

$$\theta = \min\{\{u_{ij} - \bar{x}_{ij} : (i,j) \in A^+(\bar{x}) \cap C\}, \{\bar{x}_{ij} : (i,j) \in A^-(\bar{x}) \cap C\}\}.$$

Si noti che, per come è stato costruito il grafo residuale, θ>0. Il flusso viene aggiornato secondo le seguenti regole:

$$x'_{ij} = \begin{cases} \bar{x}_{ij} + \theta & \text{se } (i,j) \in A^+(\bar{x}) \cap C\} \\ \bar{x}_{ij} - \theta & \text{se } (j,i) \in A^-(\bar{x}) \cap C\} \\ \bar{x}_{ij} & \text{altrimenti} \end{cases}$$

La variazione della funzione obiettivo è pari a:

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

Questo significa che vi è un miglioramento nel valore della soluzione solo se il ciclo C ha somma dei costi residuali negativi. Possiamo anche provare il viceversa, ovvero che se non esistono cicli negativi nel grafo residuale allora il flusso non è migliorabile, come ci viene suggerito dal seguente teorema di decomposizione del flusso. Prima di discutere del teorema introduciamo una definizione. Come abbiamo visto in precedenza diciamo che un flusso x è una circolazione se la quantità di flusso entrante in ogni nodo è pari a quella uscente cioè se:

$$-\sum_{(i,j)\in FS(i)} x_{ij} + \sum_{(j,i)\in BS(i)} x_{ji} = 0 \qquad \forall i \in N$$

Una circolazione x è detta *semplice* se gli archi (i,j) in cui $x_{ij}>0$ individuano un ciclo. Si noti che la quantità di flusso sugli archi di una circolazione semplice è costante, a causa della conservazione del flusso.

Teorema di decomposizione del flusso
Si consideri un problema di flusso di costo minimo sul grafo $G=(N,A)$.

$$\begin{aligned} \min & \sum_{(i,j)\in A} c_{ij} x_{ij} \\ & -\sum_{(i,j)\in FS(i)} x_{ij} + \sum_{(j,i)\in BS(i)} x_{ji} = b_i \qquad \forall i \in N \\ & 0 \le x_{ij} \le u_{ij} \qquad \forall (i,j) \in A \end{aligned}$$

e siano dati un flusso ammissibile $\bar{x}$ e un flusso ottimo x^*. Esistono k circolazioni semplici $x^1, \ldots, x^k$ che permettono di trasformare $\bar{x}$ in x^*, cioè tali che $\bar{x} + x^1 + \ldots + x^k = x^*$.

Dim.

Consideriamo $\hat{x} = x^* - \bar{x}$: poiché $\bar{x}$ e x^* sono entrambe ammissibili abbiamo:

$$-\sum_{(i,j)\in FS(i)} \hat{x}_{ij} + \sum_{(j,i)\in BS(i)} \hat{x}_{ji} = 0 \qquad \forall i \in N.$$

Costruiamo un grafo $\hat{G}$ a partire da G introducendo gli archi (i,j) e (j,i) per ogni arco (i,j) di A. Quando otteniamo un $\hat{x}_{ij} < 0$ trasformiamo $\hat{x}$ eliminando il flusso dall'arco (i,j) e riportandolo cambiato di segno sull'arco (j,i), operazione che non influenza la conservazione del flusso. Possiamo verificare facilmente che $\hat{x}$ così trasformato è una circolazione su $\hat{G}$. Ora dobbiamo dimostrare che la circolazione $\hat{x}$ è *decomponibile* in k circolazioni semplici.

Se $x^* = \bar{x}$, $\hat{x} = 0$, quindi $k = 0$, e il teorema vale banalmente.

Altrimenti esiste un arco (i,j) per cui $\hat{x}_{ij} > 0$. Per via della conservazione del flusso esisterà almeno un arco (j,h) per cui $\hat{x}_{jh} > 0$. Ripetendo lo stesso ragionamento in successione, dopo al più n archi raggiungiamo nuovamente un nodo già visitato, chiudendo un ciclo C_1. Sia θ_1 la minima quantità $\hat{x}_{hl}$ sugli archi $(h,l) \in C_1$, definiamo x^1 come segue: $x^1_{hl} = \theta_1$ se $(h,l) \in C_1$ e 0 altrimenti. Ora aggiorniamo $\hat{x}$ andando a sottrarre θ_1 da ogni arco di C_1. Si noti che per come abbiamo determinato θ_1, su almeno un arco (i,j) $\hat{x}_{ij}$ si è azzerato. Possiamo ripetere questo procedimento fino a che in un numero finito k di passi $\hat{x}$ è tutto nullo.

Possiamo quindi concludere che $x^* - \bar{x} = x^1 + \ldots + x^k$.

♦

Abbiamo due importanti e semplici conseguenze del precedente teorema.

Corollario

Il numero k di circolazioni semplici è limitato dal numero m degli archi di G.

Corollario

Il costo residuale dei cicli $C_1, \ldots, C_k$ è minore o uguale a zero.

Pertanto è abbastanza chiaro che una condizione di ottimalità è:

> Il flusso ammissibile $\bar{x}$ è ottimo se e solo se $G_R(\bar{x})$ non contiene cicli la cui somma dei costi residuali è negativa.

A partire dalla condizione di ottimalità enunciata sopra e dalla discussione che l'ha preceduta possiamo fornire un semplice algoritmo di soluzione per il problema del flusso di costo minimo che si basa sul successivo aggiornamento del flusso fino a che il grafo residuale non contiene più cicli negativi. L'algoritmo riceve in input la rete G e un flusso ammissibile. Al suo interno l'algoritmo fa

uso di una procedura che individua i cicli negativi sul grafo residuale. Questa procedura può essere realizzata efficientemente basandosi sul calcolo dei cammini minimi, come visto nel paragrafo 5.10. L'aggiornamento del flusso viene effettuato secondo le regole discusse in precedenza.

```
Procedure Eliminazione_cicli_negativi (G,x);
   begin
      Gr:=Grafo_residuale(x);
      while ∃ ciclo negativo C in Gr do
      begin
         aggiorna_flusso(x,C); aggiorna grafo_residuale(x)
      end
   end.
```

Esempio: una iterazione dell'algoritmo

Osservando il grafo residuale in fig. 30 possiamo individuare vari cicli negativi: 4-2-1-3-4 di costo complessivo 4, 1-3-2-1 di costo -1. Lo schema algoritmico presentato non suggerisce alcun criterio con cui scegliere i cicli. In letteratura sono presenti vari algoritmi che si differenziano in base al criterio di scelta dei cicli. Nel nostro caso scegliamo il primo ciclo. $\theta=\min\{1, 1, 3\text{-}2, 2\text{-}1\}=1$. L'aggiornamento di flusso comporta un aumento di una unità di flusso lungo gli archi di $A^+(\bar{x})$ (1,3) e (3,4), e una diminuzione di una unità per il flusso sugli archi di $A^-(\bar{x})$ (2,4) e (1,2).

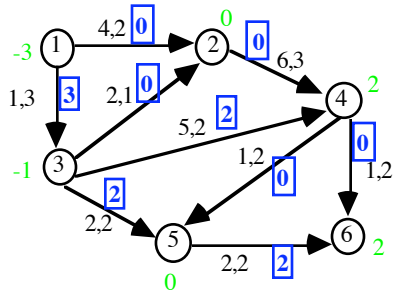

fig. 31: flusso aggiornato

Esercizio
Verificare che il flusso in fig. 31 è ottimo. Provare a eseguire l'algoritmo di eliminazione di cicli negativi considerando il ciclo 1-3-2-1.

7.3 Flusso ammissibile di partenza

Per poter applicare l'algoritmo abbiamo bisogno di un metodo che ci permetta di generare un flusso ammissibile iniziale, ammesso che questo esista. Il calcolo di un flusso ammissibile può venire realizzato in tempo polinomiale per mezzo di un algoritmo di flusso massimo su un opportuno grafo. Consideriamo un grafo *ausiliario* $G'=(N\cup\{s,t\},A')$ in cui A oltre a tutti gli archi di A include anche gli archi che collegano la sorgente s con i nodi i con un bilancio negativo (sorgenti di flusso in G) e gli archi che collegano i nodi con bilancio positivo (destinazioni di flusso) con il pozzo t. Gli archi del tipo (s,i) hanno capacità $u_{si}=-b_i$, mentre gli archi (i,t) hanno capacità $u_{it}=b_i$. Tutti gli altri

archi hanno capacità immutata rispetto al grafo originario G. I bilanci dei nodi vengono a questo punto azzerati. Un flusso ammissibile (se esiste) può venire trovato risolvendo un problema di flusso massimo da s a t. Se il valore del flusso massimo è uguale a $-\sum_{i:b_i<0} b_i$, allora esiste un flusso ammissibile ed è dato proprio dal flusso massimo trovato. Si noti che in questo caso il taglio trovato dall'algoritmo di Edmonds e Karp è $N_s=\{s\}$ $N_t=N\cup\{t\}$; in caso contrario il taglio include in N_s anche nodi di N.

Esempio: grafo ausiliario

Consideriamo il problema di flusso di costo minimo in fig. 29. Dobbiamo aggiungere i nodi s e t, gli archi da s ai nodi di offerta 1 e 3 con capacità 3 e 1 rispettivamente, e gli archi dai nodi di domanda 4 e 6 entrambi con capacità 2. Se risolvendo il problema di flusso massimo otteniamo un flusso 4, il flusso sugli archi interni è ammissibile.

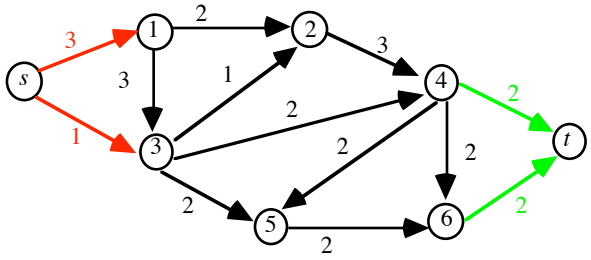

fig. 32: grafo ausiliario per il problema in fig. 29

7.4 Analisi di complessità

Analizziamo ora la complessità dell'algoritmo di eliminazione di cicli negativi realizzato senza adottare alcuna regola specifica per la scelta del ciclo negativo da eliminare, in caso di presenza di più alternative. Facciamo l'ipotesi che tutti i costi e le capacità della rete G siano interi non negativi, chiamiamo C_{max} e U_{max} rispettivamente il costo massimo e la capacità massima degli archi del grafo. Siccome la scelta della soluzione ammissibile iniziale non è affatto dettata da criteri di costo, inizialmente, nel caso pessimo, il costo della soluzione fornita è al massimo dato da $mC_{max}U_{max}$. Facendo considerazioni del tutto analoghe possiamo anche ipotizzare che la soluzione ottima nel caso estremo abbia costo 0. Osservando che ad ogni iterazione dell'algoritmo il flusso varia di almeno una unità e che questo induce una diminuzione nel valore della funzione obiettivo di almeno una unità, dovremo fare nel caso pessimo $O(mC_{max}U_{max})$ iterazioni. Sapendo che i cicli negativi possono venir riconosciuti in $O(nm)$ la complessità dell'algoritmo è $O(nm^2C_{max}U_{max})$.

Si può dimostrare che per una particolare scelta del ciclo negativo, l'algoritmo è polinomiale.

Esercizio

L'algoritmo dei cammini aumentanti per il problema di flusso massimo può essere interpretato come un algoritmo di eliminazione dei cicli negativi. Discuterne i particolari.

**7.5 Algoritmo sequenza di cammini minimi*

Consideriamo ora un altro algoritmo per il problema del flusso di costo minimo che funziona in modo incrementale, cioè, invece che partire da una soluzione ammissibile, tenta di costruire direttamente la soluzione ottima inviando il flusso dai nodi sorgente ai nodi destinazione. L'algoritmo si basa su di una diversa condizione di ottimalità, ma comunque equivalente a quella vista in precedenza. Richiamiamo il concetto di costo ridotto, già visto nel caso dell'albero dei cammini minimi. Dato un potenziale π associato ai nodi del grafo, il costo ridotto di un arco $(i,j)\in A$ è definito come segue:

$$\bar{c}_{ij} = c_{ij} + \pi_i - \pi_j.$$

Dato un cammino tra il nodo x e il nodo y P_{xy}, la somma dei costi ridotti è data da:

$$\sum_{(i,j)\in P_{xy}} \bar{c}_{ij} = \sum_{(i,j)\in P_{xy}} c_{ij} + \pi_x - \pi_y.$$

Quindi la somma dei costi ridotti di un cammino è uguale alla somma dei costi a meno di una costante che dipende solamente dagli estremi del cammino e non dagli archi che lo formano.
Siccome in un ciclo C gli estremi coincidono, la somma dei costi ridotti di un ciclo è uguale alla somma dei costi:

$$\sum_{(i,j)\in C} \bar{c}_{ij} = \sum_{(i,j)\in C} c_{ij}.$$

Questo implica che, se un ciclo è negativo rispetto ai costi ridotti, esso è negativo anche rispetto ai costi originali.
Una condizione di ottimalità equivalente a quella della non esistenza di cicli negativi sul grafo residuale è la seguente:

> Il flusso ammissibile $\bar{x}$ è ottimo se e solo se esiste un potenziale π tale che $\bar{c}_{ij} \geq 0$ per ogni arco $(i,j)\in A(\bar{x})$.

Proviamo che le due condizioni di ottimalità sono equivalenti. Supponiamo che valga la condizione sui costi ridotti:

$$\bar{c}_{ij} \geq 0 \qquad \forall (i,j)\in A(\bar{x}),$$

quindi per ogni ciclo C in $G_R(\bar{x})$

$$\sum_{(i,j)\in C} \bar{c}_{ij} \geq 0$$

ma dato che la somma dei costi ridotti è uguale alla somma dei costi originali, abbiamo

$$\sum_{(i,j)\in C} c_{ij} \geq 0.$$

Supponiamo invece che $G_R(\bar{x})$ non contenga cicli negativi; dato che il grafo non contiene cicli negativi possiamo calcolare l'albero dei cammini minimi prendendo come radice un qualsiasi nodo. L'algoritmo dei cammini minimi termina e fornisce delle etichette ottime $d[i]$. Ponendo $\pi_i=d[i]$ si ha che per ogni arco di $A(\bar{x})$ valgono le condizioni di Bellman:

$$c_{ij} + \pi_i - \pi_j = \bar{c}_{ij} \geq 0 \qquad \forall (i,j)\in A(\bar{x}).$$

Ora possiamo fornire un algoritmo basandoci sulle nuove condizioni di ottimo trovate. Nell'algoritmo che proponiamo partiamo da una soluzione x non ammissibile per cui valgono però le condizioni di ottimo; a ogni iterazione manteniamo valide le condizioni di ottimalità ma cerchiamo di ridurre la non ammissibilità della soluzione x.

Una soluzione x è detta *pseudoflusso* se rispetta i vincoli di capacità ma non necessariamente i vincoli di conservazione di flusso nei nodi:

$$-\sum_{(i,j)\in FS(i)} x_{ij} + \sum_{(j,i)\in BS(i)} x_{ji} - b_i = \bar{g}_i \qquad \forall\, i\in N.$$

La quantità $\bar{g}_i$ è detta *eccesso* del nodo i. Se $\bar{g}_i = 0$ il nodo è *bilanciato*, se $\bar{g}_i<0$ il nodo è in *difetto* di flusso e se $\bar{g}_i > 0$ il nodo è in eccesso di flusso.

L'obiettivo principale dell'algoritmo che vedremo sarà quello di inviare flusso dai nodi in eccesso ai nodi in difetto. Ovviamente questo dovrà essere fatto mantenendo le condizioni di ottimalità sui costi ridotti del grafo residuale. Il punto che risulta essere critico è il seguente. Se aumentiamo il flusso su un arco (i,j) di una quantità maggiore di zero, all'iterazione successiva il grafo residuale conterrà l'arco (j,i), il cui costo ridotto è:

$$\bar{c}_{ji} = c_{ji} + \pi_j - \pi_i = - \bar{c}_{ij}.$$

Analogo discorso vale per le variazioni in negativo. Pertanto è evidente che le operazioni di variazione del flusso sono particolarmente delicate in quanto rischiano di far venir meno le condizioni di ottimalità che l'algoritmo deve mantenere invarianti durante tutta la sua esecuzione. Quindi prima di eseguire delle variazioni del flusso sugli archi bisogna mettersi in condizione di non creare archi di costo ridotto negativo; questo significa che, prima di eseguire una variazione di flusso bisogna premurarsi di aggiornare i potenziali in modo che i costi ridotti degli archi su cui aumentiamo o diminuiamo il flusso siano nulli. Di fondamentale importanza per il funzionamento dell'algoritmo è la seguente osservazione.

Dato uno pseudoflusso x e un potenziale π che rispetta le condizioni di ottimo sui costi ridotti

$$\bar{c}_{ij} \geq 0 \qquad \forall (i,j)\in A(\bar{x}),$$

se consideriamo le etichette ottime $d[i]$ di un albero dei cammini minimi di radice s sul grafo residuale $G_R(x)$, dove s è un qualsiasi nodo di eccesso, e modifichiamo il potenziale come segue:

$$\pi'_i = \pi_i + d[i] \qquad \forall i \in N,$$

le condizioni di ottimo sui costi ridotti continuano a valere.
Infatti i nuovi costi ridotti sono dati da:

$$c'_{ij} = c_{ij} + \pi'_i - \pi'_j,$$

per l'ottimalità delle etichette vale che

$$d[j] \le d[i] + \bar{c}_{ij} = d[i] + c_{ij} + \pi_i - \pi_j,$$

quindi

$$c_{ij} + \pi_i + d[i] - \pi_j - d[j] \ge 0$$

cioè $c'_{ij} \ge 0$.
Si noti che, con l'aggiornamento del potenziale proposto, tutti gli archi dell'albero dei cammini minimi avranno costo ridotto esattamente uguale a zero quindi sono archi su cui si può variare il flusso. Basandoci sull'ipotesi fondamentale (ma non restrittiva) che tutti i costi siano maggiori o uguali a zero, l'algoritmo è il seguente:

```
Procedure Sequenza_di_cammini_minimi (G,x,π);
  begin
    D=¢; O=¢;
    for each (i,j)∈A do x_ij:=0;
    for each i∈N do
    begin π_i:=0; g_i:=-b_i; if g_i<0 then D:=D∪{i} else if g_i>0 O:=O∪{i} end;
    while O≠¢ do
    begin
      select i from O;
      SPT(i,P,d,G(x));
      for each j∈N do π_j:=π_j+d[j];
      let P_ik il cammino minimo da i a k con k∈D;
      θ:=min{g_i, -g_k, min{r_jh, (i,h)∈P_ik}};
      aumenta_flusso(θ,P_ik); update (O,D);
    end
  end.
```

L'algoritmo, dopo la fase di inizializzazione in cui fissa lo pseudoflusso iniziale e il potenziale a zero, cerca un cammino minimo da un nodo di eccesso a un nodo con difetto. Prima di spedire la massima quantità di flusso possibile, vengono aggiornati i potenziali in modo da rendere tutti gli archi del cammino trovato a costo ridotto nullo. La massima quantità di flusso inviabile è data dal minimo tra l'eccesso, il difetto e la capacità residua del cammino, calcolata come nel caso

dell'algoritmo di eliminazione dei cicli negativi o l'algoritmo dei cammini aumentanti per il flusso massimo. Si noti che, dopo l'aggiornamento dei potenziali, è possibile spedire flusso su più di un cammino. Gli algoritmi in questa classe si differenziano proprio per questi aspetti. L'algoritmo termina quando non vi sono più nodi con eccesso (o difetto) oppure quando i nodi con difetto non sono più raggiungibili dai nodi con eccesso.

Analizziamo ora la complessità dell'algoritmo. Chiamiamo

$$B = \sum_{i \in O} \bar{g}_i$$

la quantità di eccesso da spedire dai nodi origine. A ogni iterazione, supponendo di avere capacità intere, almeno una unità di flusso viene spedita, quindi la quantità iniziale di eccesso viene diminuita almeno di una unità. Pertanto avremo non più di B iterazioni. Ad ogni iterazione l'operazione più costosa è data dal calcolo dell'albero dei cammini minimi in un grafo con costi maggiori o uguali di zero. Quindi la complessità è data da $O(B\ S(n,m))$, dove $S(n,m)$ è la complessità dell'algoritmo dei cammini minimi utilizzato.

Esercizio
In caso la rete contenesse archi con costi negativi l'algoritmo dei cammini minimi successivi non è applicabile direttamente poiché lo pseudoflusso nullo e il potenziale nullo non rispettano le condizioni di ottimo. Come modificare la soluzione di partenza per garantire che valgano le condizioni di ottimo?

7.6 Assegnamento di costo minimo

Dato un grafo bipartito $G=(S,T,A)$ completo (cioè $A=S\times T$), con $|S|=|T|=n$ e pesi c_{ij} sugli archi $(i,j)\in A$, il problema di assegnamento di costo minimo consiste nel trovare un sottoinsieme di archi M tale che su ogni nodo incida esattamente un arco di M e che la somma dei pesi degli archi di M sia minima. Utilizzando delle variabili logiche x_{ij} associate agli archi, come nel caso dell'accoppiamento di cardinalità massima, il problema può venire formulato come segue:

$$\begin{aligned}
\min \sum_{(i,j)\in A} & c_{ij}\, x_{ij} \\
& \sum_{j\in T} x_{ij} = 1 && \forall i \in S \\
& \sum_{i\in S} x_{ij} = 1 && \forall j \in T \\
& x_{ij} \in \{0,1\} && \forall (i,j) \in A.
\end{aligned}$$

Il problema può venire interpretato in termini di flusso di costo minimo. Ad ogni nodo di S si associa una offerta di una unità di flusso (b_i=-1, $\forall i\in S$), e ad ogni nodo di T si associa una richiesta di una unità di flusso (b_j=1, $\forall j\in S$). Il costo di ciascun arco (i,j) è esattamente il peso c_{ij}, mentre la capacità è illimitata.

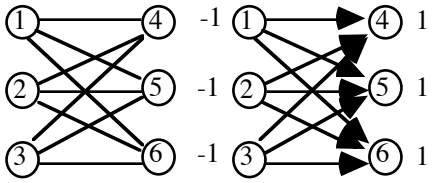

fig. 33: trasformazione del grafo bipartito nella rete di flusso

È evidente che, per le stesse motivazioni discusse nel caso del problema di accoppiamento di cardinalità massima, i due problemi sono equivalenti. Quindi risolvendo all'ottimo il problema di flusso di costo minimo otteniamo la soluzione ottima del problema di assegnamento di costo minimo. L'algoritmo dei cammini minimi successivi può trarre particolare beneficio dalle caratteristiche del problema. Infatti ad ogni iterazione dell'algoritmo viene individuato un cammino tra un nodo di S non ancora assegnato (detto *origine esposta*) e un nodo di T non ancora assegnato (detto *destinazione esposta*). Lungo il cammino trovato, che non è necessariamente composto da un solo arco, possiamo inviare una sola unità di flusso. Questo significa che a ogni iterazione il numero di origini esposte (o destinazioni esposte) calerà di una unità. Il numero di iterazioni sarà pertanto di n, e la complessità dell'algoritmo è $O(n^3)$, utilizzando per il calcolo dell'albero dei cammini minimi l'algoritmo di Dijkstra implementato con una lista non ordinata.

Esercizio

Risolvere il problema in fig. 33 con i pesi dati nella Tabella 4

c_{ij}	**4**	**5**	**6**
1	15	2	5
2	8	3	8
3	5	4	2

Tabella 4

8 Problemi di flusso multi-prodotto

I problemi di flusso che abbiamo visto fino ad ora sono caratterizzati dal fatto che il flusso circolante nella rete è omogeneo. Molte applicazioni pratiche però non possono basarsi su questa ipotesi restrittiva e devono considerare flussi distinti che concorrono all'utilizzo delle risorse comuni. Questo avviene tipicamente nel caso di distribuzione di merce. Consideriamo l'esempio riportato in fig. 34.

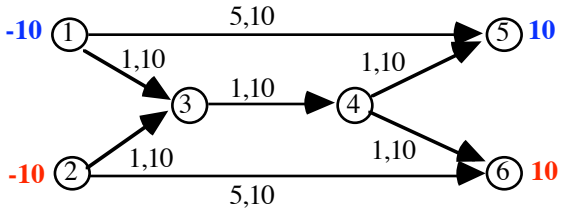

fig. 34: esempio di flusso multi-prodotto; sugli archi sono indicati i costi e le capacità

Il nodo 1 offre 10 unità del prodotto A, il nodo 2 offre 10 unità del prodotto B, i nodi 5 e 6 richiedono rispettivamente 10 unità del prodotto A e 10 unità del prodotto B. I flussi dei due

prodotti vanno quindi distinti dato che non possiamo fare arrivare al nodo 5 il prodotto B o far arrivare al nodo 6 il prodotto A. Questo però non significa risolvere indipendentemente i problemi di flusso per il prodotto A e per il prodotto B e ricomporre la soluzione sovrapponendo i due flussi poiché essi concorrono nell'utilizzo delle risorse della rete quali le capacità degli archi. Denotando con K l'insieme dei prodotti, il problema può venire formulato come segue:

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

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

$$0 \le \sum_{k\in K} x_{ij}^k \le u_{ij} \qquad \forall (i,j)\in A$$

dove le variabili x_{ij}^k indicano il flusso del bene k sull'arco (i,j). Si noti che, rispetto a un problema di flusso di costo minimo, nel problema di flusso multi-prodotto abbiamo replicato i vincoli di conservazione del flusso per ogni prodotto k, mentre il ruolo delle variabili di flusso nella funzione obiettivo e nei vincoli di capacità ora è giocato dalla somma su tutti i prodotti dei flussi sull'arco. La struttura del problema impone quindi di trovare contemporaneamente i flussi per tutti i prodotti. La soluzione di questo problema pur basandosi sulle caratteristiche dei problemi di flusso, fa ricorso ai metodi della programmazione lineare e della programmazione matematica più in generale. Approfondiremo l'argomento nei prossimi capitoli.

Vediamo ora una particolare classe di modelli di flusso multi-prodotto utilizzata nel risolvere problemi di instradamento o di progetto di reti di telecomunicazione. Consideriamo un piccolo esempio di una piccola rete Intranet con 5 nodi e 16 collegamenti unidirezionali schematizzata in fig. 35; per semplicità nella figura sono stati indicati degli archi con il doppio orientamento a cui corrispondono nel modello due archi con opposte direzioni.

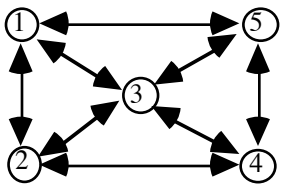

fig. 35: lo schema di una piccola rete Intranet

Oltre ai costi di spedizione c_{ij} per unità di informazione e le capacità massime u_{ij} su ciascun collegamento (i,j), i dati del problema includono le quantità di informazione d_{st} da inviare tra ogni coppia di nodi (s,t). Quest'ultima informazione è riassunta nella cosiddetta *matrice origine/destinazione*. È chiaro che siamo in presenza di un problema di flusso, ma è anche chiaro che il flusso circolante nella rete non è omogeneo, infatti il flusso di ogni coppia origine/destinazione (s,t) deve venire distinto da tutti gli altri. Dobbiamo quindi ricorrere a una formulazione di flusso multi-prodotto in cui ogni prodotto k corrisponde a una coppia (s_k,t_k) tale per

cui $d_{s_k t_k}$ è diverso da 0. Il modello fornito sopra, che introduce una variabile per ogni arco e coppia origine/destinazione, formula correttamente il problema di instradamento di costo minimo delle informazioni. Questa formulazione presenta però un piccolo difetto: il numero delle variabili introdotte, specie se la matrice origine/destinazione è molto densa, è molto elevato. Per esempio in fig. 35, se la matrice origine/destinazione contenesse elementi diversi da zero per tutte le coppie di nodi, avremmo 16×20=320 variabili. Volendo utilizzare la versione di Xpress che limita il numero di variabili a 300, il modello dato sopra sarebbe impossibile da risolvere.

Possiamo cercare di formulare il problema diversamente. L'idea del modello che proponiamo è quella di specificare il flusso non più su singoli archi, ma su interi cammini da ogni possibile origine a ogni possibile destinazione. Indichiamo con x_p la variabile di flusso lungo il cammino p. Indichiamo con P l'insieme di tutti i cammini utili alla soluzione del problema; per ogni coppia origine/destinazione (s_k,t_k) definiamo un sottoinsieme $P_k \subset P$ contenente tutti i cammini che vanno da s_k a t_k. Ovviamente i sottoinsiemi P_k per ogni k in K costituiscono una partizione dell'insieme di tutti i cammini P. Il costo di un cammino p è dato dalla somma dei costi degli archi che lo costituiscono:

$$c_p = \sum_{(i,j)\in p} c_{ij}.$$

La formulazione del problema, che viene detta *formulazione per cammini*, diventa:

$$\begin{aligned}
\min \sum_{k\in K} \sum_{p\in P_k} c_p x_p & \\
\sum_{p\in P_k} x_p = d_{s_k t_k} & \qquad \forall k\in K \\
\sum_{p\in P:(i,j)\in p} x_p \le u_{ij} & \qquad \forall (i,j)\in A \\
x_p \ge 0 & \qquad \forall p\in P
\end{aligned}$$

Il primo insieme di vincoli corrisponde ai vincoli di conservazione del flusso della prima formulazione. Il secondo insieme di vincoli riguarda la capacità degli archi: il flusso su un arco (i,j) è dato dalla somma dei flussi su tutti i cammini contenenti l'arco (i,j).

L'obiezione che si può muovere alla formulazione per cammini è che il difetto del numero di variabili non è stato risolto, anzi, il fatto di aver introdotto una variabile per ogni cammino potrebbe averlo aggravato, in quanto il numero di cammini che collegano due nodi è esponenziale nella dimensione del grafo. Vedremo però che non è necessario considerare tutti i cammini possibili tra ogni coppia di nodi ma solo un sottoinsieme, e tale sottoinsieme può venire generato dinamicamente durante la soluzione del problema tutte le volte che viene individuato qualche cammino particolarmente vantaggioso dal punto di vista della funzione obiettivo. Per arrivare a mettere a punto una simile tecnica abbiamo bisogno degli strumenti della programmazione lineare.