

Complementi di Algoritmi

Massimo Perego

Indice

1 Complessità Computazionale	2
1.1 Riducibilità polinomiale tra problemi di decisione	2
1.2 Classi \mathcal{P} e \mathcal{NP}	7
1.3 Problemi \mathcal{NP} -completi	8
2 Algoritmi Probabilistici	14
2.1 Algoritmi Montecarlo e Las Vegas	14
2.2 Classi di complessità probabilistiche	19
2.3 Estrattore di von Neumann	23
2.4 Il problema del Coupon Collector	24
2.5 Reservoir Sampling	26
2.6 Algoritmo di Karger per il taglio minimo	28
3 Hashing e Randomizzazione	34
3.1 Hashing Universale	34
3.2 Conteggio approssimato	39
3.3 Proiezioni casuali	44
4 Clustering e Randomizzazione	50
4.1 Correlation Clustering	50
4.2 k -Means	54

Capitolo 1

Complessità Computazionale

1.1 Riducibilità polinomiale tra problemi di decisione

Un **problema di decisione** X su $\{0, 1\}$ è una coppia (\mathcal{I}, q) dove

- $\mathcal{I} \subseteq \{0, 1\}^*$ è l'insieme delle descrizioni (come stringhe binarie) delle **istanze valide del problema**
- $q : \mathcal{I} \rightarrow \{0, 1\}$ è la **funzione di decisione**, che associa ogni istanza $I \in \mathcal{I}$ al suo valore, ovvero $q(I) = 1$ se positiva o $q(I) = 0$ se negativa

Un algoritmo A risolve un problema $X = (\mathcal{I}, q)$ se restituisce il valore corretto della funzione q su ogni istanza $I \in \mathcal{I}$.

Indichiamo con $|I|$ la **lunghezza della codifica** in bit di un'istanza $I \in \mathcal{I}$. In genere, non specifichiamo il modo con il quale un'istanza viene rappresentata come una stringa di bit, ma resta inteso che le istanze vengano rappresentate secondo rappresentazioni “ragionevoli” (e.g. lista di adiacenza per un grafo).

Se A è un algoritmo che risolve il problema $X = (\mathcal{I}, q)$, indichiamo con $T_A(I)$ il tempo di calcolo di A quando l'input è $I \in \mathcal{I}$. Definiamo ora

$$T_A(n) = \max \{T_A(I) \mid |I| \leq n\}$$

ovvero il massimo dei tempi di calcolo per istanze $I \in \mathcal{I}$ di lunghezza al più n . Diciamo che A risolve X in tempo polinomiale se esiste un intero k tale che $T_A(n) = \mathcal{O}(n^k)$.

Un algoritmo ha accesso a un **oracolo** per un problema $X = (\mathcal{I}, q)$ quando può ottenere in tempo unitario il valore $q(I)$ per istanze arbitrarie $I \in \mathcal{I}$.

Un problema Y è **polynomialmente riducibile** a un altro problema X , denotato con $Y \preceq_p X$ quando

1. Esiste un algoritmo che risolve Y in tempo polinomiale con accesso a un oracolo per X
2. L'oracolo viene interrogato una sola volta e l'algoritmo termina fornendo la risposta dell'oracolo

Questa nozione di riducibilità polinomiale è anche nota come riducibilità secondo Karp. Esiste anche una nozione più generale, nota come riducibilità secondo Turing, dove l'algoritmo può accedere all'oracolo un numero polinomiale di volte.

Equivalentemente, la riducibilità polinomiale (secondo Karp) può essere vista come “l'input di Y può essere trasformato in tempo polinomiale in input di X ” (per poi risolvere il problema in tempo unitario grazie all'oracolo, coincide con la prima definizione).

Se $Y \preceq_p X$ ed esiste un algoritmo che risolve X in tempo polinomiale, allora esiste un algoritmo che risolve Y in tempo polinomiale (la polinomialità è robusta rispetto alla composizione). Viceversa, se $Y \preceq_p X$ e non esiste un algoritmo che risolve Y in tempo polinomiale, allora non esiste un algoritmo che risolve X in tempo polinomiale. Intuitivamente, $Y \preceq_p X$ significa che il problema Y non è *più difficile* del problema X . Se $X \preceq_p Y$ e $Y \preceq_p X$, ovvero X e Y sono polynomialmente riducibili l'uno con l'altro, allora scriviamo $X \equiv_p Y$.

Sia $G = (V, E)$ un grafo semplice (non diretto, non pesato, senza loop e archi multipli). Un **insieme indipendente** (independent set) in G è un sottoinsieme $S \subseteq V$ di vertici non adiacenti, ovvero $\forall i, j \in S, (i, j) \notin E$. Una **copertura** (vertex cover) di G è un sottoinsieme $S' \subseteq V$ di vertici tali che ogni arco di G ha almeno un estremo in S' , ovvero $\forall (i, j) \in E$ esiste $k \in S'$ tale che $k = i$ o $k = j$.

Fatto 1.1.1. Sia $G = (V, E)$ un grafo semplice. Allora S è un insieme indipendente se e solo se $S' = V \setminus S$ è una copertura.

Dimostrazione. Sia S un insieme indipendente e sia $S' = V \setminus S$. Allora dato un qualsiasi $(i, j) \in E$, deve valere $i \notin S$ oppure $j \notin S$. Quindi $i \in S'$ oppure $j \in S'$, da cui ne segue che S' è una copertura di G .

Viceversa, sia S' una copertura di G e $S = V \setminus S'$. Allora dati $i, j \in S$ arbitrari $(i, j) \notin E$, altrimenti allora S' non sarebbe una copertura; da questo otteniamo che S è un independent set. \square

Sia *Independent Set* il problema di decisione le cui istanze sono coppie $I = (G, k)$ dove G è un grafo semplice e k è un intero. La funzione di decisione q_{IS} è tale che $q_{IS}(I) = 1$ se e solo se G contiene un insieme indipendente di taglia almeno k . Le istanze del problema **Vertex Cover** sono le stesse di Independent Set, ma la funzione di decisione q_{VC} è tale che $q_{VC}(I) = 1$ se e solo se G contiene una copertura di taglia al più k . Conseguenza immediata del Fatto 1.1.1 è che Independent Set e Vertex Cover sono polinomialmente riducibili l'uno con l'altro.

Corollario 1.1.1. *Independent Set* \equiv_p *Vertex Cover*

La relazione \equiv_p gode di transitività.

Fatto 1.1.2. Se $Z \preceq_p Y$ e $Y \preceq_p X$, allora $Z \preceq_p X$.

Dimostrazione. Dato un oracolo per X possiamo risolvere un'istanza di Z nel modo seguente: eseguiamo l'algoritmo per risolvere Z usando l'oracolo per Y , ma ogni volta che dovrebbe essere invocato l'oracolo di Y , questo viene simulato eseguendo l'algoritmo per risolvere Y , il quale usa l'oracolo per X . \square

Introduciamo ora il problema di decisione **Set Cover**: le istanze sono della forma $I = (U, \mathcal{S}, k)$ dove U è un insieme finito, $\mathcal{S} \equiv \{S_1, \dots, S_m\}$ è una collezione di sottoinsiemi di U e k è un intero. Allora $q_{SC}(I) = 1$ se e solo se esistono al più k sottoinsiemi in \mathcal{S} tali che la loro unione sia tutto U .

Fatto 1.1.3. *Vertex Cover* \preceq_p *Set Cover*

Dimostrazione. Supponiamo di avere un oracolo per Set Cover e definiamo un algoritmo polinomiale per Vertex Cover.

Data un'istanza $I = (G, k)$ di Vertex Cover con $G = (V, E)$ costruiamo un'istanza I' di Set Cover dove $U \equiv E$ e $\mathcal{S} \equiv \{S_i \mid i \in V\}$ dove S_i è l'insieme degli archi di G incidenti su i (ottenibile in tempo lineare partendo dalla descrizione di G). Quindi ogni $u \in U$ è contenuto in esattamente due elementi di \mathcal{S} .

Verifichiamo ora che U è l'unione di al più k insiemi $S_1, \dots, S_{|V|}$ se e solo se G ha una copertura con vertici di taglia al più k .

Supponiamo che l'unione di S_{i_1}, \dots, S_{i_r} sia U , con $r \leq k$. Allora, ogni arco in G è incidente a uno dei vertici i_1, \dots, i_r . Quindi $\{i_1, \dots, i_r\}$ è una copertura con vertici di taglia al più k . Viceversa, se $\{i_1, \dots, i_r\}$ è una copertura con vertici di taglia al più k allora la gli insiemi corrispondenti S_{i_1}, \dots, S_{i_r} hanno U come unione.

Quindi possiamo definire un algoritmo che implementa la funzione di decisione q_{VC} per Vertex Cover nel modo seguente: data un'istanza di I Vertex Cover, l'algoritmo costruisce in tempo polinomiale un'istanza I' di Set Cover nel modo descritto sopra. Quindi chiama l'oracolo un'unica volta per ottenere $q_{SC}(I')$ che viene restituita in output. \square

Così come possiamo vedere Set Cover come una generalizzazione di Vertex Cover, introduciamo ora Set Packing come generalizzazione di Independent Set. Le istanze di **Set Packing** sono le stesse di Set Cover, ovvero $I = (U, \mathcal{S}, k)$ dove U è un insieme finito, $\mathcal{S} \equiv \{S_1, \dots, S_m\}$ è una collezione di sottoinsiemi di U e k è un intero. la funzione di decisione q_{SP} è tale che $q_{SP} = 1$ se e solo se esistono almeno k sottoinsiemi in \mathcal{S} i quali sono a due a due disgiunti. Anche se Set Packing è apparentemente più generale di Independent Set, si può dimostrare che i due problemi sono equivalenti.

Fatto 1.1.4. *Independent Set \equiv_p Set Packing.*

Dimostrazione. Data un'istanza $\mathcal{S} \equiv \{S_1, \dots, S_m\}$ di Set Packing costruiamo in tempo polinomiale un grafo $G = (V, E)$ dove $V \equiv \{v_S \mid S \in \mathcal{S}\}$ e $(v_S, v_T) \in E$ se e solo se $S \cap T \neq \emptyset$. Allora ogni insieme indipendente in G corrisponde a un packing della stessa taglia.

Viceversa, dato un grafo $G = (V, E)$ possiamo costruire in tempo polinomiale la collezione $\mathcal{S} \equiv \{S_i \mid i \in V\}$ dove S_i è l'insieme degli archi di G incidenti su i . Allora ogni packing in \mathcal{S} corrisponde a un insieme indipendente in G della stessa taglia. \square

In molte discipline bisogna spesso risolvere problemi di ottimizzazione combinatoria vincolata. In questi problemi si cerca un assegnamento di valori per un insieme di variabili discrete in modo da soddisfare un dato insieme di vincoli. In astratto, problemi di questo tipo sono formulati come problemi di soddisfacibilità su variabili booleane.

Sia \mathcal{X} un insieme di variabili booleane x_1, \dots, x_n . Un assegnamento di valori di verità a \mathcal{X} è una funzione $\pi : \mathcal{X} \rightarrow \{0, 1\}$. Un letterale ℓ_i è la variabile x_i o la sua negazione \bar{x}_i . Una clausola $C = \ell_{i_1} \vee \dots \vee \ell_{i_k}$ è una disgiunzione di letterali. Un assegnamento π soddisfa una clausola C se e solo se c'è almeno un letterale della forma $\ell_i = x_i$ e $\pi(x_i) = 1$ oppure $\ell_j = \bar{x}_j$ e $\pi(x_j) = 0$.

Sia **SAT** il problema di decisione le cui istanze I sono insiemi di clausole \mathcal{C} su un insieme \mathcal{X} di variabili booleane. Allora $q(I) = 1$ se e solo se esiste un assegnamento $\pi : \mathcal{X} \rightarrow \{0, 1\}$ tale che soddisfi tutte le clausole in \mathcal{C} .

Una versione ridotta di SAT è **3-SAT**, le cui istanze sono insiemi di clausole ciascuna contenente esattamente 3 letterali. Esempio di istanza di 3-SAT:

- $\mathcal{X} \equiv \{x_1, x_2, x_3, x_4\}$
- $\mathcal{C} \equiv \{(\bar{x}_1 \vee x_2 \vee \bar{x}_3), (\bar{x}_2 \vee x_3 \vee x_4), (x_1 \vee x_2 \vee \bar{x}_4)\}$

Un assegnamento che soddisfa tutte le clausole è $\pi(x_i) = 0$ per $i = 1, \dots, 4$.

Teorema 1.1.1. $3\text{-SAT} \preceq_p \text{Independent Set}$.

Dimostrazione. Una definizione equivalente di 3-SAT: \mathcal{C} è soddisfacibile se e solo se è possibile scegliere esattamente un letterale in ciascuna clausola in modo tale che tra i letterali scelti non compaiano simultaneamente x_i e \bar{x}_i per nessuna delle variabili $x_i \in \mathcal{X}$.

Data un'istanza $I = \mathcal{C}$ di 3-SAT, costruiamo un'istanza $I' = (G, |\mathcal{C}|)$ di Independent Set tale che $q_{3SAT}(I) = q_{IS}(I')$. Sia $k = |\mathcal{C}|$. Il grafo $G = (V, E)$ ha $3k$ vertici, uno per ogni letterale in \mathcal{C} . I tre vertici corrispondenti ai letterali di ciascuna clausola formano una clique. Inoltre, per ogni $x_i \in \mathcal{X}$, se x_i e \bar{x}_i sono letterali in clausole distinte allora c'è un arco in G fra i vertici corrispondenti. Si noti che possiamo costruire G a partire da \mathcal{C} in tempo polinomiale rispetto a $|\mathcal{C}|$.

Per costruzione di G , l'unico modo per ottenere un insieme indipendente di taglia k nel grafo G è quello di scegliere un nodo in ciascuna delle k clique in modo che non ci siano archi fra il nodo scelto in una clique e nodi scelti

appartenenti a un'altra clique. Per costruzione, questo avviene se e solo se è possibile scegliere esattamente un letterale in ciascuna clausola in modo tale che tra i letterali scelti non compaiano simultaneamente x_i e \bar{x}_i per nessuna delle variabili $x_i \in \mathcal{X}$.

Possiamo quindi costruire un algoritmo che risolve 3-SAT in tempo polinomiale usando un oracolo per Independent Set: l'algoritmo riceve un'istanza I di 3-SAT, costruisce l'istanza corrispondente I' di Independent Set e accede all'oracolo ottenendo $q_{IS}(I')$, per poi restituire $q_{IS}(I')$. \square

1.2 Classi \mathcal{P} e \mathcal{NP}

Sia \mathcal{P} la classe dei problemi di decisione X risolvibili in tempo polinomiale. Ovvero, per ogni problema $X \in \mathcal{P}$ esiste un algoritmo che lo risolve in tempo polinomiale nella lunghezza delle istanze.

La funzione di decisione q di un problema X caratterizza una determinata proprietà delle sue istanze (per esempio, quei grafi che contengono un insieme indipendente abbastanza grande). Un **certificatore polinomiale** per X è un algoritmo $B : \mathcal{I} \times \{0, 1\}^* \rightarrow \{0, 1\}$ tale che

1. Esiste un polinomio $p(\cdot)$ tale che, per ogni istanza $I \in \mathcal{I}$, $q(I) = 1$ se e solo se esiste una stringa $z \in \{0, 1\}^{p(|I|)}$ tale che $B(I, z) = 1$
2. B termina in tempo polinomiale in $|I|$ e $|z|$

Possiamo pensare alla stringa z come un certificato del fatto che $q(I) = 1$. Per esempio, nel problema Independent Set la stringa z denota il sottinsieme di vertici che costituisce un insieme indipendente di cardinalità almeno k . Nel problema SAT, la stringa z denota un assegnamento che soddisfa tutte le clausole.

L'accesso a un certificatore polinomiale permette di verificare rapidamente se una stringa z è un certificato valido per un'istanza I di un problema. D'altra parte, se volessimo usare il certificatore per trovare un certificato qualora esso esista, ovvero stabilire il valore di $q(I)$, saremmo obbligati a eseguire $B(I, z)$ esaustivamente su tutte le $2^{p(|I|)}$ stringhe z tali che $|z| \leq p(|I|)$.

Introduciamo ora la classe \mathcal{NP} di problemi di decisione X che posseggono un certificatore polinomiale. Si noti che $\mathcal{P} \subseteq \mathcal{NP}$. Infatti, se $X \in \mathcal{P}$ allora esiste un algoritmo che calcola la funzione di decisione q in tempo polinomiale in $|I|$. Possiamo usare questo algoritmo per implementare un

certificatore polinomiale B come segue: dati $I, z \in \mathcal{I} \times \{0, 1\}^*$, B restituisce $q(I)$ ignorando z .

Quindi se $q(I) = 1$, abbiamo che $B(I, z) = 1$ per ogni $z \in \{0, 1\}^*$ (in particolare per z limitati in lunghezza da un polinomio in $|I|$). Invece, se $q(I) = 0$, allora $B(I, z) = 0$ per ogni $z \in \{0, 1\}^*$.

Anche se risolvere un’istanza di un problema in \mathcal{P} non significa necessariamente trovare un certificato z , possiamo comunque interpretare \mathcal{NP} come la classe che contiene quei problemi la cui soluzione (o certificato) è *verificabile* in tempo polinomiale, in contrasto con i problemi in \mathcal{P} , i quali problemi sono *risolvibili* in tempo polinomiale. $\mathcal{P} \equiv \mathcal{NP}$ è un problema aperto.

1.3 Problemi \mathcal{NP} -completi

Un problema di decisione X è \mathcal{NP} -completose $x \in \mathcal{NP}$ e per qualunque $Y \in \mathcal{NP}$ vale $Y \preceq_p X$. Intuitivamente, i problemi di decisione \mathcal{NP} -completi sono i “più difficili” in \mathcal{NP} , infatti vale il seguente.

Fatto 1.3.1. *Sia X un qualunque problema \mathcal{NP} -completo. Allora $X \in \mathcal{P}$ se e solo se $P \equiv \mathcal{NP}$.*

Dimostrazione. Se $\mathcal{P} \equiv \mathcal{NP}$ allora $X \in \mathcal{P}$. D’altra parte, se X è \mathcal{NP} -completo, allora dato un qualunque $Y \in \mathcal{NP}$ vale $Y \preceq_p X$. Per ipotesi, X è risolvibile in tempo polinomiale, di conseguenza lo è Y , quindi $Y \in \mathcal{P}$, da cui otteniamo $\mathcal{NP} \subseteq \mathcal{P}$. Abbiamo già dimostrato come $\mathcal{P} \subseteq \mathcal{NP}$, quindi $\mathcal{P} \equiv \mathcal{NP}$. \square

Non è chiaro che esistano problemi \mathcal{NP} -completi. Potrebbero esserci due problemi $X', X'' \in \mathcal{NP}$ tali che non esiste nessun altro $X \in \mathcal{NP}$ tale che $X' \preceq_p X$ e $X'' \preceq_p X$. Oppure, potrebbe esserci una sequenza infinita di problemi in \mathcal{NP} del tipo $X_1 \preceq_p X_2 \preceq_p \dots$ tale che non esista un problema più difficile degli altri.

Un **circuito** è un grafo diretto aciclico (senza loop, archi multipli e pesi) avente un unico nodo senza archi uscenti chiamato nodo di output. I nodi senza archi entranti possono avere un valore di verità preassegnato. I nodi senza archi entranti e senza valori preassegnati sono chiamati nodi di input. I nodi rimanenti hanno uno o due archi entranti e sono etichettati da un operatore booleano \wedge, \vee, \neg in modo tale che nodi \wedge e \vee abbiano esattamente due archi entranti e nodi \neg abbiano esattamente un arco entrante.

Un circuito calcola una funzione booleana $f : \{0, 1\}^n \rightarrow \{0, 1\}$ dove n è il numero dei nodi di input. Dato un assegnamento $(b_1, \dots, b_n) \in \{0, 1\}^n$ di valori di verità ai nodi di input, calcoliamo $f(b_1, \dots, b_n)$ come il valore di verità del nodo di output ottenuto valutando in cascata i valori di verità di ciascun nodo. La valutazione di un nodo avviene applicando l'operatore logico che lo etichetta ai valori di verità dei nodi all'altro capo degli archi entranti. Se la valutazione di ogni nodo avviene in tempo costante, allora l'intero circuito viene valutato in tempo lineare nel numero dei nodi.

Sostanzialmente, si tratta del parsing tree di una funzione booleana.

Il problema di decisione **Circuit Satisfiability** (CS) ha istanze che rappresentano circuiti. La funzione di decisione q è tale che $q(I) = 1$ se e solo se esiste un assegnamento ai nodi di input del circuito I tale che il nodo di output assume valore 1. In altre parole, $q(I) = 1$ se e solo se la funzione f calcolata dal circuito è tale che $f(b_1, \dots, b_n) = 1$ per un qualche $(b_1, \dots, b_n) \in \{0, 1\}^n$.

Teorema 1.3.1 (Cook-Levin). *CS è \mathcal{NP} -completo.*

Dimostrazione. La dimostrazione completa è omessa in quanto lunga e tecnicamente complessa. Si basa sul fatto che ogni algoritmo con un tempo di esecuzione polinomiale nella lunghezza dell'input può essere implementato da una famiglia di circuiti C_1, C_2, \dots dove C_n ha n nodi di input e un numero polinomiale di altri nodi. Per simulare l'algoritmo su un input I di lunghezza n , calcoliamo l'output del circuito C_n con input I .

Per dimostrare che un problema $X \in \mathcal{NP}$ è polinomialmente riducibile a CS consideriamo un certificatore polinomiale B per X , il quale esiste in quanto $X \in \mathcal{NP}$. Costruiamo la famiglia di circuiti C'_1, C'_2, \dots tale che C'_n ha $n + p(n)$ nodi senza archi entranti e un numero polinomiale in n di altri nodi, dove $p(\cdot)$ è il polinomio che limita la lunghezza dei certificati di B . Per ogni n , C'_n simula B su istanze I di lunghezza n . Data un'istanza $I \in \mathcal{I}$ di lunghezza n , sia $C'_n(I, \cdot)$ il circuito C'_n dove i valori dei primi n nodi senza archi entranti sono preassegnati ai bit di I , mentre i rimanenti $p(n)$ nodi sono di input. Allora $C'_n(I, \cdot)$ simula il certificatore polinomiale $B(I, \cdot)$. Ovvero, $C'_n(I, \cdot)$ è soddisfacibile se e solo se esiste $z \in \{0, 1\}^*$ con $|z| \leq p(|I|)$ tale che $B(I, z) = 1$. \square

Dimostriamo ora un caso particolare del teorema di Cook-Levin, ovvero che un particolare problema di decisione è polinomialmente riducibile a CS. Il

problema è 2-IS, con istanze I che rappresentano grafi semplici, mentre la funzione di decisione q è tale che $q(I) = 1$ se e solo se il grafo I contiene un insieme indipendente di tagli almeno 2.

Teorema 1.3.2. $2\text{-IS} \preceq_p CS$.

Dimostrazione. Sia $G = (V, E)$ un grafo su n vertici. Gli archi del grafo G possono essere codificati con una stringa $I_G \in \{0, 1\}^N$ dove $N = \binom{n}{2}$, ogni bit rappresenta una coppia di vertici e un arco fra una coppia di vertici è codificato ponendo a 1 il bit corrispondente. Indichiamo con B il certificatore polinomiale per 2-IS, il quale esiste in quanto $2\text{-IS} \in \mathcal{NP}$.

Dimostriamo ora come costruire in tempo polinomiale nella descrizione di G un circuito C con $\binom{n}{2} + n$ nodi senza archi entranti tale che $C(I_G, \cdot)$ è soddisfacibile se e solo se esiste una stringa z di lunghezza n tale che $B(I_G, z) = 1$. Questo completa la riduzione, in quanto per calcolare la funzione di decisione di 2-IS su un'istanza I_G è sufficiente costruire C , invocare l'oracolo per CS e produrre in output la risposta dell'oracolo.

Notiamo che i certificati per 2-IS possono essere codificati con una stringa $z \in \{0, 1\}^n$ che ha almeno due occorrenze di 1 nelle posizioni corrispondenti ai vertici che formano un insieme indipendente nel grafo.

Costruiamo quindi un circuito con $\binom{n}{2} + n$ nodi di input tale che i primi $\binom{n}{2}$ vengono utilizzati per codificare I_G e i successivi n nodi vengono utilizzati per codificare un certificato z . A questo punto, usiamo $2\binom{n}{2} - 1$ per verificare che z contenga almeno due occorrenze di 1 e usiamo $2\binom{n}{2}$ nodi interni per verificare che non ci sia un arco fra i nodi scelti dal certificato. \square

In Figura 1.1 si può vedere un esempio:

- Al di sopra dei nodi di input per z , ogni nodo \wedge chiede “sono stati scelti questi due nodi?”; la restante parte destra dell’albero controlla solamente che siano stati scelti almeno due nodi
- Il risultato della “scelta” viene messo in \wedge con la codifica degli archi presenti
- I risultati vengono messi in \vee , quindi risulta 1 se e solo se sono stati scelti due nodi che hanno un arco presente tra loro; viene invertito per avere 1 nel caso di Independent Set

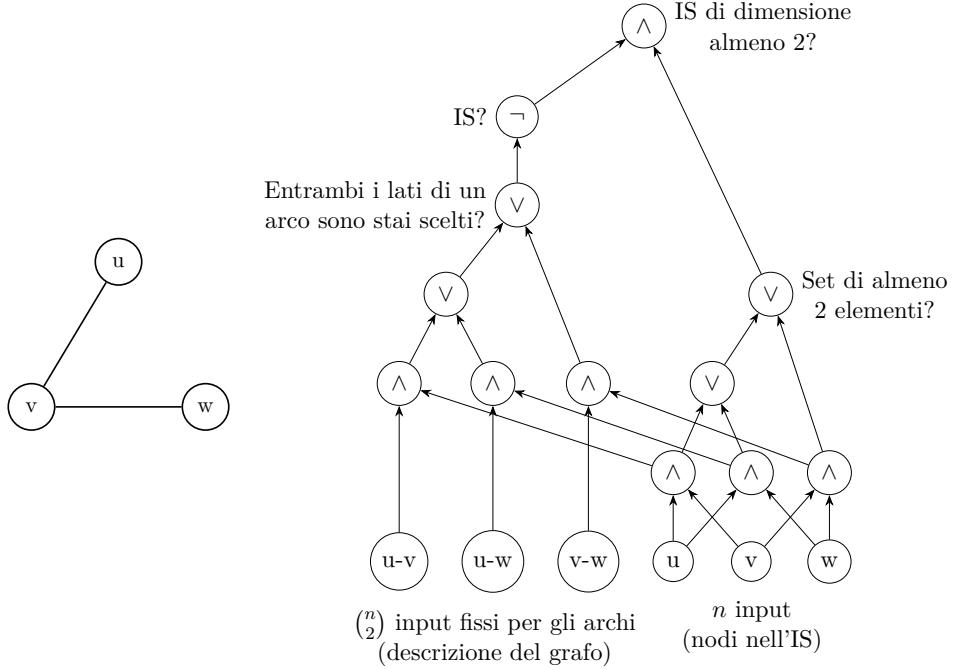


Figura 1.1: Riduzione da 2-IS a CS

- La radice si occupa di controllare che l'input rappresenti un Independent set (sottoalbero di sinistra) di dimensione almeno 2 (sottoalbero di destra)

Una volta che abbiamo stabilito che un certo problema è \mathcal{NP} -completo, possiamo trovarne molti altri usando la seguente osservazione.

Fatto 1.3.2. *Se Y è \mathcal{NP} -completo e $X \in \mathcal{NP}$ è tale che $Y \preceq_p X$, allora anche X è \mathcal{NP} -completo.*

Dimostrazione. Sia $Z \in \mathcal{NP}$ qualunque. Allora $Z \preceq_p Y$. Ma dato che $Y \preceq_p X$, allora $Z \preceq_p X$, il che implica che X è \mathcal{NP} -completo. \square

Possiamo subito applicare questa osservazione dimostrando quanto segue.

Teorema 1.3.3. *$CS \preceq_p 3-SAT$.*

Dimostrazione. Omessa. \square

Dato che CS è \mathcal{NP} -completo, ne segue che anche 3-SAT è \mathcal{NP} -completo. Ricordando inoltre che 3-SAT \preceq_p Independent Set \preceq_p Vertex Cover \preceq_p Set Cover, ne deduciamo che tutti questi problemi sono \mathcal{NP} -completo. Al contrario di 3-SAT che è \mathcal{NP} -completo, 2-SAT (ovvero usando clausole da esattamente due letterali) è un problema risolvibile in tempo lineare.

Si noti che ogni formula booleana può essere equivalentemente rappresentata come un circuito (ovvero un'istanza di CS) o come una formula CNF (ovvero un'istanza di SAT) in modo che tali istanze abbiano lunghezza polinomiale nella lunghezza della formula. Inoltre, ogni istanza I di SAT può essere rappresentata come un'istanza di 3-SAT di lunghezza polinomiale in $|I|$ (queste dimostrazioni sono il processo per la dimostrazione omessa precedentemente).

Al contrario, non è sempre possibile rappresentare una formula booleana in DNF in modo che la DNF abbia lunghezza polinomiale nella lunghezza della formula. Se questo fosse possibile avremmo che $\mathcal{P} \equiv \mathcal{NP}$. La soddisfacibilità di una DNF è decidibile in tempo lineare controllando che esista almeno una congiunzione della formula che è soddisfacibile da un qualche assegnamento.

Si noti che la definizione di \mathcal{NP} è asimmetrica: dato un problema $X = (\mathcal{I}, q) \in \mathcal{NP}$, se $q(I) = 1$ allora esiste un certificato polinomiale z tale che $B(I, z) = 1$ dove B denota il certificatore polinomiale per X . Se invece $q(I) = 0$ allora la definizione ci garantisce solo che $B(I, z) = 0$ per un numero esponenziale di certificati z . Questa asimmetria si riscontra quanto consideriamo problemi $\bar{X} = (\mathcal{I}, \bar{q})$ che sono complementari di problemi $X = (\mathcal{I}, q)$. Per *complementare* di X si intende che $\bar{q}(I) = 0$, ma non sappiamo se esista un certificato polinomiale che certifichi $\bar{q}(I) = 1$. Per esempio, se X è Independent Set, allora per certificare $\bar{q}(I) = 1$ dovremmo trovare un certificato polinomiale che attesti che il grafo **non** contenga un insieme indipendente di taglia k .

Quindi non è chiaro se $\mathcal{NP} \equiv \text{co-}\mathcal{NP}$, dove co- \mathcal{NP} indica la classe dei problemi di decisione complementari di problemi in \mathcal{NP} (se \mathcal{NP} è l'insieme dei problemi di cui posso verificare una soluzione/certificato in tempo polinomiale, per i problemi in co- \mathcal{NP} non è possibile avere una soluzione verificabilmente corretta in tempo polinomiale, è invece possibile verificare in tempo polinomiale che una soluzione restituisce 0).

Lo scenario è differente per \mathcal{P} : $X \in \mathcal{P}$ se e solo se $\bar{X} \in \mathcal{P}$. Infatti $X = (\mathcal{I}, q) \in \mathcal{P}$ implica che esiste un algoritmo per calcolare q in tempo poli-

nomiale. Ma allora si può anche calcolare \bar{q} in tempo polinomiale semplicemente calcolando q e complementando l'output. Quindi $\mathcal{P} \equiv \text{co-}\mathcal{P}$.

Dimostrare che $\text{co-}\mathcal{NP} \not\equiv \mathcal{NP}$ sarebbe un progresso ancora maggiore che dimostrare $\mathcal{P} \not\equiv \mathcal{NP}$. Vale infatti la seguente cosa.

Fatto 1.3.3. *Se $\text{co-}\mathcal{NP} \not\equiv \mathcal{NP}$ allora $\mathcal{P} \not\equiv \mathcal{NP}$.*

Dimostrazione. Dimostriamo la contrapositiva, ovvero che $\mathcal{P} \equiv \mathcal{NP}$ implica $\text{co-}\mathcal{NP} \equiv \mathcal{NP}$. Intuitivamente, dato che \mathcal{P} è chiuso rispetto all'operazione di complemento, se $\mathcal{P} \equiv \mathcal{NP}$ allora deve essere che $\text{co-}\mathcal{NP} \equiv \mathcal{NP}$. Formalmente

$$X \in \mathcal{NP} \iff X \in \mathcal{P} \iff \bar{X} \in \mathcal{P} \iff \bar{X} \in \mathcal{NP} \iff X \in \text{co-}\mathcal{NP}$$

Il che conclude la dimostrazione. \square

Possiamo caratterizzare i problemi $X = (\mathcal{I}, q) \in \text{co-}\mathcal{NP}$ tramite l'esistenza di un polinomio $p(\cdot)$ e di un certificatore polinomiale B , calcolabile in tempo polinomiale, tale che $q(I) = 0$ se e solo se esiste una stringa $z \in \{0, 1\}^{p(|I|)}$ tale che $B(I, z) = 0$. Si noti che $\mathcal{P} \subseteq \text{co-}\mathcal{NP}$, infatti, potendo implementare q in tempo polinomiale, si può calcolare $B(I, z)$ ignorando z e restituendo $q(I)$.

Una classe particolarmente interessante è quella dei problemi in $\mathcal{NP} \cap \text{co-}\mathcal{NP}$. Sia $X = (\mathcal{I}, q)$ un tale problema

- Dato che $X \in \mathcal{NP}$ esiste un polinomio $p(\cdot)$ e un certificatore polinomiale B tale che $q(I) = 1$ se e solo se esiste una stringa $z \in \{0, 1\}^{p(|I|)}$ tale che $B(I, z) = 1$
- Dato che $X \in \text{co-}\mathcal{NP}$ esiste un polinomio $p'(\cdot)$ e un certificatore polinomiale B' tale che $q(I) = 0$ se e solo se esiste una stringa $z' \in \{0, 1\}^{p'(|I|)}$ tale che $B'(I, z') = 0$

Quindi i problemi in $\mathcal{NP} \cap \text{co-}\mathcal{NP}$ sono tali che per ogni istanza I esiste un certificato polinomiale, sia quando $q(I) = 1$ sia quando $q(I) = 0$.

Si noti che se $X \in \mathcal{P}$ allora $X \in \mathcal{NP}$ ed anche $X \in \text{co-}\mathcal{NP}$. Quindi $\mathcal{P} \subseteq \mathcal{NP} \cap \text{co-}\mathcal{NP}$. D'altra parte non si sa se $\mathcal{P} \not\equiv \mathcal{NP} \cap \text{co-}\mathcal{NP}$. Ovvero non si sa se esistono problemi le cui istanze hanno sempre certificati brevi ma tuttavia non sono risolvibili in tempo polinomiale.

Capitolo 2

Algoritmi Probabilistici

2.1 Algoritmi Montecarlo e Las Vegas

Un algoritmo probabilistico è un algoritmo che ha accesso a un oracolo che, a ogni chiamata, restituisce in tempo unitario un bit casuale, ovvero una variabile casuale Y tale che $\mathbb{P}(Y = 1) = \mathbb{P}(Y = 0) = 1/2$. Inoltre, i bit restituiti in una sequenza di chiamate all'oracolo sono indipendenti.

Indicheremo con $Z \in \{0, 1\}^*$ la stringa di bit casuali indipendenti che l'oracolo restituisce in una sequenza di chiamate. Indicheremo anche con $A(I, Z) \in \{0, 1\}$ la variabile casuale che rappresenta l'output dell'algoritmo probabilistico A per un problema di decisione $X = (\mathcal{I}, q)$ e avente come input l'istanza $I \in \mathcal{I}$ e i bit casuali Z dell'oracolo. Similmente, indichiamo con $T_A(I, Z)$ la variabile casuale che rappresenta il tempo di esecuzione di A con input $I \in \mathcal{I}$ e bit casuali Z forniti dall'oracolo.

Esistono due principali tipi di algoritmi probabilistici.

Algoritmi Montecarlo. Sono algoritmi A tali che

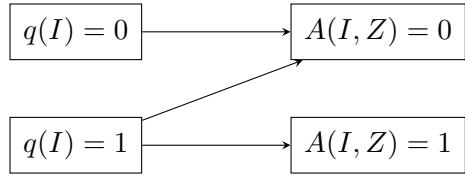
- Per ogni $I \in \mathcal{I}$, $T_A(I, Z)$ dipende solo da I , ovvero il tempo di esecuzione è deterministico
- Esiste $I \in \mathcal{I}$ per cui $\mathbb{P}(A(I, Z) \neq q(I)) > 0$, ovvero l'output non è sempre corretto

Gli algoritmi Montecarlo si dividono ulteriormente in:

- Algoritmi con errore **one-sided**; un algoritmo ha errore one-sided quando è sempre corretto almeno su uno dei suoi due possibili output. Convenzionalmente, assumeremo che l'algoritmo sia sempre corretto per output 1. Ovvero, A è Montecarlo one-sided quando, per ogni $I \in \mathcal{I}$, $\mathbb{P}(A(I, Z) = q(I) | A(I, Z) = 1) = 1$ e $\mathbb{P}(A(I, Z) = q(I) | A(I, Z) = 0) > 0$

Algoritmi con errore **two-sided**. Possono sbagliare su entrambi i possibili output. Ovvero, A è Montecarlo two-sided quando $\mathbb{P}(A(I, Z) = q(I)) > 1/2$ per ogni $I \in \mathcal{I}$

Una rappresentazione grafica della relazione fra $q(I)$ e $A(I, Z)$ per un algoritmo Montecarlo one-sided è la seguente



Questo mostra come $A(I, Z) = 1$ può solo corrispondere a $q(I) = 1$, mentre $A(I, Z) = 0$ lascia incertezza sul valore di $q(I)$. Si noti anche che quando $q(I) = 0$ l'algoritmo è sempre corretto.

Algoritmi Las Vegas. Sono algoritmi che producono sempre l'output corretto ma che hanno un tempo di esecuzione probabilistico (ovvero che dipende dai bit forniti dall'oracolo). Ovvero, un algoritmo A è Las Vegas quando $\mathbb{P}(A(I, Z) = q(I)) = 1$ per ogni $I \in \mathcal{I}$ ma il tempo di calcolo di A su una qualunque istanza $I \in \mathcal{I}$ è una variabile casuale $T_A(I, Z)$ tale che $\mathbb{E}[T_A(I, Z)] < \infty$.

Amplificazione. Un algoritmo Montecarlo one-sided può essere facilmente trasformato in un algoritmo con probabilità di errore arbitrariamente piccola attraverso un meccanismo di amplificazione

Sia

$$\mathbb{P}(A(I, Z) \neq q(I) | A(I, Z) = 0) \leq 1 - p_n, \quad \forall I \in \mathcal{I} \text{ con } |I| = n$$

Se su una data istanza l'algoritmo produce 0 in output possiamo eseguirlo nuovamente per amplificare la probabilità di avere l'output corretto. Se k esecuzioni indipendenti producono sistematicamente la risposta 0, allora la probabilità che la risposta corretta sia 1 è al più $(1 - p_n)^k \leq e^{-p_n k}$. Perché

questa probabilità sia al più un $\epsilon > 0$ piccolo a piacere è sufficiente scegliere $k \geq \frac{1}{p_n} \ln \frac{1}{\epsilon}$.

Un meccanismo di amplificazione simile ma leggermente più complesso esiste anche per gli algoritmi Montecarlo two-sided. Supponiamo che su istanze di taglia n l'algoritmo fornisca la risposta errata con probabilità al più $\frac{1}{2} - p_n < \frac{1}{2}$. Per amplificare la probabilità di ottenere la risposta corretta possiamo ripetere l'esecuzione k volte e utilizzare un voto di maggioranza sui k output prodotti (supponiamo, per semplicità, che k sia dispari).

Per analizzare il voto di maggioranza utilizzeremo il seguente lemma.

Lemma 2.1.1 (Chernoff-Hoeffding). *Siano Y_1, \dots, Y_k variabili casuali Bernoulliane (cioè con valori in $\{0, 1\}$), indipendenti e tali che $\mathbb{P}(Y_t = 1) \leq \mu$ per $t = 1, \dots, k$. Allora, per ogni $\epsilon > 0$ fissato*

$$\mathbb{P}\left(\frac{1}{k} \sum_{t=1}^k Y_t > \mu + \epsilon\right) \leq e^{-2\epsilon^2 k}$$

Dimostrazione. Omessa. □

Sia $p_E = \mathbb{P}(A(I, Z) \neq q(I))$ la probabilità di errore dell'algoritmo su un'istanza I del problema di decisione (\mathcal{I}, q) . Siano $X_1, \dots, X_k \in \{0, 1\}$ le variabili casuali indipendenti che denotano gli output prodotti dalle k esecuzioni dell'algoritmo. Sia $M_k \in \{0, 1\}$ la variabile casuale che denota il voto di maggioranza su X_1, \dots, X_k (ovvero $M_k = 1$ se e solo se $\sum_{t=1}^k X_t > \frac{k}{2}$).

Allora

$$M_k = q(I) \iff \sum_{t=1}^k Y_t < \frac{k}{2}$$

dove $Y_t = 1$ se e solo se $X_t \neq q(I)$, per $t = 1, \dots, k$. In altre parole, il voto di maggioranza M_k è corretto se e solo se l'algoritmo genera output errato in non più di $\lfloor \frac{k}{2} \rfloor$ delle k esecuzioni.

Ora, Y_1, \dots, Y_k sono variabili casuali indipendenti (perché le esecuzioni dell'algoritmo sono indipendenti), identicamente distribuite, con valori in $\{0, 1\}$ e tali che $\mathbb{P}(Y_t = 1) = p_E \leq \frac{1}{2} - p_n$ per ogni $t = 1, \dots, k$.

Applicando il lemma di Chernoff-Hoeffding (2.1.1) otteniamo che la proba-

bilità che il voto di maggioranza sia sbagliato è limitata da

$$\begin{aligned}\mathbb{P}\left(\sum_{t=1}^k Y_t > \frac{k}{2}\right) &= \mathbb{P}\left(\frac{1}{k} \sum_{t=1}^k Y_t > \left(\frac{1}{2} - p_n\right) + p_n\right) \\ &\leq \mathbb{P}\left(\frac{1}{k} \sum_{t=1}^k Y_t > p_E + p_n\right) \\ &\leq e^{-2p_n^2 k}\end{aligned}$$

Perché la probabilità $e^{-2p_n^2 k}$ sia al più un $\epsilon > 0$ piccolo a piacere è sufficiente scegliere $k \geq \frac{1}{2p_n^2} \ln \frac{1}{\epsilon}$.

Si noti che nel caso one-sided possiamo usare l'amplificazione per ridurre qualsiasi probabilità di errore strettamente minore di 1, mentre nel caso two-sided la stessa cosa vale per qualsiasi probabilità strettamente minore di $\frac{1}{2}$.

Da Las Vegas a Montecarlo one-sided. Un algoritmo Las Vegas per un problema di decisione può essere trasformato in un algoritmo Montecarlo one-sided. Per fare ciò utilizziamo la disuguaglianza di Markov.

Lemma 2.1.2 (Markov). *Sia Z una variabile casuale non negativa tale che $\mathbb{E}[Z] < \infty$. Allora per ogni $c > 0$*

$$\mathbb{P}(Z > c) \leq \frac{\mathbb{E}[Z]}{c}$$

Dimostrazione. Sia \mathcal{A} l'insieme di numeri non negativi tali che $Z \in \mathcal{A}$. Allora

$$\begin{aligned}\mathbb{E}[Z] &= \sum_{a \in \mathcal{A}} a \mathbb{P}(Z = a) = \overbrace{\sum_{a \in \mathcal{A}: a \leq c} a \mathbb{P}(Z = a)}^{\geq 0} + \sum_{a \in \mathcal{A}: a > c} a \mathbb{P}(Z = a) \\ &\geq c \sum_{a \in \mathcal{A}: a > c} \mathbb{P}(Z = a) \\ &= c \mathbb{P}(Z = c)\end{aligned}$$

$$\mathbb{E}[Z] \geq c \mathbb{P}(Z > c) \implies \mathbb{P}(Z > c) \leq \frac{\mathbb{E}[Z]}{c}$$

concludendo la dimostrazione. □

Sia A un algoritmo Las Vegas per un problema di decisione (\mathcal{I}, q) . Sia $f : \mathbb{N} \rightarrow \mathbb{R}$ la funzione tale che

$$f(n) = \max \{\mathbb{E}[T_A(I, Z)] \mid I \in \mathcal{I}, |I| = n\}$$

Dato che A è Las Vegas, $f(n) < \infty$ per ogni $n \in \mathbb{N}$. Possiamo quindi trovare una funzione $t : \mathbb{N} \rightarrow \mathbb{N}$ tale che

$$t(n) \geq \frac{3}{2}f(n), \quad n \in \mathbb{N}$$

Posso quindi costruire un algoritmo A' che simula A sull'istanza I arrestando la simulazione dopo $t(n)$ passi. Se A non ha terminato, allora A' produce 0 in output. Dato che A è Las Vegas, A' sbaglia solo quando A non termina entro $t(n)$ passi. Per la disuguaglianza di Markov, la probabilità che ciò accada è al più

$$\mathbb{P}(T_A(I, Z) > t(n)) \leq \frac{\mathbb{E}[T_A(I, Z)]}{t(n)} \leq \frac{f(n)}{t(n)} \leq \frac{2}{3}$$

Inoltre, dato che quando A non termina l'output di A' è 0, A' è one-sided dato che l'output 1 è sempre corretto. Quindi A' è un algoritmo Montecarlo one-sided con probabilità di errore al più $2/3$. Infine, si noti che il tempo di esecuzione di A' soddisfa $T_{A'}(I) \leq t(|I|) = \mathcal{O}(f(|I|))$.

Riassumendo: un algoritmo Las Vegas può diventare Montecarlo one-sided con i seguenti passi

- Trova una funzione $t(n)$ che maggiora la funzione $f(n)$, la quale rappresenta il tempo di esecuzione atteso per input di taglia n , ovvero $t(n) = m \cdot f(n)$
- L'algoritmo Montecarlo usa l'algoritmo Las Vegas, bloccandolo se quest'ultimo non termina entro $t(n)$ passi; restituisce sempre il risultato dell'algoritmo Las Vegas, oppure 0 se questo non termina
- La probabilità di errore è solo nel caso in cui esca 0; per la diseguaglianza di Markov, la probabilità che ciò accada è $\leq \frac{1}{m}$; quando esce 1, l'algoritmo è sempre corretto (quindi one-sided)

2.2 Classi di complessità probabilistiche

Un algoritmo probabilistico per un problema di decisione $X = (\mathcal{I}, q)$ può essere visto come un algoritmo deterministico che ha accesso a una stringa Z di bit casuali. L'algoritmo calcola una funzione $B : \mathcal{I} \times \{0, 1\}^* \rightarrow \{0, 1\}$ tale che, per ogni $I \in \mathcal{I}$, $B(I, Z) = q(I)$ con una certa probabilità rispetto all'estrazione della stringa Z .

In analogia con la definizione di \mathcal{NP} , possiamo definire le classi di problemi di decisione solubili in tempo polinomiale da diversi tipi di algoritmi probabilistici rivisitando la nozione di certificatore polinomiale.

La classe di problemi di decisione risolti in modo efficiente da algoritmi Montecarlo two-sided è la classe \mathcal{BPP} . Un problema di decisione $X = (\mathcal{I}, q)$ appartiene alla classe \mathcal{BPP} se esiste una funzione $B : \mathcal{I} \times \{0, 1\}^* \rightarrow \{0, 1\}$ calcolabile in tempo polinomiale e un polinomio $p(\cdot)$ tali che, per ogni istanza $I \in \mathcal{I}$ essi soddisfano

$$\mathbb{P}(B(I, Z) \neq q(I)) \leq \frac{1}{3}$$

dove la probabilità è calcolata rispetto all'estrazione di Z con probabilità uniforme da $\{0, 1\}^{p(|I|)}$.

La costante $1/3$ è arbitraria, dato che, come già visto, si può ridurre a piacimento la probabilità di errore di un algoritmo two-sided tramite amplificazione. Una definizione equivalente di \mathcal{BPP} sostituisce la diseguaglianza sopra con

$$\mathbb{P}(B(I, Z) \neq q(I)) \leq \frac{1}{2} - \frac{1}{p'(|I|)}$$

dove $p'(\cdot)$ è un polinomio. Il meccanismo di amplificazione tramite Lemma di Chernoff-Hoeffding (2.1.1) implica che è sufficiente eseguire l'algoritmo un numero di volte pari a ordine di $p'(|I|)^2$ per ottenere una probabilità di errore limitata da $1/3$. Dato che $p'(\cdot)$ è un polinomio, l'algoritmo risultante è ancora polinomiale in $|I|$.

Si noti che $\mathcal{P} \subseteq \mathcal{BPP}$, dato che avendo un algoritmo polinomiale per calcolare la funzione di decisione q possiamo implementare il certificatore B in tempo polinomiale con probabilità di errore pari a zero. Non è invece noto se $\mathcal{P} \equiv \mathcal{BPP}$, ovvero se ogni algoritmo Montecarlo two-sided possa essere “derandomizzato” in modo da ottenere un algoritmo deterministico polinomiale per lo stesso problema. Non è neanche noto se $\mathcal{BPP} \subseteq \mathcal{NP}$. D'altra parte, dato che la condizione che definisce \mathcal{BPP} è simmetrica rispetto al

valore di $q(I)$, ne deduciamo che \mathcal{BPP} è chiusa rispetto al complemento, ovvero $\mathcal{BPP} \equiv \text{co-}\mathcal{BPP}$.

La classe di problemi di decisione risolti in modo efficiente da algoritmi Montecarlo one-sided è la classe \mathcal{RP} . Un problema di decisione $X = (\mathcal{I}, q)$ appartiene alla classe \mathcal{RP} se esiste una funzione $B : \mathcal{I} \times \{0, 1\}^* \rightarrow \{0, 1\}$ calcolabile in tempo polinomiale e un polinomio $p(\cdot)$ tali che, per ogni istanza $I \in \mathcal{I}$

$$\begin{aligned}\mathbb{P}(B(I, Z) = 1) &\geq \frac{2}{3}, \quad \text{se } q(I) = 1 \\ \mathbb{P}(B(I, Z) = 0) &= 1, \quad \text{se } q(I) = 0\end{aligned}$$

dove la probabilità è calcolata rispetto all'estrazione di Z con probabilità uniforme da $\{0, 1\}^{p(|I|)}$.

Si noti che questa definizione corrisponde all'osservazione precedentemente fatta che un algoritmo Montecarlo one-sided è sempre corretto quando $q(I) = 0$. Quando $q(I) = 1$, l'algoritmo è corretto con probabilità almeno $2/3$. Quindi l'algoritmo è sempre corretto su output 1, mentre sbaglia con probabilità al più $1/3$ su output 0. Anche in questo caso la costante $1/3$ è arbitraria grazie al meccanismo di amplificazione. Possiamo dare una definizione equivalente di \mathcal{RP} sostituendo la prima delle due condizioni con

$$\mathbb{P}(B(I, Z) = 1) \geq \frac{1}{p'(|I|)}, \quad \text{se } q(I) = 1$$

dove $p'(\cdot)$ è un polinomio. Il meccanismo di amplificazione implica che è sufficiente eseguire l'algoritmo un numero di volte pari a ordine di $p'(|I|)$ per ottenere una probabilità di errore limitata da $1/3$. Dato che $p'(\cdot)$ è un polinomio, l'algoritmo risultante è ancora polinomiale in $|I|$.

Con un ragionamento simile a quello che ci ha portato a concludere che $\mathcal{P} \subseteq \mathcal{BPP}$, possiamo anche dimostrare che $\mathcal{P} \subseteq \mathcal{RP}$. Ma, a differenza di \mathcal{BPP} , questa volta possiamo stabilire una relazione tra \mathcal{RP} e \mathcal{NP} . Infatti, la definizione di \mathcal{NP} può essere equivalentemente riscritta nel modo seguente. Un problema di decisione $X = (\mathcal{I}, q)$ appartiene alla classe \mathcal{NP} se esiste una funzione $B : \mathcal{I} \times \{0, 1\}^* \rightarrow \{0, 1\}$ calcolabile in tempo polinomiale e un polinomio $p(\cdot)$ tali che, per ogni istanza $I \in \mathcal{I}$, essi soddisfano

$$\begin{aligned}\mathbb{P}(B(I, Z) = 1) &> 0, \quad \text{se } q(I) = 1 \\ \mathbb{P}(B(I, Z) = 0) &= 1, \quad \text{se } q(I) = 0\end{aligned}$$

dove le probabilità sono calcolate rispetto all'estrazione di Z con probabilità uniforme da $\{0, 1\}^{p(|I|)}$.

Dato che per la prima condizione di \mathcal{RP} , $\mathbb{P}(B(I, Z) = 1) \geq \frac{2}{3}$ implica $\mathbb{P}(B(I, Z) = 1) > 0$, mentre la seconda condizione è uguale nelle due definizioni. Di conseguenza, concludiamo che $\mathcal{RP} \subseteq \mathcal{NP}$. In altre parole, interpretiamo i bit casuali Z nella definizione di \mathcal{RP} come un certificato del fatto che $q(I) = 1$.

La classe co- \mathcal{RP} contiene i problemi che sono complementi di problemi in \mathcal{RP} . La definizione di co- \mathcal{RP} è semplicemente ottenuta invertendo $q(I) = 0$ e $q(I) = 1$ nella definizione di \mathcal{RP} . Con una dimostrazione simile a quella di $\mathcal{RP} \subseteq \mathcal{NP}$ possiamo dimostrare che $\text{co-}\mathcal{RP} \subseteq \text{co-}\mathcal{NP}$. Come vale $\mathcal{P} \subseteq \mathcal{RP}$ così possiamo dimostrare che $\mathcal{P} \subseteq \text{co-}\mathcal{RP}$.

Possiamo mettere in relazione \mathcal{RP} e co- \mathcal{RP} con \mathcal{BPP} riscrivendo la definizione di quest'ultima come

$$\begin{aligned}\mathbb{P}(B(I, Z) = 1) &\geq \frac{2}{3}, \quad \text{se } q(I) = 1 \\ \mathbb{P}(B(I, Z) = 0) &\geq \frac{2}{3}, \quad \text{se } q(I) = 0\end{aligned}$$

Arrivando così alla conclusione $\mathcal{RP} \subseteq \mathcal{BPP}$ e $\text{co-}\mathcal{RP} \subseteq \mathcal{BPP}$.

Introduciamo ora la classe $\mathcal{ZPP} \equiv \mathcal{RP} \cap \text{co-}\mathcal{RP}$. Un problema di decisione $X = (\mathcal{I}, q)$ appartiene alla classe \mathcal{ZPP} se esistono due funzioni $B, B' : \mathcal{I} \times \{0, 1\}^* \rightarrow \{0, 1\}$ calcolabili in tempo polinomiale e due polinomi $p(\cdot), p'(\cdot)$ tali che, per ogni istanza $I \in \mathcal{I}$, essi soddisfano

$$\begin{aligned}\mathbb{P}(B(I, Z) = 1) &\geq \frac{2}{3} \quad \text{e} \quad \mathbb{P}(B'(I, Z') = 1) = 1 \text{ se } q(I) = 1 \\ \mathbb{P}(B(I, Z) = 0) &\geq \frac{2}{3} \quad \text{e} \quad \mathbb{P}(B'(I, Z') = 0) = 1 \text{ se } q(I) = 0\end{aligned}$$

dove le probabilità sono calcolate rispetto all'estrazione di Z con probabilità uniforme da $\{0, 1\}^{p(|I|)}$ e di Z' con probabilità uniforme da $\{0, 1\}^{p'(|I|)}$.

Non è difficile vedere che la classe \mathcal{ZPP} è la classe dei problemi risolti da algoritmi Las Vegas che terminano in tempo atteso limitato da un polinomio nella lunghezza dell'istanza. Per farlo, abbiamo bisogno del lemma seguente.

Lemma 2.2.1 (Valore atteso distribuzione Geometrica). *Siano Z_1, Z_2, \dots variabili casuali Bernoulliane, indipendenti e tali che $\mathbb{P}(Z_t = 1) = p$ per $t \geq 1$. Sia $G = \min \{k \mid Z_k = 1\}$. Allora $\mathbb{E}[G] = \frac{1}{p}$.*

Dimostrazione.

$$\begin{aligned}
\mathbb{E}[G] &= \sum_{k=1}^{\infty} k(1-p)^{k-1} p = p \sum_{k=1}^{\infty} k(1-p)^{k-1} \\
&= -p \sum_{k=1}^{\infty} \frac{d}{dp} (1-p)^k = -p \frac{d}{dp} \sum_{k=1}^{\infty} (1-p)^k \\
&= -p \frac{d}{dp} \left(\frac{1}{1-(1-p)} - 1 \right) = -p \frac{d}{dp} \frac{1-p}{p} \\
&= -p \frac{-1}{p^2} = \frac{1}{p}
\end{aligned}$$

□

Ora, se $\mathcal{X} \in \mathcal{ZPP}$ allora posso costruire un algoritmo probabilistico A che, su input $I \in \mathcal{I}$, esegue B e B' arrestandosi non appena $B(I, Z) = 1$ oppure $B'(I, Z') = 0$. In entrambi i casi sappiamo che l'output è corretto, quindi A si arresta sempre con la soluzione corretta. La probabilità che su una particolare istanza I si verifichi $B(I, Z) = 0$ e $B'(I, Z') = 1$ è

$$\begin{aligned}
\mathbb{P}(B(I, Z) = 0 \wedge B'(I, Z') = 1) &= \mathbb{P}(B(I, Z) = 0) \mathbb{P}(B'(I, Z') = 1) \\
&\leq \begin{cases} \frac{1}{3} \cdot 1 & \text{se } q(I) = 1 \\ 1 \cdot \frac{1}{3} & \text{se } q(I) = 0 \end{cases}
\end{aligned}$$

ovvero al più $1/3$ indipendentemente dal valore di $q(I)$. Quindi la probabilità che A si arresti con la soluzione corretta è almeno $2/3$ in ogni esecuzione di B e B' . Usando il lemma sul valore atteso della Geometrica (2.2.1), il numero atteso di ripetizioni è quindi al più $\frac{3}{2} < 2$. Dato che per ipotesi B e B' terminano entrambe in tempo polinomiale, il tempo atteso di calcolo di A è polinomiale anch'esso.

D'altra parte, sia A un algoritmo Las Vegas per (\mathcal{I}, q) e sia $\mu(I) < p(|I|)$ il valore atteso del tempo di calcolo $T_A(I, Z)$ di A su input I . Per la diseguaglianza di Markov (2.1.2)

$$\mathbb{P}(T_A(I, Z) \geq \lceil 3\mu(I) \rceil) \leq \frac{1}{3}$$

Quindi se su input I arresto A dopo $\lceil 3\mu(I) \rceil$ passi, la probabilità che A non abbia terminato è al più $1/3$. Viceversa, quando A termina l'output è sempre corretto.

Possiamo quindi implementare le funzioni B e B' come segue:

- B esegue A e produce 0 se A non termina
- Quando $q(I) = 0$ l'output di $B(I, Z)$ è deterministicamente 0
- Quando $q(I) = 1$, l'output di $B(I, Z)$ è 1 con probabilità almeno $\frac{2}{3}$
- In modo simile possiamo implementare B'

Dato che $\mu(I) < p(|I|)$, B e B' terminano entrambi in tempo deterministico polinomiale.

Quindi, in particolare, $\mathcal{ZPP} \subseteq \mathcal{RP}$ come avevamo già osservato trasformando un algoritmo Las Vegas in uno Montecarlo one-sided. Ciò implica che risolvere un problema di decisione con un algoritmo Las Vegas è un risultato più forte che risolverlo con un algoritmo Montecarlo (one-sided o two-sided). Infine, dato che \mathcal{P} è incluso sia in \mathcal{RP} che in $\text{co-}\mathcal{RP}$, abbiamo che $\mathcal{P} \subseteq \mathcal{ZPP}$.

2.3 Estrattore di von Neumann

Un estrattore di causalità è una funzione che trasforma una sorgente non perfettamente casuale in una completamente casuale. Il più semplice estrattore è quello ideato da John von Neumann e risponde alla domanda: *come è possibile usare una moneta truccata per simulare dei lanci di moneta non truccata?*

Più precisamente, avendo una moneta con probabilità sconosciuta $0 < p < 1$ di restituire testa ogni volta che viene lanciata, la si vuole usare per simulare una sequenza di lanci di una moneta equa, ovvero con probabilità $1/2$ di restituire testa.

Siano X_1, X_2, \dots le variabili casuali Bernoulliane indipendenti con $\mathbb{P}(X_t = 1) = p$ che modellano i lanci della moneta truccata. Consideriamo le coppie $(X_1, X_2), (X_3, X_4), \dots$ e notiamo che i valori possibili per ogni coppia sono:

$$\begin{aligned} (0, 0) &\quad \text{con probabilità } (1-p)^2 \\ (1, 1) &\quad \text{con probabilità } p^2 \\ (0, 1) \text{ e } (1, 0) &\quad \text{con probabilità } p(1-p) \end{aligned}$$

Quindi, per ogni coppia (X_{2k-1}, X_{2k}) gli eventi $(X_{2k-1}, X_{2k}) = (0, 1)$ e $(X_{2k-1}, X_{2k}) = (1, 0)$ sono equiprobabili e forniscono la sequenza di lanci desiderata.

Algoritmo 1: Estrattore di von Neumann

Input: Sequenza di lanci X_1, X_2, \dots

```
1 for k = 1, 2, ... do
2   if  $X_{2k-1} \neq X_{2k}$            // Controlla se è una coppia utile
3     then
4       if  $X_{2k-1} = 1$  then
5         | Print "Testa"
6       else
7         | Print "Croce"
```

Praticamente, lancia la moneta truccata finché non ottieni due valori diversi di seguito: se il primo dei due è testa, il lancio “equo” è testa, croce altrimenti. Questo funziona in quanto la probabilità che esca $(0, 1)$ o $(1, 0)$ è la stessa.

Possiamo ora calcolare quanti lanci di moneta truccata servono in media per simulare un lancio di moneta non truccata. Data una sequenza Z_1, Z_2, \dots di variabili Bernoulliane indipendenti tali che $\mathbb{P}(Z_k = 1) = q$ per $k \geq 1$, la variabile casuale geometrica G è definita come $G = \min \{k = 1, 2, \dots \mid Z_k = 1\}$.

Chiaramente $\mathbb{P}(G = 1) = q$ e $\mathbb{P}(G = n) = (1 - q)^{n-1}q$ per ogni $n > 1$. Non è difficile dimostrare che $\mathbb{E}[G] = \frac{1}{q}$ (Lemma 2.2.1).

Consideriamo ora la sequenza Z_1, Z_2, \dots di variabili Bernoulliane indipendenti tali che

$$Z_k = \begin{cases} 1 & X_{2k-1} \neq X_{2k} \\ 0 & \text{altrimenti} \end{cases}$$

Per quanto detto prima, $\mathbb{P}(Z_k = 1) = 2p(1-p)$. Sia G la variabile geometrica associata alla sequenza delle Z_k . Quindi il numero medio di lanci che mi servono è

$$2\mathbb{E}[G] = \frac{1}{p(1-p)}$$

2.4 Il problema del Coupon Collector

Il problema del coupon collector è definito come segue: sia X_1, X_2, \dots una sequenza di variabili casuali indipendenti e uniformemente distribuite su n

valori distinti a_1, \dots, a_n

$$\mathbb{P}(X_t = a_i) = \frac{1}{n}, \quad i = 1, \dots, n, \quad t \geq 1$$

Calcolare $\mathbb{E}[N]$, dove $N = \min \{k \mid (\forall i \leq n) (\exists t \leq k) X_t = a_i\}$. In altre parole, N è il minimo numero di realizzazioni x_1, \dots, x_k sufficienti a osservare ciascun a_i almeno una volta.

Il nome *coupon collector* deriva dal problema di collezionare tutti gli n possibili coupon contenuti in prodotti da acquistare (per esempio, scatole di cereali), dove ogni scatola contiene uno qualsiasi dei buoni premio con probabilità uniforme.

Un problema equivalente è il seguente: supponiamo che a ogni lancio, una pallina cade con probabilità uniforme in una fra n possibili scatole. Quante palline devo lanciare in media affinché ce ne sia almeno una in ogni scatola?

Un'applicazione concreta del coupon collector è la seguente: supponiamo di voler sapere gli identificativi degli n router attraversati da una sequenza di pacchetti. Mentre non c'è abbastanza spazio in un pacchetto per memorizzare tutti gli n identificativi, è facile memorizzare in un pacchetto l'identificativo di un router a caso tra quelli attraversati. Ci si chiede allora quanti pacchetti servono in media per ottenere gli identificativi di tutti gli n router.

Per analizzare il problema, suddividiamo X_1, X_2, \dots in n blocchi di lunghezze N_1, \dots, N_n , dove N_i è il numero di estrazioni aggiuntive che servono per ottenere l' i -esimo valore distinto avendone già osservati $i - 1$. Quindi

$$N = \sum_{i=1}^n N_i$$

Le variabili casuali N_1, \dots, N_n sono tutte Geometriche. In particolare, quando $i - 1$ valori distinti sono già stati osservati, la probabilità di osservarne uno nuovo è

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

Infatti, $p_1 = 1$ e questo implica $N_1 = 1$ deterministicamente, ovviamente.

Ricordando che il valore atteso di una Geometrica di parametro p_i è $\mathbb{E}[N_i] = \frac{1}{p_i}$ (Lemma 2.2.1), per linearità del valore atteso abbiamo

$$\mathbb{E}[N] = \sum_{i=1}^n \mathbb{E}[N_i] = \sum_{i=1}^n \frac{1}{p_i} = \sum_{i=1}^n \frac{n}{n-i+1} = n \sum_{i=1}^n \frac{1}{i} = n \ln n + \Theta(n)$$

dove l'ultima uguaglianza vale perché la somma armonica $1 + \frac{1}{2} + \dots + \frac{1}{n}$ è asintotica a $\ln n + \Theta(1)$.

2.5 Reservoir Sampling

Si consideri il problema di mantenere una struttura dati che, a ogni istante di tempo, contenga k elementi estratti a caso con probabilità uniforme da uno stream di elementi in ingresso. In particolare, vogliamo sviluppare un algoritmo che soddisfi il seguente invariante: per ogni $t \geq k$, ognuno dei primi t elementi dello stream è contenuto nella struttura dati con probabilità pari a $\frac{k}{t}$.

Per esempio, vogliamo stimare le percentuali delle varie tipologie di oggetti (libri, elettronica, abbigliamento, ...) venduti su Amazon in un dato lasso di tempo. Se ogni oggetto venduto è campionato con la stessa probabilità, allora la distribuzione delle tipologie nel campiona sarà tendenzialmente uguale a quella nello stream.

Studiamo il problema nel modello streaming: a ogni istante di tempo $t = 1, 2, \dots$ l'algoritmo può accedere soltanto al t -esimo elemento x_t dello stream. Chiediamo inoltre che l'algoritmo lavori in spazio $\Theta(k)$.

Il seguente algoritmo soddisfa tutte le proprietà richieste.

Algoritmo 2: Reservoir Sampling

Input: Intero k

```

1  $R = \emptyset$                                 // Inizializza la riserva
2 for  $t = 1, 2, \dots$  do
3   | Leggi il prossimo elemento  $x_t$  nello stream
4   | if  $t \leq k$  then
5     |   | Aggiungi  $x_t$  a  $R$ 
6   | else
7     |   | Con probabilità  $\frac{k}{t}$ , sostituisci un elemento a caso in  $R$  con  $x_t$ 

```

Nel caso in cui lo stream avesse lunghezza nota N , potremmo aggiungere alla riserva ogni elemento dello stream in modo indipendente con probabilità $\frac{k}{N}$. Questo garantirebbe la proprietà che ogni elemento dello stream è contenuto nella riserva con la stessa probabilità, ma il numero di elementi effettivamente inseriti nella riserva potrebbe essere maggiore o minore di k .

Teorema 2.5.1. *Sia R_t il contenuto della riserva dopo che sono stati osservati i primi t elementi dello stream. Per ogni $t \geq k$ vale: $\mathbb{P}(x_i \in R_t) = \frac{k}{t}$ per ogni $i \leq t$.*

Per la dimostrazione useremo più volte il fatto che, per ogni coppia di eventi A, B tale che $\mathbb{P}(B) > 0$ vale $\mathbb{P}(A \cap B) = \mathbb{P}(B)\mathbb{P}(A | B)$.

Dimostrazione. La dimostrazione è per induzione su $t \geq k$:

- **Base:** $t = k$. Allora $\mathbb{P}(x_i \in R_t) = 1 = \frac{k}{t}$ dato che $t = k$
- **Step:** Fissato $t \geq k$, assumiamo l'ipotesi induttiva

$$P(x_i \in R_t) = \frac{k}{t}, \quad \forall i \leq t$$

e dimostriamo

$$\mathbb{P}(x_i \in R_{t+1}) = \frac{k}{t+1}, \quad \forall i \leq t+1$$

Se $i = t+1$, allora vale la tesi per costruzione (riga 7 dell'algoritmo). Se invece $i \leq t$, dato che $x_i \in R_{t+1}$ implica $x_i \in R_t$, abbiamo che $\mathbb{P}(x_i \in R_{t+1}) = \mathbb{P}(x_i \in R_{t+1}, x_i \in R_t)$. Possiamo quindi scrivere

$$\begin{aligned} \mathbb{P}(x_i \in R_{t+1}) &= \mathbb{P}(x_i \in R_{t+1}, x_i \in R_t) \\ &= \mathbb{P}(x_i \in R_t) \mathbb{P}(x_i \in R_{t+1} | x_i \in R_t) \\ &= \frac{k}{t} \cdot \mathbb{P}(x_i \in R_{t+1} | x_i \in R_t) \quad (\text{per IH}) \end{aligned}$$

Ora si osservi che, dato $x_i \in R_t$, abbiamo che $x_i \notin R_{t+1}$ implica

$x_{t+1} \in R_{t+1}$. Quindi possiamo scrivere

$$\begin{aligned}
\mathbb{P}(x_i \in R_{t+1} \mid x_i \in R_t) &= 1 - \mathbb{P}(x_i \notin R_{t+1} \mid x_i \in R_t) \\
&= 1 - \mathbb{P}(x_i \notin R_{t+1}, x_{t+1} \in R_{t+1} \mid x_i \in R_t) \\
&= 1 - \mathbb{P}(x_{t+1} \in R_{t+1} \mid x_i \in R_t) \mathbb{P}(x_i \notin R_{t+1} \mid x_{t+1} \in R_{t+1}, x_i \in R_t) \\
&= 1 - \frac{k-1}{t+1} \frac{1}{k} \\
&= \frac{t}{t+1}
\end{aligned}$$

dove

$$\mathbb{P}(x_{t+1} \in R_{t+1} \mid x_i \in R_t) = \mathbb{P}(x_{t+1} \in R_{t+1}) = \frac{k}{t+1}$$

per costruzione dell'algoritmo e

$$\mathbb{P}(x_i \notin R_{t+1} \mid x_{t+1} \in R_{t+1}, x_i \in R_t) = \frac{1}{k}$$

dato che x_i ha probabilità uniforme di essere selezionato dalla riserva per far posto a x_{t+1} . Quindi

$$\mathbb{P}(x_i \in R_{t+1}) = \frac{k}{t} \frac{t}{t+1} = \frac{k}{t+1}$$

che conclude la dimostrazione.

□

2.6 Algoritmo di Karger per il taglio minimo

Si consideri un grafo non orientato $G = (V, E)$. Una partizione $S, T \subseteq V$ dei vertici di G (dunque $S \cap T \equiv \emptyset$ e $S \cup T \equiv V$) è detta *ammissibile* se $S, T \not\equiv \emptyset$. Un *taglio* in G è un insieme $\Gamma(S, T) \equiv \{(u, v) \in E \mid u \in S, v \in T\}$ per una partizione ammissibile S, T dei vertici di G . In un grafo non pesato, il corso di un taglio corrisponde alla sua cardinalità $|\Gamma(S, T)|$. In quanto segue consideriamo grafi non pesati e senza cappi; tuttavia, ammettiamo la presenza di archi multipli fra coppie di nodi (*multigrafo*). In questo caso, E è un multinsieme di archi, poiché archi fra coppie di vertici distinti possono essere presenti con molteplicità diverse.

Il problema del taglio minimo (MinCut) in un multigrafo è definito nel modo seguente.

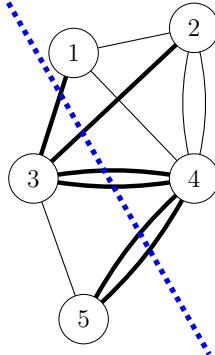


Figura 2.1: Un taglio in un multigrafo. Il taglio evidenziato è formato da 6 archi e corrisponde al multinsieme $\Gamma(S, T)$, dove $S = \{1, 2, 4\}$ e $T = \{3, 5\}$

Problema MinCut.

Istanza: Un multigrafo $G = (V, E)$.

Soluzione: Una partizione ammissibile S, T di V che minimizza il costo $|\Gamma(S, T)|$.

Il problema MinCut ha tantissime applicazioni. Per esempio, in un sistema distribuito dove i nodi rappresentano i processi e gli archi canali di comunicazione fra di essi, il taglio minimo corrisponde ad assegnare i processi a due CPU in modo che la comunicazione inter-processore – tipicamente lenta – sia minimizzata. Una seconda applicazione è la segmentazione di immagini. Qui i nodi rappresentano pixel e gli archi del grafo connettono pixel simili. Il taglio minimo corrisponde allora a una segmentazione dell’immagine in due parti tra loro il più dissimili possibile.

Il problema MinCut è facilmente risolvibile in tempo polinomiale deterministico, per esempio usando l’algoritmo di **Stoer-Wagner**, il quale ha un tempo di esecuzione dell’ordine $\mathcal{O}(|E||V| + |V|^2 \log |V|)$. Mostriamo ora un semplice algoritmo probabilistico Monte Carlo, l’algoritmo di Karger, che trova il taglio minimo con probabilità almeno $1 - \epsilon$ in tempo pari a $\mathcal{O}(|E||V|^2 \log \frac{1}{\epsilon})$.

L’algoritmo di Karger è basato sull’operazione di contrazione di un arco (vedi Figura 2.2). La *contrazione* di un arco (u, v) in un multigrafo G produce un multigrafo $G \setminus (u, v)$ definito come risultato delle seguenti operazioni:

1. Un nuovo vertice z è aggiunto al grafo

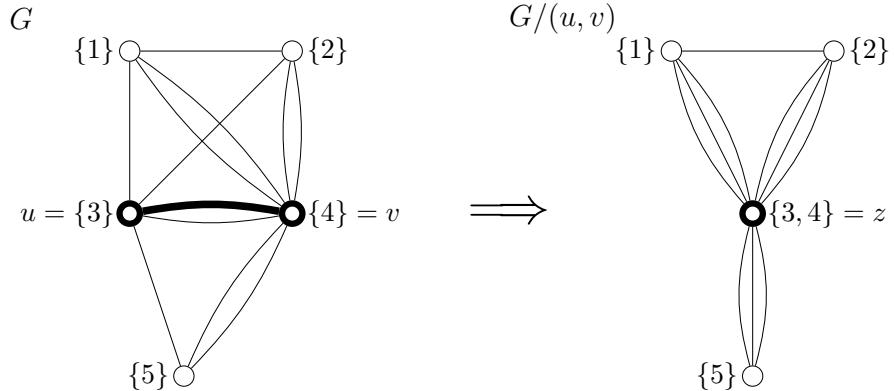


Figura 2.2: La contrazione di un arco in un multigrafo.

2. Ogni arco $(w, u) \in E$ con $w \neq v$ è sostituito da un arco (w, z)
3. Ogni arco $(w, v) \in E$ con $w \neq u$ è sostituito da un arco (w, z)
4. Gli archi del tipo (u, v) e i vertici u, v sono rimossi

Nel seguito, diciamo che z è un supervertice che contiene u e v . Quando uno o entrambi i due vertici agli estremi di un arco contratto solo a loro volta supervertici, allora i nodi in essi contenuti diventano parte del nuovo supervertice.

Introduciamo ora l'algoritmo Karger “base”, il quale contrae ripetutamente archi a caso del multigrafo fino a quando il numero di supervertici rimanenti è pari a due. Dato che la contrazione di un arco riduce di uno il numero di supervertici, l'algoritmo si fermerà dopo esattamente $|V| - 2$ passi. A questo punto, l'algoritmo produce il cut corrispondente all'unica partizione ammissibile dei due supervertici (si taglano gli archi che collegano i vertici rimanenti). Dato che i supervertici del multigrafo finale corrispondono a una partizione dei vertici del multigrafo iniziale (i due supervertici dividono in due il grafo iniziale), abbiamo ottenuto un taglio del multigrafo iniziale.

Questo algoritmo può essere implementato in tempo $\mathcal{O}(|E|)$ rappresentando la sequenza di contrazioni tramite una permutazione causale degli archi di G (dettagli omessi).

Vediamo ora qual è la probabilità che questo algoritmo ritorni una partizione ammissibile S^*, T^* fissata che definisce un taglio $\Gamma^* = \Gamma(S^*, T^*)$ di costo

Algoritmo 3: Karger-Base(G)

Input: Multigrafo $G = (V, E)$ con $|V| > 2$

- 1 **while** $|V| > 2$ **do**
- 2 Scegli un arco a caso $(u, v) \in E$
- 3 $G \leftarrow G \setminus (u, v)$

Output: l'unica partizione ammissibile S, T rimasta in G

minimo k in G ; in particolare

$$k = |\Gamma^*| = \min_{(S,T)} |\Gamma(S, T)|$$

dove il minimo è su tutte le partizioni ammissibili S, T di V . Per prima cosa, si noti che l'algoritmo ritorna Γ^* se e solo se nessun arco nel taglio viene contratto. Denotiamo ora con $X_1, \dots, X_{|V|-2}$ la sequenza di archi contratti dall'algoritmo. La probabilità che il primo arco X_1 contratto sia nel taglio corrisponde a

$$\mathbb{P}(X_1 \in \Gamma^*) = \frac{k}{|E|} \leq \frac{k}{|V|k/2} = \frac{2}{|V|}$$

perché la cardinalità di E è tale che

$$|E| = \frac{1}{2} \sum_{v \in V} d_v \geq \frac{1}{2} |V| d_{\min} \geq \frac{1}{2} |V| k$$

dove d_v è il grado di $v \in V$ in G , mentre d_{\min} è il grado minimo dei nodi del grafo. In particolare, $d_{\min} \geq k$ è vero perché il costo k di un taglio minimo è sicuramente non superiore rispetto al costo d_v del taglio $\Gamma(\{v\}, V \setminus \{v\})$ per qualsiasi vertice $v \in V$. In altre parole, il numero di archi è sicuramente maggiore del numero di vertici moltiplicato per metà del grado minimo di qualsiasi vertice, e sappiamo che $d_{\min} \geq k$.

Quindi

$$\mathbb{P}(X_1 \notin \Gamma^*) \geq 1 - \frac{2}{|V|}$$

La probabilità che il secondo arco contratto X_2 non sia nel taglio Γ^* , dato

che $X_1 \notin \Gamma^*$, è

$$\begin{aligned}\mathbb{P}(X_2 \notin \Gamma^* \mid X_1 \notin \Gamma^*) &= 1 - \mathbb{P}(X_2 \in \Gamma^* \mid X_1 \notin \Gamma^*) \\ &\geq 1 - \frac{k}{(|V| - 1)k/2} \\ &= 1 - \frac{2}{|V| - 1}\end{aligned}$$

In generale, denotando con A_i l'evento $X_i \notin \Gamma^*$ per $i \geq 1$, osserviamo che nessun arco di Γ^* è stato già contratto nel momento in cui dobbiamo scegliere X_i se condizioniamo sugli eventi A_1, \dots, A_{i-1} . Il taglio Γ^* è dunque preservato sotto questo condizionamento. In aggiunta, al passo i -esimo, il grafo presenta $|V| - i + 1$ (super)vertici e un suo taglio minimo avrà ancora costo k condizionando su A_1, \dots, A_{i-1} : le contrazioni si limitano a restringere le scelte di partizioni ammissibili dei vertici su cui valutare il costo dei tagli, e sappiamo che Γ^* è preservato.

Di conseguenza, abbiamo che

$$\mathbb{P}(A_i \mid A_1, \dots, A_{i-1}) \geq 1 - \frac{k}{(|V| - i + 1)k/2} = 1 - \frac{2}{|V| - i + 1}$$

Dunque, indicando $\mathbb{P}(A_1 \mid A_0) = \mathbb{P}(A_1)$, dove A_0 corrisponde all'evento certo, possiamo scrivere

$$\begin{aligned}\mathbb{P}(\text{restituisce } \Gamma^*) &= \mathbb{P}(X_i \notin \Gamma^*, \forall i \in [|V| - 2]) \\ &= \mathbb{P}\left(\bigcap_{i=1}^{|V|-2} A_i\right) \\ &= \prod_{i=1}^{|V|-2} \mathbb{P}(A_i \mid A_0, \dots, A_{i-1}) \\ &\geq \prod_{i=0}^{|V|-3} \frac{|V| - i - 2}{|V| - i} \\ &= \frac{\prod_{i=1}^{|V|-2} i}{\prod_{j=3}^{|V|} j} = \frac{(|V| - 2)!}{\frac{|V|!}{2!}} = \frac{1}{\binom{|V|}{2}} \quad (\dagger)\end{aligned}$$

Possiamo amplificare la probabilità di successo eseguendo più volte l'algoritmo e scegliendo la partizione ammissibile che minimizza il costo del taglio tra

tutte quelle ottenute. In particolare, se M è un numero sufficientemente grande, allora siamo in grado di ottenere un taglio di costo minimo in una delle M esecuzioni indipendenti dell'algoritmo con probabilità almeno $1 - \epsilon$, data una probabilità massima di errore $\epsilon \in (0, 1]$. Questa idea viene implementata nell'algoritmo seguente.

Algoritmo 4: Karger(G, ϵ)

Input: Multigrafo $G = (V, E)$ con $|V| > 2$, parametro di confidenza

- 1 $M \leftarrow \left\lceil \binom{|V|}{2} \ln \frac{1}{\epsilon} \right\rceil$
 - 2 **for** $i = 1, \dots, M$ **do**
 - 3 $S_i, T_i \leftarrow \text{Karger-Base}(G)$
 - 4 $j \in \arg \min_{i=1, \dots, M} |\Gamma(S_i, T_i)|$
-

Ripetendo l'algoritmo base per $M = \left\lceil \binom{|V|}{2} \ln \frac{1}{\epsilon} \right\rceil = \mathcal{O}(|V|^2 \ln \frac{1}{\epsilon})$ volte e scegliendo un taglio di costo minimo tra quelli prodotti, la probabilità che questo non abbia costo ottimo è

$$\begin{aligned}
\mathbb{P}(\text{Karger}(G, \epsilon) \text{ fallisce}) &= \mathbb{P}(|\Gamma(S_i, T_i)| > k, \forall i \in [M]) && (\text{nessun taglio prodotto è minimo}) \\
&\leq \mathbb{P}(\Gamma(S_i, T_i) \not\equiv \Gamma^*, \forall i \in [M]) && (\text{nessun taglio prodotto è } \Gamma^*) \\
&= \mathbb{P}(\text{Karger-Base}(G) \text{ non restituisce } \Gamma^*)^M && (\text{indipendenza}) \\
&= \left(1 - \frac{1}{\binom{|V|}{2}}\right)^M && (\text{per } \dagger) \\
&\leq e^{-\ln \frac{1}{\epsilon}} = \epsilon
\end{aligned}$$

dove abbiamo usato la maggiorazione $1 - x \leq e^{-x}$ per ogni $x \in \mathbb{R}$. Il tempo totale di esecuzione, considerando anche il costo di ciascuna contrazione, è quindi $\mathcal{O}(|E||V|^2 \ln \frac{1}{\epsilon})$.

Una versione più sofisticata, nota come algoritmo di Karger-Stein, trova un taglio di costo minimo con probabilità almeno $1 - \epsilon$ in tempo $\mathcal{O}\left((|V| \ln |V|)^2 \ln \frac{1}{\epsilon}\right)$.

Capitolo 3

Hashing e Randomizzazione

3.1 Hashing Universale

Sia \mathcal{U} un insieme detto *universo*, con $|\mathcal{U}| = m \gg 1$. Consideriamo una tabella hash T di dimensione $1 < n \ll m$ e una funzione di hash $h : \mathcal{U} \rightarrow \{0, \dots, n - 1\}$.

Supponiamo di memorizzare un arbitrario sottoinsieme $S \subseteq \mathcal{U}$ nella tabella di hash usando la funzione h . Ovvero, memorizziamo ogni $u \in S$ nella posizione $h(u)$ di T . Dato che la tabella di hash è molto più piccola della cardinalità di \mathcal{U} , potranno avvenire delle collisioni. Una *collisione* consiste nell'evento in cui due elementi distinti $u, v \in S$ sono tali che $h(u) = h(v)$. Ciò significa che u e v andrebbero memorizzati nella stessa posizione della tabella hash T . Questo problema viene solitamente gestito utilizzando, per esempio, delle liste associate a ogni posizione di T . Una conseguenza di questa soluzione alle collisioni riguarda il costo delle operazioni sulla tabella di hash: ogni ulteriore operazione di ricerca o inserimento sulla tabella richiederà tempo proporzionale alla lunghezza della lista associata alla posizione della tabella dove viene fatta l'operazione.

In un'analisi di caso peggiore, il tempo per eseguire un'operazione di ricerca o inserimento di un elemento $v \in \mathcal{U}$ è quindi proporzionale al massimo numero ℓ_{\max} di elementi in S diversi da v che sono stati mappati da h nella

stessa posizione di v in T . Più precisamente

$$\ell_{\max} = \max_{v \in \mathcal{U}} |\{u \in S \mid h(u) = h(v), u \neq v\}|$$

Idealmente, vorremmo avere $\ell_{\max} = \mathcal{O}(|S|/n)$, in modo da minimizzare il tempo di esecuzione di un'operazione nel caso peggiore (vogliamo che gli elementi siano distribuiti il più equamente possibile nella tabella).

Possiamo ottenere questo risultato usando la randomizzazione. Supponiamo di avere a disposizione una famiglia \mathcal{H} di funzioni $h : \mathcal{U} \rightarrow \{0, \dots, n-1\}$, e di usare una funzione di hash h che sia la realizzazione di una variabile casuale H corrispondente all'estrazione casuale (con probabilità uniforme) di un elemento da \mathcal{H} .

Supponiamo ora che \mathcal{H} soddisfi la condizione seguente

$$\mathbb{P}(H(u) = H(v)) = \frac{1}{n}, \quad \text{per ogni } u, v \in \mathcal{U}, u \neq v \quad (\dagger)$$

dove la probabilità è calcolata rispetto all'estrazione casuale della funzione di hash H da \mathcal{H} . In altre parole, ancora una volta, vogliamo che gli elementi siano il più possibile equamente distribuiti.

La condizione (\dagger) è sufficiente a garantire che

$$\max_{v \in \mathcal{U}} \mathbb{E}[|\{u \in S \mid H(u) = H(v), u \neq v\}|] \leq \frac{|S|}{n}$$

dove il valore atteso è calcolato rispetto all'estrazione di H da \mathcal{H} (il numero di collisioni atteso è minimizzato, la proprietà desiderata precedentemente). Infatti, supponiamo di aver estratto H a caso da \mathcal{H} e di averla usata per inserire $S = \{u_1, \dots, u_s\}$ nella tabella. Sia $v \in \mathcal{U}$ un elemento arbitrario che vogliamo cercare o inserire nella tabella. Definiamo le variabili casuali X_1, \dots, X_s dove $X_i = \mathbb{I}\{H(u_i) = H(v)\}$ e $\mathbb{I}\{\cdot\}$ è la funzione indicatrice di un evento, definita come

$$\mathbb{I}\{A\} = \begin{cases} 1 & \text{se } A \text{ è vero} \\ 0 & \text{altrimenti} \end{cases}$$

Per come utilizzata sopra, indica che la variabile X_i è a 1 se risulta una collisione tra u_i e v .

Si ricorda che una proprietà della funzione indicatrice di un evento A è che il suo valore atteso equivale a $\mathbb{E}[\mathbb{I}\{A\}] = \mathbb{P}(A)$. Allora

$$|\{u \in S \mid H(u) = H(v), u \neq v\}| = \sum_{i=1, u_i \neq v}^s X_i$$

Questa equazione rappresenta il numero di collisioni atteso.

Il valore atteso di questo numero è calcolato come

$$\begin{aligned} \mathbb{E} \left[\sum_{i=1, u_i \neq v}^s X_i \right] &= \sum_{i=1, u_i \neq v}^s \mathbb{E}[X_i] && (\text{linearità di } \mathbb{E}) \\ &= \sum_{i=1, u_i \neq v}^s \mathbb{P}(H(u_i) = H(v)) && (\text{definizione di } X_i) \\ &= \sum_{i=1, u_i \neq v}^s \frac{1}{n} && (\text{ipotesi su } \mathcal{H}, \text{eq. } \dagger) \\ &\leq \frac{|S|}{n} && (\text{al più } |S| \text{ elementi}) \end{aligned}$$

Non è difficile trovare famiglie \mathcal{H} che soddisfano la condizione (\dagger) . Assumiamo per semplicità che $\mathcal{U} = \{0, \dots, m-1\}$ e consideriamo la classe \mathcal{H} di tutte le funzioni $h : \{0, \dots, m-1\} \rightarrow \{0, \dots, n-1\}$. Ognuna di queste funzioni di hash corrisponde a un vettore $\mathbf{h} \in (h_0, \dots, h_{m-1}) \in \{0, \dots, n-1\}^m$ in modo che $h(u) = h_u$. Allora, il numero di vettori $\mathbf{h} \in \{0, \dots, n-1\}^m$ tali che $h_u = h_v$ è n^{m-1} , per $u, v \in \mathcal{U}$ distinti. Quindi, se estraggo H uniformemente a caso da \mathcal{H}

$$\mathbb{P}(H(u) = H(v)) = \frac{|\{\mathbf{h} \in \mathcal{H} \mid h_u = h_v\}|}{|\mathcal{H}|} = \frac{n^{m-1}}{n^m} = \frac{1}{n}$$

e la condizione (\dagger) è soddisfatta. D'altra parte, la classe \mathcal{H} non è utilizzabile in pratica in quanto mi servono $\lceil \log_2 |\mathcal{H}| \rceil = \Theta(m \log n)$ bit per memorizzare ciascuna $h \in \mathcal{H}$, e stiamo assumendo che $m \gg 1$.

Per eliminare situazioni di questo genere, aggiungiamo alla condizione (\dagger) una richiesta ulteriore, ovvero che ogni $h \in \mathcal{H}$ possa essere rappresentata con al più $\Theta(\log m)$ bit (che è anche lo spazio che occorre per rappresentare un elemento arbitrario di \mathcal{U}) e calcolata in modo efficiente (chiamiamo questa

proprietà (\ddagger)). Una famiglia \mathcal{H} di funzioni $h : \mathcal{U} \rightarrow \{0, \dots, n - 1\}$ che soddisfa entrambe queste condizioni è detta una *famiglia universale di funzioni di hash*.

Dimostriamo ora l'esistenza di famiglie universali. Oltre a $\mathcal{U} = \{0, \dots, m - 1\}$ assumiamo anche $n = p$ per un qualche p primo. Rappresentiamo ciascun elemento $u \in \{0, \dots, m - 1\}$ come un numero $[u]_p$ in base p . Più precisamente, $[u]_p = x = (x_1, \dots, x_r)$ dove $x_i \in \{0, \dots, p - 1\}$ e r è il più piccolo intero tale che $p^r \geq m$. Ovvero

$$r = \left\lceil \frac{\log_2 m}{\log_2 p} \right\rceil$$

Sia $\mathcal{A} = \{0, \dots, p - 1\}^r$ l'insieme usato per rappresentare gli elementi di \mathcal{U} . Introduciamo ora la famiglia di funzioni di hash $\mathcal{H} = \{h_a \mid a \in \mathcal{A}\}$ di tipo $h_a : \mathcal{A} \rightarrow \{0, \dots, p - 1\}$ (funzioni che vanno da ogni possibile valore nella rappresentazione scelta a una singola cifra) e definite come

$$h_a(u) = \left(\sum_{i=1}^r a_i x_i \right) \mod p, \quad \text{dove } x = [u]_p$$

Ovvero, ogni cifra del valore da “hashare” è moltiplicata per la cifra nella posizione corrispondente definita all'interno della funzione, vengono tutte sommate e poi messe modulo p , ovvero la base della rappresentazione (in binario, sarebbero due array di bit, ogni posizione di uno moltiplicata con l'altro, tutti i valori sommati e riportati in base 2).

Si noti che posso rappresentare ogni h_a con

$$\lceil \log_2 |\mathcal{A}| \rceil = \Theta(r \log p) = \Theta\left(\frac{\log m}{\log p} \log p\right) = \Theta(\log m)$$

bit e posso calcolare h_a in modo efficiente (proprietà (\ddagger))). Intuitivamente, si può vedere come sia facile da calcolare e la funzione di hash consiste effettivamente di $\log m$ valori, dove m è il massimo valore che si vuole considerare in input (per definizione di \mathcal{U}).

Per verificare la condizione (\dagger) faremo uso del lemma seguente.

Lemma 3.1.1. *Per ogni primo p , per α, β e z interi*

$$(z \not\equiv 0 \pmod p) \wedge (\alpha z \equiv \beta z \pmod p) \implies \alpha \equiv \beta \pmod p$$

Dimostrazione. Chiaramente, $\alpha z \equiv \beta z \pmod{p}$ se e solo se $z(\alpha - \beta) \equiv 0 \pmod{p}$. Inoltre, $z \not\equiv 0 \pmod{p}$ e $z(\alpha - \beta) \equiv 0 \pmod{p}$ implicano $(\alpha - \beta) \equiv 0 \pmod{p}$. Infatti, se p è primo e z non contiene p come fattore, allora $\alpha - \beta$ lo deve contenere (se invece p non fosse primo, allora z e $\alpha - \beta$ potrebbero spartirsi i fattori primi di p). \square

Siamo pronti ora a dimostrare che per \mathcal{H} vale la proprietà (\dagger) . Dato che, come abbiamo già osservato, per la stessa classe valeva anche la proprietà (\ddagger) , concludiamo che \mathcal{H} è una famiglia universale di funzioni di hash.

Teorema 3.1.1. \mathcal{H} è tale che per ogni $u, v \in \mathcal{U}$ distinti, la frazione di elementi $a \in \mathcal{A}$ tali che $h_a(u) = h_a(v)$ è al più $\frac{1}{p}$. Quindi, se H è estratta a caso da \mathcal{H} , allora

$$\mathbb{P}(H(u) = H(v)) \leq \frac{1}{p}$$

Dimostrazione. Siano $x = [u]_p$ e $y = [v]_p$. Sappiamo che $x \neq y$ poiché $u \neq v$. Allora, esiste almeno una coordinata $j \in \{1, \dots, r\}$ tale che $x_j \neq y_j$. Per estrarre H a caso in \mathcal{H} prendiamo $a \in \mathcal{A}$ estraendo a caso ciascuna coordinata $a_i \in \{0, \dots, p-1\}$. Per qualunque estrazione dei valori a_i sulle coordinate $i \neq j$ abbiamo che

$$\begin{aligned} h_a(u) = h_a(v) &\iff \left(\sum_{i=1}^r a_i x_i \equiv \sum_{i=1}^r a_i y_i \pmod{p} \right) \quad (\text{def. } h_a) \\ &\iff a_j \underbrace{(y_j - x_j)}_z \equiv \underbrace{\sum_{i:i \neq j} a_i (x_i - y_i)}_k \pmod{p} \\ &\iff a_j z \equiv k \pmod{p} \end{aligned}$$

Dimostriamo ora che questa congruenza può essere soddisfatta da non più di un valore di a_j . In altre parole, per qualunque scelta dei valori di $a_i \in \{0, \dots, p-1\}$ con $i \neq j$, esiste al più un unico valore di $a_j \in \{0, \dots, p-1\}$ tale che $a_j z \equiv k \pmod{p}$ sia vera. Per assurdo, supponiamo che esistano a_j, a'_j valori distinti per la coordinata j che soddisfino entrambi la congruenza. Allora avremmo anche che $a_j z \equiv a'_j z \pmod{p}$. Dato che $z \not\equiv 0 \pmod{p}$ per ipotesi (perché $x_j \neq y_j$ e $0 \leq x_j, y_j < p$), il Lemma 3.1.1 implica che $a_j \equiv a'_j \pmod{p}$. Siccome $0 \leq a_j, a'_j < p$, ciò significa che $a_j = a'_j$ e si ha una contraddizione. Quindi, c'è al più un solo valore della coordinata j che può rendere $h_a(u) = h_a(v)$ vera e la probabilità di estrarre questo valore fra i p possibili per a_j è al più $\frac{1}{p}$. \square

Nota a margine: per maggiore chiarezza, potremmo argomentare più formalmente per arrivare alla medesima conclusione del teorema precedente. Seguendo lo stesso ragionamento, sappiamo che

$$\{h_a \mid h_a(u) = h_a(v), a \in \mathcal{A}\} = \{h_a \mid a_j z \equiv k \pmod{p}, a \in \mathcal{A}\}$$

per via dell'equivalenza tra le due condizioni, e che dunque

$$\begin{aligned} |\{h_a \mid h_a(u) = h_a(v), a \in \mathcal{A}\}| &= |\{h_a \mid a_j z \equiv k \pmod{p}, a \in \mathcal{A}\}| \\ &= \sum_{a \in \mathcal{A}} \mathbb{I}\{a_j z \equiv k \pmod{p}\} \\ &= \sum_{a_i \in \{0, \dots, p-1\}, i \neq j} \underbrace{\sum_{a_j \in \{0, \dots, p-1\}} \mathbb{I}\{a_j z \equiv k \pmod{p}\}}_{\leq 1} \\ &\leq \sum_{a_i \in \{0, \dots, p-1\}, i \neq j} 1 \\ &= |\{(a_1, \dots, a_{j-1}, a_{j+1}, \dots, a_r) \mid a_i \in \{0, \dots, p-1\}, \forall i \neq j\}| \\ &= p^{r-1} \end{aligned}$$

dove la disegualanza è dovuta all'esistenza di al più un valore di a_j che soddisfa la congruenza. Quindi, ricordando che $|\mathcal{H}| = |\mathcal{A}| = p^r$, la frazione di elementi $a \in \mathcal{A}$ tali che $h_a(u) = h_a(v)$ è

$$\frac{|\{h_a \mid h_a(u) = h_a(v), a \in \mathcal{A}\}|}{|\mathcal{H}|} \leq \frac{p^{r-1}}{p^r} = \frac{1}{p}$$

3.2 Conteggio approssimato

Supponiamo di dover trovare, in una tabella di grandi dimensioni, tutti gli elementi che si ripetono più di un certo numero di volte. Per esempio, vogliamo trovare i prodotti visualizzati più frequentemente su Amazon, oppure le parole cercate più frequentemente su Google. Questo problema prende il nome di ricerca degli *heavy hitters* e, in astratto, richiede di trovare in una tabella di n interi tutti gli interi che si ripetono almeno n/k volte, dove $n \gg k$. Si noti che ci possono essere al più k heavy hitters e potrebbe non essercene neanche uno.

Partiamo da una versione più semplice del problema: vogliamo trovare nella tabella un numero che si ripete almeno $n/2$ volte, sapendo che tale numero è presente. Chiaramente, questo valore deve corrispondere alla mediana di

tutti i valori nella tabella e posso trovarlo in tempo $\mathcal{O}(n)$ con un algoritmo deterministico. Vediamo ora un semplicissimo algoritmo ad hoc che trova tale valore scandendo l'array una sola volta dall'inizio alla fine e usando una memoria ausiliaria sublineare (algoritmi di questo tipo si chiamano *streaming*).

Algoritmo 5: Boyer-Moore

Input: Array A

```

1  $c \leftarrow 0$                                 // Inizializza contatore maggioranza
2  $v \leftarrow \text{NULL}$                          // Inizializza maggioranza corrente
3 for  $i = 1, \dots, n$  do
4   if  $c = 0$  then
5     // Nessuna maggioranza
6      $v \leftarrow A[i]$ 
7      $c \leftarrow c + 1$ 
8   else
9     if  $A[i] = v$  then
10       // Incremento maggioranza corrente
11        $c \leftarrow c + 1$ 
12     else
13       // Decremento maggioranza corrente
14        $c \leftarrow c - 1$ 

```

Non è difficile verificare che quando l'algoritmo termina il valore corrente di v corrisponde al valore di maggioranza nella tabella. Ora ci chiediamo se esista una soluzione streaming anche per il problema di trovare gli heavy hitters. In realtà è possibile dimostrare che non esiste un algoritmo streaming che risolve il problema di ricerca degli heavy hitters con memoria ausiliaria sublineare (dimostrazione omessa).

Per riuscire a trovare una soluzione streaming, rilassiamo il problema originario introducendo una versione approssimata. Nel problema di ricerca di heavy hitters ϵ -approssimati (indicato ϵ -HH) abbiamo una tabella A di lunghezza n e due parametri k e ϵ con $\frac{1}{n} < \epsilon < \frac{1}{k}$. L'algoritmo deve produrre una lista di valori tali che:

1. Ogni valore che compare in A almeno $\frac{n}{k}$ volte è nella lista
2. Ogni valore nella lista compare almeno $\frac{n}{k} - \epsilon n$ volte in A

L'algoritmo che proponiamo è probabilistico e usa memoria ausiliaria $\Theta\left(\frac{\ln n}{\epsilon}\right)$.

Per risolvere il problema ϵ -HH utilizzeremo una struttura dati probabilistica chiamata count-min sketch. Questa struttura supporta due operazioni:

- $\text{INC}(x)$ che incrementa il contatore associato a x
- $\text{COUNT}(x)$ che ritorna il numero di volte che $\text{INC}(x)$ è stato invocato

La struttura dati è composta da ℓ tabella di hash, ciascuna di dimensione b . Siano $h_1, \dots, h_\ell : \{1, \dots, n\} \rightarrow \{0, \dots, b-1\}$ le funzioni hash associate alle ℓ tabelle. Ogni tabella di hash comprime la tabella di n elementi in una di dimensione $b \ll n$. Le ℓ tabelle diverse servono a ridurre la probabilità di errore dovuto a collisione. Il codice per le due operazioni e per la routine principale $\text{SELECTEL}(A, k)$ è estremamente semplice.

Algoritmo 6: Count-Min Sketch

```

1 Crea matrice CMS[ $\ell$ ][ $b$ ]
2 Procedure INC( $x$ )
3   for  $i = 1, \dots, \ell$  do
4     Incrementa CMS[ $i$ ][ $h_i(x)$ ]

5 Procedure COUNT( $x$ )
6   return  $\min_{i=1, \dots, \ell} \text{CMS}[i][h_i(x)]$ 

7 Procedure SELECTEL( $A, k$ )
8   Crea lista vuota
9   for  $t = 1, \dots, n$  do
10    Leggi il prossimo elemento  $x_t = A[t]$ 
11    Esegui INC( $x_t$ )
12    if COUNT( $x_t$ )  $\geq \frac{n}{k}$  then
13      Aggiungi  $x_t$  alla lista (se non è già presente)

14 return la lista

```

Sia x un valore che compare almeno una volta nella tabella A e sia n_x il numero di occorrenze di x in A . Dato che $b \ll n$ ci saranno delle collisioni, ovvero $h(x) = h(y)$ con $x \neq y$. Questo significa che

$$n_x \leq \text{CMS}[i][h_i(x)], \quad i = 1, \dots, \ell$$

Infatti $\text{INC}(x)$ verrà chiamata esattamente n_x volte, ma, a causa delle collisioni, due chiamate $\text{INC}(x)$ e $\text{INC}(y)$ tali che $h_i(x) = h_i(y)$ incrementeranno

lo stesso contatore. Quindi, dato che ogni $\text{CMS}[i][h_i(x)]$ sovrastima n_x è sensato utilizzare come valore di $\text{COUNT}(x)$ la più piccola di tali stime.

Analizziamo ora la probabilità di errore del count-min sketch quando le funzioni di hash h_1, \dots, h_ℓ sono estratte a caso e in modo indipendente da una famiglia universale \mathcal{H} di funzioni hash. Usiamo la notazione H_1, \dots, H_ℓ per indicare che le funzioni sono variabili casuali opportunamente definite. Dato x siano Z_1, \dots, Z_ℓ le variabili casuali $Z_i = \text{CMS}[i][H_i(x)]$ dove la probabilità è rispetto all'estrazione di H_i da \mathcal{H} . Allora

$$Z_i = n_x + \sum_{y \neq x} n_y \mathbb{I}\{H_i(y) = H_i(x)\}$$

Ora, dato che \mathcal{H} è una famiglia universale

$$\mathbb{P}(H_i(x) = H_i(y)) \leq \frac{1}{b}, \quad i = 1, \dots, \ell$$

Quindi

$$\begin{aligned} \mathbb{E}[Z_i] &= n_x + \sum_{y \neq x} n_y \mathbb{P}(H_i(y) = H_i(x)) && (\text{def. di } \mathbb{I}) \\ &\leq n_x + \sum_{y \neq x} \frac{n_y}{b} && (\mathcal{H} \text{ famiglia universale}) \\ &\leq n_x + \frac{n}{b} \end{aligned}$$

Introduciamo le variabili casuali non negative $X_i = Z_i - n_x$. Scegliendo $b = \frac{e}{\epsilon}$ abbiamo che $\mathbb{E}[X_i] \leq \frac{\epsilon n}{e}$. Applicando la diseguaglianza di Markov (2.1.2) alle X_i otteniamo quindi

$$\mathbb{P}(Z_i \geq n_x + \epsilon n) = \mathbb{P}\left(X_i \geq e \frac{\epsilon n}{e}\right) \leq \frac{1}{e} \quad (\dagger)$$

Ora, dato che le funzioni hash H_1, \dots, H_ℓ sono indipendenti, anche le Z_1, \dots, Z_ℓ sono indipendenti e quindi, in particolare, gli eventi $Z_i \geq n_x + \epsilon n$ ($i =$

$1, \dots, \ell$) sono indipendenti. Questo implica che

$$\begin{aligned}
\mathbb{P}(\text{COUNT}(x) \geq n_x + \epsilon n) &= \mathbb{P}\left(\min_{i=1, \dots, \ell} Z_i \geq n_x + \epsilon n\right) \\
&= \prod_{i=1}^{\ell} \mathbb{P}(Z_i \geq n_x + \epsilon n) \\
&= \mathbb{P}\left(\bigwedge_{i=1, \dots, \ell} (Z_i \geq n_x + \epsilon n)\right) \\
&\leq e^{-\ell} \tag{per †}
\end{aligned}$$

Per capire i prossimi passaggi, ricordiamo che, per qualsiasi insieme di eventi A_1, \dots, A_N vale che

$$\mathbb{P}(\exists i : A_i) = \mathbb{P}(A_i \cup \dots \cup A_n) \leq \sum_{i=1}^N \mathbb{P}(A_i)$$

Dato che vogliamo conteggi corretti con alta probabilità per ogni x nella tabella A di lunghezza n

$$\begin{aligned}
\mathbb{P}(\exists x \in A : \text{COUNT}(x) \geq n_x + \epsilon n) &= \mathbb{P}\left(\bigcup_{x \in A} (\text{COUNT}(x) \geq n_x + \epsilon n)\right) \\
&\leq \sum_{x \in A} \mathbb{P}(\text{COUNT}(x) \geq n_x + \epsilon n) \\
&\leq ne^{-\ell} \leq \delta
\end{aligned}$$

per $\ell \geq \ln \frac{n}{\delta}$.

Quindi, se fissiamo $\delta = 1/100$, abbiamo che $b = \Theta(\frac{1}{\epsilon})$ e $\ell = \Theta(\log n)$. Quindi lo spazio totale utilizzato è $\Theta(\frac{1}{\epsilon} \log n)$, ovvero logaritmico nella taglia della tabella A se ϵ non dipende da n ¹. La routine `SELECTEL(A, k)` soddisfa le seguenti proprietà:

1. Ogni valore che compare almeno $\frac{n}{k}$ volte in A è nella lista
2. Con probabilità almeno del 99%, ogni valore nella lista compare almeno $\frac{n}{k} - \epsilon n$ volte in A

¹In realtà dobbiamo anche contare lo spazio utilizzato dalla lista che contiene gli heavy hitters. Questo sarà di ordine $\mathcal{O}(k/(1 - \epsilon k))$.

3.3 Proiezioni casuali

La tecnica del conteggio approssimato permette di stimare in modo efficiente la numerosità degli elementi più frequenti in una collezione. Vediamo ora come la tecnica delle proiezioni casuali ci permette di stimare in modo efficiente le distanze fra coppie di punti nello spazio Euclideo d -dimensionale quando d è grande.

Ricordiamo che la distanza Euclidea fra due punti $\mathbf{x}, \mathbf{x}' \in \mathbb{R}^d$ è calcolata come

$$\|\mathbf{x} - \mathbf{x}'\| = \sqrt{\sum_{i=1}^d (x_i - x'_i)^2}$$

In molte applicazioni i dati possono essere rappresentati come vettori di numeri. Due esempi importanti sono le immagini (ogni coordinata è un pixel) e i testi (ogni coordinata è una parola del dizionario e il valore della coordinata è la frequenza con la quale la parola compare nel testo). Se consideriamo l'elenco dei film disponibili su Netflix come un dizionario, allora anche un utente di Netflix può essere visto come un vettore di numeri dove ogni coordinata è un film e il valore della coordinata rappresenta una valutazione del film da parte dell'utente.

In tutti questi casi, possiamo interpretare la vicinanza di due punti in \mathbb{R}^d come una misura della similarità fra gli elementi (immagini, testi, utenti) che i punti rappresentano. Quindi la capacità di calcolare in modo efficiente qual è il punto in un insieme più vicino a un dato punto (*nearest neighbor*) diventa fondamentale per, ad esempio, suggerire film a nuovi utenti basandosi sui film apprezzati da utenti che hanno un profilo simile (ovvero, le loro codifiche in \mathbb{R}^d sono vicine in termini di distanza Euclidea).

Problema Nearest Neighbor.

Istanza: Un insieme finito $S \subset \mathbb{R}^d$ e un punto $\mathbf{x} \in \mathbb{R}^d$.

Soluzione: $\arg \min_{\mathbf{x}' \in S} \|\mathbf{x} - \mathbf{x}'\|$.

Purtroppo, trovare il nearest neighbor in d dimensioni diventa computazionalmente costoso quando $d \gg 1$, come di solito succede nelle applicazioni interessanti. Per esempio, se $|S| = n$ e voglio risolvere il problema nearest neighbor calcolando le distanza fra \mathbf{x} e i punti di S impiegherò un tempo dell'ordine di nd . Se devo risolvere il problema ogni volta che viene aggiunto un utente a S impiegherò tempo nell'ordine di $\sum_{t=1}^n (td) = \Theta(n^2 d)$.

Per ovviare a questo problema mostriamo che per ogni $0 < \epsilon, \delta < 1$ esiste $k = \mathcal{O}\left(\frac{1}{\epsilon^2} \ln \frac{|S|}{\delta}\right)$ ed esiste una classe \mathcal{F} di funzioni $f : \mathbb{R}^d \rightarrow \mathbb{R}^k$ tale che

$$(1 - \epsilon) \|\mathbf{x} - \mathbf{x}'\|^2 \leq \|f(\mathbf{x}) - f(\mathbf{x}')\|^2 \leq (1 + \epsilon) \|\mathbf{x} - \mathbf{x}'\|^2, \quad \mathbf{x}, \mathbf{x}' \in S$$

con probabilità almeno $1 - \delta$ rispetto all'estrazione di $f \in \mathcal{F}$.

Per dimostrare questo risultato utilizziamo una tecnica simile al conteggio approssimato. Ovvero, usiamo k funzioni casuali analoghe alle funzioni hash del conteggio approssimato. Queste funzioni sono rappresentate da k vettori casuali $\mathbf{Z}_1, \dots, \mathbf{Z}_k \in \mathbb{R}^d$ estratti in un modo che spiegheremo a breve. La funzione casuale associata al vettore \mathbf{Z}_j è definita come

$$f_j(\mathbf{x}) = \mathbf{Z}_j^\top \mathbf{x} = \sum_{i=1}^d Z_{j,i} x_i$$

Il prodotto scalare $\mathbf{Z}_j^\top \mathbf{x}$ calcola la lunghezza della proiezione di \mathbf{x} su \mathbf{Z}_j moltiplicata per la lunghezza di \mathbf{Z}_j . L'idea è quella di calcolare una sorta di "impronta" molto più piccola del vettore originale (uno scalare al posto che d dimensioni), conservandone le proprietà essenziali, usata per fare i calcoli; viene ripetendo il calcolo su k variabili casuali per minimizzare l'errore.

Quindi, usando la funzione f_j possiamo approssimare la distanza $\|\mathbf{x} - \mathbf{x}'\|$ fra i vettori \mathbf{x} e \mathbf{x}' con la differenza $|f_j(\mathbf{x}) - f_j(\mathbf{x}')|$ fra numeri reali $f_j(\mathbf{x})$ e $f_j(\mathbf{x}')$. Per ridurre l'errore di approssimazione utilizziamo k funzioni indipendenti invece di una sola.

I vettori \mathbf{Z}_j sono ottenuti generando ciascuna componente $Z_{j,i}$ per $i = 1, \dots, d$ con estrazioni indipendenti da una distribuzione di probabilità con media zero e varianza uno, cioè

$$\mathbb{E}[Z_{j,i}] = 0 \quad \text{e} \quad \text{Var}[Z_{j,i}] = 1, \quad j = 1, \dots, k, \quad i = 1, \dots, d$$

Quindi

$$\begin{aligned}
\mathbb{E} \left[(f_j(\mathbf{x}) - f_j(\mathbf{x}'))^2 \right] &= \mathbb{E} \left[\left(\sum_{i=1}^d (x_i - x'_i) Z_{j,i} \right)^2 \right] && (\text{def. } f_j) \\
&= \mathbb{E} \left[\sum_{r=1}^d \sum_{s=1}^d (x_r - x'_r) (x_s - x'_s) Z_{j,r} Z_{j,s} \right] \\
&= \mathbb{E} \left[\sum_{i=1}^d (x_i - x'_i)^2 Z_{j,i}^2 \right] + \mathbb{E} \left[\sum_{r,s:r \neq s} (x_r - x'_r) (x_s - x'_s) Z_{j,r} Z_{j,s} \right] \\
&= \sum_{i=1}^d (x_i - x'_i)^2 \mathbb{E} [Z_{j,i}^2] + \sum_{r,s=1}^d (x_r - x'_r) (x_s - x'_s) \mathbb{E} [Z_{j,r}] \mathbb{E} [Z_{j,s}] \\
&= \sum_{i=1}^d (x_i - x'_i)^2 \text{Var} [Z_{j,i}] && (\dagger) \\
&= \sum_{i=1}^d (x_i - x'_i)^2 = \|\mathbf{x} - \mathbf{x}'\|^2
\end{aligned}$$

dove (\dagger) vale perché le $Z_{j,i}$ hanno media zero e quindi

$$\text{Var} [Z_{j,i}] = \mathbb{E} [(Z_{j,i} - \mathbb{E} [Z_{j,i}])^2] = \mathbb{E} [Z_{j,i}^2]$$

Questo dimostra che posso usare $(f_j(\mathbf{x}) - f_j(\mathbf{x}'))^2$ per stimare la distanza quadrata $\|\mathbf{x} - \mathbf{x}'\|^2$ (abbiamo dimostrato che le approssimazioni create mantengono le misure relative dei vettori).

Definiamo ora la proiezione casuale $f : \mathbb{R}^d \rightarrow \mathbb{R}^k$

$$f(\mathbf{x}) = \left(\frac{f_1(\mathbf{x})}{\sqrt{k}}, \dots, \frac{f_k(\mathbf{x})}{\sqrt{k}} \right)$$

Si noti che $f(\mathbf{x}) = M\mathbf{x}$ dove M è la matrice casuale $k \times d$ avente $\mathbf{Z}_1/\sqrt{k}, \dots, \mathbf{Z}_k/\sqrt{k}$ come righe. Questo implica che f è una trasformazione lineare, ovvero $f(a\mathbf{x} + b\mathbf{x}') = af(\mathbf{x}) + bf(\mathbf{x}')$ per ogni $\mathbf{x}, \mathbf{x}' \in \mathbb{R}^d$ e $a, b \in \mathbb{R}$. Quindi

$$\mathbb{E} [\|f(\mathbf{x}) - f(\mathbf{x}')\|^2] = \mathbb{E} [\|f(\mathbf{x} - \mathbf{x}')\|^2] = \frac{1}{k} \sum_{j=1}^k \mathbb{E} [f_j(\mathbf{x} - \mathbf{x}')^2] = \|\mathbf{x} - \mathbf{x}'\|^2$$

dato che ciascun f_j è uno stimatore della distanza quadrata (stiamo facendo la media delle k distanze calcolate, per ridurre l'errore).

Ora, detto $\mathbf{v} = \mathbf{x} - \mathbf{x}'$ e usando sempre il fatto che f è lineare

$$\frac{\|f(\mathbf{x}) - f(\mathbf{x}')\|^2}{\|\mathbf{x} - \mathbf{x}'\|^2} = \left\| f\left(\frac{\mathbf{v}}{\|\mathbf{v}\|}\right) \right\|^2$$

Quindi, se vogliamo dimostrare che

$$(1 - \epsilon)\|\mathbf{x} - \mathbf{x}'\|^2 \leq \|f(\mathbf{x}) - f(\mathbf{x}')\|^2 \leq (1 + \epsilon)\|\mathbf{x} - \mathbf{x}'\|^2 \quad \mathbf{x}, \mathbf{x}' \in \mathbb{R}^d$$

possiamo equivalentemente dimostrare che $1 - \epsilon \leq \|f(\mathbf{v})\|^2 \leq 1 + \epsilon$ per ogni $\mathbf{v} \in \mathbb{R}^d$ tale che $\|\mathbf{v}\| = 1$.

A questo punto ci serve fare assunzioni sulla distribuzione delle variabili casuali $Z_{j,i}$. Assumiamo quindi che le $Z_{j,i}$ abbiano una distribuzione Normale (ovvero, Gaussiana con media zero e varianza uno). Per le proprietà della Normale, vale che per ogni $\epsilon, \delta > 0$ fissati e per ogni $\mathbf{v} \in \mathbb{R}^d$ di norma unitaria

$$\mathbb{P}(|\|f(\mathbf{v})\|^2 - 1| > \epsilon) \leq \delta, \quad \text{per } k = \mathcal{O}\left(\frac{1}{\epsilon^2} \ln \frac{1}{\delta}\right) \quad (\ddagger)$$

dove la probabilità è calcolata rispetto all'estrazione delle $\{Z_{j,i} \mid j = 1, \dots, k, i = 1, \dots, d\}$.

Si noti che

$$\|f(\mathbf{v})\|^2 = \frac{1}{k} \sum_{j=1}^k \left(\mathbf{Z}_j^\top \mathbf{v} \right)^2$$

e inoltre

$$\mathbb{E} \left[\left(\mathbf{Z}_j^\top \mathbf{v} \right)^2 \right] = \mathbf{v}^\top \mathbb{E} [\mathbf{Z}_j \mathbf{Z}_j^\top] \mathbf{v} = \mathbf{v}^\top I \mathbf{v} = \|\mathbf{v}\|^2 = 1$$

dove abbiamo usato il fatto che la matrice $M = \mathbb{E} [\mathbf{Z}_j \mathbf{Z}_j^\top]$ ha componenti $M_{r,s} = \mathbb{E} [Z_{j,r} Z_{j,s}]$ tali che

$$M_{r,s} = \begin{cases} 0 & \text{se } r \neq s \\ 1 & \text{altrimenti} \end{cases}$$

Quindi le variabili casuali $V_j = \left(\mathbf{Z}_j^\top \mathbf{v} \right)^2$ per $j = 1, \dots, k$ sono i.i.d. con media $\mu = 1$ e (\ddagger) può essere riscritta come

$$\mathbb{P} \left(\left| \frac{1}{k} \sum_{j=1}^k V_j - \mu \right| > \epsilon \right) \leq e^{-\mathcal{O}(k\epsilon^2)}$$

Questa diseguaglianza è analoga al Lemma di Chernoff-Hoeffding (2.1.1), con l'unica differenza che qui le V_j non hanno valori limitati. La formula (‡) ci dice quindi che un risultato analogo al Lemma di Chernoff-Hoeffding vale anche per variabili casuali del tipo $(\mathbf{Z}_j^\top \mathbf{v})^2$ dove \mathbf{Z}_j sono Normali multivariate e $\|\mathbf{v}\| = 1$.

Per capire i prossimi passaggi ricordiamo che, per qualsiasi insieme di eventi A_1, \dots, A_N vale che

$$\mathbb{P}(\exists i : A_i) = \mathbb{P}(A_1 \cup \dots \cup A_N) \leq \sum_{i=1}^N \mathbb{P}(A_i) \quad (*)$$

Nel nostro caso, ci interessano gli eventi

$$A_{\mathbf{x}, \mathbf{x}'} = \left| \frac{\|f(\mathbf{x}) - f(\mathbf{x}')\|^2}{\|\mathbf{x} - \mathbf{x}'\|^2} - 1 \right| > \epsilon$$

per ognuna delle $N = \binom{n}{2} \leq n^2$ coppie di punti distinti $\mathbf{x}, \mathbf{x}' \in S$. Allora, dato un qualunque insieme $S \subset \mathbb{R}^d$ di n punti

$$\begin{aligned} \mathbb{P}(\exists \mathbf{x}, \mathbf{x}' \in S : A_{\mathbf{x}, \mathbf{x}'}) &= \mathbb{P}\left(\bigcup_{\mathbf{x}, \mathbf{x}' \in S} A_{\mathbf{x}, \mathbf{x}'}\right) \\ &\leq \sum_{\mathbf{x}, \mathbf{x}' \in S} \mathbb{P}(A_{\mathbf{x}, \mathbf{x}'}) \quad (\text{per } (*)) \\ &\leq \sum_{\mathbf{x}, \mathbf{x}' \in S} \delta \leq n^2 \delta \end{aligned}$$

per $k = \mathcal{O}\left(\frac{1}{\epsilon^2} \ln \frac{1}{\delta}\right)$.

Da questo ne deduciamo che, per $k = \mathcal{O}\left(\frac{1}{\epsilon^2} \ln \frac{n}{\delta}\right)$ vale

$$(1 - \epsilon)\|\mathbf{x} - \mathbf{x}'\|^2 \leq \|f(\mathbf{x}) - f(\mathbf{x}')\|^2 \leq (1 + \epsilon)\|\mathbf{x} - \mathbf{x}'\|^2, \quad \forall \mathbf{x}, \mathbf{x}' \in S \quad (\star)$$

con probabilità almeno $1 - \delta$ rispetto all'estrazione delle $\{Z_{j,i} \mid j = 1, \dots, k, \quad i = 1, \dots, d\}$.

Se ci accontentiamo di un errore nella stima delle distanze del 10% con probabilità del 99% rispetto all'estrazione di tutte le $Z_{j,i}$, allora ϵ e δ sono costanti e quindi $k = \mathcal{O}(\log n)$. Il costo per mappare i punti di S in \mathbb{R}^k è $ndk = nd \ln n$ e il costo per calcolare le coppie di distanze fra un \mathbf{x} e i punti in S è $n \ln n$. Se devo risolvere il problema nearest neighbor approssimato

n volte impiegherà quindi un tempo dell'ordine di $nd \ln n + n^2 \ln n \leq n^2 d$ quando $n = \mathcal{O}(2^d)$.

Se avessimo al più $s < k$ valori non nulli in ciascuna colonna di M , allora il costo per mappare un punto di S in \mathbb{R}^k sarebbe ds . È possibile dimostrare che (\star) vale per $k = \mathcal{O}\left(\frac{1}{\epsilon^2} \ln \frac{n}{\delta}\right)$ e $s = \mathcal{O}\left(\frac{1}{\epsilon} \ln \frac{n}{\delta}\right)$.

Capitolo 4

Clustering e Randomizzazione

4.1 Correlation Clustering

Il clustering è un problema fondamentale all'interno dell'apprendimento non supervisionato. Un problema di clustering è tipicamente rappresentato tramite un insieme di elementi e una misura di similarità (o dissomiglianza) definita sugli elementi dell'insieme. Quando gli elementi sono punti in uno spazio metrico, la dissomiglianza può essere misurata tramite una funzione distanza. In un contesto più generico, quando gli elementi su cui fare clustering sono membri di un insieme astratto V , la similarità è definita da una funzione simmetrica arbitraria σ , definita su coppie di elementi distinti in V .

Il Correlation Clustering (CC) è un caso speciale noto, nel quale σ è una funzione con valori in $\{-1, +1\}$, la quale stabilisce se due elementi distinti di V sono simili o meno. L'obiettivo del CC è di raggruppare i punti in V in modo da minimizzare il numero di errori, dove un errore è dato da ogni coppia di elementi con similarità -1 appartenenti allo stesso cluster, oppure ogni coppia di elementi con similarità $+1$ appartenenti a cluster diversi. Non ci sono limitazioni sul numero di cluster o la loro dimensione: tutte le partizioni di V , incluse quelle triviali, sono valide. Dati V e σ , l'errore dato da un clustering ottimale è chiamato *Correlation Clustering Index* (CCI), indicato con OPT.

Si noti che $\text{OPT} = 0$ indica che V può essere partizionato perfettamente:

ogni coppia di elementi nello stesso cluster ha similarità +1 e ogni coppia di elementi appartenente a cluster differenti ha similarità -1. Sin dalla sua introduzione, CC ha attratto molto interesse e trova numerose applicazioni in ambiti come risoluzione delle entità, analisi delle immagini e analisi dei social media.

Minimizzare l'errore del CC è difficile e il migliore algoritmo efficiente trovato finora raggiunge un fattore di approssimazione di 2 (soluzione al più $2 \cdot \text{OPT}$; in realtà appena meno). Una semplice e elegante soluzione per approssimare CC è KwikCluster. Ad ogni iterazione KwikCluster:

- Sceglie un pivot casuale π_r da V
- Calcola tutte le similarità tra π_r e ogni altro nodo in V
- Crea un cluster C contenente π_r e tutti i punti u tali che $\sigma(\pi_r, u) = +1$

L'algoritmo poi chiama se stesso sull'insieme $V \setminus C$. Per ogni istanza di CC, KwikCluster ottiene un fattore di approssimazione al massimo di 3 (3OPT).

Algoritmo 7: KwikCluster(V_r, r)

Input: Insieme di nodi rimanenti V_r , indice di round r

```

1 if  $|V_r| = 0$  then
2   return
3 if  $|V_r| = 1$  then
4   output il cluster singleton  $V_r$ 
5   return
6 Scegli un pivot  $\pi_r$  unif. a caso da  $V_r$ 
7  $C_r \leftarrow \{\pi_r\}$            // Crea nuovo cluster con il pivot
8  $C_r \leftarrow C_r \cup \{u \in V_r \mid \sigma(\pi_r, u) = +1\}$       // Popola il cluster
9 Output cluster  $C_r$ 
10 KwikCluster( $V_r \setminus C_r, r + 1$ )

```

Chiamiamo $V \equiv \{1, \dots, n\}$ il set di nodi in input, $\mathcal{E} \equiv \binom{V}{2}$ il set di tutte le coppie $\{u, v\}$ tali che $u, v \in V$ e $u \neq v$ (coppie di nodi distinti in V), $\sigma : \mathcal{E} \rightarrow \{-1, +1\}$ la funzione di similarità binaria. Un clustering \mathcal{C} è una partizione di V in cluster disgiunti $C_i : i = 1, \dots, k$. Dati \mathcal{C} è σ , l'insieme $\Gamma_{\mathcal{C}}$ di lati sbagliati contiene tutte le coppie $\{u, v\}$ che causano un errore, ovvero tali che $\sigma(u, v) = -1$ e u, v appartengono allo stesso $C_i \in \mathcal{C}$, oppure tali che $\sigma(u, v) = +1$ e u, v appartengono a due diversi cluster di \mathcal{C} . Il costo

della partizione \mathcal{C} è $|\Gamma_{\mathcal{C}}|$. Il CCI è $\text{OPT} = \min_{\mathcal{C}} |\Gamma_{\mathcal{C}}|$, ovvero la partizione con costo minimo possibile.

Un triangolo è una qualunque tripla non ordinata $T = \{u, v, w\} \subseteq V$. Chiamiamo $e = \{u, w\}$ un qualsiasi lato di un triangolo; scriviamo $e \subset T$ e $v = T \setminus e$. Diciamo che T è un *triangolo sgradevole* (in originale *bad triangle*, non sono sicuro di come tradurlo) se i segni dei valori dati da $\sigma(u, v)$, $\sigma(u, w)$ e $\sigma(v, w)$ sono $\{+, +, -\}$ (l'ordine è irrilevante).

Chiamiamo \mathcal{T} l'insieme di tutti i triangoli sgradevoli presenti all'interno di V e definiamo $\mathcal{T}(e) \equiv \{T \in \mathcal{T} \mid e \subset T\}$. Si può facilmente vedere come il numero di triangoli sgradevoli senza lati in comune è un lower bound per OPT : indipendentemente da come vengono divisi i nodi, un triangolo sgradevole aumenta di almeno 1 il costo della partizione.

Il lemma seguente mostra come anche la somma pesata di tutti i triangoli sgradevoli è un lower bound per OPT , ammesso che la somma dei pesi di tutti i triangoli sgradevoli che incidono su qualunque singolo lato sia al più 1.

Lemma 4.1.1. *Se $\{\beta_T \geq 0 \mid T \in \mathcal{T}\}$ è un insieme di pesi sui triangoli sgradevoli tale che $\sum_{T \in \mathcal{T}(e)} \beta_T \leq 1$ per ogni $e \in \mathcal{E}$, allora $\sum_{T \in \mathcal{T}} \beta_T \leq \text{OPT}$.*

Dimostrazione. Omessa. □

Cerchiamo ora un bound per l'errore atteso di KwikCluster. Chiamiamo V_r l'insieme dei nodi rimanenti all'inizio della r -esima chiamata ricorsiva.

Sia Γ_A l'insieme di lati sbagliati per il clustering emesso da KwikCluster e sia $|\Gamma_A|$ il costo di tale partizione.

Lemma 4.1.2. *Per ogni $e \in \mathcal{E}$, $e \in \Gamma_A$ se e solo se esiste una chiamata ricorsiva r e un $T \in \mathcal{T}$ tale che $T \subseteq V_r$, $T \in \mathcal{T}(e)$ e $\pi_r = T \setminus e$.*

In altre parole, una lato e causa un errore se e solo se al passo r dell'algoritmo c'è un triangolo sgradevole T formato dal pivot π_r e dai nodi di e .

Dimostrazione. Si scelga qualsiasi lato e e sia r l'iterazione in cui almeno uno dei due nodi di e viene rimosso da V_r . Allora, KwikCluster fa un errore su e se e solo se e forma un triangolo sgradevole con π_r e $\pi_r = T \setminus e$.

Quindi, se $e \in \Gamma_A$ allora esiste un'iterazione r e un triangolo $T \subseteq V_r$ tale che: $\pi_r = T \setminus e$ e $T \in \mathcal{T}(e)$. Proviamo l'implicazione inversa analizzando

caso per caso $e = \{u, w\}$. Assumiamo $T = \{u, \pi_r, w\} \subseteq V_r$, $T \in \mathcal{T}(e)$ e $\pi_r = T \setminus e$.

Caso 1: $\sigma(u, w) = +1$: dato che $T \in \mathcal{T}$, $\sigma(\pi_r, w) \neq \sigma(\pi_r, u)$. Ma allora u e w devono finire in cluster differenti per costruzione dell'algoritmo, di conseguenza e è un errore.

Caso 2: $\sigma(u, w) = -1$: dato che $T \in \mathcal{T}$, $\sigma(\pi_r, w) = \sigma(\pi_r, u) = +1$. Ma allora u e w finiscono nello stesso cluster, di conseguenza e è un errore. \square

Il Lemma 4.1.2 implica che all'iterazione r viene fatto un errore su esattamente uno dei lati di ogni $T \in \mathcal{T}$ tale che $T \subseteq V_r$ e $\pi_r \in T$. Ricordiamo che ogni triangolo sgradevole può causare un solo errore, dato che poi il pivot $\pi_r \in T$ viene rimosso da V_r . Quindi, per una qualsiasi sequenza di pivot π_1, π_2, \dots , abbiamo che

$$|\Gamma_A| = \sum_{T \in \mathcal{T}} \mathbb{I}\{(\exists r) \mid T \subseteq V_r \wedge \pi_r \in T\}$$

Per ogni $T \in \mathcal{T}$, sia A_T l'evento $\{(\exists r) \mid T \subseteq V_r \wedge \pi_r \in T\}$ che indica il contributo a un singolo errore da parte di T .

Si noti che per ogni $e \in \Gamma_A$ e per ogni coppia di $T, T' \in \mathcal{T}(e)$ con $T \neq T'$, A_T e A'_T non possono accadere assieme in quanto e può formare un triangolo sgradevole assieme a π_r in solo uno dei due casi. Quindi, per un qualsiasi lato e

$$\sum_{T \in \mathcal{T}(e)} \mathbb{I}\{A_T \wedge e \in \Gamma_A\} = 1$$

Prendendo in considerazione il valore atteso dato dalla sequenza casuale di pivot

$$1 = \sum_{T \in \mathcal{T}(e)} \mathbb{P}(A_T \wedge e \in \Gamma_A) = \sum_{T \in \mathcal{T}(e)} \mathbb{P}(e \in \Gamma_A \mid A_T) \mathbb{P}(A_T) = \sum_{T \in \mathcal{T}(e)} \frac{1}{3} \mathbb{P}(A_T)$$

dove $\mathbb{P}(e \in \Gamma_A \mid A_T) = \frac{1}{3}$ considerato che, dato r tale che $T \subseteq V_r$ e $\pi_r \in T$, e è sbagliato solo se $\pi_r \in T \setminus e$.

Applicando il Lemma 4.1.1 con $\beta_T = \frac{1}{3} \mathbb{P}(A_T)$ per ogni $T \in \mathcal{T}$ si ottiene

$$\mathbb{E}[|\Gamma_A|] = \sum_{T \in \mathcal{T}} \mathbb{P}(A_T) = 3 \sum_{T \in \mathcal{T}} \beta_T \leq 3 \text{OPT}$$

4.2 k -Means

Consideriamo il problema di partizionare un insieme finito $\mathcal{X} \subset \mathbb{R}^d$ di punti in $k > 1$ cluster. La similarità tra punti può essere misurata come distanza Euclidea, dato che siamo in \mathbb{R}^d . Identifichiamo ogni cluster $i \in \{1, \dots, k\}$ tramite il relativo centro $\mathbf{c}_i \in \mathbb{R}^d$ (non è necessario che $\mathbf{c}_i \in \mathcal{X}$) e assegniamo ogni punto $\mathbf{x} \in \mathcal{X}$ al centro più vicino (rispetto alla distanza Euclidea).

Il costo di un punto all'interno di una partizione (clustering) $\mathcal{C} = \{\mathbf{c}_1, \dots, \mathbf{c}_k\}$ è $\phi(\mathcal{C}, \mathbf{x}) = \min_{i=1, \dots, k} \|\mathbf{x} - \mathbf{c}_i\|^2$.

Il costo della partizione \mathcal{C} è $\Phi(\mathcal{C}) = \sum_{\mathbf{x} \in \mathcal{X}} \phi(\mathcal{C}, \mathbf{x})$.

Si noti che ogni punto “costa” come il quadrato della distanza dal centro più vicino. Il k -clustering ottimale \mathcal{C}^* è una qualsiasi scelta di centri (partizione) tale che minimizzi il costo, ovvero

$$\mathcal{C}^* = \arg \min_{\mathbf{c}_1, \dots, \mathbf{c}_k \in \mathbb{R}^d} \Phi(\mathbf{c}_1, \dots, \mathbf{c}_k)$$

Non è detto che l'insieme di centri ottimali sia unico. Denotiamo con $\text{OPT}(\mathcal{X})$ il costo di \mathcal{C}^* .

Problema k -means.

Istanza: insieme $\mathcal{X} \subset \mathbb{R}^d$, parametro $k > 1$.

Soluzione: qualsiasi $\mathcal{C} \subset \mathbb{R}^d$ con $|C| = k$ tale che $\Phi(\mathcal{C}) = \text{OPT}(\mathcal{X})$.

Il problema è triviale per $k = 1$, essendoci un unico centro \mathbf{c}^* che minimizza il costo, il quale corrisponde al centroide dell'insieme \mathcal{X}

$$\mathbf{c}^* = \arg \min_{\mathbf{c} \in \mathbb{R}^d} \sum_{\mathbf{x} \in \mathcal{X}} \|\mathbf{x} - \mathbf{c}\|^2 = \frac{1}{|\mathcal{X}|} \sum_{\mathbf{x} \in \mathcal{X}} \mathbf{x}$$

Questo può essere dimostrato notando che $F(\mathbf{c}) = \sum_{\mathbf{x} \in \mathcal{X}} \|\mathbf{x} - \mathbf{c}\|^2$ è una funzione convessa con minimo nel punto in cui \mathbf{c} è il centroide. Questo implica che \mathcal{C}^* è formato dai centroidi dei cluster che lo compongono.

Il problema di k -means assume implicitamente che i punti in \mathcal{X} siano campionati da k distribuzioni Gaussiane sferiche $\mathcal{N}(\boldsymbol{\mu}_i, \sigma_i^2 I)$ per $i = 1, \dots, k$ le quali medie $\boldsymbol{\mu}_1, \dots, \boldsymbol{\mu}_k$ sono i centri e le quali varianze $\sigma_1^2, \dots, \sigma_k^2$ sono upper bound per il costo ottimale

$$\boldsymbol{\mu}_i = \arg \min_{\mathbf{c}} \mathbb{E} [\|\mathbf{X} - \mathbf{c}\|^2], \quad \text{dove } \mathbf{X} \sim \mathcal{N}(\boldsymbol{\mu}_i, \sigma_i^2 I) \quad \text{e} \quad \mathbb{E} [\Phi(\boldsymbol{\mu}_1, \dots, \boldsymbol{\mu}_k)] \leq \sum_{i=1}^k \sigma_i^2$$

Risolvere il problema in \mathbb{R}^d è \mathcal{NP} -hard anche per $k = 2$ (quando $d = 2n$). Di conseguenza, il migliore algoritmo esatto per risolvere k -means si basa su

1. Enumerare tutte le $k^{|\mathcal{X}|}$ partizioni di \mathcal{X} in k componenti
2. Calcolare i centroidi $\mathcal{C} = \{\mathbf{c}_1, \dots, \mathbf{c}_k\}$ per i k elementi della partizione
3. Calcolare il costo $\Phi(\mathcal{C})$ della partizione

Indichiamo con $\mathcal{X}(i)$ il sottoinsieme di punti di \mathcal{X} tali che hanno \mathbf{c}_i come centro più vicino

$$\mathcal{X}(i) = \left\{ \mathbf{x} \in \mathcal{X} \mid \arg \min_{j=1,\dots,k} \|\mathbf{x} - \mathbf{c}_j\|^2 = i \right\}$$

L'algoritmo seguente è l'euristica più comune per risolvere k -means.

Algoritmo 8: Algoritmo di Lloyd

Input: Insieme finito $X \subset \mathbb{R}^d$, parametro k t.c. $1 < k < |\mathcal{X}|$

```

1 Scegliere uniformemente a caso  $k$  punti da  $\mathcal{X}$ 
2 repeat
3   for  $\mathbf{x} \in \mathcal{X}$  do
4     Assegna  $\mathbf{x}$  al cluster  $C_i$ , con  $i = \arg \min_{j=1,\dots,k} \|\mathbf{x} - \mathbf{c}_j\|^2$ 
5   for  $i = 1, \dots, k$  do
6      $\mathbf{c}_i \leftarrow \frac{1}{|C_i|} \sum_{\mathbf{x} \in \mathcal{X}(i)} \mathbf{x}$            //  $\mathbf{c}_i$  è il centroide di  $C_i$ 
7 until  $\mathbf{c}_1, \dots, \mathbf{c}_k$  rimangono invariati

```

Il tempo per ogni iterazione dell'algoritmo è nell'ordine di $\mathcal{O}(nkd)$. Si possono usare proiezioni casuali per mappare \mathcal{X} a \mathbb{R}^N con $N = \Theta(\ln n)$, risultando in un valore di OPTmoltiplicato per una costante. Questo riduce il tempo per iterazione a $\mathcal{O}(nk \ln n)$. Il caso peggiore per quanto riguarda il numero di iterazioni dell'algoritmo, purtroppo, è $2^{\Omega(\sqrt{n})}$.

Anche se funziona bene in casi concreti, l'algoritmo di Lloyd non approssima OPTper nessuna costante.

Teorema 4.2.1. *Per ogni $a > 1$ esistono istanze 1-dimensionali $\mathcal{X} \subset \mathbb{R}$ di k -means con $k = 3$ per le quali l'algoritmo di Lloyd restituisce una partizione \mathcal{C} tale che $\Phi(\mathcal{C}) \geq a \cdot \text{OPT}$ con probabilità arbitrariamente vicina a 1.*

Dimostrazione. Scelto un $a > 1$ e sia \mathcal{X} di dimensione n tale che $n - 2$ punti sono spaziati equamente all'interno del segmento $[0, 1]$, mentre i due punti rimanenti (outliers) sono posizionati a $2\sqrt{an}$ e $3\sqrt{an}$



La probabilità che l'algoritmo di Lloyd non scelga entrambi gli outlier come centri iniziali è calcolata come segue: ci sono $\binom{n}{3}$ modi possibili per scegliere 3 punti su n e $n - 2$ modi di scegliere 3 punti tali che tra questi ci siano i due outlier. Quindi la probabilità è

$$p_n = 1 - \frac{n-2}{\binom{n}{3}} = 1 - \frac{(n-3)!6(n-2)}{n!} = 1 - \frac{6}{n(n-1)}$$

Consideriamo quindi il caso sfavorevole in cui l'algoritmo sceglie inizialmente al più uno degli outlier. In questo caso, l'algoritmo termina con almeno due centri all'interno di $[0, 1]$ è al più un centro in $\frac{5}{2}\sqrt{an}$. Il costo $\Phi(\mathcal{C})$ di questa partizione \mathcal{C} è almeno $\frac{an}{2}$, mentre il costo del cluster ottimo (due centri sugli outlier e il rimanente su $1/2$) è $\text{OPT} = \frac{n-2}{4}$.

Quindi, $\Phi(\mathcal{C})/\text{OPT} = \Omega(a)$. Al crescere di n , $n \rightarrow \infty$, abbiamo che $p_n \rightarrow 1$, con la conseguenza che il caso sfavorevole accade con probabilità arbitrariamente alta. \square

Ora mostriamo come, se i centri si muovono, il valore di Φ decresce strettamente e può farlo al più $\mathcal{O}(k^n)$ volte, ovvero il numero di possibili partizioni in k componenti di \mathcal{X} con $|\mathcal{X}| = n$.

Lemma 4.2.1. *Se in una iterazione un qualsiasi centro viene mosso, allora Φ decresce strettamente.*

Dimostrazione. Faremo uso del fatto seguente. Per ogni $C \subset \mathbb{R}^d$ finito e per ogni $\mathbf{c} \in \mathbb{R}^d$

$$\sum_{\mathbf{x} \in C} \|\mathbf{x} - \mathbf{c}\|^2 = |C| \|\mathbf{c} - \boldsymbol{\mu}\|^2 + \sum_{\mathbf{x} \in C} \|\mathbf{x} - \boldsymbol{\mu}\|^2 \quad (\dagger)$$

dove $\boldsymbol{\mu}$ è il centroide di C .

Siano C_1, \dots, C_k e $\mathbf{c}_1, \dots, \mathbf{c}_k$ rispettivamente cluster e centri all'inizio di una iterazione (Linea 2) e siano C'_1, \dots, C'_k e $\mathbf{c}'_1, \dots, \mathbf{c}'_k$ cluster e centri al termine

dell'iterazione (Linea 7). Sia

$$\psi(C_1, \dots, C_k, \mathbf{c}_1, \dots, \mathbf{c}_k) = \sum_{i=1}^k \sum_{\mathbf{x} \in C_i} \|\mathbf{x} - \mathbf{c}_i\|^2$$

Si noti che $\psi(C_1, \dots, C_k, \mathbf{c}_1, \dots, \mathbf{c}_k) \geq \psi(C'_1, \dots, C'_k, \mathbf{c}_1, \dots, \mathbf{c}_k)$ dato che Linea 4 assegna ogni punto al centro più vicino.

Ora, se $\mathbf{c}'_i \neq \mathbf{c}_i$ per qualche i , allora

$$\psi(C'_1, \dots, C'_k, \mathbf{c}_1, \dots, \mathbf{c}_k) > \psi(C'_1, \dots, C'_k, \mathbf{c}'_1, \dots, \mathbf{c}'_k)$$

Per dimostrarlo, ricordando che \mathbf{c}'_i è il centroide di C'_i ,

$$\sum_{\mathbf{x} \in C'_i} \|\mathbf{x} - \mathbf{c}_i\|^2 \stackrel{(\dagger)}{=} |C'| \|\mathbf{c}_i - \mathbf{c}'_i\|^2 + \sum_{\mathbf{x} \in C'_i} \|\mathbf{x} - \mathbf{c}'_i\|^2 > \sum_{\mathbf{x} \in C'_i} \|\mathbf{x} - \mathbf{c}'_i\|^2$$

usando (\dagger) nel primo passaggio e $\mathbf{c}_i \neq \mathbf{c}'_i$ nel secondo. Di conseguenza

$$\begin{aligned} \Phi(C_1, \dots, C_k) &= \psi(C_1, \dots, C_k, \mathbf{c}_1, \dots, \mathbf{c}_k) \\ &> \psi(C'_1, \dots, C'_k, \mathbf{c}'_1, \dots, \mathbf{c}'_k) = \Phi(C'_1, \dots, C'_k) \end{aligned}$$

□

Questo implica anche il seguente risultato.

Teorema 4.2.2. *L'algoritmo di Lloyd termina in al più $k^{|X|}$ iterazioni per qualsiasi input (\mathcal{X}, k) .*

Dimostrazione. Si può notare che Φ è una funzione applicata alla partizione corrente $\{C_1, \dots, C_k\}$, tale partizione può assumere al più k^n valori distinti. Inoltre, l'algoritmo non termina se l'iterazione corrente ha modificato la partizione. Dato che Φ può solo decrescere quando la partizione viene modificata, l'algoritmo deve terminare in al più k^n iterazioni. □