10. Partizionamento ottimo di grafi in componenti connesse

Marco Liverani

Università degli Studi Roma Tre
Dipartimento di Matematica e Fisica
Corso di Laurea in Matematica
E-mail liverani@mat.uniroma3.it

Maggio 2014

Indice

1 Introduzione **3**

2 Problemi di partizionamento di grafi in componenti connesse **4**

3 Alcune classi di problemi di partizione **5**
3.1 Problemi di equipartizione 6
3.2 Problemi di clustering 9
3.2.1 Massima omogeneità all'interno delle componenti 9
3.2.2 Massima distanza tra le componenti 9
3.3 Taglio della partizione 11
3.4 Partizionamento continuo 12

4 Applicazioni **12**
4.1 Elaborazione delle immagini digitali 13
4.2 Progettazione di circuiti VLSI 14
4.3 Progettazione di una rete di calcolatori 15
4.4 Proiezione dei risultati elettorali 15

5 Complessità computazionale **16**

6 Algoritmi **16**
6.1 Programmazione lineare intera 18
6.1.1 Indici 18
6.1.2 Variabili 19
6.1.3 Espressione dei vincoli 19
6.2 Un algoritmo di shifting su alberi 20
6.3 Un algoritmo di shifting su cammini 23

Riferimenti bibliografici **29**

1 Introduzione

Nelle pagine seguenti viene presentato un problema di ottimizzazione combinatoria con vaste ricadute applicative in ambiti eterogenei; anche per questo motivo i problemi affrontati nel seguito assumono una rilevanza particolare nel contesto dei problemi di ottimizzazione su grafi.

È noto che il problema SUBSET-SUM, in cui, dato un insieme S di naturali e una costante $k > 0$, si richiede di individuare un sottoinsieme $S' \subseteq S$ tale che $\sum_{s \in S'} s = k$, è un problema NP-completo (lo si può dimostrare costruendo un algoritmo di riduzione di complessità polinomiale in grado di trasformare ogni istanza di 3-CNF-SAT ad una istanza di SUBSET-SUM). Anche il problema SET-PARTITION, in cui, dato un insieme S di naturali, si richiede di costruire una bipartizione $\{S_1, S_2\}$ di S tale che $S_1 \cup S_2 = S$, $S_1 \cap S_2 = \emptyset$ e $\sum_{s \in S_1} s = \sum_{s \in S_2} s = k$, o la sua generalizzazione da 2 a p componenti sono problemi NP-completi.

Quando l'insieme da partizionare in un certo numero di componenti è soggetto ad ulteriori vincoli che legano fra loro alcuni elementi dell'insieme, lo spazio delle soluzioni ammissibili si riduce e gli stessi vincoli consentono di "guidare" un algoritmo in grado di ricavare in modo costruttivo una soluzione ottima del problema, permettendo di ottenere, in alcuni casi, algoritmi di complessità polinomiale.

Se dunque l'insieme da suddividere in un certo numero di componenti è l'insieme dei vertici di un grafo e se viene imposto un vincolo di connessione delle componenti della partizione, allora il problema assume spesso le caratteristiche di un problema trattabile e risolubile in tempo polinomiale mediante un algoritmo di bassa complessità.

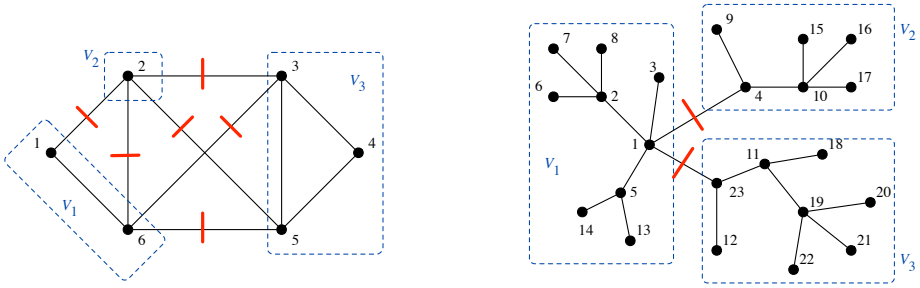
Nelle pagine seguenti vengono proposti alcuni problemi di partizionamento di grafi in componenti connesse: dato un grafo $G = (V, E)$, si richiede di suddividere l'insieme dei vertici $V(G)$ in un numero prefissato p, $0 < p \leq n$, di componenti $V_1, V_2, \ldots, V_p$, in modo tale che le componenti stesse costituiscano una partizione dell'insieme V e che il sottografo $G[V_i]$, indotto da ciascuna componente su G, sia connesso.

Quando un problema di partizionamento di un grafo viene formulato in termini di ottimizzazione, viene definita una funzione obiettivo calcolata su quantità associate ai vertici del grafo (es.: un peso associato a ciascun vertice, un parametro associato a ciascuna coppia di vertici del grafo, indicativo della "distanza" fra due vertici, ecc.) che deve essere massimizzata o minimizzata.

Sotto queste condizioni il problema generale, ossia quando il grafo non possiede delle caratteristiche particolari o dei vincoli sulla sua topologia, risulta NP-completo. Tuttavia, restringendo il problema a specifiche classi di grafi (es.: gli alberi, i cammini, i grafi a stella o a griglia, ecc.) è possibile costruire algoritmi in grado di produrre la soluzione esatta per determinate funzioni obiettivo, in tempo polinomiale.

Le applicazioni sono numerose e vanno dallo studio dei sistemi elettorali, all'analisi dei gruppi. Sono numerosi i problemi in ambito ingegneristico che possono essere ricondotti ad un problema di partizionamento vincolato: si va dalla progettazione di circuiti elettronici VLSI, alla definizione ottimale di una rete di telecomunicazione o alla ottimizzazione del contrasto in una immagine grafica digitalizzata.

Nelle pagine seguenti sono descritti con maggiore dettaglio alcuni dei principali problemi di partizionamento, distinti sulla base della funzione obiettivo che si intende ottimizzare; vengono introdotte anche le principali tecniche algoritmiche utilizzate in questo contesto, ed infine sono descritte alcune delle applicazioni che possono trarre beneficio dalla soluzione di questo genere di problemi.



(a) (b)

Figura 1: Una p-partizione in $p = 3$ componenti connesse di un grafo generico (a) e di un albero (b). Sugli spigoli sono evidenziati i "tagli" che devono essere effettuati per ottenere la p-partizione

2 Problemi di partizionamento di grafi in componenti connesse

Dato un grafo $G = (V,E)$ e un sottoinsieme $V' \subseteq V$ dell'insieme dei vertici, si definisce il **sottografo indotto** di G dall'insieme V', indicato con $G[V'] = (V',E')$, ponendo $E' = \{(u,v) : (u,v) \in E(G) \text{ e } u,v \in V'\}$. In altri termini, l'insieme degli spigoli del sottografo indotto è un sottoinsieme di E, ottenuto come restrizione di E ai soli spigoli i cui estremi appartengono a V'.

Sia $\pi = \{V_1, V_2, \ldots, V_p\}$ una collezione di sottoinsiemi di V; π è una **partizione** di V se $\cup_{k=1,\ldots,p} V_k = V$ e $V_k \cap V_h = \emptyset$ per ogni $h \neq k$, $h,k = 1,2,\ldots,p$.

Una **partizione di un grafo** $G = (V,E)$ è una collezione di p sottografi $G_1, G_2, \ldots, G_p$ ottenuti come sottografi indotti degli elementi $V_1, V_2, \ldots, V_p$ di una partizione di $V(G)$. Se $\pi = \{V_1, \ldots, V_p\}$ è la partizione di $V(G)$, per estensione indicheremo con π anche la partizione del grafo G.

La partizione π di un grafo G è **connessa** se ciascun sottografo della partizione è un sottografo connesso di G.

Diremo che π è una **p-partizione** di G se è costituita esattamente da p componenti non vuote: $V_k \neq \emptyset$ per ogni $k = 1,2,\ldots,p$. Naturalmente p è un intero tale che $0 < p \leq |V|$. Nel seguito indicheremo con Π_p l'insieme di tutte le p-partizioni connesse di un grafo G.

È utile osservare che per produrre una p-partizione in componenti connesse di un *albero* $T = (V,E)$, è necessario e sufficiente rimuovere esattamente $p-1$ spigoli. Infatti, dal momento che in un albero il cammino semplice che collega due vertici u e v è unico, è sufficiente rimuovere un solo spigolo per suddividere in due componenti il grafo stesso. Naturalmente la stessa osservazione si applica quando il grafo da partizionare è un cammino (un caso speciale di albero), mentre non può applicarsi al caso generale di un grafo, dove il numero di spigoli che devono essere rimossi per produrre una scomposizione del grafo in due componenti prive di connessioni reciproche, dipende dalla topologia del grafo stesso.

Proposizione 1. *Il numero di p-partizioni di un albero $T = (V,E)$ con n vertici è $\binom{n-1}{p-1}$.*

Dimostrazione. Per ottenere una partizione di T in p componenti connesse (sottoalberi) è necessario e sufficiente rimuovere esattamente $p-1$ spigoli o, in altri termini, assegnare $p-1$ "tagli" ad altrettanti spigoli dell'albero.

Consideriamo un vertice $r \in V(T)$ di grado 1 di T (una foglia di T) e, assumendo r come radice, imponiamo l'orientazione naturale agli spigoli di T, ottenendo così un albero con radice in r e spigoli orientati dalla radice verso le foglie. Senza perdita di generalità possiamo assumere che la radice r sia il vertice $v_1 \in V(T)$.

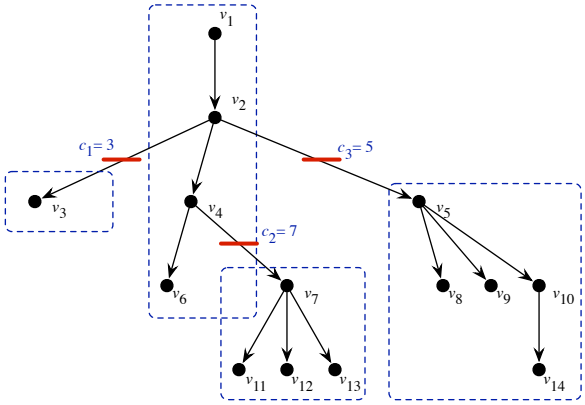


Figura 2: Una p-partizione in $p = 4$ componenti (sottoalberi) di un albero orientato con radice in v_1, con $p-1$ tagli c_1, c_2, c_3 assegnati agli spigoli

Ogni p-partizione π_p di T può essere associata quindi ad una sequenza di $p-1$ interi $(c_1, \ldots, c_{p-1})$, in modo tale che $c_k = i$ se il taglio k-esimo è posto sopra al vertice v_i, ossia sull'unico spigolo entrante nel vertice v_i. Le componenti della partizione sono così definite come sottoalberi che hanno la radice in r o in uno dei vertici v_i tale che $c_k = i$ per qualche $k = 1, \ldots, p-1$, e che sono delimitati dal basso dagli altri tagli (Figura 2).

Dunque è possibile associare ad ogni p-partizione π_p di T una sequenza di interi $(c_1, c_2, \ldots, c_{p-1})$ tale che $c_k > 1$, $c_k \neq c_h$ se $h \neq h$, $c_k \leq n$. Quindi ad ogni p-partizione di T corrisponde una combinazione di $p-1$ elementi dell'insieme $\{2, \ldots, n\}$; inoltre ad ogni combinazione di $p-1$ elementi dell'insieme $\{2, \ldots, n\}$ corrisponde una p-partizione di T. Dunque le p-partizioni di T sono tante quante le combinazioni di $p-1$ elementi di $\{2, \ldots, n\}$, ossia, esattamente $\binom{n-1}{p-1}$. □

3 Alcune classi di problemi di partizione

I problemi di ottimizzazione combinatoria il cui obiettivo è quello di produrre una p-partizione ottima di un grafo in componenti connesse, si suddividono in due famiglie principali: i problemi di *equipartizione* e i problemi di *clustering*. Nelle pagine seguenti definiremo un certo numero di problemi di ottimizzazione specifici per ciascuna delle due famiglie.

Dato un grafo $G = (V, E)$ definiamo una funzione **peso** $w : V(G) \to \mathbb{R}^+$ che associa a ciascun vertice $v \in V$ del grafo un peso $w(v) > 0$. È possibile definire anche il **peso di una componente** V_k di una p-partizione π di G ponendo $W(V_k) = \sum_{v \in V_k} w(v)$. Nelle pagine seguenti indicheremo con μ il **peso medio di una componente** della p-partizione π del grafo G, definita ponendo

$$\mu = \frac{1}{p} \sum_{k=1}^{p} W(V_k) = \frac{1}{p} \sum_{v \in V(G)} w(v) \tag{1}$$

Nei problemi di **equipartizione** di un grafo in p componenti connesse, si cerca di produrre una partizione in cui il peso di ciascuna componente sia il più vicino possibile al peso delle altre componenti. L'obiettivo ottimo sarebbe quello di produrre una p-partizione del grafo in cui $W(V_k) = \mu$ per ogni $k = 1, \ldots, p$; questo obiettivo però spesso non è raggiungibile, dal momento che si deve tenere conto

del vincolo di connessione delle componenti e del fatto che il peso di ciascuna componente è calcolato in modo discreto, sommando i pesi degli elementi che ne fanno parte, senza poter suddividere il peso di uno stesso vertice fra più componenti. Dunque il problema di ottimizzazione combinatoria è non banale ed è finalizzato a produrre una partizione *ottimale*, in cui il peso dei vertici sia distribuito equamente fra tutte le p componenti.

Dato un grafo $G = (V,E)$ è possibile definire una matrice quadrata $D = (d_{ij})$ di ordine $n = |V|$, detta **matrice di dissimilarità**, che, ad ogni coppia di vertici $v_i, v_j \in V(G)$, associa un indice d_{ij} che rappresenta la "distanza", o il grado di "diversità" o, al contrario, un indice di "somiglianza" o di "affinità" tra i due vertici (ad esempio il "grado di amicizia" reciproco tra gli studenti di uno stesso corso di laurea). Ferma restando la diversa interpretazione che può essere data al valore degli elementi della matrice D, denomineremo d'ora in avanti ciascun elemento d_{ij} come **indice di dissimilarità** tra i vertici v_i e v_j.

La matrice D deve essere definita in modo tale da rispettare i seguenti vincoli:

$$\begin{aligned} d_{ij} &\geq 0 \text{ per ogni } i,j = 1,2,\ldots,n \\ d_{ij} &= d_{ji} \text{ per ogni } i,j = 1,2,\ldots,n \\ d_{ii} &= 0 \text{ per ogni } i = 1,2,\ldots,n \end{aligned}$$

La matrice D è quindi quadrata, simmetrica e con elementi tutti nulli sulla diagonale principale.

Dato un grafo G e una matrice di dissimilarità D associata alle coppie di vertici di G è possibile definire un problema di ottimizzazione il cui obiettivo sia quello di costruire una p-partizione di G minimizzando la differenza (la *dissimilarità*) fra gli elementi di una stessa componente, oppure, massimizzando la differenza tra gli elementi di componenti diverse. Questi tipi di problemi di ottimizzazione costituiscono la famiglia dei problemi di **clustering** su grafi.

Il problema del p-partizionamento di un grafo $G = (V,E)$ viene formulato come un problema di ottimizzazione combinatoria chiedendo di massimizzare o minimizzare il valore di una funzione obiettivo $f(\pi_p)$, $f : \Pi_p(G) \to \mathbb{R}$, calcolata in base ai pesi $W(V_k)$ assegnati alle componenti della partizione del grafo o agli indici di dissimilarità d_{ij} definiti tra le coppie di vertici di G; in altre parole si chiede di trovare una partizione

$$\pi_p^* \in \Pi_p(G) \text{ tale che } f(\pi_p^*) = \min_{\pi_p \in \Pi_p(G)} f(\pi_p) \tag{2}$$

3.1 Problemi di equipartizione

In questo tipo di problema, come abbiamo visto, l'obiettivo è quello di costruire una partizione dell'insieme $V(G)$ in p componenti $V_1,\ldots,V_p$ non vuote, in modo da rendere il più uniforme possibile il peso $W(V_k) = \sum_{v_i \in V_k} w(v_i)$ delle componenti. In altri termini ci si propone di trovare una p-partizione π di V tale che i pesi delle componenti siano vicini tra loro. Osserviamo subito che il problema di partizionare un grafo completo in due componenti di uguale peso, è immediatamente riconducibile a SUBSET-SUM, che è NP-completo (Karp, 1972).

Indichiamo con μ il peso medio delle componenti della partizione, definita in (1). Per ciascuna p-partizione $\pi_p = \{V_1,\ldots,V_p\}$ di G possiamo definire il vettore $\mathbf{W} = (W_1,\ldots,W_p) \in \mathbb{Z}^p$, con $W_k = W(V_k)$. In un problema di equipartizione si vuole ottenere una p-partizione in cui il vettore $\mathbf{W}$ si avvicini il più possibile al vettore $\overline{\mu} = (\mu,\ldots,\mu)$. Per stimare la distanza fra $\mathbf{W}$ e $\overline{\mu}$ possiamo utilizzare il concetto di norma di un vettore: $\|\mathbf{W} - \overline{\mu}\|$.

Vengono così definite diverse funzioni obiettivo da minimizzare, per il problema della equipartizione in p componenti connesse di un grafo G:

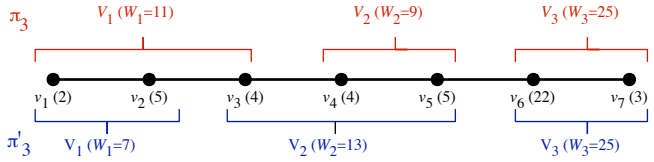


Figura 3: Il cammino P_7 con $n = 7$ vertici e due p-partizioni π_3 e π'_3 in $p = 3$ componenti connesse

- Norma L_1:

$$f(\pi_p) = \|\mathbf{W} - \overline{\mu}\|_1 = \sum_{k=1}^{p} |W(V_k) - \mu| \tag{3}$$

- Norma L_2:

$$f(\pi_p) = \|\mathbf{W} - \overline{\mu}\|_2 = \sum_{k=1}^{p} (W(V_k) - \mu)^2 \tag{4}$$

- Norma L_∞:

$$f(\pi_p) = \|\mathbf{W} - \overline{\mu}\|_\infty = \max_{k=1,\ldots,p} |W(V_k) - \mu| \tag{5}$$

Formuliamo un esempio per chiarire meglio lo scopo del problema di p-partizionamento di un grafo. Consideriamo un grafo molto semplice, costituito dal cammino P_7 con $n = 7$ vertici: $V = \{v_1, v_2, \ldots, v_7\}$ ed $E = \{(v_i, v_{i+1})$, per $i = 1, \ldots, 6\}$. Ai vertici di V associamo i pesi non negativi $W = \{w(v_1) = 2, w(v_2) = 5, w(v_3) = 4, w(v_4) = 4, w(v_5) = 5, w(v_6) = 22, w(v_7) = 3\}$; dunque $\sum_{i=1}^{7} w(v_i) = 45$ e, assumendo di voler produrre delle p-partizioni di P_7 con $p = 3$ componenti, risulta $\mu = 45/3 = 15$.

Consideriamo una delle possibili p-partizioni di P_7, $p = 3$, ad esempio la partizione $\pi_3 = \{V_1 = \{v_1, v_2, v_3\}, V_2 = \{v_4, v_5\}, V_3 = \{v_6, v_7\}\}$. Risulta quindi $W_1 = W(V_1) = 11$, $W_2 = W(V_2) = 9$ e $W_3 = W(V_3) = 25$. Il valore delle tre funzioni obiettivo calcolate per la partizione π_3 è il seguente:

$$\begin{aligned}
L_1 : f(\pi_3) &= |W_1 - \mu| + |W_2 - \mu| + |W_3 - \mu| = \\
&= |11 - 15| + |9 - 15| + |25 - 15| = 4 + 6 + 10 = 20 \\
L_2 : f(\pi_3) &= (W_1 - \mu)^2 + (W_2 - \mu)^2 + (W_3 - \mu)^2 = \\
&= 16 + 36 + 100 = 152 \\
L_\infty : f(\pi_3) &= \max\{|W_1 - \mu|, |W_2 - \mu|, |W_3 - \mu|\} = \\
&= \max\{4, 6, 10\} = 10
\end{aligned}$$

La partizione π_3 è ottimale, ossia non è possibile trovare un'altra p-partizione del cammino che migliori ulteriormente il valore della funzione obiettivo. Ad esempio la partizione π'_3 rappresentata in Figura 3 è tale che la funzione obiettivo calcolata in π'_3 assuma lo stesso valore per L_1 e L_∞ (quindi anche π'_3 è ottimale per queste due funzioni obiettivo), mentre $f(\pi'_3) = 168 > f(\pi_3) = 152$ per la norma L_2.

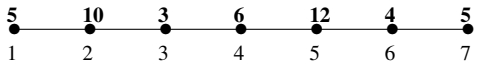


Figura 4: Un cammino P_7 con pesi non negativi assegnati ai vertici

Per ottenere una partizione di G con componenti di peso omogeneo si può cercare di minimizzare il peso della componente più pesante, massimizzare il peso della componente più leggera, o ridurre la differenza di peso tra la componente più pesante e quella più leggera; queste tre strategie possono essere formalizzate con le seguenti funzioni obiettivo (la prima da massimizzare, le altre due da minimizzare):

- Max-Min (da massimizzare):
$$f(\pi) = \min_{k=1,\dots,p} W(V_k) \tag{6}$$
- Min-Max (da minimizzare):
$$f(\pi) = \max_{k=1,\dots,p} W(V_k) \tag{7}$$
- Most Uniform Partition (MUP – da minimizzare):
$$f(\pi) = \max_{k=1,\dots,p} W(V_k) - \min_{k=1,\dots,p} W(V_k) \tag{8}$$

Se f è la norma L_1 il problema è NP-hard per gli alberi e polinomiale per classi speciali di alberi come stelle, cammini e bruchi (Aparo e Simeone, 1975, De Simone et al., 1990). In Becker et al. (1982), Becker et al. (1983), Perl e Schach (1981), vengono forniti alcuni algoritmi di *shifting* che risolvono in tempo polinomiale i problemi di equipartizione di alberi, con funzioni obiettivo (7) e (6).

I problemi MUP sono NP-hard per grafi qualsiasi (Garey e Johnson, 1979), ma esiste un algoritmo polinomiale per risolvere tale problema su cammini (Lucertini et al., 1993). Lari, nella sua tesi di dottorato ([25]), dimostra che tutti i problemi di equipartizione sono NP-hard per grafi a griglia (e quindi anche per grafi planari e bipartiti) e fornisce un algoritmo polinomiale per il problema di equipartizione con funzione obiettivo da massimizzare $f(\pi) = \min_{C\in\pi} W(C)$ su una "scala".

In [26] abbiamo formulato un nuovo algoritmo per l'equipartizione di un cammino con funzione obiettivo la norma L_∞ che migliora in modo significativo la complessità di precedenti algoritmi basati sulla tecnica della programmazione dinamica.

Anche in questo caso un esempio elementare potrà aiutarci a chiarire le differenze tra le funzioni obiettivo. Pur essendo tutte finalizzate ad un'equipartizione del grafo, ammettono partizioni ottimali differenti. Da notare che si parla sempre di soluzioni ottimali e non di un'unica soluzione ottima, perché, come è evidente, possono essere più d'una le partizioni di un grafo che portano a minimizzare (o massimizzare) la funzione obiettivo assegnata. Consideriamo un cammino costituito da sette vertici ($n = 7$) a cui assegnamo dei pesi $w_i \geq 0$ come in Figura 4 (i pesi sono indicati in grassetto). Vogliamo partizionare in tre componenti connesse il cammino ($p = 3$), per minimizzare la funzione obiettivo Min-Max e la norma L_∞ e per massimizzare la funzione Max-Min.

Si riconosce facilmente che la partizione $\pi^* = \{\{1,2,3\},\{4,5\},\{6,7\}\}$ è ottimale per tutte e tre le funzioni obiettivo, mentre la partizione $\bar{\pi}^* = \{\{1,2\},\{3,4,5\},\{6,7\}\}$ è ottimale per i problemi con funzione obiettivo Max-Min e la norma L_∞, ma non per quello basato sulla funzione Min-Max.

3.2 Problemi di clustering

Come abbiamo visto in precedenza, questo tipo di problema di ottimizzazione ha l'obiettivo di produrre una p-partizione π del grafo G assegnato, minimizzando la dissimilarità tra gli elementi aggregati in una stessa componente della partizione o massimizzando la dissimilarità tra gli elementi di componenti diverse. Nel primo caso si ottiene una partizione in cui è massima l'omogeneità all'interno di ciascuna componente, mentre nel secondo caso si ottiene la massima separazione tra le componenti della partizione. I due obiettivi sembrano analoghi, ma visto che operiamo su oggetti discreti, possono produrre partizioni ottime molto diverse fra loro.

3.2.1 Massima omogeneità all'interno delle componenti

Sia $G = (V,E)$ un grafo e $D = (d_{ij})$ una matrice di dissimilarità, quadrata di ordine n. Fissata una costante $p \geq 2$, vogliamo individuare la partizione $\pi = \{V_1, \ldots, V_p\}$ di V che minimizzi la funzione obiettivo $f(\pi) : \Pi_p \to \mathbb{R}^+$, definita sull'insieme $\Pi_p(G)$, costituito da tutte le partizioni di V in p componenti non nulle.

Per ottenere la massima omogeneità all'interno delle componenti, è possibile considerare le seguenti funzioni obiettivo da *minimizzare*:

- Massimo diametro: per ogni $k = 1,2,\ldots,p$, definiamo come $d_{V_k} = \max\{d_{ij} : v_i, v_j \in V_k\}$ il **diametro della componente** V_k, inteso come la massima "distanza" (dissimilarità) tra due elementi della compnente; allora la funzione obiettivo può essere definita ponendo

$$f(\pi) = \max_{k=1,\ldots,p} \left\{ \max_{v_i, v_j \in V_k} d_{ij} \right\} \tag{9}$$

- Somma delle distanze:

$$f(\pi) = \sum_{k=1}^{p} \left(\sum_{v_i, v_j \in V_k} d_{ij} \right) \tag{10}$$

In entrambi i casi si può osservare facilmente che una partizione π^* che minimizzi la funzione obiettivo $f(\pi)$, è tale da rendere il più simili tra loro (meno distanti) gli elementi di ogni componente. Naturalmente la partizione dell'insieme V può avvenire in due modi completamente diversi a seconda che si utilizzi come funzione obiettivo la (9) o la (10).

Ad esempio consideriamo il grafo completo K_4 i cui vertici sono gli elementi dell'insieme $V = \{v_1, v_2, v_3, v_4\}$; associamo agli spigoli del grafo (per i grafi completi esiste uno spigolo per ogni coppia di vertici) gli indici di dissimilarità d_{ij} nel seguente modo: $d_{1,2} = 1$, $d_{1,3} = 2$, $d_{1,4} = 4$, $d_{2,3} = 2$, $d_{2,4} = 3$, $d_{3,4} = 3$ (vedi Figura 5). Supponiamo di voler suddividere il grafo in due componenti connesse ($p = 2$). La partizione ottimale per minimizzare la funzione obiettivo (9) "massimo diametro" è $\pi^* = \{\{v_1, v_2, v_3\}, \{v_4\}\}$, e risulta $f(\pi^*) = 2$. Per minimizzare la funzione obiettivo (10) "somma delle distanze" la partizione ottimale è invece $\bar{\pi}^* = \{\{v_1, v_2\}, \{v_3, v_4\}\}$, per cui risulta $f(\bar{\pi}^*) = 4$.

Se G è un grafo completo e $p = 2$, con queste due funzioni obiettivo si ottengono problemi di complessità polinomiale; se $p > 2$ allora diventano NP-hard (Hansen e Jaumard, 1987). Se il grafo è un albero il problema con funzione obiettivo "massimo diametro" è NP-hard (si veda [28]).

3.2.2 Massima distanza tra le componenti

Problema correlato al precedente, ma che in certi casi conduce a risultati sorprendentemente diversi, è quello basato sulle due seguenti funzioni obiettivo, mirate entrambe a separare gli elementi, in modo da massimizzare la distanza tra i *cluster*:

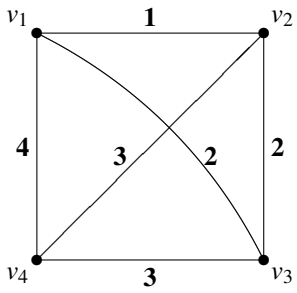


Figura 5: Grafo completo K_4 con pesi interi non negativi assegnati agli spigoli

- Minimo *split*: Si definisce lo *split* di una componente della partizione, come la minima dissimilarità con elementi esterni a tale componente: $\text{split}(V_k) = \min_{i \in V_k,\ j \in V \setminus V_k} d_{ij}$. In questo modo è possibile definire la seguente funzione obiettivo:

$$f(\pi) = \min_{V_k \in \pi} \left\{ \min_{i \in V_k,\ j \in V \setminus V_k} d_{ij} \right\} \tag{11}$$

- Somma delle distanze tra cluster:

$$f(\pi) = \sum_{k=1}^{p} \left(\sum_{i \in V_k,\ j \in V \setminus V_k} d_{ij} \right) \tag{12}$$

  In questo caso la somma delle distanze viene effettuata, per ciascuna componente, tra coppie di vertici formate da un elemento della componente V_k e gli elementi esterni a tale componente; in (10) le somme invece vengono effettuate tra elementi della stessa componente.

Anche per queste due funzioni obiettivo per problemi di clustering, come nei casi precedenti, possiamo prendere ad esempio un grafo completo K^4, con insieme dei vertici $V = \{v_1, v_2, v_3, v_4\}$ da suddividere in $p = 2$ componenti connesse. L'obiettivo è quello di trovare una partizione π^* che *massimizzi* le due funzioni (11) e (12); chiaramente, come già abbiamo osservato nel problema precedente, pur avendo entrambe come finalità quella di massimizzare la distanza tra le componenti della partizione, possono in taluni casi condurre a partizioni differenti.

Ad esempio consideriamo il grafo completo K_4 rappresentato in Figura 6; in questo caso abbiamo ridefinito le distanze tra i vertici del grafo rispetto a quelle riportate in Figura 5. La funzione obiettivo (11) "minimo split" ci porta ad individuare la partizione $\pi^* = \{\{v_1, v_2\}, \{v_3, v_4\}\}$ che fornisce un valore $f(\pi^*) = 3$, mentre con la funzione obiettivo (12) "somma delle distanze tra cluster" si ottiene la partizione ottimale $\bar{\pi}^* = \{\{v_1, v_4\}, \{v_2, v_3\}\}$, con valore $f(\bar{\pi}^*) = 22$.

Il problema di partizione connessa con funzione obiettivo "minimo split" è NP-hard per grafi a griglia (Hansen et al., 1993, Garey e Johnson, 1977). Il problema è invece polinomiale se G è un grafo completo (Delattre e Hansen, 1980), se G è un albero (Maravalle e Simeone, 1992, Hansen et al., 1993) e se G è una *scala*, cioè un grafo a griglia con due righe ed un numero arbitrario di colonne (Lari, 1994).

Il problema di partizione con funzione obiettivo "somma delle distanze tra cluster" è NP-hard per G qualsiasi e $p \geq 2$ (Welch, 1982).

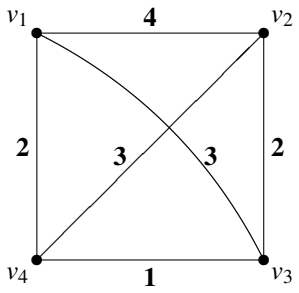


Figura 6: Grafo completo K_4 con altri pesi assegnati agli spigoli

3.3 Taglio della partizione

Oltre alle due famiglie principali di problemi per il p-partizionamento ottimo di grafi in componenti connesse, viste nelle pagine precedenti, è opportuno citare un'altra classe di problemi di ottimizzazione piuttosto ampia, quella dei problemi di *taglio della partizione*.

Dato un grafo $G = (V,E)$, si associa ad ogni spigolo un peso non negativo; data una partizione π di G, per **taglio della partizione** si intende la somma dei pesi degli spigoli i cui estremi appartengono a componenti diverse (gli spigoli "tagliati" dalla partizione). In certe applicazioni si richiede che il taglio sia massimo, in altre invece che il taglio sia minimo. È interessante osservare che se i pesi associati agli spigoli sono tutti unitari, allora il taglio fornisce una misura della maggiore o minore connessione della partizione: se il taglio della partizione π_1 è minore di quello della partizione π_2, allora le componenti della prima risulteranno maggiormente connesse di quelle della seconda.

Distinguiamo tre sotto classi di problemi relativi al taglio della partizione:

- Partizione non pesata: dato un grafo con vertici non pesati (o di peso costante), si vuole determinare una partizione in due componenti di uguale cardinalità in modo tale da minimizzare il numero di spigoli del taglio (o il peso del taglio). Questo problema è NP-hard (Garey et al., 1976).

- Taglio minimo: dato un grafo pesato sugli spigoli, l'obiettivo è quello di determinare una bipartizione dei vertici che minimizzi il taglio (non è richiesto il vincolo della pari cardinalità delle due componenti). Questo problema è risolvibile in tempo polinomiale tramite l'applicazione di algoritmi per il massimo flusso e rimane polinomiale anche per ipergrafi (Lawker, 1983). Diventa in genere di difficile soluzione se si aggiungono vincoli sulla topologia delle componenti.

- Taglio massimo: dato un grafo pesato sugli spigoli, lo si vuole ripartire in due componenti in modo che il taglio risulti essere massimo. Questo problema è NP-hard (Karp, 1972). Rimane NP-hard anche nel caso di grafi non pesati (Garey et al., 1976) e se il grado massimo dei vertici è uguale a 3 (Yannakakis, 1978). Può essere risolto in tempo polinomiale su grafi pesati se il grafo è contraibile a K_5 (Orlova e Dorfman, 1972, Hadlock, 1975, Barahona, 1982) e se non contiene cicli dispari "lunghi" (questa classe contiene i grafi bipartiti) (Grötshel e Nemhauser, 1984) e su edge-graphs non pesati (Arbib, 1987).

3.4 Partizionamento continuo

Sia $T = (V,E)$ un albero con insieme dei nodi $V = \{v_1, v_2, \ldots, v_n\}$ e insieme degli spigoli $E = \{e_1, e_2, \ldots, e_{n-1}\}$. Ciascuno spigolo e_i, $1 \leq i \leq n-1$, ha associato una *lunghezza* positiva l_i. Assumiamo che T sia *immerso* nel piano Euclideo, i suoi spigoli e nodi sono cioè punti nel piano.

Sia $A(T)$ l'insieme continuo dei punti degli spigoli di T; ciascuna coppia di segmenti può intersecarsi al più in un punto non interno.

Se Y è un sottoalbero di T, definiamo la *lunghezza* di Y, $l(Y)$, come la somma delle lunghezze dei suoi spigoli e dei suoi spigoli parziali. Definiamo, invece, *diametro* di Y, $diam(Y)$, la lunghezza del più lungo cammino semplice in Y.

Una *partizione continua* di T in p componenti è un insieme di p sottoalberi di $A(T)$ tale che nessuna coppia di sottoalberi si intersechi in più di un punto e tale che l'unione dei sottoalberi sia $A(T)$.

In analogia con il caso discreto, è possibile definire problemi di partizionamento ottimo continuo di alberi in p componenti. Un problema al quale recentemente è stata data attenzione è quello del *partizionamento continuo max-min* di alberi, si veda a proposito [6]. Indichiamo con $T_1, T_2, \ldots, T_p$ i sottoalberi di $A(T)$ indotti da una partizione continua π di T. Una partizione max-min ottima di T è una partizione che massimizza la funzione obiettivo

- Length max-min continuous:
$$f(\pi) = \min_{1 \leq k \leq p} l(T_k) \tag{13}$$

4 Applicazioni

Come abbiamo accennato nell'introduzione, le applicazioni dei problemi di partizionamento ottimo possono essere le più varie: da problemi di classificazione a problemi di ottimizzazione delle comunicazioni, problemi di modellizzazione di un campione di indagine statistica, e molti altri ancora. In estrema sintesi vale la pena citare alcuni esempi più significativi, anche per dare una misura dell'ampiezza della gamma di campi applicativi di questo tipo di problematiche.

In biologia gli elementi dell'insieme da studiare possono essere forme di vita come piante, animali, microorganismi e l'obiettivo dello studio può consistere nella definizione di un'intera tassonomia o nella delimitazione delle sottospecie di una specie. Ad esempio, Rogers e Tanimoto (1960) hanno proposto un programma per la classificazione delle piante; Sneath (1957) ha analizzato il problema di individuare una tassonomia per gruppi di batteri; Sokal e Sneath (1963), hanno utilizzato diverse tecniche di partizione per ottenere "dendrogrammi" per dati biologici.

In geologia sono frequenti problemi di classificazione delle rocce e del suolo, delle città e delle regioni, dei bacini idrici ed in generale legati al deflusso delle acque. Wolf et al. (1972) hanno usato una procedura di *clustering* per lo studio del movimento delle nuvole.

Alcuni problemi di tipo "ingegneristico" fanno spesso uso di tecniche di *clustering*, come ad esempio nel riconoscimento delle forme, nella strutturazione della base di conoscenza nei sistemi esperti ed in altre applicazioni dell'intelligenza artificiale, nel progetto di circuiti VLSI, nella progettazione di reti di calcolatori o, più in generale, di reti di telecomunicazione; nel trattamento delle immagini digitali si ha un interessante esempio di utilizzo di tecniche di equipartizione. Akers (1982), Korte et al. (1988), Arbib et al. (1989), Antenucci et al. (1990), hanno individuato problemi di partizione nel progetto di circuiti; per il trattamento delle immagini si veda ad esempio Haralick e Dinstein (1975), Narendra e Goldberg (1980), Warthon (1983), Lucertini et al. (1983).

In medicina gli oggetti dell'analisi possono essere pazienti, sintomi o test di laboratorio. L'obietivo principale è la scoperta di mezzi più efficaci ed economici per la diagnosi. Gose et al. (1972), per

esempio, hanno utilizzato un algoritmo di partizione in una procedura per identificare il tumore al seno mediante radiografia.

Nelle scienze comportamentali e sociali trovano frequente applicazione una grande varietà di problemi di partizione. Ad esempio nello studio di metodi di insegnamento, schemi di comportamento, fattori di rendimento umano, famiglie, organizzazioni, *social network*, ecc. In particolare: Kaskey et al. (1962) hanno utilizzato tecniche di *clustering* per la diagnosi dei pazienti di un istituto psichiatrico; Haralick e Haralick (1971) hanno analizzato il comportamento di bambini sordi mediante una partizione di variabili.

Le scienze politiche, economiche e delle decisioni in cui i problemi di partizione riguardano, per esempio, la segmentazione della clientela nel marketing, la sequenzializzazione e l'assegnazione di compiti a macchine in una catena di produzione, scelta di investimenti, dislocazione di impianti. In particolare citiamo: Garfinkel e Nemhauser (1969), per un'applicazione riguardante la partizione di centri di popolazione in distretti politici; Rizzi (1981) che riporta un'applicazione alle proiezioni dei risultati elettorali; Freeman e Jucker (1967), per i problemi di bilanciamento di catene di produzione.

Descriviamo, citando qualche aspetto particolare, alcuni di questi problemi.

4.1 Elaborazione delle immagini digitali

I metodi risolutivi per il problema dell'equipartizione di un cammino con n vertici in p sotto-cammini, possono essere utilizzati per risolvere un problema di elaborazione di immagini digitali. Supponiamo di aver acquisito mediante una strumentazione elettronica un'immagine digitale di qualità fotografica. In genere questo tipo di immagini sono rappresentate nella memoria della macchina tramite una matrice di punti, tanto più grande quanto maggiore è la risoluzione grafica dell'apparato digitale di acquisizione a nostra disposizione. Ogni punto della matrice può essere rappresentato mediante un numero intero c, $0 \leq c \leq max$, in cui il valore 0 rappresenta un punto completamente nero, il valore max un punto bianco[1], mentre i valori intermedi rappresentano quantità crescenti di luminosità in una "scala di grigi" che va, appunto, da nero al bianco. Una rappresentazione analoga può essere effettuata anche nel caso di immagini a colori, in cui il livello cromatico di ogni punto dell'immagine può essere scomposto in tre quantità distinte, associate ai tre colori di base rosso, verde e blu, dalla cui combinazione si può ricavare ogni altro colore in diverse tonalità.

Per problemi legati all'occupazione di memoria di questo tipo di rappresentazione, o alle caratteristiche di alcuni sistemi di riproduzione o stampa delle immagini stesse, è spesso utile ridurre la scala di livelli di grigio (o le diverse tonalità di colore), dalle originarie max gradazioni ad un numero molto più piccolo, ad esempio $p = max \cdot 10^{-3}$. È stato verificato per via sperimentale che il migliore contrasto tra i p livelli di grigio si ha rendendo omogeneo il numero di *pixel* (punti dell'immagine digitale) per ogni livello di luminosità.

Il problema si presta bene ad essere modellizzato mediante un cammino di max vertici da partizionare in p componenti connesse (sottocammini) secondo un algoritmo di equipartizione. Rappresentiamo il livello di grigio i nella scala cromatica originale mediante l'i-esimo vertice del cammino ed associamo a tale vertice un peso $w(i) \geq 0$ pari al numero di pixel aventi tale livello di luminosità nell'immagine grafica originale. Effettuando una partizione del cammino assumendo come funzione obiettivo una delle funzioni viste nel caso del problema della equipartizione (norma L_∞, norma L_1, norma L_2, funzione max-min, ecc.) si ottiene una suddivisione in p componenti connesse; ad ognuna di tali componenti assegneremo progressivamente il nuovo livello di grigio (o tonalità di colore), che dovrà essere applicato a tutti i pixel di ogni componente.

[1] Il valore numerico di max dipende dalla sensibilità cromatica dello strumento di acquisizione utilizzato; con i dispositivi attuali $max \simeq 10^6$.

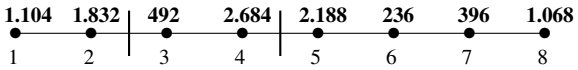


Figura 7: Equipartizione del cammino P_8 in 3 componenti (sotto-cammini) con funzione obiettivo norma L_∞

Per maggiore chiarezza facciamo un esempio con un numero di dati molto piccolo. Supponiamo di avere un'immagine di 100×100 pixel in 8 livelli di grigio e supponiamo di voler ridurre il numero di sfumature da 8 a 3. Rappresentiamo la situazione mediante un cammino ed associamo ad ogni vertice il numero di pixel che nell'immagine originale assumono tale intensità. In Figura 7 sono rappresentati in grassetto il numero di pixel caratterizzati da un determinato livello di grigio nell'immagine originale: 1.104 pixel assumono la tonalità numero 1, 1.832 la numero 2 e così via... Naturalmente $\sum_{i=1}^{8} w(i) = 10.000$, perché l'immagine ha $100 \times 100 = 10.000$ pixel, ognuno dei quali è caratterizzato da una delle 8 tonalità di grigio rappresentate dai vertici del cammino.

La partizione ottimale, considerando come funzione obiettivo la norma L_∞, è data da

$$\pi^* = \{\{1,2\},\{3,4\},\{5,6,7,8\}\}$$

quindi ai pixel che nell'immagine originaria avevano le tonalità 1 e 2 verrà assegnata la nuova tonalità c_1 (i vertici 1 e 2 del cammino ricadono nella prima componente della partizione), ai pixel che prima avevano tonalità 3 e 4 sarà assegnata la nuova tonalità c_2, mentre i rimanenti pixel assumeranno la tonalità c_3.

Tenendo conto di quanto affermato nella Proposizione 1, risulta che il numero di p-partizioni possibili in p componenti (sotto-cammini) di un cammino con n vertici è pari a $\binom{n-1}{p-1}$. Dunque, se l'immagine originale ha $n = 256$ torni di colore e si intende ridurli a $p = 16$ toni, il numero di partizioni possibili è dato da:

$$|\Pi_p(P_n)| = \binom{256-1}{16-1} = 629.921.975.126.394.617.164.575 \approx 6,29 \times 10^{23}$$

In un caso più realistico, in cui il numero di gradazioni è $n = 32.000$ e si intende ridurle a sole $p = 256$ tonalità, avremo un numero di partizioni possibili pari a

$$|\Pi_p(P_n)| = \binom{32.000-1}{256-1} \approx 6,98 \times 10^{643}$$

Dunque il problema del p-partizionamento di un cammino, pur nella sua semplicità, non si presta affatto ad essere affrontato con un approccio "esaustivo" di costruzione e valutazione di tutte le possibili partizioni ottenute per combinazione dei tagli da assegnare agli spigoli del cammino.

Un altro tipo di problema, collegato sempre all'elaborazione delle immagini digitali, potrebbe essere quello di riconoscere delle aree omogenee all'interno dell'immagine, utile, ad esempio, nell'elaborazione di immagini provenienti da un satellite, in cui ad aree caratterizzate da un livello cromatico uniforme corrispondono zone fisicamente omogenee (stesso tipo di territorio oppure stessa temperatura, ecc.). Questo problema può essere formalizzato come un problema di partizionamento di un grafo a griglia.

4.2 Progettazione di circuiti VLSI

Una fase importante nel progetto di circuiti VLSI (*Very Large Scale Integration*) è la suddivisione dei componenti in *pagine* o *strati* di capacità fissata, in modo tale che la porzione di circuito che viene

realizzata su un singolo strato possa essere rappresentata mediante un grafo planare. In generale viene richiesto che il numero di collegamenti da uno strato ad un altro sia minimo.

Il circuito può essere allora rappresentato mediante un grafo $G = (V, E)$ in cui i vertici sono in corrispondenza biunivoca con i componenti elettronici; ad ogni vertice si assegna poi un peso $w(v_i) \geq 0$ pari alla dimensione del componente rappresentato dal vertice stesso; gli spigoli del grafo rappresentano i collegamenti tra i componenti.

Con questa modellizzazione il problema di suddividere l'intero circuito in sottocircuiti (paginazione o stratificazione) può essere espresso come un problema di partizione di un grafo pesato, in cui il peso di ciascuna componente non può superare una soglia c fissata a priori (la capacità delle pagine), ed in cui si vuole minimizzare il numero di connessioni tra le componenti.

Questo problema di partizione è NP-hard, tuttavia spesso il circuito, e quindi anche il grafo che lo rappresenta, ha delle caratteristiche che rendono la soluzione più semplice. Per esempio se ogni componente ha non più di quattro connessioni con altri componenti, allora si può procedere all'implementazione direttamente su una struttura a griglia, anziché su un grafo generico.

4.3 Progettazione di una rete di calcolatori

Una topologia tipica per le reti di calcolatori vede raggruppate in cluster un certo numero di macchine dislocate in uno stesso ambiente di lavoro o in stanze fisicamente vicine tra loro, ovvero, per ragioni di implementazione del servizio che deve essere offerto, su una stessa "sottorete". Ogni cluster è a sua volta collegato, mediante una macchina appositamente dedicata a questo scopo, chiamata *router* o *gateway*, ad un tratto di collegamento che permette di comunicare con un cluster adiacente. La rete è quindi strutturata in gruppi di calcolatori (sottoreti) collegati ad una linea di comunicazione principale che connette tra loro i gruppi. Questa linea può anche avere delle biforcazioni, componendo in questo modo una struttura di grafo arbitraria. I vincoli nella progettazione di una rete di questo tipo sono dati dal massimo numero di calcolatori per ogni cluster e dalla vicinanza dei calcolatori di uno stesso cluster. In generale è poi desiderabile che due calcolatori che debbano comunicare intensamente fra loro appartengano ad uno stesso cluster, in modo da minimizzare l'uso della banda passante sulla linea di comunicazione principale.

La rete descritta si presta ad essere rappresentata facilmente mediante un grafo, o più spesso, un albero $T = (V, E)$, i cui vertici sono i singoli calcolatori e gli spigoli del grafo sono i collegamenti tra le macchine. Ad ogni spigolo viene associata una distanza $d_{i,j} \geq 0$ pari alla distanza tra le due macchine rappresentate dai vertici v_i e v_j.

Il problema è allora quello di partizionare l'albero T in p sottoalberi connessi (i cluster di calcolatori) in modo da minimizzare il peso delle componenti (ad esempio con funzioni obiettivo "massimo diametro" o "somma delle distanze" da minimizzare).

4.4 Proiezione dei risultati elettorali

Si fa ricorso all'analisi dei gruppi per la previsione dei risultati delle elezioni. In queste occasioni infatti si devono anticipare i risultati finali in base ad una rilevazione per campione dei primi spogli, da effettuare in tempi brevissimi. In questi casi non è possibile utilizzare un campione rappresentativo dell'universo delle sezioni elettorali perché non è possibile ottenere tutti i dati del campione in tempo utile. Per la determinazione del campione si ricorre quindi alla stratificazione della popolazione ed alla ponderazione dei vari strati. Ma una semplice stratificazione in base a caratteri geografici e demografici può portare a distorsioni non trascurabili, per cui si fa ricorso all'analisi dei gruppi come metodo di stratificazione. Si suddivide il Paese in un certo numero di areole e, basandosi sui risultati

Equipartizione		
Obiettivo	Cammini	Alberi
L_1	$O(np)$	NP-completo, $O(n^2p)$ per caterpillars
L_2	$O(n^2p)$	-
L_∞	$O(np\log_2 p)$	-
Max-Min	$O(np\log_2 o)$	$O(np\log_2 p)$
Min-Max	$O(np\log_2 p)$	$O(n^2p^3)$
MUP	$O(n^3)$	-
Clustering		
Obiettivo	Cammini	Alberi
Inner dissimilarity	$O(n^2)$	NP-completo (star)
Massimo diametro	$O(n^2p)$	NP-completo (star)
Minimo split	$O(n^2)$	$O(n^2)$

Tabella 1: Complessità computazionale di alcuni problemi di equipartizione e di *clustering* su alberi e su cammini

delle elezioni precedenti, si applica un algoritmo di analisi dei gruppi. Si individuano in questo modo gruppi politicamente omogenei.

5 Complessità computazionale

Generalmente, un problema di partizionamento su cammini può essere risolto in tempo polinomiale $O(pn^2)$ con l'impiego della programmazione dinamica per tutte le funzioni obiettivo di uso più comune. Utilizzando tecniche di partizionamento differenti è possibile abbassare ulteriormente la complessità di calcolo. Tali tecniche hanno portato ad algoritmi spesso molto semplici ed intuitivi, ma la cui correttezza è stata dimostrata in maniera tutt'altro che banale, utilizzando interessanti connessioni con la matematica pura.

I problemi di partizionamento precedentemente introdotti sono, invece, tutti NP-hard per grafi qualsiasi.

Quando il grafo da partizionare è un albero la complessità computazionale del problema sembra dipendere fortemente dalla funzione obiettivo. Il caso degli alberi è infatti *borderline* tra cammini e grafi qualsiasi, e si pone, quindi, come spartiacque tra problemi risolvibili in tempo polinomiale e problemi per cui si ricerca una soluzione approssimata.

Lo studio dei problemi di partizionamento sui grafi è un settore di ricerca ancora aperto in cui molti problemi risultano ancora inesplorati. Riportiamo in Tabella 1 una sintesi dei risultati noti relativi alla complessità computazionale di alcuni problemi di p-partizionamento di grafi definiti per alcune delle funzioni obiettivo descritte nelle pagine precedenti.

6 Algoritmi

Per quanto riguarda gli aspetti algoritmici nella soluzione di problemi di partizione, gli approcci utilizzati più frequentemente sono sostanzialmente tre:

- formulazione dei problemi in termini di Programmazione Lineare a Numeri Interi (PLI) ed uso di programmi applicativi standard per la risoluzione in modo esatto o approssimato;

- sviluppo di algoritmi, esatti o approssimati, per specifiche classi di problemi;

- uso di euristiche di ampia applicabilità, di tipo deterministico o probabilistico.

Naturalmente nessuno di questi approcci è ottimale sotto ogni aspetto. Infatti i pacchetti applicativi standard sono tanto meno efficienti ed in grado di gestire problemi di dimensioni elevate, quanto più generale è la classe di problemi su cui sono utilizzabili. D'altro canto lo sviluppo di algoritmi specifici per un singolo problema o per una ristretta classe di problemi, porta con sé il difetto opposto: ad una grande efficienza operativa si contrappone una scarsa riutilizzabilità della procedura su problemi differenti. Infine, la assoluta generalità degli algoritmi euristici di tipo deterministico o probabilistico, può rivelarsi fonte di inefficienza quando non vengano sfruttate a fondo le caratteristiche specifiche del problema in esame.

Di questi tre approcci, comunque, l'ultimo rappresenta spesso l'unica alternativa praticabile. Può essere utile quindi, riportare alcune delle tecniche più diffuse nell'ambito degli algoritmi per il partizionamento (vedi [5]).

- **Ottimizzazione su reti**: la ricerca di una partizione utilizzando questo metodo fa ricorso, ad esempio, ad algoritmi di massimo flusso–minimo taglio o ad algoritmi di massimo taglio (*max-cut*).

- **Migrazione di gruppi**: questa tecnica consiste nel generare una partizione iniziale e nel tentare un suo miglioramento attraverso lo scambio di un piccolo gruppo di vertici (al limite un solo vertice) tra due o più componenti della partizione. Questo passo viene ripetuto fino a quando non è più possibile un miglioramento. Questa tecnica di "ricerca locale" può prevedere l'introduzione di passi di tipo probabilistico tanto nella determinazione della soluzione iniziale quanto nella fase di miglioramento e nel criterio di arresto.

- **Simulated annealing**: differisce dalla tecnica di *migrazione di gruppi* solo per quanto riguarda il criterio di accettazione della nuova partizione generata: vengono infatti accettate anche partizioni che peggiorano il valore della funzione obiettivo, nella speranza che questo peggioramento permetta di superare eventuali minimi o massimi locali. Il criterio di interruzione della ricerca è basato pertanto su condizioni più forti e restrittive che non la sola valutazione del valore della funzione obiettivo.

- **Semi**: questa tecnica viene usata nel caso di partizioni in un numero p di componenti prefissato. Definita sull'insieme dei vertici del grafo una metrica, si individuano i p vertici che distano di più gli uni dagli altri. Questi vertici sono detti *semi*. Compatibilmente con i vincoli sulle dimensioni di ogni componente della partizione, ogni altro vertice del grafo viene assegnato al seme più vicino, o al secondo più vicino, ecc.

- **Greedy**: si tratta di organizzare il problema in una sequenza di sottoproblemi più semplici, in ognuno dei quali viene assegnato in modo ottimo a una delle componenti un insieme di vertici (al limite uno solo). Le soluzioni ottime calcolate non vengono modificate nei problemi successivi, il numero di sottoproblemi risolti è lineare nel numero dei vertici del grafo e la complessità del procedimento dipende dunque dalla complessità di risoluzione dei singoli sottoproblemi.

- **Programmazione dinamica**: come il metodo *divide et impera* ([13]) risolve il problema combinando la soluzione di sottoproblemi. Gli algoritmi di tipo *divide et impera* suddividono il problema originario in sottoproblemi indipendenti, li risolvono ricorsivamente, ed infine combinano insieme le loro soluzioni per ottenere la soluzione del problema originario. Al contrario, la *programmazione dinamica* si applica quando i sottoproblemi non sono indipendenti, cioè quando i sottoproblemi condividono degli ulteriori sottoproblemi. In questo caso un algoritmo di tipo *divide et impera* lavorerebbe più del necessario, risolvendo ripetutamente gli stessi sottoproblemi. Un algoritmo che sfrutta la *programmazione dinamica* invece, risolve ogni sottoproblema una sola volta memorizzando la soluzione per poterla riutilizzare (senza doverla ricalcolare) ogni volta che il sottoproblema viene incontrato.

Numerosi algoritmi riportati in letteratura, riguardanti la soluzione di problemi di partizionamento ottimo in componenti connesse di alberi e cammini, utilizzano la tecnica dello *shifting*, applicabile sia alla strategia della Migrazione di gruppi che a quella Greedy o del Simulated annealing. La partizione viene costruita "tagliando" alcuni spigoli in modo da disconnettere le p componenti. Sugli alberi e sui cammini (un caso particolare di albero) una p-partizione in componenti connesse si ottiene tagliando esattamente $p-1$ spigoli; lo stesso non si può dire su un grafo generico. Dunque, partendo da una assegnazione iniziale arbitraria dei tagli a $p-1$ spigoli distinti, si spostano i tagli da uno spigolo all'altro, avendo l'accortezza di non creare mai componenti vuote (è il caso in cui due o più tagli sono assegnati allo stesso spigolo). Se l'operazione di *shift* degli spigoli avviene sempre nella stessa direzione (ad esempio dalla radice alle foglie), allora si ha anche la certezza che l'algoritmo avrà termine dopo un numero finito di passi, pari al più a $|E(G)| \cdot (p-1)$. Tuttavia, per ovviare a scelte che pur essendo localmente ottime si rivelano errate nelle iterazioni successive dei passi dell'algoritmo, alcune strategie risolutive per il partizionamento di alberi (si vedano ad esempio [9] e [30]) ammettono anche la possibilità di compiere "*shift* laterali", ossia spostamenti di tagli da uno spigolo ad un altro non incidente.

6.1 Programmazione lineare intera

I problemi di p-partizione ottima di grafi in componenti connesse possono essere formulati come problemi di Programmazione Lineare Interna (PLI). Di seguito riportiamo la definizione generale delle variabili e dei vincoli in termini di programmazione lineare. Nel seguito definiamo gli elementi necessari per la formulazione di un problema di p-partizione in termini di PLI.

6.1.1 Indici

Nella formulazione del problema utilizzeremo le seguenti convenzioni generali per gli indici che identificano le variabili del sistema di equazioni lineari:

- i: indice che identifica il vertice $v_i \in V(G)$ del grafo G, per $i = 1, \ldots, n$.

- j: indice di un insieme della partizione; se il numero di componenti p della partizione è fissato (problema di p-partizione), allora $j = 1, \ldots, p$; altrimenti $j = 1, 2, \ldots, n$ e si consente che alcune componenti della partizione siano vuote.

- k: indice di uno spigolo di G; $k = 1, \ldots, m = |E|$.

6.1.2 Variabili

Di seguito sono definite le variabili con cui si intende descrivere i termini del problema di partizionamento.

- Assegnazione dei vertici del grafo:

$$x(i,j) = \begin{cases} 1 & \text{se il vertice } v_i \text{ è assegnato alla componente } V_j \text{ della partizione } \pi \\ 0 & \text{se il vertice } v_i \text{ non è assegnato alla componente } V_j \end{cases}$$

- Contenimento di spigoli

$$z(k,j) = \begin{cases} 1 & \text{se lo spigolo } k \text{ è contenuto nella componente } V_j, \text{ ossia se entrambi} \\ & \text{gli estremi dello spigolo } k \text{ sono in } V_j \\ 0 & \text{se lo spigolo } k \text{ non è contenuto nell'insieme } V_j, \text{ ossia se almeno uno} \\ & \text{dei due estremi dello spigolo } k \text{ on è in } V_j \end{cases}$$

Nel caso in cui il numero $p > 0$ di componenti della partizione sia fissato (p-partizione di G) occorre considerare anche le variabili di "attivazione" degli insiemi, con cui si indica se una determinata componente della partizione è ammissibile, perché contiene almeno un vertice del grafo, oppure no:

$$y(j) = \begin{cases} 1 & \text{se l'insieme } V_j \text{ è attivo (contiene almeno un vertice)} \\ 0 & \text{se l'insieme } V_j \text{ non è attivo (non contiene vertici)} \end{cases}$$

6.1.3 Espressione dei vincoli

Infine occorre definire i vincoli del problema, con cui restringeremo l'insieme delle soluzioni ammissibili alle partizioni $\pi \in \Pi_p$ che soddisfano le condizioni di seguito esposte, che possono variare ed essere presenti o meno, a seconda del particolare problema di ottimizzazione che si sta prendendo in esame.

La seguente condizione esprime il fatto che ogni vertice di $v_i \in V(G)$ deve essere assegnato ad una ed una sola componente della partizione π:

$$\sum_{j=1}^{p} x(i,j) = 1 \text{ per ogni } i = 1,\ldots,n$$

Relazione tra le variabili x e z: la variabile $z(k,j)$ deve assumere il valore 0 quando almeno un vertice estremo dello spigolo k non appartiene all'insieme j:

$$|A(k)| \cdot z(k,j) \leq \sum_{i \in A(k)} x(i,j)$$

dove $A(k)$ è l'insieme dei vertici estremi dello spigolo k.

Taglio della partizione: può essere considerato nei vincoli o nella funzione obiettivo

$$\sum_{k=1}^{a} w(k) - \sum_{k=1}^{m} \sum_{j=1}^{p} w(k) z(k,j)$$

dove $w(k)$ è il peso dello spigolo k.

Vincoli di capacità:

$$INF \leq \sum_{i=1}^{n} w(i)x(i,j) \leq SUP \text{ per ogni } j = 1,\ldots,p$$

dove $w(i)$ è il peso del vertice i.

Nel caso in cui il numero delle componenti non sia fissato, tutte le classi di vincoli mantengono la stessa struttura tranne l'ultima, in cui si deve tenere conto dell'attivazione delle componenti:

$$y(j) \cdot INF \leq \sum_{i=1}^{n} w(i)x(i,j) \text{ per ogni } j = 1,\ldots,p$$

Il vincolo di connessione dei sottografi indotti dalle componenti V_j della partizione di $V(G)$ introduce delle complicazioni e le formulazioni presentano un numero esponenziale di variabili o di vincoli. Di seguito sono presentate due possibili formulazioni:

1. Per ogni cammino h di G e per ogni componente V_j si introduce la seguente variabile:

$$s(h,j) = \begin{cases} 1 & \text{se tutti i vertici del cammino } h \text{ appartengono a } V_j \\ 0 & \text{se almeno un vertice del cammino } h \text{ non appartiene a } V_j \end{cases}$$

   Per ogni coppia di vertici v_i e $v_{i'}$ appartenenti ad una stessa componente V_j, deve esistere un cammino $h : v_i \rightsquigarrow v_{i'}$ interno alla componente V_j. Se $P(v_i, v_{i'})$ è l'insieme dei cammini di G che connettono v_i e $v_{i'}$, il vincolo di connessione può essere formulato come segue:

$$\sum_{h \in P(v_i, v_{i'})} s(h,j) \geq x(i,j)x(i',j) \text{ per ogni } j = 1,\ldots,p \text{ e per ogni } i,i' = 1,\ldots,n$$

   Tale formulazione ha un numero polinomiale di vincoli e, poiché il numero di cammini tra due vertici può essere esponenziale in n, un numero esponenziale di variabili.

2. Per ogni coppia di vertici v_i e $v_{i'}$, si consideri la classe $\mathcal{V}_{ii'}$ di sottoinsiemi di V che contengono v_i e non contengono $v_{i'}$. Dato un insieme $V_{ii'} \in \mathcal{V}_{ii'}$ sia $\omega(V_{ii'})$ il *cociclo* di $V_{ii'}$, cioè l'insieme di spigoli di E che hanno esattamente un estremo in $V_{ii'}$. Affinché le componenti della partizione siano connesse, per ogni coppia di vertici v_i, $v_{i'}$ che appartengono alla stessa classe $\mathcal{V}_{ii'}$ devono avere almeno uno spigolo contenuto nella componente V_j. Si devono quindi considerare i seguenti vincoli:

$$\sum_{k \in \omega(V_{ii'})} z(k,j) \geq x(i,j)x(i',j) \;\; \forall\, j, \forall\, i,i', \forall\, V_{ii'} \in \mathcal{V}_{ii'}$$

   Tale formulazione ha un numero polinomiale di variabili e, poiché le classi $\mathcal{V}_{ii'}$ possono contenere un numero esponenziale di elementi, un numero esponenziale di equazioni.

6.2 Un algoritmo di shifting su alberi

Yehoshua Perl e Stephen Schach [30] hanno proposto un algoritmo in grado di risolvere in tempo polinomiale il problema del p-partizionamento di un albero con funzione obiettivo Max-Min (6). Si tratta di un ottimo esempio dell'adozione della tecnica dello *shifting* dei tagli per la risoluzione di un problema di partizionamento su alberi.

Ricordiamo che la funzione obiettivo Max-Min, da massimizzare, consente di ottenere un'equipartizione di un grafo con pesi assegnati ai vertici e può essere espressa come segue:

$$f(\pi_p) = \min_{k=1,\ldots,p} W(V_k)$$

indicando con $\pi_p = \{V_1, \ldots, V_k\}$ una generica p-partizione del grafo e con $W(V_k)$ il peso di una componente della partizione. Sia $T = (V,E)$ un albero con $n = |V(T)|$ vertici e $m = n-1$ spigoli con pesi assegnati ai vertici.

Una p-partizione π_p di T viene prodotta assegnando $p-1$ tagli $c_1, \ldots, c_{p-1}$ ad altrettanti spigoli distinti di T. Infatti, siccome T è un albero, come abbiamo già osservato in precedenza, occorrono esattamente $p-1$ tagli per produrre una partizione di T in p componenti connesse (sottoalberi di T).

Possiamo scegliere una foglia di T (un vertice con un solo spigolo incidente) e considerare tale vertice come la radice di T, imponendo successivamente un'orientazione *top-down* agli spigoli, dalla radice fino alle foglie. Sia $v_1 \in V(T)$ la radice di T. La *root component* di π_p è la componente che contiene la radice dell'albero; una "down component" D_{c_k} di un taglio c_k è la componente limitata dall'alto da c_k; indichiamo con $W(D_{c_k})$ la somma dei pesi associati ai vertici della componente D_{c_k}.

Se il taglio c_k è assegnato allo spigolo orientato $(v_i, v_j) \in E(T)$, chiameremo v_i la *coda* del taglio e, analogamente, v_j la *testa* del taglio. Uno *shift* di un taglio c_k è lo spostamento del taglio dallo spigolo (v_i, v_j) allo spigolo (v_j, v_h), ossia uno spostamento tale che v_j, testa del taglio c_k prima dell'operazione di shift, diventi coda del taglio in seguito allo spostamento.

L'algoritmo MAXMIN di Perl e Schach (Algoritmo 1) assegna inizialmente tutti i $p-1$ tagli all'unico spigolo incidente la radice v_1 di T; quindi esegue uno shift di un taglio c_k verso il basso, con l'obiettivo di aumentare il peso della componente di peso minimo e, al tempo stesso, di massimizzare il peso della *down component* D_{c_k} del taglio c_k, ottenuta in seguito allo spostamento.

Lo shift dei tagli verso il basso, sempre in una sola direzione, garantisce che uno stesso taglio non possa essere collocato due o più volte sullo stesso spigolo e, più in generale, che l'algoritmo ha termine dopo al più $(p-1)(n-p)$ operazioni di shift: ognuno dei $p-1$ tagli, infatti, può essere spostato al massimo, nel caso in cui l'albero T sia un cammino, su $n-p$ spigoli.

Algoritmo 1 MAXMIN(G, W, p)

Input: Un albero con radice $T = (V,E)$ e un insieme di pesi $W = \{w_1, \ldots, w_n\}$ assegnati ai vertici di T, il numero $0 < p \leq n$ di componenti della p-partizione di T

Output: Una p-partizione π_p di T tale da massimizzare la funzione obiettivo Max-Min

1: assegna tutti i tagli $c_1, \ldots, c_{p-1}$ all'unico spigolo incidente la radice di T
2: trova il peso $W_{\min}$ della componente di peso minimo
3: trova uno spigolo $e \in E(T)$ privo di tagli su cui eseguire lo *shift* di un taglio c collocato su uno spigolo incidente e tale da massimizzare il valore $W(D_c)$
4: se $W(D_c) \geq W_{\min}$ allora esegui lo *shift* di c su e e torna al passo 2
5: la p-partizione π_p ottenuta è ottimale; restituisci $f(\pi_p) = W_{\min}$

Al passo 1 l'algoritmo assegna i tagli $c_1, \ldots, c_{p-1}$ allo spigolo incidente la radice v_1 di T. Successivamente, per ogni iterazione del ciclo presente nelle righe 2–4, l'algoritmo cerca di migliorare il valore della funzione obiettivo (6), individuando di volta in volta la componente $V_{\min}$ di peso $W_{\min}$ minimo che determina il valore della funzione obiettivo e cercando di aumentare il peso di una componente eseguendo lo shift verso il basso di uno dei tagli che la limita inferiormente; in questo modo si includono altri vertici di peso non negativo nella componente aumentandone il peso a scapito di una componente adiacente. Dunque (riga 3) lo spigolo c_k da spostare in basso, viene scelto in modo

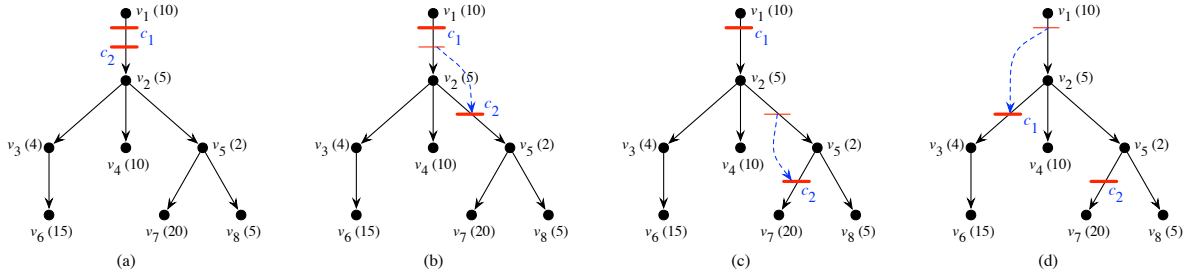


Figura 8: I passi di esecuzione dell'algoritmo MAXMIN per il partizionamento di un albero con $n = 8$ vertici in $p = 3$ sottoalberi

tale da ridurre il meno possibile il peso della *down component* $W(D_{c_k})$. L'algoritmo termina la sua esecuzione quando non è più possibile migliorare il valore della funzione obiettivo eseguendo uno shift verso il basso di un taglio (riga 4).

Ai passi 3 e 4, una volta individuata la componente di peso minimo $W_{\min}$, viene verificato l'esito che produrrebbe sul peso della *down component* corrispondente lo shift in basso di ogni taglio della partizione. Se esiste un possibile shift che, pur riducendo il peso della *down component*, produce una nuova partizione che non riduce il valore della funzione obiettivo, allora lo shift viene accettato ed eseguito effettivamente, ottenendo una nuova partizione di T.

In Figura 8 è riportato un esempio: si vuole produrre una p-partizione in $p = 3$ componenti connesse (sottoalberi) di un albero con $n = 8$ vertici. Inizialmente, con il passo 1 dell'algoritmo, i tagli c_1 e c_2 vengono posti sull'unico spigolo incidente la radice (Figura 8-a); quindi viene calcolato il peso della componente di peso minimo $W_{\min}$; la componente più leggera è la *down component* di c_1, il cui peso è pari a zero, dal momento che non contiene nessun vertice. Lo shift di un taglio che consente di produrre la *down component* più pesante è lo spostamento di c_2 dallo spigolo (v_1, v_2) allo spigolo (v_2, v_5); il peso della nuova *down component* di c_2 è $27 > W_{\min} = 0$, quindi lo shift viene accettato (passo 4), ottenendo una nuova partizione π_p dell'albero T rappresentata in Figura 8-b. In questo caso il peso della componente più leggera è $W_{\min} = 10$; il *down shift* di un taglio che produce la *down component* più pesante, è ancora lo spostamento del taglio c_2, questa volta dallo spigolo (v_2, v_5) allo spigolo (v_5, v_7); si ottiene così la nuova partizione rappresentata in Figura 8-c. Anche in questa partizione la componente più leggera è la *root component*, con $W_{\min} = 10$; questa volta è lo shift del taglio c_1 dallo spigolo (v_1, v_2) allo spigolo (v_2, v_3) a produrre la *down component* più pesante, il cui peso non è inferiore a $W_{\min}$; pertanto viene eseguito lo shift del taglio c_1, ottenendo la partizione rappresentata in Figura 8-d. La componente di peso minimo è la *down component* di c_1, con $W_{\min} = 19$; nessuno shift di un taglio consente di produrre una *down component* più pesante di $W_{\min}$, per cui ogni possibile shift produrrebbe un peggioramento del valore della funzione obiettivo Max-Min. Dunque l'algoritmo termina: la partizione π_p^t ottenuta è una partizione ottimale.

Le seguenti proposizioni sono dimostrate in [30]; entrambe le dimostrazioni, tecnicamente non difficili, richiedono però l'introduzione di concetti che per brevità non possiamo introdurre in queste pagine.

Proposizione 2. *L'algoritmo* MAXMIN *ha una complessità computazionale, nel caso peggiore, di* $O((p-1)^2 \cdot R(T) + (p-1)m)$*, dove con* $R(T) = \min_{v_i} \max_{v_j} d(v_i, v_j)$ *si è indicato il numero di spigoli nel raggio di* T.

Proposizione 3. *La partizione π_p^t ottenuta mediante l'algoritmo* MAXMIN *è una partizione ottimale per la funzione obiettivo Max-Min, ossia*

$$f(\pi_p^t) = \max_{\pi_p \in \Pi_p(T)} \min_{k=1,\ldots,p} W(V_k)$$

È interessante osservare che l'Algoritmo 1 non può essere adattato semplicemente per la risoluzione del problema del partizionamento di alberi con una funzione obiettivo differente; in altri termini le funzioni obiettivo presentate nelle pagine precedenti, pur essendo apparentemente simili tra loro, nascondono delle differenze sostanziali che non consentono di adottare il medesimo approccio nella individuazione di un algoritmo risolutore efficiente. Ad esempio, gli stessi Perl e Schach, insieme a Ronald Becker, in [9] hanno proposto un algoritmo per la risoluzione del problema per il partizionamento di alberi con funzione obiettivo Min-Max, ben diverso e più complicato rispetto all'Algoritmo presentato nelle pagine precedenti per il problema analogo con funzione obiettivo Max-Min.

Nella sezione seguente presentiamo un algoritmo proposto in [26] per la soluzione del problema del partizionamento con funzione obiettivo la norma L_∞: tale algoritmo è applicabile solo al caso particolare dei cammini, mentre non è applicabile al caso più generale di alberi generici.

6.3 Un algoritmo di shifting su cammini

Sia $G = (V,E)$ un cammino: senza perdita di generalità possiamo supporre che $V = \{1,\ldots,n\}$ ed $E = \{(i,i+1), i = 1,\ldots,n-1\}$; nel seguito indicheremo come di consueto con P_n il cammino con n vertici. Ai vertici del cammino sono assegnati dei pesi non negativi $W = \{w_1,\ldots,w_n\}$. Si vuole produrre una partizione del cammino in p componenti connesse, $0 < p \leq n$, ossia in p sotto-cammini, tale da ottimizzare la funzione obiettivo Norma L_∞ (5), che per semplicità di lettura riportiamo nuovamente di seguito:

$$f(\pi_p) = \max_{k=1,\ldots,p} |W(V_k) - \mu|$$

dove con $V_k \subseteq V$ abbiamo indicato una generica componente della p-partizione del cammino, con $W(V_k)$ la somma dei pesi dei vertici della componente V_k, $W(V_k) = \sum_{v \in V_k} w(v)$, e con μ il peso medio di una componente di una p-partizione π_p del cammino: $\mu = \frac{1}{p}\sum_{v \in V} w(v)$.

Il problema di equipartizione del cammino chiede di individuare una p-partizione π_p^* che minimizzi il valore della funzione obiettivo:

$$f(\pi_p^*) = \min_{\pi_p \in \Pi_p(G)} f(\pi_p) = \min_{\pi_p \in \Pi_p(G)} \max_{k=1,\ldots,p} |W(V_k) - \mu|$$

Naturalmente, come nel caso più generale del partizionamento di alberi, anche in questo caso la p-partizione del cammino si ottiene rimuovendo $p-1$ spigoli distinti, ovvero assegnando $p-1$ tagli ad altrettanti spigoli del cammino.

Per comprendere il funzionamento dell'algoritmo che presenteremo nelle pagine seguenti e per visualizzare il processo di dimostrazione della correttezza dell'algoritmo stesso, è opportuno osservare che possiamo costruire una rete $\mathcal{N}$ in cui ogni cammino dalla sorgente al pozzo rappresenta una possibile p-partizione del cammino.

La rete $\mathcal{N} = (V_\mathcal{N}, E_\mathcal{N})$ è un grafo in cui ogni vertice è costituito da una coppia (h,k) e i cui spigoli sono quindi coppie di coppie. Il grafo può essere costruito ponendo

$$\begin{aligned}
V_\mathcal{N} &= \{(0,0)\} \cup \{(h,k) :\ 0 < h < p,\ h \leq k \leq h+n-p\} \cup \{(p,n)\} \\
E_\mathcal{N} &= \{((0,0),(1,k)) :\ 1 \leq k \leq 1+n-p\} \cup \\
&\quad \{((h,k),(h+1,k+1)),\ldots,((h,k),(h+1,h+1+n-p)) :\ 1 \leq h \leq p-1,\ h \leq k \leq h+n-p\} \cup \\
&\quad \{((p-1,k),(p,n)) :\ n-p+1 \leq k \leq n-1\}
\end{aligned}$$

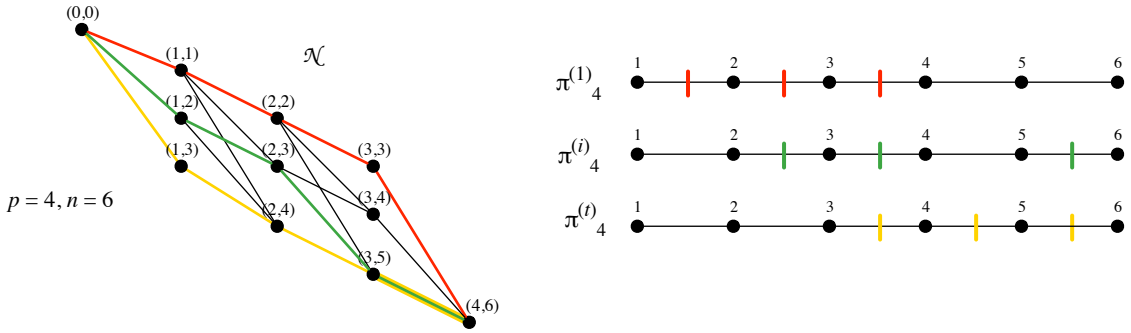


Figura 9: La rete $\mathcal{N}$ corrispondente alle p-partizioni in $p = 4$ componenti connesse di un cammino con $n = 6$ vertici; in figura sono evidenziati i cammini corrispondenti alla prima partizione $\pi_4^{(1)}$, ad una generica partizione $\pi_4^{(i)}$ intermedia e all'ultima partizione $\pi_4^{(t)}$

Il disegno riportato in Figura 9 aiuta a visualizzare il grafo $\mathcal{N}$ costruito nel caso $n = 6$ e $p = 4$. Ad ogni cammino dalla sorgente $s = (0,0)$ al pozzo $t = (p,n)$ corrisponde una possibile p-partizione di un cammino con n vertici. La corrispondenza viene creata indicando che se il cammino passa per il vertice $(h,k) \in V_{\mathcal{N}}$ allora nella partizione i primi k vertici del cammino sono distribuiti nelle prime h componenti.

Facendo riferimento all'esempio riportato nella Figura 9, il cammino colorato di verde che passa per i vertici $(1,2)$, $(2,3)$, $(3,5)$ e $(4,5)$ è associato alla partizione $\pi''_4 = (V_1, V_2, V_3, V_4)$ del cammino P_6 in cui i primi due vertici sono contenuti nella componente V_1, il terzo vertice è contenuti in V_2, il quarto e quinto vertice sono contenuti in V_3 ed infine il sesto vertice è contenuto in V_4.

Sfruttando questa rappresentazione si definisce così un ordinamento parziale tra le partizioni del cammino P_n, che in questo modo vengono ad assumere una struttura di *reticolo*, in cui il minimo, la prima partizione che indichiamo con $\pi_p^{(1)}$, è quella in cui i $p-1$ tagli sono assegnati ai primi $p-1$ spigoli; mentre il massimo, l'ultima partizione, indicata con $\pi_p^{(t)}$, è quella in cui i $p-1$ tagli sono assegnati agli ultimi $p-1$ spigoli dal cammino. Una partizione $\pi_p^{(i)}$ è "minore" di una partizione $\pi_p^{(j)}$ se sulla rete $\mathcal{N}$ il cammino corrispondente a $\pi_p^{(i)}$ non va mai "al di sotto" del cammino corrispondente a $\pi_p^{(j)}$. Se invece due cammini si intersecano allora non è possibile stabilire un ordine reciproco tra le partizioni corrispondenti; per questo motivo l'ordinamento dell'insieme delle partizioni di P_n è parziale.

In [26] è stato proposto l'algoritmo PATHSHIFTING descritto nelle pagine seguenti. La strategia è quella di disporre i tagli su uno spigolo fittizio aggiunto all'inizio del cammino e di farli poi "scivolare" da sinistra verso destra cercando di ridurre, di volta in volta, lo scarto dalla media della componente della partizione il cui scarto dalla media è massimo in valore assoluto. Se il massimo scarto dalla media $W(V_k) - \mu$ è positivo, allora l'algoritmo proverà a ridurre il peso della componente V_k spostando il primo vertice nella componente V_{k-1}: questo avviene spostando di uno spigolo verso destra il taglio che limita da sinistra la componente V_k; se invece il massimo scarto dalla media $W(V_k) - \mu$ è negativo, allora l'algoritmo proverà ad aumentare il peso della componente V_k allargando la componente includendo anche il primo vertice della componente successiva V_{k+1}, spostando a destra di uno spigolo il taglio che separa le componenti V_k e V_{k+1}. L'algoritmo termina quando si verifica una delle seguenti condizioni di stop:

1. la componente V_k di scarto massimo dalla media è composta da un solo vertice e risulta $W(V_k) -$

$\mu > 0$: in questo caso è stata trovata una partizione che ha isolato in una componente un vertice di peso talmente elevato da costituire da solo la componente di massimo scarto dalla media; l'algoritmo termina perché non è possibile migliorare il valore della funzione obiettivo, visto che non è possibile ridurre ulteriormente il peso di questa componente;

2. la componente di scarto massimo dalla media è V_1, la prima, ed ha scarto positivo ($W(V_1) - \mu > 0$); l'algoritmo termina perché in questo caso non è possibile ridurre il peso di V_1 spostando il primo vertice nella componente precedente, perché la componente precedente non esiste;

3. la componente di scarto massimo dalla media è V_p, l'ultima, ed ha scarto negativo ($W(V_1) - \mu < 0$); l'algoritmo termina perché anche in questo caso non è possibile aumentare il peso di V_p includendo il primo vertice della componente successiva, perché la componente successiva non esiste.

L'Algoritmo 2 presenta una pseudo-codifica dell'algoritmo PATHSHIFTING proposto in [26]. Per comprendere meglio il funzionamento dell'algoritmo, applichiamolo ad un caso molto semplice. Consideriamo il cammino $P_5 = (V,E)$, con $V = \{1,2,3,4,5\}$ ed $E = \{(1,2),(2,3),(3,4),(4,5)\}$. I pesi positivi assegnati ai vertici del cammino sono $W = (10,13,7,10,8)$. Supponiamo di voler ottenere una p-partizione di P_5 in tre componenti: $p = 3$. Dunque il peso medio di una componente della partizione è $\mu = \frac{10+13+7+10+8}{3} = 16$.

Le partizioni del cammino si ottengono assegnando due tagli agli spigoli del cammino stesso. Il cammino P_5 viene esteso aggiungendo il vertice 0 e lo spigolo $(0,1)$. Inizialmente tutti i tagli sono assegnati allo spigolo $(0,1)$ (vedi Figura 10), creando la prima p-partizione $\pi_p^{(1)} = (V_1^{(1)}, V_2^{(1)}, V_3^{(1)})$, con $V_1^{(1)} = V_2^{(1)} = \emptyset$ e $V_3^{(1)} = V$. La componente di scarto massimo dalla media è la terza, per cui risulta $W(V_3^{(1)}) - \mu = 48 - 16 = 32$, mentre $W(V_1^{(1)}) - \mu = W(V_2^{(1)}) - \mu = -16$. Per cui l'algoritmo provvede a ridurre il peso della componente V_3 spostando verso destra il taglio che la limita da sinistra.

In questo modo si ottiene la partizione $\pi_p^{(2)} = (V_1^{(2)}, V_3^{(2)}, V_3^{(2)})$, con $V_1^{(2)} = \emptyset$, $V_2^{(2)} = \{1\}$ e $V_3^{(2)} = \{2,3,4,5\}$. Anche in questo caso la componente di scarto massimo dalla media in valore assoluto è la terza, per cui risulta $W(V_3^{(2)}) - \mu = 22$, per cui verrà spostato verso destra il taglio che la limita da sinistra, in modo da ridurre il peso della componente V_3 ed aumentare il peso della componente V_2.

La sequenza di operazioni di *shift* dei tagli effettuata dall'algoritmo PATHSHIFTING è riportata di seguito; sopra a ciascuna componente della partizione è riportato lo scarto dalla media μ:

$$
\begin{aligned}
\pi_p^{(1)} &: \overbrace{\emptyset}^{-16} \mid \overbrace{\emptyset}^{-16} \mid \overbrace{10\ 13\ 7\ 10\ 8}^{32} && f(\pi_p^{(1)}) = 32 \\
\pi_p^{(2)} &: \overbrace{\emptyset}^{-16} \mid \overbrace{10}^{-6} \mid \overbrace{13\ 7\ 10\ 8}^{22} && f(\pi_p^{(2)}) = 22 \\
\pi_p^{(3)} &: \overbrace{\emptyset}^{-16} \mid \overbrace{10\ 13}^{7} \mid \overbrace{7\ 10\ 8}^{9} && f(\pi_p^{(3)}) = 16 \\
\pi_p^{(4)} &: \overbrace{10}^{-6} \mid \overbrace{13}^{-3} \mid \overbrace{7\ 10\ 8}^{9} && f(\pi_p^{(4)}) = 9 \\
\pi_p^{(5)} &: \overbrace{10}^{-6} \mid \overbrace{13\ 7}^{4} \mid \overbrace{10\ 8}^{2} && f(\pi_p^{(5)}) = 6 \implies \pi_p^{(5)} = \pi_p^* \\
\pi_p^{(6)} &: \overbrace{10\ 13}^{7} \mid \overbrace{7}^{-9} \mid \overbrace{10\ 8}^{2} && f(\pi_p^{(6)}) = 9 \\
\pi_p^{(7)} &: \overbrace{10\ 13}^{7} \mid \overbrace{7\ 10}^{-1} \mid \overbrace{8}^{-8} && f(\pi_p^{(7)}) = 8
\end{aligned}
$$

Algoritmo 2 PATHSHIFTING(P_n, W, p)

Input: Il cammino P_n con n vertici, i pesi W associati ai vertici di P_n, il numero di componenti p
Output: Una p-partizione π_p^* di P_n tale da minimizzare la Norma L_∞

1: assegna $p-1$ tagli allo spigolo aggiuntivo $(0, v_1)$
2: sia V_k la componente di scarto massimo dalla media (il valore $|W(V_k) - \mu|$ è massimo)
3: sia max $= |W(V_k) - \mu|$, $\pi_p^* = \pi_p$
4: $stop = 0$
5: **fintanto che** $stop = 0$ **ripeti**
6: **se** $|V_k| = 1$ e $W(V_k) - \mu > 0$ **allora**
7: $stop = 1$
8: **altrimenti**
9: **se** $W(V_k) - \mu > 0$ **allora**
10: **se** $k = 1$ **allora**
11: $stop = 1$
12: **altrimenti**
13: sposta il primo elemento di V_k in V_{k-1} eseguendo lo *shift* a destra del taglio che separa le componenti V_{k-1} e V_k
14: **fine-condizione**
15: **altrimenti se** $W(V_k) - \mu < 0$ **allora**
16: **se** $k = p$ **allora**
17: $stop = 1$
18: **altrimenti**
19: sposta il primo elemento di V_{k+1} in V_k eseguendo lo *shift* a destra del taglio che separa le componenti V_k e V_{k+1}
20: **fine-condizione**
21: **fine-condizione**
22: sia π_p la nuova partizione; sia V_k la componente di π_p di scarto massimo dalla media
23: **se** $|W(V_k) - \mu| <$ max **allora**
24: max $= |W(V_k) - \mu|$, $\pi_p^* = \pi_p$
25: **fine-condizione**
26: **fine-condizione**
27: **fine-ciclo**
28: la p-partizione π_p^* ottenuta è ottimale; restituisci π_p^*

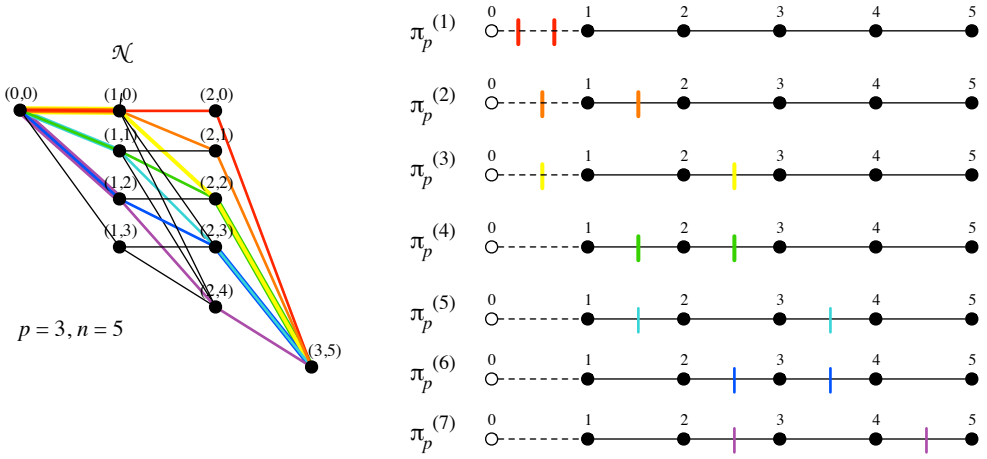


Figura 10: La rete $\mathcal{N}$ definita per rappresentare le p-partizioni del cammino P_5 prodotte durante l'esecuzione dell'algoritmo PATHSHIFTING spostando i tagli da sinistra verso destra.

L'algoritmo termina sulla partizione $\pi_p^{(7)}$ in cui è verificata la terza delle condizioni di stop esposte nelle pagine precedenti: infatti la componente di massimo scarto dalla media in valore assoluto è la terza, che ha scarto negativo $W(V_3^{(7)}) - \mu = -8$; per proseguire l'algoritmo dovrebbe quindi migliorare il valore della funzione obiettivo aumentando il peso della componente $V_3^{(7)}$ spostando verso destra il taglio che la limita a destra; ma la componente non ha alla sua destra nessun'altra componente, per cui non c'è possibilità di migliorare il valore della funzione obiettivo e dunque l'algoritmo termina.

Come si osserva nel semplice esempio appena discusso, l'algoritmo PATHSHIFTING esegue lo *shift* di un taglio anche se questo può portare ad un peggioramento del valore della funzione obiettivo; in questo modo riesce a non farsi ingannare da eventuali "minimi locali" del valore della funzione obiettivo. Ad esempio, nel caso precedente, nei passaggi che portano dalla partizione $\pi_p^{(5)}$ alla partizione $\pi_p^{(6)}$ e poi da questa alla partizione $\pi_p^{(7)}$, il valore $f(\pi_p)$ della funzione obiettivo viene peggiorato ogni volta. Il valore minimo è infatti quello che si ottiene con la partizione $\pi_p^{(5)}$, che risulta ottimale ed è appunto la soluzione prodotta dall'algoritmo.

Proposizione 4. *La complessità dell'algoritmo* PATHSHIFTING *per il partizionamento di un cammino P_n in p componenti connesse, nel caso peggiore è $O(np\log_2 p)$.*

Dimostrazione. L'Algoritmo 2 ad ogni iterazione del ciclo principale (righe 5–27) esegue la ricerca della componente V_k di scarto massimo in valore assoluto dalla media μ ed esegue quindi un'operazione di *shift* a destra di uno dei $p-1$ tagli. I tagli inizialmente sono tutti assegnati allo spigoli aggiuntivo $(0,1)$ e vengono spostati solo da sinistra verso destra; per cui ogni taglio eseguirà al più $O(n)$ *shift*; quindi il numero di iterazioni del ciclo principale è al più $O(np)$.

Ad ogni iterazione del ciclo, però, deve essere individuata la componente di scarto massimo; se il valore degli scarti dalla media in valore assoluto per ciascuna componente viene memorizzato in una struttura dati di *heap*, l'identificazione dell'elemento massimo ha costo costante unitario $O(1)$ e la ricostruzione dell'*heap* ha un costo al più logaritmico nel numero di elementi dell'*heap*: $O(\log_2 p)$.

In conclusione, l'Algoritmo PATHSHIFTING esegue al massimo $O(np\log_2 p)$ operazioni. □

Per dimostrare che l'algoritmo PATHSHIFTING è corretto e produce effettivamente una partizione ottimale, identifichiamo ogni p-partizione di P_n con la sequenza $0 = s_0 \leq s_1 \leq \cdots \leq s_p = n$, con

$s_k = i$ se e solo se il taglio c_k è sullo spigolo $(i, i+1)$. Date due partizioni π'_p e π''_p di P_n definite dai tagli $(c'_1, \ldots, c'_{p-1})$ e $(c''_1, \ldots, c''_{p-1})$ rispettivamente. Si dice che π'_p è **sopra** π''_p se $c'_k \leq c''_k$ per ogni $k = 1, \ldots, p-1$, ossia se $s'_k \leq s''_k$ per ogni $k = 1, \ldots, p$.

Il seguente Lemma dimostra che le condizioni di stop che determinano l'interruzione dell'esecuzione dell'algoritmo sono efficaci: ossia, tutte le partizioni che potrebbero essere calcolate dall'algoritmo a partire dall'ultima delle partizioni prodotte, non migliorano il valore della funzione obiettivo.

Lemma 1. *Sia $\pi_p^{(t)}$ la partizione finale su cui ha termine l'algoritmo* PATHSHIFTING. *Tutte le partizioni π_p sotto $\pi_p^{(t)}$ sono tali che $f(\pi_p) \geq f(\pi_p^{(t)})$.*

Dimostrazione. Sia $V_k^{(t)}$ la componente di $\pi_p^{(t)}$ con il massimo scostamento dalla media in valore assoluto, che determina il valore di $f(\pi_p^{(t)})$. Sia π_p una generica partizione sotto $\pi_p^{(t)}$ e sia V_k la sua componente k-esima. Se $V_k = V_k^{(t)}$ allora ovviamente risulta che $f(\pi_p) \geq f(\pi_p^{(t)})$.

Sia $V_k \neq V_k^{(t)}$. La componente $V_k^{(t)}$ è quella su cui si è verificata una delle tre condizioni di stop con cui termina l'algoritmo. Se $V_k^{(t)} = \{v_i\}$, allora $V_k^{(t)}$ è la più pesante delle componenti di $\pi_p^{(t)}$ e siccome v_i deve appartenere anche a qualche componente di π_p, allora sicuramente risulta che $f(\pi_p) \geq f(\pi_p^{(t)})$, perché il peso di v_i sommato a quello di altri vertici (tutti i pesi sono positivi), non potrà che aumentare ulteriormente il peso della componente di π_p a cui appartiene v_i.

Supponiamo che $k = 1$, ossa i che $V_k^{(t)}$ sia la prima componente di $\pi_p^{(t)}$, quindi, anche in questo caso, la componente più pesante: risulta infatti che $W(V_1^{(t)}) - \mu \geq 0$. Sia V_1 la prima componente di π_p; se π_p è sotto $\pi_p^{(t)}$, deve essere $V_1^{(t)} \subset V_1$, quindi $f(\pi_p) \geq f(\pi_p^{(t)})$.

Sia infine $k = p$, cioè sia $V_k^{(t)}$ l'ultima componente di $\pi_p^{(t)}$; in questo caso quindi $V_p^{(t)}$ è la più leggera ed ha scarto negativo dalla media: $W(V_p^{(t)}) - \mu \leq 0$. Allora se π_p è sotto $\pi_p^{(t)}$, risulta $V_p \subset V_p^{(t)}$ e quindi $f(\pi_p) \geq f(\pi_p^{(t)})$. □

L'algoritmo PATHSHIFTING non esegue una ricerca esaustiva della partizione ottimale, tuttavia utilizza un metodo per passare da una determinata partizione $\pi_p^{(q)}$ alla successiva, $\pi_p^{(q+1)}$, che consente di scartare numerose partizioni, senza però "scavalcare" una eventuale partizione ottimale. Questa proprietà viene dimostrata nel seguente Lemma.

Lemma 2. *Siano $\pi_p^{(1)}, \pi_p^{(2)}, \ldots, \pi_p^{(t)}$ le partizioni generate dall'algoritmo* PATHSHIFTING. *Sia π_p^* una partizione ottimale. Se π_p^* è sotto a $\pi_p^{(q)}$, ma non è sotto $\pi_p^{(q+1)}$, allora $\pi_p^{(q)}$ è ottima.*

Dimostrazione. Sia $\pi_p^* = \{V_1^*, \ldots, V_p^*\}$ la partizione ottimale che è al di sotto di $\pi_p^{(q)}$, ma non al di sotto di $\pi_p^{(q+1)}$. Tra $\pi_p^{(q)}$ e $\pi_p^{(q+1)}$ c'è solo un vertice di differenza in due componenti contigue, ossia i tagli delle due partizioni coincidono tranne uno, che è spostato a destra di uno spigolo. Sia $V_k^{(q)}$ la componente di $\pi_p^{(q)}$ di scarto massimo dalla media.

Se $V_k^{(q)}$ è la componente più pesante della partizione $\pi_p^{(q)}$, allora il taglio s_k che la limita da sinistra sarà $s_k + 1$ in $\pi_p^{(q+1)}$. Se invece $V_k^{(q)}$ è la componente più leggera, allora il taglio s_{k+1} che la limita a destra sarà $s_{k+1} + 1$ in $\pi_p^{(q+1)}$.

Quindi se π_p^* non è al di sotto di $\pi_p^{(q+1)}$, ma è al di sotto di $\pi_p^{(q)}$, deve essere necessariamente $\pi_p^{(q)} = \pi_p^*$. □

Basandosi sulle proprietà dimostrate nel Lemma 1 e nel Lemma 2, il seguente Teorema afferma che l'Algoritmo PATHSHIFTING individua una p-partizione ottimale del cammino P_n.

Teorema 1. *Almeno una delle partizioni* $\pi_p^{(1)}, \ldots, \pi_p^{(t)}$ *generate dall'algoritmo* PATHSHIFTING *è ottima per la funzione obiettivo norma* L_∞.

Dimostrazione. Sia π_p^* una p-partizione ottimale di P_n. La partizione $\pi_p^{(1)}$ per costruzione è al di sopra di ogni possibile p-partizione del cammino. Se una partizione è al di sopra di π_p^* allora tutte le partizioni generate prima di questa sono al di sopra di π_p^* per transitività.

Sia $m = \max\{r : \pi_p^{(r)} \text{ è sopra } \pi_p^*\}$. Dimostriamo che allora $\pi_p^{(m)}$ è ottima. Per definizione di m c'è almeno una partizione ottima sotto $\pi_p^{(m)}$.

Sia $m = t$. Allora $f(\pi_p^{(m)}) \leq f(\pi_p^*)$ per il Lemma 1 e quindi $\pi_p^{(m)}$ è ottima.

Sia $m < t$; allora per la massimalità di m, π_p^* non è sotto $\pi_p^{(m+1)}$ e quindi, per il Lemma 2, $\pi_p^{(m)}$ è ottima. □

Riferimenti bibliografici

[1] A. Aletà, J. Codina, J. Sánchez, A. González, *Graph-partitioning based instruction scheduling for clustered processors*, Proceedings of the 34^{th} International Symposium on Microarchitecture, 2001.

[2] G. Andreatta, F. Mason, *Path covering problems and testing of printed circuits*, Discrete Applied Mathematics 62, 5-13, 1995.

[3] Enzo L. Aparo, Bruno Simeone, *Un algoritmo di equipartizione e il suo impiego in un problema di contrasto ottico*, Estratto da Ricerca Operativa, N. 6, 1973, pp. 1-12.

[4] Claudio Arbib, *A polynomial characterization of some graph partitioning problems*, Information Processing Letters, 26, 1987/88, pp. 223-230.

[5] Claudio Arbib, Mario Lucertini, Sara Nicoloso, *Problemi di partizione nel Layout ottimo di circuiti integrati*, Ricerca Operativa, 49, 1989, pp. 3-54.

[6] Ronald I. Becker, Y-I. Chiang, Bruno Simeone, *A shifting algorithm for continuous tree partitioning*, Theoretical Computer Science 282, 353-380, 2002.

[7] Ronald I. Becker, Isabella Lari, Mario Lucertini, Bruno Simeone, *Max-min partitioning of grid graphs into connected components*, Networks 32, 155-125, 1998.

[8] Ronald I. Becker, Yehoshua Perl, *The shifting algorithm technique for the partitioning of trees*, Discrete Applied Mathematics 62, 15-34, 1995.

[9] Ronald I. Becker, Stephen R. Schach, Yehoshua Perl, *A shifting algorithm for min-max tree partitioning*, Journal of the Association for Computing Machinery, Vol. 29, No. 1, January 1982, pp. 58-67.

[10] Michal Benelli, Refael Hassin, *Optimal separable partitioning in the plane*, Discrete Applied Mathematics, 59, 1995, pp. 215-224.

[11] K.-M. Chao, S. T. Kuan, H.-L. Wang, B. Y. Wu, *On the uniform edge-partition of a tree*, Discrete Applied Mathemetics 155, 1213-1223, 2007.

[12] Francesco Conti, Federico Malucelli, Sara Nicoloso, Bruno Simeone, *On a 2-dimensional equipartition problem*, preprint.

[13] Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest, Clifford Stein, *Introduzione agli algoritmi e strutture dati*, seconda edizione, McGraw-Hill, 2005.

[14] P. F. Cortese, G. Di Battista, *Clustered planarity*, Proceedings of the 21^{st} ACM Symposium on Computational Geometry, 32-34, 2005.

[15] Caterina De Simone, Mario Lucertini, Stefano Pallottino, Bruno Simeone, *Fair Disconnections of Spiders, Worms and Caterpillars*, Networks, Vol. 20, 1990, pp. 323-344.

[16] Irene Finocchi, Rossella Petreschi, *Divider-based algorithms for hierarchical tree partitioning*, Discrete Applied Mathematics 136, 227-247, 2004.

[17] Michael R. Garey, David S. Johnson, *Computers and Intractability: A Guide to the Theory of NP-Completeness*, Freeman, San Francisco, CA, 1979.

[18] B. Gopalakrishnan, E. L. Johnson, *Airline crew scheduling: state-of-the-art*, Annals of Operations Research 140, 305-337, 2005.

[19] Martin Grötschel, George L. Nemhauser, *A polynomial algorithm for the max-cut problem on graphs without long odd cycles*, Mathematical Programming, 29, 1984, pp. 28-40.

[20] Scott W. Hadley, *Approximation techniques for hypergraph partitioning problems*, Discrete Applied Mathematics, 59, 1995, pp. 115-127.

[21] N. Halman, A. Tamir, *Continuous bottleneck tree partitioning problems*, Discrete Applied Mathematics 140, 185-206, 2004.

[22] B. Hendrickson, T. G. Kolda, *Graph partitioning models for parallel computing*, Parallel Computing, Systems & Applications 26, 1519-1534, 2000.

[23] B. Hendrickson, R. Leland, *A multilevel algorithm for partitioning graphs*, Parallel Computing, Systems & Applications 26, 1519-1534, 2000.

[24] S. Kundu, J. Misra, *A linear tree partitioning algorithm*, SIAM Journal of Computing, 6, 1977, pp. 151-154.

[25] Isabella Lari, *Partizioni ottime di grafi a griglia*, Tesi di Dottorato in Ricerca Operativa, 1994, Università di Roma "La Sapienza", Dipartimento di Statistica, Probabilità e Statistiche Applicate.

[26] Marco Liverani, Aurora Morgana, Bruno Simeone, Gianni Storchi, *Path equipartition in the Chebyshev norm*, European Journal of Operational Research, 123, 2000, pp. 428-436.

[27] Mario Lucertini, Yehoshua Perl, Bruno Simeone, *Most uniform path partitioning and its use in image processing*, Discrete Applied Mathematics, 42, 1993, pp. 227-256.

[28] Maurizio Maravalle, Rossella Naldini, Bruno Simeone, *Clustering on trees*, Computational Statistics & Data Analysis 24, 217-234, 1997.

[29] Takao Nishizeki, Svatopluk Poljak, *k-Connectivity and decomposition of graphs into forests*, Discrete Applied Mathematics, 55, 1994, pp. 295-301.

[30] Yehoshua Perl, Stephen R. Shach, *Max-min tree partitioning*, Journal of the Association for Computing Machinery, Vol. 28, No. 1, January 1981, pp. 5-15.

[31] Yehoshua Perl, Uzi Vishkin, *Efficient implementation of a shifting algorithm*, Discrete Applied Mathematics, 12, 1985, pp. 71-80.

[32] Erich Prisner, *Clique covering and clique partition in generalizations of line graphs*, Discrete Applied Mathematics, 56 1995, pp. 93-98.

[33] M. R. Rao, *Cluster Analysis and Mathematical Programming*, Journal of the American Statistical Association, Vol. 66, N. 335, 1971, pp. 622-626.

[34] Sartaj Sahni, Teofilo Gonzales, *P-Complete Approximation Problems*, Journal of the Association for Computing Machinery, Vol. 23, No. 3, July 1976, pp. 555-565.

[35] Bruno Simeone, *Optimal connected partitions of graphs. Selected topics in large scale discrete optimization*, Cycle of seminars at the DIMACS Center, Rutgers University, 1999.

[36] Stephen W. Wharton, *A generalized histogram clustering scheme for multidimensional image data*, Pattern Recognition, Vol. 16, N. 2, 1983, pp. 193-199.

[37] Jing-Ho Yan, Gerard J. Chang, *The path-partition problem in block graphs*, Information Processing Letters, 52, 1994, pp. 317-322.