SIR: Simulazione della diffusione di un virus attraverso un automa cellulare

Università di Bologna-Programmazione per la fisica, anno 2019/20

Nicolò Catalani, Dimitri Corradini, Lorenzo Tassotti

30 agosto 2020

ALMA MATER STUDIORUM
UNIVERSITÀ DI BOLOGNA

Introduzione

Il progetto SIR da noi realizzato si basa sul concetto di automa cellulare e sui modelli compartimentali introdotti nel secolo scorso per la descrizione della diffusione epidemica[1]. Un automa cellulare, di cui è un celebre esempio il *Game of life* del matematico inglese John Conway[2], è un tipo di modello matematico utilizzato per la descrizione di sistemi complessi, in cui viene definita una griglia costituita da celle in grado di *evolversi* e cambiare il loro stato secondo un preciso algoritmo. Nel nostro caso, tale algoritmo comporta la transizione delle celle, che rappresentano i membri della popolazione interessata, tra *suscettibili, infetti, guariti, morti*. Tale suddivisione è mutuata, con qualche differenza, da quella del cosiddetto Modello SIR, un sistema di equazioni differenziali in uso correntemente per la descrizione delle epidemie.

1 Istruzioni per la compilazione e l'esecuzione del programma

1.1 Compilazione

Il programma è suddiviso in numerosi file sorgente, motivo per cui si raccomanda l'uso di CMake. Da una finestra terminale sulla cartella SIR è possibile digitare in successione

```
mkdir build
cd build
cmake ..
make
```

1.2 Esecuzione

Dopo aver atteso il completamento, basta digitare

```
./SIR
```

per eseguire il programma. Da terminale viene immediatamente richiesta la dimensione da dare alla propria griglia e viene data la possibilità di inserire parametri di infettività, guarigione e letalità. Se si decide di procedere saltando quest'ultimo step, vengono inseriti valori standard. Si apre dunque una finestra apposita che, dopo aver premuto il tasto S, mostra le istruzioni del programma. Premendo il tasto Q sullo schermo viene visualizzata una griglia. Le celle che la costituiscono sono identificabili attraverso il colore: esse sono blu nel caso di individui non ancora contagiati (suscettibili), rosse nel caso di individui infetti, verdi nel caso dei guariti e infine nere con un piccolo teschio (!!) nel caso dei morti. Inizialmente, tutte le celle sono suscettibili; cliccando sopra a una cella, è possibile infettarla o renderla nuovamente suscettibile. Dopo aver inserito gli infetti, premendo S è possibile far partire la simulazione. Una volta che la pandemia si è fermata, ovvero quando non vi sono più infetti, premendo Q si accede a una finestra in cui è stampato un grafico dell'andamento del contagio, con relativa legenda.

1.3 Testing

Digitando

```
./run_points_tests
```

è possibile eseguire test sulle funzioni definite nel file *points.hpp*. Digitando invece

```
./run_board_tests
```

è possibile eseguire test su funzioni e operatori definiti nel file *board.hpp*.

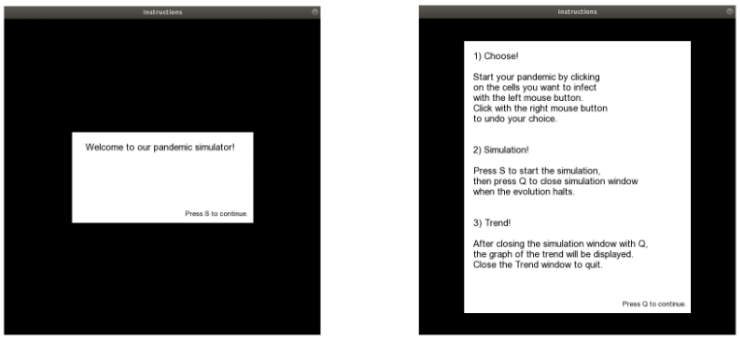

Figura 1: screenshot delle finestre di istruzioni

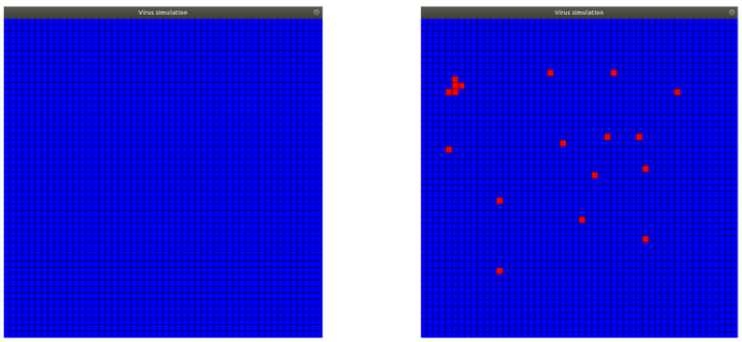

Figura 2: screenshot dell'inserimento degli infetti

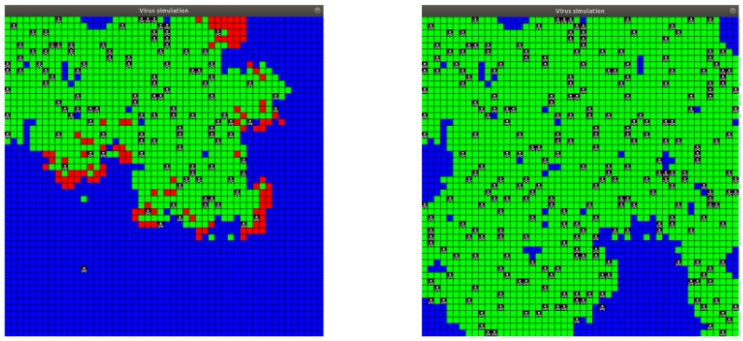

Figura 3: screenshot dell'evoluzione della griglia

2 Funzionamento, scelte implementative e progettuali

2.1 Struttura logica

Il programma si basa sulla definizione di una *enum class* chiamata `State` e contenente i quattro stati definiti nel nostro modello (*susceptible, infected, recovered, deceased*). Viene successivamente definita una classe chiamata `Board`, che rappresenta la griglia in cui sono posizionate le celle. Le sue variabili membro sono un intero (numero di celle per ogni lato della griglia) e un oggetto del tipo `std::vector<State>`. Il suo costruttore prende come argomento un intero `side` compreso tra 1 e 50, inizializza la dimensione a tale numero e il vettore con $\texttt{side}^2$ elementi, inizialmente aventi stato *susceptible*. L'accesso agli elementi della board è permesso da due operatori `()` definiti in seguito con un overload, che prendono come argomento due interi (numero della riga e della colonna all'interno della griglia): uno di essi è di sola lettura, e contiene una condizione necessaria al trattamento delle celle che si trovano al di fuori dei confini della griglia; l'altro invece permette la modifica degli elementi del vettore. L'evoluzione della griglia avviene attraverso l'apposita funzione `evolve`, che prendendo come argomento una `Board` e tre `double` (beta, gamma, mu, compresi tra 0 e 1) ne restituisce un'altra modificata. L'algoritmo al suo interno si basa sulla generazione di numeri pseudo-casuali attraverso un Mersenne Twister, che vengono confrontati successivamente con i tre parametri argomento della funzione. Viene svolto un ciclo sugli elementi della griglia; se un elemento è infetto, viene generato in maniera pseudo-aleatoria un numero compreso tra 0 e 1, che è confrontato con il parametro gamma (probabilità di guarigione). Se il numero generato risulta strettamente minore del parametro, lo stato della cella è cambiato in *recovered*, altrimenti i medesimi passaggi sono ripetuti per il parametro mu (probabilità di decesso tra chi non guarisce), cambiando conseguentemente lo stato in *deceased*. Se nessuna di queste due circostanze si verifica, la cella rimane infetta, e si procede alla simulazione dell'effettivo contagio. Viene dunque operato un ulteriore ciclo sulle celle a contatto con l'infetto. Se qualcuna di esse è suscettibile, viene generato un numero casuale, poi confrontato con il parametro beta (probabilità di infezione). Similmente a prima, se il numero generato è strettamente minore di beta, la cella viene resa *infected*. L'operatore `()` permette di non considerare le celle fuori dalla griglia, che vengono riconosciute come *recovered* e non sono quindi soggette ad alcun cambiamento. Uno `switch` statement inserito all'inizio del programma permette la selezione dei parametri di evoluzione e della dimensione della griglia in base all'inserimento da terminale da parte dell'utente.

2.2 Implementazione grafica

Per la visualizzazione dei risultati abbiamo utilizzato la libreria grafica open source *SFML* (acronimo per *Simple and Fast Multimedia Library*). Abbiamo definito una classe `Info` per generare la finestra, i rettangoli e il testo necessari all'esposizione delle istruzioni. Attraverso un'altra classe denominata `Display` abbiamo invece implementato la visualizzazione e la modifica attraverso click dal mouse della board.

Il metodo `draw` al suo interno genera infatti un'interfaccia grafica attraverso il posizionamento e la diversa colorazione di quadrati all'interno della finestra (le celle che costituiscono la griglia). Il metodo `click` permette l'interazione con la griglia attraverso il click, controllando, nel caso venga premuto un tasto del mouse, la posizione del mouse nella finestra rispetto alle celle e modificando lo stato in *infected* se si preme il tasto sinistro su una di esse, riportandolo invece a *susceptible* se viene premuto nuovamente. Infine, la griglia viene disegnata con le modifiche apportate. Una terza classe `Graph`, insieme a una *struct* denominata `Points`, permette invece la visualizzazione del grafico di andamento finale. `Points` ha come membri quattro interi; attraverso la funzione `count`, ad essi è associato il numero di celle che si trovano in ogni possibile `State` all'interno della board. Essi sono successivamente convertiti in pixel dalla funzione `convert`, e vengono inseriti in un vettore attraverso il quale, dopo che si è conclusa l'evoluzione del contagio, è possibile disegnare il grafico. Ciò avviene mediante il metodo `draw` di `Graph`, che viene chiamato alla fine del `main`, in modo da poter scalare la lunghezza del grafico in funzione dei *giorni* trascorsi (numero di cicli di evoluzione). Il metodo `draw` genera assi, legenda e curve, costituite da linee spezzate. In ognuna delle classi che fanno uso della libreria grafica sono poi inseriti ulteriori metodi di apertura o chiusura per schermate e finestre.

2.3 Strategie di testing

Abbiamo utilizzato per il testing il framework Doctest. Nel testare il programma ci siamo concentrati sulla struttura logica che rende successivamente possibile l'elaborazione grafica. Abbiamo pertanto creato unità di test per la struct `Points` e la classe `Board` , in particolare verificando il corretto funzionamento delle funzioni `count` e `convert` nel primo caso e delle eccezioni, degli operatori e di `evolve` nel secondo. Ci siamo focalizzati su casi estremi e particolari, in modo da poter identificare eventuali bug non evidenti nella normale esecuzione del programma. Nel caso della funzione `evolve` abbiamo testato attraverso un oggetto `Points` il funzionamento in alcuni particolari casi in cui l'output corretto era facilmente determinabile a priori.

3 Design e divisione in unità

Il programma è stato suddiviso in numerosi header contenenti ciascuno la definizione di una classe, le dichiarazioni dei metodi al suo interno e la definizione di costruttore e operatori. Accanto alla classe `Board` sono incluse `State` ed `evolve`, essendo strettamente collegate ad essa nel loro utilizzo. La *struct* `Points` è definita in un header a parte con le due *free functions* `count` e `convert`, per motivi simili. Nei corrispettivi file `.cpp` sono invece contenute le definizioni di tutte le funzioni che sono state dichiarate nell'header. Si è preferito evitare l'utilizzo di `inline`, eccetto per i costruttori e gli operatori all'interno delle classi (in cui è implicito), data la lunghezza e complessità dei metodi da noi definiti. Abbiamo racchiuso tutte le funzioni e le classi da noi definite nel `namespace SIR`.

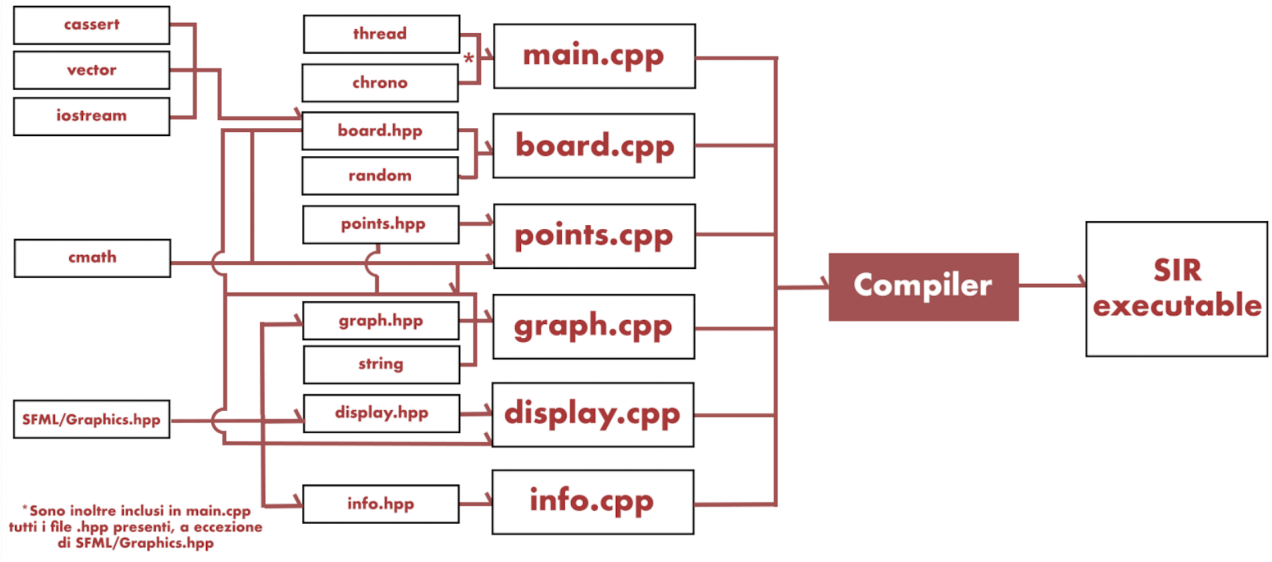

Figura 4: Schema del physical design del progetto

4 Interpretazione dei risultati

Da un confronto del grafico prodotto dal programma con quello generato da un Modello SIR (o SIRD, per la precisione), è possibile verificare una somiglianza nella forma delle curve, e conseguentemente nell'andamento di contagi, guarigioni e decessi.

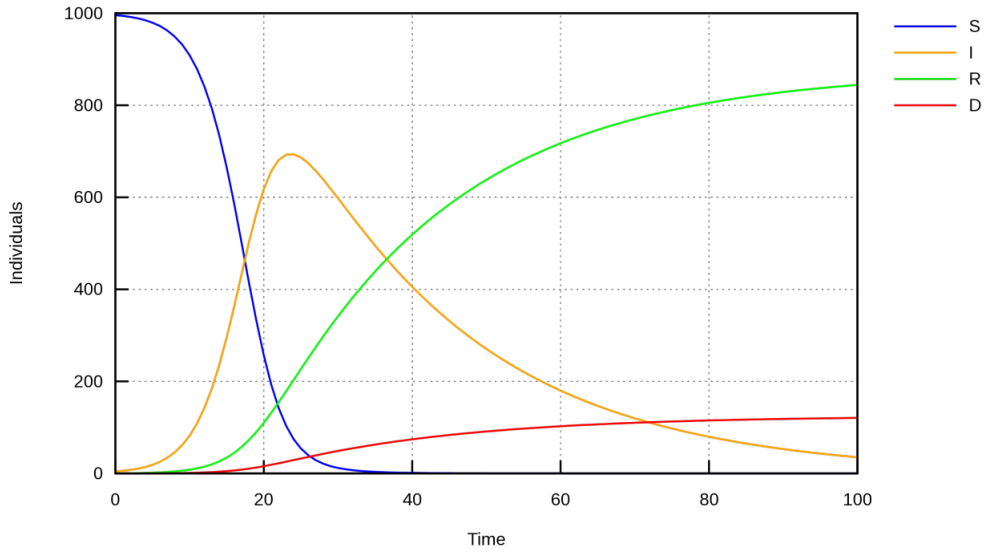

Figura 5: grafico dell'andamento di un'epidemia ottenuto attraverso il Modello SIRD

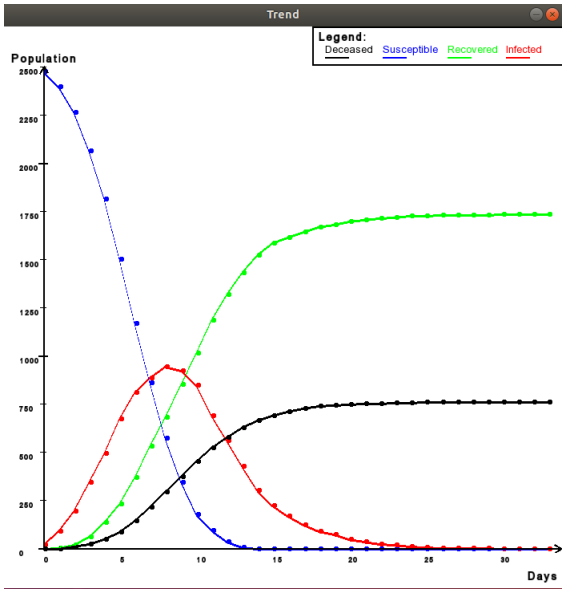

Figura 6: grafico generato dal nostro programma

Riferimenti bibliografici

[1] Kermack, W. O.; McKendrick, A. G. (1927), *A Contribution to the Mathematical Theory of Epidemics*. Proceedings of the Royal Society A. 115 (772): 700–721.

[2] Gardner, M. (1970), *Mathematical Games - The Fantastic Combinations of John Conway's New Solitaire Game 'Life'*, Scientific American (223): 120–123.