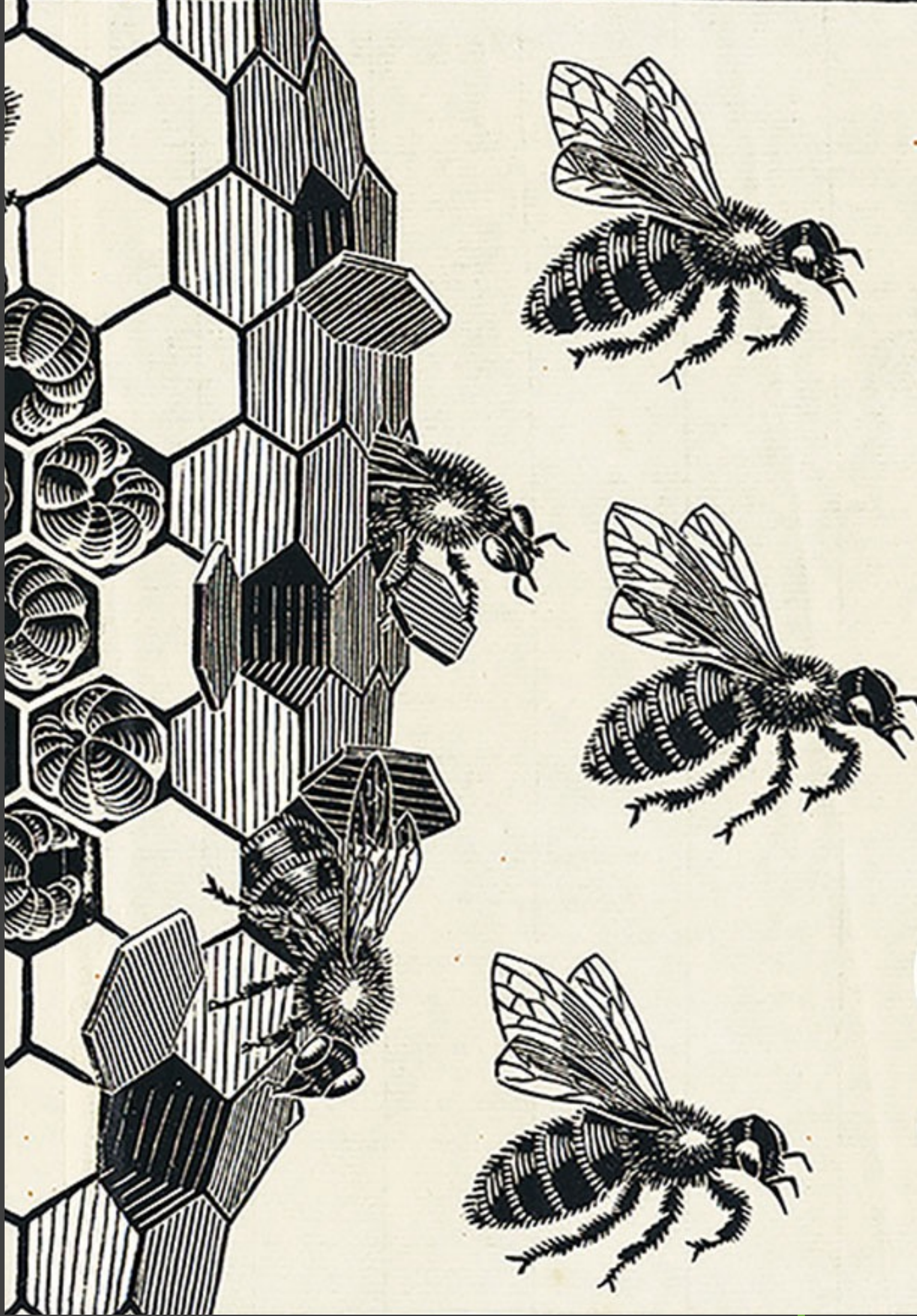
Università Roma Tre – Dipartimento di Matematica e Fisica
IN440 – Ottimizzazione Combinatoria
Prof. Marco Liverani

Partizionamento di grafi in componenti connesse

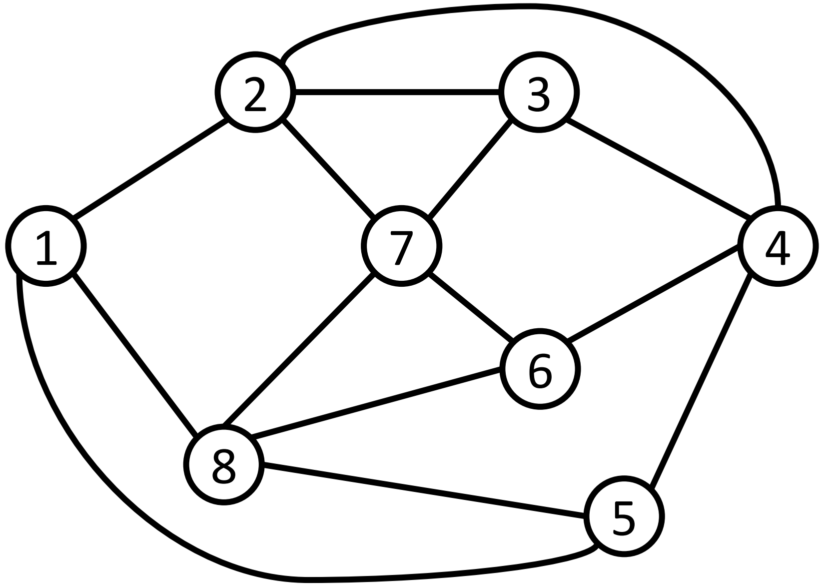


Partizionamento di insiemi e di grafi

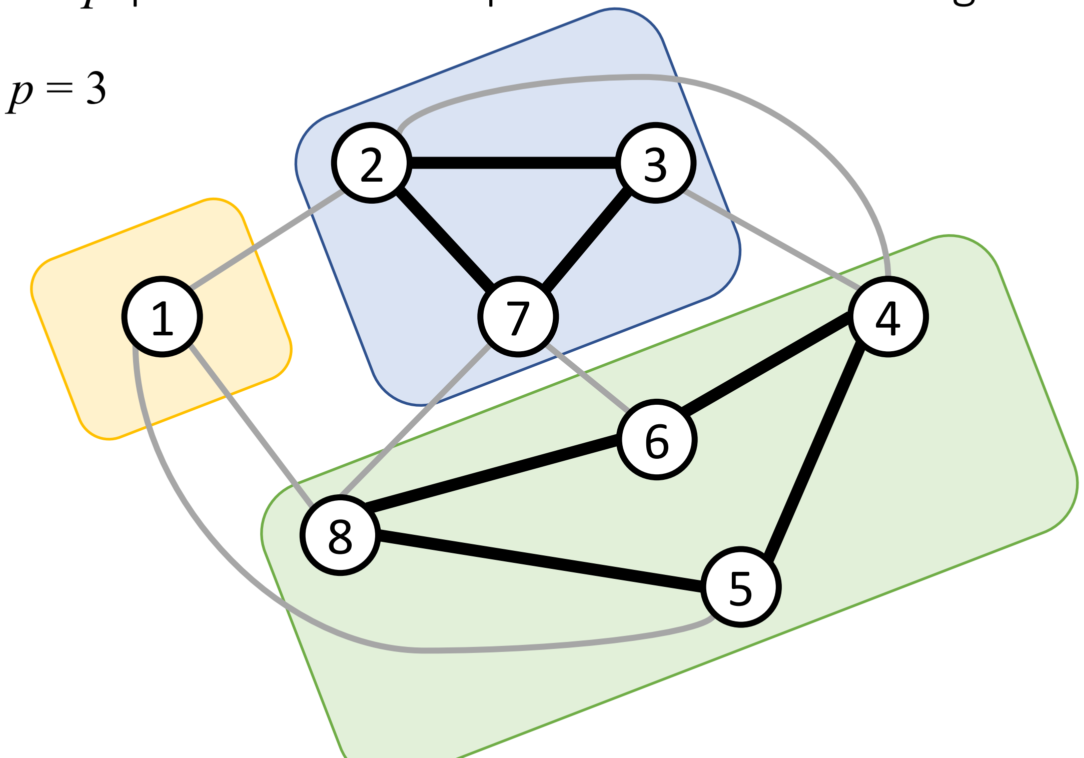
- SUBSET-SUM e SET-PARTITION sono problemi NP-completi ben noti
- Tuttavia se si aggiungono vincoli alla modalità con cui costruire i sottoinsiemi, lo **spazio delle soluzioni ammissibili si riduce** e in determinati casi si arriva a problemi polinomiali
- I problemi di partizionamento di grafi in componenti connesse sono problemi analoghi, ma con l'aggiunta del vincolo che ciascuna componente della partizione sia un sottografo connesso di G
- Possono quindi essere essere affrontati e risolti con algoritmi esatti di complessità polinomiale, almeno per alcune classi di grafi

Partizionamento di grafi in componenti connesse

- Dato un grafo $G = (V, E)$ e un intero $p > 0$, con $p \leq |V|$, si chiede di individuare una **p-partizione** π_p di V in p componenti $\pi_p = \{V_1, \ldots, V_p\}$ tali che $V_1 \cup \ldots \cup V_p = V$, $V_i \cap V_j$ per ogni $i, j = 1, \ldots, p$ con $i \neq j$
- La p-partizione deve essere tale che il sottografo di G indotto da V_k, $G[V_k]$, sia connesso per ogni $k = 1, \ldots, p$
- La p-partizione di G è **propria** se $V_k \neq \emptyset$ per ogni k
- Indichiamo con $\prod_p(G)$ l'insieme di tutte le p-partizioni in componenti connesse del grafo G



$p = 3$

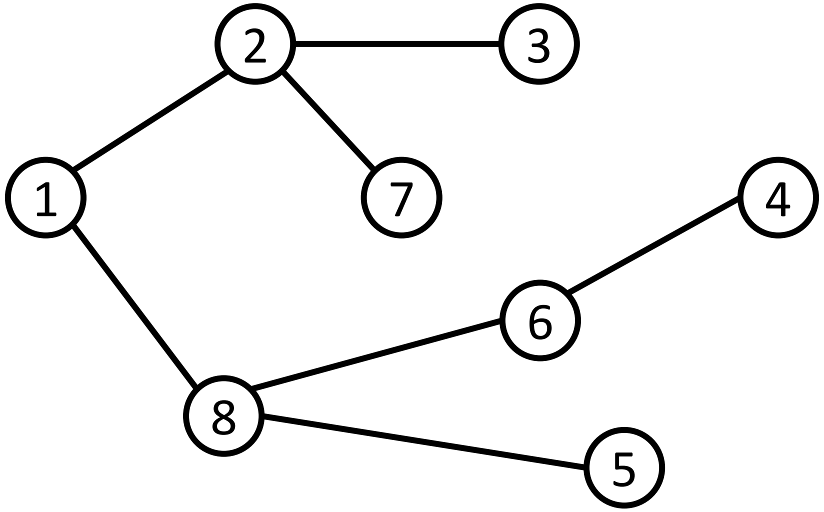


$\pi_3 = \{V_1, V_2, V_3\}$

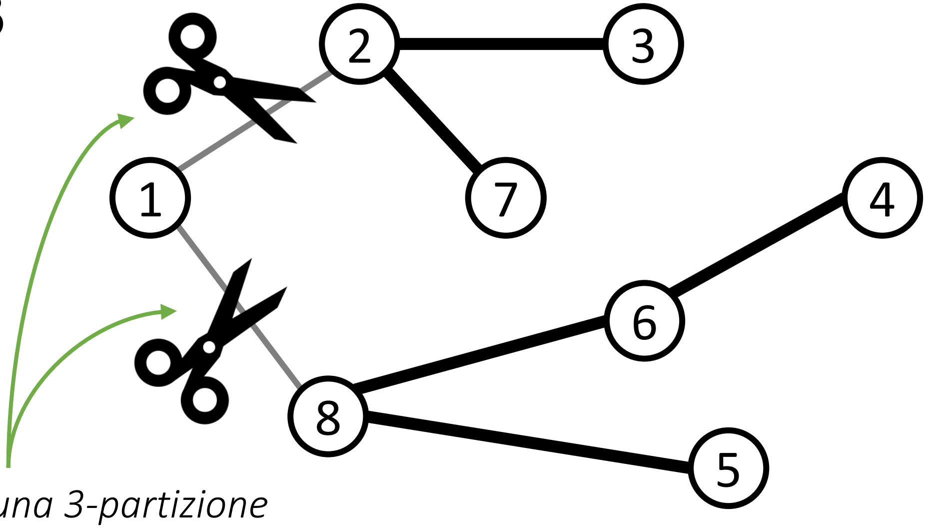
$V_1 = \{2, 3, 7\}$
$V_2 = \{4, 5, 6, 8\}$
$V_3 = \{1\}$

Partizionamento di grafi in componenti connesse

- Se il grafo è un albero T (o un cammino, un caso particolare di albero), ottenere una p-partizione del grafo in componenti connesse equivale a rimuovere esattamente $p-1$ spigoli da $E(T)$, visto che il cammino tra due vertici qualsiasi di T è unico, basta rimuovere uno spigolo per sconnettere il grafo in due componenti distinte e connesse
- Le componenti di una p-partizione di un albero sono a loro volta degli alberi



$p = 3$



si ottiene una 3-partizione dell'albero tagliando 2 spigoli

$\pi_3 = \{V_1, V_2, V_3\}$

$V_1 = \{2, 3, 7\}$
$V_2 = \{4, 5, 6, 8\}$
$V_3 = \{1\}$

Partizionamento di grafi in componenti connesse

- Se una p-partizione di un albero può essere ottenuta rimuovendo solo $p-1$ spigoli, per un grafo il numero di spigoli da rimuovere per ottenere una p-partizione varia a seconda della topologia del grafo
- Il problema può essere formulato come un problema di ottimizzazione combinatoria aggiungendo una funzione obiettivo $f(\pi_p)$ da ottimizzare:

  $$f : \prod_p(G) \longrightarrow \mathbb{R}$$

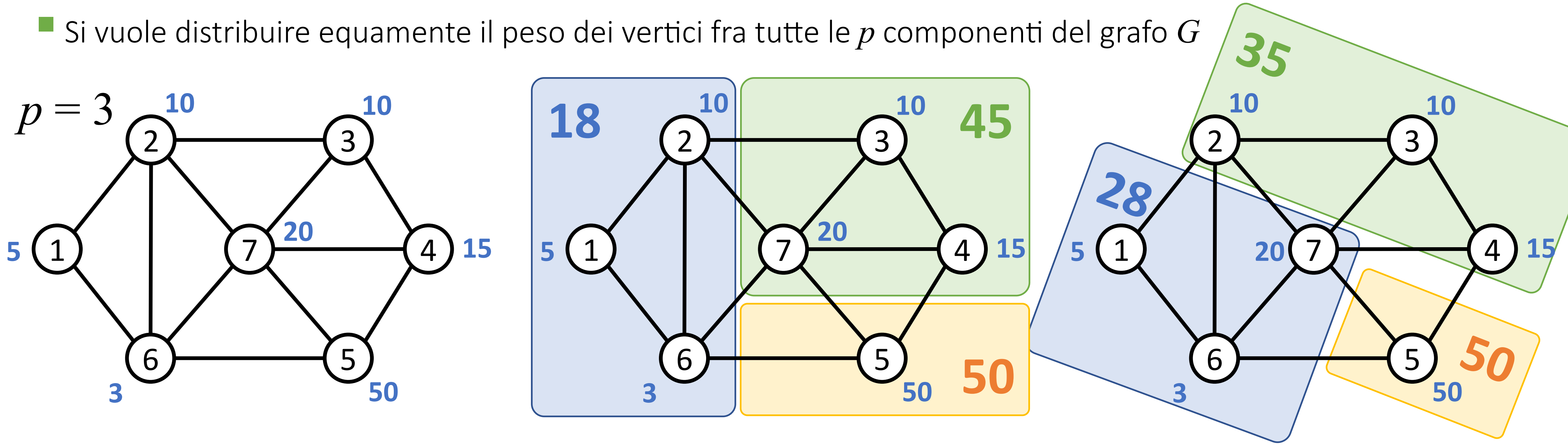
  calcolata in base alla partizione del grafo; dato il grafo $G = (V, E)$ e $p > 1$, si deve trovare la p-partizione $\pi^*_p \in \prod_p(G)$ tale che $f(\pi^*_p) = \min_{\pi_p \in \Pi_p(G)} f(\pi_p)$

- I problemi si distinguono in due famiglie:
  - ***equipartizione***: assegnato un **peso** a ciascun vertice di un grafo, si cerca la p-partizione che renda equilibrato il peso delle p componenti (si cerca la partizione con la minore differenza di peso tra le componenti)
  - ***clustering***: assegnata una **distanza** o un «**indice di dissimilarità**» a ciascuna coppia di vertici del grafo, si cerca la p-partizione tale da aumentare la distanza tra i vertici di componenti diverse o ridurre al minimo la distanza tra gli elementi di una stessa componente
- I problemi di *equipartizione* e di *clustering* su grafi generici sono NP-completi

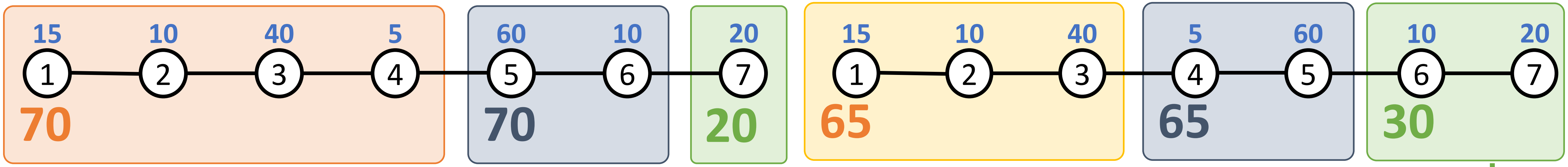
Problemi di equipartizione

- Si vuole distribuire equamente il peso dei vertici fra tutte le p componenti del grafo G

$p = 3$

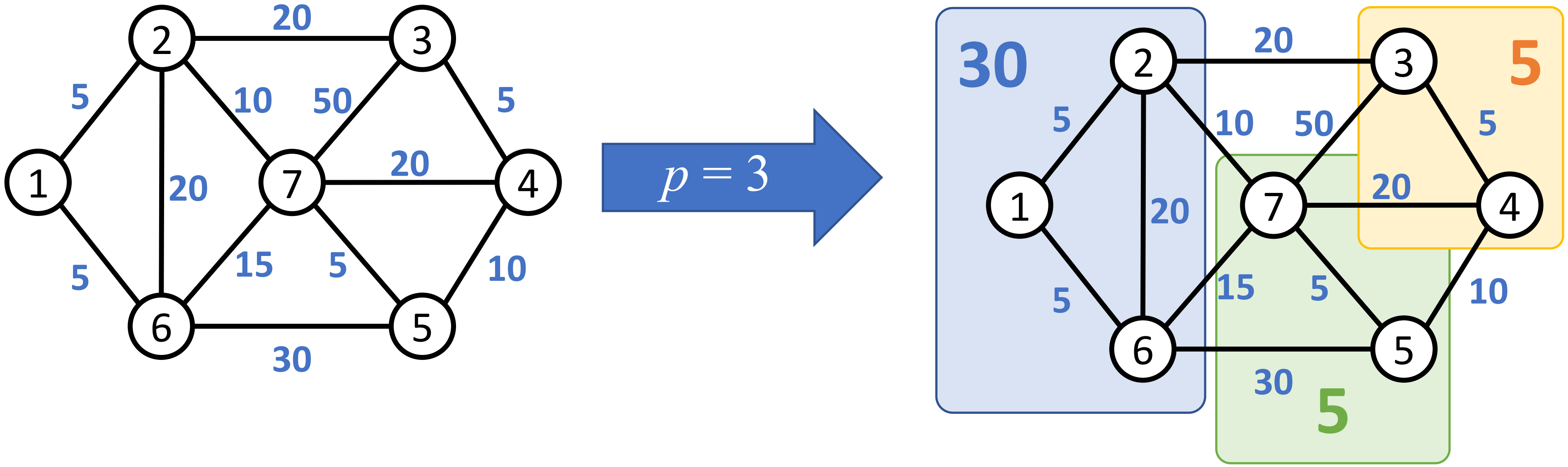


$p = 3$



Problemi di clustering

- Vogliamo aggregare nella stessa componente elementi simili, meno distanti, minimizzando la dissimilarità all’interno delle componenti o massimizzando la distanza tra componenti



- La scelta della partizione ottima dipende dalla funzione obiettivo con cui scegliamo di misurare la distanza tra le componenti o la similarità interna ad una componente

Problemi di equipartizione

- Assegnato un **peso** $w_i = w(v_i) > 0$ a ciascun vertice $v_i \in V(G)$ di un grafo, si cerca la p-partizione π_p che renda equilibrato il peso delle p componenti di π_p
- Si può definire il **peso di una componente** di $\pi_p = \{V_1,\ldots,V_p\}$: $W_k = W(V_k) = \sum_{v \in V_k} w(v)$

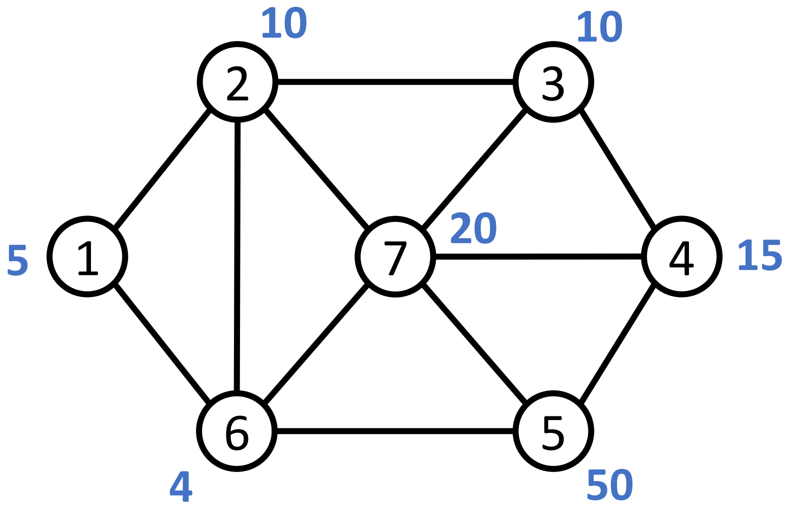
  e il **peso medio di una componente** della partizione π_p: $\mu = \dfrac{\sum_{i=1}^{n} w(v_i)}{p}$
- È chiaro che l'**obiettivo ottimo di un'equipartizione** è quello di ottenere una p-partizione π_p tale che $W(V_k) = \mu$ per ogni $k = 1, \ldots, p$
- Non sempre però, tenendo conto della natura *discreta* del problema, è possibile costruire una p-partizione con $W_k = \mu$ per ogni $k = 1, \ldots, p$ (ad esempio, μ potrebbe essere non intero)
- Vengono così definiti problemi diversi con *differenti funzioni obiettivo*, tutte finalizzate a produrre un'equipartizione del grafo

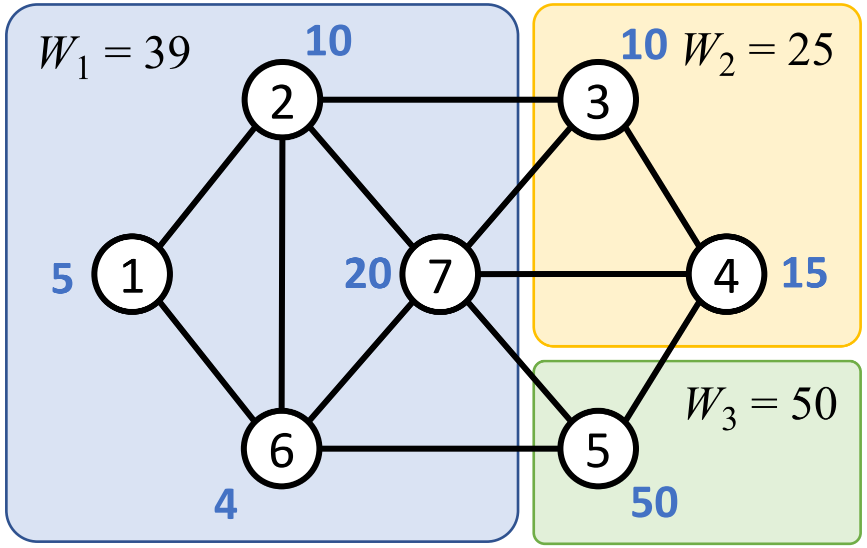
Problemi di equipartizione

- Per ciascuna p-partizione $\pi_p = \{V_1, ..., V_p\}$ di G possiamo definire il vettore $\boldsymbol{W} = (W_1, ..., W_p) \in \mathbb{Z}^p$, con $W_k = W(V_k)$
- In un problema di equipartizione si vuole ottenere una p-partizione in cui il vettore $\boldsymbol{W}$ si avvicini il più possibile al vettore $\boldsymbol{\mu} = (\mu, ..., \mu)$
- Per stimare la distanza fra $\boldsymbol{W}$ e $\boldsymbol{\mu}$ possiamo utilizzare il concetto di **norma di un vettore**: $\|\boldsymbol{W} - \boldsymbol{\mu}\|$
- Vengono così definite diverse **funzioni obiettivo** per il problema della equipartizione in p componenti connesse di un grafo G:
  - Norma L_1: $f(\pi_p) = \|\mathbf{W} - \mu\|_1 = \sum_{k=1}^{p} |W(V_k) - \mu|$
  - Norma L_2: $f(\pi_p) = \|\mathbf{W} - \mu\|_2 = \sum_{k=1}^{p} (W(V_k) - \mu)^2$
  - Norma L_∞: $f(\pi_p) = \|\mathbf{W} - \mu\|_\infty = \max_{k=1,\dots,p} |W(V_k) - \mu|$
- Il **problema di ottimizzazione** chiede di trovare una $\pi_p^* \in \Pi_p(G)$ tale che $f(\pi_p^*) = \min_{\pi_p \in \Pi_p(G)} f(\pi_p)$

Problemi di equipartizione

$$p = 3 \quad \Longrightarrow \quad \mu = 114/3 = 38$$

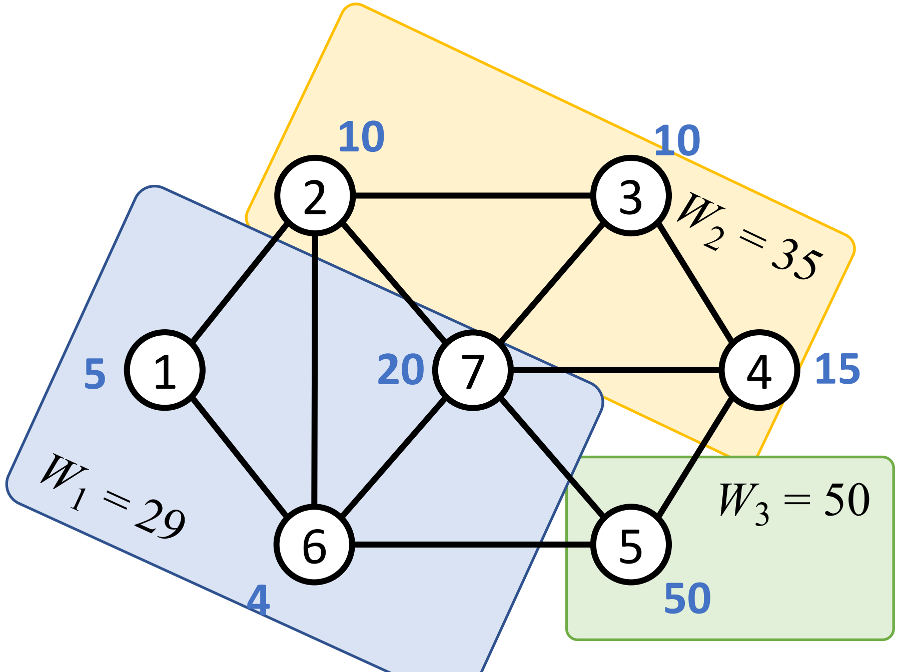




$L_1 : f(\pi_3) = \|\mathbf{W} - \mu\|_1 = 26$

$L_2 : f(\pi_3) = \|\mathbf{W} - \mu\|_2 = 314$

$L_\infty : f(\pi_3) = \|\mathbf{W} - \mu\|_\infty = 13$



$L_1 : f(\pi_3) = \|\mathbf{W} - \mu\|_1 = 24$

$L_2 : f(\pi_3) = \|\mathbf{W} - \mu\|_2 = 234$

$L_\infty : f(\pi_3) = \|\mathbf{W} - \mu\|_\infty = 12$

Problemi di equipartizione

- Altre funzioni obiettivo per problemi di equipartizione:

  - Max-Min: $f(\pi_p) = \min_{k=1,\dots,p} W(V_k) \Rightarrow \pi_p^* : f(\pi_p^*) = \max_{\pi_p \in \Pi_p(G)} f(\pi_p)$ si vuole massimizzare il minimo
  - Min-Max: $f(\pi_p) = \max_{k=1,\dots,p} W(V_k) \Rightarrow \pi_p^* : f(\pi_p^*) = \min_{\pi_p \in \Pi_p(G)} f(\pi_p)$ si vuole minimizzare il massimo
  - MUP (*most uniform partition*): $f(\pi_p) = \max_{k=1,\dots,p} W(V_k) - \min_{h=1,\dots,p} W(V_h) \Rightarrow \pi_p^* : f(\pi_p^*) = \min_{\pi_p \in \Pi_p(G)} f(\pi_p)$

    si vuole minimizzare la massima differenza

Problemi di clustering

- Assegnata una **distanza** o un **indice di dissimilarità** $d_{i,j} \geq 0$ a ciascuna coppia di vertici $v_i, v_j \in V(G)$ del grafo, si cerca la p-partizione π_p di G tale da aumentare la distanza tra i vertici di componenti diverse o ridurre al minimo la distanza tra gli elementi della stessa componente
- La distanza o indice di dissimilarità tra le coppie di vertici del grafo può essere assegnata tramite una matrice quadrata di ordine n, $D = (d_{i,j})$, tale che:

$$d_{i,j} \geq 0 \quad \text{per ogni } i, j = 1, \ldots, n$$

$$d_{i,j} = d_{j,i} \quad \text{per ogni } i, j = 1, \ldots, n$$

$$d_{i,i} = 0 \quad \text{per ogni } i = 1, \ldots, n$$

- Anche in questo caso possiamo definire numerosi problemi di ottimizzazione combinatoria al variare della funzione obiettivo da minimizzare o da massimizzare; in particolare le funzioni obiettivo sono di due tipi:
  - funzioni per la creazione di partizioni in cui sia **massima l'omogeneità all'interno delle componenti** (*inner dissimilarity*): le componenti conterranno elementi poco "distanti/dissimili" fra di loro
  - funzioni per la creazione di partizioni in cui sia **massima la separazione fra le componenti**: le componenti sono costruite in modo da aumentare la distanza minima tra gli elementi di componenti diverse

Problemi di clustering

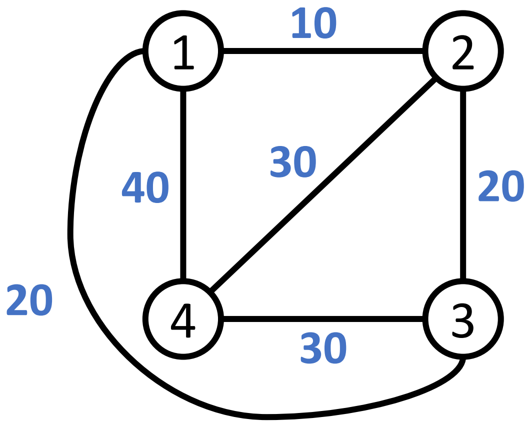
Alcune funzioni obiettivo per problemi di clustering:

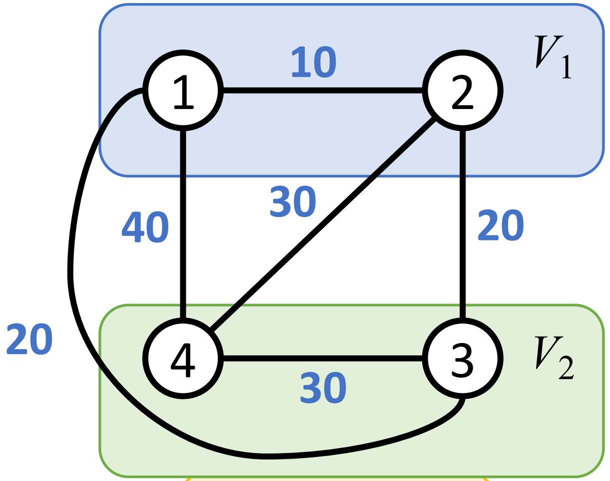
- **Massimo diametro**
  - Consente di massimizzare l'omogeneità all'interno delle componenti della partizione, chiedendo di produrre componenti in cui sia minimo il massimo diametro delle componenti stesse
  - Data una p-partizione $\pi_p \in \Pi_p(G)$ del grafo G, $\pi_p = \{V_1,\dots,V_k\}$, definiamo il **diametro di una componente** ponendo $d(V_k) = \max\{d_{i,j} : v_i, v_j \in V_k\}$
  $$f(\pi_p) = \max_{k=1,\dots,p} \left\{ \max_{v_i,v_j \in V_k} d_{i,j} \right\}$$
  - In questo caso cerchiamo π^*_p tale che $f(\pi^*_p) = \min_{\pi_p \in \Pi_p(G)} f(\pi_p)$

- **Somma delle distanze**
  - Consente di massimizzare l'omogeneità all'interno delle componenti, producendo la partizione per cui sia minima la somma delle distanze tra gli elementi della stessa componente
  $$f(\pi_p) = \sum_{k=1}^{p} \left(\sum_{v_i,v_j \in V_k} d_{i,j} \right)$$
  - Anche in questo caso si cerca π^*_p tale che $f(\pi^*_p) = \min_{\pi_p \in \Pi_p(G)} f(\pi_p)$

Problemi di clustering

- Ad esempio, consideriamo il seguente grafo G e cerchiamo una partizione in $p = 2$ componenti connesse in modo da ottimizzare le due funzioni obiettivo «*Massimo diametro*» e «*Somma delle distanze*»



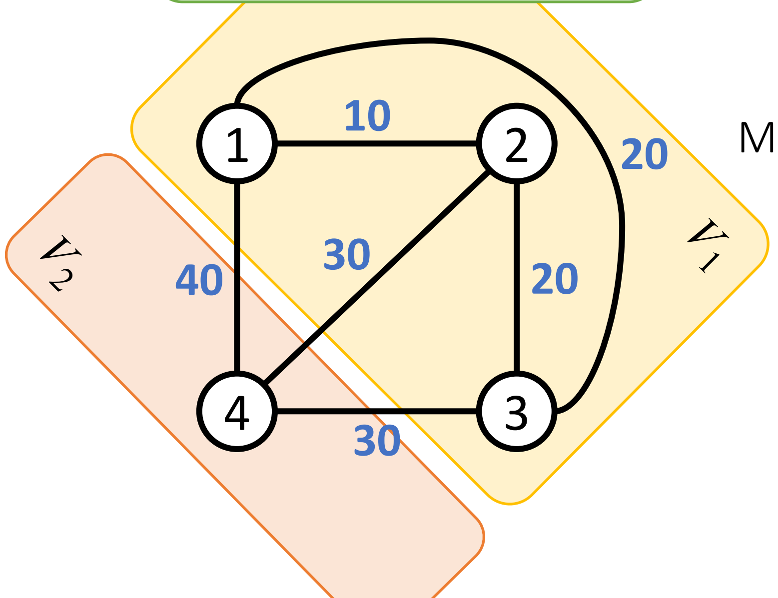


Somma delle distanze:

$$\pi^*_p = \{\{1, 2\}, \{3, 4\}\}$$

$$f(\pi_p) = \sum_{k=1}^{p} \left(\sum_{v_i, v_j \in V_k} d_{i,j} \right)$$

$$f(\pi^*_p) = 40$$



Massimo diametro:

$$\pi^*_p = \{\{1, 2, 3\}, \{4\}\}$$

$$f(\pi_p) = \max_{k=1,\ldots,p} \left\{ \max_{v_i, v_j \in V_k} d_{i,j} \right\}$$

$$f(\pi^*_p) = 20$$

Problemi di clustering

- Minimo split
  - Lo ***split*** di una componente V_k della p-partizione è la minima distanza o dissimilarità di un elemento di V_k con elementi esterni a V_k:

$$\text{split}(V_k) = \min_{v_i \in Vk, v_j \notin V_k} d_{i,j}$$

  - La funzione obiettivo minimo split chiede di massimizzare la distanza tra le componenti, massimizzando il minimo valore dello *split* fra tutte le componenti della partizione
  - La partizione ottima è $\pi^*{}_p$ tale che

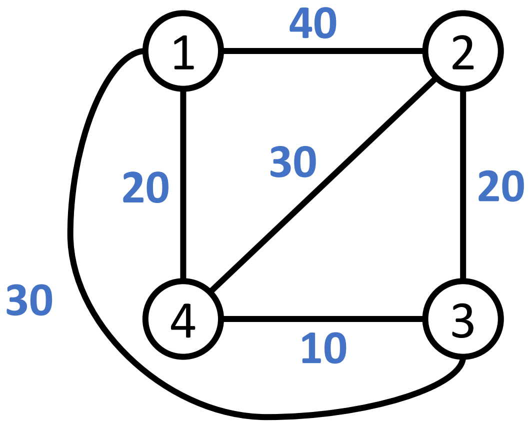
$$f(\pi_p^*) = \max_{\pi_p \in \Pi_p(G)} f(\pi_p) = \max_{\pi_p \in \Pi_p(G)} \left\{ \min_{V_k \in \pi_p} \left\{ \min_{v_i \in V_k, v_j \notin V_k} d_{i,j} \right\} \right\}$$

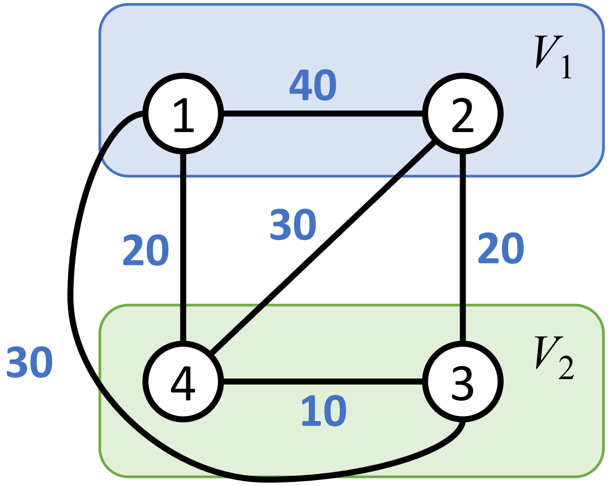
- Somma delle distanze tra cluster
  - Chiede di massimizzare la distanza tra le componenti massimizzando la somma delle distanze tra gli elementi di una componente e tutti gli elementi adiacenti, esterni alla componente stessa

$$f(\pi_p^*) = \max_{\pi_p \in \Pi_p(G)} f(\pi_p) = \max_{\pi_p \in \Pi_p(G)} \left\{ \sum_{k=1}^{p} \left(\sum_{v_i \in V_k, v_j \notin V_k} d_{i,j} \right) \right\}$$

Problemi di clustering

- Ad esempio, consideriamo il seguente grafo G e cerchiamo una partizione in $p = 2$ componenti connesse in modo da ottimizzare le due funzioni obiettivo «*Minimo split*» e «*Somma delle distanze tra cluster*»

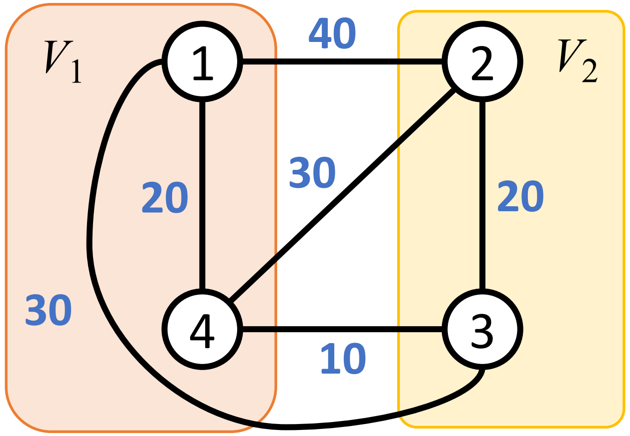




Minimo split:

$$\pi^*_p = \{\{1, 2\}, \{3, 4\}\}$$

$$f(\pi^*_p) = 30$$



Somma delle distanze tra cluster:

$$\pi^*_p = \{\{1, 4\}, \{2, 3\}\}$$

$$f(\pi^*_p) = 220$$

Altri problemi di partizionamento

Oltre all'equipartizione e al clustering sono stati definiti molti altri problemi di partizionamento di grafi:

- **Taglio della partizione**: dato $G = (V, E)$ si assegna ad ogni spigolo un peso non negativo; il valore del taglio della partizione π_p di G è la somma dei pesi degli spigoli «tagliati», i cui estremi si trovano in due componenti diverse
- Il taglio può essere massimizzato o minimizzato, a seconda della specifica funzione obiettivo; nei seguenti casi i pasi sono non negativi e $p = 2$:
  - **Taglio non pesato**: il peso assegnato agli spigoli è unitario; si vuole minimizzare il numero di spigoli «tagliati»
  - **Taglio minimo**: si vuole minimizzare il valore del taglio della bipartizione (si può risolvere con algoritmi di massimo flusso)
  - **Taglio massimo**: si vuole massimizzare il valore del taglio della bipartizione (è NP-hard su grafi qualsiasi, polinomiale su specifiche classi di grafi)

- Partizionamento continuo
  - agli spigoli del grafo è assegnata una «lunghezza» e la partizione avviene ottimizzando una funzione calcolata in base alla lunghezza degli spigoli di ciascuna componente: uno spigolo può essere tagliato in un punto qualsiasi, contribuendo per parte della sua lunghezza ad una delle due componenti e per la restante all'altra

Complessità degli algoritmi per il partizionamento di grafi

I problemi di equipartizione e clustering sono generalmente NP-completi (o NP-hard) su grafi generici, ma possono essere risolti quasi sempre con algoritmi polinomiali su specifiche classi di grafi (tipicamente cammini e alberi)

Equipartizione		
Obiettivo	**Cammini**	**Alberi**
Norma L_1	$O(n\,p)$	NP-completo, $O(n^2\,p)$ per *caterpillar*
Norma L_2	$O(n^2\,p)$	–
Norma L_∞	$O(n\,p\log_2 p)$	–
Max-Min	$O(n\,p\log_2 p)$	$O(n\,p\log_2 p)$
Min-Max	$O(n\,p\log_2 p)$	$O(n^2\,p^3)$
MUP	$O(n^3)$, $O(n^2\,p\log_2 n)$	–

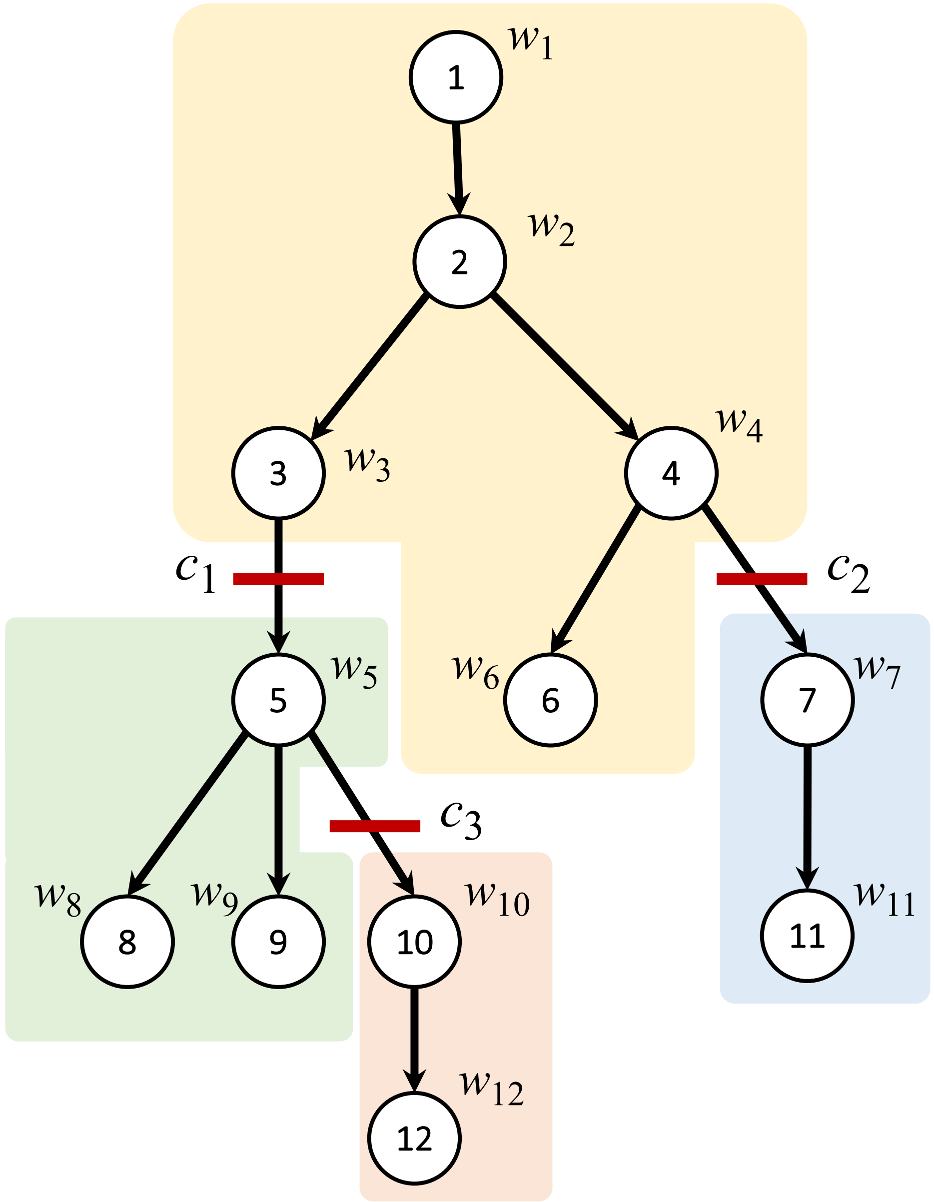
Clustering		
Obiettivo	**Cammini**	**Alberi**
Inner dissimilarity	$O(n^2)$	NP-completo
Maximum diameter	$O(n^2\,p)$	NP-completo
Minimo split	$O(n^2)$	$O(n^2)$

Metodi generali per la soluzione di problemi di partizionamento

- **Ottimizzazione su reti**: si usano varianti degli algoritmi per problemi di massimo flusso (per problemi di clustering, con pesi/distanze/dissimilarità sugli spigoli)
- **Migrazione di gruppi**: si genera una partizione iniziale e poi si passa alla seguente «migrando» un gruppo di vertici da una componente ad un'altra, fino a quando mediante operazioni di «migrazione» non si riesce più a migliorare la funzione obiettivo
- **Simulated annealing**: simile alla tecnica precedente, salvo che per le condizioni di stop: l'algoritmo accetta una «migrazione» di vertici anche se così facendo dovesse peggiorare il valore della funzione obiettivo; le condizioni che fanno terminare l'algoritmo sono basate su criteri locali
- **Semi**: vengono selezionati p vertici del grafo (*semi*) e vengono assegnati alle p componenti della partizione (es.: i vertici più distanti fra loro in un problema di clustering); poi vengono aggiunti alle componenti anche gli altri vertici sulla base di criteri di similarità rispetto ai vertici già presenti in ciascuna componente
- **Programmazione dinamica**: si risolve il problema combinando la soluzione di sottoproblemi
- **Shifting**: si sposta l'attenzione sui «tagli» assegnati agli spigoli del grafo (tipicamente un cammino o un albero per questo tipo di algoritmi); i tagli sono assegnati a priori a qualche spigolo, poi vengono spostati su uno spigolo incidente con l'obiettivo di migliorare la funzione obiettivo
- **Programmazione lineare**: si formalizza il problema con un sistema di equazioni e disequazioni lineari sulle variabili i (vertici del grafo), j (indice delle componenti della partizione), k (indice che identifica gli spigoli del grafo)

Equipartizione di alberi con funzione obiettivo Max-Min

- Vogliamo produrre una **equipartizione** di un albero $T = (V, E)$ con $|V| = n$ vertici e $m = n - 1$ spigoli, in p sottoalberi (componenti connesse) indotti da una partizione dell'insieme dei vertici $\pi_p = \{V_1, \ldots, V_k\}$
- Sui vertici dell'albero sono assegnati dei pesi non negativi w_i
- Un ***taglio*** c è uno spigolo di $(u, v) \in E(T)$ che viene rimosso per costruire la partizione di T; occorrono esattamente $p - 1$ tagli per produrre una p-partizione di T in componenti connesse
- Imponiamo un orientamento naturale all'albero T per ottenere un albero con radice (scelta arbitrariamente fra i vertici di grado 1); chiamiamo ***root component*** di π_p la componente che contiene la radice dell'albero
- Una down component D_c di un taglio c è la componente limitata dall'alto da c; indichiamo con $W(D_c)$ la somma dei pesi associati ai vertici della componente D_c



Equipartizione di alberi con funzione obiettivo Max-Min

- Il problema viene affrontato e risolto in tempo polinomiale da un algoritmo che utilizza la tecnica dello *shifting* dei tagli, inizialmente collocati sull'unico spigolo incidente la radice
- Si vuole massimizzare il peso della componente più leggera della partizione
- Ad ogni passo si individua la componente di peso minimo $W_{\min}$
- Quindi si identifica il taglio su cui eseguendo lo *shift* si ottiene la *down component* più pesante
- Se il peso della *down component* dopo l'esecuzione dello *shift* è comunque più pesante di $W_{\min}$, si esegue lo *shift* e si reitera l'algoritmo, altrimenti si termina

Algoritmo 35 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

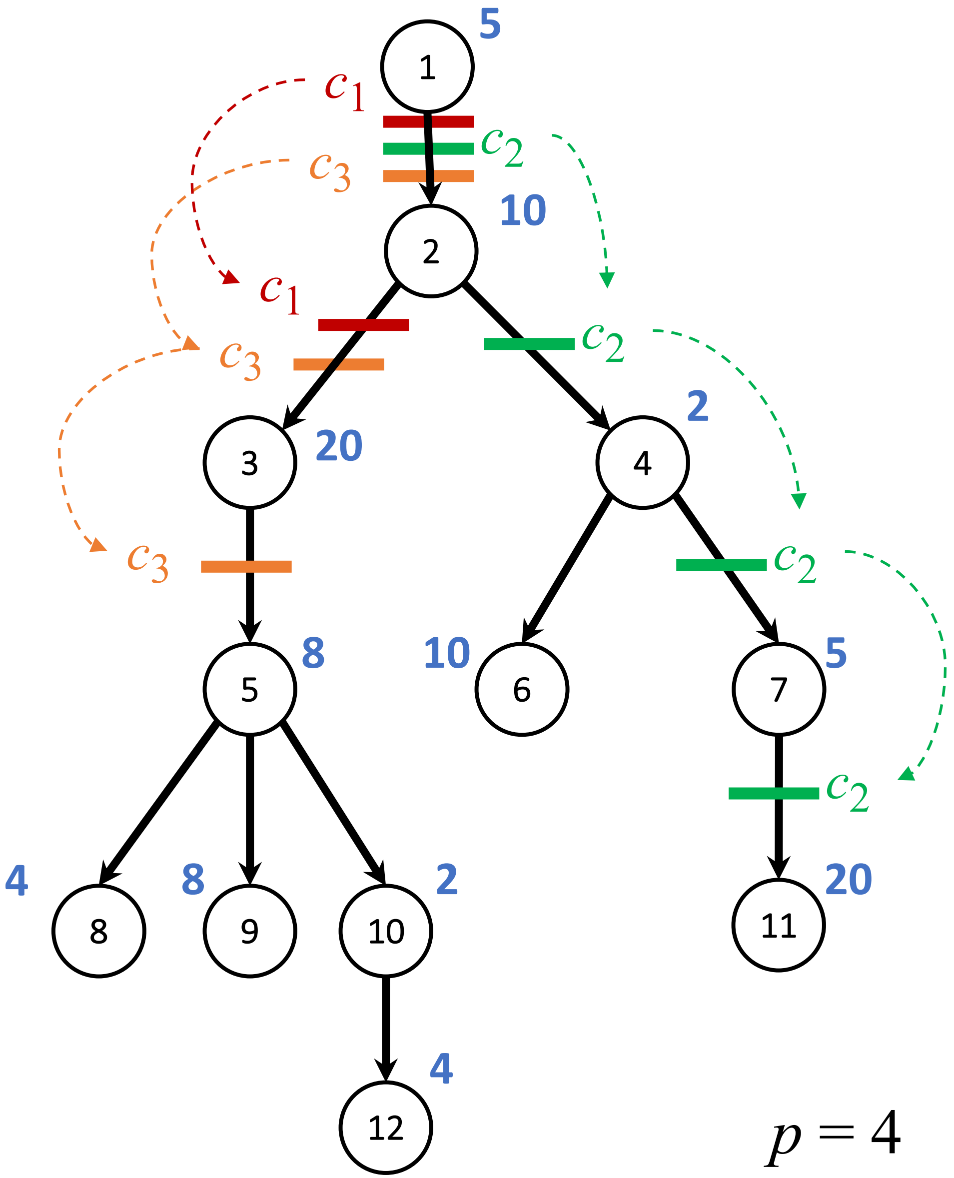
Output: Una p-partizione π_p di T tale da massimizzare la funzione obiettivo Max-Min

1: assegna tutti i tagli $c_1, \ldots, c_{p-1}$ all'unico spigolo incidente la radice r 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 $f(\pi_p) = W_{\min}$

Complessità: $O((p-1)^2 R(T) + (p-1)m)$

dove con R(T) si è indicato il raggio di T: $R(T) = min_{v_i} max_{v_j} d(v_i, v_j)$

Equipartizione di alberi con funzione obiettivo Max-Min



$p = 4$

Algoritmo 35 MaxMin(G, W, p)

Input: Un albero con radice $T = (V, E)$ e un insieme di pesi $W = \{w_1, \dots, w_n\}$ 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 $c_1, \dots, c_{p-1}$ all'unico spigolo incidente la radice r 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 $W(D_c)$

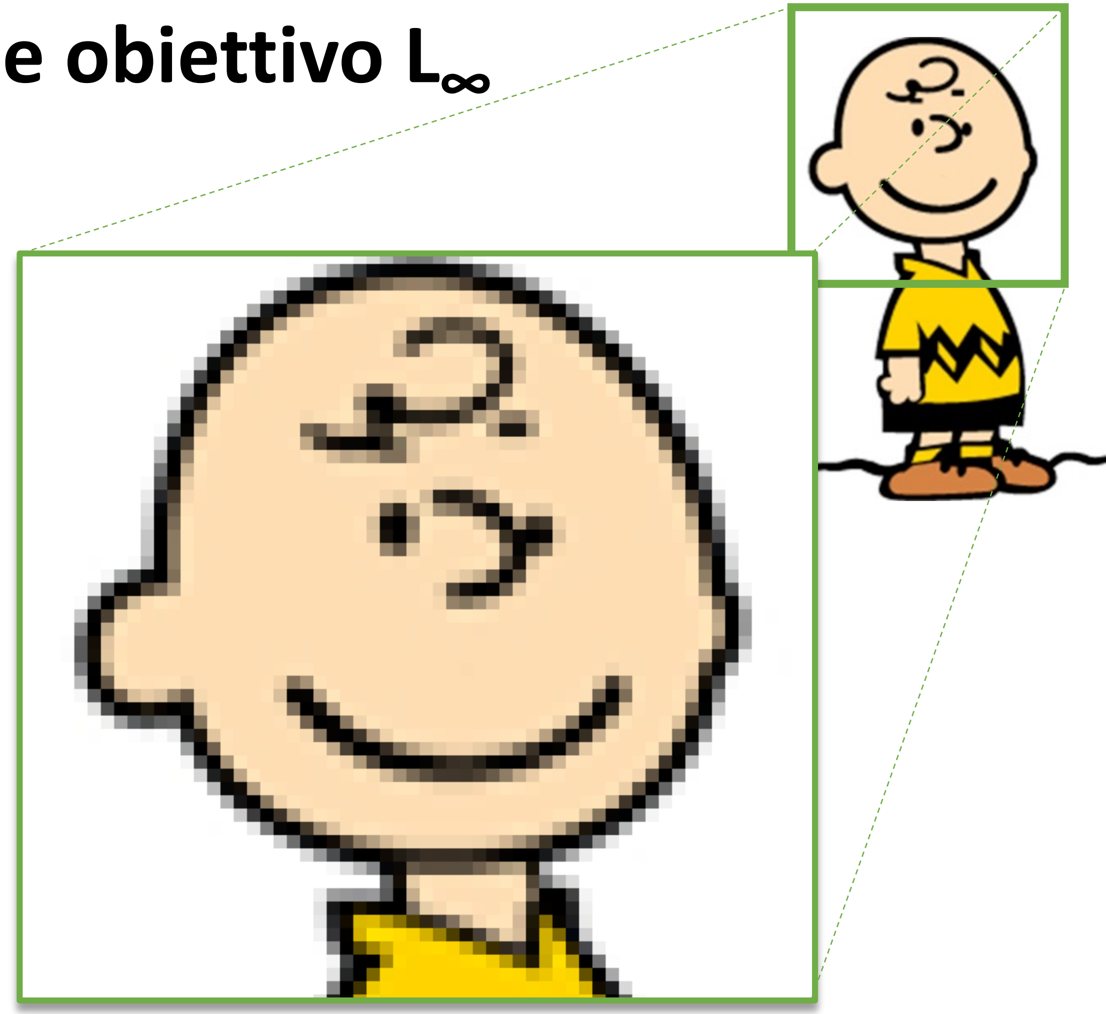
4: se $W(D_c) \geq W_{\min}$ allora esegui lo *shift* di c su e e torna al passo 2

5: la p-partizione π_p ottenuta è ottimale; restituisci $f(\pi_p) = W_{\min}$

$$\underbrace{W_0 = 32 \qquad W_1 = 20 \qquad W_2 = 20 \qquad W_3 = 26}_{W_{\min} = 20}$$

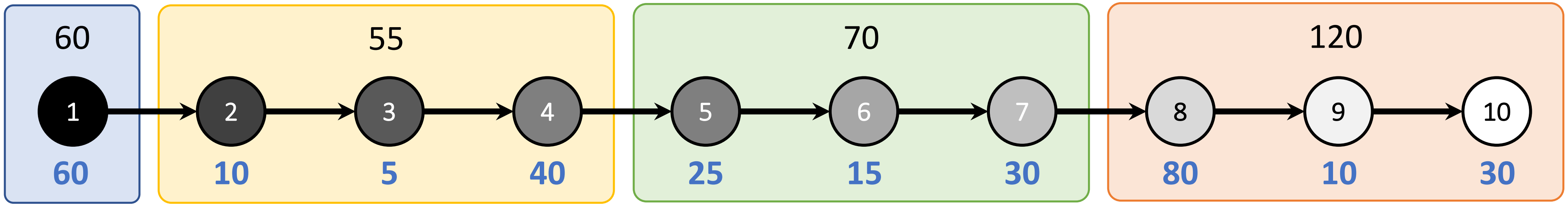
Equipartizione di cammini con funzione obiettivo L_∞

- Un'immagine grafica «*raster*» è un mosaico di punti (*pixel*) più o meno fitti, a seconda della *risoluzione* dell'immagine
- Ciascun pixel è caratterizzato da un codice numerico che rappresenta il colore associato a quel pixel
- Problema:
  - si deve **ridurre il numero di colori** con cui è stata rappresentata un'immagine passando da n colori differenti a soli $p < n$ colori distinti
  - come si scelgono i colori in modo da mantenere l'immagine ben comprensibile?
  - Esempio su un caso reale:
    - $5.500 \times 3.700 = 20.350.000$ pixel
    - $n = 300.000$ colori immagine originale
    - $p = 256$ colori immagine trasformata



Equipartizione di cammini con funzione obiettivo L_∞

- Possiamo costruire un modello del problema trasformandolo nel problema di ottimizzazione combinatoria di partizionamento ottimo di un cammino
- In particolare possiamo considerare un cammino P con n vertici; ogni vertice del cammino rappresenta uno dei colori dell'immagine, disposti secondo una scala che va dal colore più scuro (il nero) a quello più chiaro (il bianco)
  - *per semplicità consideriamo un'immagine in bianco e nero, dove invece dei colori abbiamo dei «toni di grigio»*
- Ad ogni vertice del cammino associamo un peso non negativo dato dal numero di pixel che nell'immagine hanno il colore associato al vertice del cammino
- Vogliamo partizionare il cammino in p componenti / sotto-cammini di peso uniforme, in modo tale da assegnare nell'immagine rielaborata a tutti i pixel con un colore nella componente k lo stesso colore k, scelto dalla palette di soli p colori



$n = 10$
$p = 4$

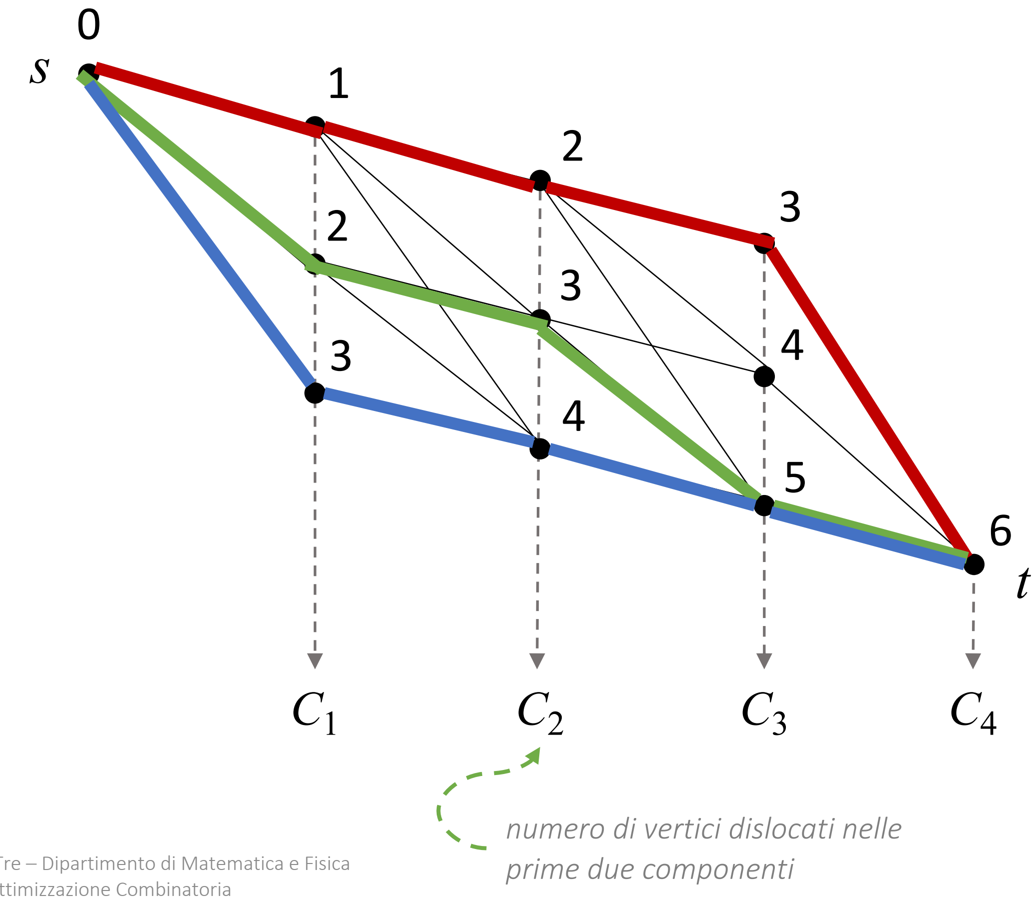
Equipartizione di cammini con funzione obiettivo L_∞

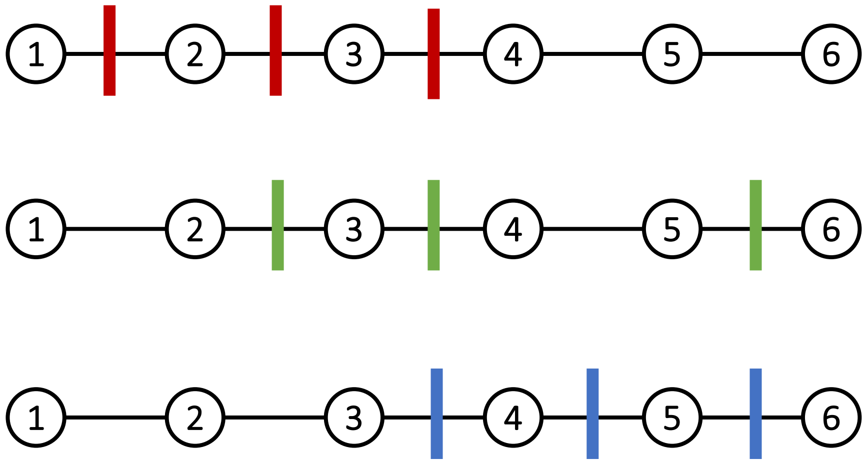
- Un grafo $G = (V, E)$ è un cammino se $V = \{1, ..., n\}$ ed $E = \{(i, i + 1), i = 1, ..., n - 1\}$
- Ad ogni vertice è assegnato un peso $w_i > 0$
- Sia $p > 0$ il numero di componenti connesse in cui intendiamo suddividere il cammino G «tagliando» $p - 1$ spigoli; risulta quindi:
  - peso medio delle p componenti connesse: $\mu = \frac{1}{p} \sum_{i=1}^{n} w_i$
  - peso delle componenti connesse $C_1, ..., C_p \subset V(G)$: $W_k = \sum_{i \in C_k} w_i$
  - funzione obiettivo «**norma di Chebyshev**» o **norma L_∞** da minimizzare: $f(\pi_p) = \max |W_k - \mu|$
- Il numero di p-partizioni di un cammino con n vertici è dato da $\frac{(n-1)!}{(p-1)!(n-p)!}$

  per generare una p-partizione dobbiamo infatti scegliere $p - 1$ spigoli da rimuovere su un insieme di $n - 1$ spigoli complessivi (se $n = 255$ e $p = 16$ allora sono più di 629×10^{21} p-partizioni diverse)

Equipartizione di cammini con funzione obiettivo L_∞

- Per studiare l'algoritmo risolutivo è utile rappresentare graficamente le possibili p-partizioni del cammino P_n su una rete N composta mediante un grafo multipartito con $p+1$ insiemi indipendenti
- Esempio: $n = 6, p = 4$





Ad ogni cammino da s a t su N corrisponde una p-partizione di P_n

Equipartizione di cammini con funzione obiettivo L_∞

- Per trovare la p-partizione π_p di P_n ottima per la funzione obiettivo Norma L_∞ procediamo assegnando i $p-1$ tagli ai primi $p-1$ spigoli del cammino, spostandoli poi verso destra uno alla volta fino a quando non sarà più possibile spostarli a destra, *senza creare delle componenti vuote*
- In questo modo si passa da una partizione che corrisponde su N ad un certo cammino da s a t, ad un cammino «più basso» su N, partendo inizialmente dal cammino più in alto di tutti
- Indichiamo con C_k la **componente di scostamento massimo dalla media** μ; l'algoritmo termina quando una di queste tre condizioni è verificata:
  1. C_k è composta da un solo elemento e $W(C_k) - \mu > 0$
  2. $W(C_k) - \mu > 0$ e $k = 1$ (la prima comp. di π_p è la più pesante)
  3. $W(C_k) - \mu < 0$ e $k = p$ (l'ultima comp. di π_p è la più leggera)
- Se nessuna di queste condizioni è verificata (e la componente C_k di scarto massimo non ha scarto nullo) l'algoritmo prosegue:
  - allargando la componente C_k se $W(C_k) - \mu < 0$, spostando a destra il taglio che la limita a destra
  - restringendo la componente C_k se $W(C_k) - \mu > 0$, spostando a destra il taglio che la limita da sinistra

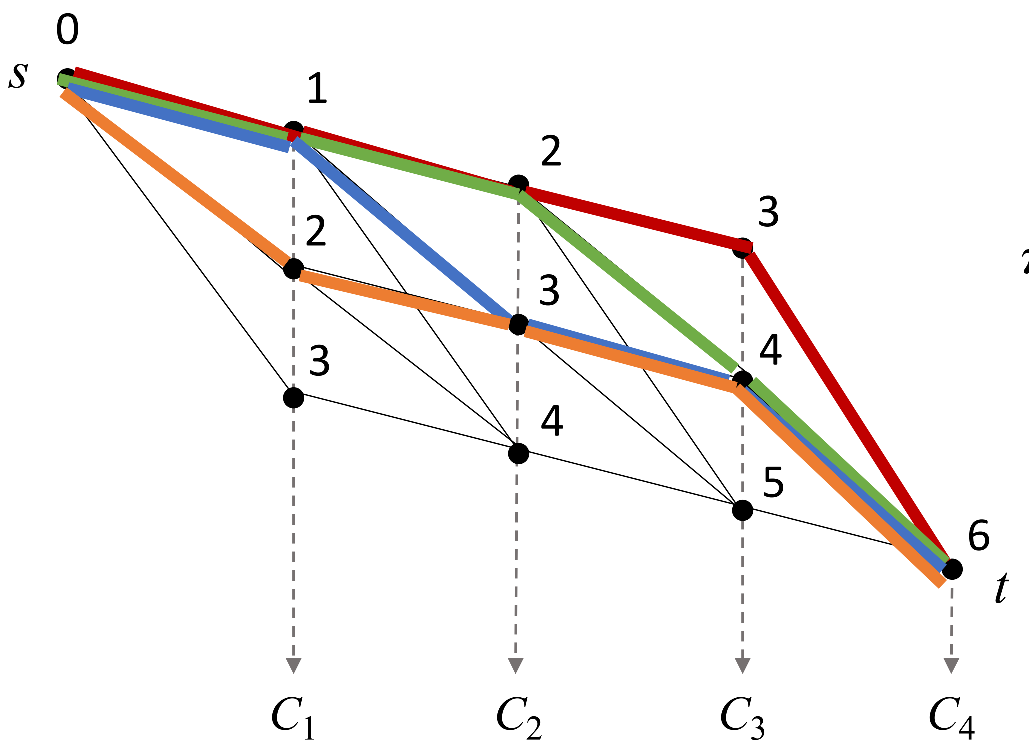
Algoritmo 36 PATHSHIFTING(n, p, W)

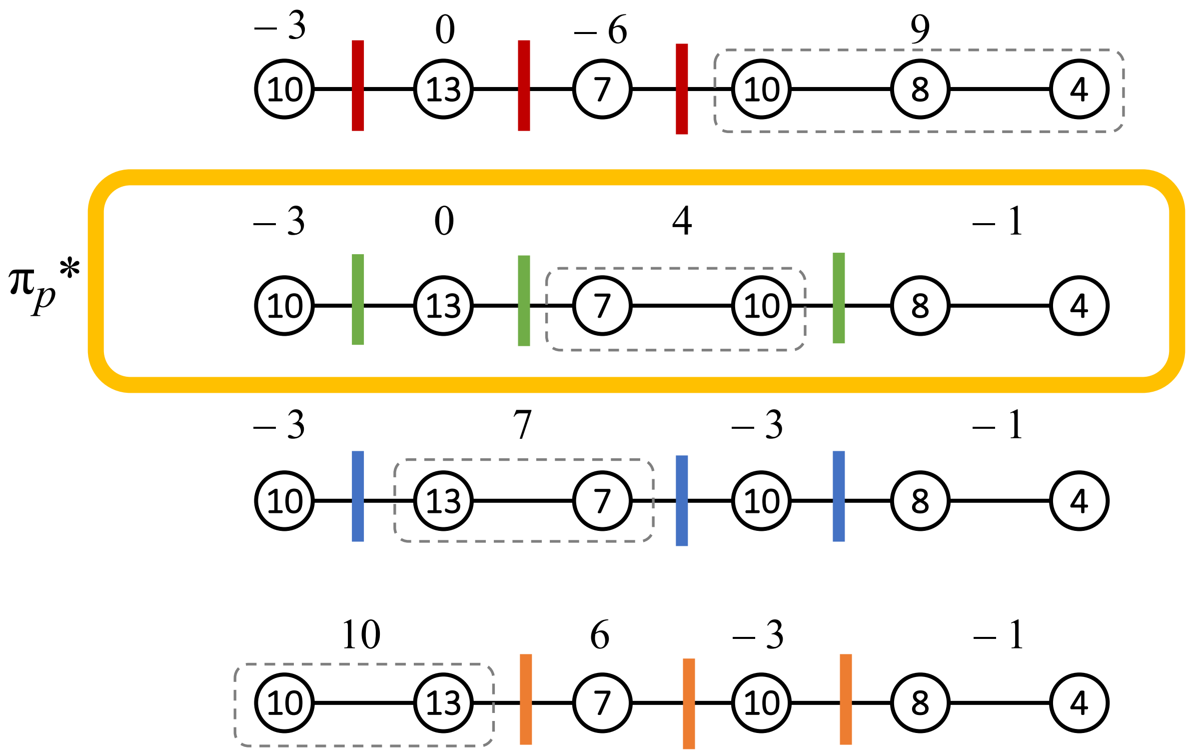
Input: $n = |V|$, il numero di componenti p, i pesi W associati ai vertici
Output: Una p-partizione π_p^* tale da minimizzare la Norma L_∞

assegna $p-1$ tagli allo spigolo aggiuntivo $(0, v_1)$
sia C_k t.c. $|W(C_k) - \mu|$ massimo; $\max = |W(C_k) - \mu|$, $\pi_p^* = \pi_p$
$stop = 0$
fintanto che $stop = 0$ e $W(V_k) - \mu \neq 0$ **ripeti**
  se $|C_k| = 1$ e $W(C_k) - \mu > 0$ **allora**
    $stop = 1$ 1
  altrimenti
    se $W(C_k) - \mu > 0$ **allora**
      se $k = 1$ **allora**
        $stop = 1$ 2
      altrimenti
        sposta il primo elemento di C_k in C_{k-1}
      fine-condizione
    altrimenti se $W(C_k) - \mu < 0$ **allora**
      se $k = p$ **allora**
        $stop = 1$ 3
      altrimenti
        sposta il primo elemento di C_{k+1} in C_k
      fine-condizione
    fine-condizione
    sia π_p la nuova partizione; sia C_k t.c. $|W(C_k) - \mu|$ massimo
    se $|W(C_k) - \mu| < \max$ **allora**
      $\max = |W(C_k) - \mu|$, $\pi_p^* = \pi_p$
    fine-condizione
  fine-condizione
fine-ciclo
la p-partizione π_p^* ottenuta è ottimale; restituisci $f(\pi_p^*) = W_{\min}$

Equipartizione di cammini con funzione obiettivo L_∞

- Esempio:
  - consideriamo il cammino $P_6 = (V, E)$, con $V = \{1, 2, 3, 4, 5, 6\}$, $E = \{(1, 2), (2, 3), (3, 4), (4, 5), (5, 6)\}$ e pesi $W = \{w_1 = 10, w_2 = 13, w_3 = 7, w_4 = 10, w_5 = 8, w_6 = 4)$
  - supponiamo di voler ottenere una p-partizione con $p = 4$; risulta quindi $\mu = (10 + 13 + 7 + 10 + 8 + 4)/4 = 13$





Equipartizione di cammini con funzione obiettivo L_∞

- La complessità dell'algoritmo PATHSHIFTING è $O(n\, p \log_2 p)$

  Dimostrazione:
  - ognuno dei $p - 1$ tagli esegue al più $n - 1$ shift verso destra
  - prima di eseguire uno *shift* occorre individuare la componente di scarto massimo dalla media
  - utilizzando un *heap binario* per memorizzare gli scarti del peso delle p componenti dalla media μ, possiamo ridurre il tempo di ricerca della componente di scarto massimo a $\log_2 p$ ■

- Identifichiamo una p-partizione di P_n con una sequenza $(s_0, s_1, s_2, \ldots, s_{p-1}, s_p)$, con $s_0 = 0$, $s_p = n$ e $s_k = i$ se e solo se il taglio k è sullo spigolo $(i, i + 1)$
- Date due partizioni π'_p e π''_p di P_n definite rispettivamente dai tagli $(c'_1, \ldots, c'_{p-1})$ e $(c''_1, \ldots, c''_{p-1})$ si dice che π'_p **è *sopra*** a π''_p se $s'_k \leq s''_k$ per ogni $k = 0, \ldots, p$
- La proprietà «*stare al di sopra*» definisce un **ordine parziale** tra le p-partizioni π_p di P_n , per cui $\prod_p(P_n)$ è un **reticolo**

Equipartizione di cammini con funzione obiettivo L_∞

- **Lemma 1.** Sia π^t_p la **partizione finale** trovata dall'algoritmo. Tutte le partizioni π_p sotto π^t_p sono tali che $f(\pi_p) \geq f(\pi^t_p)$

  Dimostrazione.

  - Sia C_k^t la componente di π^t_p che determina il valore di $f(\pi^t_p)$ (quella con il massimo scostamento dalla media). Sia π_p una generica partizione sotto π^t_p
  - Se $C_k = C_k^t$ allora ovviamente $f(\pi_p) \geq f(\pi^t_p)$
  - Sia $C_k \neq C_k^t$
    - Se $C_k^t = \{v_i\}$, allora C_k^t è la più pesante e siccome vi deve appartenere a qualche componente di π_p allora sicuramente risulta $f(\pi_p) \geq f(\pi^t_p)$
    - Sia $k = 1$, dunque C_k^t è la prima componente di π^t_p ed è quindi la più pesante: risulta $W(C_1^t) - \mu \geq 0$. Allora se π_p è sotto π^t_p, $C_1^t \subset C_1$, quindi $f(\pi_p) \geq f(\pi^t_p)$
    - Sia $k = p$, dunque C_k^t è l'ultima componente di π^t_p ed è quindi la più leggera: $W(C_1^t) - \mu \leq 0$. Allora se π_p è sotto π^t_p, $C_p \subset C_p^t$, quindi anche in questo caso risulta $f(\pi_p) \geq f(\pi^t_p)$ ■

Equipartizione di cammini con funzione obiettivo L_∞

- **Lemma 2.** Siano $\pi^1_p, \pi^2_p, \ldots, \pi^t_p$ le partizioni generate dall'algoritmo, sia π_p^* una partizione ottimale. Se π_p^* è sotto a π_p^q, ma non è sotto π_p^{q+1}, allora π_p^q è ottima.

  Dimostrazione.

  - Sia $\pi_p^* = \{C_1^*, \ldots, C_p^*\}$ la partizione ottimale che è al di sotto di π_p^q, ma non al di sotto di π_p^{q+1}
  - Tra π_p^q e π_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 un vertice
  - Sia C_k la componente di π_p^q di scarto massimo dalla media
    - Se C_k è la più pesante, allora il taglio s_k che la limita a sinistra sarà $s_k + 1$ in π_p^{q+1}
    - Se C_k è la più leggera, allora il taglio s_{k+1} che la limita a destra sarà $s_{k+1} + 1$ in π_p^{q+1}
  - Quindi se π_p^* non è al di sotto di π_p^{q+1}, ma è al di sotto di π_p^q, deve essere necessariamente $\pi_p^q = \pi_p^*$ ■

Equipartizione di cammini con funzione obiettivo L_∞

- **Teorema**. Almeno una delle partizioni $\pi^1_p, \pi^2_p, \ldots, \pi^t_p$ generate dall'algoritmo è ottima.

  Dimostrazione.
  - π^1_p è al di sopra di ogni possibile p-partizione del cammino
  - Se una partizione è al di sopra di $\pi_p{}^*$ allora tutte le partizioni generate prima di questa sono al di sopra di $\pi_p{}^*$ per transitività
  - Sia $m = \max\{r : \pi^r_p \text{ è sopra } \pi_p{}^*\}$. Dimostriamo che allora π^m_p è ottima. Per definizione di m c'è almeno una partizione ottima sotto π^m_p
  - Sia $m = t$. Allora $f(\pi^m_p) \leq f(\pi_p{}^*)$ per il Lemma 1 e quindi π^m_p è ottima
  - Sia $m < t$. Allora per la massimalità di m, $\pi_p{}^*$ non è sotto $\pi_p{}^{m+1}$ e quindi per il Lemma 2, π^m_p è ottima ■

Riferimenti bibliografici

- Marco Liverani, «*Dispense del Corso di Ottimizzazione Combinatoria:* Partizionamento ottimo di grafi in componenti connesse» (http://www.mat.uniroma3.it/users/liverani/doc/disp_oc_10.pdf)
- 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
- Claudio Arbib, *A polynomial characterization of some graph partitioning problems*, Information Processing Letters, 26, 1987/88, pp. 223-230
- Ronald I. Becker, Yehoshua Perl, *The shifting algorithm technique for the partitioning of trees*, Discrete Applied Mathematics 62, 15-34, 1995
- Marco Liverani, Aurora Morgana, Bruno Simeone, Gianni Storchi, *Path equipartition in the Chebyshev norm*, European Journal of Operational Research, 123, 2000, pp. 428-436
- Mario Lucertini, Yehoshua Perl, Bruno Simeone, *Most uniform path partitioning and its use in image processing*, Discrete Applied Mathematics, 42, 1993, pp. 227-256
- 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