Capitolo 1

Algoritmi di approssimazione

1.1 Introduzione

Esistono molti problemi NP-completi di grande importanza. È possibile affrontare la risoluzione di questi in vari modi:

- se l'input del problema è sufficientemente piccolo l'algoritmo esponenziale potrebbe essere affrontabile (algoritmi pseudopolinomiali, ad esempio per SUBSET-SUM);
- è possibile restringere la classe delle istanze a casi particolari per i quali esistono degli algoritmi polinomiali;
- ricerca esaustiva dello spazio delle soluzioni guidata da strategie atte a eliminare parte delle soluzioni (ad esempio, Branch-and-Bound);
- è possibile trovare approcci che permettono di trovare soluzioni *vicine all'ottimo* in tempo polinomiale: in questo caso si parla di *algoritmi di approssimazione*.

Sia dato il problema (di ottimizzazione) $\Pi \subseteq I \times S, s(i) \in S, i \in I$, sia $c : I \mapsto \mathbb{R}^+$ una funzione di costo e sia $A_\pi(i) \in s(i)$ l'algoritmo di approssimazione che risolve il problema dato (in modo approssimato). Allora, un algoritmo è di $\rho(\mathbf{n})$ **approssimazione** se vale

$$\forall i \in I : |i| = n \quad \max\left\{\frac{c(s^*(i))}{c(A_\pi(i))}, \frac{c(A_\pi(i))}{c(s^*(i))}\right\} \leq \rho(n)$$

e $\rho(n)$ è detto **fattore di approssimazione**.

Data l'assunzione che il costo delle soluzioni sia sempre positivo, i rapporti nella funzione di max sono sempre sensati. Per un problema di MAX, si ha che $0 < c(A_\pi(i)) \leq c(s^*(i))$, per cui il fattore di approssimazione sarà $\frac{c(s^*(i))}{c(A_\pi(i))}$; di converso, per un problema di MIN, si ha che $0 < c(s^*(i)) \leq c(A_\pi(i))$, per cui il fattore di approssimazione sarà $\frac{c(A_\pi(i))}{c(s^*(i))}$. Si noti che il rapporto di approssimazione non può mai essere minore di 1: il valore minimo che può raggiungere è

1 e ciò avviene quando l'algoritmo di approssimazione restituisce una soluzione ottima.

Alcuni problemi NP-completi ammettono algoritmi di approssimazione in tempo polinomiale che riescono a raggiungere fattori di approssimazione sempre migliori mano a mano che si usa un tempo computazionale maggiore.

Definizione. *Uno* ***schema di approssimazione*** *per un problema di approssimazione è un algoritmo di approssimazione che prende in input l'istanza del problema e un valore $\epsilon > 0$ tale per cui, per ogni ϵ fissato, lo schema è un algoritmo di (1+ϵ)-approssimazione.*

Definizione. *Uno schema di approssimazione è uno* ***schema di approssimazione in tempo polinomiale (PTAS)*** *se, fissato $\epsilon > 0$, lo schema esegue in tempo polinomiale rispetto alla taglia dell'istanza.*

Il tempo di esecuzione di uno schema di approssimazione può aumentare molto velocemente al diminuire di ϵ: ad esempio $T = O(n^{2/\epsilon})$. Idealmente, se ϵ decresce di un fattore costante, il tempo necessario per raggiungere l'approssimazione desiderata non dovrebbe crescere più che di un fattore costante.

Definizione. *Uno schema di approssimazione è uno* ***schema di approssimazione in tempo pienamente polinomiale (FPTAS)*** *se è uno schema di approssimazione e il tempo di esecuzione è polinomiale sia nella taglia n dell'istanza sia rispetto a $1/\epsilon$.*

Un possibile esempio di uno schema FPTAS è $T = O((1/\epsilon)^2 n^3)$. In tal modo se ϵ decresce di un fattore costante allora il tempo di esecuzione cresce anch'esso di un fattore costante.

1.2 Il problema VERTEX-COVER

1.2.1 Un algoritmo di 2-approssimazione

Sia dato il problema NP-completo VERTEX-COVER, definito come segue:

$$\begin{cases} I = \langle G = (V, E) \rangle, \text{G grafo non diretto}, |V| = n, |E| = m \\ \\ \text{Determinare il sottoinsieme } V^* \subseteq V \text{ di cardinalità minima tale che} \\ \forall u, v \in V \text{ se } \{u, v\} \in E \Rightarrow (u \in V^*) \vee (v \in V^*) \end{cases}$$

In *Dati e algoritmi 2* abbiamo dimostrato che VERTEX-COVER è NPH tramite la riduzione polinomiale CLIQUE $<_p$ VERTEX-COVER che opera la trasformazione

$$\langle G = (V, E) \rangle \longrightarrow \langle G^c = (V, E^c) \rangle$$

dove G^c rappresenta il grafo complementare di G. Allora, si noti che se un grafo contiene una CLIQUE V^* di cardinalità massima, allora il suo complementare contiene un VERTEX-COVER di cardinalità minima $V - V^*$.

Un possibile algoritmo risolutivo potrebbe sfruttare una strategia greedy:

- $E = \{e_1, e_2, ..., e_m\}$
- scelta greedy $e_1 = \{u, v\}$

- $V' \longleftarrow \{u, v\}$
- CLEAN-UP: elimino tutti gli archi che hanno almeno un loro vertice in V'
- termino quando $E = \emptyset$

Lo pseudo-codice di tale algoritmo è riportato come algoritmo 1.

Algorithm 1 Algoritmo di approssimazione per VERTEX-COVER

```
function APPROX_VC((G = (V, E))
    V' ← ∅
    E' ← E
    A ← ∅
    while E' ≠ ∅ do
        ** let {u, v} ∈ E' **
        A ← A ∪ {{u, v}}
        V' ← V' ∪ {u, v}
        E' ← E' − {e ∈ E' | ∃z ∈ V : (e = {u, z}) ∨ (e = {v, z})}
    end while
    return V'
end function
```

L'algoritmo prende in ingresso il grafo di cui si vuole determinare il vertex cover di cardinalità minima. All'inizio istanzia l'insieme V' che alla fine conterrà i vertici che formeranno il vertex cover, effettua una copia dell'insieme dei lati del grafo e crea l'insieme A che conterrà le scelte greedy effettuate dall'algoritmo (tornerà utile per determinare il fattore di approssimazione). L'algoritmo esegue fintantoché E' contiene dei lati: la scelta greedy viene aggiunta all'insieme A e i vertici estremi del lato selezionato dalla scelta greedy vanno a far parte del vertex cover V'. Infine, la fase di clean-up rimuove tutti i lati che hanno origine dai due vertici che sono gli estremi del lato selezionato dalla scelta greedy.

Analisi della correttezza L'algoritmo termina quando $E' = \emptyset$, quindi ogni arco viene eliminato in una qualche iterazione. Considerata una iterazione, il lato e viene eliminato poiché almeno uno dei due suoi estremi è un vertice già appartenente al vertex cover V': viene infatti eliminato l'arco selezionato dalla scelta greedy assieme a tutti i lati uscenti dai due estremi della scelta greedy (il vertice di "origine" fa già parte di V'). Poiché tale procedura avviene per ogni arco, allora l'insieme V' che ottengo è un vertex cover.

Analisi temporale Memorizzando il grafo tramite liste di adiacenza, l'algoritmo può operare in $T = \Theta(n + m) = \Theta(|\langle G \rangle|)$. La scelta greedy corrisponderà alla prima lista "nodale" non vuota: determinata tale lista (scansione lineare), si scansiona la lista concatenata associata a tale lista per determinare i lati da rimuovere (fase di clean-up).

Fattore di approssimazione Per determinare il fattore di approssimazione torna utile la raccolta delle scelte greedy effettuate dall'algoritmo (l'insieme A nel codice).

Proposizione. *L'insieme A forma un matching:* $\forall e, e' \in A \ : \ e \cap e' = \emptyset$. *Inoltre tale matching è massimale.*

Dimostrazione. Per assurdo, suppongo che $\exists e, e' \in A : e \cap e' \neq \emptyset$. Ipotizzando, senza perdita di generalità, che e venga selezionato prima di e', quando e viene selezionato, e' viene eliminato dalla fase di clean-up avendo un vertice in comune con e e di conseguenza non verrebbe selezionato in una successiva iterazione, assurdo.

Se il matching è massimale significa che $\forall f \in E \ : \ f \notin A$ l'insieme $A \cup \{f\}$ non è più un matching. Poiché il lato f non fa parte di A, allora in una qualche iterazione è stato eliminato dalla fase di clean-up dell'algoritmo perché un suo vertice era in comune con uno dei vertici del lato selezionato come scelta greedy nella stessa iterazione: tale lato e è stato aggiunto ad A, pertanto, se aggiungessimo anche f avremmo sicuramente che $e \cap f \neq \emptyset$, violando così la definizione di matching. Segue quindi che A è un matching massimale. □

Fatto questo è ora possibile dimostrare la seguente proposizione.

Proposizione. *L'algoritmo APPROX_VC ha come fattore di approssimazione* $\rho(n) = 2$.

Dimostrazione. Iniziamo con il dimostrare che $|V^*| \geq |A|$. Se $e = \{u, v\} \in A$ allora qualunque vertex cover, incluso quindi quello ottimo, deve contenere u o v (o entrambi), altrimenti il lato e non sarebbe coperto. Di conseguenza ogni vertex cover contiene **almeno** un estremo di ogni lato del matching, da cui la tesi.

Proseguiamo con il dimostrare che $|V'| = 2|A|$. Ad ogni iterazione, considerata la scelta greedy $\{u, v\}$, a V' vengono aggiunti i vertici estremi di quel lato, mentre il lato stesso viene aggiunto ad A. Date le caratteristiche dei due insiemi, uno un vertex cover, l'altro un matching, e gli elementi che vengono aggiunti ad ogni iterazione, V' conterrà necessariamente il doppio degli elementi di A (di fatto V' è la collezione di nodi che formano i lati di A).

Combinando i risultati ottenuti si ha:

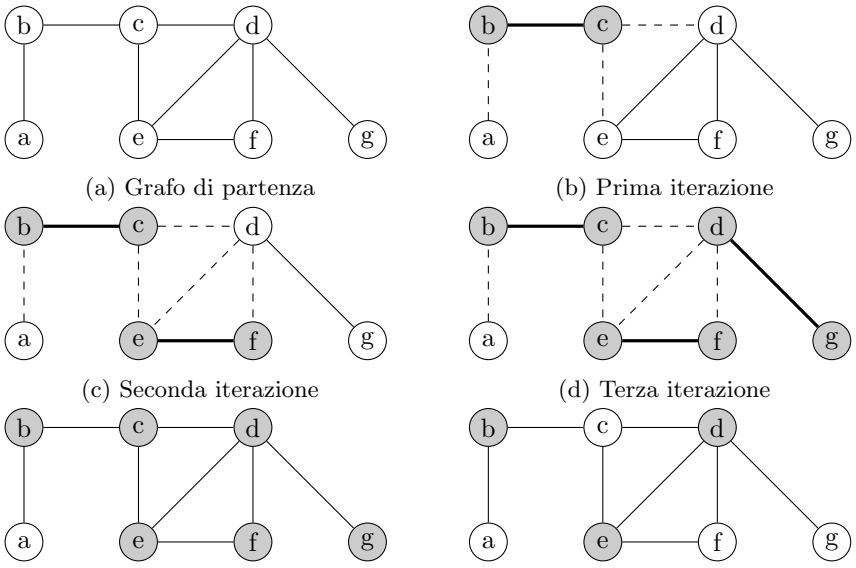
$$|V^*| \geq \frac{|V'|}{2} \Rightarrow \frac{|V'|}{|V^*|} \leq 2 \Leftrightarrow \frac{c(A_\pi(i))}{c(s^*(i))} \leq 2 = \rho(n)\,,$$

ottenendo quindi che l'algoritmo APPROX_VC è di 2-approssimazione. □

Esempio. *Applichiamo l'algoritmo di 2-approssimazione per il problema VERTEX-COVER al grafo in figura 1.1*

- *come scelta greedy nella prima iterazione scegliamo il lato* $\{b, c\}$*, quindi* $V' = \{b, c\}$ *e possiamo eliminare dal grafo i lati* $\{a, b\}, \{c, d\}, \{c, e\}$*;*
- *alla seconda iterazione scegliamo come scelta greedy il lato* $\{e, f\}$*, quindi* $V' = \{b, c, e, f\}$ *e possiamo eliminare i lati* $\{e, d\}, \{d, f\}$*;*
- *alla terza (ed ultima) iterazione scegliamo l'unico lato rimasto,* $\{d, g\}$*, quindi* $V' = \{b, c, d, e, f, g\}$ *e l'algoritmo termina non essendo disponibili ulteriori lati.*

L'esempio restituisce un vertex cover di cardinalità 6*, tuttavia, data la semplicità del grafo, è facilmente verificabile che il vertex cover ottimo è* $V^* = \{b, d, e\}$*, di cardinalità* 3*: l'algoritmo di approssimazione, con le scelte effettuate, ha fornito il risultato approssimato peggiore.*



(a) Grafo di partenza (b) Prima iterazione

(c) Seconda iterazione (d) Terza iterazione

(e) Il vertex cover approssimato (f) Il vertex cover ottimo

Figura 1.1: Esempio grafico di applicazione dell'APPROX_VC

1.2.2 La riduzione polinomiale non preserva l'approssimazione

In questo paragrafo forniremo un esempio di come un algoritmo di approssimazione per un problema NPH non si trasforma in un algoritmo di approssimazione per un altro problema NPH con la stessa qualità (ovvero lo stesso fattore di approssimazione).

Definiamo il problema (decisionale) CLIQUE come segue:

$$\begin{cases} \langle G = (V, E), k \rangle, \text{G grafo non diretto}, |V| = n, |E| = m, k \in \mathbb{N}^+ \\ \\ \text{Esiste in G una clique di taglia k?} \end{cases}$$

Sappiamo che vale la seguente equivalenza:

$$\langle G, k \rangle \in \text{CLIQUE} \Leftrightarrow \langle G^c, |V| - k \rangle \in \text{VERTEX-COVER}$$

dove $G^c = (V, E^c)$, $E^c = \{\{u, v\} \mid \forall u, v \in |V|, u \neq v, \{u, v\} \notin E\}$, è il grafo complementare di G.

Per risolvere il problema di ottimizzazione CLIQUE sfrutto la riduzione polinomiale $CLIQUE <_p VERTEX-COVER$: si vuole trovare la clique di taglia massima di G supponendo che tale clique esista e sia $|V^*| = k \geq n/2$. Come prima cosa trasformo l'istanza di CLIQUE in una istanza di VERTEX-COVER secondo la funzione di riduzione vista in precedenza. Successivamente, si applica l'algoritmo di approssimazione per VERTEX-COVER all'istanza trasformata, ottenendo il VERTEX-COVER minimo (approssimato) V', dal quale si può ricavare la clique (massima?) $V - V'$.

Avendo supposto che l'istanza di CLIQUE sia una istanza positiva la cui clique abbia taglia almeno pari alla metà della cardinalità di V, com'è la qualità

dell'algoritmo esposto sopra? Se $|V^*| = k$ in G, allora $|V^*_{VC}| = n - k$ in G^c. L'algoritmo APPROX_VC restituisce il vertex cover di taglia $|V'| \leq 2|V^*_{VC}| = 2(n-k)$, per cui la taglia della clique sarà $|V - V'| \geq n - 2(n-k) = 2k - n$. Ora, supponendo che $|V^*| = k = n/2 + 1$, si ottiene che $|V - V'| = 2k - n = 2(n/2 + 1) - n = 2$, quindi:

$$\frac{c(s^*(\langle G, k\rangle))}{c(A_{VC}(\langle G^c, |V| - k\rangle))} = \frac{\frac{n}{2} + 1}{2} \approx \frac{n}{4} \leq \rho(n).$$

L'esempio mostra come l'algoritmo di 2-approssimazione per VERTEX-COVER porti ad un algoritmo per CLIQUE di $\frac{n}{4}$-approssimazione.

1.3 Il problema TSP

Dato $G = (V, E)$, grafo non diretto, si chiama *circuito hamiltoniano* un ciclo semplice[1] che tocca tutti i vertici del grafo. Il problema di HAMILTON è definito come segue:

$$\begin{cases} \langle G = (V, E)\rangle \\ \text{Esiste in G un ciclo hamiltoniano?} \end{cases}$$

Il problema di HAMILTON ci permette di definire il problema del *Traveling Salesman Problem (TSP)*:

$$\begin{cases} \langle G_c = (V, E), c, k\rangle, G_c \text{ grafo completo su } V, \\ c : V \times V \to \mathbb{N} - \{0\} \text{ simmetrica}, k \in \mathbb{N} - \{0\} \\ \\ \text{Esiste in } G_c \text{ un ciclo hamiltoniano la cui somma dei costi} \\ \text{sugli archi del circuito sia minore o uguale a } k? \end{cases}$$

Definizione. *Un circuito hamiltoniano in un grafo completo è chiamato* ***tour****.*

Teorema. *Il problema TSP è un problema NP-completo.*

Dimostrazione. Utilizzo la riduzione $HAMILTON <_p TSP$.

L'istanza $\langle G = (V, E)\rangle$ di HAMILTON è un grafo generico: la funzione di riduzione polinomiale trasformerà tale istanza in $\langle G_c = (V, E'), c, k\rangle$, dove G_c sarà il grafo completo di G, $E' = \{\{u, v\} \mid \forall u, v \in V, u \neq v\}$, $k = |V|$ e la funzione di costo sarà

$$c(u, v) = \begin{cases} 1 & \text{se } \{u, v\} \in E \\ 2 & \text{altrimenti} \end{cases}$$

Sia $\langle G = (V, E)\rangle \in$ HAMILTON: allora il tour esistente in G esiste anche in G_c essendo semplicemente il grafo completo ottenuto da G aggiungendo i lati mancanti (ovviamente nessuno di loro farà parte di quel tour). Per la funzione di riduzione, abbiamo che il costo del tour sarà pari a $1 \cdot |V|$, essendo il tour

[1] Un ciclo semplice è un ciclo che non passa per lo stesso vertice più di una volta (eccetto che per il primo e l'ultimo vertice del ciclo).

di lunghezza $|V|$ e formato completamente da lati in E: la somma dei costi è uguale a k, pertanto soddisfa il problema TSP.

Di converso, si supponga che $\langle G = (V, E)\rangle \notin$ HAMILTON: allora, un qualsiasi tour in G_c potrà essere formato da lati di G e necessariamente da almeno un lato non in G. Supponiamo che il tour in G_c abbia $|V| - 1$ lati di E e un lato di $E' - E$: allora la somma dei costi sugli archi vale $1 \cdot (|V| - 1) + 2 \cdot 1 = |V| + 1 > k$, non soddisfacendo TSP. □

Definiamo il problema di ottimizzazione per TSP come segue:

$$\begin{cases} \langle G_c = (V, E), c\rangle, G_c \text{ grafo completo su } V, \\ c : |V| \times |V| \to \mathbb{N} - \{0\} \text{ simmetrica} \\ \\ \text{Esiste in } G_c \text{ un ciclo hamiltoniano la cui somma dei costi è minima?} \end{cases}$$

Teorema. *Se P≠NP, non può esistere alcun algoritmo di approssimazione per TSP con $\rho(n)$ calcolabile in tempo polinomiale, con $n = |V|$.*

Dimostrazione. Dimostriamo che se esistesse un algoritmo $A_{TSP}^{\rho(n)}$ di $\rho(n)$-approssimazione per TSP allora potrei decidere HAMILTON in tempo polinomiale.

Sia $\langle G = (V, E)\rangle$ l'istanza per HAMILTON. Costruisco l'istanza per TSP similmente a quanto fatto nella dimostrazione precedente: $\langle G_c = (V, E_c), c\rangle$ con

$$c(u, v) = \begin{cases} 1 & \text{se } \{u, v\} \in E \\ |V|\rho(|V|) + 1 & \text{se } \{u, v\} \notin E. \end{cases}$$

La taglia dell'istanza è polinomiale essendo polinomiale $\rho(|V|)$.

Applico quindi $A_{TSP}^{\rho(n)}$ all'istanza $\langle G_c = (V, E_c), c\rangle$ ottenendo un tour T di costo c_T.

1. $\langle G = (V, E)\rangle \in$ HAMILTON $\Rightarrow \exists T^*$ in G_c di costo $|V|$. Allora $A_{TSP}^{\rho(n)}$ restituisce un tour T di costo $c_T \leq |V|\rho(|V|)$: questo è vero perché l'approssimazione garantisce che il costo della soluzione approssimata non sia maggiore di un fattore $\rho(|V|)$ della soluzione ottima;

2. $\langle G = (V, E)\rangle \notin$ HAMILTON $\Rightarrow$ un qualsiasi tour T in G_c deve contenere almeno un lato non in E di costo $|V|\rho(|V|) + 1$. Allora, per T^* si ha che $c_{T^*} \geq |V| - 1 + |V|\rho(|V|) + 1 = |V| + |V|\rho(|V|) > |V|\rho(|V|)$, ovvero $A_{TSP}^{\rho(n)}$ ritorna una soluzione di costo maggiore di $|V|\rho(|V|)$.

In 2 è riportato l'algoritmo che permette(rebbe) di decidere HAMILTON sfruttando $A_{TSP}^{\rho(n)}$.

Ovviamente, avendo ipotizzato che P≠NP, tutto ciò è assurdo, concludendo quindi la dimostrazione. □

1.4 Il problema TRIANGLE-TSP

Per quanto abbiamo visto, TSP è un problema NP-completo che non può essere approssimato. Nonostante ciò, è possibile considerare una sottoclasse di TSP,

Algorithm 2 Algoritmo polinomiale per HAMILTON

```
function DECIDE_HAMILTON(⟨G = (V, E)⟩)
    ** creo l'istanza ⟨G_c = (V, E_c), c⟩ **
    T ← A_TSP^ρ(n)(⟨G_c = (V, E_c), c⟩)
    if cost(T) ≤ |V|ρ(|V|) then
        return 1
    else
        return 0
    end if
end function
```

chiamata TRIANGLE-TSP, che considera il caso in cui la funzione di costo soddisfa la disuguaglianza triangolare, ovvero

$$c(u, w) \leq c(u, v) + c(v, w) \quad \forall u, w \in V, u \neq w \neq v.$$

Definiamo inoltre la funzione di costo su un sottoinsieme di archi come segue:

$$c(A) = \sum_{\{u,v\} \in A} c(u, v) \quad \forall A \subseteq E.$$

Tale problema rimane un problema NP-completo[2]. È possibile formulare due algoritmi di approssimazione che risolvono TRIANGLE-TSP.

1.4.1 Un algoritmo di 2-approssimazione

Un possibile algoritmo che risolve TRIANGLE-TSP calcola il MINIMUM SPANNING TREE (MSP)[3] del grafo completo $G = (V, E)$ e in seguito procede con una visita in pre-ordine di tale albero: la lista dei vertici visitati restituisce un tour. Il motivo per cui si calcola un albero ricoprente di costo minimo è che tale costo rappresenta un limite inferiore alla lunghezza di un tour ottimo per TSP. L'algoritmo 3 descrive lo pseudocodice dell'algoritmo in questione.

Algorithm 3 Algoritmo di 2-approssimazione per TRIANGLE-TSP

```
function APPROX_TRIANGLE-TSP_TOUR(⟨G = (V, E), c⟩)
    r ← ** scegli un nodo in V come radice **
    T ←MINIMUM_SPANNING_TREE(⟨G = (V, E), c, r⟩)
    H ←PREORDER_VISIT(T)
    return H
end function
```

Teorema. *APPROX_TRIANGLE-TSP_TOUR è un algoritmo in tempo polinomiale di 2-approssimazione per TRIANGLE-TSP.*

[2]La dimostrazione è identica a quella per dimostrare che TSP ∈ NPC, è sufficiente notare che la funzione di costo utilizzata nella riduzione polinomiale soddisfa la disuguaglianza triangolare.

[3]Un albero ricoprente di un grafo è un albero che contiene tutti i vertici del grafo e un sottoinsieme degli archi, costituito da tutti e soli gli archi che connettono i vertici con uno e un solo cammino.

Dimostrazione. È chiaro che questo algoritmo esegue in tempo polinomiale: MINIMUM_SPANNING_TREE($\cdot$) è un algoritmo quadratico nel numero di nodi del grafo mentre PREORDER_VISIT($\cdot$) esegue in tempo lineare.

Sia H^* il tour ottimo per il grafo completo $G = (V, E)$ dato. Eliminando un qualsiasi arco dal tour otteniamo un albero ricoprente T, per cui

$$c(T) \leq c(H^*).$$

Una visita completa di T lista i vertici quando vengono visitati per la prima volta e quando vengono rivisitati tornado dalla visita di un loro sotto-albero: sia quindi W tale lista. La visita completa attraversa ogni arco di T esattamente due volte (una per "scendere" nel sotto-albero e una per "risalire" il sotto-albero verso il nodo genitore e proseguire con la visita):

$$c(W) = 2c(T) \leq 2c(H^*).$$

Si noti bene che W non rappresenta un tour in quanto alcuni vertici vengono visitati più di una sola volta.

Poiché il grafo soddisfa la disuguaglianza triangolare, è possibile eliminare dalla lista W un qualsiasi vertice senza che il costo di W incrementi: eseguiamo tale operazioni con tutti vertici che sono in lista come successivi alla loro prima visita. Sia quindi H il ciclo associato a questa "pulitura" di W: questo rappresenta un tour poiché ogni vertice viene visitato esattamente una volta ed è ciò che viene calcolato dall'algoritmo proposto. Poiché H è calcolato eliminando vertici da W, si ha

$$c(H) \leq c(W) \leq 2c(H^*) \Rightarrow \frac{c(H)}{c(H^*)} \leq 2 = \rho(n).$$

□

Esempio. *In figura 1.2 un esempio di applicazione dell'algoritmo di approssimazione per TRIANGLE-TSP. In (a) il grafo non diretto completo: i vertici sono stati posizionati sulle intersezioni della griglia così da rendere la funzione di costo la distanza euclidea, la quale soddisfa la diseguaglianza triangolare. In (b) il MST di radice* a. *In (c) la visita completa in pre-ordine: la sequenza dei vertici visitati è* $W = \langle a, b, c, b, h, a, d, e, f, e, g, e, d, a\rangle$. *Con un puntino sono indicati i vertici visitati per la prima volta e che sopravvivono la "pulitura" di* W. *In (d) il tour ottenuto dalla visita completa, ovvero il risultato dell'algoritmo di approssimazione (costo totale di circa 19,074). In (e) una soluzione ottima per il grafo originale (costo totale di circa 14,715).*

1.4.2 L'algoritmo di Christofides

Per introdurre l'algoritmo di Christofides dobbiamo prima introdurre il concetto di multigrafo.

Definizione. *Un multigrafo è un grafo il quale, ad ogni arco, è associata una molteplicità:*

$$\mathcal{G} = (\mathcal{V}, \mathcal{E})$$

$$\forall e \in \mathcal{E} \quad m(e) \geq 1$$

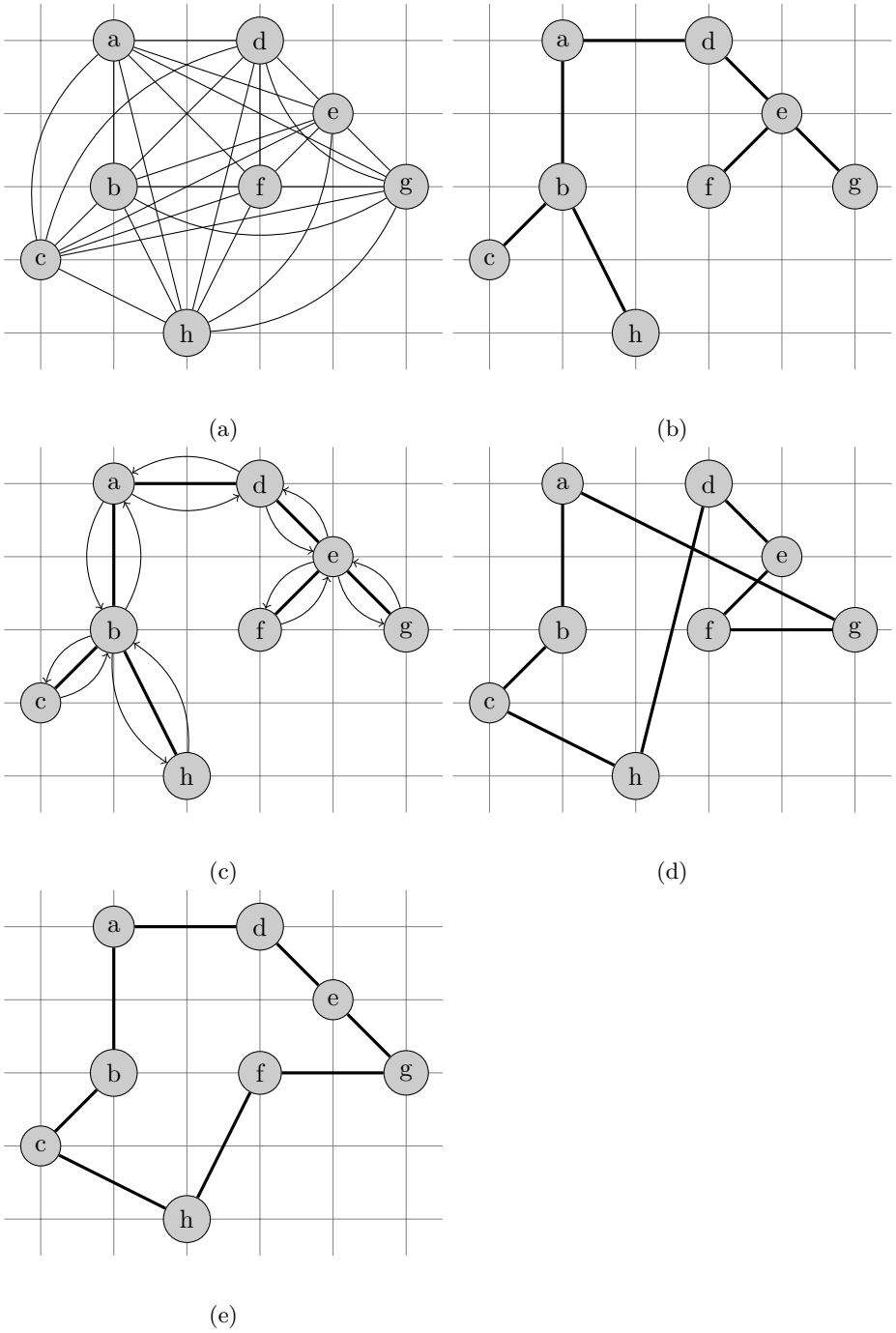


Figura 1.2: Esempio di applicazione dell'algoritmo di approssimazione per TSP

Definizione. *Dato un multigrafo $\mathcal{G}$ non diretto, esso è* ***euleriano*** *se esiste un ciclo euleriano, ovvero esiste un ciclo non semplice che attraversa ogni copia di ogni arco una sola volta.*

Teorema. *Un multigrafo è euleriano se e solo se è connesso ed ogni suo vertice ha grado pari.*

Un ciclo euleriano (*euler tour*), può essere determinato in tempo lineare $\Theta(|\mathcal{V}| + |\mathcal{E}|)$.

L'obiettivo è quello di determinare un sottografo $\mathcal{G}'$ di $\mathcal{G}$ (possibilmente con archi replicati) che contenga tutti i nodi di $\mathcal{V}$ e che sia euleriano, con $\sum_{e \in \mathcal{E}_{\mathcal{G}'}} \approx c(C^*)$. In pratica è lo stesso concetto dell'albero di copertura minimo poiché tale albero può essere visto come un multi-grafo dove ogni arco dell'albero vale per due per "contare" i due attraversamenti della visita completa in pre-ordine.

Christofides parte anch'esso dell'albero di copertura minimo (sia T^*), per il quale sappiamo che $c(T) \leq c(C^*)$, con C^* il ciclo hamiltoniano ottimo. All'MST aggiungo il minimo numero di lati in modo da far diventare l'MST euleriano (attenzione: devono costare poco, altrimenti rischio di trovare una soluzione di molto peggiore rispetto l'ottimo). È possibile determinare un sottoinsieme $\mathcal{M}$ di archi tale per cui $\bar{\mathcal{G}}(\mathcal{V}, \mathcal{E}_{T^*} \cup \mathcal{M})$ è euleriano e $c(\mathcal{E}_{T^*} \cup \mathcal{M}) \leq \frac{3}{2} c(C^*)$. Non rimane quindi che trovare $\mathcal{M}$.

Proposizione. *Dato un multi-grafo non orientato, sia $\mathcal{V}_{ODD}$ l'insieme dei nodi di grado dispari*[4]*. Allora, $|\mathcal{V}_{ODD}|$ è pari.*

Dimostrazione. È noto che

$$\sum_{v \in \mathcal{V}} deg(v) = 2|\mathcal{E}|.$$

La sommatoria può essere spezzata considerando da una parte i nodi di grado pari e dell'altra i nodi di grado dispari:

$$\sum_{v \in \mathcal{V}} deg(v) = \sum_{v \in \mathcal{V}_{ODD}} deg(v) + \sum_{v \in \mathcal{V}_{EVEN}} deg(v) = 2|\mathcal{E}|.$$

Ora, la sommatoria dei gradi dei nodi di grado pari è pari e sapendo che la somma complessiva deve essere pari, si deduce che anche la somma dei gradi dei nodi di grado dispari debba essere pari. Per l'appunto, poiché la sommatoria dei gradi dei nodi di grado dispari deve sommare ad un numero pari, significa che il numero di termini che vengono sommati (e che sono dispari) è pari, da cui la tesi. □

Dalla proposizione precedente sappiamo che il numero di nodi di grado dispari è pari e sappiamo inoltre che per rendere l'MST euleriano è necessario che tutti i nodi siano di grado pari: l'insieme dei lati che cerchiamo forma un **matching**.

Definizione. *Dato un multi-grafo $\mathcal{G} = (\mathcal{V}, \mathcal{E})$, un sottoinsieme $\mathcal{M} \subseteq \mathcal{E}$ forma un* ***matching*** *se $\forall e_1, e_2 \in \mathcal{M}$ si ha che $e_1 \cap e_2 = \emptyset$.*

[4] Il grado di un nodo è il numero di lati incidenti in quel nodo è ed definito come $deg(v)$, $v \in \mathcal{V}$.

Definizione. *Un matching M è **perfetto** se $\forall v \in \mathcal{V}\ \exists e \in \mathcal{M} \mid v \in e$.*

Dalla definizione segue che una condizione necessaria affinché il matching sia perfetto è che $|\mathcal{V}|$ sia pari.

L'algoritmo procede come segue:

- si considera il sottografo di $\mathcal{G}$ (l'istanza di TRIANGLE-TSP) indotto da $\mathcal{V}^{\mathcal{T}^*}_{\mathcal{ODD}}$:
$$\mathcal{G}' = (\mathcal{V}^{\mathcal{T}^*}_{\mathcal{ODD}}, \mathcal{E}')$$
$$\mathcal{E}' : \forall u, v \in \mathcal{V}^{\mathcal{T}^*}_{\mathcal{ODD}},\ u \neq v,\ \exists\{u, v\} \in \mathcal{E} \Rightarrow \{u, v\} \in \mathcal{E}';$$
- si determina un matching perfetto $\mathcal{M}^*$ di costo minimo in $\mathcal{G}'$: per fare ciò è possibile utilizzare l'algoritmo ungherese che restituisce la risposta in tempo $\Theta(|\mathcal{V}^{\mathcal{T}^*}_{\mathcal{ODD}}|^3) = O(n^3)$;
- si considera il sottografo $\bar{\mathcal{G}} = (\mathcal{V}, \mathcal{E}_{T^*} \cup \mathcal{M}^*)$: poiché ora tutti i nodi hanno grado pari allora il grafo è euleriano;
- si determina un ciclo euleriano;
- si determina il tour C tramite *short-cutting* (si ricorda che vale la disuguaglianza triangolare tra i costi dei lati).

La procedura restituisce un tour C che soddisfa la relazione

$$c(C) \leq c(\mathcal{E}_{\mathcal{T}^*}) + c(\mathcal{M}^*) \leq c(C^*) + c(\mathcal{M}^*);$$

è quindi sufficiente dimostrare che $c(\mathcal{M}^*) \leq c(\mathcal{T}^*)/2$.

Sia $C^* = \langle v_1, v_2, v_3, \cdots, v_{|\mathcal{V}|}, v_1\rangle$ il tour hamiltoniano di costo ottimo. Tramite *short-cutting* elimino da C^* tutti i vertici che hanno grado pari in T^*: si ottiene così un ciclo $O^* = \langle v_1^{ODD}, v_2^{ODD}, \cdots, v^{ODD}_{|\mathcal{V}^{\mathcal{T}^*}_{\mathcal{ODD}}|}, v_1^{ODD}\rangle$ tale per cui $c(O^*) \leq c(C^*)$. Ora, poiché i vertici di grado dispari sono in numero pari, è possibile identificare un matching perfetto. In particolare, se coloro di bianco e di nero in maniera alternata i lati che compongono il ciclo O^*, ottengo due matching perfetti, rispettivamente $\mathcal{M}^*_{\mathcal{B}}$ e $\mathcal{M}^*_{\mathcal{N}}$, e vale $c(\mathcal{M}^*_{\mathcal{B}}) + c(\mathcal{M}^*_{\mathcal{N}}) = c(O^*) \leq c(C^*)$. Dall'ultima relazione segue che

$$\min\{c(\mathcal{M}^*_{\mathcal{B}}), c(\mathcal{M}^*_{\mathcal{N}})\} \leq \frac{1}{2}c(C^*),$$

ovvero esiste un matching perfetto tra i nodi di $\mathcal{V}^{\mathcal{T}^*}_{\mathcal{ODD}}$ di costo al più uguale a $c(C^*)/2$.

Si noti che il matching perfetto potrebbe riutilizzare un lato di $\mathcal{E}_{\mathcal{T}^*}$: ciò è lecito in quanto si sta lavorando su multi-grafi.

Per il problema TRIANGLE-TSP la miglior approssimazione che si è riusciti ad ottenere è proprio l'approssimazione 3/2 di Christofides. Un noto risultato riguardante questo problema è il seguente: se P≠NP, un algoritmo di $\rho(n)$-approssimazione per TRIANGLE-TSP è tale per cui $\rho(n) \geq 220/219 \approx 1,0045$.

Un sottocaso importante di TRIANGLE-TSP è EUCLIDEAN-TSP, dove l'insieme dei vertici rappresenta dei punti del piano mentre i costi rappresentano distanze euclidee (le quali soddisfano la disuguaglianza triangolare). Tale problema ammette un PTAS con $\rho(n) \leq (1 + \epsilon)\ \forall\epsilon$ e tempo $T = \Theta(n^{1/\epsilon})$. Non esiste invece un FPTAS.

1.5 Il problema SET-COVER

Nel problema SET-COVER, l'istanza è una coppia di insiemi $(X, \mathcal{F})$, dove X è l'insieme universo, di cardinalità finita, e rappresenta un insieme di oggetti, mentre $\mathcal{F}$ è il booleano di X, ovvero

$$\mathcal{F} \subseteq \{S : S \subseteq X\} = B(X) \ : \ \forall x \in X \ \exists S \in \mathcal{F} \mid x \in S;$$

da ciò segue che $X = \cup_{S \in \mathcal{F}} S$, detta **proprietà di copertura**. L'obiettivo del problema SET-COVER è quello di determinare un insieme di copertura di X, ovvero, dato l'insieme di copertura $\mathcal{C} \in \mathcal{F}$, deve valere $X = \cup_{S \in \mathcal{C}} S$: in particolare, l'insieme di copertura ottimo $\mathcal{C}^*$ è tale da essere quello di cardinalità minima. È banale affermare che $|\mathcal{C}^*| \leq |\mathcal{F}|$.

La versione decisionale del problema è la seguente:

$$\begin{cases} \langle X, \mathcal{F}, k \rangle, k \in \mathbb{N} \\ \\ \exists \mathcal{C} \in \mathcal{F} \text{ che copre } X \ : \ |\mathcal{C}| \leq k? \end{cases}$$

Proposizione. *Il problema SET-COVER è NP-completo.*

Dimostrazione. È possibile dimostrare che il problema è NP-hard tramite la riduzione polinomiale VERTEX-COVER $<_p$ SET-COVER come segue:

$$f(\langle G = (V, E), k \rangle) = \langle X, \mathcal{F}_G, k_G \rangle$$

dove $X = E$, $\mathcal{F}_G = \{S_{v_1}, S_{v_2}, \cdots, S_{v_{|V|}}\}$ con $S_{v_i} = \{e \in E \mid v_i \in E\}$, $k_G = k$.

Se $\langle G = (V, E), k \rangle \in$ VERTEX-COVER significa che $\exists V' \subseteq V \ : \ |V'| = k$ tale da coprire ogni lato del grafo. Allora, per ogni nodo $v_i \in V'$, $i = 1, 2, \cdots, k$, esiste un corrispondente insieme $S_{v_i} = \{e \in E \mid v_i \in E\}$. Tali insiemi formano un set cover di cardinalità k poiché, essendo ogni v_i facente parte del vertex cover, significa che i lati selezionati dai vari S_{v_i} coprono ogni lato del grafo: se esistesse un lato $e = \{w, z\} \in E$ tale da non venir selezionato da nessun S_{v_i} allora $w, z \neq v_i \ \forall \ i = 1, 2, \cdots, k$, contraddicendo il fatto che V' sia un vertex cover.Segue quindi che $f(\langle G = (V, E), k \rangle) \in$ SET-COVER.

Di converso, se $f(\langle G = (V, E), k \rangle) \in$ SET-COVER significa che $\exists \mathcal{C} = \{S_{v_1}, S_{v_2}, \cdots, S_{v_{k_G}}\}$ il quale copre l'intero insieme di lati X_G. Quindi, per ogni lato $\{w, z\} \in X_G = X$ esiste un insieme $S_{v_i} \subseteq \mathcal{C}$ tale che $(v_i = w) \vee (v_i = z)$. Segue che l'insieme $\{v_1, v_2, \cdots, v_{k_G}\}$ forma un vertex cover di cardinalità $k = k_G$ di G e quindi $\langle G = (V, E), k \rangle \in$ VERTEX-COVER. □

Un algoritmo di approssimazione per SET-COVER è possibile trovarlo in 4.

Analisi di correttezza Quando l'insieme U è vuoto, significa che tutti gli elementi sono stati coperti. Poiché $\forall x \in X \ \exists S \ : \ x \in S$, allora prima o poi S verrà selezionato in quanto la cardinalità degli insiemi della funzione arg max è almeno pari a 1, pertanto l'insieme U decresce monotonicamente fino a ridursi all'insieme vuoto.

Algorithm 4 Algoritmo di approssimazione per SET-COVER

function APPROX_SET_COVER($\langle X, \mathcal{F} \rangle$)
  $U \leftarrow X$
  $\mathcal{C} \leftarrow \emptyset$
  while $U \neq \emptyset$ **do**
    $S \leftarrow \text{argmax}\{|T \cap U| : T \in \mathcal{F}\}$
    $U \leftarrow U - S$
    $\mathcal{C} \leftarrow \mathcal{C} \cup \{S\}$
  end while
  return C
end function

Analisi di complessità Poiché $|S \cap U| \geq 1$, il numero di iterazioni al caso peggiore è $O(|X|)$. Analogamente, poiché ogni sottoinsieme di $\mathcal{F}$ può essere selezionato al più una volta, un altro limite superiore al numero delle iterazioni è $O(|\mathcal{F}|)$. Segue che il numero di iterazioni è $O(\min\{|X|, |\mathcal{F}|\})$.

Come implemento la funzione argmax$(\cdot)$? Nella maniera banale, posso scandire gli elementi di $\mathcal{F}$ e, per ognuno di essi, i propri elementi: segue che il tempo necessario per tale funzione è $O(|\mathcal{F}| \cdot |X|)$, quindi $T = O(\min\{|X|, |\mathcal{F}|\} \cdot |X| \cdot |\mathcal{F}|)$ che, al caso peggiore, può essere cubica nella taglia dell'istanza ($\Theta(|\langle X, \mathcal{F} \rangle|^3)$). Tramite liste concatenate e un array di supporto è possibile implementare questa funzione in tempo $T = O(\sum_{S \in \mathcal{F}} |S|) = O(|\langle X, \mathcal{F} \rangle|)$.

Fattore di approssimazione Sia U_t l'insieme degli elementi scoperti all'**inizio** dell'iterazione $t \geq 1$ ($U_1 = X$). Sia $U_{\bar{t}} = \emptyset$. Si vuole determinare un limite superiore a t, $\bar{t}$: $\bar{t} - 1$ iterazioni sono sufficienti a coprire X (inoltre, se $\bar{t}$ è il minimo, $|\mathcal{C}| \leq \bar{t} - 1$).

Supponiamo che il costo della soluzione ottima (quindi non approssimata) sia $k = |\mathcal{C}^*|$. Considero l'iterazione t: all'inizio di tale iterazione abbiamo che $U_t \subseteq X$ e $\exists S_1, S_2, \cdots, S_k \in \mathcal{F} \,|\, U_t \subseteq \cup_{i=1}^{k} S_i$.

Tecnica del *pigeonhole*: deve esistere un insieme che contiene almeno il numero medio di elementi di U_t, ovvero

$$\exists \bar{i} \mid |U_t \cap S_{\bar{i}}| \geq \frac{|U_t|}{k}.$$

Prova:

$$U_t = U_t \cap (\cup_{i=1}^{k} S_i) = \cup_{i=1}^{k} (U_t \cap S_i)$$

$$|U_t| = |\cup_{i=1}^{k} (U_t \cap S_i)| \leq \sum_{i=1}^{k} |U_t \cap S_i|$$

Ora, se supponessimo che $\forall i : |U_t \cap S_i| < |U_t|/k$, otterremmo il seguente assurdo:

$$|U_t| \leq \sum_{i=1}^{k} |U_t \cap S_i| < \sum_{i=1}^{k} \frac{|U_t|}{k} = |U_t|.$$

Per cui, alla fine dell'iterazione t, seleziono necessariamente un insieme di almeno $|U_t|/k$ elementi, coprendo così almeno $|U_t|/k$ elementi di U_t, pertanto all'inizio della seguente iterazione avremo che $|U_{t+1}| \leq |U_t| - |U_t|/k$. Tale

disequazione forma la ricorrenza

$$|U_{t+1}| \leq |U_t| \left(1 - \frac{1}{k}\right)$$

che per *unfolding* diviene (sia $n = |X|$)

$$|U_{t+1}| \leq |U_1| \left(1 - \frac{1}{k}\right)^t = n \left(1 - \frac{1}{k}\right)^t$$

e poiché è sempre vero che $(1 - x) \leq e^{-x}$ (dove l'uguaglianza vale solo per $x = 0$), ottengo

$$|U_{t+1}| < ne^{-t/k}.$$

Scegliendo $\bar{t} = k \ln n$ si ottiene che

$$|U_{\bar{t}+1}| < ne^{-\frac{k}{k} \ln n} = 1 \Rightarrow |U_{\bar{t}+1}| = 0 \Rightarrow |\mathcal{C}| \leq \bar{t} = k \ln n = |\mathcal{C}^*| \ln n$$

da cui segue

$$\rho(n) = \frac{|C|}{|C^*|} \leq \ln n.$$

Si noti l'uso improprio della taglia di X piuttosto che della taglia dell'istanza.

Fattore di approssimazione: analisi più fine È possibile effettuare una analisi più fine del fattore di approssimazione ricorrendo ai **numeri armonici**.

Definizione. *Per ogni numero naturale, il k-esimo numero armonico è definito come segue:*

$$H_k = \sum_{i=1}^{k} \frac{1}{i}$$

con $H_0 = 0$.

Proposizione. *Per il k-esimo numero armonico valgono le seguenti diseguaglianze:*

$$\ln(k+1) = \int_1^{k+1} \frac{1}{x} \, \mathrm{d}x \leq H_k < 1 + \int_1^k \frac{1}{x} \, \mathrm{d}x = 1 + \ln k$$

Premesso ciò, è possibile dimostrare il seguente risultato.

Proposizione. *Per il problema SET-COVER, il fattore di approssimazione soddisfa la seguente diseguaglianza:*

$$\rho(\langle X, \mathcal{F} \rangle) \leq H_{\max\{|S| \,:\, S \in \mathcal{F}\}}$$

Dimostrazione. Innanzitutto, cominciamo con l'effettuare la *pesatura* degli elementi di X: ad ogni elemento $x \in X$ associo un peso c_x che dipende dalle condizioni sotto le quali x viene coperto durante l'algoritmo. Ciò che si vuole ottenere con la pesatura è quello di catturare il tasso di decrescita dell'insieme U, quindi ad x associamo un peso piccolo se viene catturato assieme a molti altri suoi colleghi (ovvero "in un colpo solo" copro tanti elementi di X), mentre

Capitolo 3

Algoritmi randomizzati

3.1 Introduzione

Nel capitolo precedente abbiamo visto un esempio di algoritmo randomizzato con Miller Rabin. Ora, introduciamo gli algoritmi randomizzati in maniera generale, e in seguito forniremo alcuni esempi di algoritmi di questo tipo.

Gli algoritmi randomizzati sono algoritmi che utilizzano la randomizzazione **al loro interno**. L'analisi viene fatta ad **input fissato** e tutte le quantità notevoli in un algoritmo randomizzato sono variabili aleatorie: la complessità computazionale $T_A(n)$ e la correttezza (v.a. binaria, vale 1 se l'algoritmo è corretto, 0 altrimenti).

Dato il solito problema computazionale $\Pi \subseteq \mathcal{I} \times \mathcal{S}$, possiamo evidenziare due classi di algoritmi randomizzati:

1. **algoritmi Las Vegas**: sono algoritmi che non sbagliano mai, cioè vale

   $$P(A(i)\Pi i) = 1;$$

   un esempio di algoritmo Las Vegas è il quicksort randomizzato.

   Gli algoritmi Las Vegas sfruttano la randomizzazione per migliorare la complessità $T_A(n)$, sia in media che nell'analisi in alta probabilità, fornendo una limitazione superiore alla $P(T_A(n) \geq f(n))$.

2. **algoritmi Montecarlo**: sono algoritmi che possono sbagliare; un esempio di algoritmo Montecarlo lo abbiamo visto nel precedente capitolo con l'algoritmo di Miller Rabin. Negli algoritmi Montecarlo sia la complessità computazionale che la correttezza sono variabili aleatorie che vengono studiate in media e in alta probabilità. In particolare, per la correttezza si cerca di fornire una limitazione superiore a $P(A(i) \not\Pi i)$.

3.2 Analisi in alta probabilità

Nella sezione precedente abbiamo menzionato l'analisi in alta probabilità. Di seguito ne forniamo una definizione e introduciamo gli strumenti matematici necessari.

Definizione. *Dato un problema computazionale* $\Pi \subseteq \mathcal{I} \times \mathcal{S}$*, un algoritmo* A_Π *ha complessità* $T(n) = O(f(n))$ *con alta probabilità*[1] *se esistono delle costanti* $c, d > 0,\ n_0 \mid n > n_0\ \forall n,\ \forall i\ :\ |i| = n$ *tali per cui*

$$P(A_\Pi(i) \text{ termina in tempo } \geq cf(n)) \leq \frac{1}{n^d}.$$

Lemma (Lemma di Markov)**.** *Sia* T *una v.a. intera non negativa, con* $P(T > b) = 0$ *per un qualche* $b \in \mathbb{Z}^+$*. Allora per ogni* $1 \leq t \leq b$ *si ha*

$$t \cdot P(T \geq t) \leq E[T] \leq t + (b - t)P(T \geq t).$$

Proposizione. *Nelle ipotesi dell'analisi in alta probabilità, se l'algoritmo non può mai esibire esecuzioni di tempo pari o maggiori a* $n^a,\ a \leq d$*, allora*

$$E[T_A(n)] = O(f(n)).$$

Dimostrazione. Per dimostrare questa proposizione sfruttiamo il lemma di Markov. Sia $b = n^a$ e applichiamo Markov per $t = cf(n)$:

$$\begin{aligned} E[T_A(n)] &\leq cf(n) + \frac{n^a - cf(n)}{n^d} \\ &\leq cf(n) + 1 \\ &= O(f(n)) \end{aligned}$$

essendo il numeratore dello stesso ordine di n^a e tenendo conto che per ipotesi $a \leq d$. □

Dal lemma di Markov è possibile estrapolare la seguente limitazione superiore:

$$P(T_A(n) \geq t) \leq \frac{E[T_A(n)]}{t};$$

tale limite superiore tuttavia è estremamente debole. Sia $t = kE[T_A(n)]$, allora

$$P(T_A(n) \geq kE[T_A(n)]) \leq \frac{E[T_A(n)]}{kE[T_A(n)]} = \frac{1}{k}.$$

Questa analisi fornisce la probabilità che il tempo di esecuzione sia concentrato attorno alla sua media: questo tipo di probabilità è molto interessante perché se la probabilità che la concentrazione attorno alla media è alta, a tutti gli effetti pratici l'algoritmo si può considerare quasi deterministico perché non presenta grandi varianze.

Per determinare delle limitazioni più interessanti ci spostiamo dal caso generale del teorema di Markov e ci occupiamo dell'analisi della concentrazione di **certe** v.a. attorno alla loro media sfruttando i *bound di Chernoff*.

Sia $X = \sum_{i=1}^{n} X_i$ una v.a. dove X_i sono delle v.a. indicatrici (ovvero $P(X_i = 1) = p_i$ e $P(X_i = 0) = 1 - p_i$) mutuamente indipendenti ($X_i \perp X_j\ \forall i \neq j$). Segue che $E[X_i] = p_i$, da cui

$$\mu = E[X] = E\left[\sum_{i=1}^{n} X_i\right] = \sum_{i=1}^{n} E[X_i] = \sum_{i=1}^{n} p_i.$$

[1] In inglese *with high probability*, abbreviato in *whp*.

Se consideriamo un $\delta > 0$ e analizziamo lo scostamento di X rispetto alla sua media tramite il lemma di Markov otteniamo una limitazione superiore per niente interessante:

$$P(X > (1+\delta)\mu) \leq \frac{\mu}{(1+\delta)\mu} = \frac{1}{1+\delta}.$$

Teorema (Bound di Chernoff)**.** *Sia $X = \sum_{i=1}^{n} X_i$ una v.a. dove X_i sono delle v.a. indicatrici mutuamente indipendenti. Allora, $\forall \delta > 0$*

$$P(X > (1+\delta)\mu) < \left(\frac{e^{\delta}}{(1+\delta)^{(1+\delta)}} \right)^{\mu}.$$

Dimostrazione. Prendo un valore arbitrario $t > 0$ e considero la v.a. $Y_t = e^{tX}$. Allora,

$$\begin{aligned} P(X > (1+\delta)\mu) &= P(tX > t(1+\delta)\mu) \\ &= P(e^{tX} > e^{t(1+\delta)\mu}) \\ &= P(Y_t > e^{t(1+\delta)\mu}). \end{aligned}$$

Applicando il lemma di Markov (nella sua formulazione stretta) si ottiene

$$P(Y_t > e^{t(1+\delta)\mu}) < \frac{E[Y_t]}{e^{t(1+\delta)\mu}}.$$

Calcoliamo il valore atteso di Y_t:

$$\begin{aligned} E[Y_t] &= E[e^{tX}] \\ &= E[e^{t\sum_{i=1}^{n} X_i}] \\ &= E[e^{\sum_{i=1}^{n} tX_i}] \\ &= E\left[\prod_{i=1}^{n} e^{tX_i}\right] \\ &= \prod_{i=1}^{n} E[e^{tX_i}] \qquad (X_i \text{ indipendenti}) \end{aligned}$$

Calcoliamo ora il valore atteso $E[e^{tX_i}]$

$$E[e^{tX_i}] = e^{t\cdot 0}(1-p_i) + e^{t\cdot 1}p_i = 1 + p_i(e^t - 1)$$

e ricordando l'espansione in serie di e^x otteniamo che $1 + x < e^x$, pertanto

$$E[e^{tX_i}] = 1 + p_i(e^t - 1) < e^{p_i(e^t - 1)}.$$

Da quest'ultimo risultato posso concludere il calcolo di $E[Y_t]$ come segue

$$\begin{aligned} E[Y_t] &= \prod_{i=1}^{n} E[e^{tX_i}] \\ &= \prod_{i=1}^{n} (1 + p_i(e^t - 1)) \\ &< \prod_{i=1}^{n} e^{p_i(e^t-1)} \\ &= e^{\sum_{i=1}^{n} p_i(e^t-1)} \\ &= e^{(e^t-1)\sum_{i=1}^{n} p_i} \\ &= e^{(e^t-1)\mu}. \end{aligned}$$

Ritornando all'applicazione del lemma di Markov, otteniamo

$$P(Y_t > e^{t(1+\delta)\mu}) < \frac{e^{(e^t-1)\mu}}{e^{t(1+\delta)\mu}}.$$

Studiando la funzione rispetto a t del membro di destra della diseguaglianza per ottenerne un minimo, si ha che

$$\frac{d}{dt}\left(\frac{e^{(e^t-1)\mu}}{e^{t(1+\delta)\mu}}\right) = 0 \Leftrightarrow e^t\mu - (1+\delta)\mu = 0 \Leftrightarrow t = \ln(1+\delta).$$

Riapplicando il lemma di Markov per $t = \ln(1+\delta)$ otteniamo la tesi:

$$P(Y_t > e^{t(1+\delta)\mu}) < \frac{e^{(e^{\ln(1+\delta)}-1)\mu}}{e^{\ln(1+\delta)[(1+\delta)\mu]}} = \left(\frac{e^{\delta}}{(1+\delta)^{(1+\delta)}}\right)^{\mu}.$$

□

Da questo teorema seguono due corollari.

Corollario. *Se $\delta < 1$ si ha*

$$P(X > (1+\delta)\mu) < e^{-\frac{\delta^2}{3}\mu}.$$

Dimostrazione. Dalla dimostrazione del teorema di Chernoff si è ottenuto che

$$P(X > (1+\delta)\mu) < \frac{e^{(e^{\ln(1+\delta)}-1)\mu}}{e^{\ln(1+\delta)[(1+\delta)\mu]}} = e^{[\delta-(1+\delta)\ln(1+\delta)]\mu}.$$

Si consideri l'espansione di Taylor di $\ln(1+\delta)$:

$$\ln(1+\delta) = \sum_{i=1}^{+\infty} (-1)^{i+1}\frac{\delta^i}{i}$$

Allora

$$\begin{aligned}(1+\delta)\ln(1+\delta) &= \delta + \sum_{i=2}^{+\infty}(-1)^i\delta^i\left(\frac{1}{i-1}-\frac{1}{i}\right)\\ &> \delta + \frac{\delta^2}{2} - \frac{\delta^3}{6}\\ &\geq \delta + \frac{\delta^2}{2} - \frac{\delta^2}{6}\\ &\geq \delta + \frac{\delta^2}{3}\end{aligned}$$

da cui segue la tesi

$$P(X > (1+\delta)\mu) < e^{[\delta-\delta-\delta^2/3]\mu} = e^{-\frac{\delta^2}{3}\mu}.$$

□

Corollario. *Se $\delta < 1$ si ha*

$$P(X < (1-\delta)\mu) < e^{-\frac{\delta^2}{2}\mu}.$$

Dimostrazione. Si procede come prima: si ricava un limite superiore alla probabilità seguendo la stessa procedura fatta per dimostrare il teorema di Chernoff e poi si sfrutta l'espansione di Taylor di $\ln(1-\delta)$.

Consideriamo un valore arbitrario $t > 0$ e consideriamo la v.a. $Y_t = e^{-tX}$. Allora,

$$\begin{aligned}P(X < (1-\delta)\mu) &= P(-X > -(1-\delta)\mu)\\ &= P(e^{-tX} > e^{-t(1-\delta)\mu})\\ &= P(Y_t > e^{-t(1-\delta)\mu})\end{aligned}$$

Applicando il teorema di Markov (formulazione stretta) si ottiene

$$P(X < (1-\delta)\mu) < \frac{E[Y_t]}{e^{-t(1-\delta)\mu}}.$$

Calcoliamo il valore atteso di Y_t:

$$\begin{aligned}E[Y_t] &= \prod_{i=1}^{n} E[e^{-tX_i}]\\ &= \prod_{i=1}^{n}(1+p_i(e^{-t}-1))\\ &< \prod_{i=1}^{n} e^{p_i(e^{-t}-1)}\\ &= e^{\sum_{i=1}^{n} p_i(e^{-t}-1)}\\ &= e^{(e^{-t}-1)\mu}\end{aligned}$$

da cui segue

$$P(X < (1-\delta)\mu) < \frac{e^{(e^{-t}-1)\mu}}{e^{-t(1-\delta)\mu}}.$$

Per ottenere il bound migliore deriviamo l'espressione del membro di destra per ottenere un minimo:

$$\frac{d}{dt}\left(\frac{e^{(e^{-t}-1)\mu}}{e^{-t(1-\delta)\mu}}\right) = 0 \Leftrightarrow -e^{-t}\mu + (1-\delta)\mu = 0 \Leftrightarrow t = \ln\frac{1}{(1-\delta)}.$$

Riapplicando quindi il teorema di Markov per $t = \ln(1-\delta)^{-1}$ si ottiene:

$$\begin{aligned} P(X < (1-\delta)\mu) &< \frac{e^{\left(e^{-\ln\frac{1}{(1-\delta)}}-1\right)\mu}}{e^{-\ln\frac{1}{(1-\delta)}(1-\delta)\mu}} \\ &= \frac{e^{-\delta\mu}}{e^{-\ln\frac{1}{(1-\delta)}(1-\delta)\mu}} \\ &= e^{[-\delta-(1-\delta)\ln(1-\delta)]\mu}. \end{aligned}$$

Non rimane che considerare l'espansione di Taylor di $\ln(1-\delta)$

$$\ln(1-\delta) = \sum_{i=1}^{+\infty}(-1)\frac{\delta^i}{i}$$

per ottenere che

$$\begin{aligned} (1-\delta)\ln(1-\delta) &= -\delta + \sum_{i=2}^{+\infty}\delta^i\left(\frac{1}{i-1} - \frac{1}{i}\right) \\ &> -\delta + \frac{\delta^2}{2} \end{aligned}$$

e, infine, la tesi

$$P(X < (1-\delta)\mu) < e^{[-\delta+\delta-\delta^2/2]\mu} = e^{-\frac{\delta^2}{2}\mu}.$$

□

3.3 Quicksort randomizzato

Il quicksort è un algoritmo di ordinamento *in place* basato sulla partizione dell'insieme di chiavi S (si assuma senza perdita di generalità che tutte le chiavi siano distinte). Nel caso randomizzato, la scelta del pivot $y \in S$ è casuale. L'algoritmo riceve quindi in input l'insieme S da ordinare e il pivot $s \in S$ secondo cui effettuare le due partizioni $S_1 = \{x \in S \mid x < y\}$ e $S_2 = \{x \in S \mid x > y\}$ e restituisce in output la sequenza ordinata di S come $\mathrm{ord}(S) = \langle \mathrm{ord}(S_1), y, \mathrm{ord}(S_2)\rangle$.

Se il pivot fosse sempre la mediana di S, allora $|S_1|, |S_2| \leq \lceil\frac{n-1}{2}\rceil$. La complessità dell'algoritmo (basata sui confronti) sarebbe quindi

$$T(n) = 2T\left(\lceil\frac{n-1}{2}\rceil\right) + n - 1 = \Theta(n\log n).$$

Il calcolo della mediana ha un costo almeno lineare del tipo $O(7n)$. Proviamo quindi a rilassare il vincolo richiedendo che $|S_1|, |S_2| \leq \frac{3}{4}n$: dato che S_1 e S_2 sono una partizione di S, allora si ottiene anche che $|S_1|, |S_2| \geq \frac{n}{4}$.

Analizzando l'albero delle chiamate, notiamo che ogni livello contribuisce con lavoro $\leq n$, l'albero ha n foglie e al livello i-esimo la taglia di una partizione è $|S'| \leq (\frac{3}{4})^i n$: il numero di livelli è quindi logaritmico nella taglia dell'istanza. Definiamo quindi con h_{MAX} il massimo livello dell'albero delle chiamate:

$$\left(\frac{3}{4}\right)^{h_{MAX}} n \geq 1 \Leftrightarrow h_{MAX} \leq \log_{\frac{4}{3}} n = O(\log n);$$

allora

$$T(n) \leq n \cdot h_{MAX} = O(n \log n).$$

Sia $y \leftarrow \text{RANDOM}(S)$ con probabilità uniforme e si consideri la sequenza ordinata degli elementi di S

$$x_1 < x_2 < \cdots < x_{\frac{n}{4}} < x_{\frac{n}{4}+1} < \cdots < x_{\frac{3}{4}n} < x_{\frac{3}{4}n+1} < \cdots < x_n;$$

allora

$$P(y \in [x_{\frac{n}{4}+1}, x_{\frac{3}{4}n}]) = \frac{1}{2}.$$

Pertanto, se $y \in [x_{\frac{n}{4}+1}, x_{\frac{3}{4}n}]$ allora

$$\begin{cases} |S_1| \geq \frac{n}{4} \\ |S_2| \geq \frac{n}{4} \\ |S_1| + |S_2| \leq n-1 \end{cases} \Rightarrow |S_1|, |S_2| \leq \frac{3}{4}n.$$

Quindi, se siamo nel caso fortunato per il quale il pivot viene scelto all'interno dell'intervallo $[x_{\frac{n}{4}+1}, x_{\frac{3}{4}n}]$, segue che la lunghezza dei cammini sarà al massimo pari a $\log_{4/3} n$. L'algoritmo quindi "sbaglia" una volta su due e di conseguenza, **in media**, ogni cammino ha lunghezza $2 \log_{4/3} n$.

In 17 lo pseudocodice del quicksort randomizzato (si assume che tutti gli elementi in S siano distinti).

Algorithm 17 Quicksort randomizzato

```
function R_QUICKSORT(S)
    if |S| ≤ 1 then
        return ⟨S⟩
    end if
    y ←RANDOM(S)
    S1 ← {x ∈ S | x < y}
    S2 ← {x ∈ S | x > y}
    X1 ←R_QUICKSORT(S1)
    X2 ←R_QUICKSORT(S2)
    return ⟨X1, y, X2⟩
end function
```

Analisi in alta probabilità Consideriamo l'i-esimo cammino π_i e una costante $a > 1$ che fisseremo in seguito con l'analisi. Si consideri l'evento "$|\pi_i| \geq a \log_{4/3} n$": questo evento è equivalente all'evento "durante i primi $a \log_{4/3} n$ livelli del cammino π_i il numero di scelte "fortunate" è stato al più uguale a

$\log_{4/3} n$". Si noti che in alta probabilità questa implicazione corrisponde ad una maggiorazione del primo evento rispetto al secondo.

Definiamo ora la variabile aleatoria

$$X_i = \begin{cases} 1 & \text{se il pivot al livello } i \text{ è "buono"} \\ 0 & \text{altrimenti} \end{cases};$$

poiché la scelta del pivot è i.i.d. rispetto ad ogni iterazione, allora anche la variabile aleatoria X_i è i.i.d.. Osserviamo inoltre che è possibile definire la seguente v.a.

$$X = \sum_{i=1}^{a \log_{4/3} n} X_i$$

con la quale vengono contate il numero di scelte "fortunate" del pivot. Per come è definita X, questa soddisfa uno dei lemmi di Chernoff.

Proseguiamo quindi con l'analisi con l'obiettivo di applicare il corollario 3.2. Il nostro obiettivo è quello di calcolare

$$P(|\pi_i| \geq a \log_{4/3} n) \leq P(X \leq \log_{4/3} n)$$

poiché se il cammino è lungo almeno $a \log_{4/3} n$ allora le scelte fortunate sono al più $\log_{4/3} n$.

Per far ciò abbiamo bisogno del valore atteso di X:

$$\begin{aligned} E[X] &= E[\sum_{i=1}^{a \log_{4/3} n} X_i] \\ &= \sum_{i=1}^{a \log_{4/3} n} E[X_i] \\ &= \sum_{i=1}^{a \log_{4/3} n} \frac{1}{2} \\ &= \frac{1}{2} a \log_{4/3} n. \end{aligned}$$

Abbiamo inoltre bisogno di modulare opportunamente a e δ in modo da poter applicare il corollario: deve quindi essere soddisfatta la relazione

$$\log_{4/3} n = (1-\delta)\mu = (1-\delta)\frac{1}{2} a \log_{4/3} n.$$

Una possibile scelta è quella di fissare $a = 8$ e $\delta = 3/4$:

$$(1-\delta)\frac{1}{2} a \log_{4/3} n = \left(1 - \frac{3}{4}\right)\frac{1}{2} 8 \log_{4/3} n = \log_{4/3} n.$$

Pertanto, con $\delta = 3/4$ e $\mu = 4\log_{4/3} n$, si ha

$$\begin{aligned}
P(|\pi_i| \geq a\log_{4/3} n) &\leq P(X \leq \log_{4/3} n) \\
&= P\left(X \leq \left(1 - \frac{3}{4}\right) 4\log_{4/3} n\right) \\
&< e^{-\frac{\frac{9}{16} 4\log_{4/3} n}{2}} \\
&= e^{-\frac{9}{8}\log_{4/3} n} \\
&< e^{-\log_{4/3} n} \\
&= e^{-\frac{\ln n}{\ln \frac{4}{3}}} \\
&= (e^{-\ln n})^{\frac{1}{\ln \frac{4}{3}}} \\
&= \frac{1}{n^{\frac{1}{\ln \frac{4}{3}}}} \\
&\approx \frac{1}{n^{3,47}} \\
&< \frac{1}{n^3}
\end{aligned}$$

Tutto quello che è stato fatto finora è valido per un cammino fissato. Durante l'esecuzione dell'algoritmo vengono percorsi vari cammini e la condizione appena esposta non deve accadere per nessuno dei cammini di esecuzione. Definito l'evento $E_i =$"$|\pi_i| \geq 8\log_{4/3} n$", attraverso l'*union bound* troviamo

$$P\left(\bigcup_{i=1}^{n} E_i\right) \leq \sum_{i=1}^{n} P(E_i) < \frac{n}{n^3} = \frac{1}{n^2}.$$

3.4 Taglio minimo di un multigrafo

In questa sezione vediamo un algoritmo Montecarlo.

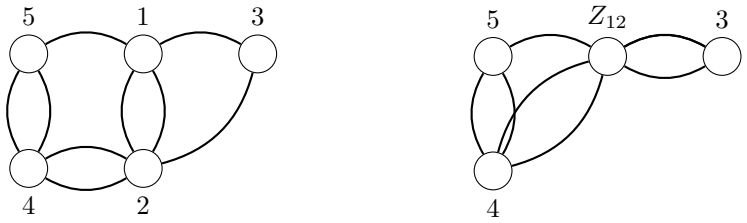
Definizione. *Un* ***multigrafo*** $\mathcal{G} = (V, \mathcal{E})$ *è un grafo definito su un insieme di vertici* V *e su un multiinsieme*[2] *di archi* $\mathcal{E}$ *dove ad ogni arco è associata una molteplicità.*

In questa sezione considereremo solamente multigrafi non orientati.

Definizione. *Un* ***taglio*** $\mathcal{C} \in \mathcal{G} = (V, \mathcal{E})$ *è un multiinsieme di archi* $\mathcal{C} \subseteq \mathcal{E}$ *tali che* $\mathcal{G}' = (V, \mathcal{E} - \mathcal{C})$ *è disconnesso. Se* $\mathcal{G}$ *è disconnesso allora* $\mathcal{C} = \emptyset$ *è un taglio.*

Definiamo l'operazione di contrazione su un multigrafo.

[2]Un multiinsieme è una generalizzazione di un insieme: in esso sono accettati elementi ripetuti. Di conseguenza è possibile associare ad ogni elemento una molteplicità, definita come il numero di volte che l'elemento compare nel multiinsieme, indicata con la notazione $m(e)$ con e arco del multigrafo. Per i multiinsiemi la classica notazione con parentesi graffe ($\{\}$) viene sostituita con la notazione con doppie parentesi graffe ($\{\{\}\}$)



(a) $\mathcal{G}$ (b) $\mathcal{G}/\{1,2\}$

Figura 3.1: Esempio di una contrazione

Definizione. *Dato un multigrafo $\mathcal{G} = (V, \mathcal{E})$ e dato un arco $\{u, v\} \in \mathcal{E}$, la **contrazione** di $\mathcal{G}$ rispetto all'arco $\{u, v\}$ è il multigrafo $\mathcal{G}/\{u, v\} = (V', \mathcal{E}')$ con*

$$\begin{aligned} V' &= V - \{u, v\} \cup \{Z_{u,v}\} \\ \mathcal{E}' &= \mathcal{E} - \{\{\ \{u, x\} \in \mathcal{E}\ \}\} - \{\{\ \{v, x\} \in \mathcal{E}\ \}\} \\ &\quad \cup \{\{\ \{Z_{u,v}, x\}\ :\ (x \neq u, v) \wedge ((\{u, x\} \in \mathcal{E}) \vee (\{v, x\} \in \mathcal{E}))\ \}\} \end{aligned}$$

In figura 3.1 è possibile vedere un esempio di contrazione.
Dalla definizione di contrazione segue immediatamente che

$$\begin{gathered} |V'| = |V| - 1 \\ |\mathcal{E}'| \leq |\mathcal{E}| - 1 \Leftrightarrow |\mathcal{E}'| = |\mathcal{E}| - \mathrm{m}(\{u, v\}); \end{gathered}$$

l'ultima espressione deriva dal fatto che gli unici lati che vengono tolti sono quelli tra u e v e questi sono **almeno** 1.

Vale la seguente proposizione.

Proposizione. *Per un qualsiasi taglio $\mathcal{C}' \in \mathcal{G}/e$ esiste un taglio $\mathcal{C} \in \mathcal{G}$ tale che $|\mathcal{C}'| = |\mathcal{C}|$.*

Dimostrazione. Sia $\mathcal{G}/e = (V', \mathcal{E}')$ e si supponga che $\mathcal{C}' \subseteq \mathcal{E}'$ sia un taglio. Considero il multiinsieme $\mathcal{C}$ in $\mathcal{G}$ come il multiinsieme degli archi in $\mathcal{G}$ ottenuto ripristinando gli archi di $\mathcal{C}'$ che contengono il nodo Z_{uv}:

$$\{Z_{uv}, x\} \in \mathcal{C}' \Rightarrow \{u, x\} \in \mathcal{C}\ \wedge\ \{v, x\} \in \mathcal{C}.$$

Così facendo si ha che $|\mathcal{C}| = |\mathcal{C}'|$. In $\mathcal{G}/e$ la rimozione degli archi in $\mathcal{C}'$ disconnette in particolare il nodo Z_{uv} da un qualche nodo $x \in V'$; ogni cammino $\Pi_{Z_{uv},x}$ in $\mathcal{G}/e$ contiene almeno un arco di $\mathcal{C}'$. Supponiamo per assurdo che $\mathcal{C}$ non sia un taglio di $\mathcal{G}$. Allora in $\mathcal{G}$ tra i nodi u e x esiste un cammino $\Pi_{u,x} \subseteq \mathcal{E} - \mathcal{C}$. Di conseguenza, poiché $\Pi_{u,x}$ non contiene archi di $\mathcal{C}$, il corrispondente cammino $\Pi_{Z_{uv},x}$ in $\mathcal{G}/e$ non usa archi di $\mathcal{C}'$ (per costruzione di $\mathcal{C}$): dovendo concludere che $\mathcal{C}'$ non è un taglio, si arriva ad un assurdo. □

Da questa proposizione segue immediatamente il seguente corollario.

Corollario. *Il processo di contrazione **non diminuisce** la cardinalità del taglio minimo, ovvero*

$$|\mathrm{MIN\_CUT}(\mathcal{G}/e)| \geq |\mathrm{MIN\_CUT}(\mathcal{G})|.$$

Dimostrazione. A seguito della dimostrazione della proposizione precedente, si osserva che

$$\{|\mathcal{C}'| \ : \ \mathcal{C}' \text{ taglio in } \mathcal{G}/e\} \subseteq \{|\mathcal{C}| \ : \ \mathcal{C} \text{ taglio in } \mathcal{G}\}.$$

Poiché il minimo di un sottoinsieme è almeno pari al minimo dell'insieme che lo contiene, segue la tesi. □

Corollario. *Se $\mathcal{C}$ è un taglio di $\mathcal{G}$ e $e \notin \mathcal{C}$ allora il multiinsieme di archi $\mathcal{C}'$ corrispondente a $\mathcal{C}$ in $\mathcal{G}/e$ è ancora un taglio.*

Dimostrazione. Sia $e = \{u, v\}$ l'arco rispetto al quale si effettua la contrazione. Per ipotesi, la rimozione di $\mathcal{C}$ da $\mathcal{E}$ lascia i nodi u, v nella stessa componente connessa (in particolare sono connessi direttamente) e comporterà l'esistenza di almeno un nodo, sia x, disconnesso da u e v. Tutti i cammini $\Pi_{u,x}$ e $\Pi_{v,x}$ contengono almeno un arco di $\mathcal{C}$. In $\mathcal{G}/e$ ogni cammino $\Pi_{Z_{uv},x}$ contiene un arco di $\mathcal{C}'$ per costruzione. Eliminando $\mathcal{C}'$ da $\mathcal{G}/e$ si ottiene la disconnessione di Z_{uv} da x, concludendo così che $\mathcal{C}'$ è un taglio. □

3.4.1 L'algoritmo di Karger

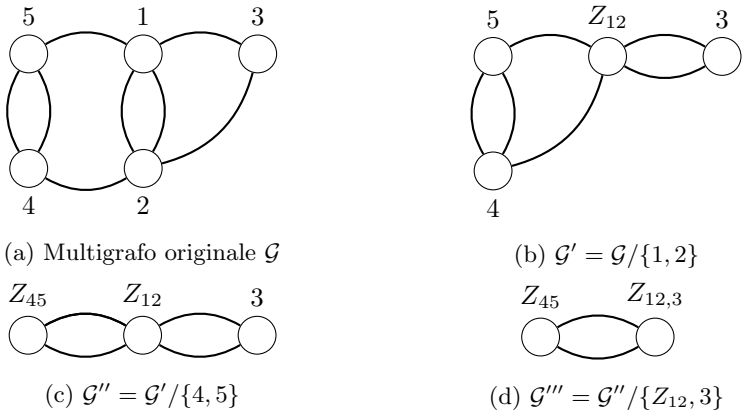
In questa sezione forniamo l'algoritmo di Karger per il calcolo del taglio minimo di un multigrafo. In 18 è descritta la procedura per effettuare la contrazione piena di un multigrafo: essa termina quando il grafo avrà solamente due nodi e verrà restituito il numero di archi presenti nel fascio che connette tali nodi. La randomizzazione è data dalla scelta dell'arco secondo cui fare la contrazione: si noti che per ogni esecuzione, tale scelta è indipendente dalle precedenti. Poiché la costruzione del grafo contratto è costituita da un numero costante di operazioni sui nodi e sugli archi, il tempo di esecuzione dell'algoritmo è lineare nella taglia del multigrafo: $T_{FC} = O(m)$ con $m = |\mathcal{E}|$. Segue un esempio di applicazione dell'algoritmo.

Algorithm 18 Full contraction

```
function FULL_CONTRACTION(G = (V, E))
    for i ← 1 to |V| − 2 do
        e ←RANDOM(E)
        G ← G/e
    end for
    return |E|
end function
```

Esempio. *Un'applicazione dell'algoritmo FULL_CONTRACTION è possibile vederla in figura 3.2. L'esempio restituisce un taglio minimo di cardinalità 2 che è proprio la soluzione ottima per il grafo di partenza. Si noti che per il corollario 3.4 la procedura garantisce solo che il taglio minimo di un grafo contratto non sia mai più piccolo del taglio minimo del grafo originale.*

In 19 è descritto l'algoritmo di Karger per il computo del taglio minimo di un multigrafo. L'algoritmo è un semplice algoritmo di accumulazione del minimo. Si noti inoltre che le chiamate alla procedura di contrazione sono



(a) Multigrafo originale $\mathcal{G}$

(b) $\mathcal{G}' = \mathcal{G}/\{1,2\}$

(c) $\mathcal{G}'' = \mathcal{G}'/\{4,5\}$

(d) $\mathcal{G}''' = \mathcal{G}''/\{Z_{12},3\}$

Figura 3.2: Esempio di esecuzione di FULL_CONTRACTION

tutte indipendenti tra loro. Il secondo parametro di ingresso dell'algoritmo ci permette di regolare il numero di contrazioni da eseguire e di conseguenza, come vedremo dall'analisi, ci permette di limitare la probabilità che l'algoritmo fallisca. Si deduce immediatamente che $T_K = O(km)$.

Algorithm 19 Algoritmo di Karger

```
function KARGER(G = (V', E), k)
    min ← +∞
    for i ← 1 to k do
        t ←FULL_CONTRACTION(G = (V', E))
        if t < min then
            min ← t
        end if
    end for
    return min
end function
```

Prima di proseguire con l'analisi effettuiamo alcuni richiami di probabilità.

Richiami di probabilità Dati due eventi E_1 e E_2 tra loro indipendenti, la probabilità che si verifichino entrambi è

$$P(E_1 \cap E_2) = P(E_1)P(E_2).$$

Dati due eventi E_1 e E_2, con $P(E_1) \neq 0$, la probabilità che il secondo evento si verifichi sapendo che si è verificato il primo (**probabilità condizionale**) è

$$P(E_2|E_1) = \frac{P(E_1 \cap E_2)}{P(E_1)} \Leftrightarrow P(E_1 \cap E_2) = P(E_2|E_1)P(E_1).$$

L'ultima formulazione è possibile generalizzarla per una sequenza di k eventi (si dimostra per induzione su k):

$$P\left(\bigcap_{i=1}^{k} E_i\right) = P(E_1) \prod_{i=2}^{k} P\left(E_i | \bigcap_{j=1}^{i-1} E_j\right).$$

Analisi Fissiamo un dato taglio minimo $\mathcal{C}^*$ di $\mathcal{G}$. Vogliamo studiare la probabilità che nessun arco di $\mathcal{C}^*$ venga selezionato per una contrazione (altrimenti tale arco verrebbe eliminato dalla contrazione e non potrebbe far parte del taglio).

Definizione. *In un multigrafo, il* ***grado*** *di un nodo è*

$$d(v) = \sum_{v \in e} m(e) = |\{\{ \ \{v, x\} \in \mathcal{E}, \ x \in V, \ x \neq v\}\}|.$$

Dalle definizione si osserva che $\{\{ \ \{v, x\} \in \mathcal{E} \ \}\}$ è un taglio (se si elimina questo multiinsieme si disconnette v dal resto del multigrafo). Da questo segue immediatamente che $|\mathcal{C}^*| \leq d(v) \ \forall v \in V$.

Proposizione. *Sia $\mathcal{G} = (V, \mathcal{E})$ un multigrafo. Se $\mathcal{G}$ ha un taglio minimo $\mathcal{C}^*$ con $|\mathcal{C}^*| = t$, allora*

$$|\mathcal{E}| \geq t\frac{n}{2},$$

con $n = |V|$.

Dimostrazione. La dimostrazione segue immediatamente dall'osservazione precedente:

$$|\mathcal{E}| = \frac{\sum_{v \in V} d(v)}{2} \geq \sum_{v \in V} \frac{t}{2} = t\frac{n}{2}.$$

□

Definiamo l'evento E_i ="l'i-esima scelta di arco casuale in FULL_CONTRACTION non tocca un arco di $\mathcal{C}^*$". Allora quello che vogliamo studiare è

$$P\left(\bigcap_{i=1}^{n-2} E_i\right).$$

Calcoliamo le probabilità passo per passo:

1. alla prima iterazione la probabilità di selezionare un arco del taglio è

$$P(E_1) \geq 1 - \frac{t}{n\frac{t}{2}} = 1 - \frac{2}{n},$$

   ovvero è estremamente improbabile;

2. alla seconda iterazione la probabilità diviene

$$P(E_2|E_1) \geq 1 - \frac{t}{(n-1)\frac{t}{2}} = 1 - \frac{2}{n-1},$$

   notando che la probabilità comincia a crescere;

3. all'i-esima iterazione, dato l'andamento mostrato dalle probabilità, si ha

$$P\left(E_i | \bigcap_{j=1}^{i-1} E_j\right) \geq 1 - \frac{2}{n-i+1}.$$

Quindi

$$\begin{aligned} P\left(\bigcap_{i=1}^{n-2} E_i\right) &\geq \prod_{i=1}^{n-2}\left(1-\frac{2}{n-i+1}\right) \\ &= \prod_{i=1}^{n-2}\frac{n-i-1}{n-i+1} && \text{(prodotto telescopico)} \\ &= \frac{2}{n(n-1)} \\ &= \frac{1}{\binom{n}{2}} \\ &\geq \frac{2}{n^2} \end{aligned}$$

La probabilità non è molto grande ma nemmeno così pessima: facendo girare l'algoritmo $\Theta(n^2)$ volte si ha la certezza che non fallisca. L'algoritmo infatti privilegia i tagli minimi poiché un taglio minimo viene selezionato con una probabilità che è solo inversamente proporzionale ad un polinomio mentre i tagli in generale sono in numero esponenziale. Scegliamo $k = \frac{n^2}{2} d \ln n$ (il fattore logaritmico ci servirà per l'analisi in alta probabilità). Allora[3]

$$\begin{aligned} P(\text{KARGER non ritorna } |\mathcal{C}^*|) &= P(\text{tutte le } k \text{ contrazioni falliscono}) \\ &\leq \left(1-\frac{2}{n^2}\right)^{\frac{n^2}{2} d \ln n} \\ &< e^{-d \ln n} \\ &= \frac{1}{n^d} \end{aligned}$$

e in questo modo è possibile rendere la probabilità che l'algoritmo di Karger fallisca piccola a piacere pilotando la costante d.

In conclusione si ottiene che $T_K(n,m) = O(mn^2 \log n)$. Questa complessità non è molto lontana da quelle ottenute attraverso altri approcci di risoluzione (ad esempio il calcolo del flusso): l'algoritmo di Karger in questo fallisce perché, sebbene all'inizio le probabilità che per una contrazione venga scelto un arco del taglio minimo siano molto basse, presto queste degradano. Nella prossima sezione vedremo un miglioramento dell'algoritmo.

3.4.2 L'algoritmo di Karger-Stein

Un approccio migliorativo dell'algoritmo di Karger è dato dall'algoritmo di Karger-Stein. L'idea è quella di effettuare molte contrazioni parziali (circa $n^2 \ln n$) in parallelo: l'obiettivo è quello di fare in modo che l'algoritmo valorizzi le prime contrazioni in alta probabilità.

L'approccio è di carattere ricorsivo: in 20 è descritta la procedura per eseguire una contrazione parziale che verrà utilizzata nell'algoritmo *devide and*

[3]Si ricordi che

$$\lim_{x\to\infty}\left(1+\frac{1}{x}\right)^x = e.$$