Università degli studi di Parma

Facoltà di Scienze Naturali, Fisiche e Matematiche

Corso di Laurea in Fisica Teorica

Sviluppi teorici e applicativi delle metriche entropiche di Rohlin

Relatore:
prof. Mario Casartelli

Correlatore:
prof.ssa Raffaella Burioni

Tesi presentata da:
Dawid Crivelli Wieslaw

Anno Accademico 2010-2011

Indice

Capitolo 1

Partizioni su insiemi

Definiamo in questo capitolo alcuni concetti fondamentali: le partizioni su un insieme, il significato di entropia, lo spazio delle partizioni con le operazioni binarie associate e una distanza tra partizioni. Studiamo le proprietà di questa distanza e come amplificarla. Diamo infine qualche esempio sui possibili tipi di partizioni che si possono incontrare su un insieme discreto.

1.1 Generalità

Introduciamo il formalismo e risultati generali per spazi di partizioni e metriche di Rohlin, seguendo l'approccio in [**?** **?**]. Sia $(\mathbf{M}, \mathscr{M}, \mu)$ uno spazio di probabilità, ovvero un insieme $\mathbf{M}$, una σ-algebra $\mathscr{M}$ di sottoinsiemi di $\mathbf{M}$, una misura normalizzata μ su $\mathscr{M}$. Nei casi trattati $\mathbf{M}$ può essere una sequenza di simboli, un reticolo bidimensionale, un grafo arbitrario.

Introducendo una relazione di equivalenza su $\mathbf{M}$, possiamo definire una particolare classe di sottoinsiemi. Una *partizione* di $\mathbf{M}$ è una collezione finita $\alpha \equiv (A_1, A_2, \ldots, A_N)$ di sottoinsiemi disgiunti misurabili che ricoprono $\mathbf{M}$, cioè $A_i \cap A_j = \emptyset$ se $i \neq j$ e $\bigcup_k A_k = \mathbf{M}$. Gli $\{A_k\}$ sono chiamati *atomi* di α e sono una rappresentano le classi di equivalenza degli elementi di $\mathbf{M}$. L'insieme di tutte le partizioni misurabili è denotato con $\mathscr{Z} \equiv \mathscr{Z}(\mathbf{M})$. La partizione unitaria ν consiste del singolo atomo coincidente con $\mathbf{M}$. È possibile introdurre un ordine parziale su $\mathscr{Z}$, con la relazione $\alpha \leq \beta$ quando β è un raffinamento di α: questo accade quando ogni atomo A_k è esattamente composto da atomi di β, cioè $A_k = \left\{ \bigcup_j B_j \mid B_j \in \beta \right\}$. In questo caso, si dice che α è un *fattore* di di β. La partizione banale $\nu \leq \alpha$, $\forall \alpha$.

I termini come *unità* e *fattore* dipendono dalla definizione di uno pseudo-prodotto commutativo ed associativo, la composizione $\gamma = \alpha \vee \beta$ (o anche $\gamma = \alpha\beta$ ove non vi sia ambiguità). Il *prodotto* è la partizione meno fine di tutte le partizioni con $\gamma \geq \alpha$ e $\gamma \geq \beta$, i cui atomi sono le intersezioni non vuote di tutti gli atomi di α e β. Chiaramente il prodotto con l'unità si comporta come l'identità del prodotto, con $\alpha\nu = \alpha$ per ogni α, mentre $\alpha\eta = \alpha$ quando $\eta \leq \alpha$. Queste proprietà rendono il risultato della composizione una specie di "minimo comune multiplo". Dalla definizione segue anche che $\alpha \vee \alpha = \alpha$, ovvero il prodotto è idempotente.

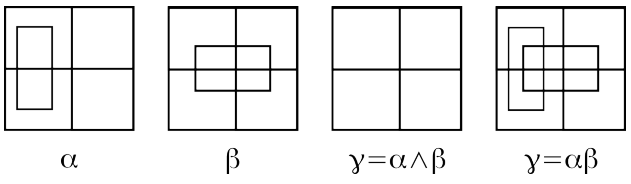

Figura 1.1.1: Esempio di prodotto e intersezione tra due partizioni

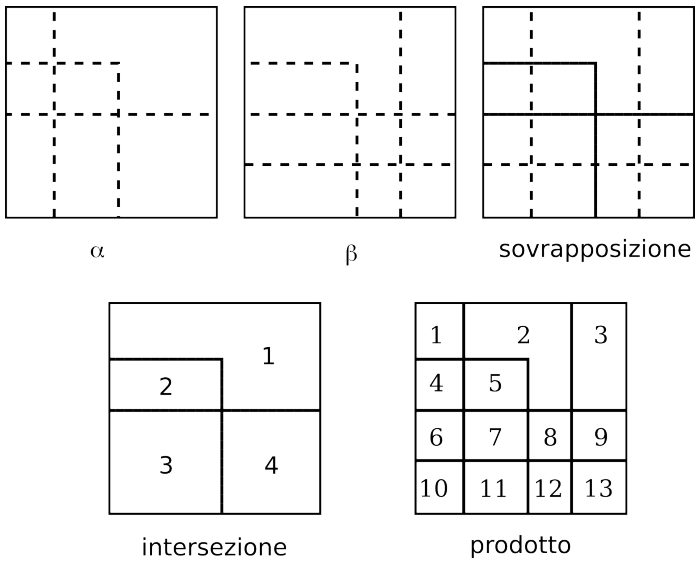

Figura 1.1.2: Intersezione e prodotto come operazioni sui bordi. Le partizioni α e β hanno bordi tratteggiati, con fase complementare, che danno l'impressione di linea continua quando sovrapposti. È evidente in questo modo il tratto di bordi in comune. Gli atomi sono stati numerati per rendere evidente la differenza tra le due operazioni.

L'operazione complementare che implementa un "massimo comune divisore" nello spazio delle partizioni è l'*intersezione* $\sigma = \alpha \wedge \beta$, definita come la partizione più fine tale che $\sigma \leq \alpha$ e $\sigma \leq \beta$. In questo caso, $\alpha \wedge \nu = \nu$ per ogni α, mentre $\alpha \wedge \beta = \nu$ implica che α e β sono *relativamente primi*, cioè non hanno un fattore comune.

Un altro modo di calcolare e visualizzare le operazioni è in termini di *bordi interni* di atomi della partizione, come si vede in figura 1.1.2. Il prodotto $\alpha \vee \beta$ corrisponde alla partizione avente come bordi l'unione dei bordi di α e β, mentre l'intersezione $\alpha \wedge \beta$ ha come bordi l'intersezione di quelli di α e β. Quando nell'intersezione i bordi non si chiudono, sono cancellati dal risultato e gli atomi raggruppati. Poichè la partizione banale non ha bordi tra atomi, si ricavano immediatamente le sue proprietà nel prodotto e nell'intersezione.

Una partizione può rappresentare un esperimento probabilistico con risultati disgiunti $A_1, \ldots, A_N$, dove l'evento *atomico* A_k ha probabilità $\mu(A_k)$. Un *fattore* è quindi un sottoesperimento dell'esperimento più fine, che raggruppa diversi risultati come equivalenti: ad esempio, "pari o dispari" è un sottoesperimento con due atomi, dell'esperimento $\{1,2,3,4,5,6\}$ del lancio di un dado.

Sullo spazio $\mathscr{Z}$ possiamo definire dei funzionali *entropia* $H : \mathscr{Z} \to \mathbb{R}^+$, definiti su ogni partizione. In particolare l'*entropia di Shannon*

$$H_S(\alpha) = -\sum_{i=1}^{n} \mu(A_i) \ln \mu(A_i) \tag{1.1.1}$$

L'entropia di Shannon è una misura dell'informazione media ottenuta dall'esperimento. Si vede immediatamente che la partizione banale ν, non codificando alcuna informazione ha entropia nulla. Se $\beta = (B_1, \ldots, B_n)$ è un'altra partizione, l'entropia condizionata di α rispetto a β è

$$H_S(\alpha|\beta) = -\sum_{i=1}^{n}\sum_{k=1}^{m} \mu(A_i \cap B_k) \ln \frac{\mu(A_i \cap B_k)}{\mu(B_k)} = H_S(\alpha \vee \beta) - H_S(\beta) \tag{1.1.2}$$

dove si prende per convenzione che $x \ln x = 0$ se $x = 0$. L'entropia condizionata è l'informazione media residua ottenuta da α quando il risultato per β è noto. Si noti che l'entro-

pia di Shannon dipende solo dalla distribuzione delle misure degli atomi, non dalla loro natura o "forma", che potrebbe non avere significato in spazi astratti. Le mutue relazioni tra atomi (e possibilmente le loro forme) al contrario influenzano direttamente l'entropia condizionale.

Definiamo una metrica sullo spazio delle partizioni $\mathscr{Z}(\mathbf{M})$ tramite la distanza di Rohlin d_{R}

$$d_{\mathrm{R}} = H(\alpha|\beta) + H(\beta|\alpha)$$

che misura la complessiva non-similarità tra le partizioni α e β. È possibile dare una definizione alternativa di questa distanza, sfruttando la seconda scrittura della probabilità condizionale, riscrivendo d_{R} come

$$d_{\mathrm{R}} = 2H(\alpha \vee \beta) - H(\alpha) - H(\beta) \tag{1.1.3}$$

La simmetria $d_{\mathrm{R}}(\alpha,\beta) = d_{\mathrm{R}}(\beta,\alpha)$, la positività e la condizione $d_{\mathrm{R}}(\alpha,\alpha) = 0$ sono manifeste, mentre la disuguaglianza triangolare è soddisfatta se H soddisfa alle condizioni di un funzionale entropia [**?**]:

$$\begin{aligned} H(\alpha \vee \beta \mid \sigma) &= H(\alpha \mid \beta) + H(\beta \mid \alpha \vee \sigma) \\ H(\alpha \vee \beta \mid \sigma) &\leq H(\alpha \mid \sigma) + H(\beta \mid \sigma) \\ H(\alpha \mid \sigma) &\leq H(\beta \mid \sigma) \quad \text{se} \quad \alpha \subset \beta \\ H(\alpha \mid \sigma) &\leq H(\alpha \mid \beta) \quad \text{se} \quad \sigma \supset \beta \end{aligned}$$

Se $\mathbf{M}$ è finito, una *configurazione* o *stato* $\mathbf{a}$ su $\mathbf{M}$ è una funzione che assegna ad ogni punto $x_i \in \mathbf{M}$ un valore $a_i = f(x_i)$ nell'alfabeto $\mathbb{K}$. Tutte le possibili configurazioni formano uno spazio $\mathscr{C} \equiv \mathscr{C}(\mathbf{M})$. Su $\mathscr{C}$ la distanza di Hamming è definita come

$$d_{\mathrm{H}}(\mathbf{a},\mathbf{b}) = \mathscr{N} \sum_i \rho(a_i, b_i)$$

dove $\rho(a_i, b_i)$ è una distanza su $\mathbb{K}$ e $\mathscr{N}$ una possibile costante di rinormalizzazione, che noi porremo uguale a 1. Da notare come questa distanza operi tra elementi diversi dalla distanza di Rohlin, pur partendo sempre da $\mathbf{M}$.

1.1.1 Distanza topologica

Oltre all'entropia di Shannon, definita tramite la misura μ, è possibile una definizione alternativa che non vi fa ricorso. Per questo motivo, è chiamata *entropia topologica*. Dato uno spazio compatto, come lo spazio$\mathbf{M}$[1] da noi considerato, il teorema di Heine-Borel[**?**] assicura che da ogni ricoprimento tramite insiemi aperti è possibile estrarre una sottocopertura finita. In particolare, $\exists \delta \in \mathbb{N}^+$ numero minimo di aperti con cui generare tale ricoprimento. La partizione di uno spazio è ricoprimento *minimale*[2], motivo per cui il numero di atomi della partizione satura la disuguaglianza del teorema, $\delta = n$. Una volta stabilito l'esistenza di δ e avendone calcolato il risultato, poniamo l'entropia topologica pari al logaritmo naturale di δ

$$H_T(\alpha) = \ln(\delta) = \ln(n) \tag{1.1.4}$$

Nei casi in cui non vi sia una naturale misura sullo spazio $\mathbf{M}$, la definizione topologica può essere utile per cercare una distanza che meglio renda la differenza intrinseca tra partizioni. Tuttavia, non esiste una definizione di entropia condizionale per H_T e per definire la distanza di Rohlin utilizziamo la sua definizione in termini di prodotto tra partizioni.

[1] Addirittura $\mathbf{M}$ nel nostro caso è discreto, per cui la compattezza è equivalente all'essere finito.

[2] Nulla vieta di avere altri ricoprimenti perfettamente ridondanti, ma sempre con un numero finito di aperti

Si vede come H_T è un buon funzionale entropia, infatti se $\alpha < \beta$, allora $H_T(\alpha) < H_T(\beta)$, in quanto una partizione strettamente più fine ha banalmente un numero maggiore di atomi. Da questa condizione si ha che per il prodotto

$$\alpha \vee \beta \geq \alpha \implies H_T(\alpha \vee \beta) \geq H_T(\alpha)$$

per α, β generici e la distanza di Rohlin topologica, definita usando il funzionale H_T nell'equazione 1.1.3 risulta definita positiva. Notiamo che la distanza è ben definita solo quando si considera partizioni poichè la distanza tra ricoprimenti non minimali può non essere 0 anche tra ricoprimenti diversi – ha senso solo considerare la restrizione alla classe di equivalenza del ricoprimento minimale corrispondente.

1.2 Riduzione

L'essenziale dissimilirità tra due partizioni potrebbe essere confusa ed indebolita dalla presenza di un fattore comune ampio, come ad esempio accade se gli atomi della partizione hanno lunghezza media molto breve, nel qual caso la maggioranza dei confini risulta essere la stessa. Si cerca quindi di eliminare fattori comuni il più possibile, con una *riduzione* che ci si aspetta aumenti la distanza relativa. Tuttavia questa operazione, analoga alla riduzione in minimi termini per frazioni, non è unicamente definita, in quanto le partizioni, a differenza degli interi, non ammettono una univoca fattorizzazione in fattori primi. Il ruolo dei fattori primi (ovvero fattori irriducibili) è giocato dalle sottopartizioni *dicotomiche,* che sono tuttavia ancora estremamente ridondanti ($2^{n-1}-1$ per partizioni con n atomi).

A partire dalla partizione $\alpha \equiv (A_1, \ldots, A_n)$, definiamo quindi una famiglia ristretta $\mathbf{E}(\alpha)$ di *fattori dicotomici elementari* $\tilde{\alpha}_1, \tilde{\alpha}_2, \ldots, \tilde{\alpha}_n$ con le seguenti caratteristiche:

1. $\mathbf{E}(\alpha)$ deve essere ben definita per ogni $\alpha \in \mathscr{Z}$
2. $\mathbf{E}(\alpha)$ non deve contenere più di n (il numero di atomi in α) fattori elementari
3. $\vee_{k=1}^{n} \tilde{a}_k = \alpha$

Una scelta universale coniste nel prendere come fattori dicotomici $\tilde{\alpha}_k \equiv (A_k, A_k^c)$ le partizioni formate dai singoli atomi e dai loro complementi in $\mathbf{M}$. Fattori di questo tipo sono chiamati *universali semplici.*

Una volta che per due partizioni α e β le famiglie di fattori dicotomici $\mathbf{E}(\alpha)$ e $\mathbf{E}(\beta)$ sono state definite, abbiamo diversi possibili processi di riduzione.

Definizione 1. Riduzione tramite partizione comune

1. Si definisce il massimo fattore comune $\sigma = \alpha \wedge \beta$
2. Si tralasciano da $\mathbf{E}(\alpha)$ e $\mathbf{E}(\beta)$ i fattori che **non** sono relativamenti primi con σ, e indichiamo i fattori rimanenti come $\hat{\alpha}_k$ e $\hat{\beta}_k$ rispettivamente. Questo vuol dire che $\hat{\alpha}_k \wedge \sigma = \hat{\beta}_j \wedge \sigma = \nu$.

Definizione 2. Riduzione con eliminazione fattori in comune

1. Si tralasciano da $\mathbf{E}(\alpha)$ e $\mathbf{E}(\beta)$ i fattori che compaiono in entrambe le partizioni. Se indichiamo i fattori rimanenti come $\hat{\alpha}_k$ e $\hat{\beta}_k$ rispettivamente, questo vuol dire che $\forall A_k, \nexists B_j \in \beta | A_k = B_j$ e viceversa.

Definizione 3. Riduzione con eliminazione fattori simili

1. Si tralasciano da $\mathbf{E}(\alpha)$ e $\mathbf{E}(\beta)$ i fattori che hanno un corrispondente "simile" nell'altra partizione. Questo vuol dire che scartiamo da $\hat{\alpha}$ il fattore

$$A_k \text{ se } \exists\ B_j \in \beta, \text{ tale che } \mu(A_k \triangle B_j) \leq \epsilon$$

e viceversa da $\hat{\beta}$ il fattore

$$\beta_k \text{ se } \exists\, A_j \in \alpha, \text{ tale che } \mu(B_k \triangle A_j) \leq \epsilon$$

Il simbolo $A_k \triangle B_j$ indica la differenza simmetrica tra i due atomi, ovvero i siti che appartengono ad un atomo ma non all'altro.

Alla fine, per tutti i tipi di riduzione, definiamo le partizioni ridotte come $\hat{\alpha} = \vee_k \hat{\alpha}_k$ e $\hat{\beta} = \vee_k \hat{\beta}_k$, ovvero il prodotto dei fattori dicotomici "sopravvissuti". Nel capitolo sugli algoritmi presenteremo metodi ottimali per il calcolo dei fattori dicotomici per ogni criterio presentato, che presentano una notevole complessità se eseguiti nel modo naïve.

Per il resto della sezione concentreremo la nostra attenzione sulla riduzione tramite fattore comune massimo. Si motiva la scelta del confronto con il fattore comune poichè vi sono casi in cui le partizioni non hanno atomi in comune, ma ciononostante si ha che $\sigma \neq \nu$. Questo accade, per esempio, quando $\alpha < \beta$ strettamente e non vi sono fattori comuni elementari. In questo caso allora $\sigma = \alpha$ e $\hat{\alpha} = \nu$ con questo metodo di riduzione, mentre $\hat{\alpha} = \alpha$ tralasciando i fattori comuni. Può capitare inoltre che anche se le partizioni sono già ridotte, non sono prime tra di loro. In particolare, $H(\sigma)$ indica la similarità tra le partizioni.

Osservazione La mappa simmetrica definita dalla distanza tra sequenze ridotte

$$\begin{aligned} \mathscr{Z} \times \mathscr{Z} &\to \mathbb{R} \\ \alpha \otimes \beta &\mapsto d_R(\hat{\alpha}, \hat{\beta}) \end{aligned}$$

non definisce una distanza su $\mathscr{Z}$. Infatti il passaggio $\alpha \to \hat{\alpha}$ dipende strettamente da β e viceversa, non è quindi possibile scrivere

$$d_R(\hat{\alpha}, \hat{\beta}) = \hat{d}_R(\alpha, \beta)$$

con $\hat{d}_R$ una distanza.

1.2.1 Amplificazione

Il processo di riduzione porta alla definizione di partizioni con complessità possibilmente inferiore, ovvero $H(\hat{\alpha}) \leq H(\alpha)$. Questo va nel verso opposto quando si considera l'effetto sulla distanza, che invece aumenta.

Il rapporto di *amplificazione* R misura quanto la riduzione ha messo in risalto la differenza tra partizioni. Ne dimostriamo la caratteristica fondamentale:

$$\mathrm{R} = \frac{\mathrm{d_R}(\hat{\alpha}, \hat{\beta})}{\mathrm{d_R}(\alpha, \beta)} \geq 1$$

Proposizione. $d_{\mathrm{R}}(\hat{\alpha}, \hat{\beta}) \geq d_{\mathrm{R}}(\alpha, \beta)$

Dimostrazione. Ricordando che $\sigma = \alpha \wedge \beta$ possiamo scrivere $\alpha = \sigma\hat{\alpha}$ e $\beta = \sigma\hat{\beta}$: infatti σ contiene tutti i fattori tralasciati durante la riduzione. Utilizzando ora l'idempotenza del prodotto, $\sigma = \sigma\sigma$, possiamo riscrivere la tesi utilizzando l'equazione (1.1.3)

$$2\,H(\sigma\hat{\alpha}\hat{\beta}) - H(\sigma\hat{\alpha}) - H(\sigma\hat{\beta}) \leq 2\,H(\hat{\alpha}\hat{\beta}) - H(\hat{\alpha}) - H(\hat{\beta})$$

scambiando l'ordine dei termini si ottiene

$$2\,H(\sigma\hat{\alpha}\hat{\beta}) - 2\,H(\hat{\alpha}\hat{\beta}) \leq H(\sigma\hat{\alpha}) - H(\hat{\alpha}) + H(\sigma\hat{\beta}) - H(\hat{\beta})$$

e sfruttando la formula (1.1.2) per l'entropia condizionata, la tesi si riduce a

$$2\,H(\sigma|\hat{\alpha}\hat{\beta}) \leq H(\sigma|\hat{\alpha}) + H(\sigma|\hat{\beta})$$

ma questo è chiaramente vero, in quanto

$$H(\sigma|\hat{\alpha}\hat{\beta}) \leq H(\sigma|\hat{\alpha}) \quad \text{e} \quad H(\sigma|\hat{\alpha}\hat{\beta}) \leq H(\sigma|\hat{\beta})$$

per le proprietà dell'entropia, poichè il termine condizionante è sicuramente maggiore, ovvero

$$\hat{\alpha}\hat{\beta} \geq \hat{\alpha} \quad \text{e} \quad \hat{\alpha}\hat{\beta} \geq \hat{\beta}$$

□

Da notare che la dimostrazione vale per la distanza di Rohlin definita sia tramite H_S che H_T.

Risulta importante la scelta della famiglia di fattori dicotomici $\mathbf{E}(\alpha)$, che è dettata dalla topologia e geometria dello spazio delle configurazioni. La scelta della famiglia di fattori dicotomici universali semplici è sempre possibile, poichè la determinazione di A_k^c a partire da A_k è un'operazione ben definita in qualunque spazio topologico. Prendere come fattori elementari la parte interna di contorni di cluster ad esempio, richiede un concetto di orientabilità e la possibilità di definire contorni, ovvero insiemi con codimensione 1 su varietà – mentre vorremmo estendere l'analisi anche a grafi generici, privi di strutture geometriche predefinite. Già nel caso lineare è possibile prendere fattori dicotomici diversi e algoritmicamente più performanti, a patto di restringere lo studio alle partizioni con atomi formati da cluster connessi.

1.3 Possibili tipi di partizionamento

L'applicabilità dei metodi discussi è assolutamente generica, estendibile a qualunque spazio di probabilità finito si voglia considerare, vediamo dunque di dare esempi dei possibili spazi **M** su cui abbiamo lavorato, con i relativi fattori dicotomici e consequenze computazionali.

Essendo lo studio svolto su calcolatore, lo spazio **M** e la sua $\sigma-$algebra sono finiti e discreti. I siti appartenenti ad **M** possono essere sempre numerati ordinati in modo opportuno x_i, $i \in \{1,\ldots,L\}$, dove L è il numero totale di siti, che si tratti di un reticolo o di un grafo. Poichè la partizione induce una relazione di equivalenza sull'insieme **M**, indichiamo che due siti sono equivivalenti se appartengono allo stesso atomo

$$i \sim j \iff \exists A_k \text{ tale che } i \in A_k,\ j \in A_k$$

Per partizionare i siti in atomi disguinti, richiediamo che la *configurazione* (o *stato*) $\mathscr{C}$, associ ad ogni sito una lettera dell'alfabeto $\mathbb{K}$, considerato finito,$|\mathbb{K}| < \infty$. Il simbolo associato al sito i-esimo, non essendovi ambiguità, sarà d'ora in poi indicato con $f(i)$. Se lo stato del sistema è descritto con variabili continue (o vi è un numero enorme di possibili lettere nell'alfabeto, si pensi ad una variabile a 64 bit rappresentante un numero reale), si può sempre ridurre l'alfabeto ad un insieme $\{k_j\}$ discretizzando i valori della configurazione con il criterio:

$$f(i) := \bar{k} \quad \text{tale che} \quad |f(i-\bar{k})| = \min_{k_j \in \mathbb{K}} |f(i) - k_j|$$

oppure mettendo nello stesso atomo siti vicini che hanno valori distanti meno di ϵ:

$$i \sim j \iff |f(i) - f(j)| \leq \epsilon$$

1.3.1 Sequenze lineari connesse

Consideriamo sequenze lunghe L, provenienti da due casi:

- Problemi di meccanica statistica, in cui la configurazione è una variabile aleatoria, generata algoritmicamente da una catena di Markov in base al modello di Ising monodimensionale, nel qual caso l'alfabeto corrisponde a $\{-1,+1\}$.

- Sequenze di origine biologica, in particolare sequenze di amminoacidi (proteine), in cui $|\mathbb{K}| = 25$ (dettagli a pagina 32)

La distanza di Hamming d_H è sensibile solo a variazioni puntuali dei valori in $\mathscr{C}$. I siti che compongono la sequenze non si influenzano, una variazione su un sito può solo variare di $\{0,+1\}$ la distanza totale.

Ad ogni configurazione possiamo associare una partizione in $\mathscr{Z}$, in cui gli atomi sono formati dai cluster, cioè sottoinsiemi connessi di **M** a valori omogenei in $\mathbb{K}$. Questo stabilisce una mappa $\Phi : \mathscr{C} \to \mathscr{Z}$ da ogni configurazione ad una partizione corrispondente, rendendo possibile il confronto tra $d_{\mathrm{H}}(\mathbf{a},\mathbf{b})$ in $\mathscr{C}$ e $d_{\mathrm{R}}(\alpha,\beta)$ in $\mathscr{Z}$, dove $\alpha = \Phi(\mathbf{a})$ e $\beta = \Phi(\mathbf{b})$. La relazione è chiaramente del tipo molti-a-uno, infatti assegnando ad un segmento di lettere omogeneo in $\mathscr{C}$ un diverso simbolo, non cambia la partizione corrispondente. In simboli, esprimiamo la relazione come

$$i \sim j \iff j = i+1,\ f(i) = f(j)$$

È evidente quindi che variazioni locali, ad esempio il cambiamento di un singolo simbolo, possono non modificare affatto la partizione

$$\{\ldots, T, T, C, A, A, \ldots\} \overset{\Phi}{\equiv} \{\ldots, T, T, M, A, A, \ldots\}$$

che presenta sì una perdita di informazione, ma permette quindi anche di filtrare molto "rumore" e si è dimostrata una ottima scelta nel caso biologico e una necessità nello studio di sequenze di Ising – la hamiltoniana del sistema non distingue i valori associati ai singoli siti, ma solo differenze tra siti vicini. La suddivisione in partizioni si comporta nello stesso modo, estraendo quindi solo informazioni fisicamente rilevanti.

Esempio. Supponiamo di aver partizionato una stringa come {A,A,A,B,B,C,C,D,D,D} o $\{+,+,+,-,-,+,+,-,-,-\}$, aventi la stessa partizione

$$\alpha = \{(1,2,3),(4,5),(6,7),(8,9,10)\}$$

il calcolo esplicito dell'entropia è il seguente:

$$\begin{aligned} A_1 &= (1,2,3) & \mu(A_1) &= \frac{3}{10} \\ A_2 &= (4,5) & \mu(A_2) &= \frac{2}{10} \\ A_3 &= (6,7) & \mu(A_3) &= \frac{2}{10} \\ A_4 &= (8,9,10) & \mu(A_4) &= \frac{3}{10} \end{aligned}$$

Ora, $H_{\mathrm{S}}(\alpha) = -2\,(0.3)\ln(0.3) - 2\,(0.2)\ln(0.2) \simeq 1.36$, mentre $H_{\mathrm{T}}(\alpha) = \ln(4) \simeq 1.38$. Come si vede i valori sono abbastanza simili.

Fattori dicotomici nel caso lineare connesso

Per una partizione $\alpha \equiv (A_1, \ldots, A_n)$, definiamo i fattori dicotomici $\tilde{\alpha}_1, \tilde{\alpha}_2, \ldots, \tilde{\alpha}_n \in \mathbf{E}(\alpha)$ nel modo seguente:

$$\tilde{\alpha}_k = \left\{ \bigcup_{j=1}^{k} A_j,\ \bigcup_{j=k+1}^{n} A_j \right\}$$

dove ricordiamo che n è il numero di atomi. In termini di siti quindi, il fattore dicotomico k-esimo corrisponde alla partizione con tutti i siti corrispondenti ai primi k atomi presi insieme, ed il complementare corrispondente ai siti a partire dall'atomo $(k+1)$-esimo fino ad L, come mostrato in figura 1.3.1.

Con questa particolare scelta, resa possibile dalla topologia connessa e ordinata, poichè abbiamo l'ordine ereditato da $\mathbb{N}$, il processo di riduzione è estremamente semplificato ed efficiente. Partendo da una partizione α, la partizione ridotta $\hat{\alpha}$ è quella in cui sono stati rimossi i bordi in comune tra α e σ, come si vede in figura 1.3.2

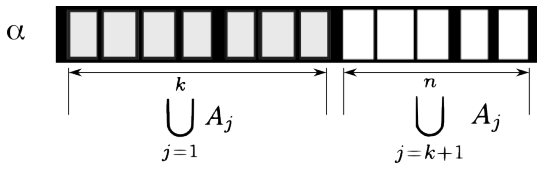

Figura 1.3.1: Fattori dicotomici lineari

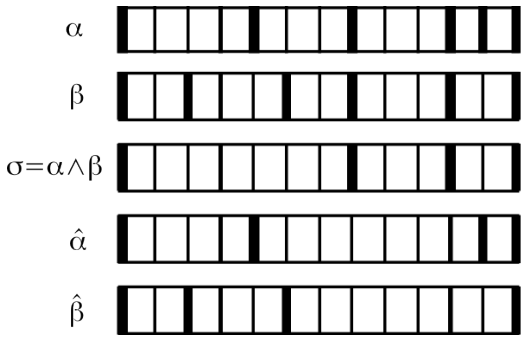

Figura 1.3.2: Riduzione nel caso di sequenze lineari. La riga spessa indica la separazione tra atomi. Si nota come i bordi della partizione comune siano quelli comuni sia ad α che a β, mentre nelle partizioni ridotte non compaiono più i bordi esistenti anche in σ

1.3.2 Sequenze non connesse

Lo studio delle sequenze biologiche, in cui vi è una notevole frammentazione, dovuta alla bassa probabilità di avere molti simboli consecutivi uguali, ha motivato la ricerca di diversi criteri di partizionamento. Il più semplice è considerare come appartenenti allo stesso atomo siti con lo stesso simbolo, non adiacenti ma con la possibilità di saltare al massimo n_s siti in avanti

$$i \sim j \iff |j-i| \leq n_s,\ f(i)=f(j)$$

In questo caso, la topologia nello spazio $\mathscr{X}$ non è più connessa, per cui l'utilizzo dei fattori dicomici lineari non è possibile. Un'alternativa è quella di prendere per l'atomo $A_k = \{n_{k_1}, n_{k_2}, \ldots, n_{k_n}\}$ una partizione dicotomica costituita da tutti i siti tra il primo e l'ultimo appartenente a A_k e dal complementare, sconnesso, A_k^c

$$\hat{\alpha}_k = \left\{ \bigcup_{i=n_{k_1}}^{n_{k_n}} n_i,\ \left(\bigcup_{i<n_{k_1}} n_i \right) \bigcup \left(\bigcup_{i>n_{k_n}} n_i \right) \right\}$$

Dati gli scarsi vantaggi di questo approccio, si è scelto tuttavia anche in questo caso di utilizzare i *fattori universali semplici.* Una volta venuta a meno la connessione degli atomi, questo tipo di partizionamento presenta le complessità e caratteristiche di un criterio arbitrario su grafo generico – si veda il prossimo paragrafo per le illustrazioni delle operazioni tra partizioni di questo tipo.

1.3.3 Sequenze su reticoli

In questo caso, si considera le partizioni in cui fanno parte dello stesso atomo siti con lo stesso simbolo, ma con il vincolo che devono essere *primi vicini* sul reticolo

$$i \sim j \iff d(i,j)=1,\ f(i)=f(j)$$

dove la distanza indica il numero di passi sul reticolo. Possiamo tuttavia generalizzare arbitrariamente la condizione di vicinanza, per includere secondi vicini, siti con lo stesso simbolo arbitrariamente posti, ecc.

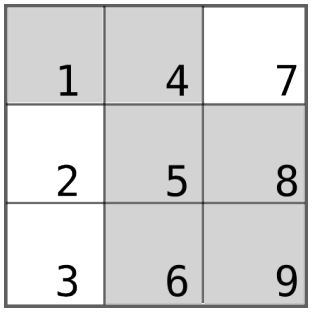

Figura 1.3.3: Mappa in memoria di un reticolo bidimensionale. In grigio e bianco sono evidenziati i valori dalla configurazione, mentre i numeri nell'angolo rappresentano l'ordine dei siti.

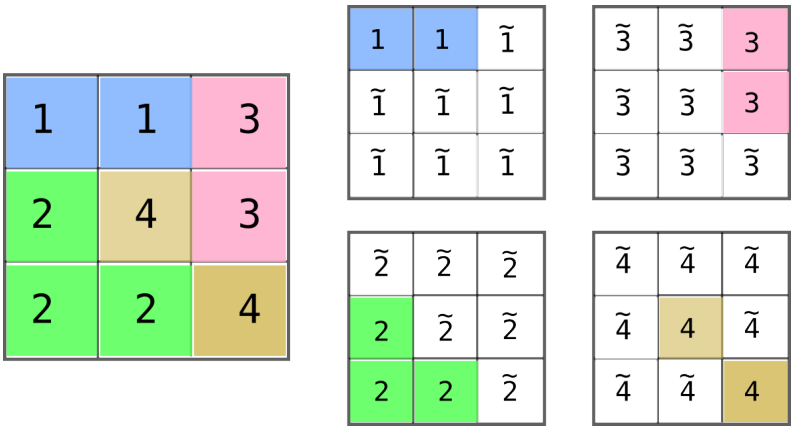

Figura 1.3.4: Partizione con 4 atomi e i 4 fattori dicotomici universali semplici corrispondenti. I numeri indicano l'etichetta corrispondente ad ogni atomo, scelto arbitrariamente. Gli atomi sono del tipo nonconnesso, in cui le etichette indicano l'appartenenza; similmente i fattori dicotomici contengono gli atomi sconnessi ed il loro complementare indicato dal label tildato.

Illustriamo le possibili operazioni con un reticolo bidimensionale, che tuttavia generalizza immediatamente al grafo arbitrario. In particolare numeriamo anche in questo caso i siti, cosa necessaria per mappare nella memoria di un computer la configurazione. La mappatura in memoria (figura 1.3.3) comporta necessariamente la disposizione di siti "vicini" sul reticolo su posizioni in cui non risultano più contigue in memoria[3]. Nel caso della figura la partizione corrispondente è

$$A = \{(1,4,5,6,8,9),(2,4),(7)\}$$

evidentemente non-connessa. In questo caso i fattori dicotomici anche risultano sconnessi, come nell'esempio di un reticolo 3x3 in figura 1.3.4, in cui si è preso come atomi i siti con lo stesso valore indipendentemente dalla loro posizione.

Nel caso in cui gli atomi sconnessi il prodotto tra partizioni rimane banale da implementare mentre l'intersezione è estremamente complicata – un'implementazione ottimale è data nel capitolo sugli algoritmi.

[3] Ricordiamo la convenzione "per colonne" di rappresentazione di una matrice bidimensionale nella maggior parte dei linguaggi di programmazione, a partire dal Fortran in poi

1.3.4 Grafi arbitrari

L'insieme dei siti in questo caso è sempre ordinabile con $i \in \{1,\dots,L\}$. Definiamo la *matrice di adiacenza* A_{ij}

$$A_{ij} = \begin{cases} 1 & \text{se i siti sono connessi} \\ 0 & \text{altrimenti} \end{cases}$$

che ci permette di dare la relazione di equivalenza come

$$i \sim j \iff A(i,j) = 1,\ f(i) = f(j)$$

In questo modo ci si slega completamente da qualunque nozione di vicinanza geometrica. In linea di principio, è possibile rimuovere anche la condizione $f(i) = f(j)$, e lasciare alla matrice di adiacenza l'implementazione della relazione di equivalenza. Non vi sono condizioni da porre su A_{ij}, qualunque scelta è accettabile e va a modificare il partizionamento, senza che questo risulti mai incoerente.

Capitolo 2

Algoritmi

Dopo aver visto i possibili tipi di partizione, mostriamo come calcolare qualunque quantità richiesta in modo ottimale. Definiamo prima le operazioni limitate a partizioni con atomi connessi, sia per la loro minore complessità, sia perché nel caso generico si sfrutta la riduzione al caso più semplice già trattato.

2.1 Partizioni connesse

Nel caso monodimensionale è sufficiente identificare il *bordo sinistro* di ogni atomo, per ricostruire completamente la partizione. Partendo da una sequenza in un array (o segmento unidimensionale finito) `S[i]` lungo L, definiamo l'array binario della partizione `B[i]`, lungo L, con indici che vanno da `0` a `L-1`

$$\texttt{B[i]} = \begin{cases} 1 & \text{il sito } \texttt{i} \text{ identifica un nuovo atomo} \\ 0 & \text{altrimenti} \end{cases}$$

algoritmicamente si procede in due step:

1. il primo sito sicuramente inizia un atomo, quindi `B[0]=1;`
2. iterando su `i`, poniamo `B[i]= (S[i] != S[i-1]);` infatti un nuovo atomo corrisponde ad un valore della sequenza *diverso* dal precedente

Ogni partizione espressa in termini binari ha sempre 1 come primo elemento.

Facendo un esempio, ad una stringa possiamo semplicemente far corrispondere l'array binario corrispondente

```
S=AAAABBBAAAACCCDDDD
B=1000100100010010000
```

Avendo un array che indica esattamente dove si trovano i bordi dei vari atomi, le operazioni di intersezione e prodotto sono immediate. Il prodotto corrisponde alla partizione che contiene i bordi di entrambe le partizioni (OR binario); l'intersezione ha un bordo quando compare contemporaneamente nelle due partizioni (AND binario); il prodotto ridotto contiene invece ogni bordo di una e una sola delle partizioni (XOR binario) – se il bordo appartiene ad entrambe, appartiene al fattore comune, che è stato ridotto.

In tabella 2.1 sono mostrate tutte le sequenze generabili a partire da due partizioni binarie `A` e `B`, con le rispettive entropie, che adesso mostriamo come calcolare.

Vettore	Risultato	Codice C	H_S	H_T
α	1011000110011011101	A[]={1,0,1,1,...};	2.16	2.30
β	1000100100111001000	B[]={1,0,0,0,...};	1.73	1.79
$\sigma = \alpha \wedge \beta$	1000000100010011000	C[i]=A[i] & B[i];	1.33	1.39
$\alpha \vee \beta$	1011100111011101101	P[i]=A[i] | B[i];	2.40	2.49
$\hat{\alpha} \vee \hat{\beta}$	1011100010101001001	Prid[i]=A[i] ^ B[i];	2.09	2.19
$\hat{\alpha}$	1011000010001001001	Arid[i]=A[i] ^ C[i];	1.80	1.94
$\hat{\beta}$	1000100000100000000	Brid[i]=B[i] ^ C[i];	1.06	1.09

Tabella 2.1: Rappresentazione delle operazioni a partire da partizioni binarie. In rosso gli elementi riconducibili a partizioni di α, in blu quelli dovuti a β, in verde gli elementi di σ. Nel prodotto si vede l'effetto sovrapposto di blu e rosso. Nelle partizioni ridotte in verde sono stati evidenziati gli zeri dovuti alla riduzione, ovvero in corrispondenza agli atomi (1 verdi) di σ.

2.1.1 Entropia di una partizione binaria

Il calcolo più semplice è quello dell'entropia topologica, H_T. Avendo un vettore che indica esattamente l'inizio di ogni atomo, basta contare il numero di 1 presenti nel vettore e farne il logaritmo[1] .

Per l'entropia di Shannon H_S la cosa è leggermente più complicata[2]:

1. si trovano tutte le posizioni i_k degli 1 nel vettore dato
2. si fa la differenza degli interi cosi trovati, $\mu_k = i_k - i_{k-1}$
3. l'insieme $\{\mu_k\}$ rappresenta le lunghezze degli atomi, ovvero gli intervalli tra 1 e l'ultimo 0 (se esiste) dello stesso atomo

Per l'ultimo atomo ci vuole un trattamento speciale, $\mu_{end} = \mathrm{L} - i_{end}$, ovvero si calcola la differenza tra la lunghezza della sequenza e l'ultimo 1 nel vettore. Il calcolo dell'entropia del vettore $\{\mu_m\} = \{\mu_k\} \cup \mu_{end}$ è molto semplice:

$$L = \sum_m \mu_m$$

$$H = -\frac{\sum_m \mu_m \ln \mu_m}{L} + \ln L$$

la formula utilizzata permette di non dover dividere ogni μ_k per L, e di eseguire un'unica divisione alla fine. L'utilizzo delle misure nonnormalizzate degli atomi consente anche di poter precalcolare tutti i logaritmi necessari, in quanto $0 \leq \mu_k \leq L$, migliorando la velocità di esecuzione della routine del 600%.

L'algoritmo presenta complessità $\mathcal{O}(L)$ in quanto si scorre l'array completo, anche se una volta sola: chiaramente non si può far di meglio.

2.1.2 Distanza di Rohlin

Una volta che si è in grado di fare il prodotto, la riduzione e il calcolo dell'entropia, si può immediatamente calcolare la distanza di Rohlin. Per questo motivo è necessario calcolare anche le partizioni ridotte, che sono superflue per il calcolo del prodotto ridotto, ma nella distanza entrano in gioco le entropie dei fattori.

In tabella 2.2 mostriamo le 4 possibili distanze date le due partizioni binarie, sfruttando solamente i numeri calcolati nell'esempio precedente.

[1] in Matlab è immediato: `Htop = log(sum(B));`

[2] In codice Matlab, si definisce prima `H=@(a) -sum(a.*log(a))/sum(a) +log(sum(a));` Avendo il vettore binario A, l'entropia cercata è: `H(diff(find([A,1])));`

Distanza	Formula	Valore	R
Nonridotta, Shannon	$2H_S(\alpha\vee\beta)-H_S(\alpha)-H_S(\beta)$	0.93	-
Ridotta, Shannon	$2H_S(\hat{\alpha}\hat{\beta})-H_S(\hat{\alpha})-H_S(\hat{\beta})$	1.33	1.46
Nonridotta, topologica	$2H_T(\alpha\vee\beta)-H_T(\alpha)-H_T(\beta)$	0.87	-
Ridotta, topologica	$2H_T(\hat{\alpha}\hat{\beta})-H_T(\hat{\alpha})-H_T(\hat{\beta})$	1.35	1.53

Tabella 2.2: Tutte le possibili distanze ricavate a partire dalla tabella 2.1, con relativo fattore di amplificazione

2.2 Partizionamento generico

Nel caso generico non basta indicare l'inizio di ogni atomo, poichè questi possono avere forma (e buchi) di ogni tipo. È necessario etichettare ogni sito con un numero naturale, indicando con $\lambda_\alpha(i)$ l'etichetta corrispondente al sito i-esimo nella partizione α. Ogni etichetta identifica l'atomo di appartenenza:

$$i\sim j \iff \lambda(i)=\lambda(j)$$

Non si possono fare ipotesi sulla posizione relativa dei siti nè sulle etichette, che formano un vettore di lunghezza L.

2.2.1 Struttura dei vicini

Abbiamo definito il criterio di partizionamento come una relazione di equivalenza *globale*, che non dipende dalla relativa posizione dei singoli siti. In pratica, spesso andiamo a ricostruire la relazione a partire da equivalenze *locali*, definite tra siti "vicini"[3], mettendo nello stesso atomo siti "lontani" che sono equivalenti tra di loro tramite una sequenza di equivalenze locali.

Il metodo per partizionare una configurazione generica sullo spazio **M** consiste nel raccogliere tutta l'informazione in una matrice dei vicini `NNB`, concettualmente la versione ridotta (ad un numero opportuno di vicini) e *sparse* della matrice di adiacenza, per poi usare un algoritmo generale di clustering.

In linea di principio, il numero dei vicini per sito non è costante — associamo un vettore di lunghezza variabile sito per sito, a seconda della geometria e configurazione. Per reticoli regolari chiaramente possiamo fissare il numero massimo dei vicini[4], per ottimizzare il codice, indicando eventualmente con un simbolo convenzionale la fine del vettore (ad es. ci si ferma quando si trova -1). A seconda dell'input, possiamo distinguere vari casi:

- abbiamo in partenza la configurazione completa $\mathscr{C}$ su un reticolo regolare, consideriamo come collegati al sito i-simo tutti i primi vicini j aventi lo stesso simbolo, cioè
  $$d(i,j)=1,\ f(i)=f(j)$$
  e li aggiungiamo al vettore `NNB[i][k++]=j`, dove i indica la colonna, k cresce con il numero di j trovati. A questo caso appartiene ad esempio lo studio delle distanze tra reticoli con configurazioni di Ising: si considerano solo i primi vicini aventi lo stesso spin

- nel caso delle partizioni con salto, aggiungiamo a `NNB` solo il primo sito distante meno di n_s avente lo stesso simbolo, non serve collegare i più lontani (saranno già vicini del precedente)

- non guardiamo affatto le configurazioni, e i vicini sono dati interamente dalla matrice di adiacenza: `NNB[i]=nonzero(A[i])`

[3] Si intende qui nel senso più generale possibile, indipendentemente dalla posizione effettiva, ma legati da una qualunque relazione. Non vi è quindi alcuna perdita di generalità.

[4] Sia per primi vicini, che per un numero allargato, questo è costante.

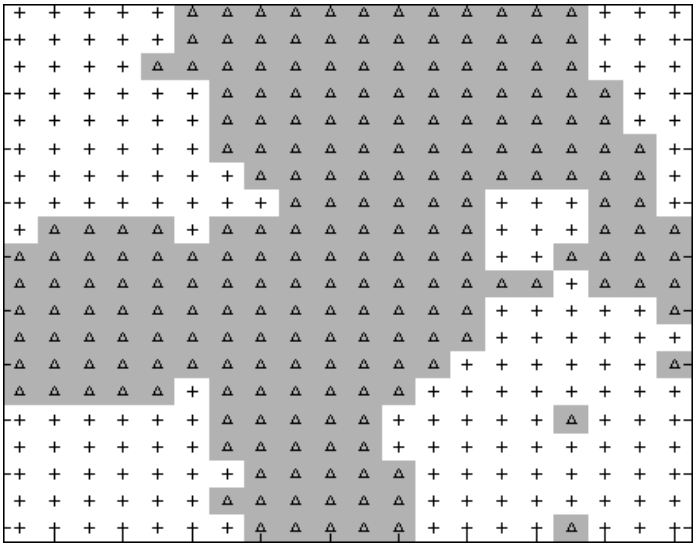

Figura 2.2.1: Reticolo semplice con $|\mathbb{K}| = 2$

Il caso in cui associamo insieme tutti i simboli dello stesso tipo presenti sul grafo, non è più utile partizionare a partire dall'elenco dei vicini[5]. Si utilizza un diverso algoritmo, una variante del sort descritto a pagina 22, per ottenere una configurazione in cui tutti i simboli uguali sono consecutivi.

Facciamo l'esempio del reticolo bidimensionale ai primi vicini, come si vede in figura 2.2.1. Scorrendo il reticolo in modo ordinato, dall'alto verso il basso, da sinistra verso destra, ovvero secondo la naturale disposizione dei siti in memoria, si vede che per allargare progressivamente (a macchia d'olio) le partizioni, è conveniente considerare i siti vicini già visitati. Per il sito i-esimo si guarda quindi il vicino in alto (*nearest neighbour up*, nnu) e quello a sinistra (*nearest neighbour left*, nnl). In figura 2.2.2 sono disegnate le linee che collegano ciascun sito con il corrispettivo `nnl` e `nnu`.

In generale reticoli regolari di dimensione d sono generati da d vettori ad entrate intere: per controllare tutte le relazioni di vicinanza di ogni sito del reticolo è quindi sufficiente esaminare d vicini per sito, invece del numero di coordinazione $z = 2d$. Avere un collegamento tra due siti è una relazione simmetrica, per cui non è necessario controllare in entrambi i versi. Se invece considerassimo solo un sottoinsieme dello spazio **M** per avere tutti i collegamenti bisogna effettivamente analizzare z primi vicini.

Problema. Qual è la struttura ottimale (minima) di vicini per riottenere la partizione data?

Basta unire uno-per-uno i siti all'interno di ogni atomo. Ovvero ogni sito ha un solo vicino, il sito che lo precede facente parte dello stesso atomo (o se stesso, se è il primo o unico). Questo è chiaramente sufficiente per ottenere di nuovo la partizione, si può scrivere come un vettore lungo L, indipendentemente dalla geometria del sistema e dal numero effettivo di vicini: è un modo estramamente comodo ed efficiente di catturare le informazioni a partire da `NNB`, necessarie per la riduzione, ma che possono occupare molto spazio in memoria. Denominiamo l'array contenente il vicino precedente come `previous[i]`.

2.2.2 Partizionamento

Una volta completato l'elenco essenziale dei vicini, si procede al partizionamento. L'algoritmo è noto in letteratura come algoritmo di Hoshen-Kopelmann[**?**], di cui utilizziamo la versione ottimizzata da Mark Newman[**?**]. Il problema trattato di *clustering* è equivalente a molti altri: bond-percolation, colorazione di grafo generico, ecc.

[5] I vicini sarebbero definiti dalla partizione una volta individuata.

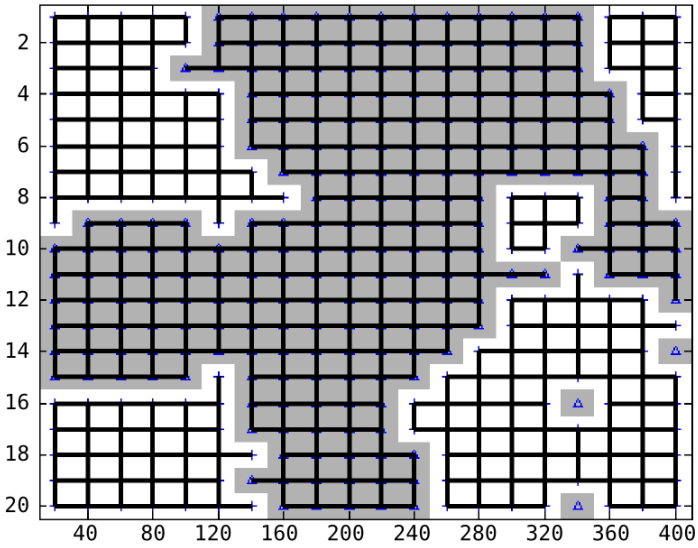

Figura 2.2.2: Collegamento dei siti con stesso simbolo, solo tramite vicino up e vicino left, essendo sufficienti a completare tutti i collegamenti (condizioni al contorno nonperiodiche).

Per il procedimento si crea un array `labels[]`, lungo L, che alla fine dell'algoritmo è codificato nel seguente modo:

`labels[i]<0` `i` è la *root* o *radice* del suo cluster, e `-labels[i]` rappresenta la dimensione del cluster – ovvero quanti siti vi appartengono in totale

`labels[i]>0` il sito punta ad un altro cluster, con *root* alla posizione `labels[i]`

L'array è un insieme di radici e di puntatori ad altri siti – rappresenta cioè un albero, con collegamenti da ogni foglia (sito), alla radice corrispondente, che contiene il numero di siti che le appartengono. Per sapere a che cluster appartiene un qualunque sito, definiamo la funzione `findroot`:

```
int findroot(int i,int labels[]) {
        if (labels[i]<0) return i;
        return labels[i] = findroot(labels[i],labels);
}
```

definita ricorsivamente sull'albero. Se il sito ha un label negativo, allora è radice. Altrimenti, risaliamo al sito a cui punta e reitero il procedimento. Ad ogni passo si esegue la *path compression*: ogni sito su cui si passa è aggiornato per puntare direttamente alla radice – in questo modo la volta successiva il percorso non deve essere ricostruito.

L'algoritmo è il seguente:

1. ad ogni sito si associa l'atomo banale, formato dal sito stesso, `labels[i]=-1`. In questo modo da subito ad ogni sito corrisponde un cluster, che verrà eventualmente allargato unendolo con i suoi vicini.

2. per ogni sito `i`, in ordine crescente, controlliamo (fino ad esaurimento) il vettore `NNB[i]`.

   (a) cerchiamo il cluster di appartenenza, `ri=findroot(i,labels)`

   (b) per ogni vicino `j`, identifichiamo a sua volta il cluster, `rj=findroot(j,labels)`

   (c) se la radice è la stessa, ovvero appartengono già allo stesso cluster, non si fa nulla

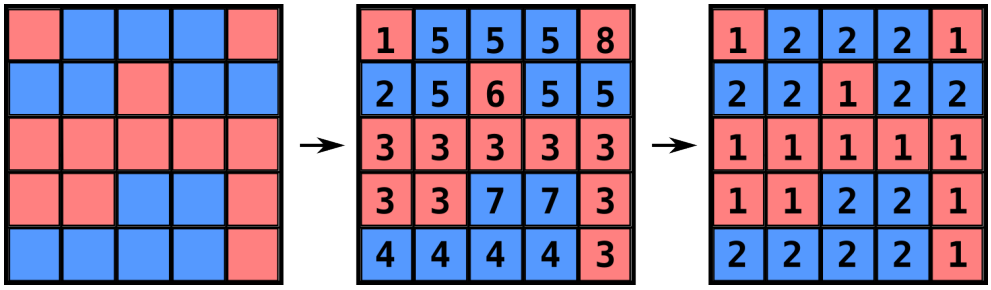

Figura 2.2.3: Funzionamento dell'algoritmo di Hoshen-Kopelman: dalla configurazione indicata dai due colori, si formano i cluster guardando solo ai primi vicini, assegnando una nuova etichetta quando non vi è un primo vicino dello stesso tipo; successivamente si riconoscono globalmente le etichette equivalenti, anche attraverso le condizioni al bordo periodiche.

(d) se le radici sono diverse, si uniscono i cluster. C'è sicuramente un cluster più grosso (o uguale) e uno più piccolo, rispettivamente di radice `rmax` e rmin. Per fare il numero minimo di cambiamenti, si aggiunge alla radice il numero di siti accorpati

```
labels[rmax] -= labels[rmin]
```

e si trasforma `rmin` in una foglia del cluster più grosso:

```
labels[rmin] = rmax
```

La volta successiva che si cerca la root dei siti che appartenevano al cluster più piccolo, si risalirà fino alla nuova radice — il cluster è stato effettivamente assimilato, semplicemente cambiando il valore di un puntatore.

3. alla fine tutti i siti sono stati controllati, `r=findroot(i, labels)` punta alla radice di appartenenza, lo spazio è partizionato, l'intero `r` identifica in maniera univoca le classi di equivalenza.

Riassumento, l'algoritmo corrisponde sostanzialmente a due passaggi, che nel nostro caso avvengono contemporaneamente ma un sito alla volta, illustrati in figura 2.2.3:

- Collegamento dei vicini (FIND)
- Raggruppamento graduale dei cluster così trovati (UNION)

L'algoritmo è ottimale, con runtime $\mathcal{O}(L \ln L)$ e senza bisogno di memoria aggiuntiva. Il fattore logaritmico è dovuto alla fase di unione dei cluster, che in generale richiede $\mathcal{O}(\ln L)$ passi per sito per trovare la radice di appartenenza.

2.2.3 Relabeling

Una volta che abbiamo il vettore `labels[]`, si può utilizzare le informazioni ivi contenute. Si costruisce un vettore di atomi, definendo per ciascuno una struttura che contiene informazioni quali le dimensioni, dove inizia, dove finisce, il *hash* corrispondente.

Per avere il numero degli atomi e inizializzare il vettore corrispondente basta contare quanti siti hanno `labels[i]<0`. Contemporaneamente possiamo associare ad ogni atomo le sue dimensioni, `-labels[i]`, il suo sito di inizio `i`, e l'ultimo sito trovato, temporaneamente `i`. Ogni elemento del vettore labels punta ad un sito (o indica lo stesso come radice) per cui è contenuto nell'intervallo $[0,\dots,L]$ ed è possibile tenere conto di ogni atomo che si incontra scorrendo tutti i siti e utilizzare questo contatore come nuova etichetta: non più la radice, ma il numero dell'atomo corrispondente in $[0,\dots,n]$.

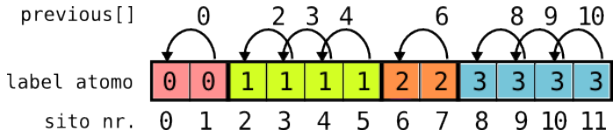

Figura 2.2.4: Per ogni sito è stato indicato l'atomo di appartenenza, sotto l'indice, sopra il valore del vettore che punta al sito precedente appartenente allo stesso atomo.

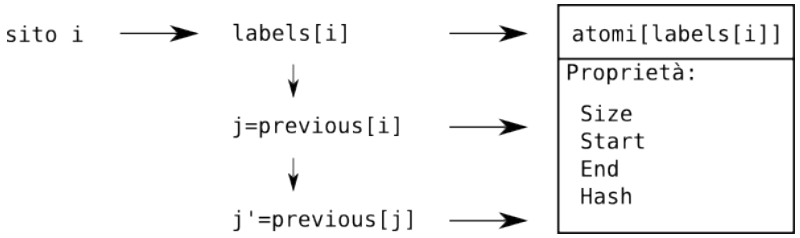

Figura 2.2.5: Sintetica rappresentazione della struttura in memoria che rappresenta le quantità collegate ad ogni sito

Avendo un elenco di atomi contente le dimensioni di ciascuno, si calcola l'entropia della partizione in modo diretto, sommando le dimensioni dei vari atomi moltiplicate per il loro logaritmo e normalizzando.

Generiamo inoltre il vettore ottimale dei vicini: `previous[]`. Per ogni sito `i`, conosciamo infatti l'atomo di appartenenza e il precedente sito trovato appartenente allo stesso atomo: `previous[i]` conterrà proprio questo dato. Aggiorniamo poi il puntatore all'ultimo sito trovato con `i` stesso. Andando a leggere il vettore `previous[]` in modo ordinato, abbiamo una lista ordinata, che ad ogni intero associa un numero strettamente minore o uguale (se il sito non ha un precedente).

2.2.4 Hashing

Definiamo un *hash*[6] come un modo univoco di associare ad ogni atomo un elemento dello spazio delle stringhe a 64bit — il numero degli atomi è sicuramente molto inferiore a 2^{64} per cui con un buon algoritmo è improbabile che due atomi diversi ottengano lo stesso identificativo[**?**][**?**].

La conoscenza di un atomo corrisponde alla lista di siti che lo compongono, tuttavia sarebbe oneroso tenere da parte un'altra volta questa lista, ordinata stavolta per atomo, oppure andarla a ricostruire ogni volta. Per controllare l'uguaglianza tra due atomi di due diverse partizioni, bisognerebbe fare un confronto sito per sito. Se ogni atomo diverso è distinguibile solo con il confronto di un intero (operazione atomica), il confronto tra due atomi è immediato e $\mathcal{O}(1)$ in tempo, indipendentemente dalle dimensioni dell'atomo.

La chiave sta nel trovare un buon algoritmo di hashing, che sparpagli i bit di informazione contenuti dall'atomo in maniera uniforme nello spazio delle stringhe binarie a 64bit, in modo che a input diverso corrisponda *sempre* (a meno di una probabilità idealmente prossima a 2^{-64}) un identificativo diverso. L'algoritmo utilizzato in questo caso è l'estremamente semplice *hash DJB*[7], dopo un confronto con alcuni altri algoritmi[**?**], risultati dare molte collisioni, ovvero capitava che triple di interi diverse dessero risultati uguali[**?**].

Gli atomi sono equivalenti ad array di interi, per cui l'algoritmo prende un array di lunghezza arbitraria di interi a 32bit e restituisce un intero a 64bit, partendo da un valore costante iniziale, fissato empiricamente a 5381. L'aggiunta di un intero `value` alla variabile `hash` avviene nel modo sequente:

[6] Etimologicamente, *to hash* vuol dire maciullare, sparpagliare
[7] Acronimo dell'ideatore, D. J. Bernstein

ovvero

```
void DJBHash_step(u_int64_t &hash, u_int32_t value) {
    hash = (hash << 5) + hash + (0xff000000 & value);
    hash = (hash << 5) + hash + (0x00ff0000 & value);
    hash = (hash << 5) + hash + (0x0000ff00 & value);
    hash = (hash << 5) + hash + (0x000000ff & value);
}
```

la variabile a 32bit che codifica il valore del sito viene spezzata in 4 pezzi da 1 byte ciascuno; ogni byte viene sommato al hash, mentre questo viene rimischiato tramite uno shift e una somma del valore precedente. Senza spezzettare i 4 byte il rimescolamento dei bit risulta insufficiente e il hash perde di unicità.

Il hash viene calcolato intanto che si scorre l'array dei labels e si costruisce il vettore `previous`. Per ogni sito `i` recuperiamo l'indice dell'atomo corrispondente `labels[i]`, che contiene una variabile preposta all'immagazzinamento del hash, al quale aggiungiamo il valore di `i` di volta in volta.

2.3 Operazioni tra partizioni generiche

2.3.1 Prodotto

Dovendo ragionare sui label assegnati ai siti, cerchiamo un modo di labellare in modo univoco i siti nella partizione prodotto, con un'operazione sito per sito, ovvero locale. Dalla definizione si ottiene che un sito i appartiene all'atomo $(\alpha \vee \beta)_{jk}$ se appartiene a $A_j \cap B_k$, ovvero se ha $\lambda_\alpha(i) = j$ e $\lambda_\beta(i) = k$.

Il prodotto deve mantenere la topologia, quindi prendendo due siti "vicini", ovvero appartenti allo stesso atomo in entrambe le partizioni, devono rimanere nello stesso atomo del prodotto

$$m, n \in A_j \text{ e } m, n \in B_k \implies m, n \in (\alpha \vee \beta)_{jk}$$

ma i siti m, n hanno label $\lambda_\alpha(m) = \lambda_\alpha(n) = j$ e $\lambda_\beta(m) = \lambda_\beta(n) = k$ rispettivamente per le due partizioni — devono avere lo stesso label nel prodotto, che può dipendere solo dal label degli atomi di partenza. L'unica scelta possibile è assegnare label (j,k) (o equivalente) ad ogni sito del prodotto, ovvero la coppia (ordinata) dei label di partenza. Una volta eseguito l'assegnamento, gli atomi sono riconosciuti cercando label dello stesso valore. Tutto il procedimento è illustrato in figura 2.3.1.

Se i label sono rappresentati da interi a 32bit, la coppia può essere rappresentata da un intero a 64bit, avente nei primi 32bit il valore del primo label, e negli ultimi quello del secondo: in questo modo tutte le operazioni sono atomiche su un moderno processore.

Ricapitolando: al sito i-esimo, partendo da $\lambda_\alpha(i) = j$ e $\lambda_\beta(i) = k$, si assegna nel prodotto $\lambda_{\alpha\vee\beta}(i) = (j,k)$:

```
prod[i] = alpha[i] << 32 | beta[i];
```

come si vede dalla formula esplicita, $(j,k) \neq (k,j)$ in quanto gli indici identificano atomi appartenenti a partizioni diverse, con siti in genere diversi – se al posto del OR binario si volesse usare un'altra operazione, va cercata una asimmetrica nei due indici, per non unire atomi distinti. Ad esempio una partizione con labels {1,2} è equivalente a {2,1} (sempre due atomi distinti); il prodotto è ancora una partizione con due atomi, ma le etichette del prodotto sono: {(1,2),(2,1)} – devono essere chiaramente diverse.

Anche in questo caso il prodotto comporta un numero di operazioni sito-per-sito, con complessità $\mathcal{O}$(L).

2.3.2 Entropia

Per calcolare il numero di siti di un atomo, ovvero corrispondente ad ogni etichetta univoca, bisogna innanzitutto scorrere l'elenco per cercare tutte le etichette e ogni volta che

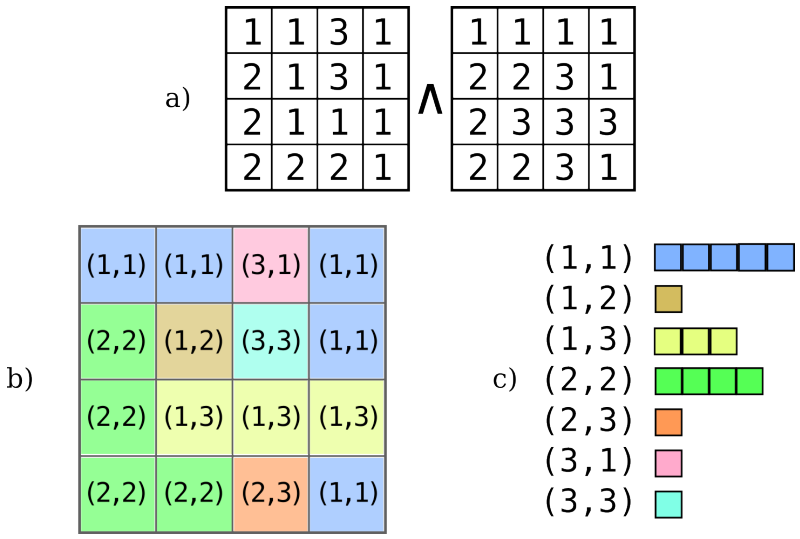

Figura 2.3.1: Calcolo di un prodotto tra partizioni. a)partizioni di partenza; b)partizione prodotto; c)lunghezze degli atomi ordinati del prodotto,al fine di calcolare l'entropia

si trova una non precedentemente catalogata, scorrere l'intero array in avanti e contare quante volte appare. Questa è chiaramente un'operazione con complessità

$$\sum_{i=1}^{L}(L-i)=\mathcal{O}(L^2)$$

Ma si può fare meglio. Infatti non è necessario scorrere N volte l'intero elenco dei labels. Si può riordinare questo elenco tramite un'operazione di *sort*[8]. L'andamento asintotico è noto in letteratura essere $\mathcal{O}(L\log_2 L)$ nel numero di siti, con una costante di proporzionalità dipendente dall'algoritmo, ma non particolarmente onerosa.

Ordinando l'elenco delle etichette (non importa neanche se in ordine crescente o decrescente) si ottiene un array in cui sicuramente tutti gli elementi uguali sono consecutivi – si procede a questo punto al calcolo dell'entropia come se si trattasse di una sequenza unidimensionale.

L'altro metodo, computazionalmente più semplice, $\mathcal{O}(L)$ sia in runtime che in memoria, che consiste nel numerare temporaneamente tutti i label trovati in ordine crescente e rietichettarli con la successione $\{1,2,\dots\}$ non è gestibile, in quanto i label possono essere distribuiti in modo "sparso" nello spazio delle coppie (j,k). In generale vi possono essere al massimo L atomi nel prodotto, ma $L\times L$ coppie, quindi sicuramente non tutte sono realizzate. La complessità dell'organizzarle in qualche altro modo, ad esempio in un albero binario bilanciato o un hash – tutte superano in fretta la semplicità del *sort* e anche nel caso migliore hanno lo stesso andamento asintotico. In pratica l'andamento è polinomialmente peggiore, per motivi di mal gestione della cache L1 e L2 del processore, in quanto sia gli alberi che i hash risultano nonlocali, mentre un sort sposta gli elementi in modo ordinato e accede alla memoria in modo più lineare.

Una volta calcolata l'entropia di ogni partizione, sia di Shannon che topologica, la distanza di Rohlin segue immediatamente dalla formula 1.1.3 a pagina 7, senza nessuna modifica.

2.3.3 Differenza simmetrica

Pe per calcolare la differenza tra due atomi, illustriamo come calcolare in modo efficiente la differenza simmetrica tra due insiemi.

[8] Nello specifico ci affidiamo al *quicksort* delle librerie glibc

```
A1 = {9, 8, 5, 4, 1, 0}
A2 = {8, 7, 5, 4, 2, 1}

allineamento1   9 8 - 5 4 3 - 1 0
allineamento2   - 8 7 5 4 - 2 1 -
differenza?     + . + . . + + . + == 5
```

Figura 2.3.2: Differenza simmetrica tra insiemi ordinati, rappresentata testualmente

Dati due insiemi ordinati di dimensione n e m, è possibile calcolare la differenza simmetrica in modo lineare, con al massimo $2(n+m+1)$ operazioni, mentre per due insiemi disordinati sono necessarie $\mathcal{O}(n\,m)$ operazioni. Il procedimento è illustrato in figura 2.3.2.

La prima fase è quella di allineamento, ovvero si scorrono i due array rappresentanti gli insiemi, aggiungendo spazi all'uno o all'altro, in maniera tale che i valori nonvuoti siano uguali nelle due righe. Successivamente si contano le differenze tra le due stringhe di simboli allineati.

Chiaramente si possono fare entrambe le operazioni contemporeanemente, contando $+1$ ogni volta che si aggiunge uno spazio. Quando si vuol sapere se due atomi hanno al massimo n_ϵ siti diversi, ci si può fermare dopo aver inserito n_ϵ spazi. Non è poi neanche necessario rappresentare gli spazi in memoria, basta scorrere in avanti il puntatore corrispondente per allinare al volo intanto che si leggono gli insiemi stessi.

L'operazione è estremamente efficiente se si ha l'elenco corrispondente ai siti dei due insiemi. Nel nostro caso, a partire da un qualunque sito, è sufficiente usare il vettore `previous[]` per ottenere il sito precedente, sicuramente ordinato.

2.3.4 Intersezione

Definiamo ora il modo operativo di calcolare la partizione comune, $\sigma = \alpha \wedge \beta$. Descriviamo il processo prima in termini di atomi, poi in termini di siti, per arrivare all'algoritmo ottimale.

La partizione intersezione, σ essendo la partizione più fine tale che $\sigma \leq \alpha$ e $\sigma \leq \beta$, ha atomi formati dall'unione minimale di atomi sia di α che di β. Ovvero ogni atomo $C_k = \cup_j A_j = \cup_m B_m$. Chiaramente andare a cercare le combinazioni di A_j e B_m necessarie è un'impresa disperata.

In termini di siti, vediamo che ogni atomo C_k collega siti appartenenti a diversi atomi di α se questi sono collegati in β. Questo corrisponde già ad un utile criterio:

$$i \sim j \text{ in } \sigma \iff i \sim j \text{ in } \alpha \vee i \sim j \text{ in } \beta \tag{2.3.1}$$

che diventa ancora più utile se ci si ricorda che la relazione di equivalenza tra siti è mantenuta a livello locale dalla struttura `NNB` precedentemente introdotta. Ovvero possiamo esprimere la relazione di equivalenza 2.3.1 a livello locale, sito per sito, ottenendo la corretta partizione globale alla fine!

La partizione σ ha anch'essa una struttura di vicini corrispondenti:

- i è vicino di j anche se è solo vicino in α
- i è vicino di j anche se è se è vicino in β

la partizione σ sarà quindi ricostruita tramite un'opportuna struttura `NNB`, che ha in generale il doppio dei vicini $\tilde{z}(\sigma) = \tilde{z}(\alpha) + \tilde{z}(\beta) = 2\tilde{z}$, ovvero sito-per-sito l'unione dei vicini di α e β.

- $\texttt{NNB}_\sigma\texttt{[i][2k]=NNB}_\alpha\texttt{[i][k]}$
- $\texttt{NNB}_\sigma\texttt{[i][2k+1]=NNB}_\beta\texttt{[i][k]}$

dove per $\texttt{NNB}_\alpha$ si intende la struttura corrispondente utilizzata per il partizionamento di α, $0 \leq k < \tilde{z}$.

Una volta definita la struttura dei vicini, si partiziona l'array contenente tutti i siti con l'algoritmo descritto a pagina 18.

Ottimizzazione Non è in realtà necessario aggiungere i vicini di ogni sito, come definiti originariamente dalla struttura `NNB` corrispondente. Esiste un vettore di vicini ottimale, che ricostruisce esattamente le partizioni originarie: `previous[]`.

Il problema dell'intersezione è quindi ridotto a considerare 2 vicini per ogni sito, invece che $2d$ senza quest'ottimizzazione! Per un qualunque sito definiamo quindi

- $\texttt{NNB}_\sigma$`[i][0]=`$\texttt{previous}_\alpha$`[i]`
- $\texttt{NNB}_\sigma$`[i][1]=`$\texttt{previous}_\beta$`[i]`

2.4 Riduzione

Una volta che si è in grado di calcolare l'intersezione, è possibile calcolare le partizioni ridotte. Vediamo ora come implementare i diversi criteri esposti a pagina 8. Lo scopo principale è quello di evitare la costruzione , il prodotto e l'intersezione dei fattori dicotomici, per motivi di tempo e occupazione in memoria.

Visto che le partizioni ridotte servono solo al calcolo del loro prodotto, costruiremo per ciascuno i giusti labels. Non serve costruire i vettori `previous[]`, nè le strutture degli atomi, ma solo calcolare l'entropia a partire dai labels con il metodo presentato a pagina a pagina 22 visto che è necessaria per la distanza di Rohlin.

2.4.1 Confronto con l'intersezione

Per ridurre due partizioni α e β definiamo innanzitutto l'intersezione $\sigma = \alpha \wedge \beta$ calcolata con l'algoritmo appena presentato. In generale per fare molti confronti, sarà necessario usare una variabile temporanea che contenga l'intersezione di volta in volta.

Costruiamo le partizioni dicotomiche semplici, ovvero a partire da ogni atomo $A_k \in \alpha$, definiamo $\hat{\alpha}_k = \{A_k, A_k^c\}$:

Costruzione in memoria si tiene l'elenco degli atomi, ovvero un array di strutture `atomi[]`, su cui possiamo scorrere. Per ogni atomo (ad esempio il k-esimo) conosco il sito finale, `atomi[k].end`, ricostruisco gli altri siti appartenenti tramite iterazione del vettore `previous[atomi[k].end]`. Ho inoltre il sito di inizio, `atomi[k].start`.

Confrontiamo ogni partizione dicotomica $\hat{\alpha}_k$ con l'intersezione, $\rho = \hat{\alpha}_k \wedge \sigma$. Se $\rho \neq \nu$, la partizione banale, scarto la partizione dicotomica $\hat{\alpha}_k$. Si hanno i possibili casi:

Intersezione nonbanale $\exists C_m \in \sigma$ tale che $A_k = C_m$: in questo caso l'atomo di α corrisponde esattamente ad un atomo di σ, ovvero hanno tutti i siti in comune. La loro intersezione conterrà almeno l'atomo $\hat{\alpha}_k$, per cui sarà non banale $\Longrightarrow$ scarto A_k.

Intersezione banale Se A_k non corrisponde esattamente a un qualche C_m, allora σ unisce più atomi di α. In questo caso, la partizione σ collega siti di A_k con siti del suo complementare, banalizzando l'intersezione con la partizione dicotomica $\Longrightarrow$ $\hat{\alpha}_k \wedge \sigma = \nu$.

Come si calcola se esiste esattamente un atomo C_m corrispondente? Visto che stiamo iterando sull'elenco degli atomi di α, conosciamo sicuramente un sito appartenente ad A_k che chiamiamo j, `j=atomi[k].end` e l'atomo corrispondente al sito j nella partizione σ, tramite il vettore delle etichette di σ. Fatto questo, abbiamo le strutture degli atomi corrispondenti al sito j per entrambe le partizioni, per cui conosciamo il *hash* corrispondente a C_m e a A_k. Se il *hash* è uguale, allora gli atomi sono uguali! Per il confronto del fattore dicotomico generato dall'atomo A_k abbiamo quindi in totale fatto 4 operazioni veloci:

1. Dall'indice k si trova la struttura contente un sito j e il hash, `atomi[k]` — 1 lookup
2. Si cerca l'indice dell'atomo corrispondente in σ, $\texttt{labels}_\sigma$`[j]` — 1 lookup
3. Si trova la struttura corrispondente all'atomo in σ e si legge il *hash* — 1 lookup
4. Si confrontano i due hash — 1 confronto

Rimane da moltiplicare insieme tutte le partizioni dicotomiche rimaste, per ottenere la partizione ridotta $\hat{\alpha} = \vee_k \hat{\alpha}_k$. Numeriamo i fattori dicotomici rimasti con un indice crescente, a partire da 2. Vediamo come in ogni partizione dicotomica vi sia un atomo della partizione originaria e il suo complementare: gli atomi sono disgiunti, per cui il prodotto dei singoli atomi risulta vuoto. Rimangono da moltiplicare tutti i complementari per gli atomi rimasti: genero nuovamente esattamente gli atomi che non ho scartato nella riduzione e un ulteriore atomo (generalmente sconnesso) composto dall'unione di tutti gli atomi scartati.

Ricostruzione Creiamo una partizione banale con tutti i label pari a 1, a rappresentare l'atomo di "background". Mettiamo un label $k \geq 2$ ai siti di ogni atomo A_k che non è stato scartato nel confronto. Questo è semplice, basta porre uguali a k tutti i siti partendo da `j=atomi[k].end` e iterando tramite `j=previous[j]` fino a che non arriviamo all'inizio dell'atomo. Alla fine abbiamo una partizione in cui l'atomo disconnesso con gli scarti ha label 1, mentre gli altri siti hanno label numerati a partire da 2. Il processo è chiaramente proporzionale al numero di siti che abbiamo dovuto riscrivere, quindi

$$\mathcal{O}(L) + \mathcal{O}\left(\sum_k \mu(A_k)\right) = \mathcal{O}(L)$$

Abbiamo costruito in tempo lineare la partizione ridotta, in quanto per ogni atomo c'è un confronto atomico e il relabeling del risultato corrispondente.

2.4.2 Confronto diretto

In questo tipo di riduzione, confrontiamo direttamente gli atomi di due partizioni, α e β, scartando le partizioni dicotomiche corrispondenti, nel caso in cui gli atomi soddisfino il criterio prescelto.

Per evitare problemi di definizione, consideriamo la partizione β fissata, costruendo $\hat{\alpha}$ a partire dal confronto di α e β.[9]

Abbiamo due criteri di confronto:

1. Scartiamo l'atomo A_k se è uguale a $B_m \in \beta$ definito sugli stessi siti (usando il confronto tra hash)
2. Scartiamo l'atomo A_k se differisce da $B_m \in \beta$ corrispondente meno di ϵ in misura:

$$\mu\left(A_k \triangle B_m\right) \leq \epsilon$$

Si procede molto similmente al caso del confronto con σ, in quanto iteriamo su tutti gli atomi di α, troviamo i siti corrispondenti all'atomo cercato, da questi gli atomi di β sopra quei siti e eseguiamo il confronto.

La differenza simmetrica è stata definita in termini di numero di siti diversi n_ϵ, mentre sopra in termini della misura normalizzata che deve essere $\leq \epsilon$, definiamo dunque:

$$\epsilon = \frac{n_\epsilon}{L}$$

La partizione $\hat{\alpha}$ è ricostruita a partire dagli atomi rimasti come spiegato nella sezione precedente. L'unica differenza è insomma che il confronto viene fatto con gli atomi corrispondenti di β e non di σ e che il criterio non è l'uguaglianza, ma l'uguaglianza a meno di ϵ (che può essere 0).

[9] Poi si fa il contrario, tenendo fisso α, generando $\hat{\beta}$.

Capitolo 3

Proteine dei virus dell'influenza

Esponiamo in questo capitolo i risultati ottenuti dalle distanze tra sequenze di proteine, ottenute dai database biologici, dei virus dell'influenza A umana, di tipi H1N1 e H3N2.

Lo studio delle sequenze comprende fasi separate:

1. Raccolta delle sequenze e preanalisi
2. Compilazione matrice delle distanze tra le sequenze
3. Clustering a partire dalle distanze
4. Estrazione informazioni dai cluster e predizioni

Introduciamo innanzitutto il virus dell'influenza e alcune sue caratteristiche fondamentali, che motivano la scelta di che cosa analizzare con i metodi entropici. Successivamente esaminiamo nel dettaglio i passi elencati sopra.

3.1 Il virus dell'influenza

I virus dell'influenza sono caratterizzati da genomi segmentati in forma di RNA a singolo filamento negativo, dipendenti da una RNA-polimerasi di origine virale per la trascrizione prima della replicazione[**?**].

I virus sono dei tre tipi A, B e C, rappresentanti tre dei cinque generi della famiglia *Orthomyxoviridae*, caratterizzati da genomi segmentati. I virus hanno un'origine genetica comune, tuttavia la divergenza evolutiva ha reso impossibile il riassortimento di materiale genetico tra i tre diversi tipi.

I virus dell'influenza B sono caratterizzati da un minore tasso di mutazione (circa $1/3$ rispetto all'influenza A) e attaccano quasi esclusivamente l'uomo, quindi non tendono a scatenare pandemie. I virus del tipo C hanno un tasso di mutazione ancor più lento e sono asintomatici, per cui concentriamo la nostra attenzione sul virus A, il più pericoloso e fonte di annuali epidemie – definite come trasmissioni prolungate, diffuse da persona a persona.

3.1.1 Influenza A

La forma del virus A è sferica, dal diametro di circa 100 nm, ricoperta di numerose proteine, circa 500 molecole di HA e 100 di NA, che protudono dalla superficie. Sono in particolare *antigeni*, termine che vuol dire **gen**eratore di **anti**corpi, in quanto il sistema immunitario riconosce e reagisce al virus proprio tramite gli antigeni.

All'interno il virus contiene 8 segmenti di RNA, che codificano le proteine per la replicazione. Vediamo ora nel dettaglio le funzioni delle proteine di superficie:

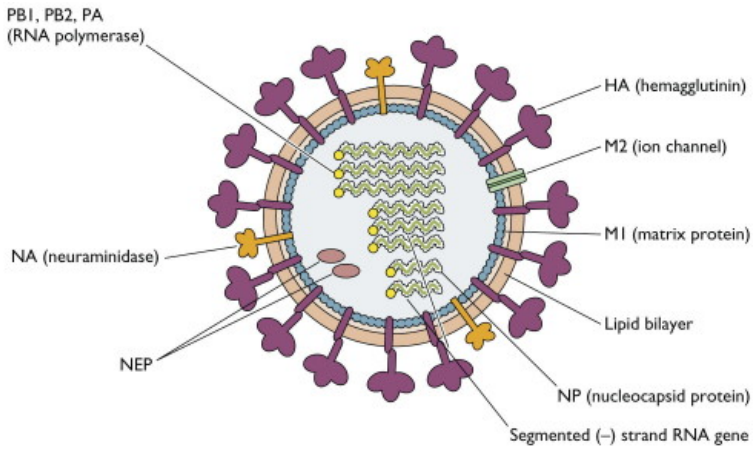

Figura 3.1.1: Forma del virus dell'influenza A, in cui sono evidenziati gli antigeni all'esterno e la disposizione del materiale genetico negli 8 segmenti.

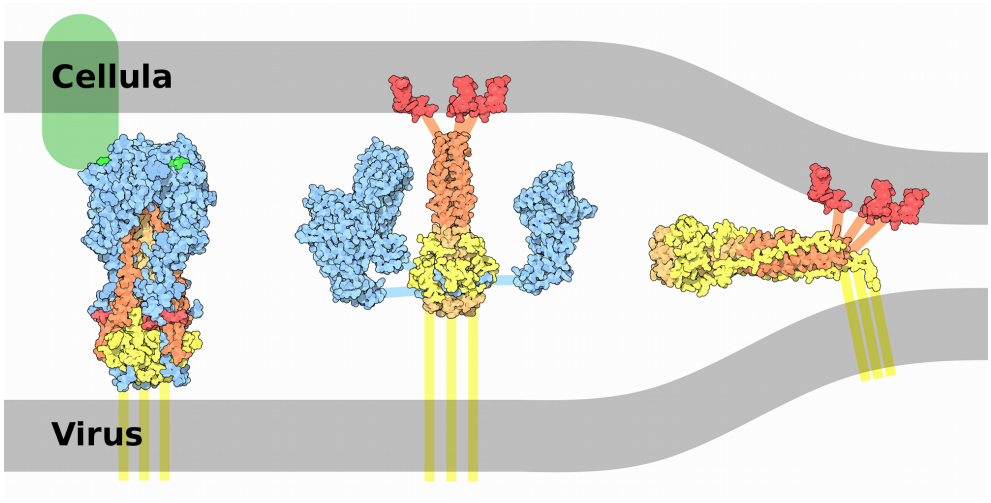

Figura 3.1.2: Illustrazione della molecola HA che protendendo dalla superficie del virus, aggancia la cellula e rende possibile l'iniezione del materiale genetico.

HA Emoagglutinina, proteina responsabile per il binding del virus alla cellula infettata. L'emoagglutinina forma delle "punte" sulla superficie del virus che servono per agganciarsi alla cellula ospite. L'aggancio è necessario per l'efficiente trasferimento del materiale genetico all'interno della cellula, un processo che può essere bloccato dagli anticorpi che si combinano all'emoagglutinina. La forma tridimensionale e il processo con cui la molecola attacca la cellula sono mostrate in figura 3.1.2[**?**].

NA Neuraminidase, proteina enzimatica che aiuta il rilascio della progenie virale dalle cellule infette.

Le rimanenti proteine M formano il capside (l'involucro del virus), le proteine NP e NS rilasciano il materiale nella cellule, le proteine PA e PB gestiscono la replicazione del RNA virale.

I virus di tipo A sono classificati in base ai possibili tipi di proteine di superficie: almeno 16 tipi di HA e 9 tipi di NA sono noti al momento della scrittura. Tutti le combinazioni sono espresse negli uccelli, che si considera essere gli originatori dell'influenza e di aver propagato il virus alle altre specie. Le combinazioni sono indicate dalle sigle HxNy, dove x e y sono gli indici corrispondenti, ad esempio:

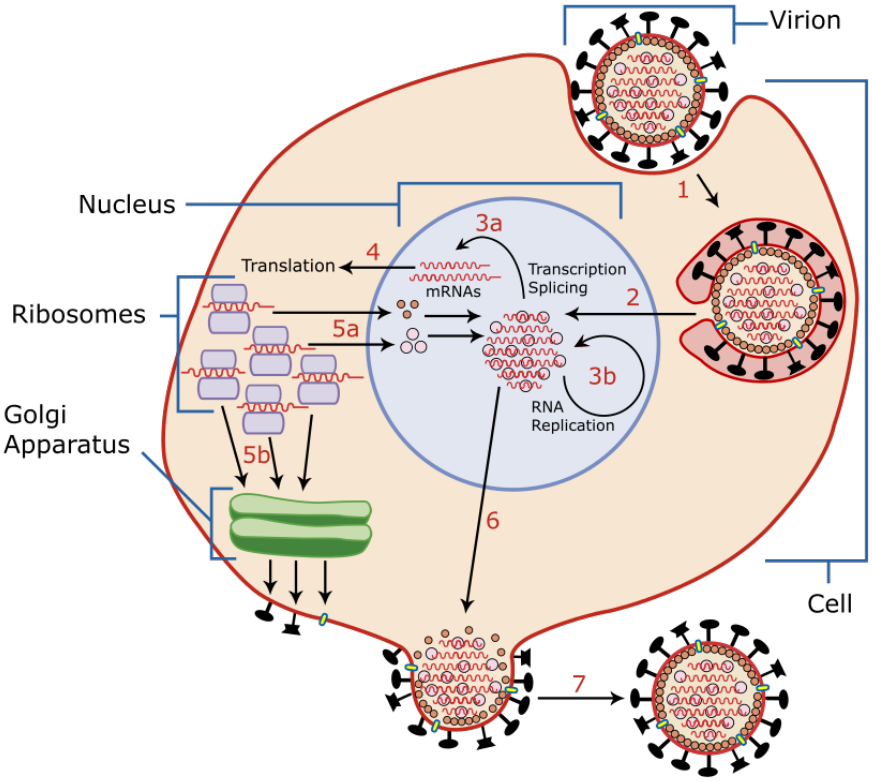

Figura 3.1.3: Processo di riproduzione del virus, in tutte le sue fasi.

H1N1	Virus di questo sottotipo hanno causato le pandemie nel 1918 e 2009
H3N2	La "solita" influenza stagionale
H5N1	Una recente variante di origine aviaria dalla forte virulenza, poco diffusa in occidente

Ogni sequenza campionata nei database biologici, è caratterizzata dalla stringa *strain*, ad esempio

A/Human/Fujian/411/2002 (H3N2)

indicante che si tratta di un virus del tipo A, sottotipo H3N2, isolato a Fujian nel 2002 da un essere umano (l'indicazione di provenienza da umano è di solito tralasciata), ed è la 411 sequenza campionata in quell'anno in quel laboratorio.

La pressione evolutiva è concentrata sulle proteine HA e NA, in quanto obiettivi del sistema immunitario. La particolare struttura del genoma del virus dell'influenza e la funzione delle proteine virali consentono due tipi di evoluzione per evadere le risposte immunitarie adattative a lungo termine in vari ospiti:

antigenic drift che corrisponde al normale processo di mutazione puntuale del materiale genetico. La replicazione è affidata all'RNA, che non supporta meccanismi di correzione dell'errore, con in media un errore di trascrizione per ogni replicazione del virus, il che vuol dire che ogni singolo virus è diverso dagli altri. L'enorme tasso di mutazione è la causa della rapida obsolescenza dell'immunità acquisita, che di solito non si estende da una stagione all'altra.

antigenic shift corrispondente alla ricombinazione del materiale genetico provenienti da virus diversi o da sottospecie diverse. Questo accade quanto una singola cellula è

infettata contemporaneamente da più virus. Il virus dell'influenza ha svariati segmenti indipendenti, che si possono rimescolare all'interno della cellula infetta. I virus risultanti avranno materiale genetico e proteico fornito dalla combinazione casuale di quelle dei virus che hanno attaccato la cellula. In questo modo da un virus H5N1 e H3N2 si possono formare combinazioni come H3N1 e H5N2 (oltre alle combinazioni di tutte le altre proteine all'interno del virus, non rilevate dal sistema immunitario).

L'*antigenic shift* è la mutazione più pericolosa, in quanto cambia radicalmente l'aspetto del virus per il sistema immunitario, rendendolo irriconoscibile e quindi non più attaccato dagli anticorpi. La pericolosità dello shift è che il virus A dell'influenza non attacca solo umani, ma anche maiali, uccelli, cavalli, pipistrelli e moltissimi altri tipi di animale. L'evoluzione di ciascun tipo procede quindi indipendentemente, fino a che tipi diversi di influenza non attaccano lo stesso individuo e ricombinano i materiali genetici.

Questo è quanto è accaduto nel caso dell'influenza suina. I maiali possono ammalarsi, con sintomi simili all'uomo, di influenze aviarie, suine ed umane contemporaneamente. In particolare la pandemia del 2009, del sottotipo H1N1, è stata causata dalla ricombinazione di sequenze di questi 3 tipi, con antigeni tipici degli uccelli e altre proteine di provenienza mammifera, permettendo al virus di attaccare anche gli esseri umani, il cui sistema immunitario era impreparato al nuovo tipo di virus, anche con trasmissione da uomo a uomo, scatenando una pandemia.

3.1.2 Vaccini

I primi vaccini contro l'influenza sono stati introdotti negli anni 40 del secolo scorso. La vaccinazione ha un effetto statisticamente rilevante nel prevenire un grande numero di casi d'infezione, ridurre i costi e le morti causate dalle complicazioni dell'influenza, presentando un rapporto costo-benefici positivo per tutte le categorie di popolazione definite "a rischio" per le quali viene raccomandata. Il vaccino conferisce però limitata *cross-immunity*, ovvero immunità per ceppi diversi da quello preso ad obiettivo dal vaccino — ciò comporta la necessità di una vaccinazione quasi ad ogni stagione influenzale[**?**].

Dato la grande variabilità antigenica dei virus dell'influenza, in media la composizione dei vaccini è cambiata ogni 2-5 anni a partire dal 1972. La composizione raccomandata dei vaccini per l'emisfero nord è decisa a febbraio, per l'emisfero sud a settimbre, circa mezzo anno prima dell'epidemia stagionale, rispettivamente a settimbre e a marzo, in modo da precedere l'epidemia e garantire la massima copertura per la popolazione.

I focolai di epidemie localizzate, della durata media dai 3 ai 6 mesi, generano la maggior parte delle nuove sequenze virali tramite antigenic drift[**?**], per poi diffonderle nel resto del mondo. L'effetto del drift è difficile da individuare, per il rimescolamento dei virus nella popolazione, per via degli shift, per bias nel processo di campionamento, ecc. Non sia se sia la somma graduale di mutazioni nel tempo accumulate durante le epidemie annuali, o sia un processo puntuato e stocastico che occorre al presentarsi di una mutazione che sfugge al controllo dell'immunità acquisita. La popolazione dei virus è sottoposta ad un collo di bottiglia evolutivo nel periodo estivo, durante il quale la popolazione di organismi infetti è minima e determina il ceppo principale della stagione successiva. È di notevole interesse scientifico e medico individuare quindi al più presto i ceppi dominanti per la stagione epidemica.

Le raccomandazioni per il vaccino sono compilate dal WHO**?**], tramite il suo Global Influenza Surveillance Network. La rete comprende 4 WHO Collaborating Centres (WHO CCs) e 137 istituti in 103 stati, riconosciuti dal WHO come Centri Nazionali per l'Influenza (NIC). I NIC raccolgono campioni nel loro paese, eseguono l'isolamento del virus e una caratterizzazione antigenica preliminare. Inviano poi i nuovi ceppi isolati per l'analisi genetica e antigenica di alto livello alle sedi del WHO, i cui risultati formano la base delle raccomandazioni.

Il vaccino è trivalente, protegge contro 3 diversi ceppi virali. Per tutte le stagioni il vaccino comprende gli stessi tipi, ma diversi ceppi. Per la stagione 2012-2013 le raccomandazioni sono le seguenti**?**]:

1. un sottotipo A/H1N1, A/California/7/2009
2. un sottotipo A/H3N2, A/Victoria/361/2011
3. un tipo B, B/Wisconsin/1/2010

Il ceppo è scelto tra i virus dello stesso sottotipo, da un'analisi antigenica *in vitro*. I campioni provenienti dai NIC vengono classificati in base alla potenza della reazione con l'antisiero sviluppato a partire dai ceppi candidati al vaccino della stagione successiva. Il ceppo che ha un numero maggiore di riscontri viene incluso nel vaccino. Il criterio è sempre in ritardo, in quanto non è in grado di predirre la prevalenza dei virus nella stagione successiva, ma solamente indicare qual è la più diffusa nella stagione precedente la raccomandazione. Negli ultimi anni questo ha causato 6 errori su 19, dove per errori si intende la non inclusione del ceppo dominante la stagione epidemica. In particolare la raccomandazione per la stagione 2011-2012 era:

1. un sottotipo A/H1N1, A/California/7/2009
2. un sottotipo A/H3N2, A/Perth/16/2009
3. un tipo B, B/Brisbane/60/2008

la raccomandazione per A/Perth è risultata inappropriata (pur con l'effetto mitigante di una parziale cross-immunity), in quanto un numero maggiore di casi è stato riportato durante la stagione epidemica per il ceppo A/Victoria, come predetto dall'algoritmo in [**?**].

3.2 Scelta delle sequenze da analizzare

Vi sono numerosi database di materiale genetico dell'influenza, creati attorno al 2004 per organizzare in maniera sistematica i risultati del sequenziamento nei diversi paesi del mondo e rendere possibili rapide ricerce, confronti, analisi tra sequenze virali dell'influenza di ogni tipo. I database raccolgono materiale di svariate specie, dalle sequenze più antiche, comprendenti quelle del virus della pandemia del 1918 fino alle sequenze raccolte l'11 ottobre 2011 — non sono state ancora caricate sequenze della stagione influenzale corrente.

Vi sono svariati database, tutti permettenti la scelta del sottotipo di influenza, delle proteine da visualizzare, la dettagliata provenienza e data, tutti finanziati dal NIAID, con sequenze dal Influenza Genome Sequencing Project, GenBank e altri database non specializzati, rendendo semplice la ricerca, il confronto e l'analisi di molte sequenze diverse:

Influenza Research Database o IRD, noto anche come FluDB[1] [**?**]

Influenza Virus Resource organizzato dal NCBI [2] [**?**]

Per rapportarsi ai risultati in [**?**] si è scelto le sequenze dal 1993 al presente, raccolte nel continente nord-americano, in cui la emoagglutinina è sequenziata completamente (al massimo in paio di "buchi"). Quando si è fatto il confronto con le sequenze della neuroaminidasi, sono state scelte le sequenze che hanno entrambe le proteine complete.

Rimane il problema di cosa fare per le sequenze con "buchi": l'entropia e la distanza di Rohlin sono ben definite solo se le sequenze hanno la stessa lunghezza. Abbiamo quindi allineato le sequenze con un software apposito, Muscle[**?**], che inserisce sequenze di simboli '-' nei *gap* per riallineare le sequenze — prendiamo quindi un alfabeto esteso, che comprende i simboli aggiunti. Come si vede dalla tabella, il numero dei gap è piccolo se consideriamo la proteina HA, e anche nel caso della NA la maggior parte delle sequenze non ha gap e meno dell'1% ne ha più di 3.

[1] http://www.fludb.org
[2] http://www.ncbi.nlm.nih.gov/genomes/FLU/FLU.html

Lunghezza	Conta
<560	2
564	3
565	2
566	1617
Totale	1624

a) HA

Lunghezza	Conta
<465	10
467-468	52
468	52
469	1510
Totale	1624

b) NA

Tabella 3.1: Distribuzione delle lunghezze delle sequenze di HA e NA come ottenute dal FluDB

Amminoacidi ambigui	Simbolo
Asparagina o acido aspartico	B
Glutammina o acido glutamico	Z
Leucina o isoleucina	J
Non identificato	X oppure '.'
Gap introdotto dall'alignment	–

Tabella 3.2: Tabella dei simboli aggiuntivi per l'alfabeto delle proteine

L'alfabeto di base $\mathbb{K}$ ha 20 simboli, ai quali si aggiungono 4 simboli che definiscono ambiguità, definiti in tabella 3.2. Il simbolo corrispondente ad amminoacido non identificato lo consideriamo distinto dal simbolo di allineamento '-'. Alla fine $\mathbb{K}$ ha 25 simboli, il che non è un problema, in quanto il metodo di partizionamento distingue solo simboli uguali o diversi.

C'è da stabilire cosa fare con le sequenze ridondanti, ovvero le sequenze ottenute da campioni diversi, con la stessa sequenza di proteine HA e NA (se solo una è uguale, le consideriamo sempre distinte), che pure corrispondono a una notevole parte di quelle raccolte. Le caratteristiche qualitative del clustering non dipendono chiaramente dall'avere dei doppioni, poichè sono sicuramente messi nello stesso cluster; è utile avere le sequenze con doppioni anche per pesare i cluster e capire quali sono i più rilevanti – la numerosità indica la diffusione del particolare ceppo. La rimozione dei doppioni va comunque effettuata una volta scaricate le sequenze, i database online sono in grado di farla, ma identificano e rimuovono separatamente le sequenze corrispondenti a HA e NA, per cui non vi è una identificazione 1-1 tra i campioni delle due proteine e non si riesce a fare lo studio contemporaneamente.

Abbiamo quindi vari campioni di sequenze sia per l'influenza di tipo H3N2 che di tipo H1N1, a seconda delle scelte che si possono fare, elencati in tabella 3.3. Non vi sono particolari differenze nei risultati considerando le sequenze con o senza doppioni, ogni volta espliciteremo di quale dataset stiamo parlando.

3.3 Clustering

Il *clustering* è definito come il modo di raggruppare oggetti nello stesso cluster sono più simili tra loro che a oggetti appartenenti ad altri cluster. Gli utilizzi sono molteplici: ri-

	Totale	Deduplicate	Ridondanti
H3N2 usate in [**?**]	1470	736	734 (50%)
H1N1 usate in [**?**]	2506	954	1552 (62%)
H3N2 dal IRD	1624	908	716 (44%)
H1N1 dal IRD	2908	1122	1784 (62%)

Tabella 3.3: Numero di sequenze considerate nei vari casi

conoscere oggetti come simili nonostante le differenze, distinguere il primo piano dallo sfondo, ottenere una efficiente compressione utilizzando solo l'indicativo del cluster di appartenenza, ecc. In particolare ci poniamo lo scopo di associare ogni sequenza virale ad un cluster, in maniera da poter parlare di k ceppi invece di N sequenze, con $k \ll N$, che è il modo più naturale di ragionare sul problema.

Una clusterizzazione è una partizione di un insieme, in cui ad ogni elemento i viene associato in modo rigido a un cluster C di appartenenza:

$$i \to C_i$$

attraverso una opportuna scelta di un criterio, basandosi sui dati ottenuti, sulle informazioni aggiuntive che si possono fornire, sul tipo di elementi dell'insieme. In generale il problema del clustering, ovvero la disposizione degli elementi in k cluster in modo tale da minimizzare una funzione globale di costo, è NP completo per la natura combinatoria del problema. Vi sono molti metodi euristici per ottenere soluzioni approssimate, di cui noi abbiamo verificato l'applicabilità, ma una descrizione completa è argomento di ricerca a sé stante [**?**].

L'algoritmo più semplice e utilizzato di questi è *kmeans*, che deve il nome al numero k di cluster da generare come parametro di input. Si richiede che i punti siano embedded in uno spazio vettoriale, rendendo difficile l'applicazione, dato che abbiamo solo una matrice di distanze. L'algoritmo cerca di separare gli elementi in gruppi di uguale varianza, minimizzando il "momento d'inerzia" definito come la somma delle distanze dall'elemento centrale di ogni cluster. Per ogni cluster si raggruppano gli elementi attorno a dei centri scelti inizialmente a random (di qui l'assunzione dell'embedding in $\mathbb{R}^n$), per poi spostare i punti da un cluster all'altro e ricalcolando i centri, fino il momento d'inerzia totale non si stabilizza attorno ad un minimo. Tuttavia partendo da una soluzione random si raggiunge ogni volta un minimo e una clusterizzazione finale diversa, spesso con pessimi risultati. Un modo innovativo di generare l'embedding e garantire un buon comportamento di kmeans è presentato nella sezione sui metodi spettrali.

Introduciamo ora il metodo principale utilizzato e anche il confrontato con alcuni algoritmi di clustering spettrale. Infine studiamo un criterio per decidere in quanti cluster partizionare l'insieme di partenza.

3.3.1 Clustering gerarchico

Una classe di algoritmi che prescinde dall'embedding in spazi vettoriali è quella *gerarchica,* in particolare *agglomerativa*[**?**]: "dal basso verso l'alto", in cui ogni sequenza è inizialmente considerata come un cluster banale e coppie di cluster vengono aggregate man mano che si sale lungo la gerarchia.

In generale è sufficiente avere un insieme di elementi da clusterizzare e una matrice di dissimilarità che serve per indicare come unire i cluster. La matrice di dissimilarità deve essere simmetrica a entrate positive – criteri esattamente soddisfatti dalla matrice di distanze derivate dalla metrica di Rohlin. Al primo passo definiamo n cluster banali, contenenti un solo elemento; in questo modo possiamo parlare sempre di cluster, e non distinguere fasi intermedie con cluster e elementi ancora non aggiunti. Ad ogni passo i due cluster che hanno distanza *minima* tra loro vengono aggregati in un nuovo cluster binario. Rimane da definire la distanza tra due generici cluster $\mathscr{A}$ e $\mathscr{B}$:

- Single linkage: la distanza tra cluster è definita come la minima distanza tra le coppie di elementi appartenenti ai due cluster.

  $$d(\mathscr{A},\mathscr{B}) = \min\{\, d(x,y) : x \in \mathscr{A}, y \in \mathscr{B}\}$$

  Non è un buona scelta in quanto la probabilità di aggregazione cresce con le dimensioni dei cluster presenti, i primi cluster formati quindi continuano a crescere per aggiunta di nuovi elementi – si forma un'albero molto sbilanciato in cui tutte le sequenze sono aggiunte gradualmente al primo cluster

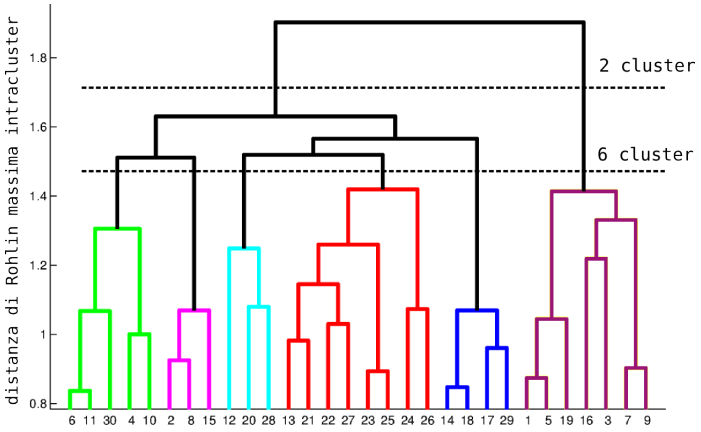

Figura 3.3.1: Dendrogramma costruito dai dati sulle distanze delle sequenze del H3N2 (mostrata solo parte superiore e non tutte le foglie). Sulle ordinate la distanza massima amessa dal complete linkage per accorpare sottocluster; si nota come salendo, cluster diversi si trovano uniti fino a che non si raggiunge la massima distanza e l'accorpamento termina. Tagliando ad altezze opportune, abbiamo due esempi: un taglio che forma due cluster, un taglio che corrisponde ai 6 cluster colorati.

- UPGMA, o average linkage: si prende la distanza media tra le coppie

$$d(\mathscr{A},\mathscr{B}) = \frac{1}{|\mathscr{A}|\cdot|\mathscr{B}|}\sum_{x\in\mathscr{A}}\sum_{y\in\mathscr{B}} d(x,y)$$

  con un comportamento intermedio tra single e complete

- Complete linkage: la distanza è la massima tra tutte le coppie degli elementi

$$d(\mathscr{A},\mathscr{B}) = \max\{\, d(x,y) : x\in\mathscr{A}, y\in\mathscr{B}\}$$

  i cluster tendono a formarsi di dimensioni simili e ben distinti tra di loro, per questo motivo è il criterio da noi scelto

Dopo l'ultimo passaggio tutti gli elementi si trovano in un unico cluster, che contiene un albero di sottocluster – rappresentabili con un dendrogramma, in cui l'altezza di ogni ramo è pari alla distanza tra le foglie. Partendo dall'alto e tagliando all'altezza minima possibile si possono ottenere p cluster dall'insieme iniziale (figura 3.3.1), con un algoritmo completamente generico e dipendente solo dalla matrice delle distanze.

Gli algoritmi gerarchici cercano una soluzione approssimata del problema del clustering in maniera *greedy* e hanno lo svantaggio che gli elementi una volta associati in un cluster non sono più spostati – un'associazione subottimale in partenza si propaga senza correzioni. Da qui la necessità del confronto con un altro metodo per avere la conferma che i cluster visti non sono stati introdotti dalla scelta del linkage completo. Il vantaggio della greediness è computazionale, con un andamento in tempo $\mathscr{O}(N^2)$ [**? ?**] e l'utilizzo in memoria solo della matrice delle distanze. È possibile fare in un secondo decine di clusterizzazioni utilizzando i pacchetti inclusi in Matlab (vi sono reimplementazioni open altrettanto performanti in Python e R). In codice Matlab, dalla matrice `distanza`, con `p` clusters richiesti si ottiene il vettore `c` di etichette intere comprese tra 1 e p:

```
Z = linkage(squareform(distanza),'complete');
c = cluster(Z,'maxclust',p);
```

3.3.2 Clustering spettrale

Sotto questo nome cade un'ampia gamma di possibili metodi, basati tutti sugli autovettori di un laplaciano discreto opportunamente definito. Per un ottima review si veda [**?**]. Gli algoritmi si dividono grosso modo in due categorie:

- Riduzione dimensionale a partire dagli autovettori del laplaciano
- Taglio del grafo in sottopartizioni

Le sequenze che vogliamo clusterizzano formano i vertici di un grafo completo pesato G, in cui ogni sequenza è collegata ad altre con un peso determinato dalla similarità, che possiamo ottenere dalla matrice delle distanze tramite la funzione di similarità gaussiana

$$A(i,j) = \begin{cases} \exp\left(-\frac{d(i,j)}{2\sigma}\right) & \text{se } d(i,j) \neq 0 \\ 0 & \text{se } d(i,j) = 0 \end{cases}$$

definita a diagonale nulla, con parametro σ che la distanza tra vicini. Per ogni vertice i indichiamo il grado, fornito dalla somma dei pesi con tutti gli altri vertici

$$d_i = \sum_j A_{ij}$$

e la matrice diagonale ottenuta dai gradi così definiti con D.

Poiché il grafo è pesato con pesi simmetrici positivi $A(i,j) = A(j,i) \geq 0$, è completo e non diretto, possiamo definire il laplaciano nonnormalizzato come

$$L = D - A$$

Il laplaciano ha molte proprietà:

- L è simmetrico e definito positivo
- su ogni vettore $f \in \mathbb{R}^n$ agisce come una forma quadratica

$$^t f\, L f = \sum_{i,j=1}^{n} A_{ij} \left(f_i - f_j\right)^2$$

- l'autovalore più piccolo è 0, con autovettore costante $(1,1,\ldots,1)$
- la molteplicità dell'autovalore 0 è il numero di componenti connesse, ogni autovettore è l'equivalente discreto della funzione caratteristica sulle componenti connesse, con entrate 1 corrispondenti ai vertici che comprende e 0 altrove
- L ha autovalori reali nonnulli, $0 \leq \lambda_1 \leq \lambda_2 \leq \cdots \leq \lambda_n$

Possiamo anche definite un laplaciano normalizzato come

$$L_{rw} := I - D^{-1}A$$

dove il pedice "rw" è dovuto alla similarità della definizione in problemi di random walk. Tutte le proprietà elencate per L valgono anche per L_{rw} invariate, sono solo riscalati gli autovalori. Indichiamo con "i primi p autovettori" gli autovettori corrispondenti agli autovalori *più piccoli* non nulli, ovvero scartando gli autovettori indicatori delle componenti connesse.

3.3.2.1 Riduzione dimensionale

L'algoritmo per il clustering di n elementi in p clusters, si basa sulla costruzione di L o L_{rw} ed è il seguente[**?**]:

1. In input la Matrice di similarità $A(i,j)$ e il numero p di cluster da costruire
2. Costruiamo la matrice dei gradi D, e il laplaciano $L_{rw} = I - D^{-1}A$

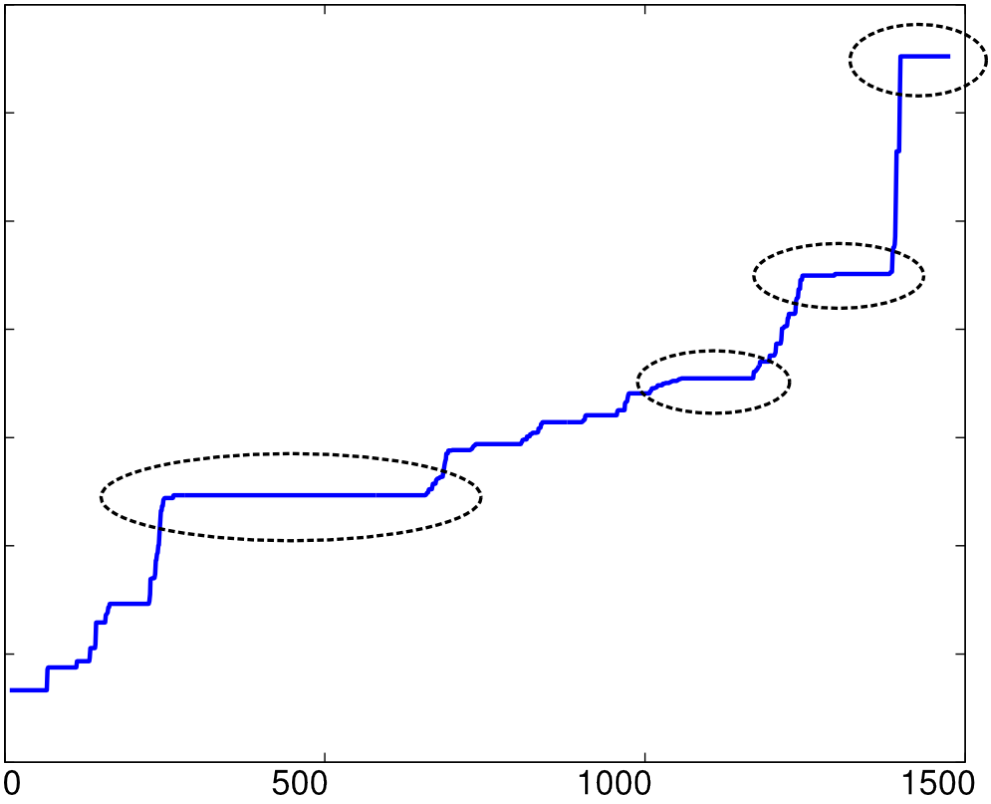

Figura 3.3.2: Andamento del secondo autovettore del laplaciano normalizzato. Il vettore è stato ordinato per valore, in modo che valori simili si trovassero vicini – ciò rende evidente la struttura a gradini, che grossomodo corrispondono ai cluster cercati, evidenziata in alcuni casi più evidenti.

3. Calcolo dei primi p autovettori generalizzati $u_1, \ldots, u_p$ del problema agli autovalori $L_{rw}\, u = \lambda\, u$

4. Sia $U \in \mathbb{R}^{n \times p}$ la matrice contenente i vettori $u_1, \ldots, u_p$ come colonne.

5. Per ogni $i = 1, \ldots, n$, sia $y_i \in \mathbb{R}^p$ il vettore corrispondente alla riga i-esima di U, ovvero ad ogni punto dell'insieme di partenza è stato fatto co

6. Troviamo ora i cluster $C_1, \ldots, C_p$ tramite l'algoritmo *kmeans* sui punti y di $\mathbb{R}^p$

Al punto 5 avviene la cosa più interessante, ad ogni elemento dell'insieme iniziale, caratterizzato solo dalle sue relazioni di distanza e similarità con gli altri elementi, è stato associato un punto nello spazio $\mathbb{R}^p$ – la riduzione dimensionale permette di utilizzare, se il numero p è piccolo, kmeans con successo. Abbiamo trovato una rappresentazione funzionale per il clustering.
In figura 3.3.2 illustriamo il risultato su un solo autovettore – sono già evidenti alcuni cluster. Gli autovalori invece danno una indicazione di quanto sono connessi tra di loro i cluster: primi autovalori piccoli, prossimi allo zero, indicano un cluster quasi sconnesso; un grande *gap spettrale* indica invece una migliore coesione dei cluster (figura 3.3.3). La conta degli autovalori sotto un certo valore dà un'idea del numero di cluster ottimale.

3.3.3 Taglio del grafo

Si può procedere diversamente per partizionare un grafo in cluster: si può eseguire un taglio per dividere in due pezzi separati il grafo; si itera il procedimento di taglio su ogni sottografo così creato fino a raggiungere il numero desiderato di cluster. Bisogna cercare

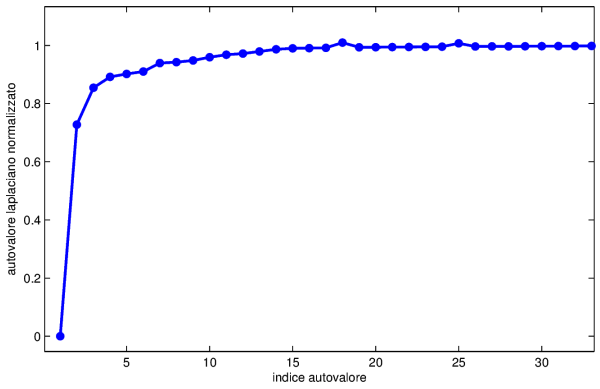

Figura 3.3.3: Primi autovalori del laplaciano normalizzato. Vi è un evidente gap tra l'autovalore zero e il primo nonnullo che significa una difficile separazione dei dati in cluster. Gli autovalori successivi sono molto vicini tra di loro, anche se si potrebbe riconoscere ad occhio che i primi 5 sono separati dai successivi, suggerendo un numero di cluster almeno pari a 5.

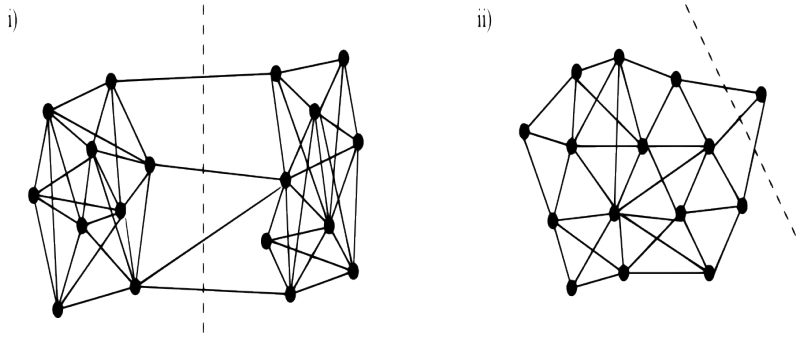

Figura 3.3.4: Criteri per il taglio del grafo. i) taglio "ottimale" ii) taglio che conta solo i legami rimossi

un criterio per ottenere un taglio ottimale – selezionare la componente con meno legami con il resto non è sufficiente, in quanto il più delle volte si sconnettono punti isolati, come si vede in figura 3.3.4.

Introduciamo un paio di definizioni: dati due sottoinsiemi di vertici A e B, introduciamo il peso dei legami relativi

$$W(A,B) = \sum_{i \in A,\, j \in B} A(i,j)$$

Per ogni sottoinsieme del grafo A_k, con complemento $\bar{A}_k$, possiamo definire la cardinalità ed il volume

$$|A| = \text{numero dei vertici in } A$$
$$\text{vol}(A) = \sum_{i \in A} d_i$$

che pesano rispettivamente il numero di vertici e l'importanza per il resto del grafo, misurata in termini di peso dei link.

Indicando con $A_1, \ldots, A_p$ una possibile partizione del grafo il più semplice criterio di taglio è quello che trova direttamente la partizione $A_1, \ldots, A_p$ che minimizza il funzionale

$$\text{cut}\left(A_1, \ldots, A_p\right) = \frac{1}{2} \sum_k W\left(A_k, \bar{A}_k\right)$$

ovvero riduce al minimo i legami tra le parti tagliate e il complementare. Tuttavia come si è visto in figura 3.3.4 porta ad un risultato non soddisfacente – bisogna i tagli normalizzare con il volume dei cluster formati, per evitare la formazione di cluster isolati. Ottimizziamo quindi il funzionale *normalized-cut*, o *ncut*:

$$\text{ncut}\left(A_1,\ldots,A_p\right)=\frac{1}{2}\sum_k \frac{W\left(A_k,\bar{A}_k\right)}{\text{vol}(A_k)}=\sum_k \frac{\text{cut}\left(A_k,\bar{A}_k\right)}{\text{vol}(A_k)}$$

che rappresenta il giusto compromesso, normalizzando ogni cluster formato con il peso dell'insieme dei legami del cluster. Il criterio più popolare in letteratura è il cosiddetto *Rcut*, in cui al posto del volume, si normalizza con la cardinalità dei cluster formati

$$\text{Rcut}\left(A_1,\ldots,A_p\right)=\frac{1}{2}\sum_k \frac{W\left(A_k,\bar{A}_k\right)}{|A_k|}$$

ma non è adatto ai nostri scopi, in quanto tende a selezionare cluster in cui il numero dei vertici (delle sequenze per noi) è quanto più simile – sappiamo invece che abbiamo molti dati recenti e pochi dell'inizio anni 90, per cui cluster "vecchi" avranno una cardinalità molte volte inferiore ai nuovi.

La soluzione esatta è nuovamente NP-completa e si cerca uno schema di approssimazione – si scopre[**?**][pag. 12 **?**] che il modo migliore di approssimare le p partizioni è dato dall'utilizzo dell'algoritmo descritto in 3.3.2.1 a pagina 35. Tuttavia l'utilizzo di kmeans può portare in pratica a dei problemi per p grandi, come la mancata convergenza di kmeans, scelte povere dei centroidi iniziali, ecc.

In [**?**] si dimostra come l'applicazione ricorsiva, utilizzando il secondo autovalore del laplaciano per bipartire il grafo, è ottimale e ha *worst-case* garantito – la bipartizione del grafo è una clusterizzazione in una dimensione, caso semplice che non ha patologie. Chiaramente ad ogni iterazione bisogna costruire il laplaciano e ricalcolare il valore dell'autovettore su ogni sottografo considerato, un processo che può sembrare lungo, ma nel caso di grafi molto sparse è altamente ottimizzato[**?**] e rappresenta lo stato dell'arte per il clustering di preferenze commerciali (Amazon, ebay, ecc).

Per il calcolo utilizziamo il software[3] creato per [**?**], che ottimizza un diverso funzionale, che tende nuovamente a equilibrare la cardinalità dei cluster, tramite una generalizzazione nonlineare del laplaciano

$${}^t f\, L_k\, f=\sum_{i,j=1}^{n} A_{ij}\left|f_i-f_j\right|^k$$

Il software è comunque in grado di utilizzare il più semplice criterio di *ncut* quando k=2 e partizionare ricorsivamente il nostro insieme in maniera estremamente efficiente, generando anche tutti i clusters da intermedi 1 a p.

3.3.4 Numero ottimale di clusters

In tutti gli algoritmi di clustering esposti, è necessario stabilire *a priori* il numero dei cluster cercati. Questo presenta naturalmente uno svantaggio, visto che vogliamo scoprire la vera struttura dei cluster e non imporla. Cerchiamo quindi un criterio che codifichi l'aspettazione naïve nei confronti di un buon algoritmo, ovvero che i cluster "non siano né troppo grossi, né troppo spezzettati".

[3] www.ml.uni-saarland.de/code/pSpectralClustering/pSpectralClustering.html