Programmazione per la fisica
Relazione progetto finale

Di Masi Francesca
Martini Giulia
Righetti Lucia

29/11/2022

Link Repository GitHub: https://github.com/giuliamartini/progetto_sird.git

Indice

1 Introduzione

Il progetto qui presentato consiste nell'implementazione del modello epidemiologico SIRD (Susceptible, Infected, Recovered, Dead) in linguaggio di programmazione C++. Il programma esegue una simulazione di una pandemia su automa cellulare e restituisce un grafico dell'andamento del contagio. Una volta inseriti i dati di una possibile epidemia quali susceptibles iniziali, tasso di contagiosità β, l'infettività (raggio d'azione dell'infezione), tasso di guarigione γ, il programma è in grado di fornire una simulazione grafica dell'epidemia e la sua evoluzione in termini numerici. Durante la simulazione si è tenuta in considerazione una possibile campagna vaccinale che comincia dopo 20 giorni dall'inizio del contagio. Dopo 40 giorni i Recovered tornano ad essere Susceptible. La simulazione grafica dell'evento epidemiologico tramite automa cellulare è una griglia di dimensioni definite dall'utente che simula i contagi dovuti al contatto tra Susceptible e Infected ed evidenzia la velocità di trasmissione dell'epidemia. Come condizione al contorno si considera una "cornice" di Person inizializzate ad Empty di larghezza pari al parametro infectivity; nella relazione ci si riferisce al bordo della griglia come all'inizio di questa cornice esterna. É possibile vedere gli effetti della campagna vaccinale in termini di protezione della popolazione e rallentamento dei contagi.
In seguito viene proposto un confronto con l'implementazione del modello SIR di equazioni differenziali al fine di verificare la consistenza dei risultati ottenuti.

2 Implementazione del modello SIRD tramite automa cellulare

L'implementazione del modello consiste in tre file:

- un header file chiamato "sird.hpp"
- un file chiamato "sird.cpp"
- un file chiamato "main.cpp" che include il main e le funzioni utili per la stampa dei risultati su terminale

Nell'header file viene definita una struct chiamata "State": l'epidemia nella sua evoluzione viene gestita per giorni, ogni oggetto di classe State quindi memorizza il numero di Susceptible, Infected, Recovered, Dead e Vaccinated per un dato giorno tramite la funzione counts, rappresenta così la situazione giornaliera dell'epidemia. Gli individui sono rappresentati dall'enum class "Person" (Empty, Susceptible, Infected, Recovered, Dead, Vaccinated). La classe "Population" è l'oggetto tramite il quale si valuta la simulazione: contiene un vettore di vettori di elementi di tipo Person inizializzati a Empty di default. La simulazione comincia con il piazzamento di oggetti di tipo Person, inizializzati a Empty di default, sulla board di tipo Population tramite il metodo pubblico placePeople(), che cambia lo stato di un numero scelto dall'utente di oggetti di tipo Person in Susceptible. Viene collocato un Infected al centro della board tramite il metodo pubblico setInfected(). La simulazione procede tramite il metodo pubblico evolve() che valuta la situazione della board del giorno successivo secondo le probabilità di infettarsi β, di guarire γ, di morire nel caso in cui non si guarisca δ, e che un individuo si vaccini ϵ. Viene simulato lo spostamento dei Person tramite il metodo pubblico move(), che, con probabilità uniforme di fare uno spostamento in una delle nelle quattro direzioni, consente a un Person non Empty di spostarsi in un posto adiacente della griglia. I metodi della classe Population sono dichiarati in "sir.hpp" e definiti in "sir.cpp".
Nel file "main.cpp " viene definita una funzione che permette di visualizzare su terminale l'automa cellulare. La simulazione termina quando non ci sono più Infected sulla board. Al termine della simulazione vegono inoltre stampati su terminale il numero di S, I, R, D e V per ogni giorno e viene aggiornato un file "graph.csv" con tali dati. Da main.cpp viene in seguito eseguito il programma Python "graph.py" che fornisce i grafici dei dati di "graph.csv".

2.1 Compilazione ed esecuzione del programma

La compilazione viene effettuata con il compilatore g++ con le seguenti opzioni per rilevare problemi nel codice:

```
g++ -Wall -Wextra -g -fsanitize=address
```

Ad essere compilati saranno i file “sird.cpp” e “main.cpp”

```
g++ -Wall -Wextra -g -fsanitize=address sird.cpp main.cpp
```

Se la compilazione va a buon fine, quindi senza alcun tipo di errore, warning o memory leak si può procedere con l’esecuzione del programma. All’utente si richiede di inserire 6 parametri da standard input:

```
user@user:~/plague$ ./a.out

Please enter the value of beta

0.8

    ...
```

Verranno chiesti i valori di beta, gamma, delta, epsilon, infectivity (un intero che rappresenta la distanza entro cui c’è interazione tra Infected e Susceptible o Vaccinated), la dimensione della board e il numero di susceptibles iniziale. Questi parametri devono rispettare alcuni vincoli che il programma segnala in caso di errore. Il programma, con i seguenti valori di esecuzione d’esempio, fornisce gli output riportati sotto.

```
./a.out 0.8 0.3 0.05 0.7 3 4000 50
```

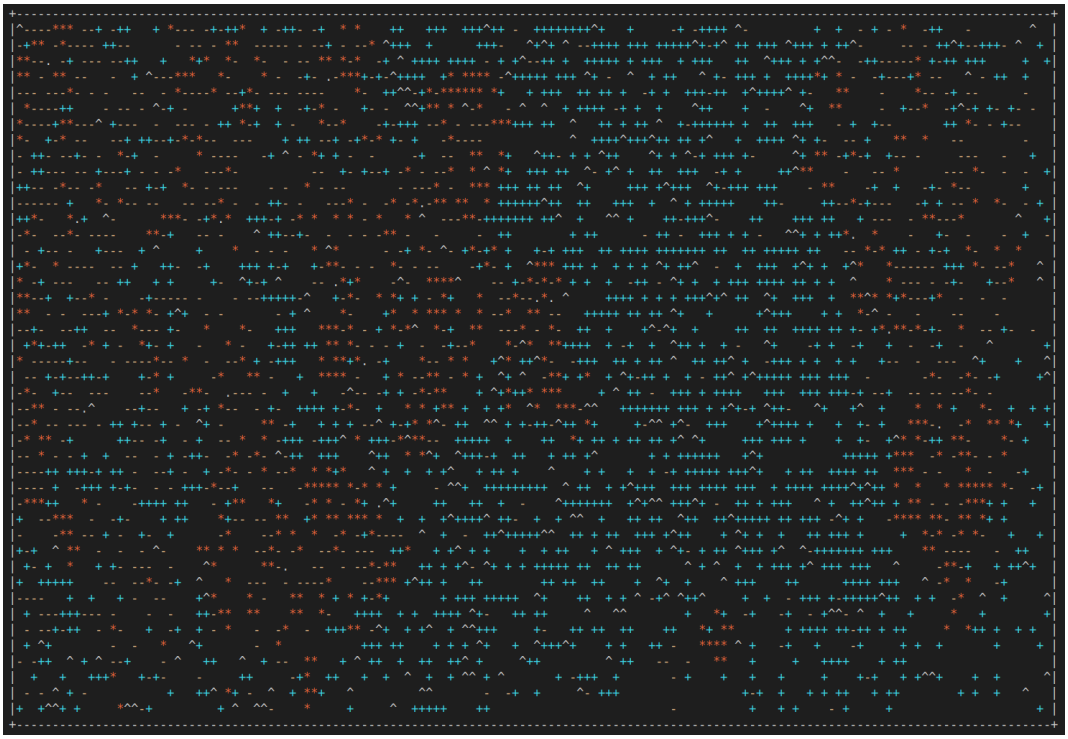


Figura 1: Frame esempio della simulazione

I caratteri ‘+’ blu sono i Susceptible, i ‘∗’ rossi sono gli Infected i ‘−’ arancioni i Recovered, i ‘∧’ verdi i Vaccinated, i ‘.’ bianchi i Dead. I Dead dopo un giorno diventano Empty. Il programma viene iterato finchè è presente almeno un Infected sulla board. La simulazione si interrompe quando si esauriscono gli Infected.

Day	Susceptible	Infected	Recovered	Dead	Vaccinated
0	3496	5	0	0	0
1	3437	63	0	1	0
2	3391	88	20	2	0
3	3292	154	50	5	0
4	3236	159	90	16	0
5	3198	154	126	23	0
6	3115	172	188	26	0
7	3052	174	243	32	0
8	3005	157	299	40	0
9	2936	184	338	43	0
10	2885	162	409	45	0
11	2855	138	457	51	0
12	2794	158	495	54	0
13	2730	177	537	57	0
14	2638	217	584	62	0
15	2577	206	647	71	0
16	2507	218	701	75	0
17	2402	246	770	83	0
18	2302	268	842	89	0
19	2222	250	930	99	0
20	2134	256	992	106	13

Figura 2: file csv con i dati ottenuti dal modello SIRD

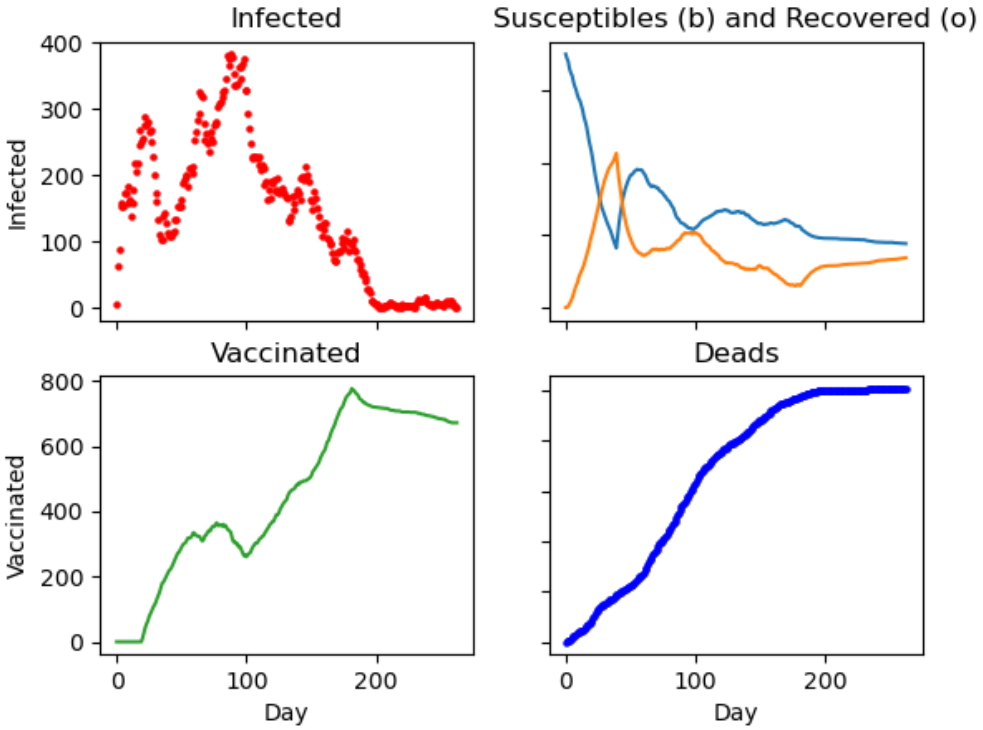


Figura 3: grafici dei dati ottenuti dal modello SIRD

Si ottengono degli output (Figura 2 - Figura 3) che dimostrano come l'epidemia si estingua dopo aver avuto in media due picchi. Si ottengono infatti due alti picchi di infetti in corrispondenza dell'inizio del contagio e dal giorno 40 in cui parte dei Recovered tornano ad essere Susceptibles. Si estingue il contagio in corrispondenza con l'aumento dei vaccini.

2.2 Metodi pubblici della classe Population

2.2.1 Metodi infection e interaction

Questi due metodi sono fondamentali per la gestione dei Person di tipo Infected. Il metodo interaction() serve per contare il numero di Infected attorno ad un Susceptible o attorno a un Vaccinated. La distanza dal Susceptible considerato è misurata tramite una metrica "a croce", realizzata tramite due cicli for, rispettivamente sulle righe e sulle colonne della griglia. Interaction restituisce un vettore di interi: l'intero alla posizione i-esima del vettore corrisponde al numero di infetti alla distanza (i+1)-esima dal Susceptible valutato. La distanza entro il quale ci può essere interazione fra Infetto e Susceptible è fornita dall'utente tramite il parametro infectivity.
Il metodo infection() prende come argomento due int : r e c che forniscono la posizione del "Person" indagato , il valore dell'infettività e il parametro beta. Il metodo Restituisce un bool: false se il Person si infetta e true se il Person non si infetta. Applica il metodo interaction() per contare gli infetti fino a distanza infectivity dal Person. Genera un numero reale casuale compreso tra 0.0 e 1.0 utilizzando uno strumento della library ⟨random⟩. Confronta beta/i con il numero random (dove i è la distanza i-esima dal Person, la probabilità di infettarsi rappresentata da $\frac{\beta}{i}$, infatti, decresce all'aumentare della distanza) . Nel caso in cui il numero random è minore di $\frac{\beta}{i}$ il Person sarà Infected il giorno successivo; in caso contrario rimarrà nello stato precedente. Il confronto con un numero random viene eseguito per ogni infetto nel raggio di azione definito da infectivity attorno al Person.

2.2.2 Metodi recovery, death, vaccination

Il metodo recovery() prende come argomento il parametro gamma, che rappresenta la probabilità di guarire, e lo confronta con un numero reale casuale compreso tra 0.0 e 1.0 restituendo un bool: true se l'Infected guarisce e diventerà il giorno successivo un Recovered, false se non succede.
I metodi death() e vaccination() sono costruiti in modo analogo a recovery(). Il primo confronta delta, probabilità di morire se non guariti, con un numero generato casualmente e restituisce un bool true se un Infected diventerà Dead e un bool false se non muore. Il secondo metodo invece prende come argomento il parametro epsilon, la probabilità di vaccinarsi, ritornando un bool true se Person si vaccina e un bool false se non si vaccina.

2.2.3 Metodo evolve

Il metodo evolve() si occupa dell'evoluzione della board il giorno successivo. Il metodo scorre la griglia tramite due cicli for sulle righe e sulle colonne e degli if statement che valutano l'evoluzione di ogni Person in posizione (r,c) nei diversi casi:

- Susceptible: chiama il metodo infection(), se il risultato è un bool true allora la persona sarà Infected il giorno successivo, altrimenti rimarrà Susceptible.
- Vaccinated: chiama nuovamente il metodo infection() che valuta se la persona diventerà Infected al giorno successivo o rimarrà Vaccinated, moltiplicando il parametro beta di infection con una costante fissata a priori che rappresenta l'efficacia del vaccino.
- Infected: chiama inizialmente il metodo recovery() e nel caso di bool true trasforma un Infected in Recovered. Se il risultato è un bool false viene chiamato il metodo death(). Se questo metodo ritorna un bool true allora Infected sarà Dead il giorno seguente, altrimenti rimarrà Infected.
- Recovered: in questo caso si considera che un guarito rimanga tale il giorno seguente.
- Dead: come per Recovered si considera che un morto rimanga tale il giorno seguente.

2.2.4 Metodo move

Il metodo move() simula lo spostamento delle persone nella griglia. Ogni Person non Empty può infatti, con probabilità uniforme, spostarsi in una delle quattro direzioni adiacenti purché queste siano Empty. Generando un numero intero casuale distrubito tra 0.0 e 4.0 si valutano tramite uno switch statement i 5 diversi case: Person può muoversi quindi su, giù, a sinistra, a destra o rimanere fermo. Nel caso in cui Person si trovi sul bordo della griglia viene "rimbalzato" su di esso e rimane nella sua posizione originale.

2.2.5 Metodi che massVaccination, susceptibleAgain

Il metodo massVaccination() viene chiamato dopo 20 giorni dall'inizio dell'evoluzione e permette ad una parte di Susceptible di vaccinarsi. Questo metodo prende come argomenti infectivity, un intero "toBeVaccinated" che indica il numero massimo di persone che possono essere vaccinate al giorno ed il parametro epsilon. Viene definito un vettore di Position su cui si effettua un push_back() delle posizioni in cui sono presenti i Susceptible. La funzione shuffle() disponibile nella library ⟨algorithm⟩ permette di riorganizzare gli elementi del range in modo casuale tramite un generatore di numeri casuali uniformi std::random_device. Si valuta dunque, con il metodo vaccination(), se la vaccinazione sia andata a buon fine o meno, modificando Person in Vaccinated in caso di esito positivo.
Il metodo susceptibleAgain() viene chiamato dopo 40 giorni dell'inizio e fa sì che parte dei Recovered ritornino Susceptible. Analogamente al caso di massVaccination() viene definito un vettore di Position su cui si effettua il push_back() delle posizioni dei Recovered. Con shuffle() si riorganizzano gli elementi del vettore e nelle k posizioni della griglia vengono posizionati dei nuovi Susceptible.

2.3 Test

Abbiamo effettuato alcuni test basilari per verificare il codice contentuto in "sird.cpp". Abbiamo controllato il corretto funzionamento dei metodi placePeople() e setInfected() posizionando uno o più infetti nella board. Successivamente abbiamo verificato che il numero totale della popolazione coinvolta nella pandemia rimanesse stabile durante l'evoluzione.

3 Implementazione del modello SIR e confronto con il modello precedente

In questa parte del progetto si propone uno dei modelli più semplici per descrivere lo sviluppo di una pandemia: il modello di equazioni differenziali SIR. Vengono inseriti da input il numero di suscettibili e infetti iniziali e il programma simula l'evoluzione dell'epidemia in termini numerici.

L'implementazione del modello consiste in tre file:

- un header file chiamato "sir_diff.hpp"
- un file chiamato "sir_diff.cpp"
- un file chiamato "main_diff.cpp" che include il main e le funzioni utili per la stampa dei risultati su terminale

Nell'header file viene definita una struct chiamata State: l'epidemia nella sua evoluzione viene gestita per giorni, ogni oggetto di classe State quindi memorizza il numero di Susceptibles, Infected, Recovered, i parametri gamma e beta e rappresenta la situazione giornaliera dell'epidemia. A partire da un oggetto State infatti si costruisce un oggetto della medesima classe che rappresenta il giorno successivo. Questo compito viene svolto dalla classe Epidemic: a partire da uno State iniziale definito dall'utente, tramite il metodo pubblico "evolve" calcola gli stati successivi secondo le equazioni definite dal modello. Il metodo evolve dichiarato in "sir_diff.hpp" è definito in "sir_diff.cpp"
Un'istanza Epidemic sulla quale è stato chiamato il metodo evolve è quindi un vettore di oggetti di tipo State i cui dati verranno stampati su terminale dalle funzioni definite nel file di tipo .cpp.
È infatti nel file "main_diff.cpp" che viene definita una funzione print che stampa i risultati della simulazione giorno per giorno. L' output grafico è una tabella di 4 colonne, una per ogni classe

di popolazione (S,I,R). Dal main_diff.cpp viene poi eseguito il programma Python graph_diff.py che stampa i grafici della simulazione.

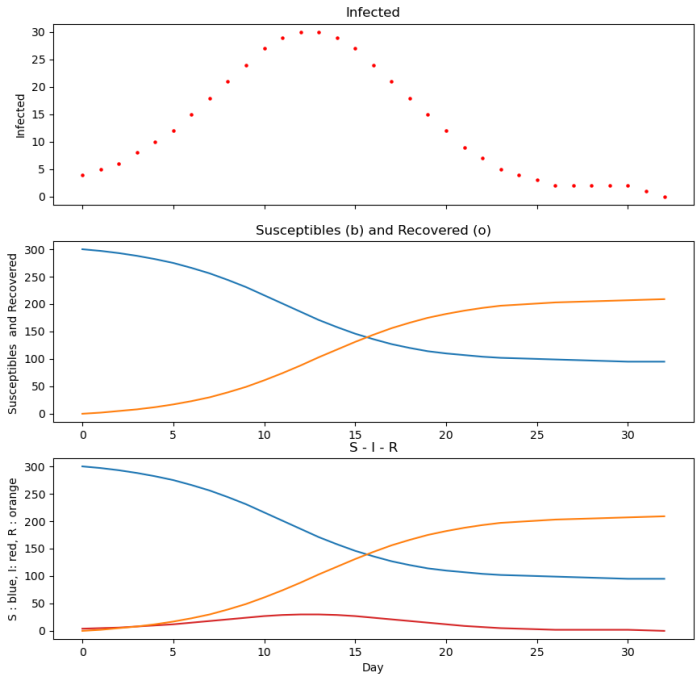


Figura 4: grafico ottenuto dal modello SIR

3.1 Compilazione ed esecuzione del programma

La compilazione viene effettuata con il compilatore g++ con le seguenti opzioni per rilevare problemi nel codice:

```
g++ -Wall -Wextra -g -fsanitize=address
```

Ad essere compilati saranno i file “sird.cpp” e “main.cpp”

```
g++ -Wall -Wextra -g -fsanitize=address sir_diff.cpp main_diff.cpp
```

Se la compilazione va a buon fine, quindi senza alcun tipo di errore, warning o memory leak si può procedere con l’esecuzione del programma. All’utente si richiede di inserire 5 parametri da standard input:

```
user@user:~/plague$ ./a.out

Please enter the value of beta

0.8

    ...
```

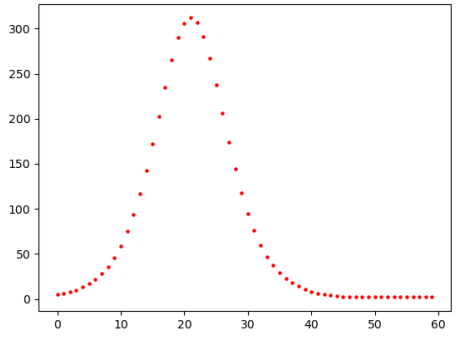
Verranno richiesti da standard input i valori di beta, gamma , il numero iniziale di Susceptible, il numero iniziale di Infected e il numero di giorni di durata della simulazione.

day	S	I	R
1	300	4	0
2	297	5	2
3	293	6	5
4	288	8	8
5	282	10	12
6	275	12	17
7	266	15	23
8	256	18	30
9	244	21	39
10	231	24	49
11	216	27	61
12	201	29	74
13	186	30	88
14	171	30	103
15	158	29	117
16	146	27	131
17	136	24	144
18	127	21	156
19	120	18	166
20	114	15	175
21	110	12	182
22	107	9	188
23	104	7	193
24	102	5	197
25	101	4	199
26	100	3	201
27	99	2	203
28	98	2	204
29	97	2	205
30	96	2	206
31	95	2	207
32	95	1	208
33	95	0	209

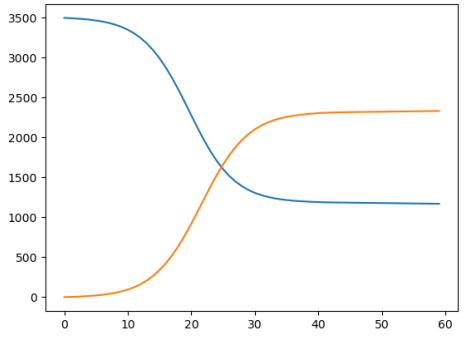
Figura 5: tabella dati ottenuta dal modello SIR

3.2 Confronto tra i due modelli

E' stato effettuato un confronto valutando in maniera qualitativa i grafici in output dei due modelli. Dal programma che implementa l'automa cellulare sono stati commentati i metodi "massVaccination" e "resusceptible" in maniera provvisoria, poiché nelle equazioni differenziali da noi utilizzate per implementare il modello Sir non sono considerate la possibilità di infettarsi due volte da parte di un soggetto né la vaccinazione dei soggetti. I grafici dei due modelli fatti evolvere con stessi valori in input a confronto sono riportati sotto.

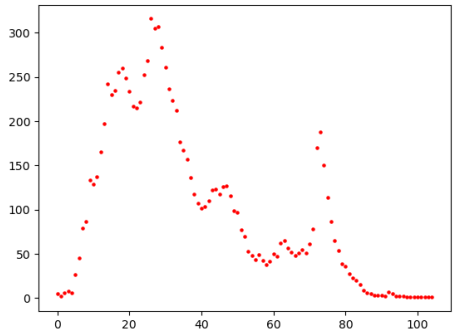


(a) I da modello sir

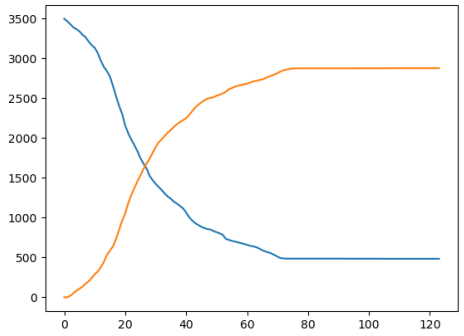


(b) R(gialli) e S(blu) da modello Sir

Figura 6: Grafici ottenuti dal modello Sir



(a) I da modello con automa cellulare



(b) R(gialli) e S(blu) da modello con automa cellulare

Figura 7: Grafici ottenuti dal modello Sir

4 Conclusioni

Lo scopo di questo progetto è stato quello di creare un programma in grado di dare una descrizione quantitativa dell'evoluzione di un'epidemia. Il modello SIR è il modello più semplice e più approssimato in grado di fornire tale descrizione e per questo motivo non risulta sufficiente a descrivere realisticamente un simile evento. Si è cercato di simulare una popolazione su automa cellulare per verificare che l'evoluzione del modello fornisse dati simili al modello teorico. Nel caso in cui sono state prese in considerazioni fattori come la vaccinazione e la possibilità di infettarsi più volte nel corso della pandemia abbiamo ritrovato un andamento simile alle cosiddette "ondate" di soggetti infetti. Il modello permette di evidenziare l'efficacia di strategie contenitive come il vaccino. Si può notare infatti l'efficacia di tale strategia in termini di individui infettati e velocità di diffusione e arresto dell'epidemia.
L'implementazione del modello qui esposta presenta sicuramente delle possibilità di miglioramento dal punto di vista dell'ottimizzazione del codice come pure dal punto di vita delle capacità del programma. Alcuni miglioramenti potrebbero essere effettuati sulla qualità dell'output grafico e altri sulla dinamica del contagio e aggiungere un possibile lockdown raggiunta una certa soglia di Infetti.