Università degli Studi Roma Tre
Facoltà di Scienze M.F.N.
Corso di Laurea in Matematica

Tesi di Laurea Magistrale in Matematica
di
Natascia Piroso

Strategie risolutive e algoritmi per problemi di partizionamento ottimo di grafi

Sintesi

Candidato
Natascia Piroso

Relatore
Prof. Marco Liverani

Anno Accademico 2006-2007
Luglio 2007

Classificazione AMS: 05A18, 05C85, 68R05, 68R10, 90C39, 90C90.
Parole chiave: graph, optimization, partitioning, clustering, algorithms, strategies.

Introduzione

Il presente lavoro affronta lo studio dei problemi di partizionamento ottimo su grafi, una vasta e importante famiglia di problemi di ottimizzazione combinatoria. In linea generale, un problema di partizionamento ottimo richiede di suddividere un insieme di oggetti, correlati da un tessuto più o meno complesso di relazioni, in insiemi a due a due disgiunti in modo tale da soddisfare vincoli di varia natura ed in modo da raggiungere o, quanto meno, tendere il più possibile ad un obiettivo prefissato. Problemi di questo tipo si presentano in numerosissimi ambiti applicativi scientifici e ingegneristici in cui la situazione pratica oggetto d'analisi può essere schematizzata sotto forma di grafo. La riproduzione di immagini digitali, la progettazione di circuiti VLSI, l'uso razionale della memoria distribuita di un calcolatore parallelo, la visualizzazione di grafi di grandi dimensioni, lo scheduling degli equipaggi e la collocazione di servizi fondamentali all'interno di un territorio sono solo alcuni esempi di applicazioni pratiche che fanno uso di tecniche di partizionamento e *clustering*.

Il problema reale modellizzato dal grafo impone vincoli che limitano l'arbitrarietà della partizione. Solitamente, sussistono vincoli di natura topologica sui sottografi indotti dalla partizione. Le grandezze caratterizzanti del problema concorrono alla definizione di una funzione della partizione stessa, scelta in modo tale da ottenere, minimizzandola o massimizzandola, la migliore partizione possibile del grafo, conformemente all'obiettivo preposto. In molte applicazioni reali la dimensione del problema può essere notevole, e si rende quindi necessaria l'adozione di tecniche risolutive efficienti sotto il profilo della complessità computazionale. La risoluzione di tali problemi è tanto più complicata, quanto più complessi sono l'obiettivo che ci si pone e la struttura del problema. Tuttavia, anche obiettivi e strutture apparentemente semplici, possono generare problemi computazionalmente molto complicati: sono i cosiddetti problemi NP-completi.

Uno degli obiettivi di questa tesi è quello di classificare i principali problemi di partizionamento all'interno di quello che si presenta come un panorama piuttosto segmentato. Il lavoro si è basato su numerosi articoli esistenti in letteratura e si è mosso verso la definizione di un percorso organico con cui proporre una chiave di lettura originale del materiale bibliografico. In particolare, si è cercato di porre l'accento sulle tecniche di ottimizzazione e le strategie risolutive che permettono di risolvere in maniera esatta o approssimata alcuni dei problemi di partizionamento che più frequentemente sorgono nell'ambito di applicazioni pratiche piuttosto comuni. Ogni tecnica risolutiva analizzata è stata presentata dapprima in linea generale, discutendone, laddove possibile, le basi teoriche che ne permettono l'applicazione ad opportune classi di problemi. In un secondo istante, vengono invece forniti esempi dettagliati di possibili applicazioni della strategia risolutiva a problemi teorici specifici di effettiva utilità pratica. Quest'ultima trattazio-

ne, centrale in ogni capitolo, viene condotta in modo da mettere in luce il processo di sviluppo di un algoritmo, focalizzando sulle idee portanti che possono aver spinto verso una determinata scelta risolutiva. A tal fine, ogni problema viene presentato attraverso un esempio pratico sul quale poter sperimentare le difficoltà o l'efficacia di un determinato approccio; una volta delineata una strategia risolutiva convincente, se ne descrive in dettaglio l'algoritmo implementativo; in ultima analisi, viene discussa la correttezza del procedimento risolutivo attraverso un processo dimostrativo rigoroso e quasi mai banale.

È stato inoltre realizzato, come contributo applicativo originale a completamento del lavoro di tesi , il package `pgrafi`, un pacchetto adatto a trattare grafi sviluppato utilizzando il linguaggio `Java`. `pgrafi` raccoglie classi progettate appositamente per la rappresentazione, la manipolazione e la visualizzazione di grafi, e per l'implementazione di algoritmi di base e di algoritmi di ottimizzazione su grafi. Il pacchetto è stato utilizzato come semplice strumento di ausilio nello studio e nell'implementazione di alcuni degli algoritmi di partizionamento analizzati nel corso di questo lavoro.

Formulazione del problema

Consideriamo un grafo $G = (V, E)$ con n vertici etichettati da $1, 2, \ldots, n$. Sia G non orientato e connesso. Una partizione di G è una suddivisione dei suoi vertici in sottoinsiemi a due a due disgiunti che prendono il nome di componenti della partizione. Consideriamo solo quelle partizioni di G in componenti che inducono sottografi connessi di G. Si parla in questo caso di *partizione connessa* del grafo. Indichiamo con:

- p il numero delle componenti di una partizione, $1 \leq p \leq n$;
- $\Pi_p(G)$ l'insieme di tutte le possibili partizioni (connesse) di G in p componenti non vuote;
- $\pi = (C_1, C_2, \ldots, C_p)$ una generica partizione in $\Pi_p(G)$.

Una p-partizione di G è una partizione $\pi \in \Pi_p(G)$. L'obiettivo del partizionamento può essere espresso per mezzo di una opportuna funzione $f : \Pi_p(G) \to \mathbb{R}^+$ che chiamiamo *funzione obiettivo*. Il valore minimo o massimo, a seconda del caso, della funzione obiettivo è detto *valore ottimo* e si denota con f^*. Una partizione $\pi \in \Pi_p(G)$ tale che $f(\pi) = f^*$ è una *partizione ottima* e viene indicata con π^*; essa non è in generale unica. Dato un grafo $G = (V, E)$ e un intero $1 \leq p \leq |V|$, il problema del partizionamento con funzione obiettivo f si risolve quindi nel trovare

$$\min\{f(\pi) : \pi \in \Pi_p(G)\}$$

oppure

$$\max\{f(\pi) : \pi \in \Pi_p(G)\}.$$

Equipartizione

Un primo gruppo di problemi significativi si ottiene associando ad ogni elemento di V un peso non negativo, definendo un'applicazione $w : V \to \mathbb{R}^+$. Un grafo di questo tipo modellizza spesso situazioni reali in cui si è interessati ad ottenere partizioni dell'insieme dei vertici in componenti che risultino omogenee rispetto al loro peso. Indichiamo con w_i il peso non negativo associato al vertice i; il peso di un sottoinsieme $S \subseteq V$ è invece definito da

$$W(S) := \sum_{i \in S} w_i.$$

Ha senso quindi considerare il peso $W(C)$ di una componente C e il peso medio delle componenti della partizione $\mu := W(V)/p$. L'obiettivo ideale, nel caso dei problemi di equipartizione, è quello di suddividere il grafo in p componenti tutte di uguale peso μ, ma naturalmente, trattandosi di un problema di ottimizzazione discreta, quasi mai tale configurazione rientra nello spazio delle soluzioni ammissibili. Attraverso l'ottimizzazione di diverse funzioni si può però tendere all'obiettivo ideale di una partizione uniforme.

L'obiettivo di *equipartizione* del grafo può essere conseguito convenientemente minimizzando una delle seguenti funzioni obiettivo:

norma L_1

$$f(\pi) = \sum_{k=1}^{p} |W(C_k) - \mu|; \tag{1}$$

norma L_2

$$f(\pi) = \sum_{k=1}^{p} (W(C_k) - \mu)^2; \tag{2}$$

norma L_∞

$$f(\pi) = \max_{k=1,\dots,p} |W(C_k) - \mu|. \tag{3}$$

Ottimizzando tali funzioni si cerca di ridurre al minimo lo scostamento del peso delle componenti della partizione dal peso medio μ.

L'equipartizione può essere ottenuta anche massimizzando il peso della componente di peso minimo o, viceversa, minimizzando il peso della componente di peso massimo. Formalmente, il problema consiste nel trovare una p-partizione $\pi \in \Pi_p(G)$ massimizzando, (rispettivamente minimizzando) le seguenti funzioni obiettivo:

max-min

$$f(\pi) = \min_{1 \le k \le p} W(C_k); \tag{4}$$

min-max

$$f(\pi) = \max_{1 \leq k \leq p} W(C_k). \tag{5}$$

Un ulteriore nucleo di problemi di equipartizione coinvolge le cosiddette *partizioni-*(L, U), partizioni in cui il peso di ciascuna componente è compreso tra i valori L ed U, con $L \leq U$. Un classico problema di questo tipo consiste nel cercare una partizione-(L, U) in p componenti del grafo dato, minimizzando la quantità $U - L$. Questo equivale a minimizzare la differenza tra il peso della componente più leggera ed il peso della componente più pesante; pertanto la funzione obiettivo (da minimizzare, appunto) può essere espressa con:

most uniform

$$f(\pi) = \max_{k=1,\ldots,p} W(C_k) - \min_{k=1,\ldots,p} W(C_k). \tag{6}$$

Il problema del partizionamento di un grafo per mezzo della funzione obiettivo most uniform è stato risolto nel caso di un cammino in [25], facendo riferimento a tre specifici problemi di partizionamento ad esso collegati. Dato un cammino con n vertici, tali problemi richiedono di trovare una partizione-(L, U) con:

- il minimo numero di componenti possibile;
- il massimo numero di componenti possibile;
- un numero fissato di componenti.

Indipendentemente dal loro utilizzo nella ricerca di una partizione più uniforme possibile, i problemi appena introdotti sono di per sè interessanti, non solo nel caso dei cammini, ma anche per altri tipi di grafi. Esiste inoltre un certo numero di applicazioni ad essi riconducibili.

Clustering

Assegnando ad ogni coppia di vertici del grafo un indice di dissimilarità, si ottiene un secondo importante gruppo di problemi di partizionamento, quello dei cosiddetti problemi di *clustering*. Una modellizzazione di questo tipo può essere utilizzata laddove l'interesse è quello di classificare un insieme di oggetti in sottoinsiemi, o cluster, accentuando uno o entrambi i seguenti aspetti:

omogeneità oggetti appartenenti ad uno stesso cluster devono essere il più possibile simili tra loro;

separazione oggetti appartenenti a cluster distinti devono essere quanto più possibile dissimili tra loro.

Le dissimilarità tra vertici sono espresse per mezzo di una matrice $D = (d_{ij})$, detta *matrice di dissimilarità*, definita da:

$$\begin{aligned} d_{ij} = d_{ji} \quad & \forall\, (i,j) \in V \times V \\ d_{ij} \geq 0 \quad & \forall\, (i,j) \in V \times V \\ d_{ii} = 0 \quad & \forall\, i \in V \end{aligned}$$

Possiamo formalizzare il problema come segue: dato un grafo $G = (V, E)$, con matrice di dissimilarità D, e dato un intero p, $1 \leq p \leq |V|$, trovare una partizione $\pi = (C_1, C_2, \ldots, C_p)$ che minimizzi o massimizzi un'opportuna funzione obiettivo $f(\pi)$ definita sull'insieme $\Pi_p(G)$.

L'omogeneità tra vertici in uno stesso cluster può essere ottenuta minimizzando una delle seguenti funzioni obiettivo:

maximum diameter

$$f(\pi) = \max_{k=1,\ldots,p} \max_{i,j \in C_k} d_{ij}; \tag{7}$$

sum of diameters

$$f(\pi) = \sum_{k=1}^{p} \max_{i,j \in C_k} d_{ij}; \tag{8}$$

inner-dissimilarity

$$f(\pi) = \sum_{k=1}^{p} \sum_{i,j \in C_k} d_{ij}. \tag{9}$$

Possiamo, invece, massimizzare una delle seguenti funzioni per perseguire l'obiettivo di separazione tra cluster:

minimum split

$$f(\pi) = \min_{k=1,\ldots,p} \min_{i \in C_k,\, j \notin C_k} d_{ij}; \tag{10}$$

sum of splits

$$f(\pi) = \sum_{k=1}^{p} \min_{i \in C_k,\, j \notin C_k} d_{ij}; \tag{11}$$

outer-dissimilarity

$$f(\pi) = \sum_{1 \leq h < k \leq p} \sum_{i \in C_h,\, j \in C_k} d_{ij}. \tag{12}$$

Altri problemi di partizionamento discreto

Trattiamo brevemente, qui di seguito, altri problemi classici di partizionamento presenti in letteratura.

Consideriamo un grafo $G(V, E)$ con pesi non negativi associati agli spigoli. Data una partizione $\pi \in \Pi_p(G)$, il taglio definito da π, che indichiamo con $cut_\pi(G)$ è dato dall'insieme degli archi che connettono vertici appartenenti a componenti distinte; il peso del taglio è invece dato dalla somma dei pesi degli spigoli appartenenti al taglio. In molte applicazioni si è interessati a trovare una p-partizione del grafo con taglio di peso minimo (massimo), obiettivo che si ottiene minimizzando (massimizzando):

cut

$$f(\pi) = \sum_{(i,j) \in cut_\pi(G)} l_{ij} \tag{13}$$

dove con l_{ij} denotiamo il peso dello spigolo (i, j).

Consideriamo, come sopra, un grafo pesato sugli spigoli. Indichiamo con p_{ij}, la lunghezza del più breve cammino che connette i vertici i e j. Può essere interessante considerare la seguente funzione obiettivo (da minimizzare):

p-median

$$f(\pi) = \sum_{k=1}^{p} \min_{i \in C_k} \sum_{j \in C_k} p_{ij}. \tag{14}$$

L'obiettivo, in questo caso, è quello di trovare una partizione del grafo tale che in ogni componente ci sia un vertice dal quale sia *agevole* raggiungere ogni altro vertice appartenente alla componente stessa.

Concludiamo accennando al problema del ricoprimento di un grafo per mezzo di cammini elementari (***path covering problem***). Il problema può essere posto in modi diversi. Dato un grafo $G = (V, E)$ si vuole trovare un insieme $\mathcal{C}$ di cammini elementari in G che soddisfi le seguenti condizioni:

tipo di ricoprimento L'insieme di cammini $\mathcal{C}$ su G deve ricoprire gli archi (*edge-covering*) oppure i vertici del grafo (*vertex-covering*).

vincoli di ricoprimento Due cammini qualsiasi in $\mathcal{C}$ non possano avere spigoli in comune (*edge-disjoint paths*) oppure nessun vertice del grafo può appartenere a più di un cammino in $\mathcal{C}$ (*vertex-disjoint paths*).

obiettivo Viene generalmente richiesto un ricoprimento del grafo che contenga il minimo numero possibile di cammini. In alcune applicazioni, ad ogni spigolo può essere associato un peso e l'obiettivo è quello di trovare un ricoprimento con massimo peso complessivo possibile.

Anche il problema di trovare il minimo numero di cammini ricoprenti per un dato grafo è di per sè interessante e non banale. In letteratura il problema viene chiamato ***path partition problem*** ed il minimo numero di cammini che ricoprono il grafo secondo i vincoli specificati prende il nome di ***path partition number***. Di fatto, molti lavori che trattano l'argomento, forniscono un metodo per trovare la cardinalità del ricoprimento minimo, ma tali metodi non sono costruttivi, nel senso che non permettono di trovare un ricoprimento ottimo. Si veda, ad esempio, [9].

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.

Un sottoinsieme $Y \subseteq A(T)$ si dice un *sottoalbero* se è chiuso e connesso. Se Y è un sottoalbero, 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 *max-min partizionamento continuo* di alberi. Indichiamo con $T_1, T_2, \ldots, T_p$ i sottoalberi di $A(T)$ indotti da una partizione continua π di T. Una max-min partizione ottima di T è una partizione che massimizza la funzione obiettivo

length max-min continuous

$$f(\pi) = \min_{1 \leq k \leq p} l(T_k)$$

Non ci risultano lavori che trattino l'analogo problema di *min-max*.

Complessità computazionale di un problema di partizionamento ottimo

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.

Contributi originali

La programmazione dinamica consiste in un metodo di risoluzione tabulare per problemi di ottimizzazione. Il problema dato viene suddiviso in sottoproblemi via via più piccoli, la cui soluzione viene calcolata procedendo dai sottoproblemi di dimensione minima ai sottoproblemi di dimensione maggiore, fino ad ottenere il valore della soluzione ottima. Nel corso della computazione i risultati ottenuti vengono memorizzati in una tabella e vengono pure memorizzate informazioni ausiliarie che permetterano, una volta calcolato il valore ottimo, di risalire ad una soluzione ottima ad esso associato. Può accadere che alcuni sottoproblemi abbiano uno o più sottoproblemi comuni. La programmazione dinamica si avvantaggia di questo tipo di struttura poiché risolve ciascun sottoproblema una volta per tutte: la risposta è memorizzata in una tabella e non viene ricalcolata ogni volta che deve essere risolto lo stesso problema. Anche gli algoritmi basati sul metodo *divide-et-impera* suddividono il problema iniziale in un certo numero di sottoproblemi, ma poi risolvono ricorsivamente i sottoproblemi, e infine fondono insieme le soluzioni ottenute per determinare la soluzione del problema originale. Una tecnica di tipo ricorsivo ricalcolerebbe, però, la soluzione di sottoproblemi comuni ogniqualvolta essi si ripresentassero, compromettendo l'efficienza dell'algoritmo che la impiega.

Partizionamento di cammini

In [3] è stato proposto un algoritmo di programmazione dinamica che risolve il problema dell'equipartizione di un cammino in p componenti connesse nel caso in cui la funzione obiettivo sia la norma L_1, la norma L_2, o la norma L_∞. Abbiamo generalizzato il procedimento descritto in [3] in modo da poterlo applicare ad un'ampia classe di funzioni obiettivo, in particolare alle funzioni obiettivo introdotte in (1)-(5) e in (7)-(14). La complessità computazionale dell'algoritmo (esatto) che abbiamo ottenuto è $O(pn^2)$.

Sia $P = (V, E)$ un cammino con n vertici. Osserviamo preliminarmente che esiste una corrispondenza biunivoca tra p-partizioni di P e sottoinsiemi di E con $p-1$ elementi. Infatti, una p-partizione è univocamente determinata dalla scelta di $p-1$ archi da *tagliare* e viceversa.

Sia π^* una p-partizione ottima di P con valore ottimo associato f^*. Un generico taglio definito da π^* suddivide P in due sottocammini P_1 e P_2 sui quali π^* induce una p_1-partizione e una p_2-partizione rispettivamente, tale che $p_1 + p_2 = p$. Il problema del p-partizionamento del cammino P viene quindi spezzato nei due problemi di p_1-partizionamento e p_2-partizionamento dei cammini P_1 e P_2. Inoltre π^* definisce su P_1 e P_2 una partizione ottima rispettivamente in p_1 e p_2 componenti. Se così non fosse, esisterebbe una partizione $\tilde{\pi}$ di P, ottenuta combinando insieme le partizioni ottime di P_1 e P_2, con valore $f(\tilde{\pi})$ migliore di f^*. Abbiamo in questo modo caratterizzato la sottostruttura di una soluzione ottima come sottostruttura ottima e possiamo dunque tentare di definire una algoritmo di programmazione dinamica che risolva il problema in tempo polinomiale.

Supponiamo che la funzione obiettivo sia della forma:

$$f(\pi) = \theta(C_1) * \theta(C_2) * \cdots * \theta(C_p) \tag{15}$$

con $*$ operazione binaria associativa e commutativa su $\mathbb{R}^+$, e θ applicazione che associa ad ogni possibile sottocammino di P un elemento in $\mathbb{R}^+$. Senza perdita di generalità, supponiamo che f debba essere minimizzata (altrimenti basterà sostituire ad ogni calcolo di minimo un calcolo di massimo). Data una matrice quadrata $B = (\beta_{ij})$ di ordine m e un vettore $\bar{v} = (v_1, \ldots, v_m)^t$ di lunghezza m, definiamo la seguente operazione dipendente da f:

$$B *_f \bar{v} := \begin{pmatrix} \min(\beta_{11} * v_1, \beta_{12} * v_2, \ldots, \beta_{1m} * v_m) \\ \min(\beta_{22} * v_2, \ldots, \beta_{2m} * v_m) \\ \vdots \\ \min(\beta_{mm} * v_m) \end{pmatrix} \tag{16}$$

Sia ora $A = (\alpha_{ij})$ la matrice quadrata di ordine n definita da:

$$\alpha_{ij} = \begin{cases} \theta(C_{ij}) & \text{per } i \leq j \\ 0 & \text{altrimenti} \end{cases} \tag{17}$$

dove con C_{ij} indichiamo la componente (o sottocammino) $\{i, i+1, \ldots, j\}$. Infine, sia $A^{(k)}$, con $k = 2, \ldots, p-1$, la sottomatrice quadrata di A di ordine $m := n - p + 1$ ottenuta selezionando m righe ed m colonne contigue di A a partire dalla k-sima riga e dalla k-sima colonna, e sia $\bar{a}^{(p)}$ il vettore ottenuto selezionando l'ultima colonna e le ultime m righe di A.

La seguente procedura calcola il valore della soluzione ottima di un problema di partizionamento di un cammino in p componenti per una qualsiasi

funzione obiettivo della forma (15):

$$\begin{aligned}
&\bar{a}^{(p-1)} = A^{(p-1)} *_f \bar{a}^{(p)} \\
&\bar{a}^{(p-2)} = A^{(p-2)} *_f \bar{a}^{(p-1)} \\
&\vdots \\
&\bar{a}^{(2)} = A^{(2)} *_f \bar{a}^{(3)} \\
&f^* = \min(\alpha_{11} * a_1^{(2)}, \alpha_{12} * a_2^{(2)}, \ldots, \alpha_{1m} * a_m^{(2)})
\end{aligned} \quad . \tag{18}$$

Per trovare, oltre al valore ottimo, anche una partizione ottima, occorre solo modificare lievemente la procedura. In corrispondenza con l'operazione (16), si consideri un vettore $\bar{c} = (c_1, c_2, \ldots, c_m)$. Se $\bar{v}' = (v'_1, v'_2, \ldots, v'_m) = B *_f \bar{v}$, definiamo c_i come un indice k qualsiasi per il quale si abbia $\beta_{ik} * v_k = v'_i$. A ciascun $a^{(t)}$, con $2 \leq t \leq p-1$, si associ un vettore $c^{(t)}$ calcolato in questo modo, in base all'operazione che definisce $a^{(t)}$ e sia inoltre c l'indice di una componente di $(\alpha_{11} * a_1^{(2)}, \alpha_{12} * a_2^{(2)}, \ldots, \alpha_{1m} * a_m^{(2)})$ pari a $f*$. Sia T_i, con $1 \leq i \leq p-1$, il nodo dopo il quale è posto l'i-esimo taglio. La seguente procedura permette di determinare una partizione ottima di P:

$$\left.\begin{aligned}
&T_1 = c \\
&i = T_{k-1} - k + 2 \\
&T_k = T_{k-1} + c_i^{(k)}.
\end{aligned}\right\} \qquad \text{per } k = 2, \ldots, p-1. \tag{19}$$

Assumendo di poter calcolare la matrice A in (17) con un numero di operazioni di ordine non superiore a $O(pn^2)$ e assumendo che l'operazione $*$ abbia costo $O(1)$, le procedure (18) e (19) permettono di risolvere il problema del partizionamento di un cammino in un numero predefinito di componenti in tempo $O(pn^2)$ per tutte le funzioni obiettivo della forma (15). La tabella 1 mostra che le (1)-(5) e le (7)-(14) sono di questa forma; inoltre, per queste funzioni obiettivo il calcolo della matrice A ha costo non superiore a $O(n^2)$ e l'operazione $*$ ha sempre costo $O(1)$.

Mostriamo, a titolo di esempio, come calcolare A per la funzione obiettivo *inner-dissimilarity*. Sia $D = (d_{ij})$ una matrice di dissimilarità. Dalla tabella 1, α_{ij} è la somma delle dissimilarità tra tutte le possibili coppie di vertici appartenenti alla componente $\{i, \ldots, j\}$, per $i \leq j$:

$$\alpha_{ij} = \begin{cases} \sum_{i \leq h \leq k \leq j} d_{hk} & \text{per } i \leq j \\ 0 & \text{altrimenti} \end{cases} \tag{20}$$

Per ogni i e j, $1 \leq i \leq j \leq n$, definiamo la seguente matrice quadrata M_{ij}, ottenuta direttamente da D selezionandone le righe e le colonne comprese tra i e j, e ponendo a zero tutti gli elementi al di sotto della diagonale

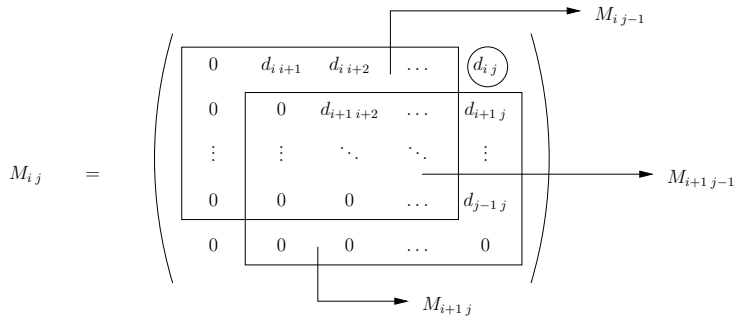

Figura 1: Il termine α_{ij} si ottiene sommando tutti gli elementi della matrice M_{ij}, la quale contiene al suo interno le matrici $M_{i\,j-1}$, $M_{i+1\,j}$ e $M_{i+1,j-1}$. A queste ultime sono associati i termini $\alpha_{i\,j-1}$, $\alpha_{i+1\,j}$ e $\alpha_{i+1,j-1}$ rispettivamente, che possono essere utilizzati, una volta noti, per il calcolo in tempo costante del termine cercato, come indicato dalla formula (22).

principale:

$$M_{ij} := \begin{pmatrix} 0 & d_{i\,i+1} & d_{i\,i+2} & \dots & d_{i\,j} \\ 0 & 0 & d_{i+1\,i+2} & \dots & d_{i+1\,j} \\ \vdots & \vdots & \ddots & \ddots & \vdots \\ 0 & 0 & 0 & \dots & d_{j-1\,j} \\ 0 & 0 & 0 & \dots & 0 \end{pmatrix} \tag{21}$$

È chiaro che α_{ij} si ottiene sommando tra loro tutti gli elementi della matrice M_{ij}. Precisamente:

$$\alpha_{ij} = \sum_{k=i}^{j-1}(d_{k\,k+1} + d_{k\,k+2} + \ldots + d_{k\,j}) = \sum_{k=i}^{j-1}\sum_{h=k+1}^{j} d_{k\,h}.$$

Se M_{ij} ha dimensione 1 o 2, non c'è nulla da calcolare: $\alpha_{ii} = d_{ii} = 0$ per ogni i, $1 \le i \le n$, e $\alpha_{i\,i+1} = d_{i\,i+1}$ per ogni i, $1 \le i \le n-1$. Fissiamo ora $\hat{\imath}$ e $\hat{\jmath}$ tali che $\hat{\imath} \le \hat{\jmath}$ e $k := \hat{\jmath} - \hat{\imath} + 1 \ge 3$; stiamo cioè supponendo che la matrice $M_{\hat{\imath}\hat{\jmath}}$ abbia dimensione maggiore o uguale a tre. Supponiamo inoltre di aver calcolato α_{ij} per ogni i e j tali che $j - i + 1 < k$ (ovvero per ogni i e j tali che $M_{i,j}$ abbia dimensione strettamente inferiore a k).

Allora $\alpha_{\hat{\imath}\hat{\jmath}} = \alpha_{\hat{\imath}\,\hat{\jmath}-1} + \alpha_{\hat{\imath}+1\,\hat{\jmath}} - \alpha_{\hat{\imath}+1\,\hat{\jmath}-1} + d_{\hat{\imath}\,\hat{\jmath}}$. I termini al secondo membro sono associati a matrici di dimensione strettamente inferiore a k e quindi, per quanto supposto, α_{ij} può essere calcolato in tempo $O(1)$. Riassumendo:

$$\alpha_{ij} = \begin{cases} 0 & \text{per } i = j; \\ d_{ij} & \text{per } j = i+1; \\ \alpha_{i\,j-1} + \alpha_{i+1\,j} - \alpha_{i+1\,j-1} + d_{i\,j} & \text{altrimenti.} \end{cases} \tag{22}$$

La figura 1 aiuta a comprendere il procedimento. Il calcolo di ciascun

α_{ij} richiede tempo costante e si dimostra che il numero totale di termini α_{ij} da calcolare è dell'ordine di $O(n^2 - p^2)$.

Most uniform su cammini

La funzione obiettivo *most uniform* è considerata da molti autori la migliore funzione per l'equipartizione di un grafo. Ottimizzare la (6) significa, infatti, minimizzare la differenza tra il peso della componente più leggera e il peso della componente più pesante della partizione. Il relativo problema di partizionamento è stato finora considerato più difficile di altri problemi di equipartizione. Nel caso di un albero il problema risulta ancora aperto, mentre nel caso di un cammino è stato proposto in [25] un algoritmo di complessità computazionale $O(pn^2 \log n)$. L'algoritmo si basa sulla risoluzione ripetuta di un problema di partizionamento correlato, quello della ricerca di una partizione-(L, U) in p componenti. Una risoluzione per mezzo della programmazione dinamica è stata considerata, ma solo passando per i problemi appena citati. Questa strada non ha permesso di ottenere un algoritmo di complessità accettabile. Applicando la procedura descritta nella sezione precedente siamo riusciti a dimostrare che passare attraverso problemi di partizionamento ausiliari non è necessario: il problema del partizionamento di un cammino con la funzione obiettivo *most uniform* può essere affrontato in via diretta.

Applicando la procedura descritta nella sezione precedente, siamo riusciti a dimostrare che passare attraverso problemi di partizionamento ausiliari non è necessario: il problema del partizionamento di un cammino con la funzione obiettivo *most uniform* può essere affrontato in via diretta.

La generalizzazione dell'algoritmo in [3] appena presentata contiene al suo interno la strategia risolutiva per il problema di partizionamento di cammini legato alla funzione obiettivo (6), che può allora essere risolto con $O(pn^2)$ operazioni elementari.

Sia A definita come in (17) con $\theta(C_{ij}) = W(C_{ij})$ Sia $A^{(k)}$, con $k = 2, \ldots, p-1$, la sottomatrice quadrata di A di ordine m ottenuta selezionando m righe ed m colonne contigue di A a partire dalla k-sima riga e dalla k-sima colonna.

Per ogni $k \in \{p, p-1, \ldots, 2\}$ vengono introdotti due vettori $\bar{a}^{(k)}_{\max}$ e $\bar{a}^{(k)}_{\min}$. La differenza elemento per elemento

$$\bar{a}^{(k)}_{\max} - \bar{a}^{(k)}_{\min} := ((\bar{a}^{(k)}_{\max})_1 - (\bar{a}^{(k)}_{\min})_1, \ldots, (\bar{a}^{(k)}_{\max})_m - (\bar{a}^{(k)}_{\min})_m)$$

corrisponde ai contributi dati alla funzione obiettivo dalle migliori partizioni possibili calcolate per ogni possibile sottocammino relativo alle componenti $C_i, C_{i+1}, \ldots, C_p$.

Si ponga $\bar{a}^{(p)}_{\max} = a^{(p)}_{\min} = (\alpha_{p\,n}, \alpha_{p+1\,n}, \ldots, \alpha_{n\,n})^t$ ovvero pari al vettore colonna corrispondente alle ultime m righe dell'n-sima colonna di A. Per

$k = p-1, \ldots, 2$, il calcolo di $(a^{(k)}_{\max})_i$ e $(a^{(k)}_{\min})_i$, l'i-esimo elemento di $a^{(k)}_{\max}$ e di $a^{(k)}_{\min}$ rispettivamente, si basa sulla seguente procedura in tre passi:

1. Si determinano due vettori, $W_{\max}$ e $W_{\min}$, di lunghezza $m-i+1$, come segue:

$$W_{\max} = \begin{pmatrix} \max\{A^k_{i\,i}, (a^{k+1}_{\max})_i\} \\ \max\{A^k_{i\,i+1}, (a^{k+1}_{\max})_{i+1}\} \\ \vdots \\ \max\{A^k_{i\,m}, (a^{k+1}_{\max})_m\} \end{pmatrix}$$
$$W_{\min} = \begin{pmatrix} \min\{A^k_{k\,i}, (a^{k+1}_{\min})_i\} \\ \min\{A^k_{k\,i+1}, (a^{k+1}_{\min})_{i+1}\} \\ \vdots \\ \min\{A^k_{i\,m}, (a^{k+1}_{\min})_m\} \end{pmatrix} \tag{23}$$

2. Si determina l'intero $t \in \{1, 2, \ldots, m-i+1\}$ tale che

$$(W_{\max})_t - (W_{\min})_t = \min_{s=1,\ldots,m-i+1} [(W_{\max})_s - (W_{\min})_s] \tag{24}$$

3. Si pone

$$\begin{array}{l} (a^{(k)}_{\max})_i = (W_{\max})_t \\ (a^{(k)}_{\min})_i = (W_{\min})_t \\ (Component/Case)_{k\,i} = t. \end{array} \tag{25}$$

Data una componente C_k, $1 \le k \le p-1$, $(Component/Case)_{k,i}$ indica il numero ottimo di vertici nella componente C_k in una partizione ottima dei vertici $k+i-1, \ldots, n$, nelle $p-k+1$ componenti $C_k, C_{k+1}, \ldots, C_p$.

A questo punto possiamo riportare l'intero algoritmo:

DinamicPathMostUniform(P, w, p)

1: calcola A
2: $\bar{a}^{(p)}_{\max} = (\alpha_{p\,n}, \alpha_{p+1\,n}, \ldots, \alpha_{n\,n})^t$
3: $a^{(p)}_{\min} = (\alpha_{p\,n}, \alpha_{p+1\,n}, \ldots, \alpha_{n\,n})^t$
4: **per** $k = p-1, \ldots, 2$ **ripeti**
5: **per** $i = 1, \ldots, m$ **ripeti**
6: determina $(a^{(k)}_{\max})_i$, $(a^{(k)}_{\min})_i$ e $(Component/Case)_{k,i}$ con la procedura in tre passi (23)-(25)
7: determina $(a^{(1)}_{\max})_1$, $(a^{(1)}_{\min})_1$ e $(Component/Case)_{1,1}$
8: $f^* = (a^{(1)}_{\max})_1 - (a^{(1)}_{\min})_1$
9: $T =$ ConstructOptimalPartition$(Component/Case, p)$
10: **restituisci** f^*, T

La procedura ConstructOptimalPartition calcola il vettore dei tagli di una p-partizione ottima del cammino P, inferendolo dalla tabella $Component/Case$, come mostrato dalla (19).

Altre tecniche di partizionamento

Per alcuni problemi di ottimizzazione, l'uso di tecniche di programmazione dinamica per decidere ad ogni passo la scelta migliore risulta particolarmente oneroso; lo stesso risultato potrebbe essere ottenuto con algoritmi più semplici ed efficienti.

Gli *algoritmi greedy* adottano la strategia di prendere quella decisione che, al momento, appare come la migliore. In altri termini, viene sempre presa quella decisione che, localmente, è la decisione ottima, senza preoccuparsi del fatto che tale decisione possa portare o meno ad una soluzione ottima del problema nella sua globalità.

L'algoritmo per la determinazione di una p-partizione-(L, U) di un cammino proposto in [25], ad esempio, adotta una strategia di tipo greedy e viene eseguito in tempo lineare nel numero di vertici del cammino. L'algoritmo è molto intuitivo e la sua correttezza viene dimostra utilizzando proprietà teoriche degli insiemi parzialmente ordinati. Utilizzando la programmazione dinamica si riesce solo a definire un algoritmo di complessità computazionale $O(pn^2)$, troppo alta se si vuole utilizzare ripetutamente la soluzione di questo problema al fine del partizionamento ottimo di un cammino con la funzione obiettivo *most uniform*, come è stato fatto in [25].

Il problema di clustering definito dalla funzione obiettivo *inner dissimilarity* è NP-completo anche nel caso particolare in cui il grafo da partizionare sia una stella. Con una semplice euristica greedy è però possibile ottenere una partizione di un albero in p componenti con scostamento del valore della funzione obiettivo dal valore ottimo non superiore a $\frac{1}{e}$, garantito a priori. La submodularità di una funzione d'insiemi associata alla (9) assicura questo risultato.

Un'altra tecnica molto utilizzata per risolvere problemi di partizionamento è quella della migrazione di gruppi: dopo aver generato una partizione iniziale si tenta il suo miglioramento attraverso lo scambio di un gruppo di vertici tra due o più componenti della partizione; il *gruppo di migrazione* è determinato sulla base di scelte locali e la scelta contempla solitamente un passo di tipo greedy.

Nel caso degli alberi, esiste una corrispondenza biunivoca tra sottoinsiemi di cardinalità $p-1$ dell'insieme degli spigoli del grafo e p-partizioni. Un caso particolare di migrazione di gruppi si ottiene *tagliando*[1] $p-1$ spigoli del grafo e spostando ad ogni passo un *taglio* dallo spigolo cui è assegnato ad uno spigolo adiacente. Un algoritmo di questo tipo prende il nome di

[1] *tagliare* uno spigolo significa eliminarlo dall'insieme degli spigoli del grafo.

algoritmo di *shifting*. In [27] viene descritto un algoritmo di shifting che risolve in maniera esatta il problema del *max-min* partizionamento ottimo di un albero in tempo $O((p-1)^2 \cdot r(T) + (p-1) \cdot n)$, dove $r(T)$ è il raggio dell'albero. L'algoritmo permette operazioni di *shifting* in una sola *direzione di percorrenza dell'albero*, contenendo di fatto la complessità dell'algoritmo. Per l'analogo problema di *min-max*, operazioni di shifting in una sola direzione non sono sufficienti: occorrono anche *shift laterali* di aggiustamento della partizione, come argomentato in [7].

La tecnica della migrazione di gruppi e, in particolare, le tecniche di shifting, possono talvolta essere migliorate se si ammettono partizioni che peggiorano il valore della funzione obiettivo (*simulated annealing*). In alcuni problemi di partizionamento, cercare ad ogni passo esclusivamente di migliorare la funzione obiettivo può portare a non intersecare alcuna partizione ottima. Permettere peggioramenti della funzione obiettivo può invece consentire di superare eventuali minimi locali. Il criterio di interruzione della ricerca deve essere però basato su condizioni più forti e restrittive che non la sola valutazione del valore della funzione obiettivo. Una tecnica di questo tipo è stata impiegata in [24] per la risoluzione del problema del p-partizionamento di un cammino con la norma L_∞ e ha permesso di definire un algoritmo di complessità computazionale $O(pn \log p)$, migliore quindi dell'algoritmo di programmazione dinamica descritto in [3] per la risoluzione dello stesso problema.

Funzione obiettivo	$\boldsymbol{f(\pi)}$	$\boldsymbol{\alpha_{ij}}$	***operazione*** $*$
norma L_1	$\sum_{k=1}^{p} \lvert W(C_k) - \mu \rvert$	$\left\lvert \sum_{h\in\{i,\dots,j\}} w_k - \mu \right\rvert$	$+$
norma L_2	$\sum_{k=1}^{p} (W(C_k) - \mu)^2$	$\left(\sum_{h\in\{i,\dots,j\}} w_k - \mu \right)^2$	$+$
norma L_∞	$\max_{k=1,\dots,p} \lvert W(C_k) - \mu \rvert$	$\left\lvert \sum_{h\in\{i,\dots,j\}} w_k - \mu \right\rvert$	max
max-min	$\min_{1\le k\le p} W(C_k)$	$\sum_{h\in\{i,\dots,j\}} w_k$	min
min-max	$\max_{1\le k\le p} W(C_k)$	$\sum_{h\in\{i,\dots,j\}} w_k$	max
maximum diameter	$\max_{k=1,\dots,p} \max_{i,j\in C_k} d_{ij}$	$\max_{h,k\in\{i,\dots,j\}} d_{hk}$	max
sum of diameters	$\sum_{k=1}^{p} \max_{i,j\in C_k} d_{ij}$	$\max_{h,k\in\{i,\dots,j\}} d_{hk}$	$+$
inner dissimilarity	$\sum_{k=1}^{p} \sum_{i,j\in C_k} d_{ij}$	$\sum_{\substack{h,k\in\{i,\dots,j\} \\ h\le k}} d_{hk}$	$+$
minimum split	$\min_{k=1,\dots,p} \min_{\substack{i\in C_k, \\ j\notin C_k}} d_{ij}$	$\min_{\substack{h\in\{i,\dots,j\} \\ k\notin\{i,\dots,j\}}} d_{hk}$	min
sum of splits	$\sum_{k=1}^{p} \min_{\substack{i\in C_k, \\ j\notin C_k}} d_{ij}$	$\min_{\substack{h\in\{i,\dots,j\} \\ k\notin\{i,\dots,j\}}} d_{hk}$	$+$
outer dissimilarity	$\sum_{1\le h<k\le p} \sum_{\substack{i\in C_h, \\ j\in C_k}} d_{ij}$	$\frac{1}{2} \sum_{\substack{h\in\{i,\dots,j\} \\ k\notin\{i,\dots,j\}}} d_{hk}$	$+$
p-median	$\sum_{k=1}^{p} \min_{i\in C_k} \sum_{j\in C_k} p_{ij}$	$\min_{h\in\{i,\dots,j\}} \sum_{k\in\{i,\dots,j\}} p_{hk}$	$+$
cut	$\sum_{(i,j)\in\, cut(G)} l_{ij}$	$l_{j\,j+1}$	$+$

Tabella 1: Le funzioni obiettivo (1)-(5) e (7)-(14) sono della forma (15) con $\theta(C_{ij}) = \alpha_{ij}$ e $*$ come indicati in tabella.

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] E. L. Aparo, B. Simeone, *Un algoritmo di equipartizione e il suo impiego in un problema di contrasto ottico*, Ricerca Operativa 6, Milano, 1-12, 1973.

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

[5] R. I. Becker, I. Lari, M. Lucertini, B. Simeone, *Max-min partitioning of grid graphs into connected components*, Networks 32, 155-125, 1998.

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

[7] R. I. Becker, Y. Perl, S. R. Schach, *A shifting algorithm for min-max tree partitioning*, Journal of the ACM 29, 58-67, 1982.

[8] J. Bishop, *Java gently*, third edition, Addison-Wesley, 2001.

[9] J.-H. Yan, G. J. Chang, *Information Processing Letters* 52, 317-322, 1994.

[10] 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.

[11] T. H. Cormen, C. E. Leiserson, R. L. Rivest, C. Stein, *Introduzione agli algoritmi e strutture dati*, seconda edizione, McGraw-Hill, 2005.

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

[13] C. De Simone, M. Lucertini, S. Pallottino, B. Simeone, *Fair dissections of spiders, worms, and caterpillars*, Networks 20, 323-344, 1990.

[14] I. Finocchi, R. Petreschi, *Divider-based algorithms for hierarchical tree partitioning*, Discrete Applied Mathematics 136, 227-247, 2004.

[15] M. L. Fisher, G. L. Nemhauser, L. A. Wolsey, *Analysis of approximations for maximizing a submodular set function*, Mathematical programming 14, 265-294, 1978.

[16] M. R. Garey, D. S. Johnson, *Computers and intractability. A guide to the theory of NP-completeness. A Series of Books in the Mathematical Sciences*, W.H.Freeman and Company, San Francisco, 1979.

[17] M. Garland, G. Kumar *Visual exploration of complex time-varying graphs*, IEEE Transactions on Visualization and Computer Graphics 12, Issue 5, 805-812, 2006.

[18] G. Grätzer, *General Lattice Theory*, second edition, Birkhäuser, 1998.

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

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

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

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

[23] R. M. Karp, *Reducibility among combinatorial problems*. Complexity of computer computations, R. E. Miller and J. W Thatcher, Eds., Plenum Press. New York, 85-104, 1972.

[24] M. Liverani, A. Morgana, B. Simeone, G. Storchi, *Path equipartition in the Chebyshev norm*, European Journal of Operational Research 123, 428-436, 2000.

[25] M. Lucertini, Y. Perl, B. Simeone, *Most uniform path partitioning and its use in image processing*, Discrete Applied Mathematics 42, 227-256, 1993.

[26] M. Maravalle, R. Naldini, B. Simeone, *Clustering on trees*, Computational Statistics & Data Analysis 24, 217-234, 1997.

[27] Y. Perl, S. R. Schach, *Max-min tree partitioning*, Journal of the ACM 28, 5-15, 1981.

[28] Y. Perl, U. Vishkin, *Efficient implementation of a shifting algorithm*, Discrete Applied Mathematics 12, 71-80, 1985.

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