Introduzione alla teoria della percolazione

Leonardo Ongari *Università degli studi di Parma*
Cremona, Italia
leonardo.ongari@studenti.unipr.it

***Sommario*—La teoria della percolazione si occupa di sviluppare modelli matematici per descrivere l'omonimo fenomeno fisico, caratterizzato dallo scorrimento di un *fluido* all'interno di un *mezzo*. In questa relazione vengono mostrate le caratteristiche principali dei modelli, le nozioni teoriche associate ed alcune analisi sugli algoritmi utilizzabili per effettuare simulazioni consistenti.**

***Keywords*—Percolazione, Modelli, Simulazioni**

I. Introduzione

La teoria della percolazione nasce con l'obiettivo di ottenere un modello matematico per descrivere il fenomeno fisico associato. Questo fenomeno descrive lo scorrimento di un fluido all'interno di un materiale tipicamente poroso. È importante specificare che il significato del termine "percolazione" può riferirsi a diversi contesti, a seconda del campo in cui si sta operando. Alcuni esempi sono:

- un soluto che diffonde attraverso un solvente;
- elettroni che migrano attraverso un reticolo atomico;
- molecole che penetrano un solido poroso;
- una malattia che infetta una comunità.

La formalizzazione matematica del problema ha reso possibile la creazione di un *modello*. Un punto chiave molto importante di un modello è il concetto di *astrazione*, caratteristica che permette di operare senza considerare alcune variabili dipendenti dal contesto [1].

II. Background

Un metodo efficace per l'astrazione del concetto consiste in una rappresentazione tramite un reticolo, in inglese "lattice".

Definizione 1 (Reticolo)**.** Un reticolo L di dimensione n è una coppia $\langle S, B\rangle$, dove:

- $S = \{s_i : i \in \mathbb{N},\, 0 \leq i < n\}$ è l'insieme dei siti;
- $B = \big\{(s_i, s_j) \in S^2 : s_i \text{ e } s_j \text{ siano primi vicini}\big\}$ è l'insieme dei legami.

Due siti possono essere considerati primi vicini se, secondo l'ordinamento locale dei siti, hanno distanza uguale a 1. In un reticolo in cui vale questa relazione è possibile descrivere due tipi di percolazione: percolazione di sito e percolazione di legame.

Percolazione di legame

È la prima versione del modello fornita da Broadbent e Hammersley [1]. Ogni legame ha probabilità p di essere "aperto", quindi probabilità $1-p$ di essere "chiuso". Se due siti formano un legame aperto, vi è una connessione diretta tra i due. Al contrario, un legame chiuso elimina la connessione. In questa versione, "avviene percolazione" se esiste un percorso (insieme di connessioni dirette) che attraversa l'intero reticolo. La percolazione può verificarsi sia in verticale (top-bottom) sia in orizzontale (left-right).

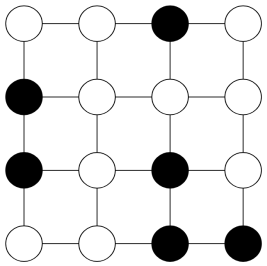

Figura 1: Esempio di reticolo quadrato bidimensionale

Percolazione di sito

In questo modello ogni sito ha una probabilità p di essere occupato, di conseguenza probabilità $1 - p$ di essere vuoto. In Fig. 1 viene mostrato un reticolo bidimensionale quadrato con nodi occupati (neri) e vuoti (bianchi). Questo è il modello che verrà utilizzato per lo studio dell'argomento e dei vari algoritmi e verrà approfondito in dettaglio nella Sez. III.

Soglia di percolazione

I due modelli appena introdotti rappresentano soluzioni valide per lo studio del fenomeno. Nonostante la somiglianza, vi sono differenze concettuali che si riflettono anche nel calcolo di alcuni valori caratteristici [2], [3], [4].

Definizione 2 (Soglia di percolazione)**.** Sia L un reticolo di dimensione infinita[1]. Sia p la probabilità di occupazione di un sito o di apertura di un legame, a seconda del modello scelto. Sia p uguale per ogni elemento del reticolo. La soglia di percolazione per L è definita come la probabilità p_c tale per cui:

- se $p > p_c$, allora vi è percolazione;
- se $p < p_c$, allora non vi è percolazione.

È possibile visualizzare il concetto di soglia nel grafico mostrato in Fig. 2, in cui l'asse delle ascisse è associato alla probabilità p, mentre l'asse delle ordinate è associato alla probabilità che avvenga percolazione p_{perc}.

[1]Con il termine "infinito" si fa riferimento all'estensione intuitiva delle varie proprietà della struttura, come avviene in matematica per il concetto di *limite all'infinito*.

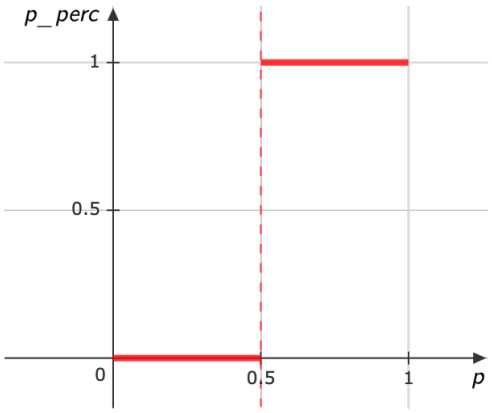

Figura 2: Grafico relativo alla soglia di percolazione ($p_c = 0.5$).

Nei casi reali, ovvero per reticoli di taglia finita, non è possibile stabilire un'affermazione così forte. Ciò che è possibile preservare dalla definizione è che esiste un **punto critico** p_c relativo alla probabilità p, oltre al quale è più probabile che avvenga percolazione e, al contrario, al di sotto del quale è meno probabile che questo si verifichi.

Lattice	p_c **(Site)**	p_c **(Bond)**
Cubic (body-centered)	0.246	0.1803
Cubic (face-centered)	0.198	0.119
Cubic (simple)	0.3116	0.2488
Diamond	0.43	0.388
Honeycomb	0.6962	0.65271*
4-Hypercubic	0.197	0.1601
5-Hypercubic	0.141	0.1182
6-Hypercubic	0.107	0.0942
7-Hypercubic	0.089	0.0787
Square	0.592746	0.50000*
Triangular	0.50000*	0.34729*

Tabella I: Punti critici (p_c) per reticoli regolari.

In letteratura sono presenti diversi studi sulle caratteristiche di vari reticoli e i rispettivi valori di soglia. La Tab. I mostra i punti critici per diversi reticoli regolari, ovvero composti da elementi ripetuti della stessa forma. La colonna **Lattice** indica la forma del reticolo, mentre le colonne **Site** e **Bond** distinguono i valori per percolazione di sito e di legame, rispettivamente [5]. Vi è una lieve, ma evidente, discrepanza tra i valori nelle due colonne. In generale, la rappresentazione tramite occupazione dei siti è considerata più generica rispetto alla sua controparte, questo perché la percolazione di legame può essere riformulata in termini di percolazione di sito, ma non si può affermare il contrario. I valori affiancati da un asterisco possono essere trovati tramite calcoli analitici, grazie ad alcune caratteristiche della forma del reticolo, sono quindi considerati *conosciuti*. È interessante notare come la tabella sia composta per lo più da valori non conosciuti, cioè valori ottenuti da simulazioni al calcolatore.

Cluster-finding

Nel modello che opera sui siti, la percolazione viene rilevata in seguito a un processo di *cluster-finding*, che consiste nel partizionare il reticolo in diverse classi, tramite una relazione di equivalenza.

Definizione 3. Una relazione di equivalenza R su un insieme A è una relazione binaria che gode delle seguenti proprietà:

- $\forall x \in A : xRx$ (riflessività);
- $\forall x, y \in A : xRy \rightarrow yRx$ (simmetria);
- $\forall x, y, z \in A : xRy \wedge yRz \rightarrow xRz$ (transitività).

La relazione utilizzata è strettamente collegata al concetto di primi vicini. La definizione di primi vicini per un sito può variare a seconda della tipologia (forma e dimensioni) del reticolo utilizzato.

Distribuzione binomiale

Per formalizzare in modo completo il modello, è necessario introdurre la nozione di *variabile aleatoria*. Non verrà trattato l'argomento nel dettaglio, per lasciare più spazio alle implementazioni. In questo contesto, si introduce il concetto di variabile aleatoria come una variabile il cui valore dipende da un evento non deterministico. Questo evento è descritto attraverso funzioni di densità specifiche, che seguono leggi di *distribuzione* della probabilità [6]. Esistono diverse tipologie di distribuzioni, tra cui quella binomiale che, riassumendo, descrive un esperimento di prove ripetute. Questo tipo di distribuzione è caratterizzato da una funzione di densità a 2 parametri

$$\mathcal{B}_{n,p}(k) = \binom{n}{k} p^k (1-p)^{n-k} \tag{1}$$

dove n rappresenta il numero di prove e p la probabilità di successo di ciascuna. La variabile k indica il risultato di cui vogliamo verificare la probabilità. L'Eq. 1 rappresenta *"la probabilità che si ottengano k successi ripetendo n volte una prova con probabilità di successo p"*.

A. Distribuzione Gaussiana

La distribuzione normale o gaussiana è una delle distribuzioni di probabilità più importanti in statistica e probabilità. La sua funzione di densità probabilistica (PDF) è data da:

$$f(x) = \frac{1}{\sigma\sqrt{2\pi}} e^{-\frac{(x-\mu)^2}{2\sigma^2}}, \quad x \in \mathbb{R}, \tag{2}$$

dove μ rappresenta la media e σ^2 la varianza. Questa distribuzione emerge naturalmente in numerosi contesti scientifici e tecnici grazie al *Teorema del Limite Centrale*.

B. Legge dei Grandi Numeri in forma di Chebyshev

La *Legge dei Grandi Numeri* afferma che la media aritmetica di una sequenza di variabili aleatorie indipendenti e identicamente distribuite (iid) converge alla loro media attesa quando il numero di osservazioni tende all'infinito. Una

formulazione basata sulla disuguaglianza di Chebyshev è la seguente:

$$P\left(\left|\frac{1}{n}\sum_{i=1}^{n} X_i - \mathbb{E}[X]\right| \geq \epsilon\right) \leq \frac{\text{Var}(X)}{n\epsilon^2}, \quad \forall \epsilon > 0, \qquad (3)$$

dove X_i sono variabili aleatorie iid con media finita $\mathbb{E}[X]$ e varianza finita $\text{Var}(X)$. Questo implica che all'aumentare di n, la probabilità che la media campionaria si discosti dalla media attesa oltre una certa soglia si riduce.

C. Teorema del Limite Centrale

Uno dei risultati fondamentali della teoria della probabilità è il *Teorema del Limite Centrale* (TLC), il quale afferma che, date $X_1, X_2, \ldots, X_n$ variabili aleatorie indipendenti e identicamente distribuite con media μ e varianza finita σ^2, la variabile normalizzata:

$$Z_n = \frac{\sum_{i=1}^{n} X_i - n\mu}{\sigma\sqrt{n}} \qquad (4)$$

converge in distribuzione a una variabile casuale normale standard $\mathcal{N}(0,1)$ per $n \to \infty$:

$$Z_n \sim \mathcal{N}(0,1) \qquad (5)$$

Questo risultato giustifica l'uso della distribuzione normale in molte applicazioni pratiche, in particolare nella statistica inferenziale e nella teoria dell'approssimazione.

III. Implementazione

Prima soluzione

In questa relazione verranno discusse proprietà e risultati dell'algoritmo Hoshen-Kopelman [7]. Tuttavia, per dimostrarne la correttezza e provarne l'efficienza, verranno effettuati dei confronti con un altro algoritmo di base, che chiameremo algoritmo A, la cui correttezza è ben nota data la sua semplicità. L'Alg. 1 mostra una prima soluzione al problema del partizionamento ed è composto da due iterazioni principali: una "esterna" per tutti i siti occupati, una "interna" per i siti adiacenti. Quest'ultima si appoggia ad un vettore che funge da *pila*, le cui dimensioni variano in maniera dinamica. Lo scopo della pila è quello di aggiungere, ad ogni iterazione interna, i siti occupati adiacenti al nodo corrente, per essere poi processati singolarmente. Ragionando sul funzionamento del codice, ci si convince piuttosto facilmente della ridondanza causata dalla pila. In altri termini, un sito occupato può essere processato più volte:

- una e una sola volta nell'iterazione esterna;
- una volta per ogni vicino occupato.

Algoritmo 1 Pseudocodice dell'algoritmo A

```
largest_label ← 1
label ← zeros[n_columns, n_rows]
for [x, y] ← [0, 0] to n_columns, n_rows do
    if [x, y] = 1 ∧ label[x, y] = 0 then
        pila ← [x, y]
        label[x, y] ← largest_label
        j ← 1
        while j ≤ length(pila) do
            elem ← pila[j]
            x ← elem[1], y ← elem[2]
            left ← [x − 1, y]
            right ← [x + 1, y]
            down ← [x, y + 1]
            top ← [x, y − 1]
            if left = 1 ∧ label[left] = 0 then
                append left to pila
                label[left] ← largest_label
            end if
            if right = 1 ∧ label[right] = 0 then
                append right to pila
                label[right] ← largest_label
            end if
            if up = 1 ∧ label[up] = 0 then
                append up to pila
                label[up] ← largest_label
            end if
            if down = 1 ∧ label[down] = 0 then
                append down to pila
                label[down] ← largest_label
            end if
            j ← j + 1
        end while
        largest_label ← largest_label + 1
    end if
end for
```

Paradigma Union-Find

Come appena accennato, in questa relazione viene presentato l'algoritmo Hoshen-Kopelman, che fa parte della famiglia di algoritmi *union-find*, utilizzati nel calcolo di componenti connesse e, in questo caso, per il processo di cluster-finding. Questo paradigma prevede l'utilizzo di due procedure principali, che ne compongono il nome, in determinati punti del codice:

- *Union*: effettua una fusione tra due insiemi disgiunti quando ci si accorge che appartengono in realtà alla stessa classe di equivalenza;
- *Find*: determina a quale insieme appartiene l'elemento in elaborazione.

Nell'Alg. 2 vengono mostrati i passaggi per l'implementazione del paradigma nel contesto della percolazione [8]. Rispetto all'algoritmo iniziale, rimane un'iterazione sui siti per verificarne l'occupazione, ma appare diversa la logica applicata. In particolare, scompare l'utilizzo della pila con la conseguente ridondanza e, per ogni sito occupato, vengono considerati solo due primi vicini: sopra ($above$) e a sinistra ($left$). Se nessuno dei due vicini è occupato, al sito viene assegnata una nuova etichetta ($label$), aggiornata tramite un contatore; se soltato uno dei due vicini è occupato, il sito eredita la sua etichetta; se entrambi i vicini sono occupati e hanno la stessa etichetta, il comportamento è analogo al caso di

Algoritmo 2 Pseudocodice dell'algoritmo Hoshen-Kopelman

```
largest_label ← 0
label ← zeros[n_columns, n_rows]
Lofl ← [0 : n_columns]

for [x, y] ← [0, 0] to [n_columns, n_rows do
    if [x, y] = 1 then
        left ← label[x − 1, y]
        above ← label[x, y − 1]
        if (left = 0) ∧ (above = 0) then
            largest_label ← largest_label + 1
            tmp ← largest_label
        else if (left = 1) ∧ (above = 0) then
            tmp ← left
        else if (left = 0) ∧ (above = 1) then
            tmp ← above
        else
            tmp ← min{left, above}
            bad_label ← max{left, above}
            UNION(bad_label, tmp)
        end if
        label[x, y] ← tmp
    end if
end for

procedure UNION(x, y)
    Lofl[FIND(x)] ← FIND(y)
end procedure

function FIND(x)
    while Lofl[x] ≠ x do
        x ← Lofl[x]
    end while
    return x
end function
```

un vicino occupato; se invece i vicini hanno etichette diverse, il sito eredita quella minore, prestando attenzione al fatto che in realtà le due facciano riferimento allo stesso cluster. Infatti, in questo procedimento si possono avere siti di uno stesso cluster con etichette diverse (provare per credere!). Quest'ultimo punto è cruciale: è necessario tenere conto di questa informazione senza impattare negativamente sul costo computazionale dell'algoritmo; una rinomina totale delle etichette sarebbe infatti troppo costosa e si perderebbe il vantaggio ottenuto rispetto ad A. Per coprire questo aspetto, per ogni cluster dovrà esserci una etichetta "di rappresentanza", che chiameremo *good label*. Ogni altra etichetta viene considerata una *bad label* e deve essere in qualche modo collegata alla rispettiva *good label*. Il metodo di collegamento scelto consiste nell'utilizzo di un array $Lofl$ (Label of label), che viene aggiornato durante l'esecuzione. A questo proposito, entrano in gioco le nuove procedure:

- *Union*: ha il compito di collegare due etichette[2] in modo che facciano riferimento alla stessa *good label* e viene descritta facilmente in termini di *Find*;
- *Find*: viene implementata come iterazione di punto fisso della funzione rappresentata dall'array $Lofl$. In altri termini, si cerca in modo iterativo la condizione di arresto $Lofl[x] = x$.

[2]Vanno collegate le *good label* attuali delle etichette dei siti $above$ e $left$. Infatti, questa procedura viene svolta solo quando entrambi i siti sono occupati ed hanno diverse etichettature.

Stime della soglia ed errori associati

Per stabilire in maniera formale se esista o meno un cluster percolante (ricordiamo: un cluster percolante attraversa l'intero reticolo), va fatto un confronto tra le etichette presenti sui bordi del reticolo: se esiste almeno un'etichetta presente su due bordi opposti, significa che il cluster identificato da essa attraversa il reticolo in quella direzione. Nell'algoritmo A è sufficiente confrontare le etichette, mentre per Hoshen-Kopelman occorre confrontare le *good label* dei siti presenti sui bordi. Il processo consiste nel verificare che vi sia una intersezione non nulla tra le etichette presenti sul bordo sinistro e quelle sul bordo destro (left-right) oppure, in maniera analoga, tra il bordo in alto e quello in basso (top-bottom). Un paragone tra le rilevazioni dei due algoritmi mostrerà come i risultati ottenuti nei due casi siano perfettamente coincidenti. Per generare valori significativi ed effettuare confronti tra gli algoritmi, è stato necessario definire valori dei parametri richiesti nella simulazione e iterare su di essi. Si è definito un array $L = [20, 60, 40, 80]$ per le dimensioni. Si è costruito un array p per le probabilità di occupazione, tramite punti non equidistanti:

1) $p_a = 0.025 : 0.025 : 0.5$;
2) $p_b = 0.5125 : 0.0125 : 0.7$;
3) $p_c = 0.725 : 0.025 : 1$;
4) $p = [p_a\ p_b\ p_c]$.

La sintassi utilizzata richiama volontariamente quella del linguaggio Matlab: la scritta $inf : inc : sup$ indica una serie di valori che parte da inf e arriva a sup, tramite un incremento costante inc. Si è scelto infine un numero arbitrario N di iterazioni per determinare, fissando i parametri precedenti, la frequenza di percolazione, calcolata banalmente dividendo n_{perc} (numero di esperimenti in cui avviene percolazione) per N, in questo caso uguale a 1000.

$$\forall i \in [1..N] : \begin{cases} t(i) = 1, & \text{se avviene percolazione} \\ t(i) = 0, & \text{altrimenti.} \end{cases} \tag{6}$$

Per ogni test $i \in [1..N]$ vi sono quindi 2 casi: avviene o non avviene percolazione. I risultati dei test sono memorizzati all'interno di un array t tramite valori di verità. Si denota con $t(i) \in \{0, 1\}$ il risultato dell'i-esimo test a parametri fissati. Grazie alla notazione introdotta, si nota facilmente che:

$$\sum_{i=1}^{N} t(i) = n_{perc} \tag{7}$$

Seguendo questi passaggi, si può valutare $t(i)$ come se fosse una misurazione di una variabile aleatoria T. Si può pensare che T segua una legge di distribuzione binomiale della forma

$$T \sim \mathcal{B}_{n,p}(k)$$

Ma che tipo di binomiale? Si potrebbe attribuire $n = N$ e chiaramente siamo interessati a $k = n_{perc}$. Tuttavia, non si conoscono i valori esatti del parametro p_{perc} sulla probabilità di percolazione e occorre quindi procedere per *stime*. Il fatto che la frequenza di percolazione sia significativa rispetto alla probabilità della stessa è garantito, per input sufficientemente grandi, dalla Legge dei Grandi Numeri espressa in forma di Chebyshev [9]. L'affidabilità delle misurazioni effettuate

è invece discussa per mezzo del Teorema del Limite Centrale (TLC) [10]. Questi strumenti matematici permettono di affermare proprietà statistiche sulle stime del parametro p_{perc} e sull'errore associato. Formalmente, si calcola la media aritmetica sugli esperimenti come

$$f_{perc} = \frac{n_{perc}}{N} = \frac{1}{N}\sum_{i=1}^{N} t(i) \tag{8}$$

Sappiamo dall'Eq. 3 che, per N sufficientemente grande, f_p converge in probabilità al valore atteso di T. Inoltre, la varianza associata è

$$D_{f_{perc}} = \frac{1}{N^2}\sum_{i=1}^{N} D_T = \frac{D_T}{N} \tag{9}$$

dove D_T è la varianza della variabile aleatoria T e può essere stimata in Matlab seguendo le misurazioni del campione:

$$\tilde{D}_T = D_t = \frac{1}{N-1}\sum_{i}^{N} (t(i) - f_{perc})^2 \tag{10}$$

Ci si avvale a questo punto anche dell'Eq. 5 e ne consegue che

$$f_{perc} \sim \mathcal{N}(\sigma^2, \mu) = \frac{1}{\sqrt{2\pi\sigma^2}} e^{-\frac{(x-\mu)^2}{2\sigma^2}} \tag{11}$$

o, in altri termini, la frequenza di percolazione rilevata nella simulazione segue una legge di distribuzione gaussiana. Si può dire di più: i parametri della funzione sono proprio le stime effettuate sul campione. Questi passaggi risultano molto importanti perché permettono di ricondurre esperimenti "arbitrari" a casistiche ben note. Sostanzialmente, dato che conosciamo la natura della distribuzione gaussiana, ci è concesso affermare che valori lontani dalla media $\mu = f_{perc}$ per più di 3σ, dove $\sigma = \sqrt{\tilde{D}_T}$, possano essere esclusi dai candidati per il valore atteso di p_{perc}. Partendo dai campioni restituiti dalla simulazione, si è quindi arrivati a costruire un *intervallo di confidenza* all'interno del quale il valore ricercato ha probabilità $1-\alpha$ di apparire, dove α rappresenta l'area sottesa dalla funzione di densità f_{perc} negli intervalli $(-\infty, \mu-3\sigma)$ e $(\mu+3\sigma, +\infty)$. In Fig. 3 è mostrato un esempio grafico per capire meglio il concetto.

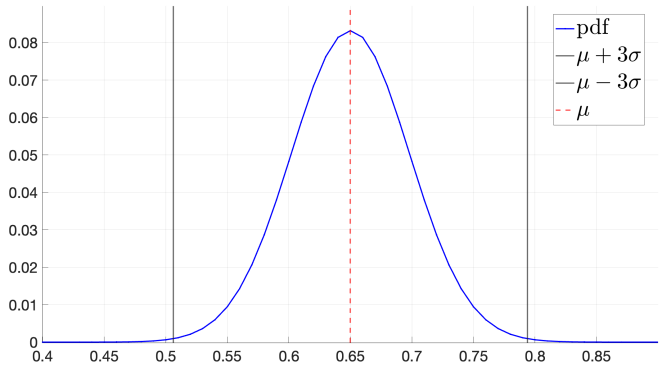

Figura 3: Funzione di densità probabilistica normale per f_{perc}.

Altre osservabili

Si può svolgere il procedimento di stima dei parametri di una distribuzione non conosciuta anche per altre quantità, dette "osservabili". Ognuna di queste, nel nostro caso, è riconducibile ad una funzione gaussiana per lo stesso motivo descritto nel paragrafo precedente. Anche la correttezza della stima si discute nel senso di convergenza a una distribuzione normale, come già mostrato. Di seguito è esposta una panoramica delle osservabili studiate, oltre alla frequenza di percolazione.

P_1: Rappresenta la probabilità che il cluster più grande trovato (dimensione s_{max}) sia grande quanto l'intero reticolo. È lecito pensare (e si avrà una conferma) che questa quantità sia trascurabile per valori piccoli della probabilità di occupazione p, e avrà andamento crescente fino a $p = 1$, dove sicuramente vi sarà un unico cluster di dimensione s_{max}.

$$P_1 = \frac{s_{max}}{|L|} \tag{12}$$

P_2: Rappresenta la probabilità che il cluster più grande trovato sia grande quanto il numero medio di siti occupati. Occorre fare una precisazione: i valori di questa quantità potrebbero essere superiori ad 1, poiché il denominatore è soltato una stima tramite valore medio (numero di siti moltiplicati per la probabilità di occupazione). Anche in questo caso, comunque, ci si aspetta un andamento simile al precedente.

$$P_2 = \frac{s_{max}}{p|L|} \tag{13}$$

P_3: Rappresenta la probabilità che il cluster più grande trovato sia grande quanto il numero esatto di siti occupati. In questo caso, rispetto a P_2, al denominatore è presente un valore calcolato analiticamente. Questa differenza elimina la presenza di valori anomali maggiori di 1, ma la quantità calcolata, soprattutto per un numero elevato di tentativi, rimane molto simile.

$$P_3 = \frac{s_{max}}{\sum_s s \cdot n_s} \tag{14}$$

$RACS$: Rappresenta la probabilità che un sito appartenga ad un cluster che non sia quello di dimensone massima. In questo caso, ci si aspetta un andamento "opposto" rispetto alle altre quantità: per probabilità alte di occupazione ci si aspetta di avere un cluster che sia molto più grande degli altri, di conseguenza la probabilità che un sito non vi appartenga si abbassa.

$$RACS = \frac{\sum_{s \neq s_{max}} s \cdot n_s}{\sum_t t \cdot n_t} \tag{15}$$

IV. Risultati

I risultati ottenuti tramite Matlab sono stati valutati in termini di coerenza con le nozioni teoriche esposte fino ad ora. In particolare, si discuterà a proposito di:

- confronto tra gli output dei due algoritmi descritti;
- quantità ottenute nel calcolo delle osservabili, con i rispettivi errori;
- confronto tra i tempi di esecuzione dei due algoritmi.

Correttezza dell'algoritmo HK

Come anticipato, la correttezza dell'algoritmo implementato è stata definita rispetto all'output dell'algoritmo A, che funge quindi da riferimento. Sono stati eseguiti molti test con vari parametri di input, ma ogni volta "fissando" il reticolo, per avere un confronto sulla stessa struttura. In Fig. 4 viene mostrato un confronto dei valori ottenuti tramite i due algoritmi, variando sia la dimensione del reticolo (ogni linea rappresenta le esecuzioni a dimensione fissata), sia la probabilità di occupazione. Quest'ultima è considerata un parametro di input per una funzione $f(x)$ ed è quindi riscontrabile sull'asse delle ascisse. Risulta interessante

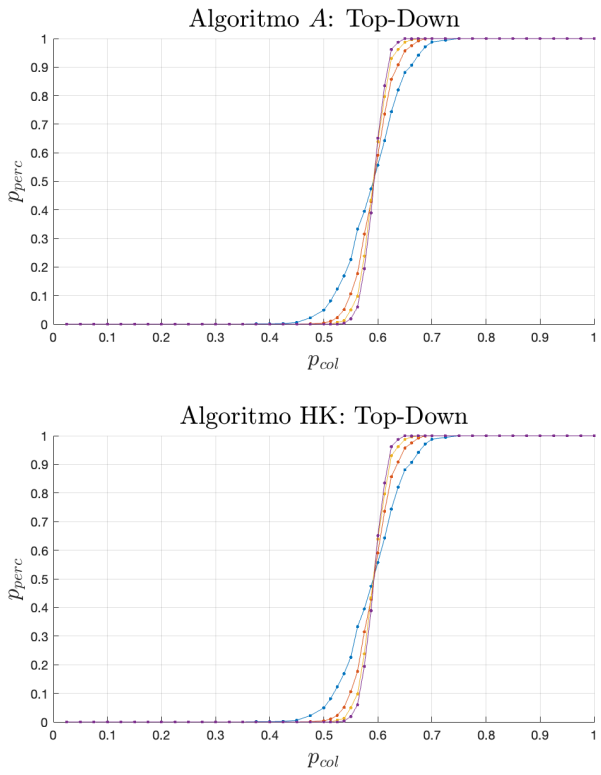

Figura 4: Confronto tra le frequenze di percolazione top-bottom ottenute nell'esecuzione dei due algoritmi.

eseguire un confronto con la Fig. 2 relativa al reticolo di taglia infinita: per dimensioni del reticolo più grandi, ci si avvicina infatti al suddetto grafico. Un'altra caratteristica importante da verificare è che le frequenze di percolazione top-bottom registrate coincidano con quelle left-right, nel limite dell'errore commesso. Le modalità di svolgimento sono le stesse, con l'aggiunta delle misurazioni dell'errore, calcolato come radice quadrata della quantità $D_{f_{perc}}$ descritta nell'Eq. 9. Nel grafico in Fig. 5 viene mostrato un confronto tra i valori ottenuti nei due casi con l'algoritmo HK. Ciò che balza all'occhio è la superiorità degli errori intorno a 0.6, valore attorno al quale si osserva una crescita piuttosto rapida della frequenza di percolazione. Si compone di tre iterazioni: una per variare la taglia del reticolo, una per variare la probabilità di occupazione e un'ultima per eseguire un numero sufficiente di esperimenti affinché i risultati siano significativi. Le frequenze dei due algoritmi sono calcolate con metodi diversi ma equivalenti: nel primo caso si "conta" quante volte la variabile caratteristica ha assunto valore 1 e si divide il totale per il numero di esperimenti N; nel secondo caso si utilizza la funzione built-in di Matlab `mean(x)`, che svolge gli stessi passaggi. Il Cod. 1 mostra le istruzioni in Matlab per calcolare le quantità mostrate nei due grafici. Si è omessa l'implementazione in codice Matlab delle funzioni `CercaCluster3` e `CercaClusterHK`, del tutto analoghe all'implementazione mostrata tramite pseudocodice negli Alg. 1 e 2.

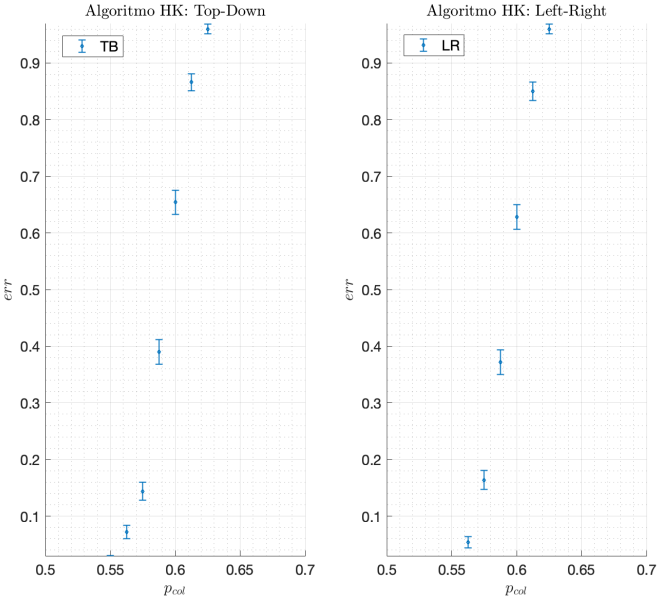

Figura 5: Confronto tra le frequenze di percolazione top-bottom e left-right, con rispettivi errori.

```
for ij = 1:length(L)
  for ii = 1:length(p)
    pp = p(ii);
    s3TB = 0;
    s3LR = 0;
    sHKTB = 0;
    sHKLR = 0;
    espTB = zeros(N,1);
    espLR = zeros(N,1);
    for j = 1:N
      ret = rand(L(ij))<pp;
      res3 = CercaCluster3(ret);
      resHK = CercaClusterHK(ret);
      s3TB = s3TB + res3.percolazioneTB;
      s3LR = s3LR + res3.percolazioneLR;
      espTB(j) = resHK.percolazioneTB;
      espLR(j) = resHK.percolazioneLR;
    end
    probPercTB3(ij,ii) = s3TB / N;
    probPercLR3(ij,ii) = s3LR / N;
    probPercTBHK(ij,ii) = mean(espTB);
    probPercLRHK(ij,ii) = mean(espLR);
    erroreTB(ij,ii) = std(espTB) / sqrt(N);
    erroreLR(ij,ii) = std(espLR) / sqrt(N);
  end
end
```

Codice 1: Porzione di codice relativa al confronto tra algoritmi e alla frequenza di percolazione top-bottom e left-right.

Calcolo delle osservabili

Lo stesso procedimento necessario per calcolare la frequenza di percolazione è stato svolto per le altre osservabili. In Fig. 6 si possono riscontrare i comportamenti attesi, descritti nella sezione precedente. Anche in questo caso, viene acclusa l'implementazione in Matlab nel Cod. 2. Insieme alle quantità, vengono calcolati anche i rispettivi errori, con tecniche analoghe a quanto visto fin ora. Per queste quantità valgono infatti le stesse considerazioni fatte sulla natura di f_{perc}, legate alla validità della media e varianza garantita per mezzo di TLC e Legge dei Grandi Numeri.

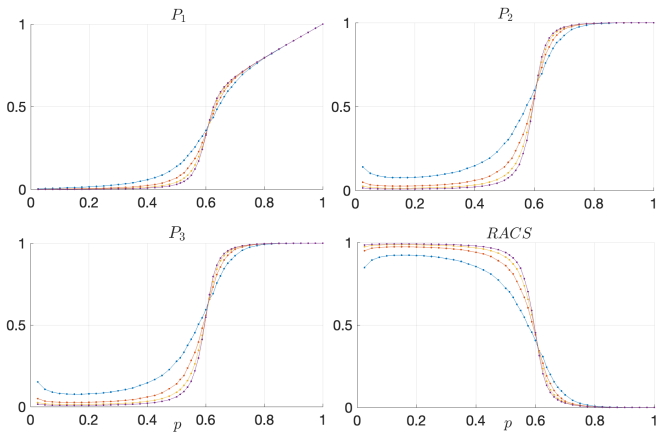

Figura 6: Distribuzioni delle osservabili calcolate in $N = 1000$ esperimenti al variare dei parametri.

```
for ij = 1:length(L)
  for ii = 1:length(p)
    pp = p(ii);
    for j = 1:N
      ret = rand(L(ij))<pp;
      resHK = CercaClusterHK(ret);

      MYsz(ij,j,ii) = mean(resHK.cluSz);
      MYmaxSz(ij,j,ii) = max(resHK.cluSz);
      MYnumCLU(ij,j,ii) = length(resHK.cluSz);
      MYnumCol(ij,j,ii) = sum(resHK.cluSz);
    end
  end

  sMax=squeeze(MYmaxSz(ij,:,:));
  P1=sMax./L(ij)^2;
  errore1=std(P1)./sqrt(length(P1));
  P1=mean(P1);

  P2=sMax./(p*L(ij)^2);
  errore2=std(P2)./sqrt(length(P2));
  P2=mean(P2);

  occupied=squeeze(MYnumCol(ij,:,:));
  P3=sMax./occupied;
  P3=squeeze(P3);
  errore3=std(P3)./sqrt(size(P3,2));
  P3=mean(P3);

  occupiedReduced =
    squeeze(MYnumCol(ij,:,:)-MYmaxSz(ij,:,:));
  racs=occupiedReduced./occupied;
  erroreRacs=std(racs)/sqrt(length(racs));
  racs=mean(racs);
end
```

Codice 2: Porzione di codice per il calcolo delle osservabili.

Per completezza, si aggiunge anche il grafico in Fig. 7 che confronta i vari errori ottenuti nel calcolo di ciascuna. Occorre prestare attenzione alla scala sugli assi dei grafici: per questioni pratiche di visualizzazione, si è preferito avere risultati più chiari, ma su scale leggermente diverse.

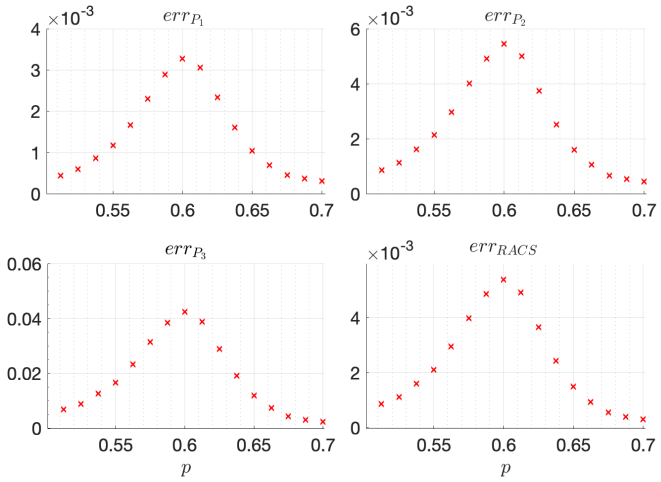

Figura 7: Errori ottenuti nel calcolo delle osservabili.

Tempi di esecuzione

Come ultimo risultato verranno discussi i tempi di esecuzione di ciascun algoritmo. Prima di farlo, però, occorre precisare che Matlab non è l'ambiente di sviluppo migliore per valutare differenze di costo computazionale, a causa dei seguenti svantaggi (almeno per questo caso) che lo costituiscono:

- operazioni matriciali: Matlab opera su strutture dati che rappresentano matrici, non è quindi ottimizzato per esecuzioni di tipo iterativo;
- ricorsione: le strutture dati utilizzate per rappresentare il controllo di flusso non sempre sono adatte a sostenere lo spazio richiesto da chiamate ricorsive;
- "tic toc": è il metodo per misurare i tempi di esecuzionein Matlab; non ha un buon funzionamento in caso di quantità elevate di prove ripetute, soprattutto se molto vicine tra loro.

In fondo, Matlab rimane un linguaggio relativamente di alto livello rispetto ad alcune sue controparti come C++ per opencv. Esistono alcune funzionalità che permettono di generare codice C++ o per GPU, ma non verranno affrontate in questa relazione. Va comunque ricordato che Matlab ha molti altri pregi, tra cui la flessibilità per la creazione di modelli, caratteristica fondamentale per il progetto. Per tutte queste ragioni, i due algoritmi sono stati confrontati solo sulla parte relativa al cluster-finding. Il motivo dietro a questo è che nell'algoritmo Hoshen-Kopelman sono necessarie chiamate ricorsive per capire se due siti appartengono allo stesso cluster, e questa proprietà va verificata per tutti i siti dei bordi, causando un *overhead*.

In Fig. 8 sono presenti 4 grafici che mostrano un confronto sugli andamenti dei tempi di esecuzione calcolati nelle due casistiche. Anche in questo caso, la statistica viene fatta

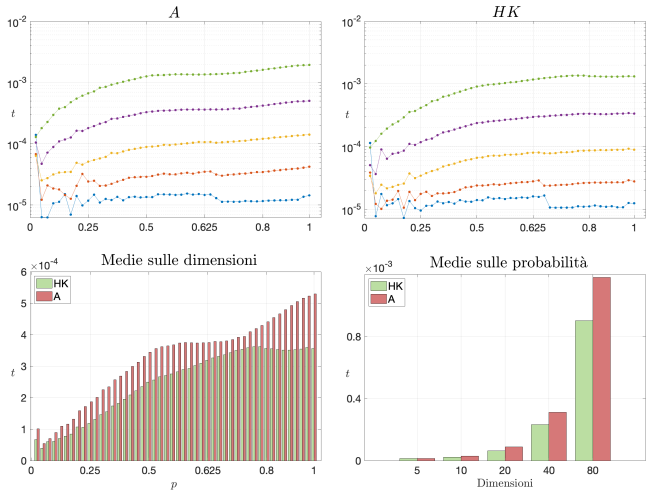

Figura 8: Confronto dei tempi di esecuzione dei due algoritmi.

fissando i parametri (dimensione e probabilità di occupazione), per poi effettuare un numero N arbitrariamente grande di esperimenti. Viene quindi eseguito prima N volte l'algoritmo A, registrando il tempo impiegato e dividendolo per N (si ottiene quindi una media per l'esecuzione singola). Lo stesso viene fatto per l'algoritmo HK. Ciò che si ottiene sono due matrici $|P| \times |D|$, dove $D = [5\ 10\ 20\ 30\ 40]$ è l'array delle dimensioni e P è l'array delle probabilità di occupazione.

I primi due grafici rappresentano proprio le due matrici ottenute, dove gli assi y sono in scala logaritmica per avere un confronto visivo migliore. I colori indicano le diverse dimensioni dei reticoli negli esperimenti. Anche se di poco, in generale si notano tempi minori nell'algoritmo HK. Nel terzo grafico viene fatta una media sulle dimensioni, in modo che rimangano solo due serie di dati da confrontare. In pratica, si può pensare alle quantità mostrate nel terzo grafico come "medie" di quelle mostrate nei primi due. Nell'ultimo caso, invece, la media viene fatta sulle probabilità e la lunghezza delle due serie di dati da confrontare è di 5 elementi, come il numero elementi nell'array delle dimensioni. Anche in questo caso, le tempistiche di HK sono minori rispetto ad A, come si aveva ragione di credere.

V. Conclusioni

In conclusione, il progetto ha permesso di analizzare il fenomeno della percolazione attraverso un approccio frequentista, utilizzando due diversi algoritmi: uno standard A e l'algoritmo di Hoshen-Kopelman. L'analisi statistica sui risultati ottenuti ha fornito una visione chiara dei meccanismi di percolazione e delle differenze di efficienza tra i due metodi implementati. Un naturale sviluppo futuro potrebbe essere lo studio del fenomeno su reticoli di forme diverse o in dimensioni superiori, come il caso tridimensionale, per esplorare eventuali variazioni nei comportamenti osservati. Inoltre, un'interessante prospettiva di miglioramento sarebbe l'integrazione del codice Matlab con C++ tramite gli strumenti dedicati, permettendo una maggiore ottimizzazione delle strutture dati e un incremento dellefficienza computazionale.

Riferimenti bibliografici

[1] S. R. Broadbent and J. M. Hammersley, "Percolation processes: I. crystals and mazes," *Mathematical Proceedings of the Cambridge Philosophical Society*, vol. 53, no. 3, pp. 629-641, 1957.

[2] C. Stover and E. W. Weisstein, "Bond percolation." [Online]. Available: https://mathworld.wolfram.com/BondPercolation.html

[3] ——, "Site percolation." [Online]. Available: https://mathworld.wolfram.com/SitePercolation.html

[4] ——, "Percolation threshold," *from Wolfram MathWorld*, 2002. [Online]. Available: https://mathworld.wolfram.com/

[5] D. Stauffer and A. Aharony, *Introduction to percolation theory*. Taylor & Francis, 2018.

[6] F. Edition, A. Papoulis, and S. U. Pillai, *Probability, random variables, and stochastic processes*. McGraw-Hill Europe: New York, NY, USA, 2002.

[7] J. Hoshen and R. Kopelman, "Percolation and cluster distribution. i. cluster multiple labeling technique and critical concentration algorithm," *Phys. Rev. B*, vol. 14, pp. 3438–3445, Oct 1976. [Online]. Available: https://link.aps.org/doi/10.1103/PhysRevB.14.3438

[8] T. Fricke, "The hoshen-kopelman algorithm," *URL: http://www. ocf. berkeley. edu/~ fricke/projects/hoshenkopelman/hoshenkopelman. html*, 2004.

[9] W. Feller, *An introduction to probability theory and its applications. 2*. Wiley, 1957.

[10] B. V. Gnedenko and A. N. Kolmogorov, *Limit distributions for sums of independent random variables*. Addison-wesley, 1968, vol. 2420.