Appunti di Algoritmica Avanzata

Luca Vallerini

A.A. 2016/2017

I seguenti appunti sono tratti dalle lezioni del corso di *Algoritmica avanzata* tenuto dal professor Geppino Pucci per il corso di laurea magistrale in Ingegneria Informatica presso l'Università degli Studi di Padova nell'A.A. 2016/2017. Vengono date per note le conoscenze e le competenze acquisite durante il corso di *Dati e algoritmi 2*.

Sono stati usati inoltre il libro di riferimento del corso (T.H. Cormen, C.E. Leiserson, R.L. Rivest, C. Stein., *Introduction to Algorithms - Third Edition*, MIT Press, Cambridge Mass. USA, 2009), e le dispense fornite dal docente sul sito del corso (`https://www.dei.unipd.it/~geppo/AA/index.html`).

Alcune dimostrazioni presenti nella dispensa sono state lasciate dal docente come esercizio per casa pertanto non ne assicuro la correttezza.

La dispensa è stata scritta da me e non è stata né revisionata né concordata con il docente per cui non ne assicuro la correttezza.

Quest'opera è distribuita con licenza Attribution-ShareAlike 4.0 International (`http://creativecommons.org/licenses/by-sa/4.0/`).

Indice

Lista degli algoritmi

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:

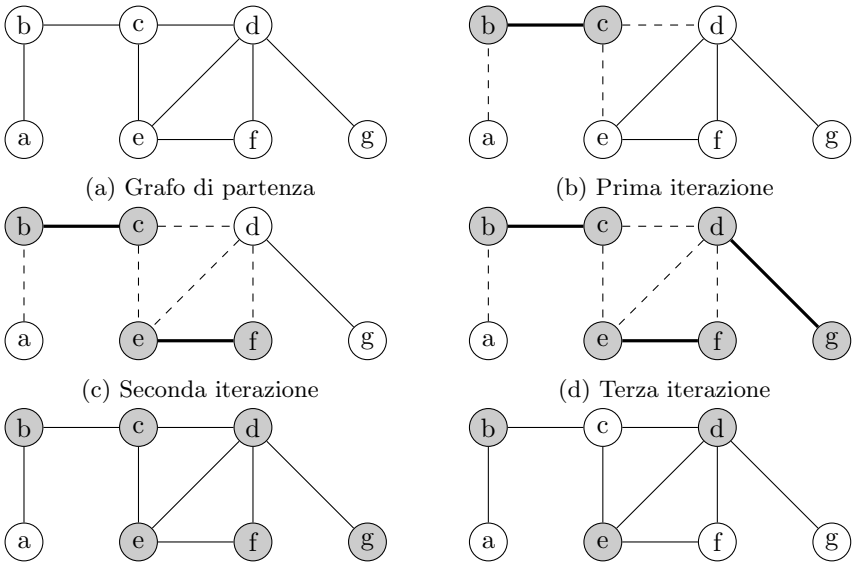
$$|V^*| \geq \frac{|V'|}{2} \Rightarrow \frac{|V'|}{|V^*|} \leq 2 \Leftrightarrow \frac{c(A_\pi(i))}{c(s^*(i))} \leq 2 = \rho(n)\,,$$

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:*

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

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

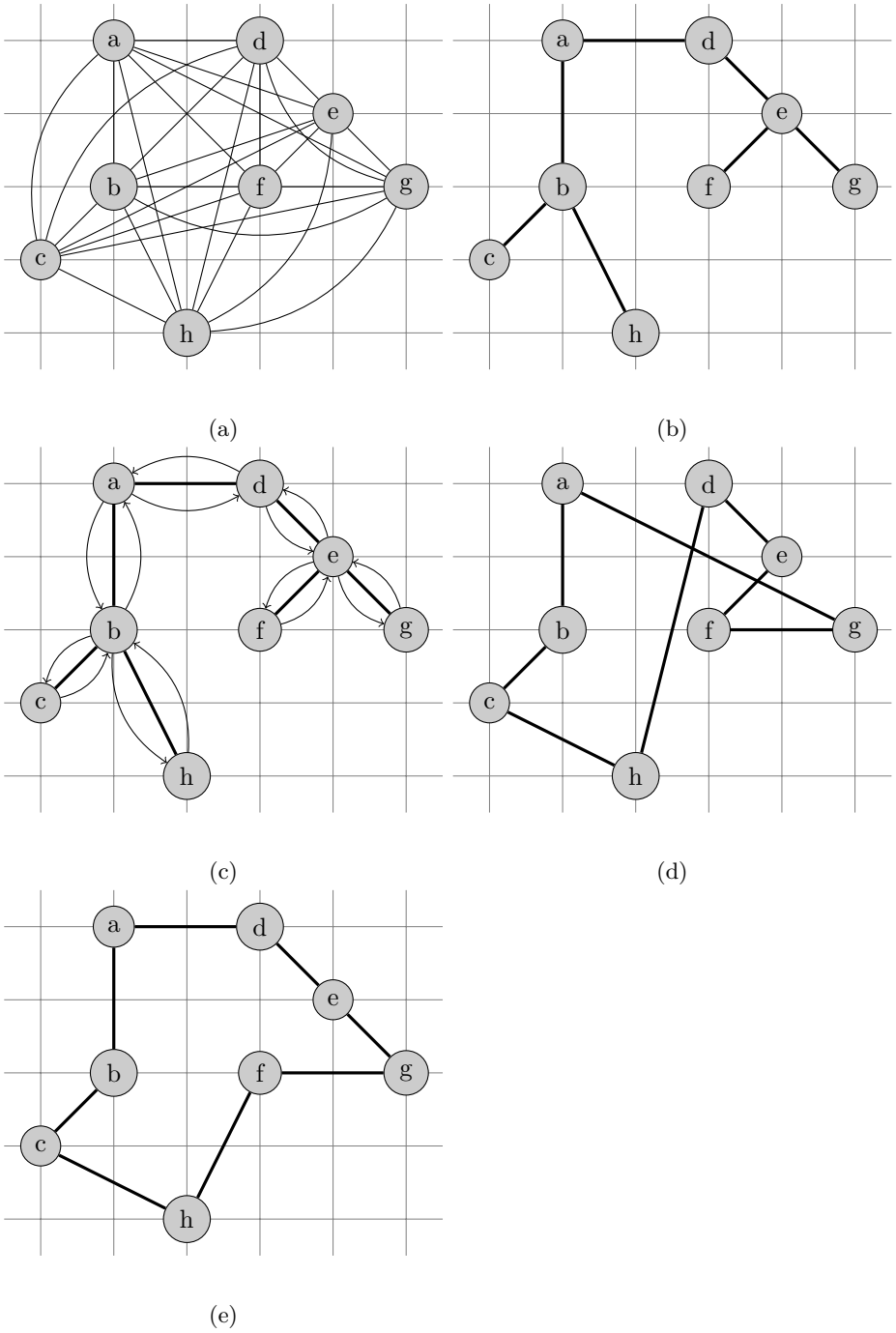


Figura 1.2: Esempio di applicazione dell'algoritmo di approssimazione per TSP

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

gli associamo un peso grande nel caso opposto. Quindi, $\forall x \in X$, sia i la prima iterazione che copre x: allora il peso associato ad x è

$$c_x = \frac{1}{|S_i \cap U_i|}$$

con $S_1, S_2, \cdots, S_t$ l'insieme selezionato durante la $1, 2, \cdots, t$-esima iterazione.

Si osservi che

$$\sum_{x \in S_i \cap U_i} c_x = \frac{|S_i \cap X_i|}{|S_i \cap X_i|} = 1.$$

Segue che

$$|\mathcal{C}| = \sum_{i=1}^{|\mathcal{C}|} 1 = \sum_{i=1}^{|\mathcal{C}|} \sum_{x \in S_i \cap U_i} c_x = \sum_{x \in X} c_x$$

da cui

$$\sum_{S \in \mathcal{C}^*} \sum_{x \in S} c_x \geq \sum_{x \in X} c_x = |\mathcal{C}|$$

poiché $\forall x \in X \ \exists S \subseteq \mathcal{C}^* \ : \ x \in S$, essendo $\mathcal{C}^*$ un insieme ricoprente, e l'elemento x compare in **almeno uno** degli insiemi S selezionati per l'insieme ricoprente.

Ora, quello che noi vogliamo dimostrare è che $\forall S \in \mathcal{F}$

$$\sum_{x \in S} c_x \leq H_{|S|}$$

da cui seguono le seguenti relazioni:

$$\begin{aligned} |\mathcal{C}| &\leq \sum_{S \in \mathcal{C}^*} \sum_{x \in S} c_x \\ &\leq \sum_{S \in \mathcal{C}^*} H_{|S|} \\ &\leq \sum_{S \in \mathcal{C}^*} H_{\max\{|S| \ : \ S \in \mathcal{F}\}} \\ &\leq |\mathcal{C}^*| H_{\max\{|S| \ : \ S \in \mathcal{F}\}} \end{aligned}$$

ovvero

$$\frac{|\mathcal{C}|}{|\mathcal{C}^*|} \leq H_{\max\{|S| \ : \ S \in \mathcal{F}\}}.$$

Si consideri un generico insieme $S \in \mathcal{F}$ e si definisca per $1 \leq i \leq |\mathcal{C}|$ una funzione che esprima il numero di elementi **scoperti** di S dopo la fine dell'i-esima iterazione come segue:

$$\begin{aligned} n_i &= |S \cap U_{i+1}| \\ n_0 &= |S| \end{aligned}$$

Si noti che tale funzione rappresenta una successione non crescente[5]:

$$n_0 \geq n_1 \geq \cdots \geq n_k = 0$$

[5]In generale $\exists i \ : \ n_{i-1} = n_i$, pertanto non posso affermare che sia decrescente.

dove $k \leq |C|$ è l'iterazione in cui S viene coperto completamente per la prima volta.

Si noti che all'i-esima iterazione il numero di elementi che vengono coperti per la prima volta è $|S_i \cap U_i| = \max\{|T \cap U_i| : T \in \mathcal{F}\}$. Inoltre, per definizione, $n_{i-1} = |S \cap U_i|$: da ciò segue che $|S_i \cap U_i| \geq n_{i-1}$ poiché la scelta greedy S_i garantisce che S non copra più elementi **nuovi** di S_i, altrimenti S sarebbe stato scelto al posto di S_i.

Allora:

$$\sum_{x \in S} c_x = \sum_{i=1}^{k} \sum_{x \in (S_i \cap U_i) \cap S} c_x$$

dove $x \in (S_i \cap U_i) \cap S$ rappresenta l'insieme degli elementi nuovi selezionati all'i-esima iterazione **e** che appartengono all'insieme S considerato

$$= \sum_{i=1}^{k} \sum_{x \in (S_i \cap U_i) \cap S} \frac{1}{|S_i \cap U_i|}$$

$$= \sum_{i=1}^{k} \frac{n_{i-1} - n_i}{|S_i \cap U_i|}$$

dove $n_{i-1} - n_i$ è da leggere come il numero di elementi scoperti di S alla fine della $(i-1)$-esima iterazione, a cui viene sottratto il numero di elementi scoperti di S alla fine dell'i-esima iterazione, ottenendo così il numero di elementi di S coperti con l'i-esima iterazione

$$\leq \sum_{i=1}^{k} \frac{n_{i-1} - n_i}{n_{i-1}}$$

$$= \sum_{i=1}^{k} \sum_{j=n_i+1}^{n_{i-1}} \frac{1}{n_{i-1}}$$

$$\leq \sum_{i=1}^{k} \sum_{j=n_i+1}^{n_{i-1}} \frac{1}{j} \qquad \text{(poiché } j \leq n_{i-1}\text{)}$$

$$= \sum_{i=1}^{k} \left(\sum_{j=1}^{n_{i-1}} \frac{1}{j} - \sum_{j=1}^{n_i} \frac{1}{j} \right)$$

$$= \sum_{i=1}^{k} (H_{n_{i-1}} - H_{n_i}) \qquad \text{(somma telescopica)}$$

$$= H_{n_0} - H_{n_k}$$

$$= H_{|S|} - H_0$$

$$= H_{|S|}$$

concludendo così l'analisi. □

Esistono dei problemi che possono beneficiare di questo fattore di approssimazione. Si consideri il problema 3-D-VERTEX-COVER, una restrizione di VERTEX-COVER per la quale il grado massimo del grafo è 3. Nonostante la sparsità del grafo (VERTEX-COVER per un albero è un problema facile), è possibile dimostrare che il problema è difficile. Applicando la formulazione di

SET-COVER per il problema 3-D-VERTEX-COVER, la collezione di sottoinsiemi sarà composta da sottoinsiemi di cardinalità al più 3 poiché per ipotesi il grafo ha grado 3. Segue quindi che il fattore di approssimazione è

$$\rho \leq H_3 = 1 + \frac{1}{2} + \frac{1}{3} = \frac{11}{6} < 2,$$

migliore del fattore di approssimazione dell'algoritmo approssimato per VERTEX-COVER.

1.6 Il problema SUBSET-SUM

Analizziamo ora un FPTAS per il problema SUBSET-SUM.

Il problema SUBSET-SUM in forma decisionale è il seguente:

$$\begin{cases} \langle S, t \rangle, \ \forall s \in S \ : \ s \in \mathbb{N}, t \in \mathbb{N}^+ \\ \exists S' \subseteq S | \sum_{s \in S'} s = t? \end{cases}$$

Il problema di ottimo invece, considerata una istanza $i = \langle S, t \rangle$, definito l'insieme di tutte le approssimazioni per difetto di t, $S(i) = \{S' \subseteq S \mid \sum_{s \in S'} s \leq t\}$, e definita la funzione di costo $c(S') = \sum_{s \in S'} s$, mira a trovare l'insieme S' di costo massimo.

Stiamo cercando un FPTAS, quindi $\forall \epsilon > 0 \ \exists S_\epsilon^{'} \mid c(S^*)/c(S_\epsilon') \leq (1 + \epsilon)$ con $T(n, \epsilon) = \text{polynomial}(n, \epsilon)$.

Per prima cosa forniamo un algoritmo di enumerazione esaustiva di tutte le soluzioni ammissibili: come vedremo, tale algoritmo è un algoritmo esponenziale tuttavia ci fornirà un punto di partenza per elaborare un algoritmo di approssimazione.

Sia l'insieme $S = \{x_1, x_2, \cdots, x_n\}$. Allora:

- sia L_i la lista ordinata e senza duplicati di tutti i costi ammissibili, ovvero minori o uguali a t, di sottoinsiemi di $\{x_1, x_2, \cdots, x_i\}$;
- sia $L_i +_t x$ una operazione che restituisce una lista ordinata ottenuta sommando x ad ogni elemento di L_i ed eliminando i valori non più ammissibili;
- sia MERGE_SD(L_1, L_2) la funzione che restituisce la lista ordinata ottenuta fondendo ordinatamente le liste L_1 e L_2 ed eliminandone i duplicati.

Algorithm 5 Algoritmo di enumerazione esaustiva per SUBSET-SUM

```
function EXP_SS(S, t)
    * S = {x_0, x_1, ..., x_n} *
    L_0 ← ⟨0⟩
    for i ← 1 to n do
        L_i ←MERGE_SD(L_{i-1}, L_{i-1} +_t x_i)
    end for
    return MAX(L_n)
end function
```

L'algoritmo 5 presenta lo pseudocodice per l'algoritmo esaustivo. Al caso peggiore, l'operazione di *merge* restituisce una lista di lunghezza pari al

doppio della lista dell'iterazione precedente, quindi $|L_i| = 2^i \in \Omega(2^i)$: come conseguenza, il tempo di esecuzione al caso peggiore è $T = O(2^n)$.

L'idea che sta dietro all'FPTAS è quella di sfoltire le liste L_i: ogni qualvolta un costo ammissibile è ben approssimato, tengo solo il valore ammissibile come rappresentante della soluzione migliore. Utilizzo sempre una approssimazione per difetto per evitare di "sfondare" t componendo le soluzioni approssimate.

Per fare questo è necessario introdurre il concetto di diradamento. Dato un valore $0 < \delta < 1$, un δ-TRIM di una lista L è la lista $L' \subseteq L \ : \ \forall y \in L \ \exists z \in L' \mid \frac{y}{1+\delta} \leq z \leq y$ (approssimazione per difetto). Il diradamento tiene solo valori approssimati di y tali per cui l'errore relativo compiuto è al più δ:

$$\frac{y-z}{y} \leq \frac{y(1-1/(1+\delta))}{y} \leq 1 - \frac{1}{1+\delta} = \frac{\delta}{1+\delta} \leq \delta$$

Algorithm 6 Algoritmo di δ-diradamento

```
function TRIM(L, δ)
    * sia L = ⟨y_0, y_1, ··· , y_m⟩ *
    last ← 0
    L' ← ⟨y_0⟩
    for i ← 1 to m do
        if y_i > y_last(1 + δ) then
            L' ← L' + y_i
            last ← i
        end if
    end for
    return L'
end function
```

L'algoritmo 6 restituisce un δ-diradamento: in particolare, il diradamento che restituisce è tale da essere di cardinalità minima.

Proprietà di scelta greedy

Dimostrazione. La scelta greedy deve essere contenuta nella soluzione ottima poiché il primo elemento della lista non può essere approssimato da nessun'altro elemento se non da se stesso visto che la lista è ordinata. □

Proprietà di sottostruttura ottima

Dimostrazione. Sia $L^* = y_0 + L'$ la soluzione ottima del problema. Bisogna dimostrare che L' è la soluzione ottima del problema residuo L_r. Si osserva che la fase di clean up elimina tutti e solo gli elementi approssimati dalla scelta greedy (y_0), di conseguenza il resto della soluzione deve contenere una soluzione ammissibile del problema residuo. In particolare, questa soluzione è la soluzione ottima: se così non fosse allora esisterebbe una lista $L'' \subseteq L_r \ : \ |L''| < |L'|$ tale per cui la lista $L'' + y_0$ rappresenterebbe una lista ammissibile per il problema originale L di taglia minore di L^* che abbiamo assunto ottima per ipotesi (si avrebbe $|L^*| = |L'' + y_0| < |L'| + y_0 = |L^*|$). Dall'assurdo segue quindi che L' è una soluzione ammissibile del problema residuo di cardinalità minima. □

La lista $L' = \langle z_0, z_1, \cdots, z_{k-1} \rangle$ restituita dall'algoritmo di diradamento è tale per cui

$$\forall i > 0 \ : \ \frac{z_{i+1}}{z_i} > 1 + \delta;$$

si faccia attenzione che tale proprietà non vale per $i = 0$ perché z_0 potrebbe essere nullo.

Lo pseudocodice per l'FPTAS per SUBSET-SUM è disponibile come algoritmo 7.

Algorithm 7 Algoritmo FPTAS per SUBSET-SUM

```
function APPROX_SUBSET_SUM(S, t, ε)
    * S = {x_0, x_1, ··· , x_n} *
    L_0 ← ⟨0⟩
    for i ← 1 to n do
        L_i ←MERGE_SD(L_{i−1}, L_{i−1} +_t x_i)
        L_i ←TRIM(L_i, ε/2n)
    end for
    return MAX(L_n)
end function
```

Correttezza Si prova per induzione. La lista L_0 contiene il solo valore 0, pertanto è ammissibile. Se ipotizzo che la lista L_{i-1} sia ammissibile, allora la lista L_i continua ad essere ammissibile perché i costi non ammissibili vengono scartati dall'operazione di *merge*. L'operazione di diradamento non aggiunge mai costi (al più ne toglie qualcuno), pertanto al termine dell'i-esima iterazione la lista L_i rimane ammissibile.

Lemma. *Sia $y = \sum_{x \in S'} x$ un costo ammissibile di un sottoinsieme $S' \subseteq \{x_1, x_2, \cdots, x_i\}$. Allora $\exists z \in L_i$, al termine dell'iterazione i, tale che*

$$\frac{y}{\left(1 + \frac{\epsilon}{2n}\right)^i} \leq z \leq y.$$

Dimostrazione. La dimostrazione procede per induzione sul numero di iterazioni.

Il caso base è rappresentato da $i = 0$: in questo caso la lista contiene solo il valore nullo, il quale è l'unico ad approssimare se stesso. Ora, suppongo che la proposizione sia corretta fino a $k \leq i - 1$. Considero l'iterazione $k = i$:

1. se $x_i \notin S' \Rightarrow S' \subseteq \{x_1, x_2, \cdots, x_{i-1}\}$ e vale l'ipotesi induttiva, ovvero $\exists z \in L_{i-1}$ tale che
$$\frac{y}{\left(1 + \frac{\epsilon}{2n}\right)^{i-1}} \leq z \leq y$$
   (a) z sopravvive al diradamento dell'iterazione i: in tal caso $z \in L_i$ e
$$\frac{y}{\left(1 + \frac{\epsilon}{2n}\right)^i} < \frac{y}{\left(1 + \frac{\epsilon}{2n}\right)^{i-1}} \leq z \leq y$$
   ed è lo z cercato;

(b) z non sopravvive al diradamento, ovvero $\exists z' \in L_i$ tale che

$$\frac{y}{\left(1+\frac{\epsilon}{2n}\right)\left(1+\frac{\epsilon}{2n}\right)^{i-1}} \leq \frac{z}{\left(1+\frac{\epsilon}{2n}\right)} \leq z' \leq z \leq y.$$

La garanzia di errore si è amplificata ma è rimasta sotto controllo.

2. se $x_i \in S'$, $y = c(S')$, allora $S' = S'' \cup \{x_i\}$, $S'' = \{x_1, \cdots, x_{i-1}\}$ e $c(S'') = y - x_i$. Per ipotesi induttiva $\exists z' \in L_{i-1}$ tale che

$$\frac{y - x_i}{\left(1+\frac{\epsilon}{2n}\right)^{i-1}} \leq z' \leq y - x_i$$

e sicuramente dopo il *merge* otteniamo

$$\frac{y - x_i}{\left(1+\frac{\epsilon}{2n}\right)^{i-1}} + x_i \leq z' + x_i \leq y$$

(a) se $z'+x_i$ non viene eliminato dal diradamento nell'iterazione i, quindi $z' + x_i \in L_i$, si ha

$$\begin{aligned} y \geq z' + x_i &\geq \frac{y - x_i}{\left(1+\frac{\epsilon}{2n}\right)^{i-1}} + x_i \\ &\geq \frac{y - x_i}{\left(1+\frac{\epsilon}{2n}\right)^{i-1}} + \frac{x_i}{\left(1+\frac{\epsilon}{2n}\right)^{i-1}} \\ &= \frac{y}{\left(1+\frac{\epsilon}{2n}\right)^{i-1}} \\ &\geq \frac{y}{\left(1+\frac{\epsilon}{2n}\right)^{i}} \end{aligned}$$

(b) se $z' + x_i$ viene eliminato dal diradamento, ovvero $z' + x_i \notin L_i$, significa che $\exists z'' \in L_i$ tale che

$$\begin{aligned} y \geq z' + x_i &\geq z'' \\ &\geq \frac{z' + x_i}{\left(1+\frac{\epsilon}{2n}\right)} \\ &\geq \frac{y - x_i}{\left(1+\frac{\epsilon}{2n}\right)^{i-1}\left(1+\frac{\epsilon}{2n}\right)} + \frac{x_i}{\left(1+\frac{\epsilon}{2n}\right)} \\ &\geq \frac{y - x_i}{\left(1+\frac{\epsilon}{2n}\right)^{i}} + \frac{x_i}{\left(1+\frac{\epsilon}{2n}\right)^{i}} \\ &= \frac{y}{\left(1+\frac{\epsilon}{2n}\right)^{i}} \end{aligned}$$

□

Il fattore di approssimazione sarà dato dal rapporto tra il costo della soluzione ottima e il costo della soluzione ottenuta dall'algoritmo di approssimazione:

$$\rho = \frac{y^*}{\max(L_n)}.$$

Al termine dell'algoritmo, per il lemma precedente si ha che $\exists z \in L_n$ tale che

$$\frac{y^*}{\left(1+\frac{\epsilon}{2n}\right)^n} \leq z \leq y^*$$

essendo y^* una soluzione ammissibile per S. Abbiamo la seguente situazione:

$$\frac{y^*}{\max(L_n)} \leq \frac{y^*}{z} \leq \frac{y^*}{\frac{y^*}{\left(1+\frac{\epsilon}{2n}\right)^n}} = \left(1+\frac{\epsilon}{2n}\right)^n \overset{?}{\leq} (1+\epsilon)$$

Ricordando che per $|x| < 1$

$$1 + x \leq e^x \leq 1 + x + x^2$$

$$\ln(1+x) \geq \frac{x}{1+x}$$

e che

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

e notando che $\left(1+\frac{\epsilon}{2n}\right)^n$ è crescente, allora $\forall n$ vale

$$\left(1+\frac{\epsilon}{2n}\right)^n \leq \lim_{n\to+\infty} \left(1+\frac{\epsilon/2}{n}\right)^n = e^{\epsilon/2} \leq 1 + \frac{\epsilon}{2} + \left(\frac{\epsilon}{2}\right)^2 \leq 1 + \frac{\epsilon}{2} + \frac{\epsilon}{2} = 1 + \epsilon.$$

confermando che l'algoritmo proposto è uno schema di approssimazione.

Rimane da verificare che l'algoritmo è uno FPTAS, ovvero che esegue in tempo polinomiale sia rispetto alla taglia dell'istanza che rispetto al fattore $1/\epsilon$.

Proposizione. *Le liste L_i sono polinomiali.*

Dimostrazione. Sia $L_i = \langle z_0 = 0, z_1, z_2, \cdots, z_{k-1}\rangle$, $|L_i| = k$. Il diradamento garantisce che $\forall j \geq 1$

$$\frac{z_{j+1}}{z_j} > \left(1+\frac{\epsilon}{2n}\right).$$

Osservo che

$$\frac{z_{k-1}}{z_1} = \frac{z_{k-1}}{z_{k-2}} \cdot \frac{z_{k-2}}{z_{k-3}} \cdots \frac{z_3}{z_2} \cdot \frac{z_2}{z_1} > \left(1+\frac{\epsilon}{2n}\right)^{k-2}$$

e, tenendo conto che $z_{k-1} \leq t$ essendo un costo ammissibile e che $z_1 > 0$:

$$t \geq z_{k-1} > \left(1+\frac{\epsilon}{2n}\right)^{k-2} z_1 \geq \left(1+\frac{\epsilon}{2n}\right)^{k-2};$$

dall'ultima espressione prendo il logaritmo del primo e dell'ultimo termine:

$$\ln t > (k-2)\ln\left(1+\frac{\epsilon}{2n}\right)$$

da cui ricavo k, la lunghezza della lista:

$$\begin{aligned} k &< \frac{\ln t}{\ln\left(1+\frac{\epsilon}{2n}\right)} + 2 \\ &\leq \frac{\ln t}{\frac{\epsilon/2n}{1+\epsilon/2n}} + 2 \\ &= \frac{2n}{\epsilon}\left(1+\frac{\epsilon}{2n}\right)\ln t + 2 \\ &\leq \frac{3n}{\epsilon}\ln t + 2 \end{aligned}$$

dimostrando così che le liste sono polinomiali nella taglia dell'istanza, $k = |L_i| = O\left(\frac{n}{\epsilon} \ln t\right)$. □

Da tutto ciò segue immediatamente che

$$T(n, \epsilon) \in O\left(\sum_{i=1}^{n} |L_i|\right) = O\left(\frac{n^2}{\epsilon} \ln t\right)$$

che è confermato intuitivamente poiché sia l'operazione di *merge* che quella di *trim* sono proporzionali alla lunghezza delle liste coinvolte.

L'ultima analisi ci conferma che l'algoritmo che abbiamo proposto è uno FPTAS:

- fissata la grandezza dell'input, è polinomiale in $1/\epsilon$;
- fissato ϵ, abbiamo che $n = |S| \leq |\langle S, t\rangle|$, inoltre $\ln t \leq |\langle S, t\rangle|$ poiché, a meno di costanti moltiplicative, è il numero di bit necessari per rappresentare t, pertanto è possibile concludere che $n^3 \leq |\langle S, t\rangle|^3$, confermando la polinomialità rispetto alla grandezza dell'input.

Allora si ottiene che

$$T(|\langle S, t\rangle|, \epsilon) \in O\left(\frac{1}{\epsilon} |\langle S, t\rangle|^3\right).$$

Nella pratica è sostanzialmente quadratico poiché il logaritmo del target è trascurabile.

Capitolo 2

Algoritmi per la teoria dei numeri

2.1 Elementi di teoria dei numeri

La teoria dei numeri rappresenta il sostrato teorico per la comprensione dei protocolli di crittografia a chiave pubblica e privata.

Definizione. *Dati $a, d \in \mathbb{Z}$, $d \neq 0$, si dice che d* ***divide*** *a, e si scrive $d|a$ se $\exists k \in \mathbb{Z} \mid a = kd$. Se $d > 0$, d si dice che è un* ***divisore*** *di a.*

Proposizione. *Valgono le seguenti proprietà.*

1. $d|0 \ \forall \ d \in \mathbb{Z} - \{0\}$

   Dimostrazione. Vale
   $$d|0 \Rightarrow \exists k \in \mathbb{Z} \mid 0 = kd$$
   e scegliendo $k = 0$ la proposizione vale per ogni intero d non nullo. □

2. *se $a \neq 0$ e $d|a$ allora $|d| \leq |a|$*

   Dimostrazione. Vale
   $$d|a \Rightarrow \exists k \in \mathbb{Z} \mid a = kd$$
   Se $a > 0$, allora anche $kd > 0$, e questo succede se k e d sono entrambi positivi o entrambi negativi. Nel primo caso, se $k = 1$ allora $a = d$, altrimenti $d < a$ essendo a un multiplo intero di d; nel secondo caso, se $k = -1$ allora $a = d$, altrimenti $d < a$ essendo d negativo e a positivo.

   Di converso, se $a < 0$ allora $kd < 0$, e questo succede se alternativamente k e d sono uno positivo e l'altro negativo. Se $k < 0$ (quindi $d > 0$) allora a è un multiplo intero negativo di d, quindi $|d| \leq |a|$. Lo stesso ragionamento vale scambiando il ruolo di k e d. □

3. *se $(a|b) \ \wedge \ (b|a)$ allora $|a| = |b|$*

Dimostrazione. L'ipotesi è equivalente a

$$\exists k_1, k_2 \mid b = k_1 a \wedge a = k_2 b$$

Sostituendo la seconda espressione nella prima si ottiene che

$$b = k_1 a = k_1 k_2 b$$

e ciò è verificato se e solo se $k_1 k_2 = 1$, da cui segue la tesi. □

Definizione. *Un numero $p \in \mathbb{Z}^+$, $p > 1$, è **primo** se ammette come divisori solamente i numeri 1 e p.*

Teorema (Teorema di divisione)**.** *Per $\forall a \in \mathbb{Z}$, $\forall n \in \mathbb{Z}^+$, $\exists! q, r$, con $q \in \mathbb{Z}$, $r \in \mathbb{Z}^+ \cup \{0\}$, $0 \leq r < n$, tali che $a = qn + r$.*

L'intero q è detto **quoziente** della divisione intera ed è spesso indicato come $q = a \operatorname{div} n = \lfloor a/n \rfloor$, mentre r è detto **resto** della divisione intera ed è indicato come $r = a \bmod n$.

Definizione. *Dato $a, b, d \in \mathbb{Z}$, con $d > 0$, allora d si dice **divisore comune** di a e b se $(d|a) \wedge (d|b)$.*

Proposizione. *Se d è divisore comune di $a, b \in \mathbb{Z}$ allora $\forall x, y \in \mathbb{Z}$ $d|ax + by$.*

Dimostrazione. Se d è divisore comune sia di a che di b, allora significa che $\exists k_1, k_2 \in \mathbb{Z} : (a = k_1 d) \wedge (b = k_2 d)$.

Considerati due interi arbitrari x, y si ha:

$$ax + by = k_1 dx + k_2 dy = (k_1 x + k_2 y)d = kd$$

pertanto, per la definizione di divisore, d divide $ax + by$, e poiché la scelta di x e y è arbitraria, vale $\forall x, y$. □

2.2 Il massimo comun divisore

Definizione. *Siano $a, b \in \mathbb{Z}$ con $|a| + |b| > 0$, allora il **massimo comun divisore** (greatest common divisor) è*

$$\gcd(a, b) = \max\{d > 0 \ : \ (d|a) \ \wedge \ (d|b)\}.$$

L'imposizione sulla somma dei moduli esclude il solo caso in cui entrambi i valori sono nulli: se infatti entrambi fossero interi nulli, l'insieme su cui calcolare il massimo non sarebbe superiormente limitato. Per convenzione si estende la definizione come $\gcd(0, 0) = 0$.

Proposizione. *Valgono le seguenti proprietà:*

1. *Se $|a|, |b| \neq 0$, allora $1 \leq \gcd(a, b) \leq \min\{|a|, |b|\}$*

   Dimostrazione. La minorazione è banale: il massimo comun divisore è un divisore, pertanto deve essere un intero positivo non nullo.

   Per la maggiorazione, ragioniamo per assurdo e assumiamo che il massimo comun divisore, sia d, sia maggiore del più piccolo tra a e b. Assumendo senza perdita di generalità che $|a| \leq |b|$, per la definizione di divisore si ha che $a = k_1 d$ ma, avendo assunto per ipotesi che $d > |a|$, l'uguaglianza non può essere mai verificata, da cui la tesi. □

2. $\gcd(a, b) = \gcd(b, a) = \gcd(-a, b)$

   Dimostrazione. È sufficiente notare che la definizione di massimo comun divisore non pone alcun vincolo sull'ordine o sul segno degli argomenti. □

3. $\gcd(a, 0) = |a|$

   Dimostrazione. Per la definizione di massimo comun divisore, l'insieme è dato dall'intersezione dell'insieme dei divisori di a e di quello dei divisori di 0: poiché il secondo è illimitato (qualunque intero non nullo divide 0), l'intersezione restituisce il solo insieme dei divisori di a, e il massimo di quest'ultimo è proprio $|a|$. □

4. $\gcd(a, ka) = |a|\ \forall k \in \mathbb{Z}$

   Dimostrazione. Il massimo divisore di un qualunque intero $a \neq 0$ è $|a|$. Il secondo argomento è un multiplo intero di a, ovvero $|a| < |ka|$, pertanto la tesi segue dalla prima proprietà. Se $a = 0$, allora la tesi segue direttamente dall'estensione della definizione al caso in cui entrambi gli argomenti sono nulli. □

Teorema (Identità di Bézout)**.** *Dati $a, b \in \mathbb{Z}$ tali per cui $|a| + |b| > 0$, allora* $\gcd(a, b) = \min\{d > 0 \ : \ \exists x, y \in \mathbb{Z} \ : \ d = ax + by\}$.

Dimostrazione. Sia $s = \min\{d > 0 \ : \ \exists x, y \in \mathbb{Z} \ : \ d = ax + by\}$.

Si considerino due interi arbitrari $x, y \in \mathbb{Z}$ e sia $c = ax + by$. Se divido c per s, per il teorema di divisione $\exists! q, r \in \mathbb{Z}$, $0 \leq r < s$, tali che $c = ax + by = qs + r$. Per come è definito s, $\exists \bar{x}, \bar{y}$ tali che $s = a\bar{x} + b\bar{y}$. Allora

$$r = c - qs = ax + by - q(a\bar{x} + b\bar{y}) = a(x - q\bar{x}) + b(y - q\bar{y}).$$

Poiché $0 \leq r < s$ è necessario che $r = 0$: se così non fosse esisterebbe una combinazione lineare intera più piccola di s. Pertanto, essendo il resto nullo, si ha che $s|c$. Quindi

$$s|a \cdot 1 + b \cdot 0$$

$$s|a \cdot 0 + b \cdot 1$$

ovvero $(s|a) \wedge (s|b)$ per cui segue che $s \leq \gcd(a, b)$.

Ora, poiché $(\gcd(a, b)|a) \wedge (\gcd(a, b)|b)$, si ha che $\gcd(a, b)|ax + by\ \forall x, y \in \mathbb{Z}$: scegliendo $x = \bar{x}$ e $y = \bar{y}$ si ottiene che $\gcd(a, b)|a\bar{x} + b\bar{y} = s$ ovvero $\gcd(a, b) \leq s$. Avendo trovato precedentemente che $s \leq \gcd(a, b)$ segue che $s = \gcd(a, b)$. □

2.2.1 L'algoritmo di Euclide

L'algoritmo di Euclide (circa 300 a.C.) si basa su due proprietà.

Proposizione. *Dati due numeri interi $a, b \geq 0$:*

1. *se $b = 0$ allora* $\gcd(a, b) = \gcd(a, 0) = a$*;*
2. *altrimenti,* $\gcd(a, b) = \gcd(b, a \bmod b)$.

Dimostrazione. Se $b = 0$ la dimostrazione segue direttamente dalla definizione di massimo comun divisore.

Ora, assumendo che $b > 0$, definisco $d = \gcd(a, b)$ e $d' = \gcd(b, a \bmod b)$. Si vuole dimostrare che $d = d'$.

Per Bézout, il $\gcd(a, b) = \min\{d > 0 \ : \ \exists x, y \in \mathbb{Z} \ : \ d = ax + by\}$. Inoltre, per il teorema di divisione, $a = qb + r = qb + a \bmod b$, da cui si ottiene che $a \bmod b = a - qb$. A questo punto, è possibile definire $d' = \gcd(b, a - qb)$ e per Bézout $\exists x', y'$ tali che $\gcd(b, a - qb) = bx' + (a - qb)y' = ay' + b(x' - qy')$. Da ciò si ha che $\gcd(a, b) \leq ay' + b(x' - qy') = \gcd(b, a \bmod b)$.

Poiché $d' = \gcd(b, a \bmod b)$, allora $(d'|b) \wedge (d'|a \bmod b)$. Per quanto visto nel paragrafo sopra, per il teorema di divisione si ottiene $a = qb + a \bmod b$ da cui, per la considerazione precedente, $a = qb + a \bmod b = qk_1d' + k_2d' = (qk_1 + k_2)d'$. Avendo ottenuto che a è un multiplo intero di d', è possibile concludere che $(d'|a) \ \wedge \ (d'|b)$, ovvero che $\gcd(a, b) \geq d' = \gcd(b, a \bmod b)$.

Mettendo insieme i risultati ottenuti segue la tesi. □

Algorithm 8 Algoritmo di Euclide

```
function EUCLID(a, b)
    if b = 0 then
        return a
    end if
    return EUCLID(b, a mod b)
end function
```

In 8 è proposto l'algoritmo di Euclide.

Esempio. *Si vuole trovare il massimo comun divisore di* 10 *e* 5. *La sequenza di chiamate dell'algoritmo di Euclide è la seguente:*

$$EUCLID(10, 5) \longrightarrow EUCLID(5, 0) \longrightarrow 5$$

Esempio. *Si vuole trovare il massimo comun divisore di* 100 *e* 99. *La sequenza di chiamate dell'algoritmo di Euclide è la seguente:*

$$EUCLID(100, 99) \longrightarrow EUCLID(99, 1) \longrightarrow EUCLID(1, 0) \longrightarrow 1$$

Il calcolo del resto della divisione è una operazione quadratica nella taglia dell'istanza (numero di bit di a e b). L'algoritmo termina sempre al più dopo b iterazioni poiché il secondo argomento decresce monotonicamente: questo, in realtà, non è molto positivo perché significa che il numero di iterazioni è esponenziale nella rappresentazione in bit. Si vedrà che il numero di iterazioni è molto più basso.

L'algoritmo presenta inoltre un caso particolare quando $0 \leq a < b$: si spreca la prima iterazione poiché la prima chiamata ricorsiva effettua solo lo swap degli argomenti (si passa da EUCLID(a, b) a EUCLID(b, a)).

D'ora in poi analizziamo il caso $a > b > 0$. Infatti, se non ci si trova in questo caso significa che $(a \leq b) \ \vee \ (b = 0)$:

- se $b = 0$ ci si trova nel caso di base, quindi viene effettuata solo la chiamata esterna;

- se $a \leq b$: se $a = b$ vengono effettuate due chiamate, se $a < b$ viene effettuata una chiamata che ripristina la condizione $a > b > 0$.

Per poter effettuare il calcolo del numero di chiamate ricorsive che l'algoritmo effettua è necessario introdurre i numeri di Fibonacci.

Numeri di Fibonacci

Definizione. *Il k-esimo **numero di Fibonacci** è definito ricorsivamente come segue:*

$$F_k = \begin{cases} 1 & k = 1, 2 \\ F_{k-1} + F_{k-2} & k > 2 \end{cases}$$

Esiste inoltre una formula analitica chiusa[1]*:*

$$F_k = \frac{1}{\sqrt{5}} \left(\frac{1+\sqrt{5}}{2} \right)^k - \frac{1}{\sqrt{5}} \left(\frac{1-\sqrt{5}}{2} \right)^k = \left[\frac{1}{\sqrt{5}} \left(\frac{1+\sqrt{5}}{2} \right)^k \right]$$

Il secondo addendo nella formula analitica rappresenta il coefficiente di arrotondamento all'intero più vicino del primo addendo. È possibile quindi fornire la seguente maggiorazione:

$$F_k \leq \frac{1}{2\sqrt{5}} \left(\frac{1+\sqrt{5}}{2} \right)^k$$

Proposizione. *Per $a > b > 0$, se EUCLID(a, b) effettua $k \geq 1$ chiamate (compresa quella esterna), allora*

$$a \geq F_{k+2} \quad e \quad b \geq F_{k+1}.$$

Dimostrazione. La dimostrazione procede per induzione.

Caso di base Se $k = 1$ e $a > b > 0$ allora $b \geq 1 = F_2$ e $a \geq 2 = F_3$.

Ipotesi induttiva Assumo la proposizione vera per un numero di chiamate minore di k.

Tesi Considero k chiamate. La chiamata esterna EUCLID(a, b) comporta la chiamata EUCLID$(b, a \bmod b)$, la quale a sua volta effettuerà $k-1$ iterazioni: posso quindi applicare l'ipotesi induttiva. Per cui

$$b \geq F_{(k-1)+2} = F_{k+1} \text{ e } a \bmod b \geq F_{(k-1)+1} = F_k$$

da cui segue

$$b + a \bmod b = b + (a - qb) = b + \left(a - \left\lfloor \frac{a}{b} \right\rfloor b \right) \leq b + a - b = a$$

ovvero

$$a \geq b + a \bmod b = F_{k+1} + F_k = F_{k+2}.$$

□

[1] La notazione $[x]$ denota la funzione che restituisce il numero intero più vicino a $x \in \mathbb{R}$.

Come detto precedentemente, l'algoritmo effettua al più b iterazioni. È quindi sufficiente trovare il primo valore di k, sia $\bar{k}$, tale per cui $F_{k+1} > b$ per limitare superiormente il numero di chiamate. Quindi

$$b \overset{?}{<} F_{k+1} \leq \frac{1}{2\sqrt{5}} \left(\frac{1+\sqrt{5}}{2} \right)^{k+1}$$

da cui

$$k + 1 > \log_{\frac{1+\sqrt{5}}{2}} 2\sqrt{5}b.$$

Pertanto $\bar{k} = O(\log b)$, lineare nella taglia dell'input.

Combinando il risultato precedente alla quadraticità del calcolo del resto della divisione intera (si veda la sezione 4.2.1), il tempo di esecuzione dell'algoritmo di Euclide è

$$T_{EUCLID}(|<a,b>|) = O(\lceil |<a,b>| \rceil^3).$$

Nella pratica, il caso peggiore si ottiene quando si vuole calcolare il massimo comun divisore di due numeri di Fibonacci consecutivi, mentre per numeri di 1000 bit al massimo vengono eseguite 10 iterazioni. Inoltre, in genere non viene mai implementato l'algoritmo ricorsivo ma una sua versione iterata.

2.2.2 L'algoritmo esteso di Euclide

L'identità di Bézout stabilisce che $\gcd(a,b) = a\bar{x} + b\bar{y}$ per un qualche $\bar{x}, \bar{y} \in \mathbb{Z}$. Si vuole quindi estendere l'algoritmo di Euclide in modo che restituisca non solo il massimo comun divisore ma anche la coppia di valori $(\bar{x}, \bar{y})$.

Caso di base Il $\gcd(a,0) = a = a \cdot 1 + b \cdot 0$, pertanto l'algoritmo restituirà gli elementi $\{a, (1,0)\}$.

Caso ricorsivo Si procede per induzione. Si supponga che EXTENDED_EUCLID$(b, a \bmod b)$ restituisca gli elementi $\{d, (x', y')\}$. Allora $d = bx' + a \bmod by'$, mentre per il teorema di divisione è possibile scrivere $a = \lfloor a/b \rfloor b + a \bmod b$. Mettendo assieme le due espressioni si ottiene

$$d = bx' + (a - \lfloor a/b \rfloor b)y' = ay' + b(x' - \lfloor a/b \rfloor y')$$

da cui segue che $d = \gcd(a,b)$ e l'algoritmo può restituire gli elementi $\{d, (y', x' - \lfloor a/b \rfloor y')\}$.

L'algoritmo è quindi quello descritto in 9.

Algorithm 9 Algoritmo esteso di Euclide

```
function EXTENDED_EUCLID(a, b)
    if b = 0 then
        return {a, (1, 0)}
    end if
    {d, (x', y')} ←EXTENDED_EUCLID(b, a mod b)
    return {d, (y', x' − ⌊a/b⌋y')}
end function
```

Esempio. *Si vuole applicare l'algoritmo esteso di Euclide (abbreviato E_E) alla coppia di numeri* 10 *e* 5*. La sequenza di chiamate è la seguente:*

$$E\_E(10,5) \longrightarrow E\_E(10,5)$$

e i valori di ritorno sono i seguenti (si risale la ricorsione):

$$\{5,(1,0)\} \longrightarrow \{5,(0,1)\}$$

Come verifica, si ottiene:

$$5 = 10 \cdot 0 + 5 \cdot 1$$

Esempio. *Si vuole applicare l'algoritmo esteso di Euclide (abbreviato E_E) alla coppia di numeri* 100 *e* 94*. La sequenza di chiamate è la seguente:*

$$E\_E(100,94) \longrightarrow E\_E(94,6) \longrightarrow E\_E(6,4) \longrightarrow E\_E(4,2) \longrightarrow E\_E(2,0)$$

e i valori di ritorno sono i seguenti (si risale la ricorsione):

$$\{2,(1,0)\} \longrightarrow \{2,(0,1)\} \longrightarrow \{2,(1,-1)\} \longrightarrow \{2,(-1,16)\} \longrightarrow \{2,(16-17)\}$$

Come verifica, si ottiene:

$$2 = 100 \cdot 16 + 94 \cdot (-17)$$

2.3 Aritmetica modulare

La funzione

$$mod: \ \mathbb{Z} \times \mathbb{Z}^+ \longmapsto \mathbb{Z}^+ \cup \{0\}$$

calcola il resto della divisione intera.

Proposizione. *Per* $\forall a, b \in \mathbb{Z}$ *valgono le seguenti proprietà:*

1. $(a+b) \bmod n = (a \bmod n + b \bmod n) \bmod n$.

   Dimostrazione. Per il teorema di divisione si ha

   $$(a+b) = qn + r \Rightarrow r = (a+b) \bmod n$$
   $$a = q'n + r' \Rightarrow r' = a \bmod n$$
   $$b = q''n + r'' \Rightarrow r'' = b \bmod n$$

   Sempre per il teorema di divisione possono scrivere $r' + r'' = \bar{q}n + \bar{r}$, con $\bar{r} = (a \bmod n + b \bmod n) \bmod n$. Lavorando sulla somma si ottiene

   $$r' + r'' = a - q'n + b - q''n = \bar{q}n + \bar{r} \Rightarrow a + b = (q' + q'' + \bar{q})n + \bar{r}$$

   e, per l'unicità di quoziente e resto, deve essere $r = \bar{r}$. □

2. $(ab) \bmod n = ((a \bmod n)(b \bmod n)) \bmod n$.

Dimostrazione. Per il teorema di divisione si ha

$$(ab) = qn + r \Rightarrow r = (ab) \bmod n$$
$$a = q'n + r' \Rightarrow r' = a \bmod n$$
$$b = q''n + r'' \Rightarrow r'' = b \bmod n$$

Sempre per il teorema di divisione possono scrivere $r'r'' = \bar{q}n + \bar{r}$, con $\bar{r} = ((a \bmod n)(b \bmod n)) \bmod n$. Lavorando sul prodotto si ottiene

$$r'r'' = (a - q'n)(b - q''n) = ab - aq''n - bq'n + q'q''n^2 = \bar{q}n + \bar{n}.$$

Sostituendo le espressioni di a e b nel secondo e terzo addendo si ottiene

$$\begin{aligned} ab &= q'q''n^2 + q''nr' + q'q''n^2 + q'nr'' + q'q''n^2 + \bar{q}n + \bar{r} \\ &= (3q'q''n^2 + q'r'' + q''r' + \bar{q})n + \bar{r} \end{aligned}$$

e, per l'unicità di quoziente e resto, $r = \bar{r}$. □

3. $a \bmod n = b \bmod n \Leftrightarrow \exists k \in \mathbb{Z} \ : \ (a - b) = kn.$

Dimostrazione. Per il teorema di divisione si ha

$$a = qn + r$$
$$b = q'n + r'$$

Sottraendo b da a si ottiene

$$a - b = qn + r - (q'n + r') = (q - q') + r - r' = \bar{q}n$$

dove $\bar{q} \in \mathbb{Z}$ e $r - r' = 0$ per ipotesi. □

4. se $m|n$ *allora* $(x \bmod n) \bmod m = x \bmod m.$

Dimostrazione. Per il teorema di divisione, si ha

$$\begin{aligned} x &= qn + r && (r = x \bmod n) \\ r &= q'm + r' && (r' = (x \bmod n) \bmod m) \\ x &= q''m + r'' && (r'' = x \bmod m) \end{aligned}$$

Per dimostrare la proposizione bisogna dimostrare che $r' = r''$. Poiché $m|n$ allora $\exists k \in \mathbb{Z} \mid n = km$, per cui

$$x = qn + r = qkm + q'm + r' = (qk + q')m + r'$$

e, poiché per il teorema di divisione quoziente e resto sono unici, deve essere $q'' = qk + q'$ e $r'' = r'$. □

5. se $a < 0$ *allora* $a \bmod n = (n + a) \bmod n.$

Dimostrazione. Si ha

$$a = qn + r$$
$$n + a = q'n + r'$$

da cui segue $a = (q' - 1)n + r'$, ovvero $q = q' - 1$ e $r = r'$. □

2.3.1 Strutture quozienti

Introduciamo ora la relazione congruenza modulo n.

Definizione. *In aritmetica modulare la relazione* ***congruenza modulo*** n *tra i numeri interi* a, b, n, $n \neq 0$, *è definita come*

$$a \equiv b \bmod n \Leftrightarrow a \bmod n = b \bmod n.$$

La relazione di congruenza modulo n è una relazione di equivalenza su $\mathbb{Z} \times \mathbb{Z}$:

- proprietà riflessiva: $a \equiv a \bmod n \ \forall a \in \mathbb{Z}, \forall n \in \mathbb{Z}^+$.

  Dimostrazione. Per la terza proprietà della funzione mod si ha

  $$a \bmod n = a \bmod n \Leftrightarrow \exists k \in \mathbb{Z} \mid (a - a) = kn.$$

  La tesi segue dal fatto che ogni intero non nullo divide 0. □

- proprietà simmetrica: $a \equiv b \bmod n \ \Rightarrow b \equiv a \bmod n \ \forall a, b \in \mathbb{Z}, \forall n \in \mathbb{Z}^+$.

  Dimostrazione. L'implicazione è equivalente alla seguente implicazione

  $$(a - b) = kn \Rightarrow (b - a) = k'n.$$

  Per giungere alla tesi è sufficiente scegliere $k' = k$, ovvero se n divide $(a - b)$ allora divide anche l'opposto. □

- proprietà transitiva: $a \equiv b \bmod n \wedge b \equiv c \bmod n \ \Rightarrow a \equiv c \bmod n \ \forall a, b, c \in \mathbb{Z}, \forall n \in \mathbb{Z}^+$.

  Dimostrazione. La proprietà è equivalente a

  $$((a - b) = kn) \ \wedge \ ((b - c) = k'n) \Rightarrow (a - c) = k''n.$$

  Ricavando b dalla seconda espressione e sostituendola nella prima si ottiene

  $$a - c = kn - k'n = (k - k')n$$

  e scegliendo $k'' = k - k'$ si ottiene la tesi. □

La relazione congruenza modulo n partiziona il suo supporto, $\mathbb{Z}$, in partizioni di equivalenza: in particolare, individua n **classi di equivalenza**, essendo n il numero di resti distinti. Per un dato $a \in \mathbb{Z}$, la classe di equivalenza si indica come

$$[a]_n = \{x \in \mathbb{Z} \ : \ x \bmod n = a \bmod n\}.$$

Si noti che le classi di equivalenza $0, 1, 2, \cdots, n - 1$ sono in corrispondenza biunivoca con i primi n numeri naturali, detti **rappresentanti principali** delle classi, poiché ognuno di essi appartiene ad una classe differente. In particolare, $\forall r \in \mathbb{Z} \ : \ 0 \leq r < n$ è possibile esprimere la classe di equivalenza tramite una progressione aritmetica:

$$[r]_n = \{r + kn \ : \ k \in \mathbb{Z}\}.$$

Definizione. *Si definisce* ***struttura quoziente*** *l'insieme di insiemi*

$$\mathbb{Z}_{\mathrm{n}} = \{[r]_n \ : \ 0 \leq r < n\}.$$

Passiamo ora a definire le operazioni di somma e prodotto.

Definizione. *La somma in* $\mathbb{Z}_{\mathrm{n}}$ *è definita come* $[a]_n + [b]_n = [a+b]_n$.

Proposizione. *La somma è ben definita.*

Dimostrazione. Perché l'operazione sia ben definita è necessario che il risultato della somma sia indipendente dai rappresentanti delle classi. È necessario dimostrare che $\forall k_1, k_2 \in \mathbb{Z}$

$$[a + k_1 n]_n + [b + k_2 n]_n = [a+b]_n.$$

Si ha

$$\begin{aligned}(a + k_1 n + b + k_2 n) \bmod n &= (a + b + (k_1 + k_2)n) \bmod n \\ &= (a+b) \bmod n \in [a+b]_n\end{aligned}$$

per ogni $k_1, k_2 \in \mathbb{Z}$. □

Definizione. *Il prodotto in* $\mathbb{Z}_{\mathrm{n}}$ *è definito come* $[a]_n \cdot [b]_n = [ab]_n$.

Proposizione. *Il prodotto è ben definito.*

Dimostrazione. Perché l'operazione sia ben definita è necessario che il risultato del prodotto sia indipendente dai rappresentanti delle classi. È necessario dimostrare che $\forall k_1, k_2 \in \mathbb{Z}$

$$[a + k_1 n]_n \cdot [b + k_2 n]_n = [ab]_n.$$

Si ha

$$\begin{aligned}((a + k_1 n)(b + k_2 n)) \bmod n &= (ab + ak_2 n + bk_1 n + k_1 k_2 n^2) \bmod n \\ &= (ab + (ak_2 + bk_1 + k_1 k_2 n)n) \bmod n \\ &= (ab) \bmod n \in [ab]_n\end{aligned}$$

per ogni $k_1, k_2 \in \mathbb{Z}$. □

Utilizzando i soli rappresentanti principali di una classe, $a, b \in \mathbb{Z}_{\mathrm{n}}$, è possibile denotare la somma e il prodotto con l'usuale notazione ma con i seguenti significati:

$$\begin{aligned}a + b &= (a+b) \bmod n \\ ab &= (ab) \bmod n\end{aligned}$$

Proposizione. *L'operazione di somma sopra definita rende* $(\mathbb{Z}_{\mathrm{n}}, +)$ *un gruppo (additivo).*

Dimostrazione. Per dimostrare che la struttura algebrica $(\mathbb{Z}_{\mathrm{n}}, +)$ è un gruppo è necessario dimostrare le seguenti quattro proprietà:

1. **Proprietà di chiusura.** Per ogni $a, b \in \mathbb{Z}_n$, si ha $a+b = (a+b) \bmod n \in \mathbb{Z}_n$, pertanto l'operazione di somma è chiusa rispetto a $\mathbb{Z}_n$.

2. **Proprietà associativa.** La proprietà associativa viene ereditata dalla somma tra interi. Infatti, dati tre rappresentanti principali $a, b, c \in \mathbb{Z}_n$, si ha:

$$(a+b)+c = ((a+b)+c) \bmod n = (a+(b+c)) \bmod n = a+(b+c)$$

3. **Esistenza dell'elemento neutro.** L'elemento neutro è rappresentato dalla classe $[0]_n$, infatti, $\forall [a]_n \in \mathbb{Z}_n$,

$$[0]_n + [a]_n = [0+a]_n = [a]_n \quad \text{e} \quad [a]_n + [0]_n = [a+0]_n = [a]_n$$

4. **Esistenza dell'inverso (opposto).** Per ogni $a \in \mathbb{Z}_n$, il suo inverso è $(-a) \in \mathbb{Z}_n$. Infatti

$$a + (-a) = (a + (-a)) \bmod n = (a-a) \bmod n = 0$$
$$(-a) + a = ((-a) + a) \bmod n = (-a+a) \bmod n = 0$$

□

Proposizione. *L'operazione di prodotto sopra definita non rende* $(\mathbb{Z}_n, \cdot)$ *un gruppo.*

Dimostrazione. La proprietà di gruppo che non viene verificata è quella dell'esistenza dell'inverso per ogni elemento di $\mathbb{Z}_n$: in particolare, non esiste l'inverso per la classe $[0]_n$.

È possibile dimostrare che l'elemento neutro è $[1]_n$, infatti, per ogni $[a]_n \in \mathbb{Z}_n$:

$$[a]_n \cdot [1]_n = [a \cdot 1]_n = [a]_n \quad \text{e} \quad [1]_n \cdot [a]_n = [1 \cdot a]_n = [a]_n$$

Calcolando l'elemento neutro di $[0]_n$ si ottiene

$$[0]_n \cdot [b]_n = [0 \cdot b]_n \neq [1]_n \quad \text{e} \quad [b]_n \cdot [0]_n = [b \cdot 0]_n \neq [1]_n$$

per ogni $[b]_n \in \mathbb{Z}_n$. □

Per quanto visto sopra, eliminiamo da $\mathbb{Z}_n$ la classe $[0]_n$. Anche in questo caso l'esistenza dell'inverso è problematica.

Proposizione. *L'operazione di prodotto sopra definita non rende* $(\mathbb{Z}_n - \{[0]_n\}, \cdot)$ *un gruppo.*

Dimostrazione. L'elemento neutro rimane $[1]_n$ (vedi dimostrazione precedente). Dimostriamo ora la mancanza dell'inverso per ogni elemento di $\mathbb{Z}_n$ con un controesempio. Si consideri $\mathbb{Z}_4$:

$$[2]_4[1]_4 = [2]_4$$
$$[2]_4[3]_4 = [6]_4 = [2]_4$$
$$[2]_4[2]_4 = [4]_4 = [0]_4 \notin (\mathbb{Z}_n - \{[0]_n\})$$

da cui segue che in $\mathbb{Z}_4$ non esiste l'inverso. Si noti che dall'ultima espressione segue inoltre che in $\mathbb{Z}_n - \{[0]_n\}$ il prodotto non è chiuso. □

Definizione. *Sia* $\mathbb{Z}_n^* = \{a \in \mathbb{Z}_n \mid \gcd(a, n) = 1\}$.

Si osservi che se a e n sono primi tra loro, allora la classe di equivalenza $[0]_n$ **non** appartiene a $\mathbb{Z}_n^*$. È una definizione sensata? Dimostriamolo.

Dimostrazione. È necessario dimostrare che per ogni $k \in \mathbb{Z}$ si ha

$$\gcd(a, n) = 1 \Rightarrow \gcd(a + kn, n) = 1.$$

Infatti, se $\gcd(a, n) = 1$, allora, $\forall x, y \in \mathbb{Z}$,

$$1 = ax + ny = ax + knx - knx + ny = (a + kn)x + n(y - kn)$$

da cui segue che $\gcd(a + kn, n) = 1$, $\forall k \in \mathbb{Z}$. □

L'insieme $\mathbb{Z}_n^*$ raccoglie tutte le classi di $\mathbb{Z}_n$ per le quali il loro rappresentante è un numero primo rispetto a n.

Proposizione. *Il prodotto sopra definito rende* $(\mathbb{Z}_n^*, \cdot)$ *un gruppo (moltiplicativo).*

Dimostrazione. Dimostriamo le quattro proprietà che rendono una struttura algebrica un gruppo.

1. **Proprietà di chiusura.** Siano $a, b \in \mathbb{Z}_n^*$: bisogna dimostrare che

   $$(\gcd(a, n) = 1) \wedge (\gcd(b, n) = 1) \Rightarrow \gcd((ab) \bmod n, n) = 1.$$

   Per l'identità di Bézout si ha

   $$1 = ax_1 + ny_1 \quad \text{e} \quad 1 = bx_2 + ny_2$$

   e moltiplicando i due membri tra loro si ottiene

   $$1 = ab(x_1x_2) + n(ax_1y_2 + bx_2y_1 + y_1y_2n)$$

   da cui $\gcd(ab, n) = 1 \Rightarrow \gcd((ab) \bmod n, n) = 1$.

2. **Proprietà associativa.** Per ogni $a, b, c \in \mathbb{Z}_n^*$ si ha

   $$(ab)c = ((ab)c) \bmod n = (a(bc)) \bmod n = a(bc).$$

3. **Esistenza dell'elemento neutro.** L'elemento neutro è $[1]_n$, infatti, per ogni $[a]_n \in \mathbb{Z}_n^*$:

   $$[a]_n \cdot [1]_n = [a \cdot 1]_n = [a]_n \quad \text{e} \quad [1]_n \cdot [a]_n = [1 \cdot a]_n = [a]_n.$$

4. **Esistenza dell'inverso**[2]**.** Bisogna dimostrare che

   $$\forall a \in \mathbb{Z}_n^* \ \exists a^{-1} \mid aa^{-1} = a^{-1}a = 1.$$

[2]Data una classe $[a]_n \in \mathbb{Z}_n$, si denota il suo inverso come $[a]_n^{-1}$, o, lasciando sottintesa la notazione di classe, a^{-1}.

Poiché $a \in \mathbb{Z}_n^*$, si ha che

$$\begin{aligned}
\gcd(a,n) = 1 &\Rightarrow 1 = ax + ny \\
&\Rightarrow ax = 1 - ny \\
&\Rightarrow (ax) \bmod n = 1 \qquad \text{(teorema di divisione)} \\
&\Rightarrow ((a \bmod n)(x \bmod n)) \bmod n = 1 \\
&\Rightarrow a^{-1} = x \bmod n.
\end{aligned}$$

Rimane da dimostrare se $a^{-1} \in \mathbb{Z}_n^*$:

$$\begin{aligned}
1 = xa + ny &\Rightarrow \gcd(x,n) = 1 \\
&\Rightarrow \gcd(x \bmod n, n) = 1 \\
&\Rightarrow x \bmod n \in \mathbb{Z}_n^*
\end{aligned}$$

concludendo così la dimostrazione.

□

Definizione. *Si definisce come segue la* ***funzione di Eulero****:*

$$\phi(n) = |\,\mathbb{Z}_n^*\,| = |\{a \ : \ 1 \le a \le n-1 \mid \ \gcd(a,n) = 1\}|.$$

La sua descrizione analitica compatta è la seguente:

$$\phi(n) = n \prod_{\substack{p|n \\ p \text{ primo}}} \left(1 - \frac{1}{p}\right).$$

Dalla definizione è possibile osservare che

- se n è primo, allora $\forall x \ : \ 1 \le x \le n-1$, si ha $\gcd(x,n) = 1$, da cui si ricava facilmente che $|\,\mathbb{Z}_n^*\,| = n - 1 = \phi(n)$;
- se $n = pq$ con p e q numeri primi molto grandi, allora

$$\phi(n) = pq\left(1 - \frac{1}{p}\right)\left(1 - \frac{1}{q}\right) = (p-1)(q-1).$$

Teorema (Teorema di Eulero)**.** *Per ogni $a \in \mathbb{Z}_n^*$ si ha $a^{\phi(n)} \bmod n = 1$.*

Corollario (Piccolo teorema di Fermat)**.** *Se $n \in \mathbb{Z}$ è primo, allora $\forall a \in \mathbb{Z} \ : \ 1 \le a < n$*

$$a^{n-1} \equiv 1 \bmod n.$$

2.4 Il teorema cinese del resto

Il teorema cinese del resto fornisce una corrispondenza tra una sistema di equazioni modulo un insieme di numeri primi tra loro a coppie e un'equazione modulo il loro prodotto.

Il teorema cinese del resto ha due applicazioni importanti. Consideriamo l'intero n fattorizzato come $n = n_1 n_2 \cdots n_k$, dove i fattori n_i sono primi tra

loro a coppie. Come prima cosa, il teorema descrive la struttura di $\mathbb{Z}_n$ come identica alla struttura di $\mathbb{Z}_{n_1} \times \mathbb{Z}_{n_2} \times \cdots \times \mathbb{Z}_{n_k}$ come addizioni e moltiplicazioni modulo n_i componente per componente nella componente i. Inoltre, il poter lavorare in $\mathbb{Z}_{n_i}$ è più efficiente che lavorare nell'intera struttura $\mathbb{Z}_n$ potendo quindi progettare algoritmi più efficienti in termini di operazioni sui bit.

Teorema (Il teorema cinese del resto)**.** *Sia $n = n_1 n_2 \cdots n_k$, dove i fattori n_i sono primi tra loro a coppie. Si consideri la corrispondenza*

$$a \leftrightarrow (a_1, a_2, \cdots, a_k)$$

dove $a \in \mathbb{Z}_n$, $a_i \in \mathbb{Z}_{n_i}$ e $a_i = a \bmod n_i$ $i = 1, 2, \cdots, k$. Allora la corrispondenza di cui sopra è una mappa uno-a-uno (biunivoca) tra $\mathbb{Z}_n$ e $\mathbb{Z}_{n_1} \times \mathbb{Z}_{n_2} \times \cdots \times \mathbb{Z}_{n_k}$. Le operazioni eseguite su $\mathbb{Z}_n$ possono essere equivalentemente eseguite sulla corrispondente k-esima tupla eseguendo le operazioni indipendentemente in ogni coordinata nel sistema appropriato. Cioè, se

$$a \leftrightarrow (a_1, a_2, \cdots, a_k)$$
$$b \leftrightarrow (b_1, b_2, \cdots, b_k)$$

allora

$$(a + b) \bmod n \leftrightarrow ((a_1 + b_1) \bmod n_1, (a_2 + b_2) \bmod n_2, \cdots, (a_k + b_k) \bmod n_k)$$
$$(a - b) \bmod n \leftrightarrow ((a_1 - b_1) \bmod n_1, (a_2 - b_2) \bmod n_2, \cdots, (a_k - b_k) \bmod n_k)$$
$$(ab) \bmod n \leftrightarrow ((a_1 b_1) \bmod n_1, (a_2 b_2) \bmod n_2, \cdots, (a_k b_k) \bmod n_k)$$

Dimostrazione. La trasformazione tra le due rappresentazioni è abbastanza immediata. Passare da a a $(a_1, a_2, \cdots, a_k)$ è facile e richiede solo k operazioni modulo.

Calcolare a da $(a_1, a_2, \cdots, a_k)$ è un po' più complicato. Definiamo $m_i = n/n_i$ per $i = 1, 2, \cdots, k$; quindi m_i è il prodotto di tutti gli n_j diversi da n_i: $m_i = n_1 n_2 \cdots n_{i-1} n_{i+1} \cdots n_k$. Definiamo

$$c_i = m_i(m_i^{-1} \bmod n_i)$$

per $i = 1, 2, \cdots, k$. Questa equazione è sempre ben definita: poiché m_i e n_i sono numeri primi tra loro, allora è garantita l'esistenza di $m_i^{-1} \bmod n_i$. Così è possibile calcolare a in funzione degli $a_1, a_2, \cdots, a_k$ come segue:

$$a \equiv (a_1 c_1 + a_2 c_2 + \cdots + a_k c_k) \bmod n.$$

Ora dimostriamo che tale equazione assicura che $a \equiv a_i \bmod n_i$ per $i = 1, 2, \cdots, k$. Si noti che se $j \neq i$, allora $m_j \equiv 0 \bmod n_i$, implicando che $c_j \equiv m_j \equiv 0 \bmod n_i$. Si noti inoltre che $c_i \equiv 1 \bmod n_i$. Otteniamo quindi la seguente corrispondenza:

$$c_i \leftrightarrow (0, 0, \cdots, 0, 1, 0, \cdots, 0),$$

un vettore di 0 eccetto che nell'i-esima componente, dove c'è un 1. Pertanto per ogni i, si ha

$$\begin{aligned} a &\equiv a_i c_i \bmod n_i \\ &\equiv a_i m_i (m_i^{-1} \bmod n_i) \bmod n_i \\ &\equiv a_i \bmod n_i \end{aligned}$$

ovvero il metodo di calcolo descritto produce come risultato a che soddisfa i vincoli $a \equiv a_i \bmod n_i$ per $i = 1, 2, \cdots, k$. La corrispondenza è biunivoca potendo effettuare la trasformazione in entrambi i versi. □

Si hanno i seguenti corollari.

Corollario. *Se $n_1, n_2, \cdots, n_k$ sono numeri primi tra loro a coppie e $n = n_1 n_2 \cdots n_k$, allora per ogni intero $a_1, a_2, \cdots, a_k$, il sistema di equazioni*

$$x \equiv a_i \bmod n_i,$$

per $i = 1, 2, \cdots, k$, ha un'unica soluzione modulo n per l'incognita x.

Corollario. *Se $n_1, n_2, \cdots, n_k$ sono numeri primi tra loro a coppie e $n = n_1 n_2 \cdots n_k$, allora per ogni intero a e x,*

$$x \equiv a \bmod n_i$$

per $i = 1, 2, \cdots, k$ se e solo se

$$x \equiv a \bmod n.$$

Esempio. *Sia trovare il valore dell'incognita x del seguente sistema di equazioni:*

$$x \equiv 2 \bmod 5$$
$$x \equiv 3 \bmod 13$$

Si vuole trovare il valore di x, che sappiamo essere unico per il secondo corollario. Per risolvere questo esercizio si applica la stessa procedura applicata per la dimostrazione del teorema cinese del resto.

Dal sistema si identificano i seguenti valori: $a_1 = 2$, $a_2 = 3$, $n_1 = 5$, $n_2 = 13$, con n_1 e n_2 primi tra loro, e $n = n_1 n_2 = 65$. Allora, $m_1 = n/n_1 = 13$ e $m_2 = n/n_2 = 5$. È necessario calcolare i valori $m_i^{-1} \bmod n_i$, $i = 1, 2$: per far ciò è necessario applicare l'algoritmo esteso di Euclide[3] ai valori m_i e n_i. Facendo ciò si ottiene che $13^{-1} 1 \equiv 2 \bmod 5$ e $5^{-1} 1 \equiv -5 \bmod 13 \equiv 8 \bmod 13$. A seguire si calcolano i valori $c_i = m_i(m_i^{-1} \bmod n_i)$:

$$c_1 = 13(2 \bmod 5) = 26$$
$$c_2 = 5(8 \bmod 13) = 40$$

da cui segue

$$x \equiv (a_1 c_1 + a_2 c_2) \bmod n \equiv 42 \bmod 65.$$

2.5 Crittografia e RSA

2.5.1 Introduzione

L'utente X genera due chiavi: una chiave pubblica P_X e una chiave privata (segreta) S_X. Queste due chiavi vengono utilizzate per criptare un messaggio m. La chiave pubblica è una chiave che deve essere condivisa (conosciuta) tra i due interlocutori mentre la chiave privata deve rimanere segreta. Si osservi che:

[3] Si veda la dimostrazione dell'esistenza dell'inverso in $\mathbb{Z}_n^*$.

- $P_X(m)$ e $S_X(m)$ devono essere calcolabili efficientemente;
- $S_X(m) = P_X^{-1}(m)$, ovvero $P_X(S_X(m)) = S_X(P_X(m)) = m$;
- nota P_X, la funzione S_X non può essere calcolata efficientemente in assenza di altre informazioni.

Dove si usa RSA? Principalmente in protocolli di comunicazione e di autenticazione (che possono anche essere combinati tra loro).

Protocolli di comunicazione Si supponga che Bob voglia inviare un messaggio M ad Alice attraverso un canale non sicuro, ovvero un canale dove può essere in ascolto Charlie, e che questo messaggio debba essere conosciuto solo da Alice. Bob, anziché inviare sul canale il messaggio in chiaro, invia la sua versione criptata $y = P_A(M)$ con la chiave **pubblica** di Alice: alla ricezione, Alice può decrittare con successo il messaggio inviato da Bob tramite la propria chiave privata, ovvero $M = S_A(y)$.

Protocolli di autenticazione Si supponga che Alice voglia autenticarsi nei confronti di Bob e quest'ultimo vuole essere certo che la persona che si vuole autenticare sia proprio Alice e non Charlie (ancora una volta, il canale di comunicazione è da assumere come non sicuro). Per raggiungere questo scopo, Alice invia un messaggio $\langle M, S_A(M)\rangle$ a Bob: quest'ultimo può assicurarsi che esso sia stato inviato proprio da Alice se $M = P_A(S_A(M))$.

La forza di RSA La forza di RSA sta nel fatto che le chiavi private generate non sono calcolabili efficientemente a partire da quelle pubbliche senza avere ulteriori informazioni. In particolare, il tutto si basa sulla complessità dei problemi PRIMALITY e FACTORING.

Il problema (decisionale) PRIMALITY è definito come segue:

$$\begin{cases} I = \langle n\rangle \mid n \in \mathbb{N} \\ D = \text{"}n \text{ è primo?"} \end{cases}$$

Nel 2004 è stato dimostrato che tale problema appartiene alla classe P, tuttavia l'algoritmo migliore al momento disponibile (algoritmo dei tre indiani) è di fatto impraticabile.

Il problema (decisionale) FACTORING è definito come segue:

$$\begin{cases} I = \langle n, l, u\rangle \\ D = \exists p, q \in \mathbb{N} \ : \ (n = pq) \ \wedge \ (l \leq p \leq u) \end{cases}$$

dove l e u sono rispettivamente un *lower bound* e un *upper bound*. È noto che tale problema è un problema in NP tuttavia non è ancora dimostrato che tale problema appartenga a NPC (si ritiene che non vi appartenga). Il miglior algoritmo per FACTORING ha complessità $T(n) = \Theta(e^{\sqrt[3]{\frac{64}{9}|n|(\log_2 n)^2}})$, impossibile da utilizzare in pratica. Anche entrando nel campo della quantistica, il miglior algoritmo ha complessità cubica (algoritmo di Shor).

2.5.2 Calcolo delle chiavi

Ogni partecipante genera casualmente due **grandi** numeri primi p e q (con $p \neq q$): per far ciò, si estraggono a caso due numeri grandi e si verifica se essi sono primi o meno. Il numero di bit utilizzati dipende dalla tecnologia: attualmente si utilizzano circa 512 bit, ottenendo quindi $n = pq$ a 1024 bit. In seguito il dominio di lavoro sarà $\mathbb{Z}_n$. Viene inoltre calcolata la funzione di Eulero $\phi(n) = (p-1)(q-1)$. A questo punto è possibile procedere con il calcolo delle chiavi:

- **Chiave pubblica** Si seleziona un $e \in \mathbb{Z}$ **piccolo** (scelto casualmente su pochi bit) primo con $\phi(n)$. La chiave pubblica sarà $P_X = (e, n)$: un messaggio M viene cifrato come $P_X(M) = M^e \bmod n$;
- **Chiave privata** Si prende in considerazione l'intero e scelto per la chiave pubblica: allora, poiché $e \in \mathbb{Z}^*_{\phi(n)}$, esiste un d tale per cui $d \equiv e^{-1} \bmod \phi(n)$, e tale valore è facilmente calcolabile tramite EXTENDED_EUCLID$(e, \phi(n))$. La chiave segreta sarà $S_X = (d, n)$ e un messaggio M verrà cifrato come $S_X(M) = M^d \bmod n$.

Tali funzioni sono calcolabili efficientemente? Si ha

$$\begin{aligned} M^e \bmod n &= (M \cdot M \cdots M) \bmod n \\ &= (((MM) \bmod n)M \bmod n) \cdots) \bmod n \end{aligned}$$

L'algoritmo 10 effettua tale calcolo. Considerando il numero di prodotti, tale algoritmo ha complessità

$$T(e) = T\left(\lfloor e/2 \rfloor\right) + 2 = \Theta(\log e)$$

per il *master theorem*. Come è noto da *Dati e algoritmi 2*, l'algoritmo di *squaring* è più efficiente.

Algorithm 10 Elevazione a potenza modulo n

```
function POWER(M, e, n)
    if e = 0 then
        return 1
    end if
    if e = 1 then
        return M
    end if
    if EVEN(e) then
        temp ← POWER(M, e/2, n)
        return (temp * temp) mod n
    else
        temp ← POWER(M, (e − 1)/2, n)
        return (temp * temp * M) mod n
    end if
end function
```

Proposizione. *Le chiavi generate secondo la ricetta descritta sopra sono tali per cui*

$$P_X(S_X(M)) = S_X(P_X(M)) = M$$

per ogni $M \in \mathbb{Z}_n$.

Dimostrazione. Si ha

$$\begin{aligned} P_X(S_X(M)) &= (M^d \bmod n)^e \bmod n \\ &= (\underbrace{(M^d \bmod n)(M^d \bmod n)\cdots(M^d \bmod n)}_{e \text{ volte}}) \bmod n \\ &= M^{ed} \bmod n \\ &= S_X(P_X(M)) \end{aligned}$$

pertanto la dimostrazione verte nel dimostrare la relazione

$$M^{ed} \bmod n \stackrel{?}{=} M = M \bmod n$$

Essendo $n = pq$ con $\gcd(p, q) = 1$, possiamo applicare il secondo corollario del teorema cinese del resto e quindi è sufficiente provare che

$$\begin{cases} M^{ed} \bmod p = M \bmod p \\ M^{ed} \bmod q = M \bmod q \end{cases}$$

Si ha che $d = e^{-1} \bmod \phi(n)$, ovvero $ed \equiv 1 \bmod \phi(n)$ (inverso moltiplicativo): questo significa che $\exists h \in \mathbb{Z}^+$ tale per cui

$$ed = 1 + h\phi(n) = 1 + h(p-1)(q-1).$$

Segue che

$$M^{ed} \bmod p = M^{1+h(p-1)(q-1)} \bmod p$$

da cui

1. se $M \bmod p = 0$ allora $M^{ed} \bmod p = 0$ e quindi segue immediatamente che $M^{ed} \equiv M \bmod p$;

2. se $M \bmod p \neq 0$ allora

$$\begin{aligned} M^{ed} \bmod p &= M^{1+h(p-1)(q-1)} \bmod p \\ &= (M(M^{h(q-1)})^{p-1}) \bmod p \\ &= (M(M^{p-1})^{h(q-1)}) \bmod p \\ &= ((M \bmod p)((1)^{h(q-1)} \bmod p)) \bmod p \\ &= M \bmod p \end{aligned}$$

   applicando il piccolo teorema di Fermat e considerando che $M^{p-1} \bmod p = 1 = 1 \bmod p$ e che, poiché $M \in \mathbb{Z}_n$ e $n > p$, $M \bmod p = p$.

Si conclude così che $M^{ed} \bmod p = M \bmod p$.

Applicando la stessa identica procedura alla seconda relazione si giunge alla tesi. □

2.5.3 Attacchi ad RSA

Si noti quanto sia importante la fattorizzazione in RSA: se la fattorizzazione fosse semplice RSA sarebbe assolutamente insicuro.

Introduciamo ora il problema del **logaritmo discreto**: dati $a, b \in \mathbb{Z}_n^*$ si vuole determinare un k tale che $a^k \equiv b \bmod n$. Tale problema si applica al fatto che, data $P_X(M) = M^e \bmod n$, con M un messaggio qualunque, determinando x tale per cui $(M^e)^x \equiv M \bmod n$ si è trovato il valore d utile per generare la chiave privata. Con abuso di notazione possiamo affermare che il problema della fattorizzazione è riducibile polinomialmente al problema del logaritmo discreto.

Come visto nel paragrafo precedente, dalla tripla (n, p, q) siamo in grado di calcolare la coppia $(n, \phi(n))$ utile per la generazione delle chiavi. È possibile effettuare il contrario? Ovvero, a partire da $\phi(n)$, si riesce a fattorizzare n?

Dimostrazione. Ipotesi: $n = pq$, p e q primi e ignoti, $\phi(n)$ noto. Si ha

$$\phi(n) = (p-1)(q-1) = pq - (p+q) + 1 \Rightarrow p + q = n - \phi(n) + 1.$$

Inoltre

$$\begin{aligned}(p-q)^2 &= p^2 + q^2 - 2pq \\ &= p^2 + q^2 - 2pq + 2pq - 2pq \\ &= p^2 + q^2 + 2pq - 4pq \\ &= (p+q)^2 - 4pq \\ &= (n - \phi(n) + 1)^2 - 4n\end{aligned}$$

da cui $p - q = \sqrt{(n - \phi(n) + 1)^2 - 4n}$ (la radice è calcolabile facilmente).

Allora

$$\begin{cases} p + q = n - \phi(n) + 1 \\ p - q = \sqrt{(n - \phi(n) + 1)^2 - 4n} \end{cases} \Leftrightarrow \begin{cases} p = \frac{n - \phi(n) + 1 + \sqrt{(n - \phi(n) + 1)^2 - 4n}}{2} \\ q = \frac{n - \phi(n) + 1 - \sqrt{(n - \phi(n) + 1)^2 - 4n}}{2} \end{cases}$$

□

Analizziamo ora alcune tecniche di attacco a RSA.

1. **Decomposizione di $n = pq$ se p e q sono molto vicini tra loro**

   Sia $p > q$ e $q = p - k$ con k "piccolo". Allora $n = pq = p(p-k) = p^2 - kp$ che è equivalente all'equazione $p^2 - kp - n = 0$: essendo k "piccolo" posso permettermi un algoritmo esaustivo che risolve l'equazione provando ogni k, fermandosi appena le soluzioni p_1 e p_2 sono intere. L'algoritmo esegue in un tempo $T \sim c(p - q)$, quindi è necessario che la differenza $p - q$ sia sufficiente grande da rendere impraticabile questa soluzione.

2. **Attacchi per scelte di chiavi pubbliche con lo stesso esponente**

   In genere l'intero e viene scelto di pochi bit. Sia e "molto piccolo", siano presenti nel sistema $1 \leq i \leq e$ utenti e sia (e, n_i) la coppia di dati da cui generare le chiavi pubbliche e private per ogni utente. Si intercettino e messaggi

   $$M^e \bmod n_1, M^e \bmod n_2, \cdots, M^e \bmod n_e$$

assumendo che $n_i \neq n_j$ per $\forall i \neq j$. È inoltre ragionevole assumere che $\gcd(n_i, n_j) = 1 \ \forall i \neq j$ poiché la probabilità di scegliere due numeri primi che abbiano un fattore in comune è praticamente nulla, pertanto $n = \prod_{i=1}^{e} n_i$. I messaggi intercettati formano le coordinate di un punto in $\mathbb{Z}_\text{n}$. Ragionando con il teorema cinese del resto, si ha

$$f(M^e \bmod n) = (M^e \bmod n_1, M^e \bmod n_2, \cdots, M^e \bmod n_e)$$

ovvero

$$M^e \bmod n = f^{-1}(M^e \bmod n_1, M^e \bmod n_2, \cdots, M^e \bmod n_e)$$

Posso quindi ricostruire il valore $M^e \bmod n$ efficientemente. Si consideri un messaggio M che viene criptato rispetto a $n_1, n_2, \cdots, n_e$. Allora

$$M \in \begin{cases} \mathbb{Z}_{n_1} \\ \mathbb{Z}_{n_2} \\ \cdots \\ \mathbb{Z}_{n_e} \end{cases} \Rightarrow M < \begin{cases} n_1 \\ n_2 \\ \cdots \\ n_e \end{cases} \Rightarrow M^e < n_1 n_2 \cdots n_e = n$$

poiché M è un rappresentante principale dei domini dei messaggi. Da ciò posso ricostruire direttamente il valore *non modulare* M^e, allora è sufficiente calcolare $\sqrt[e]{M^e}$ in $\mathbb{Z}$ (calcolabile polinomialmente nel numero di bit che compongono M).

3. **Attacchi per scelte di chiavi pubbliche con lo stesso modulo**

   Consideriamo ora il caso in cui due utenti condividano lo stesso modulo n ma usino esponenti e_1, e_2 differenti, con la particolarità che $\gcd(e_1, e_2) = 1$. Anche in questo caso un messaggio di broadcast M potrebbe essere catturato da un malintenzionato decrittando i messaggi cifrati $C_1 = M^{e_1} \bmod n$ e $C_2 = M^{e_2} \bmod n$. Poiché e_1 e e_2 sono primi tra loro, posso scrivere l'identità di Bézout

   $$1 = e_1 r + e_2 s \quad r, s \in \mathbb{Z}$$

   con $r < 0$ (siccome sia $e_1, e_2 > 0$, allora uno tra r e s deve essere negativo). Allora è possibile esprimere il messaggio originario come

   $$\begin{aligned} M &= M^1 \\ &= M^{e_1 r + e_2 s} \\ &= (C_1^r C_2^s) \bmod n \\ &= ((C_1^{-1})^{-r} C_2^s) \bmod n \end{aligned}$$

   Ciò è lecito se C_1 ammette inverso moltiplicativo in $\mathbb{Z}_\text{n}$, ovvero $\gcd(C_1, n) = 1$: ciò è sempre "quasi" vero perché se non fosse così il massimo comun divisore tra C_1 e n sarebbe uguale ad uno dei fattori di n, da cui cadrebbe l'intero criptosistema. Questo fatto è praticamente impossibile perché in $\mathbb{Z}_\text{n}$ il numero di numeri primi multipli di p (o di q) è piccolissimo e quindi la generazione casuale difficilmente becca questo tipo di numeri durante la cifratura.

4. **Attacchi basati sull'autenticazione**

   Supponiamo che un utente malizioso abbia intercettato un messaggio $y = M^e \bmod n$ mandato da Bob ad Alice (quindi si è usata la chiave pubblica di Alice) e supponiamo che tale utente malizioso sia interessato a conoscere il messaggio inviato da Bob. Supponiamo inoltre che Alice sia disposta a firmare a cuor leggero qualsiasi messaggio.

   Charlie sceglie a caso un intero r tale che $\gcd(r, n) = 1$ (ovvero un numero in $\mathbb{Z}_n^*$, e la probabilità che tale numero non sia in $\mathbb{Z}_n^*$ è assolutamente trascurabile). Allora Charlie manda ad Alice da firmare il messaggio $y' = r^e y$ (e è noto essendo l'esponente di Alice, la chiave pubblica). Alice ritorna a Charlie il messaggio

   $$(y')^d \bmod n = r^{ed} y^d \bmod n = rM \bmod n$$

   da cui si ottiene $M = r^{-1}$. Si noti che r è utile per confondere Alice: in questo modo Alice non si accorge di firmare nuovamente un messaggio firmato precedentemente.

2.6 Numeri primi

Come visto nella sezione precedente, alla base della crittografia RSA vi è la teoria dei numeri primi: si ha la necessità di generare dei numeri primi molto grandi e casuali. L'approccio per generare un grande numero primo casualmente consiste nel generare un numero intero casualmente e poi effettuare un test di primalità sul numero generato: la prima operazione è molto semplice (effettuo un lancio di una moneta per ogni bit del numero da generare) mentre la verifica se un numero intero è primo o meno non è triviale. Dal 2004 esiste un algoritmo che verifica se un numero intero è primo in tempo polinomiale (algoritmo dei tre indiani), tuttavia odiernamente rimane impraticabile.

2.6.1 Teoria dei numeri primi

Sia $\Pi(n) = |\{p \in \mathbb{Z}^+ \mid p \leq n \ \wedge \ p \text{ primo}\}|$. Si può dimostrare che

$$\lim_{n \to +\infty} \frac{\Pi(n)}{\frac{n}{\ln n}} = 1.$$

Segue che i numeri primi sono densi in $\mathbb{Z}$, infatti[4]

$$P(\text{numero estratto sia primo}) \geq \frac{1}{\ln n}.$$

Sappiamo che per qualsiasi numero composto $n = pq$ si ha che $q \leq \sqrt{n}$. Forti di questo vincolo, si potrebbe pensare di applicare il crivello di Eratostene per verificare la primalità di n: tale approccio è infattibile poiché la complessità di un algoritmo del genere è $\Theta(\sqrt{n}) = \Theta(\sqrt{2}^{|\langle n \rangle|})$.

Dal piccolo teorema di Fermat possiamo ricavare il seguente corollario.

Corollario (Corollario del piccolo teorema di Fermat)**.** *Se $n \in \mathbb{Z}$ è primo, allora $\forall a \in \mathbb{Z}_n^*$ si ha $a^{n-1} \equiv 1 \bmod n$.*

[4] Si reitera con un processo di Bernoulli per circa $\ln n$ volte.

Diamo la seguente definizione.

Definizione. *Sia $n \in \mathbb{Z}$ un numero composto. Dato $a \in \mathbb{Z}_n^*$, n è detto* ***pseudoprimo di base*** *a se $a^{n-1} \equiv 1 \bmod n$.*

Esistono dei numeri composti tali per cui la definizione appena fornita è valida $\forall a \in \mathbb{Z}_n^*$: tali numeri sono detti **numeri di Carmichael**.

2.6.2 Test di primalità di Pomerance

Il corollario e la definizione fornite nel paragrafo precedente potrebbero costituire un criterio per il test di primalità. Seppur molto pochi, il test di primalità può incappare in un numero di Carmichael. L'algoritmo che verrà fornito è un algoritmo deterministico e basa il test sui numeri pseudoprimi in base 2, ovvero, se n è composto, si ha

$$2^{n-1} \bmod n \not\equiv 1$$

con alta probabilità. Vale infatti il seguente teorema.

Teorema (Teorema di Pomerance)**.** *Dato un intero x composto tale che $x \leq n$, allora*

$$P(x \text{ è uno pseudoprimo in base } 2) \leq c \cdot e^{-\frac{1}{2} \ln n \cdot \frac{\ln \ln \ln n}{\ln \ln n}}.$$

Si noti che al crescere di n l'esponenziale scende molto velocemente a zero.

L'algoritmo 11 descrive il test di pseudoprimalità: si noti che l'algoritmo può sbagliare quando ritorna che il numero di input è primo poiché tale numero potrebbe essere uno pseudoprimo in base 2.

Algorithm 11 Test di pseudoprimalità

```
function PSEUDOPRIMALITY(x)
    if MOD_EXP(2, x − 1, x) ≠ 1 then
        return COMPOSITE                          ▷ sicuramente
    end if
    return PRIME                                  ▷ si spera!
end function
```

L'algoritmo 12 fornisce un numero intero primo casuale. L'input S di tale algoritmo è utile per dimensionare la probabilità di errore dell'algoritmo.

Algorithm 12 Generazione di un numero primo casuale (Pomerance)

```
function SELECT_RANDOM_PRIME(S, n)
    repeat
        x ← RANDOM(1, n)
        if PSEUDOPRIME(x)=PRIME then
            return x
        end if
    until S ln n
    return FAILURE
end function
```

Analisi L'analisi di un algoritmo di questo tipo consiste nel determinare la probabilità di correttezza (o analogamente la probabilità di errore). Calcoliamo quindi la probabilità che l'algoritmo fallisca, ovvero che tutti i numeri estratti siano numeri che imbrogliano il test di pseudoprimalità:

$$\begin{aligned} P(\text{FAILURE}) &\leq P(\text{tutti i numeri estratti sono composti}) \\ &\leq \left(1 - \frac{\Pi(n)}{n}\right)^{S \ln n} \\ &\leq \left(1 - \frac{1}{\ln n}\right)^{S \ln n} \\ &\leq e^{-S} \end{aligned}$$

essendo che

$$\left(1 - \frac{1}{k}\right)^{k} \leq e^{-1} \ \forall k \neq 0.$$

Definendo l'evento

$E_i =$ "all'i-esima iterazione viene selezionato uno pseudoprimo in base 2"

segue che

$$\begin{aligned} &P(\text{SELECT\_RANDOM\_PRIME ritorna COMPOSITE}) \\ &\leq P(\text{viene selezionato } \textit{almeno} \text{ uno pseudoprimo in base 2}) \\ &= P(\bigcup_{i=1}^{S \ln n} E_i) \\ &\leq \sum_{i=1}^{S \ln n} P(E_i) \\ &\leq S \ln n \cdot c e^{-\frac{1}{2} \ln n \cdot \frac{\ln \ln \ln n}{\ln \ln n}} \\ &\leq \frac{S}{\log^k n} \ \ \forall k \neq 0 \quad \xrightarrow[S \text{ costante}]{} 0 \end{aligned}$$

Tempo di esecuzione L'algoritmo esegue delle esponenziazioni modulari, pertanto l'algoritmo ha complessità cubica rispetto alla lunghezza della rappresentazione binaria di n *per ogni invocazione* di PSEUDOPRIME: ogni operazione modulare costa circa n^2 e ne devono essere fatte circa lunghezza di n (l'esponenziazione richiede di fare un numero di moltiplicazioni che è circa lineare nella lunghezza dell'esponente). Di conseguenza l'algoritmo di selezione ha una complessità temporale di circa $S|\langle n \rangle|^4$.

2.6.3 Test di primalità di Miller Rabin

Nella sezione precedente abbiamo visto un algoritmo per l'estrazione di un numero primo casuale. In questa sezione vediamo invece come determinare se un numero n fissato è primo o composto. Per far ciò introduciamo prima due algoritmi: il primo effettua la conversione in base decimale a partire dalla rappresentazione binaria mentre il secondo si occupa dell'esponenziazione veloce.

Data un numero $e \in \mathbb{Z}$, la sua rappresentazione binaria è data da

$$\vec{e} = \langle e_k, e_{k-1}, \cdots, e_0 \rangle, \quad k = \lfloor \log_2 e \rfloor$$

e da essa è possibile ricavare la sua rappresentazione decimale come

$$e = \sum_{j=0}^{k} 2^j e_j.$$

L'algoritmo 13 implementa, in una versione leggermente differente, quanto appena descritto.

Algorithm 13 Conversione da rappresentazione binaria a rappresentazione decimale

```
function BIN2DEC(ē)
    k ← ē.length − 1
    c ← 0
    for i ← k downto 0 do
        c ← c ∗ 2
        if e_i = 1 then
            c ← c + 1
        end if
    end for
    return c
end function
```

L'algoritmo 14 implementa l'esponenziazione veloce.

Algorithm 14 Esponenziazione veloce

```
function MOD_EXP(M, ē, n)
    k ← ē.length − 1
    d ← 1
    for i ← k downto 0 do
        d ← (d ∗ d) mod n
        if e_i = 1 then
            d ← (M ∗ d) mod n
        end if
    end for
    return d
end function
```

Terminata questa premessa, procediamo con l'introduzione dell'algoritmo di Miller Rabin partendo dal certificato di non primalità. Per determinare un certificato di non primalità effettuiamo i seguenti due passi:

1. a differenza del test di pseudoprimalità di Pomerance, che prova la primalità solo sugli pseudoprimi in base 2, Miller Rabin effettua il test per vari valori di $a \in \mathbb{Z}_n^*$ estratti casualmente;
2. l'esponenziazione $a^{n-1} \bmod n$ viene calcolata con una leggera modifica dell'algoritmo per l'esponenziazione veloce in maniera tale da individuare in una qualche iterazione se $\exists d \not\equiv \pm 1 \bmod n \mid d^2 \equiv 1 \bmod n$.

Miller Rabin cerca l'esistenza di radici quadrate non banali dell'unità: se ne esiste almeno una allora n non è un numero primo.

Teorema. *Se* $\exists d \in \mathbb{Z}_n^* \mid d \not\equiv \pm 1 \bmod n \mid d^2 \equiv 1 \bmod n$ *allora* n *è un numero composto.*

Dimostrazione. Provo la contropositiva, ovvero che se n è primo allora le uniche radici quadrate dell'unità sono congruenti a $\pm 1 \bmod n$. Sia x una radice dell'unità. Allora

$$\begin{aligned} x^2 - 1 &\equiv 0 \bmod n \\ (x+1)(x-1) &\equiv 0 \bmod n \\ (x+1)(x-1) &= kn \quad k \in \mathbb{Z} \end{aligned}$$

Poiché n è primo, se vale l'ultima espressione, allora $n \mid (x+1) \ \vee \ n \mid (x-1)$. Per assurdo, se così non fosse, ovvero se $n \not| (x+1) \ \wedge \ n \not| (x-1)$, allora

$$\gcd(n, x+1) = 1 \ \wedge \ \gcd(n, x-1) = 1 \Rightarrow \gcd(n, (x+1)(x-1)) = 1$$

e ciò è impossibile avendo assunto che $(x+1)(x-1)$ sia un multiplo intero di n. Di conseguenza,

- se $n \mid (x+1)$ allora esiste $k \mid (x+1) = kn$, da cui segue

$$x = -1 + kn \Rightarrow x \equiv -1 \bmod n;$$

- se $n \mid (x-1)$ allora esiste $k \mid (x-1) = kn$, da cui segue

$$x = +1 + kn \Rightarrow x \equiv 1 \bmod n.$$

□

Il certificato di non primalità è il testimone che garantisce che n è non primo se composto. Questo test funziona anche per i numeri di Carmichael. L'estrazione della base dell'esponenziazione viene effettuata randomicamente: gli $a \in \mathbb{Z}_n^+ - \mathbb{Z}_n^*$ sono molto pochi, inoltre, se anche si estraesse un tale numero, poiché non ha inverso moltiplicativo, il certificato di non primalità garantisce che tale numero è un numero composto.

Alcune considerazioni prima di proporre i due algoritmi. Se n è pari, è immediato assumere che $n = 2$ è primo e che un qualsiasi $n > 2$ è composto. Soffermandoci quindi sui soli numeri dispari, considero la rappresentazione binaria di $n - 1$:

$$(n-1)_2 \ : \ \underbrace{|\cdots\cdots\cdots\cdots|1}_{\text{u}} \underbrace{|0\cdots 0|}_{\text{t}}$$

dove u rappresenta il numero rappresentato dalla cifre più significative e t è il numero di zeri (consecutivi) alla fine della rappresentazione binaria. Di conseguenza, si ha che $(n-1) = u \cdot 2^t$. Allora, definito $d = a^u \bmod n$ e notando che $a^{n-1} \bmod n = a^{u2^t} \bmod n = (a^u)^{2^t} \bmod n$, in 15 è descritto l'algoritmo che fornisce il certificato di non primalità (se possibile). Segue in 16 l'algoritmo di Miller Rabin.

Algorithm 15 Certificato di non primalità per Miller Rabin

```
function CERTIFICATE(a, n)
    ** (n − 1) = u · 2^t **
    d ←MOD_EXP(a, u, n)
    for i ← 1 to t do
        d′ ← (d ∗ d) mod n
        if d′ ≡ 1 mod n then
            if d ≢ ±1 mod n then
                return COMPOSITE
            else
                return NONWITNESS
            end if
        end if
    end for
    return COMPOSITE
end function
```

Algorithm 16 Algoritmo di Miller Rabin

```
function MILLER_RABIN(S, n)
    if n = 2 then
        return PRIME
    end if
    if EVEN(n) then
        return COMPOSITE
    end if
    for i ← 1 to S do
        a ← RANDOM(1 · · · n − 1)
        if CERTIFICATE(a, n)=COMPOSITE then
            return COMPOSITE
        end if
    end for
    return PRIME
end function
```

Analisi L'algoritmo di Miller Rabin ha un tempo di esecuzione pari a $T_{MR}(S, n) = O(S \cdot |\langle n \rangle|^3)$. La correttezza invece deriva dal seguente teorema.

Teorema. *Per qualsiasi $n > 2$ dispari e composto*

$$|\{a \in \mathbb{Z}_{\mathrm{n}}^{+} \ : \ CERTIFICATE(a, n) = NONWITNESS\}| \leq \frac{n-1}{2}.$$

Dimostrazione. Si fornisce come prova solo una intuizione per il caso più semplice: si consideri un numero n composto, dispari e **non di Carmichael**. Allora

$$\exists \bar{a} \in \mathbb{Z}_{\mathrm{n}}^{*} \ : \ \bar{a}^{n-1} \not\equiv 1 \bmod n.$$

Si considerino i valori $b \mid b^{n-1} \equiv 1 \bmod n$ con $b \in \mathbb{Z}_{\mathrm{n}}^{*}$. Allora, si può dimostrare che l'insieme $\{$non certificati in $\mathbb{Z}_{\mathrm{n}}^{*}\}$ forma un sottogruppo proprio di $\mathbb{Z}_{\mathrm{n}}^{*}$ e sia T la cardinalità di tale insieme. Per il teorema di Lagrange[5] si ha $T \mid |\mathbb{Z}_{\mathrm{n}}^{*}|$, di conseguenza $T \leq |\mathbb{Z}_{\mathrm{n}}^{*}|$, da cui $T \leq \frac{|\mathbb{Z}_{\mathrm{n}}^{*}|}{2} \leq \frac{n-1}{2}$ ($|\mathbb{Z}_{\mathrm{n}}^{*}| \leq n-1$). □

La probabilità di errore di Miller Rabin è

$$\begin{aligned} &P(\text{Miller Rabin sbaglia}) \\ &\leq P(\text{Miller Rabin ritorna PRIME per un numero composto} \\ &\leq \left(\frac{1}{2}\right)^{S} \end{aligned}$$

poiché per fallire deve sbagliare ad ogni iterazione, le quali sono indipendenti[6]) . Se considero $S = k \log_2 |\langle n \rangle|$, allora

$$P(\text{Miller Rabin sbaglia}) \leq \frac{1}{|\langle n \rangle|^k}$$

così da legare la probabilità di errore alla taglia dell'input.

[5]Il teorema di Lagrange afferma che la cardinalità di un sottogruppo divide la cardinalità del gruppo da cui ha origine.

[6]Segue dal teorema precedente che P(CERTIFICATE=NONWITNESS) $\leq \frac{\frac{n-1}{2}}{n-1} = \frac{1}{2}$.

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.

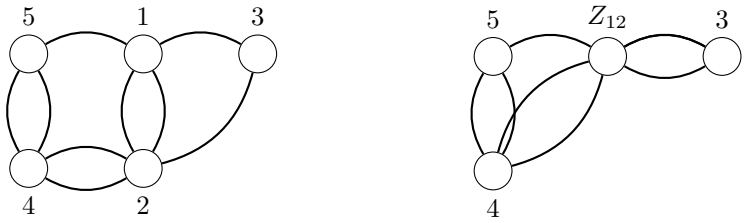
Definizione. *Un* ***multigrafo*** $\mathcal{G} = (V, \mathcal{E})$ *è un grafo definito su un insieme di vertici* V *e su un multiinsieme*[2] *di archi* $\mathcal{E}$ *dove ad ogni arco è associata una molteplicità.*

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

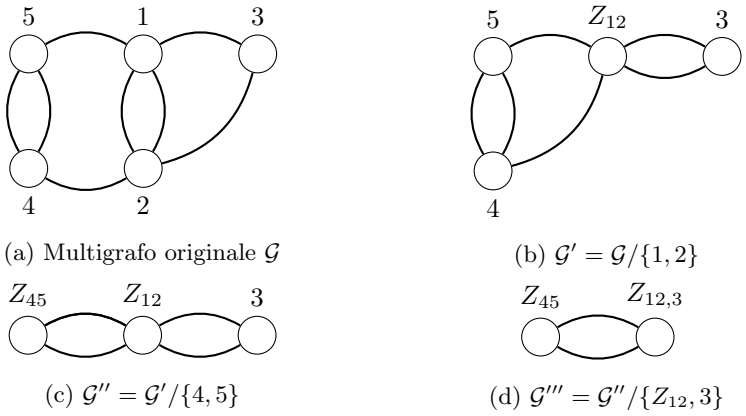
In questa sezione forniamo l'algoritmo di Karger per il calcolo del taglio minimo di un multigrafo. In 18 è descritta la procedura per effettuare la contrazione piena di un multigrafo: essa termina quando il grafo avrà solamente due nodi e verrà restituito il numero di archi presenti nel fascio che connette 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

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

conquer 21. Si noti che l'algoritmo considera come caso base un multigrafo di non più di 8 nodi: ciò sarà motivato dalla successiva analisi. È inoltre possibile dimostrare che la scelta migliore di k è

$$k^* = n - \lceil n/\sqrt{2} + 1 \rceil.$$

Algorithm 20 Partial contraction

```
function PARTIAL_CONTRACTION(G = (V, E), k)
    for i ← 1 to k do
        e ←RANDOM(E)
        G ← G/e
    end for
    return G
end function
```

Algorithm 21 Algoritmo di Karger-Stein

```
function KARGER_STEIN(G = (V, E))
    if |V| ≤ 8 then
        ** risolvi direttamente **
    end if
    k ← f(|V|)
    G1 ←PARTIAL_CONTRACTION(G, k)
    G2 ←PARTIAL_CONTRACTION(G, k)
    return MIN(KARGER_STEIN(G1), KARGER_STEIN(G2))
end function
```

Analisi Dall'analisi di Karger svolta nella sezione precedente possiamo calcolare la probabilità che il taglio minimo sopravviva alla contrazione parziale:

$$\begin{aligned}
P(\text{"min cut sopravvive in } \mathcal{G}_1\text{"}) &= P(\text{"min cut sopravvive in } \mathcal{G}_2\text{"}) \\
&\geq \prod_{i=1}^{n-\lceil n/\sqrt{2}+1 \rceil} \left(\frac{n-i-1}{n-i+1} \right) \\
&= \frac{\lceil n/\sqrt{2}+1 \rceil (\lceil n/\sqrt{2}+1 \rceil - 1)}{n(n-1)} \\
&\geq \frac{\frac{n}{\sqrt{2}} \cdot \frac{n}{\sqrt{2}}}{n(n-1)} \\
&= \frac{n^2}{2n(n-1)} \\
&\geq \frac{n^2}{2n^2} \\
&= \frac{1}{2}
\end{aligned}$$

Questa probabilità è valida per tutti i livelli della ricorsione essendo indipendente rispetto ai nodi.

Il tempo di esecuzione invece **non** è una variabile aleatoria:

$$T(n) = \begin{cases} c_1 & n \leq 8 \\ 2T(\lceil n/\sqrt{2} + 1 \rceil) + O(n^2) & n > 8 \end{cases}$$

dove la contrazione abbassa la taglia a $n - k$ nodi e il lavoro svolto dalla fase di *conquer* è data dalle due contrazioni parziali $O(m) = O(n^2)$. Ignorando il "+1" nella taglia dell'istanza è possibile studiare questa ricorrenza con il Master Theorem: $a = 2$, $b = \sqrt{2}$, $w(n) = n^2$ e con la funzione di soglia $n^{\log_b a} = n^{\log_{\sqrt{2}} 2} = n^2$ tale ricorrenza rientra nel caso due del Master Theorem, pertanto $T(n) = \Theta(n^2 \log n)$.

Ora dimostreremo che $P(n) = P(\text{"Karger-Stein è corretto"}) \geq c/\ln n$: assunto ciò è sufficiente eseguire l'algoritmo $(d/c)\ln^2 n$ per ottenere l'alta probabilità, infatti

$$\begin{aligned} P(\text{"nessuna esecuzione non ritorna il min cut"}) &\leq \left(1 - \frac{c}{\ln n}\right)^{\frac{d}{c}\ln^2 n} \\ &= \left[\left(1 - \frac{c}{\ln n}\right)^{\frac{\ln n}{c}}\right]^{d \ln n} \\ &< e^{-d \ln n} \\ &= \frac{1}{n^d} \end{aligned}$$

Per dimostrare ciò sfrutto la ricorrenza anche per il calcolo delle probabilità: $P(n) = P(\text{"successo se } |V| = n\text{"})$, quindi

$$P(n) = 1 \quad \text{se} \quad n \leq 8$$

mentre, per $n > 8$,

$$\begin{aligned} P(n) &= 1 - P(\text{"falliscono entrambe le contrazioni parziali o le chiamate ricorsive"}) \\ &= (1 - P(\text{"una contrazione parziale o la relativa chiamata ricorsiva falliscono"}))^2 \\ &= 1 - (1 - P(\text{"ha successo sia la contrazione parziale che la relativa chiamata ricorsiva"}))^2 \\ &\geq 1 - \left(1 - \frac{1}{2}P(\lceil n/\sqrt{2} + 1 \rceil)\right)^2 \end{aligned}$$

Com'è la progressione delle taglie dalla radice dell'albero delle chiamate (livello 0) fino al raggiungimento di una foglia (livello $h-1$)? È possibile dimostrare che al livello i la taglia dell'istanza vale

$$S_i \leq \frac{n}{(\sqrt{2})^i} + 7.$$

Il numero massimo di livelli è $h = \lceil 2 \ln n + 1 \rceil$, per cui $h - 1 = \lceil 2 \ln n \rceil$ e

$$\begin{aligned} S_{h-1} &\leq \frac{n}{(\sqrt{2})^{\lceil 2 \ln n \rceil}} + 7 \\ &\leq \frac{n}{(\sqrt{2})^{2 \ln n}} + 7 \\ &\leq \frac{n}{(\sqrt{2})^{\log_{\sqrt{2}} n}} + 7 \\ &= \frac{n}{n} + 7 \\ &= 8 \end{aligned}$$

motivando così perché come caso base consideriamo $n \leq 8$.

Consideriamo ora la funzione di probabilità non in funzione della taglia dell'istanza ma in funzione del numero di livelli. Allora, essendo $P(n) = \bar{P}(\lceil 2 \ln n + 1 \rceil)$, si ha

$$\begin{cases} \bar{P}(h) \geq 1 - \left(1 - \frac{1}{2} P(h-1)\right)^2 \\ \bar{P}(1) = 1 \end{cases}$$

Per induzione parametrica dimostriamo che $\bar{P}(h) \geq 1/h$:

Base Per $h = 1$ si ha $\bar{P}(1) = 1 = 1/h$, quindi il predicato è vero.

Ipotesi induttiva Assumo il predicato vero fino a $h - 1$, $\bar{P}(h-1) \geq 1/(h-1)$.

Induzione

$$\begin{aligned} \bar{P}(h) &\geq 1 - \left(1 - \frac{1}{2} \bar{P}(h-1)\right)^2 \\ &= 1 - 1 - \frac{1}{4} \bar{P}^2(h-1) + P(h-1) \\ &= \bar{P}(h-1) - \frac{1}{4} \bar{P}^2(h-1). \end{aligned}$$

Considerando la funzione $f(x) = x - \frac{1}{4} x^2$ ristretta all'intervallo $[0, 1]$ osservo che la funzione è monotona crescente. Segue

$$\begin{aligned} \bar{P}(h) &\geq \bar{P}(h-1) - \frac{1}{4} \bar{P}^2(h-1) \\ &\geq \frac{1}{h-1} - \frac{1}{4} \left(\frac{1}{h-1}\right)^2 \\ &\geq \frac{1}{h-1} - \frac{1}{2(h-1)^2} \\ &= \frac{1}{h-1} - \frac{1}{2(h-1)(h+1)} \\ &\geq \frac{1}{h-1} - \frac{1}{h(h-1)} && (2(h-1) \geq h \text{ per } h \geq 2) \\ &= \frac{1}{h}. \end{aligned}$$

□

Dimostrato che $\bar{P}(h) \geq 1/h$, segue che per $n > 1$

$$\begin{aligned} P(\text{"Karger-Stein è corretto"}) &\geq \frac{1}{\lceil 2\ln n + 1 \rceil} \\ &\geq \frac{1}{2\log_2 n + \log_2 n} \\ &= \frac{1}{3\log_2 n} \\ &= \frac{1}{3\frac{\ln n}{\ln 2}} \\ &= \frac{(\ln n)/3}{\ln n} \end{aligned}$$

con $(\ln n)/3 = c$, come volevamo.

3.5 Polling: simulazione Montecarlo

Sia data una urna U contentente palline bianche (B, successo) e nere (N, tutto il resto). Sia $|U| = n$. Sia data inoltre una stima deterministica conservativa del numero delle palline bianche: $|B| \geq \alpha_{min} n$, $0 \leq \alpha_{min} \leq 1$. L'interesse è nel determinare $|B| = \alpha n$, con $0 \leq \alpha_{min} \leq \alpha \leq 1$.

Ogni approccio deterministico richiede l'estrazione di tutte le palline dall'urna. Un approccio randomizzato invece effettua una estrazione casuale (uniforme) di palline dall'urna con reinserimento. Nella pratica è difficile mantenere l'uniformità dell'estrazione (si pensi agli *exit poll*).

In 22 è descritto un algoritmo randomizzato. Il parametro ϵ è un parametro di confidenza che rappresenta l'errore relativo che si commette nell'approssimare α con x/k.

Algorithm 22 Algoritmo di polling

```
function APPROXIMATE_α(U, ε, α_min)
    n ← |U|
    k ← f(n, ε, α_min)
    x ← 0
    for i ← 1 to k do
        p ←RANDOM(U)
        if p.color = B then
            x ← x + 1
        end if
    end for
    return x/k
end function
```

Analisi Supponiamo di voler fornire una garanzia sull'errore relativo, ovvero che

$$P\left(\frac{|\frac{x}{k} - \alpha|}{\alpha} > \epsilon\right) < \frac{1}{n}.$$

Sia $X_i = 1$ la v.a. che rappresenta l'i-esima estrazione risultata come bianca (e $X_i = 0$ altrimenti). Allora il numero totale di estrazioni avvenute con successo è pari a $X = \sum_{i=}^{k} X_i$. Inoltre, si ha che $P(X_i = 1) = \frac{\alpha n}{n} = \alpha$, da cui segue che $E[X_i] = \alpha$, ovvero che $E[X] = k\alpha = \mu$. Essendo X una somma di Bernoulli possiamo applicare i bound di Chernoff:

$$\begin{aligned} P\left(\frac{|\frac{x}{k} - \alpha|}{\alpha} > \epsilon\right) &= P\left(\frac{|X - k\alpha|}{k\alpha} > \epsilon\right) \\ &= P\left(\frac{|X - \mu|}{\mu} > \epsilon\right) \\ &< 2e^{-\frac{\epsilon^2\mu}{3}} \\ &= 2e^{-\frac{\epsilon^2 k\alpha}{3}} \\ &< 2e^{-\frac{k\epsilon^2\alpha_{min}}{3}} \end{aligned}$$

poiché valgono le seguenti uguaglianze tra eventi

$$\begin{aligned} \text{“}\frac{|X - \mu|}{\mu} > \epsilon\text{”} &\Leftrightarrow \text{“}\frac{X - \mu}{\mu} > \epsilon\text{”} \cup \text{“}\frac{\mu - X}{\mu} > \epsilon\text{”} \\ &\Leftrightarrow \text{“}X > (1+\epsilon)\mu\text{”} \cup \text{“}X < (1-\epsilon)\mu\text{”} \end{aligned}$$

ed effettuo la maggiorazione considerando l'elemento più grande (corollario 3.2). Ora, scegliendo

$$k = \frac{3}{\alpha_{min}\epsilon^2}\ln(2n)$$

si ottiene

$$2e^{-\frac{3}{\alpha_{min}\epsilon^2}\ln(2n)\frac{\epsilon^2\alpha_{min}}{3}} = 2e^{-\ln(2n)} = 2\frac{1}{2n} = \frac{1}{n}$$

confermando la garanzia che volevamo ottenere.

Il numero di estrazioni da effettuare è $k = \Theta(\log n)$: con l'approccio randomizzato abbiamo ottenuto un guadagno esponenziale rispetto allo spoglio deterministico. Questo tipo di tecnica è anche detta **simulazione Montecarlo**.

3.6 Coupon collecting: enumerazione randomizzata

Sia dato l'insieme $U = \{1, \cdots, n\}$ e la primitiva RANDOM$(1, n)$ che effettua una estrazione casuale (uniforme) di un intero tra 1 e n. Si vuole determinare il numero di estrazioni (con reinserimento) necessarie per vedere tutti gli elementi dell'insieme.

Sia $Z_i \sim Geom(p_i)$ una v.a. geometrica[4] di parametro p_i che rappresenta il numero di estrazioni necessarie a scoprire un nuovo elemento dopo averne

[4]Una variabile aleatoria geometrica conta il numero di lanci fino al primo successo. Il parametro rappresenta la probabilità di successo, quindi $P(Z_i = j) = (1 - p_i)^{j-1}p_i$. Segue immediatamente che $E[Z_i] = 1/p_i$.

scoperti $i-1$. Il numero di estrazioni complessive per scoprire tutti gli elementi è dato da $Z = \sum_{i=1}^{n} Z_i$, da cui $E[Z] = \sum_{i=1}^{n}(1/p_i)$. La probabilità p_i rappresenta la probabilità che venga estratto un nuovo elemento dopo averne scoperti $i-1$:

$$p_i = \frac{n-(i-1)}{n} = \frac{n-i+1}{n}.$$

Allora

$$\begin{aligned} E[Z] &= \sum_{i=1}^{n} \frac{n}{n-i+1} \\ &= n \sum_{i=1}^{n} \frac{1}{n-i+1} \\ &= n \sum_{j=1}^{n} \frac{1}{j} \qquad\qquad (j = n-i+1) \\ &= nH_n \\ &= n(\ln n + O(1)) \\ &= n \ln n + O(n). \end{aligned}$$

Fissiamo un elemento $j \in \{1, \cdots, n\}$. Allora

$$P(\text{"non aver scoperto } j \text{ dopo } r \text{ estrazioni"}) = \left(1 - \frac{1}{n}\right)^r < e^{-r/n}.$$

Se $r = 2n \ln n$ otteniamo

$$P(\text{"non aver scoperto } j \text{ dopo } r \text{ estrazioni"}) < e^{-r/n} = \frac{1}{n^2}.$$

Quindi

$$\begin{aligned} P(\text{"successo"}) &= P(\text{"scoperti tutti gli elementi con } 2n \ln n \text{ estrazioni"}) \\ &= 1 - P(\text{"insuccesso"}) \\ &= 1 - P(\text{"un qualche elemento non è stato scoperto"}). \end{aligned}$$

Definendo l'evento A="un qualche elemento non è stato scoperto" esso può essere espresso come

$$A = \bigcup_{i=1}^{n} A_i$$

dove A_i="l'elemento i non è stato scoperto". Allora

$$\begin{aligned} 1 - P(A) &= 1 - P\left(\bigcup_{i=1}^{n} A_i\right) \\ &\geq 1 - \sum_{i=1}^{n} P(A_i) \\ &\geq 1 - n\frac{1}{n^2} \\ &= 1 - \frac{1}{n} \end{aligned}$$

ottenendo la probabilità di successo in alta probabilità.

Capitolo 4

Esercitazioni

Vediamo ora una serie di esercitazioni sugli argomenti trattati durante il corso.

4.1 Approssimazione

4.1.1 Il problema SUBSET-SUM

In 23 proponiamo un secondo algoritmo di approssimazione per il problema SUBSET-SUM (vedi sezione 1.6).

Algorithm 23 Algoritmo di approssimazione per SUBSET-SUM

```
function APPROX_SUBSET-SUM(S, t)
    ⟨s_1, ··· , s_n⟩ ←SORT_DECREASING(S)
    sum ← 0
    i ← 1
    while sum + s_i ≤ t ∧ i ≤ n do
        sum ← sum + s_i
        i ← i + 1
    end while
    return sum
end function
```

L'algoritmo proposto è un algoritmo di 2-approssimazione. Se la variabile $sum = \sum_{i=1}^{n} s_i$ allora $\rho = 1$ poiché $sum^* = \sum_{i=1}^{n} s_i$. Altrimenti, $\exists \bar{i} \ : \ 2 \leq \bar{i} \leq n \mid sum = \sum_{i=1}^{\bar{i}-1} s_i \ (sum + s_{\bar{i}} > t)$. Vale inoltre la seguente proposizione.

Proposizione. *Vale sempre* $sum \geq t/2$.

Dimostrazione. Suppongo per assurdo che $sum < t/2$. Allora $s_1 < t/2 \Rightarrow \forall j \ s_j < t/2$. L'algoritmo termina all'iterazione $\bar{i}$ poiché $sum + s_{\bar{i}} > t$. Otteniamo

$$\begin{cases} sum < t/2 \\ s_{\bar{i}} < t/2 \end{cases} \Rightarrow sum + s_{\bar{i}} < t$$

condizione per la quale l'algoritmo non terminerebbe. □

Da tutto ciò segue

$$\rho = \frac{c(S^*)}{sum} \leq \frac{t}{t/2} = 2.$$

4.1.2 Il problema TRIANGLE-TSP

Proposizione. *Nell'ipotesi P≠NP, non può esistere un FPTAS per il problema TRIANGLE-TSP.*

Dimostrazione. Per questo genere di dimostrazione sfrutto ancora una volta il problema HAMILTON. Anche per TRIANGLE-TSP vale la riduzione polinomiale da HAMILTON vista nella sezione 1.3 poiché la funzione di costo associata ai lati del grafo soddisfa la disuguaglianza triangolare.

Supponiamo che l'FPTAS esista e che sia $\mathrm{A}(\langle G^c, c\rangle, \epsilon)$: per definizione di FPTAS, T_A è polinomiale rispetto sia alla taglia dell'istanza che a $1/\epsilon$. Se seleziono $\epsilon = 1/2|V|$, ottengo che $T_A = poly(1/\epsilon) = poly(2|V|) = poly(|\langle G^c, c\rangle|)$, potendo così spingere molto con l'approssimazione.

Procedo quindi applicando la funzione di riduzione polinomiale a HAMILTON e in seguito eseguo l'algoritmo. Allora:

1. se $\langle G = (V, E)\rangle \in$ HAMILTON allora esiste un ciclo hamiltoniano in G, ovvero esiste un tour in G^c di costo $|V|$. Allora l'algoritmo ritorna una soluzione di costo al più $(1+\epsilon)|V| = (1+\frac{1}{2|V|})|V| = |V| + \frac{1}{2}$: poiché tale valore non è intero la soluzione avrà necessariamente costo minore di $|V|+1$;

2. se $\langle G = (V, E)\rangle \notin$ HAMILTON allora in G^c il tour di costo minimo usa almeno un arco di costo 2, pertanto il costo della soluzione ottima è almeno pari a $|V|+1$. Allora l'algoritmo restituisce una soluzione di costo pari ad almeno $|V|+1$.

Si è instaurato un *se e solo se* che permetterebbe di testare HAMILTON in tempo polinomiale, il che è assurdo essendo nell'ipotesi che P≠NP. Da ciò segue che non può esistere un FPTAS per TRIANGLE-TSP. □

4.1.3 Il problema del LOAD-BALANCING

Siano dati n *job* $J_1, J_2, \cdots, J_n$ che richiedono tempo, rispettivamente, $t_1, t_2, \cdots, t_n$. Si supponga che tutte le attività siano indipendenti e che qualsiasi *job* possa essere eseguito su una qualsiasi delle m macchine disponibili. Una soluzione ammissibile del problema è una partizione degli indici dei *job* tra m insiemi rappresentanti le macchine: $\mathcal{P} = \{M_1, M_2, \cdots, M_m\}$. Per la macchina i il tempo complessivo di occupazione è $T_i = \sum_{j \in M_i} t_j$. L'obiettivo è quello di minimizzare $T^* = \max\{T_i \ : \ 1 \leq i \leq m\}$. Si osservi che per qualsiasi soluzione ammissibile vale $\sum_{j=1}^{n} t_j = \sum_{k=1}^{m} T_k$. In 24 lo pseudocodice dell'algoritmo.

Proposizione. *Vale*

$$T^* \geq \max\left\{ \frac{1}{m}\sum_{j=1}^{n} t_j, \max_{1\leq j\leq n}\{t_j\} \right\}.$$

Algorithm 24 Algoritmo di approssimazione per LOAD-BALANCING

function GREEDY_SCHEDULING($\vec{t}, m$)
  for $i \leftarrow 1$ **to** m **do**
    $T_i \leftarrow 0$
    $M_i \leftarrow \emptyset$
  end for
  for $j \leftarrow 1$ **to** n **do**
    $k \leftarrow \text{argmin}\{T_i \ : \ 1 \leq i \leq m\}$
    $M_k \leftarrow M_k \cup \{j\}$
    $T_k \leftarrow T_k + t_j$
  end for
  return MAX($\vec{T}$)
end function

Dimostrazione. Nella soluzione ottima esiste una macchina h che esegue t_j, quindi $T_h^* \geq t_j$. Quindi $T^* = \max_{1 \leq i \leq m} T_i \geq \max_{1 \leq j \leq n} t_j$.

Per assurdo suppongo che $T^* < \frac{1}{m} \sum_{j=1}^{n} t_j$. Allora $T_i^* < \frac{1}{m} \sum_{j=1}^{n} t_j$, da cui

$$\sum_{i=1}^{m} T_i^* < \sum_{i=1}^{m} \frac{1}{m} \sum_{j=1}^{n} t_j = \sum_{j=1}^{n} t_j$$

giungendo così ad un assurdo. □

Proposizione. *L'algoritmo proposto è un algoritmo di 2-approssimazione.*

Dimostrazione. Sia $T^G = \max_{1 \leq i \leq m} \{T_i^G\} = T_{\bar{i}}^G$ e sia $\bar{j}$ l'indice dell'ultimo *job* assegnato alla macchina $\bar{i}$. All'iterazione $\bar{j}$ il tempo $T_{\bar{i}}^G - t_{\bar{j}}$ era il minimo tra tutti i tempi T_i^G, ovvero $T_i^G \geq T_{\bar{i}}^G - t_{\bar{j}}$ per $1 \leq i \leq m$. Allora

$$m(T_{\bar{i}}^G - t_{\bar{j}}) \leq \sum_{i=1}^{m} T_i^G \leq \sum_{j=1}^{n} t_j \leq mT^*$$

dove l'ultima relazione segue dalla proposizione precedente. Da ciò segue che $T_{\bar{i}}^G - t_{\bar{j}} \leq T^*$, ovvero

$$T^G = T_{\bar{i}}^G \leq T^* + t_{\bar{j}} \leq T^* + T^* = 2T^*.$$

Concludendo si ottiene la tesi

$$\rho = \frac{T^G}{T^*} \leq 2.$$

□

4.2 Aritmetica intera

4.2.1 Divisione intera e modulo

Ora proponiamo un algoritmo *divide and conquer* con il quale risolvere il calcolo della divisione intera e del modulo tra due interi. Ragioniamo con la rappresentazione binaria degli interi.

Dati due interi a e b, per il teorema di divisione $\exists!\ q, r$ interi tali che $a = qb+r$, con $0 \leq r < b$. È possibile osservare come il resto della divisione sia robusto rispetto alla sottrazione di multipli di b:

$$\begin{aligned} a &= qb + r \\ a - mb &= qb - mb + r \\ a' &= (q - m)b + r \end{aligned}$$

L'algoritmo che verrà qui proposto si basa proprio su questa osservazione. Perché esso sia efficiente è conveniente che m sia multiplo della base in modo da poter applicare il velocissimo algoritmo di *shift*.

L'algoritmo si delinea secondo queste idee:

- **caso di base**: se $a < b$ allora la coppia quoziente e resto vale $(q, r) = (0, a)$;
- **caso non di base**: se $a \geq b$, allora sia $k \mid 2^k b \leq a < 2^{k+1} b$. Quindi

$$a = qb + r \Rightarrow a - 2^k b = (q - 2^k)b + r \Rightarrow (q', r') = (q - 2^k, r)$$

Perché l'algoritmo funziona a dovere? Ricordiamo infatti che è un algoritmo *divide and conquer*, il quale richiede la riduzione della taglia dell'istanza di ingresso ad ogni iterazione ricorsiva. Infatti, essendo $a < 2^{k+1} b \Leftrightarrow a/2 < 2^k b$ si ha

$$a - a^{2b} < a - \frac{a}{2} = \frac{a}{2}$$

garantendo così almeno il dimezzamento della taglia. In questo modo è anche possibile stabilire che il numero di iterazioni è logaritmico nella taglia dell'istanza. In 25 lo pseudocodice dell'algoritmo. Considerata come taglia il numero di bit del dividendo, la complessità dell'algoritmo è

$$T(n) = T(n-1) + O(n) = O(n^2)$$

dove il dimezzamento del dividendo è rappresentato dalla diminuzione di una unità del numero di bit della sua rappresentazione binaria, mentre il lavoro delle procedure di SHIFT, SUM e SUB, come noto, è lineare.

Algorithm 25 Algoritmo *divide and conquer* per il calcolo di quoziente e resto

```
function R_QR(a, b)
    if a < b then
        return (0, a)
    end if
    k ← a.length − b.length
    if SHIFT(b, k)> a then
        k ← k − 1
        (q′, r′) ←R_QR(SUB(a, SHIFT(b, k)), b)
    end if
    return (SUM(q′, SHIFT(1, k)), r′)
end function
```

4.2.2 Il minimo comune multiplo

In questa sezione forniamo un algoritmo per il calcolo del minimo comune multiplo tra due interi.

Definizione. *Il **minimo comune multiplo** (least common multiple) è definito come segue*

$$\operatorname{lcm}(a,b) = \min\{c > 0 \mid (a|c) \ \wedge \ (b|c)\}.$$

Vale la seguente proposizione.

Proposizione. *Per qualsiasi intero $a, b > 0$ si ha*

$$\operatorname{lcm}(a,b) = \frac{ab}{\gcd(a,b)}.$$

Dimostrazione. Dalla relazione proposta segue che

$$ab = \gcd(a,b)\operatorname{lcm}(a,b) = dm$$

con $d = \gcd(a,b)$ e $m = \operatorname{lcm}(a,b)$. Poiché $(d|a) \ \wedge \ (d|b)$ allora $d|ab$ e di conseguenza, per definizione, $\exists n \ : \ ab = dn$. La dimostrazione verte quindi nel dimostrare che $m = \operatorname{lcm}(a,b) = n$.

Valgono le seguenti considerazioni

$$\begin{cases} d|a \Rightarrow a = kd \\ d|b \Rightarrow b = hd \end{cases} \Rightarrow ab = kdb = ahd = dn$$

da cui segue che $(n = kb) \ \wedge \ (n = ah)$, ovvero che n è multiplo comune di a e b, quindi $n \geq \operatorname{lcm}(a,b) = m$.

Dato che $m = \operatorname{lcm}(a,b)$, $(m = ar) \ \wedge \ (m = bs)$. Per l'identità di Bézout $\exists d \mid d = ax + by$. Quindi

$$\begin{aligned} md &= axm + bym \\ &= axbs + byar \\ &= ab(xs + yr) \\ &= dn(xs + yr) \end{aligned}$$

da cui, semplificando, si ottiene che $m = n(xs + yr)$, ed essendo $(xs + yr) > 0$, si ha che $n|m$ e quindi $n \leq m$.

Mettendo insieme i due risultati ottenuti si ottiene la tesi. □

Forti della dimostrazione appena fornita, in 26 è proposto l'algoritmo per il calcolo del minimo comune multiplo.

Algorithm 26 Algoritmo per il calcolo del minimo comune multiplo

```
function LCM(a, b)
    return ab/EUCLID(a, b)
end function
```

4.2.3 Unicità dell'inverso moltiplicativo in $\mathbb{Z}_n^*$

In questa sezione dimostreremo l'unicità dell'inverso moltiplicativo in $\mathbb{Z}_n^*$.

Proposizione. *Dato $n \in \mathbb{Z}$, $\forall a, b \in \mathbb{Z}^+$, si ha*

$$(n|ab) \ \wedge \ (\gcd(a, n) = 1) \Rightarrow n|b.$$

Dimostrazione. Dalla prima ipotesi $(n|ab)$ segue che $\exists k \in \mathbb{Z}^+ \mid ab = kn$. Dalla seconda ipotesi, per l'identità di Bézout $\exists x, y \in \mathbb{Z} \mid 1 = ax + ny$. Quindi

$$\begin{aligned} b &= b \cdot 1 \\ &= bax + bny \\ &= abx + nby \\ &= knx + nby \\ &= n(kx + by) \end{aligned}$$

da cui segue che $n|b$. □

Proposizione (Unicità dell'inverso moltiplicativo)**.** *In $\mathbb{Z}_n$, se $ax_1 \equiv ax_2 \equiv 1 \bmod n$, con $a \in \mathbb{Z}_n^*$, allora $x_1 \equiv x_2 \bmod n$.*

Dimostrazione.

$$ax_1 \equiv ax_2 \bmod n \Leftrightarrow (ax_1 - ax_2) \equiv 0 \bmod n$$

ovvero $\exists k \in \mathbb{Z} \mid a(x_1 - x_2) = kn$. Poiché $a \in \mathbb{Z}_n^*$, allora $\gcd(a, n) = 1$, ed avendo appena dimostrato che $n|a(x_1 - x_2)$, per la proposizione precedente segue che $n|(x_1 - x_2)$, ovvero che $x_1 - x_2 = hn$, da cui $x_1 \equiv x_2 \bmod n$. □

4.3 Randomizzazione

4.3.1 Il problema INDIPENDENT-SET

Forniamo le seguenti definizioni.

Definizione. *Dato un grafo $G = (V, E)$, un **indipendent set** è un insieme $V' \subseteq V \ : \ \forall u, v \in V', u \neq v$ si ha $\{u, v\} \notin E$.*

Definizione. *Dato un grafo $G = (V, E)$, un arco $e = \{u, v\} \in E$ è **interno** ad un insieme $V' \subseteq V$ se $\{u, v\} \in V'$.*

Dato un grafo $G = (V, E)$, con $n = |V|$ e $m = |E|$, si vuole determinare (se esiste) un suo insieme indipendente. Per la risoluzione di questo problema forniamo due approcci, entrambi basati sulla randomizzazione.

Approccio Montecarlo La risoluzione parte da un'idea banale: si estraggono casualmente r nodi dall'insieme V e si verifica se questo sottoinsieme di nodi forma un insieme indipendente del grafo. L'algoritmo 27 è un algoritmo Montecarlo in quanto la verifica di V' può fallire. La primitiva RANDOM(V, r) indica l'estrazione senza reinserimento di r elementi dall'insieme V.

Algorithm 27 Algoritmo Montecarlo per INDIPENDENT-SET

```
function IS1(G(V, E), r)
    V' ←RANDOM(V, r)
    if V' è indipendente then
        return V'
    else
        return FAILURE
    end if
end function
```

Non rimane che dimensionare il valore di r in modo che sia garantito che $P(\text{IS1 non fallisce}) \geq 1/2$. Fissiamo un arco $e \in E$. Allora

$$\begin{aligned} P(e \text{ interno in } V') &= \frac{\binom{n-2}{r-2}}{\binom{n}{r}} \\ &= \frac{(n-2)!}{(r-2)!(n-r)!} \cdot \frac{r!(n-r)!}{n!} \\ &= \frac{(r-1)r}{n(n-1)} \\ &\leq \frac{r^2}{n^2} \qquad \text{(se } r < n\text{)} \end{aligned}$$

Si definisca la v.a. indicatrice $X_e = 1 \Leftrightarrow e$ è interno a V'. Allora, per quanto visto prima, $P(X_e = 1) \leq \frac{r^2}{n^2}$. Sia $X = \sum_{e \in E} X_e$. La probabilità che l'algoritmo non fallisca è equivalente alla probabilità che non vi siano archi interni all'insieme indipendente:

$$P(X = 0) = 1 - P(X \geq 1)$$

e

$$P(X \geq 1) = P(\exists e \in E \mid X_e = 1) \leq mP(X_e = 1) \leq \frac{mr^2}{n^2}$$

grazie all'applicazione dello *union bound*. Lo stesso risultato è possibile ottenerlo applicando il lemma di Markov[1] (lemma 3.2):

$$P(X \geq 1) \leq \frac{E[X]}{1} = E[X] = mE[X_e] = mP(X_e = 1) \leq \frac{mr^2}{n^2}.$$

Proseguendo

$$P(X = 0) = 1 - P(X \geq 1) \geq 1 - \frac{mr^2}{n^2} \overset{?}{\geq} \frac{1}{2}$$

che avviene quando

$$r^2 \leq \frac{n^2}{2m} \Leftrightarrow r \leq \sqrt{\frac{n^2}{2m}} = \frac{n}{\sqrt{2m}} \Rightarrow r_{max} = \lfloor n/\sqrt{2m} \rfloor.$$

[1] Si noti che per $T = 1$ il lemma di Markov fornisce lo *union bound*.

Il risultato ottenuto non è strabiliante. Infatti, se consideriamo un grafo denso dove $m = \Theta(n^2)$ ($m = cn^2, c < 1$) si ottiene

$$r = \lfloor n/n\sqrt{2c} \rfloor = \lfloor 1/\sqrt{2c} \rfloor = O(1).$$

Nel caso sparso invece, dove $m = O(n)$, si ottiene un buon risultato:

$$r = \lfloor n/\sqrt{2m} \rfloor \geq \lfloor n/\sqrt{2cn} \rfloor = \lfloor \sqrt{n}/\sqrt{2c} \rfloor = \Omega(\sqrt{n}).$$

Dov'è che l'algoritmo non si comporta intelligentemente? L'algoritmo tira a caso r vertici e fallisce se vi è un arco interno a questi r vertici: questo non è un atteggiamento particolarmente furbo perché costringe a considerare un n piccolo se si ottenere una probabilità che non vi sia un arco interno all'insieme dei nodi selezionati che è abbastanza grande.

Approccio Las Vegas In questo secondo approccio forniamo un algoritmo Las Vegas (algoritmo 28). L'idea è quella di effettuare un *sampling* S dell'insieme dei nodi V, selezionando ogni nodo di V e aggiungendolo ad S con probabilità p. Successivamente effettuiamo una "pulitura" dell'insieme S: per ogni arco di E, se l'arco è interno ad S elimino da tale insieme uno dei due nodi (in questo modo l'arco diventa esterno). Alla fine della fase di pulitura l'insieme ottenuto è un insieme indipendente in quanto tutti gli archi interni sono stati eliminati.

Algorithm 28 Algoritmo Las Vegas per INDIPENDENT-SET

```
function IS2(G(V, E), p)
    S ← ∅
    for i ← 1 to |V| do
        ** aggiungi v_i a S con probabilità p **
    end for
    for j ← 1 to |E| do
        if e_j = {u, v} interno a S then
            S ← S − {u}
        end if
    end for
    return S
end function
```

Sia X la v.a. rappresentante il numero di nodi selezionati dalla fase di sampling: $E[X] = np$. Sia $Y = \sum_{e \in E} Y_e$ la v.a. che rappresenta il numero di elementi eliminati dalla fase di *cleaning*, con $Y_e = 1$ se e solo se l'arco e è interno a S all'inizio della fase di pulitura. L'analisi non cattura il fatto che un arco inizialmente interno possa diventare esterno mano a mano che la fase di pulitura avanza, conseguentemente l'algoritmo andrà meglio (in generale) rispetto a quanto verrà evidenziato dall'analisi. In sostanza, la probabilità che un arco sia interno al *sample* prima della fase di pulitura è più piccola della probabilità che esso sia interno anche alla fine di tale fase perché questo arco potrebbe essere stato rimosso durante il processo di pulitura. Proprio per questo

motivo possiamo scrivere la seguente maggiorazione:

$$\begin{aligned} P(Y_e = 1) &\leq P(e = \{u, v\} \text{ interno alla fine della fase di sampling}) \\ &= P(u, v \text{ vengano selezionati durante il sampling}) \\ &= p^2 \end{aligned}$$

poiché la selezione dei nodi è indipendente. Allora $E[Y] = E[\sum_{e \in E} Y_e] \leq mp^2$, da cui $E[|S|] = E[X - Y] \geq np - mp^2$.

Studiando la funzione $f(p) = np - mp^2$ in $[0, 1]$ (essendo una distribuzione di probabilità) osservo che tale funzione è una parabola e che ha un massimo in $p = n/2m$. Allora

$$\begin{aligned} E[|S|]|_{p=\frac{n}{2m}} &\geq n\frac{n}{2m} - m\frac{n^2}{4m^2} \\ &= \frac{n^2}{4m} \\ &= \Theta(\lfloor n/\sqrt{2m} \rfloor^2). \end{aligned}$$

Con questo algoritmo si ottiene un insieme indipendente di cardinalità quadratica rispetto alla versione Montecarlo. Le prestazioni nel caso di grafi sparsi rimangono equivalenti mentre otteniamo un netto miglioramento nel caso di grafi densi poiché $n^2/4m = \Theta(n)$.

4.3.2 Somma di variabili aleatorie geometriche

Sia data una sequenza $Z_1, Z_2, \cdots, Z_n$ di variabili aleatorie geometriche[2] indipendenti e identicamente distribuite. Forniremo un bound sulla coda della distribuzione $Z = \sum_{i=1}^{k} Z_i$: segue immediatamente che $E[Z] = \sum_{i=1}^{k} E[Z_i] = k/p = \mu$.

Effettuo un cambio dello spazio di probabilità portandoci nel mondo dei lanci di monete. In particolare eseguo il seguente esperimento (p è la probabilità che la moneta restituisca una testa):

1. lancio una moneta fino alla prima testa e associo tale esperimento alla v.a. Z_1;
2. lancio una moneta fino alla prima testa e associo tale esperimento alla v.a. Z_2;
3. ...
4. lancio una moneta fino alla prima testa e associo tale esperimento alla v.a. Z_k.

Si vuole maggiorare $P(Z > t\mu)$ per $t > 1$. L'evento "$Z > t\mu$" implica l'evento E_Z ="nei primi $t\mu$ lanci di moneta sono uscite meno di k teste", pertanto

[2]Una variabile aleatoria geometrica $Z \sim Geom(p)$ è tale per cui $P(Z = j) = p(1-p)^{j-1}$, con $j \geq 1$. Inoltre la sua media vale $E[Z] = 1/p$.

$P(Z > t\mu) \leq P(E_Z)$. Si considerino le v.a. indicatrici $X_1, X_2, \cdots, X_{t\mu}$. Allora

$$P(E_Z) = P\left(\sum_{j=1}^{t\mu} X_j < k\right)$$

$$E[X] = E\left[\sum_{j=1}^{t\mu} X_j\right] = t\mu p = tk \Rightarrow k = E[X]/t$$

Quindi, essendo $1/t < 1$ e dovendo esprimere $1/t = 1-\delta$, ponendo $\delta = 1-1/t$ e applicando il terzo bound di Chernoff si ottiene

$$\begin{aligned} P(X < k) &= P(X < E[X]/t) \\ &= P(X < (1-\delta)E[x]) \\ &< e^{-\frac{\delta^2}{2}E[X]} \qquad (Chernoff) \\ &= e^{-\frac{1}{2}\left(1-\frac{1}{t}\right)^2 kt} \\ &> P(Z > t\mu) \end{aligned}$$

Si osservi che nel bound non compare la probabilità p. Quando $p \to 1$ allora $P(Z > t\mu) \to 0$ (bastano k lanci): il bound però "non garantisce" ciò ma questo non è un problema, semplicemente il bound non è ottimale per $p \to 1$.

Nella successiva sezione vedremo un esempio di applicazione di questo bound nel caso di p costante.

4.3.3 Ricerca binaria tollerante ai guasti in lettura

Sia dato un vettore ordinato A e si voglia effettuare una ricerca binaria su di esso. L'aleatorietà in questo problema è data dal possibile fallimento di una lettura: in particolare, la procedura A.read(i) restituirà la componente $A[i]$ con $p = 1/2$ e restituirà ERROR con probabilità $1 - p$. L'algoritmo che proponiamo in 29 è un algoritmo Las Vegas. La correttezza dell'algoritmo segue immediatamente da quella dell'algoritmo di ricerca binaria e dal fatto che il ciclo dell'algoritmo non è infinito. La struttura delle chiamate è in ogni caso una catena con le istanze che mano a mano vengono al più dimezzate. L'algoritmo è lineare, con il caso peggiore rappresentato dalla ricerca senza successo.

La lettura dell'elemento di mezzo del vettore per ogni chiamata può essere rappresentato tramite una v.a. geometrica: ogni variabile Z_i rappresenterà il numero di letture effettuate al livello i-esimo. Poiché ad ogni livello la chiamata ricorsiva accetta in ingresso una istanza di taglia al più dimezzata, il numero di chiamate totali sarà al più $\log_2 n$. Considerato come lavoro dell'algoritmo la lettura della componente del vettore, la somma di tutte queste v.a. restituirà il lavoro complessivo dell'algoritmo. Pertanto

$$E\left[\sum_{i=0}^{\log_2 n - 1} Z_i\right] = \log_2 n \frac{1}{p} = 2\log_2 n$$

da cui segue, per l'analisi svolta nella sezione precedente,

$$P\left(\sum_{i=0}^{\log_2 n - 1} Z_i > t2\log_2 n\right) < e^{-\left(1-\frac{1}{t}\right)^2 \frac{t}{2}\log_2 n} \stackrel{?}{>} \frac{1}{n}$$

Algorithm 29 Algoritmo Las Vegas per la ricerca binaria

```
function SEARCH(A, i, j, k)
    if i > j then
        return NOT FOUND
    end if
    m ← ⌊(i + j)/2⌋
    temp ← A.read(m)
    while temp = ERROR do
        temp ← A.read(m)
    end while
    if temp = k then
        return m
    end if
    if temp > k then
        return SEARCH(A, i, m − 1, k)
    else
        return SEARCH(A, m + 1, j, k)
    end if
end function
```

Dimensionando opportunamente t si ottiene la garanzia voluta. Ad esempio, per $t = 4$:

$$e^{-\frac{9}{16}\cdot\frac{4}{2}\log_2 n} = e^{-\frac{9}{8}\log_2 n} < e^{-\ln n} = \frac{1}{n}.$$

4.3.4 Scostamento dalla media di v.a. con valori in $\{-1, +1\}$

Siano date n variabili aleatorie $X_1, X_2, \cdots, X_n$, i.i.d., con $X_i \in \{-1, +1\}$ e con $P(X_i = -1) = P(X_i = 1) = p = 1/2$. Si vuole studiare la variabile aleatoria $X = \sum_{i=1}^{n} X_i$: in particolare si vuole trovare un bound per $P(X \geq \delta)$, con $\delta > 0$, e si vuole determinare un $\bar{\delta}$ tale che $P(X \geq \bar{\delta}) < 1/n$.

Per far ciò trasformo le variabili aleatorie X_i in variabili aleatorie indicatrici Y_i. Effettuo questo tramite una trasformazione affine:

$$Y_i = \frac{X_i + 1}{2} \Rightarrow P(Y_i = 0) = P(Y_i = 1) = p = \frac{1}{2};$$

poiché la funzione applicata è lineare, le v.a. Y_i rimangono mutuamente indipendenti.

Allora

$$\begin{aligned}
P(X \geq \delta) &= P\left(\sum_{i=1}^{n} X_i \geq \delta\right) \\
&= P\left(\sum_{i=1}^{n} (2Y_i - 1) \geq \delta\right) \\
&= P\left(\sum_{i=1}^{n} Y_i \geq \frac{n+\delta}{2}\right) \\
&= P\left(\sum_{i=1}^{n} (2Y_i - 1) \geq \mu\left(1 + \frac{\delta}{2\mu}\right)\right) \\
&< e^{-\left(\frac{\delta}{2\mu}\right)^2 \frac{\mu}{3}} \\
&= e^{-\frac{\delta^2}{6n}} \\
&\overset{?}{<} \frac{1}{n}
\end{aligned}$$

e ponendo

$$\frac{\delta^2}{6n} = \ln n \Leftrightarrow \bar{\delta} = \sqrt{6n \ln n}$$

si ottiene quanto richiesto.

4.3.5 (ϵ, δ)-approssimazione

Si voglia testare la realizzazione di un esperimento: testa la realizzazione di un evento E_v, con $P(E_v) = v$, $0 < v < 1$ (escludiamo i casi degeneri). Si vuole stimare v eseguendo un certo numero di volte $m = m_{\epsilon,\delta}$ l'esperimento. Si vuole inoltre limitare superiormente tale numero in modo che garantisca che la stima X di v sia tale per cui

$$P\left(\frac{|X - v|}{v} > \epsilon\right) < \delta, \qquad 0 < \epsilon, \delta < 1$$

L'algoritmo è proposto in 30. La procedura EXPERIMENT(), che rappresenta l'esperimento che realizza l'evento E_v, restituisce 1 se ha avuto successo e 0 altrimenti.

Algorithm 30 Algoritmo di (ϵ, δ)-approssimazione

```
function ESTIMATE(m)
    COUNT ← 0
    for i ← 1 to m do
        b ← EXPERIMENT()
        COUNT ← COUNT + b
    end for
    return COUNT/m
end function
```

Si ha

$$\begin{aligned}\left(\frac{|X-v|}{v}>\epsilon\right) &= P\left(\frac{|\frac{COUNT}{m}-v|}{v}>\epsilon\right)\\ &= P\left(\frac{|COUNT-mV|}{mV}>\epsilon\right)\\ &\geq P(COUNT>(1+\epsilon)mV)+P(COUNT<(1-\epsilon)mV)\\ &< e^{-\frac{\epsilon^2 mV}{3}}+e^{-\frac{\epsilon^2 mV}{2}}\\ &< 2e^{-\frac{\epsilon^2 mV}{3}}\\ &\overset{?}{<}\delta\end{aligned}$$

da cui

$$\begin{aligned}2e^{-\frac{\epsilon^2 mV}{3}} &< \delta\\ e^{-\frac{\epsilon^2 mV}{3}} &< \frac{\delta}{2}\\ e^{\frac{\epsilon^2 mV}{3}} &> \frac{2}{\delta}\\ \frac{\epsilon^2 mV}{3} &> \ln\frac{2}{\delta}\\ m_{\epsilon,\delta} &> \frac{3}{\epsilon^2}\ln\frac{2}{\delta}\left(\frac{1}{v}\right)\end{aligned}$$

4.3.6 Generazione di bit senza bias

Si voglia generare dei bit senza bias a partire da bit con bias. Si consideri la procedura BIAS() che restituisce 1 con probabilità p e 0 con probabilità $1-p$. Effettuando due lanci di moneta, posso ottenere i seguenti risultati:

$$\begin{aligned}&00 \text{ con probabilità } (1-p)^2\\ &11 \text{ con probabilità } p^2\\ &10 \text{ con probabilità } p(1-p)\\ &01 \text{ con probabilità } (1-p)p\end{aligned}$$

osservando che le ultime due configurazioni sono equiprobabili pur partendo da una moneta con bias.

In 31 è presentato l'algoritmo per la generazione di bit senza bias a partire da bit con bias. In particolare tale algoritmo termina con probabilità 1, ovvero non è possibile che l'algoritmo non termini a causa di un loop infinito.

Proposizione. *L'algoritmo di generazione di bit senza bias termina con probabilità 1.*

Si definiscano gli eventi

$$A_i = \text{"UNBIAS() termina all'}i\text{-esima iterazione"} \quad i \geq 1$$

Algorithm 31 Algoritmo per la generazione di bit senza bias

```
function UNBIAS
    repeat
    a ← BIAS()
    b ← BIAS()
    until a ≠ b
    return a
end function
```

e si osservi che per tali eventi

$$A_i \cap A_j = \emptyset \quad i \neq j$$
$$\bigcup_{i=1}^{\infty} A_i = \Omega$$

Allora

$$\begin{aligned} P(\text{UNBIAS() ritorni 0}) &= \sum_{i=1}^{\infty} P(\text{UNBIAS() ritorni 0}|A_i)P(A_i) \\ &= \frac{1}{2}\sum_{i=1}^{\infty} P(A_i) \\ &= \frac{1}{2} \end{aligned}$$

dimostrando così che il risultato di UNBIAS() è senza bias.

La probabilità che l'iterazione termina il repeat è

$$\begin{aligned} P(\text{"iterazione termina il repeat"}) &= P(a = 0 \ \wedge \ b = 1) + P(a = 1 \ \wedge \ b = 0) \\ &= (1-p)p + p(1-p) \\ &= 2p(1-p) \end{aligned}$$

ed è una v.a. geometrica (si cerca la probabilità del primo successo dopo una serie di soli insuccessi), quindi

$$E[\text{numero iterazioni}] = \frac{1}{2p(1-p)}.$$

Infine

$$E[\text{UNBIAS()}] = 2E[\text{numero iterazioni}] = \frac{1}{p(1-p)}.$$

4.4 Randomizzazione e approssimazione

Un algoritmo $A(i)$ è un algoritmo randomizzato di ρ-approssimazione se

$$\max\left\{\frac{E[c(A(i))]}{c(S^*)}, \frac{c(S^*)}{E[c(A(i))]}\right\} \leq \rho.$$

4.4.1 Il problema MAX-SAT

Il problema MAX-SAT è definito come segue

$$\begin{cases} I: \ \langle \phi(x_1, \cdots, x_2), k \rangle \ : \ \phi(x_1, \cdots, x_2) = c_1 \wedge \cdots \wedge c_m \in 3-CNF, 0 \leq k \leq m \\ \\ D: \ \exists \vec{b} \in \{0,1\}^n \mid \phi(\vec{b}) \text{ contiene almeno } k \text{ clausole vere?} \end{cases}$$

È banale dimostrare che MAX-SAT è NP-hard sfruttando la riduzione da 3-CNF-SAT ponendo $k = m$.

Un possibile algoritmo randomizzato di approssimazione è descritto in 32.

Algorithm 32 Algoritmo per MAX-SAT

```
function APPROX_MAX-SAT(φ(x_1, ··· , x_2))
    ** sia φ(x_1, ··· , x_2) = c_1 ∧ ··· ∧ c_m **
    for i ← 1 to n do
        b_i ←RANDOM({0, 1})
    end for
    count ← 0
    for j ← 1 to m do
        if c_j(b⃗) = 1 then
            count ← count + 1
        end if
    end for
    return count
end function
```

Dall'algoritmo segue che

$$\rho(\phi) = \frac{\text{"massimo numero di clausole soddisfacibili"}}{E[count]} \leq \frac{m}{E[count]}.$$

Ancora una volta ci appoggiamo ad un v.a. indicatrice X_i: essa vale 1 se la clausola c_i è soddisfatta dall'assegnamento (randomico) $\vec{b}_R$, per $1 \leq i \leq m$. Segue quindi che $count = \sum_{i=1}^{m} X_i$. Allora

$$\begin{aligned} E[count] &= E\left[\sum_{i=1}^{m} X_i\right] \\ &= \sum_{i=1}^{m} E[X_i] \\ &= \sum_{i=1}^{m} P(X_i = 1) \\ &= \sum_{i=1}^{m} P(c_i \text{ soddisfatta da } \vec{b}_R). \end{aligned}$$

Si ha

$$\begin{aligned} P(c_i \text{ soddisfatta da } \vec{b}_R) &= 1 - P(c_i \text{ non è soddisfatta da} \vec{b}_R) \\ &= 1 - P(y_i^1 = 0)P(y_i^2 = 0)P(y_i^3 = 0) \\ &= 1 - \frac{1}{8} \\ &= \frac{7}{8} \end{aligned}$$

essendo $(P(y_i^j = 0) = 1/2)$, per $1 \leq i \leq m$ e $1 \leq j \leq 3$.

Riprendendo la sequenza di uguaglianze precedente si ottiene

$$\begin{aligned} E[count] &= \sum_{i=1}^{m} P(c_i \text{ soddisfatta da } \vec{b}_R) \\ &= \sum_{i=1}^{m} \frac{7}{8} \\ &= \frac{7}{8}m. \end{aligned}$$

Segue quindi che

$$\rho(\phi) \leq \frac{m}{\frac{7}{8}m} = \frac{8}{7}.$$

4.4.2 Il problema MAX-CUT

Si fornisce la seguente definizione.

Definizione. *Dato un grafo $G = (V, E)$ non orientato, un NODE-CUT è una partizione di V, $(W, V - W)$, con $W \subseteq V$. La taglia del node-cut è*

$$s(W, V - W) = |\{e = \{u, v\} \in E \mid (u \in W) \wedge (v \in V - W)\}|.$$

Il problema MAX-CUT è un problema di ottimo che richiede, dato un grafo, di trovare il node-cut di taglia massima. Si può osservare che l'algoritmo di Karger restituisce sempre un node-cut e in particolare esso è un node-cut di taglia minima. Si può dimostrare che il problema MAX-CUT è un problema NP-hard.

Algoritmo di 2-approssimazione Un possibile approccio risolutivo basato sull'approssimazione parte da una semplice idea: considero il node-cut iniziale vuoto, ovvero $(\emptyset, V)$ e man mano miglioro il taglio corrente con una strategia di *local search*, spostando un nodo da una parte all'altra della partizione. In 33 è proposto lo pseducodice dell'algoritmo. L'algoritmo non presenta un loop infinito poiché la funzione di costo è monotona crescente; inoltre al massimo l'algoritmo esegue m iterazioni. Il taglio ottimo è quindi $s(W^*, V - W^*) \leq m$. Dimostrando che l'algoritmo restituisce un taglio pari ad almeno $m/2$ si ottiene la tesi per cui l'algoritmo proposto è di 2-approssimazione.

Proposizione. *Considerato un nodo $v \in W$, e sia $d(v)$ il grado di v, si ha che $t_v \geq d(v)/2$ vicini di v sono in $V - W$.*

Algorithm 33 Algoritmo di 2-approssimazione per MAX-CUT

function DET_APPROX_MAX-CUT($G(V, E)$)
  $W \leftarrow \emptyset$
  while true do
    if $(\exists v \in W \mid s(W - \{v\}, V - W \cup \{v\}) > s(W, V - W)) \ \vee \ (\exists v \in V - W \mid s(W \cup \{v\}, V - W - \{v\}) > s(W, V - W))$ **then**
      ** sposto il nodo **
    else
      exit
    end if
  end while
  return $s(W, V - W)$
end function

Dimostrazione. Per assurdo, suppongo che $t_v < d(v)/2 \Leftrightarrow 2t_v < d(v)$. Considero il taglio $(W', V - W') = (W - \{v\}, V - W \cup \{v\})$. Allora

$$\begin{aligned} s(W', V - W') &= s(W, V - W) - t_v + (d(v) - t_v) \\ &= s(W, V - W) + (d(v) - 2t_v) \\ &> s(W, V - W) \end{aligned}$$

tuttavia ciò non è possibile perché l'algoritmo, se c'è qualcosa di migliorabile, non terminerebbe. □

La proposizione è ovviamente simmetrica e quindi vale anche per l'altra parte del taglio. Da questa proposizione segue

$$\begin{aligned} s(W, V - W) &= \frac{1}{2} \sum_{v \in V} t_v \\ &\geq \frac{1}{2} \sum_{v \in V} \frac{d(v)}{2} \\ &= \frac{1}{2} \cdot \frac{2|E|}{2} \\ &= \frac{m}{2} \end{aligned}$$

da cui

$$\rho = \frac{s(W^*, V - W^*)}{s(W, V - W)} \leq \frac{m}{\frac{m}{2}} = 2.$$

Approccio randomizzato In 34 è proposto un algoritmo randomizzato per MAX-CUT.

Per analizzare l'algoritmo proposto introduciamo la v.a. $X_e = 1$, $\forall e = \{u, v\} \in E$, se

$$\{[(u \in W) \ \wedge \ (v \in V - W)] \ \vee \ [(u \in V - W) \ \wedge \ (v \in W)]\}.$$

Allora, dalla definizione di node-cut, segue che

$$s(W, V - W) = \sum_{e \in E} X_e$$

Algorithm 34 Algoritmo randomizzato per MAX-CUT

```
function RANDOM_APPROX_MAX-CUT(G(V, E))
    W ← ∅
    for i ← 1 to |V| do
        b ← RANDOM({0, 1})
        if b = 0 then
            W ← W ∪ {v_i}
        end if
    end for
    return s(W, V − W)
end function
```

da cui

$$E[s(W, V - W)] = \sum_{e \in E} E[X_e] = \sum_{e \in E} P(X_e = 1)$$

dove

$$\begin{aligned} P(X_e = 1) &= P([(u \in W) \ \wedge \ (v \in V - W)] \ \vee \ [(u \in V - W) \ \wedge \ (v \in W)]) \\ &= P([(b_u = 0) \ \wedge \ (b_v = 1)] \ \vee \ [(b_u = 1) \ \wedge \ (b_v = 0)]) \\ &= P(b_u = 0)P(b_v = 1) + P(b_u = 1)P(b_v = 0) \\ &= \frac{1}{2} \cdot \frac{1}{2} + \frac{1}{2} \cdot \frac{1}{2} \\ &= \frac{1}{2}. \end{aligned}$$

Da ciò segue che $E[s(W, V - W)] = m/2$, ovvero

$$\rho = \frac{c(S^*)}{E[c(A(i))]} \leq \frac{m}{\frac{m}{2}} = 2.$$

L'algoritmo proposto ha quindi la stessa qualità dell'algoritmo dell'algoritmo precedente tuttavia permette di risparmiare tempo in quanto esegue una semplice scansione lineare.