UNIVERSITÀ DEGLI STUDI DI
NAPOLI FEDERICO II

Scuola Politecnica e delle Scienze di Base
Corso di Laurea in Ingegneria Informatica

Elaborato finale in **Intelligenza Artificiale**

Software basati sugli agenti per modellare l'epidemia di Covid-19 con simulazioni sul territorio campano e napoletano

Anno Accademico 2019-2020

Candidato:

Yuri Spaziani

matr. N46003377

A chi mi è stato vicino.

Ai professionisti che hanno ispirato questo percorso.

Ai bravi professori che lo hanno accompagnato.

Indice

Introduzione

Negli scorsi decenni, molti specialisti di malattie infettive hanno fatto luce su importanti meccanismi circa la patogenesi e la diagnosi delle stesse. Tuttavia, nonostante tali enormi passi avanti in termini di conoscenza, affrontiamo ancora grandi sfide nel rilevamento di epidemie e nello sviluppo di politiche e programmi di controllo efficaci a gestirle o evitarle.

Con una sempre maggiore potenza computazionale a disposizione, tecniche di modeling di tali fenomeni hanno conosciuto, grazie alla loro capacità di arricchire la nostra comprensione dei percorsi causali delle malattie infettive, una più ampia diffusione, e contribuiscono ad aiutare i decisori politici ad implementare strategie di controllo efficaci a prevenire e controllare la diffusione di tali malattie. Tali tecniche di modeling computazionale offrono l'abilità di analizzare diverse possibilità di contenimento della malattia e di esaminare diversi "what-if".

Lo scoppio dell'epidemia di Covid-19, che ha causato quasi 2 milioni di morti nel corso del 2020 – cifra superata nel 2021 [1], ha enfatizzato la necessità di modelli più dettagliati, che includessero le politiche comportamentali di distanziamento sociale adottate dai governi e che avessero un'accentuata correlazione tra i dati prodotti dal modello e quelli forniti dall'analisi della realtà.

Due approcci popolari e diffusi negli studi di simulazione al calcolatore in epidemiologia sono la dinamica dei sistemi e la modellazione ad agenti.

La modellazione di sistemi dinamici corrisponde alla modellazione di equazioni differenziali non lineari; pertanto, essa è un tipo di modellazione basata sulle equazioni che può facilmente esprimere la relazione causa-effetto tra le molteplici variabili di un sistema complesso. Il modello del mondo di Forrester, uno dei primi e più noti esempi, fu usato per indagare la futura grandezza della popolazione mondiale. Utilizzava variabili come l'inquinamento ed il consumo delle risorse naturali, e tali variabili interagivano tra loro tramite nessi causali e cicli in retroazione [2].

I modelli di system dynamics per la diffusione di malattie infettive implementano solitamente principi strutturali derivanti dai tradizionali modelli epidemiologici matematici; tali principi sono aggregati in variabili. Il classico modello di questo tipo per la propagazione delle malattie infettive è il Suscettibile-Infetto-Risolto (Susceptible-Infectious-Recovered, SIR), sviluppato nel 1927 e che ha fornito intuizioni fondamentali sulla diffusione delle malattie [3][4]. In tali modelli, gli individui sono aggregati in gruppi, composti da molteplici individui con le stesse proprietà astratte. Tuttavia, l'approccio basato sulla dinamica dei sistemi è raramente utilizzato a livello individuale [5].

Nonostante i modelli aggregati possano offrire notevoli insight ed abbiano permesso la derivazione dei concetti fondanti dell'epidemiologia matematica, presentano distinte limitazioni quando il fulcro del sistema risiede nelle caratteristiche delle interazioni e dei contatti sociali attraverso i quali l'infezione si diffonde.

Spronata da crescenti risorse di calcolo e dalla necessità di valutazioni di scenario realistiche, la modellazione ad agenti è divenuta sempre più popolare. Questo riflette la possibilità per tali modelli di avere maggior flessibilità nel rappresentare una popolazione come un sistema di agenti interagenti e con abilità e caratteristiche eterogenee.

Entrambi questi approcci offrono degli importanti punti di vista sulle dinamiche di infezione, ma alle proprie fondamenta hanno delle assunzioni estremamente differenti. In system dynamics, le persone all'interno della stessa categoria sono assunte essere omogeneamente mescolate e con un'eguale probabilità per ciascun individuo di diffondere la malattia a qualsiasi altro individuo [6]. Se la malattia si diffonde principalmente attraverso contatti con individui infetti, assumere omogeneità e mescolanza perfetta può

ridurre l'accuratezza e minare la validità del modello. Si può comunque assumere una mescolanza preferenziale da parte di determinati gruppi, tuttavia la rappresentazione di tale mescolanza può essere molto complicata e difficile da gestire.

Al contrario, i modelli ad agenti non solo possono prevedere effetti in retroazione, ma sono anche molto flessibili nell'implementare l'eterogeneità delle caratteristiche individuali e per valutare le interazioni tra individui in una determinata rete. Tuttavia, i modelli ad agenti presentano un cospicuo trade-off, in quanto hanno elevati costi computazionali.

Molti modelli, sia deterministici che stocastici, sono creati per valutare l'impatto e l'efficienza di differenti misure di controllo e prevenzione, come vaccinazioni, screening, contact tracing, trattamento mirato, lockdown [7][8][9][10].

La trattazione, prendendo come spunto l'esempio di alcuni modelli di successo di entrambe le tipologie, cercherà di esaminare brevemente le differenze che intercorrono tra di esse, per poi soffermarsi sulla metodologia basata sugli agenti. Quest'ultima verrà infine utilizzata con lo scopo di simulare lo scoppio dell'epidemia di SARS-CoV-2 e la conseguente prima ondata nella regione Campania e nella provincia di Napoli.

Capitolo 1: Dinamica dei sistemi

System Dynamics è un approccio volto alla comprensione dei comportamenti dinamici e dei meccanismi dei sistemi complessi nel tempo, per poterli gestire propriamente. I sistemi dinamici incorporano spesso elementi come delay temporali, forze causali, feedbacks, interazioni e relazioni non lineari. Il cuore di tale approccio risiede nei feedback e nei loop causali, che forniscono una piattaforma per catturare gli effetti causali circolari e concettualizzare la struttura di un sistema complesso. Il comportamento complesso spesso sorge dalle interazioni di due tipi di loop in retroazione, chiamati "reinforcing" (o feedback positivo) e "balancing" (o feedback negativo) [11].

I loop rinforzanti spesso hanno la funzione di amplificare o accelerare un comportamento divergente del sistema, mentre i loop bilancianti hanno l'effetto di contrastare tale divergenza e far tendere il sistema all'equilibrio. Gli stock e flussi sono esemplari per illustrare la struttura di un sistema dinamico. Matematicamente parlando, un sistema dinamico è una combinazione di equazioni differenziali non lineari ben rappresentate dagli stock e flussi.

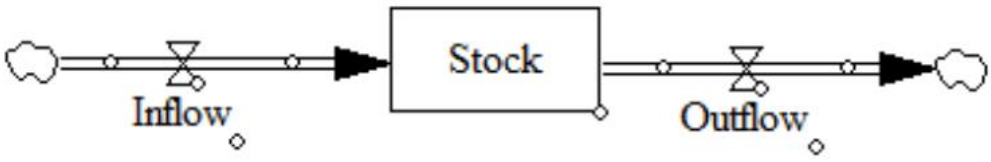

Figura 1: Struttura generale della notazione diagrammatica degli stock e flussi. [11]

In un sistema dinamico si possono descrivere sistemi del mondo reale in termini di quantità connesse e continue all'interno di feedback causali, ed ogni deduzione può essere tratta e derivata da tale struttura in retroazione [12].

La rappresentazione di Figura 1 è diagrammatica; in termini matematici, invece, l'equazione differenziale può essere rappresentata come [11]:

$$\frac{\mathrm{dstock(t)}}{\mathrm{dt}} = \mathrm{Inflow(t)} - \mathrm{Outflow(t)}$$

Negli scorsi decenni, la dinamica dei sistemi si è diffusa in diversi campi come l'economia, il management, la politica pubblica, l'ecologia, l'epidemiologia. Con lo scopo di comprendere la natura dei sistemi complessi e adottare decisioni razionali e ragionevoli, tale approccio è stato attuato nella risoluzione di problemi che spaziano dalle strategie corporative alla dinamica dei sistemi ecologici, dalla gestione della catena di distribuzione alla battaglia contro le malattie infettive [11].

1.1 Dinamica dei sistemi nell'epidemiologia

I modelli matematici sulla trasmissione delle malattie all'interno delle popolazioni umane sono riconosciutamente utili ad aiutare i decisori politici e gli epidemiologi ad interpretare i trend epidemiologici, a comprendere le dinamiche del diffondersi della malattia e a misurare l'efficienza della prevenzione e del controllo, come quelli esistenti per il morbillo, l'HIV ed altre infezioni emergenti [13].

Negli anni Venti, il modello SIR (Susceptible-Infectious-Recovered) e varianti come il SEIR (Susceptible-Exposed-Infectious-Recovered) furono introdotti e stabilirono quelle che sarebbero diventate le fondamenta di gran parte dell'epidemiologia matematica [3]. Il SIR è composto da tre gruppi di individui: 1) suscettibili, ossia in grado di contrarre la malattia; 2) infetti, cioè coloro che sono stati infettati e sono in grado di infettare; 3) risolti, ovvero coloro che hanno già contratto la malattia e sono guariti. Il SEIR, invece, mostrato in Figura 2, altro non è che un'estensione del SIR, e prende in considerazione l'esistenza di un periodo d'incubazione in cui l'individuo è stato infettato ma non può

ancora infettare. Tale comportamento è espresso aggiungendo al modello la categoria degli esposti (E). Un importante parametro in tali modelli è il numero di riproduzione di base (R0), che rappresenta il numero di suscettibili che saranno infettati da uno stesso infetto all'interno del ciclo corrente, quindi rappresenta il grado di contagiosità della specifica epidemia.

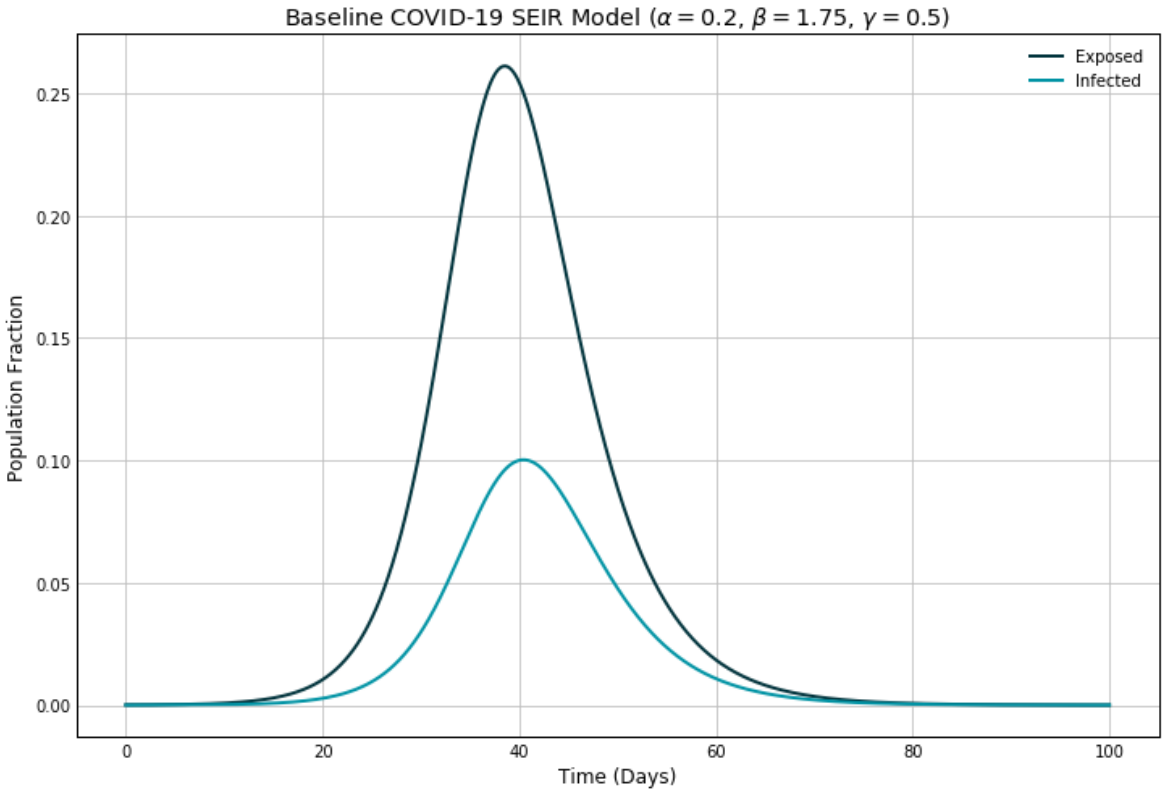

Figura 2: Modello SEIR per Covid-19. [15]

Con lo scoppio dell'epidemia di Covid-19 e la sua rapida diffusione nel mondo, gran parte dei governi ha adottato misure di lockdown, lo stato d'emergenza, ha chiuso scuole ed università in presenza e le imprese hanno ridotto i loro viaggi ed incentivato politiche di lavoro da casa. Sono inoltre stati imposti la distanza di almeno un metro nei luoghi pubblici, l'utilizzo della mascherina e talvolta anche un coprifuoco. A tali politiche ci si riferisce col nome di distanziamento sociale.

L'idea è quella di rendere meno frequenti i contatti tra persone in modo da ostacolare la diffusione della malattia. Gli effetti sono spesso illustrati con immagini come quella in

Figura 3, dove la curva rossa viene appiattita per disperdere l'epidemia nel tempo quanto più possibile. In questo modo, ci sono sufficienti risorse per aiutare le persone malate, e il tasso di sopravvivenza aumenta.

Nell'ottica di includere il social distancing nel modello SEIR, un'altra importante variante è nata: il SEIR-SD (SEIR-Social Distancing) [15].

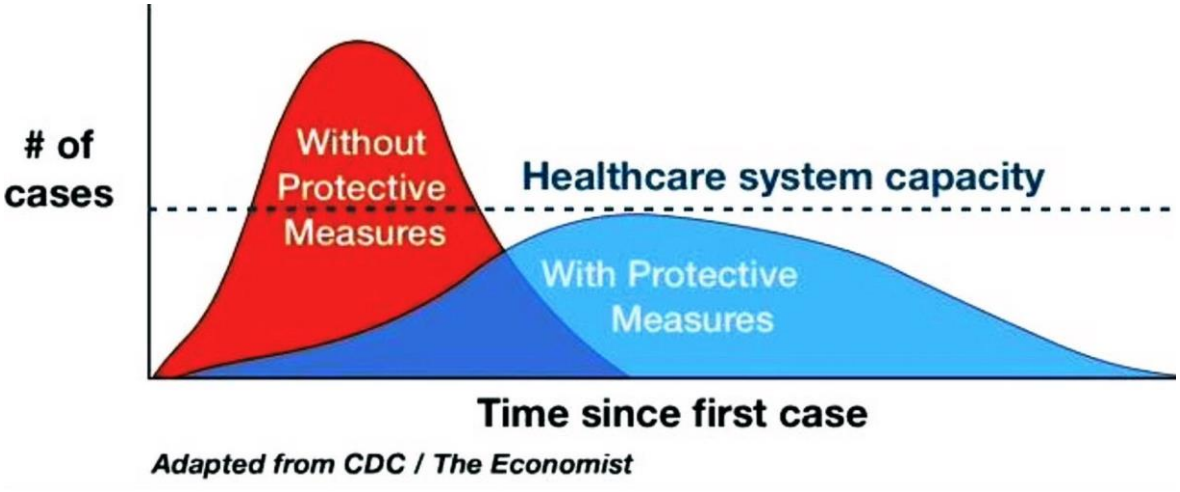

Figura 3: Appiattimento della curva per poter gestire l'epidemia. [15]

1.2 SEIR con Social Distancing

Il SEIR con distanziamento sociale è algebricamente definito come [15]:

$$\frac{dS}{dt} = -\frac{\rho\beta IS}{N}$$
$$\frac{dE}{dt} = \frac{\rho\beta IS}{N} - \alpha E$$
$$\frac{dI}{dt} = \alpha E - \gamma I$$
$$\frac{dR}{dt} = \gamma I$$

Dove S sono i suscettibili, E sono gli esposti, I sono gli infetti, R sono i risolti. N è il numero totale di individui nella popolazione (N = S + E + I + R). Il parametro non negativo reale β è la frequenza con la quale avvengono contatti all'interno della popolazione. Il parametro non negativo reale γ è l'inverso del periodo medio di infezione ($\frac{1}{\tau_{infetto}}$). α, invece, è il parametro caratterizzante il SEIR base, è non negativo e reale e rappresenta l'inverso del periodo di incubazione ($\frac{1}{\tau_{incubazione}}$). Il SEIR-SD, infine,

aggiunge al modello il parametro ρ, reale e non negativo, che assume valori in [0,1]. 0 è il caso ideale in cui tutta la popolazione è perfettamente isolata ed in quarantena, mentre quando è 1 le equazioni del modello si riducono a quelle di un SEIR, in quanto il distanziamento sociale non è preso in considerazione [3][15].

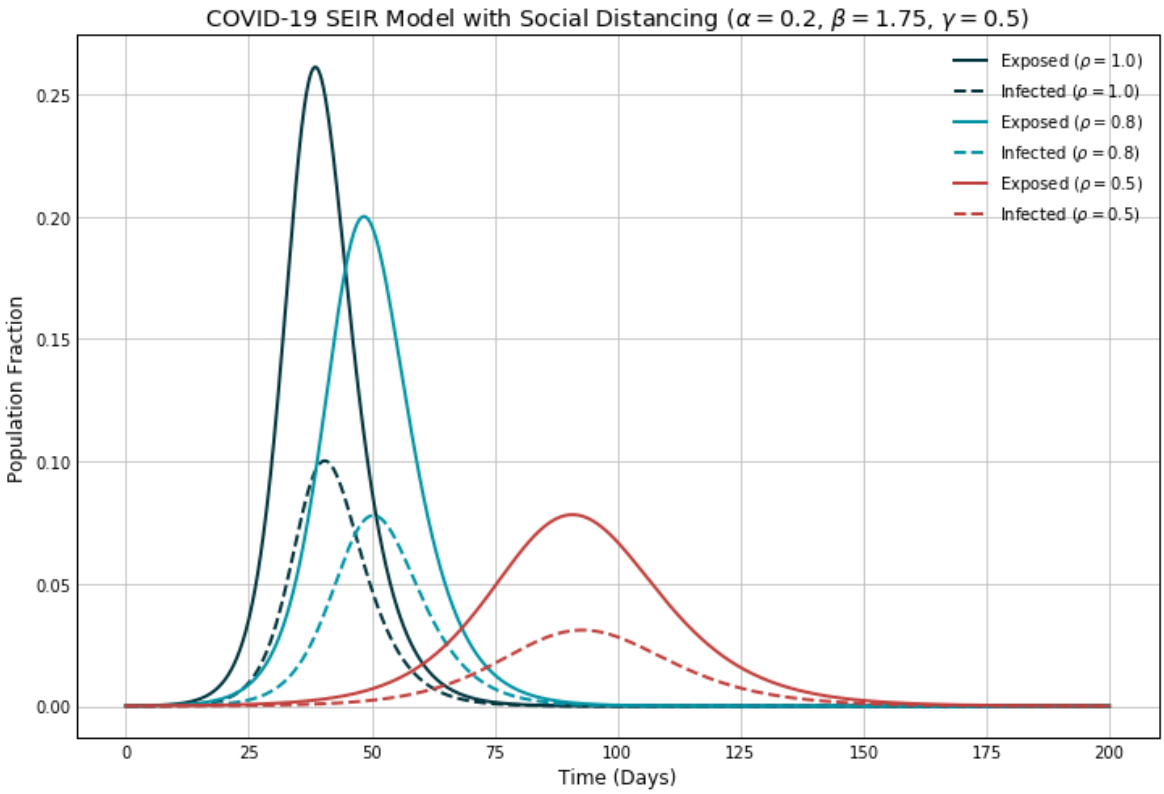

Figura 4: Modello SEIR-SD per il Covid-19. [15]

1.2.1 SEIR-SD in Campania

Numerose applicazioni di tale modello hanno visto luce nel periodo recente [16][17], conseguentemente lo scoppio dell'epidemia di SARS-CoV-2. Uno studio svolto in Campania [18] ci permette di osservare le previsioni sull'evoluzione dell'epidemia in tale territorio nei suoi primi giorni.

Per tale scopo è stata assunta una funzione variante nel tempo che descrivesse il distanziamento sociale in corrispondenza delle norme adottate dai decisori politici. Tale funzione è studiata per andare a limitare in modo sempre maggiore i contatti nel tempo, fino allo spegnersi dell'epidemia. Invece, i restanti parametri del modello sono stati ricavati tramite tecniche di computazione evolutiva (Differential evolution, DE).

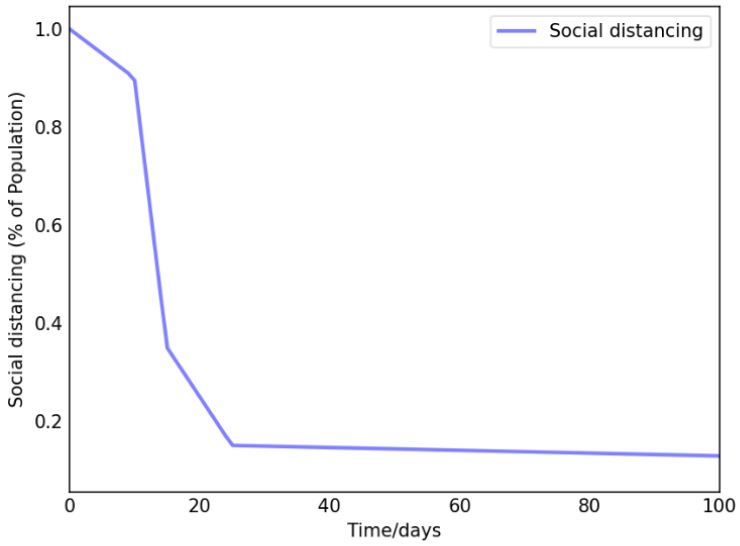

Figura 5: Funzione variante nel tempo che descrive un distanziamento sociale dinamico. [18]

I risultati del modello così strutturato, con parametri $\alpha = 0.33$, $\beta = 2.25$, $\gamma = 0.50$ ed un R0 pari a 4.5, coerentemente con le scoperte di [19] che suggeriscono che l'R0 cada tra 1.40 e 6.49, sono rappresentati dai seguenti grafici.

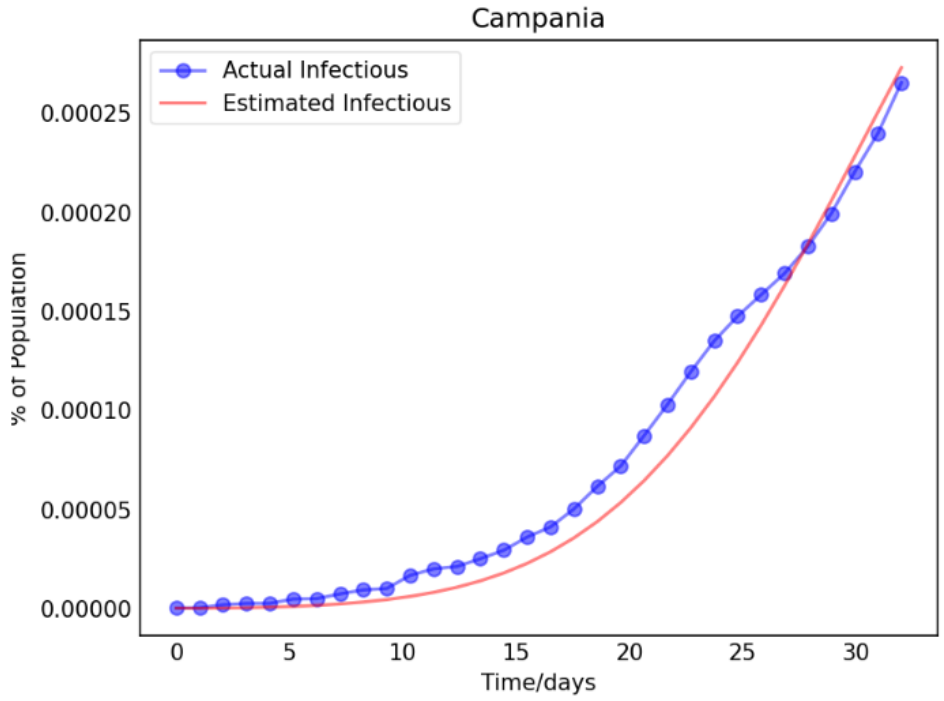

Figura 6: L'evoluzione nel tempo dell'epidemia da Covid-19 in Campania basandosi sui parametri inseriti nel modello. [18]

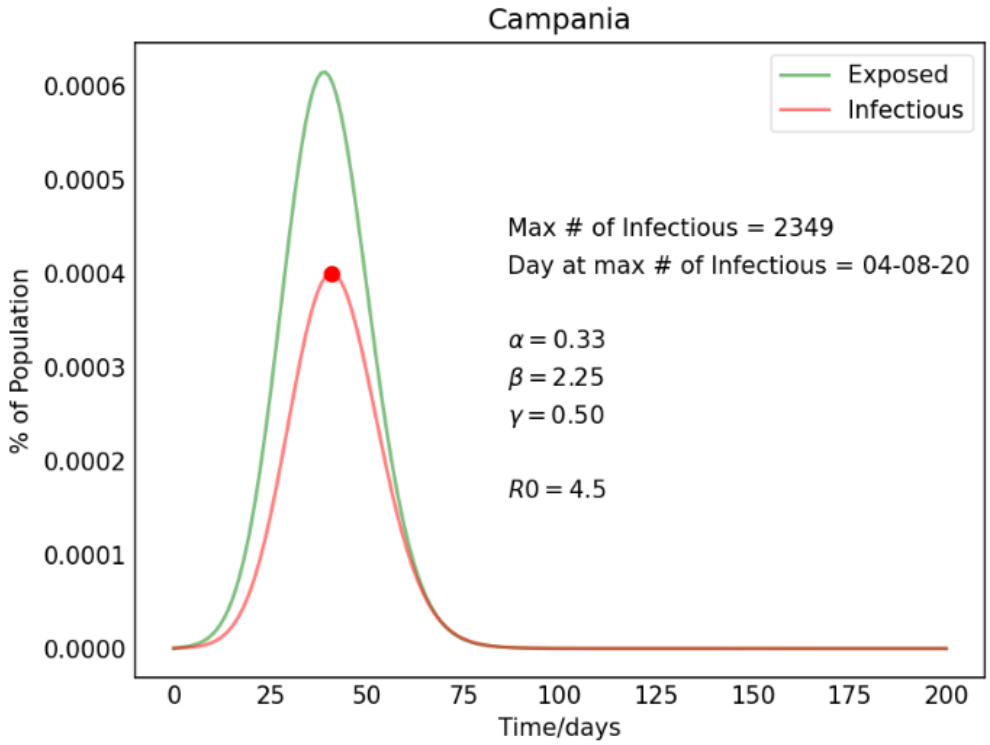

Figura 7: Infetti giornalieri previsti dal modello. [18]

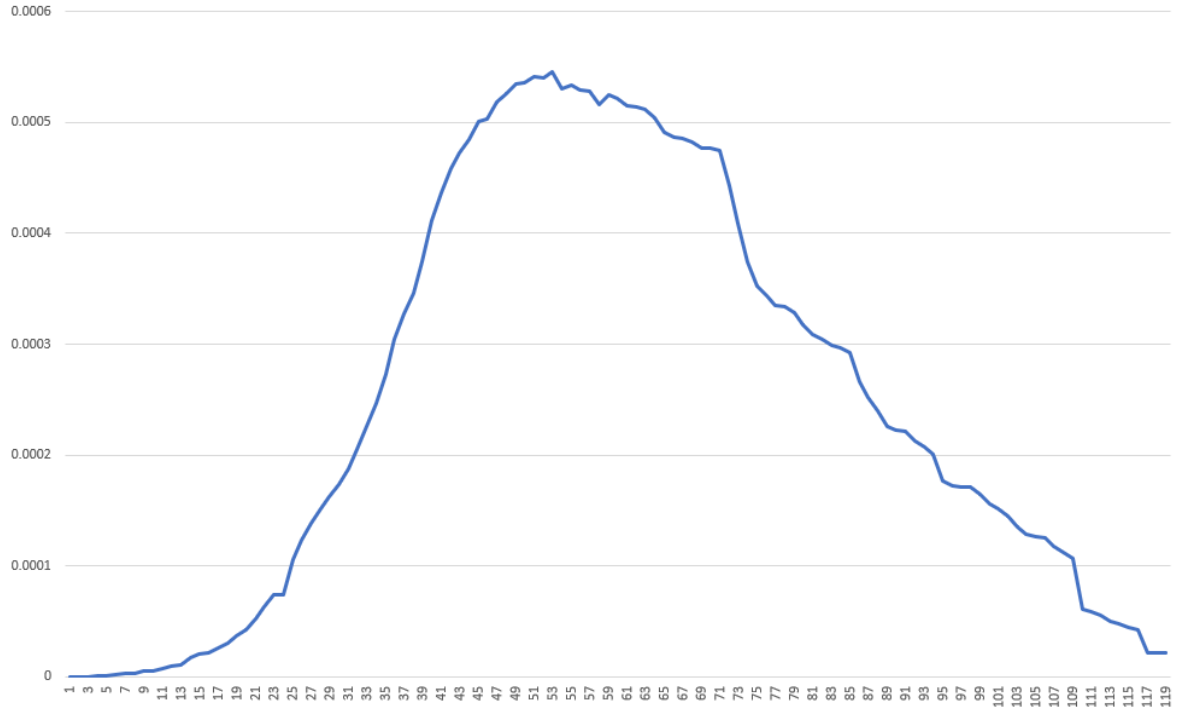

Figura 8: Diagnosticati attivi in Campania dal 24/02/2020 al 22/06/2020. [20]

È importante notare che le previsioni del modello si riferiscono agli attivi in un determinato giorno, e non ai diagnosticati. Si può comunque osservare che, avendo l'Italia

allentato le norme restrittive in data 18 maggio 2020 [21], tali previsioni corrispondono solo in parte ai dati raccolti successivamente. Tale modello è dunque ideale per comprendere la diffusione della malattia, i cambiamenti prodotti dall'introduzione del distanziamento sociale o comprendere la collocazione temporale del picco dei contagi, ma molti comportamenti complessi della popolazione, causati da eterogenicità in mescolanza e reti di contatto vengono "mascherati" dalla suddivisione in gruppi con caratteristiche omogenee.

Nonostante in questo contesto semplificato non vengano prese in considerazione le differenze nella suscettibilità all'infezione né le variazioni nei pattern di contatto tra individui eterogenei, il modello SIR e le sue varianti hanno esercitato una profonda e duratura influenza sul modeling delle malattie infettive. In contrasto al SIR, che descrive meglio quelle malattie che conferiscono immunità a vita, come il vaiolo, sono sorti altri modelli come il SIS (Susceptible-Infected-Susceptible) ed il SIRS (Susceptible-Infected-Recovered-Susceptible), che meglio caratterizzano quelle malattie o infezioni che avvengono in maniera ripetuta e per cui la guarigione non conferisce immunità a vita, come molte malattie sessualmente trasmissibili, l'influenza, così come infezioni con un'immunità che svanisce rapidamente, come la pertosse [22].

Alcuni studi arricchiscono il modello base suddividendo i gruppi in sottogruppi, in modo da generare una struttura più realistica [23][24][26]. Tale disaggregazione, che prende il nome di disaggregazione sugli attributi [25], può essere utilizzata per stratificare un modello, così da rappresentare una più complessa gerarchia della popolazione, o per integrare caratteristiche personali per generare dei comportamenti dinamici più complessi. Per evidenziare le variazioni nei pattern di mescolamento (mettendo in luce, ad esempio, che persone che parlano la stessa lingua sono più inclini a mescolarsi ed interagire tra loro) viene utilizzata una matrice di mescolamento per adattare la formulazione semplificata della forza infettiva $\frac{dS}{dt}$. Tale matrice stabilisce un coefficiente di contatto diverso tra gruppi diversi [27].

Capitolo 2: Modellazione ad agenti

Un modello ad agenti (Agent-Based Model, ABM), che è un tipo di modello computazionale, è un sistema multi-agente composto da un numero di agenti interagenti costruiti in modo da soddisfare dei propri obiettivi all'interno dell'ambiente [27]. Spesso il suo scopo è quello di simulare, in maniera semplificata, processi esistenti nel mondo reale. La caratteristica che distingue un modello ad agenti è che gli agenti sono rappresentati esplicitamente per interagire l'un l'altro attraverso lo scambio di messaggi o il trasferimento di dati all'interno di un network. Gli agenti sono degli attori sociali discreti e con il loro proprio comportamento, dunque posseggono la capacità di reagire all'ambiente computazionale. Gli agenti, inoltre, frequentemente sono anche capaci di muoversi e di avere l'abilità di registrare i propri stati storici. Dato lo stato, la storia o l'ambiente degli agenti, questi, seguendo un insieme di regole, eventi o strategie incorporati, possono raggiungere dei comportamenti sofisticati [27]. Convenzionalmente, gli agenti posseggono le seguenti proprietà [28]:

- **Autonomia.** Ci sono degli stati built-in all'interno degli agenti che consentono loro di compiere decisioni in conseguenza del loro stato corrente. Gli agenti sono capaci di imparare ed adattare le loro azioni in base all'esperienza. I diagrammi di stato sono una delle forme usate per rappresentare il loro stato corrente o precedente, così come le transizioni da uno stato ad un altro.
- **Abilità sociale.** Questo termine si riferisce all'interazione tra agenti attraverso qualche tipo di linguaggio comunicativo. In termini di programmazione, significa che gli agenti possono inviare e ricevere messaggi da altri agenti.

- **Reattività.** Gli agenti si trovano all'interno di un ambiente e possono percepire il loro ambiente ed altri agenti situati nel loro quartiere.
- **Proattività.** Gli agenti hanno infine la caratteristica di adottare comportamenti diretti al raggiungimento di un obiettivo. Spesso hanno obiettivi che riguardano il seguire la propria iniziativa e compiere azioni in modo da raggiungere tali obiettivi.

La metodologia della modellazione basata sugli agenti al momento è ampiamente usata in varie discipline di scienze sociali, incluse la psicologia, la salute pubblica [23] e le scienze politiche. Essendo un potente strumento di simulazione, è sorto un gran numero di applicazioni nel mondo reale che riproduce i fenomeni naturali o sociali emergenti di un sistema con lo scopo di spiegarli. Esempi di tali applicazioni sono simulazioni sul mercato e sulla diffusione.

Negli ultimi anni c'è un crescente interesse nell'utilizzare i modelli basati sugli agenti per simulare i comportamenti del mercato; una delle prime applicazioni commerciali fu il modello ad agenti per il NASDAQ stock market [29]. Nel modello ad agenti del NASDAQ, i partecipanti, le istituzioni, le regole del mercato e le loro interazioni sono simulate approssimando i processi esistenti nel mercato del mondo reale: gli agenti investitori sono in grado di comprare o vendere le azioni seguendo varie strategie, da quelle semplici alle più complicate che includono del learning. Grazie alla capacità della modellazione ad agenti di rappresentare il sistema dalla prospettiva dei comportamenti di individui eterogenei e delle attività del mondo reale piuttosto che processi o medie astratte, il modello del NASDAQ può simulare l'impatto dei cambiamenti sul mercato finanziario sotto diverse condizioni e circostanze, inoltre può fornire degli avvertimenti circa degli esiti inaspettati di determinate strategie e può monitorare il comportamento degli agenti in risposta alle diverse regole impostate [29] [30].

La diffusione spesso rappresenta il processo di diffusione delle idee, in cui le persone sono influenzate da coloro che li circondano, ed è un processo fondamentale osservato in diverse circostanze psicologiche, sociali ed economiche. La modellazione ad agenti

può essere utilizzata per descrivere molti fenomeni di diffusione all'interno dei sistemi umani, come i modelli di adozione di un prodotto o di diffusione di una malattia. L'uso della modellazione ad agenti in scienze mediche è aumentato molto negli ultimi anni [31][32][33], anche precedentmente alla pandemia. Con lo scopo di esplorare la propagazione delle malattie infettive all'interno di una popolazione definita, uno studio utilizza un approccio basato sulla modellazione ad agenti per simulare la diffusione delle malattie in un ambiente urbano con un geographic information system (GIS) e spacial network integration [33]. Questo lavoro utilizza come caso di studio un'epidemia di morbillo, implementandola in un modello ad agenti all'interno di una popolazione chiusa, dove le interazioni tra gli individui sono associate a determinati luoghi e la mobilità degli agenti è incapsulata in una rete di trasporti. Il modello è utilizzato per esaminare diversi scenari e rispondere a molteplici domande all'interno di un ambiente geografico simulato. La metodologia GIS viene ripresa in seguito anche da uno studio seguente lo scoppio dell'epidemia di Covid-19 [46]; tali modelli, infatti, basandosi quasi interamente sul luogo in cui si trovano gli agenti in un dato momento, forniscono utili informazioni su quali sono i più probabili focolai di contagio. Dai risultati di [33], spesso risultano essere luoghi con un'elevata concentrazione di persone, come scuole o università.

2.1 Confronto con la dinamica dei sistemi

Le tecniche di modellazione ad agenti catturano molto bene la struttura della rete attraverso la quale la malattia si diffonde, se essa è una struttura costante. In un mondo dove pattern persistenti di contatto sono un'importante caratteristica di molte istituzioni umane, l'abilità di ragionare su tali rappresentazioni arricchisce la nostra comprensione degli elementi determinanti di specifici pattern epidemici e cattura i trade-off che portano gli interventi dei decisori politici. In contrasto ai modelli aggregati, nei quali la popolazione è assunta essere totalmente ed omogeneamente mescolata, i modelli ad agenti sono flessibili nel simulare la trasmissione delle malattie all'interno di una rete. Data una struttura di rete d'interesse, integrare tale rappresentazione in un modello

corrisponde a determinare le connessioni e caratteristiche sociali [34]. C'è una gran produzione nell'ambito della ricerca che ha studiato la topologia delle reti ed il loro impatto sui vari processi della diffusione delle malattie [32]. Solitamente la rete simulata è definita in termini di una distribuzione degli individui nello spazio o del modo in cui le loro connessioni sono stabilite. Siccome infezioni differenti si diffondono attraverso percorsi diversi, la rete deve specificare tale percorso per soddisfare il contesto di una particolare malattia infettiva [34]. All'interno di una struttura di rete usata per la diffusione delle epidemie, possono essere introdotti i livelli di suscettibilità o di rischio relativo per gli individui sulla base di caratteristiche diverse come l'età, l'etnia o il genere. Possono inoltre essere applicate variazioni attorno a un valore medio del tasso di infezione anche per individui dello stesso gruppo. Data una tale concentrazione sul livello dell'individuo, è molto più facile catturare informazioni storiche sugli stessi; in più, tali informazioni possono essere utilizzate nella calibrazione del modello e successivamente usate per la costruzione di politiche adattive, che cambiano sulla base delle caratteristiche della persona coinvolta.

Oltre a delle strutture di rete semplicemente stabili, alcuni studi cercano di rappresentare le reti come delle proprietà che emergono dal movimento degli agenti. Uno studio cattura i pattern di movimento umani e dati sugli agenti, sulle loro routine di movimento e di località per derivare le reti di trasmissione [35]. Quest'idea è anche alla base delle app per smartphone che sono state rilasciate durante l'emergenza Covid-19, come l'app Immuni.

Sotto lo stimolo ed il supporto di una potenza computazionale sempre maggiore, la modellazione ad agenti è una soluzione sempre più considerata grazie alla sua abilità di rappresentare pattern di comportamento più complessi – come quelli dipendenti dalla storia individuale o dal learning – e di catturare in maniera naturale le dinamiche dei sistemi del mondo reale. Tuttavia, alla luce del fatto che i modelli compartimentali tradizionali hanno fatto la storia ed hanno un molto maggior numero di applicazioni passate, nei recenti anni c'è stata molta discussione circa i trade-off della modellazione ad agenti. Sterman e Rahmandad [6] hanno comparato i modelli ad agenti con i modelli

basati su equazioni differenziali nel contesto di un modello SEIR. Gli esperimenti sulla diffusione della malattia sono fatti su dei modelli SEIR stocastici e basati sugli agenti e modelli SEIR classici, e le implicazioni sulla scelta di un modello piuttosto che di un altro sono discusse nel dettaglio. L'assunzione di una mescolanza perfetta e di omogeneità dei modelli classici SEIR, viene rilassata nella versione agent-based attraverso l'integrazione delle topologie dei network e dell'eterogeneità degli individui; l'esito delle simulazioni dimostrano che alcune dinamiche e pattern comportamentali che emergono dal modello ad agenti sono sensibilmente diverse da quelle osservate nei modelli basati su equazioni differenziali. Le interazioni degli individui possono generare effetti all'interno della rete che possono eventualmente portare a deviazioni significative rispetto ai trend previsti. L'architettura del modello impatta anche sul modo in cui le assunzioni che facciamo sul modello influenzano le nostre valutazioni della realtà. Inoltre, i risultati suggeriscono che la granularità e l'eterogeneità delle caratteristiche degli individui è scarsamente catturata o analizzata nei modelli aggregati, ed alcune dinamiche che riguardano l'eterogeneità non siano costruibili o riproducibili all'interno del modello deterministico.

Un'altra differenza tra i due tipi di modelli è che, per un dato scenario con una specifica parametrizzazione del modello, il risultato del modello stocastico ad agenti è solitamente espresso come una distribuzione di esiti diversi, mentre un modello deterministico genera un'unica traiettoria che rappresenta il pattern epidemiologico utilizzando l'approssimazione dei parametri a valore medio [6]. Per un sistema non lineare, utilizzare la media delle distribuzioni all'interno del sistema può generare risultati molto differenti rispetto a quelli che si otterrebbero applicando al sistema diversi elementi singoli di tali distribuzioni. Per i decisori politici che lavorano con dei sistemi non lineari, le medie ottenute nei modelli aggregati deterministici non sempre forniscono un'approssimazione precisa della media del totale delle simulazioni si un modello basato sugli individui e può eventualmente portare a delle stime biased della reale situazione [30]. Il fatto che un modello ad agenti catturi la variabilità dell'esito offre flessibilità extra nell'analisi dei casi estremi, nell'applicazione di esplicite

preferenze sul rischio e nella comprensione del grado di variabilità tra casi diversi, come la variabilità dei risultati del sistema associati con ciascun set di interventi. Un altro studio compara queste due metodologie in un modello SIR, evidenziando come la grandezza della popolazione può causare esiti delle simulazioni diversi tra i due modelli [36]. Enfatizza, infatti, che per popolazioni piccole, approssimazioni sulla quantità di individui che condividono le stesse caratteristiche può portare a risultati molto differenti di quelli ottenuti considerando degli individui discreti.

Dopo l'emergenza mondiale dell'epidemia di SARS, alcuni epidemiologi si accorsero dell'importanza per la salute pubblica di comprendere la natura delle infezioni e delle reti di trasmissione [37]. Il processo di contact tracing (o infection tracing) è riconosciuto come una parte integrale dei programmi di controllo delle malattie infettive. Più precisamente, esso dà la possibilità di individuare quali siano i contatti avvenuti con un elevato potenziale di rischio, identificare i casi attivi tra di essi prima che diffondano l'infezione ed, infine, fornire cure e trattamenti a tali casi tracciati, portando ad un abbassamento dell'incidenza della malattia.

Molti modelli epidemiologici sono sviluppati per investigare sui programmi di controllo dell'epidemia come il contact tracing, e su come generare effettive misure di prevenzione in modo da sopraffare malattie infettive esistenti e che si presenteranno come il Covid-19 o le MST, l'HIV, SARS, tubercolosi. Un modello che integra una rappresentazione del contact tracing costruito su di una popolazione interagente in maniera casuale, deriva ed analizza le soglie critiche e la probabilità dello scoppio di un'epidemia. Tale lavoro inoltre indica che può essere sviluppato un modello deterministico che approssima il comportamento del modello di tracing stocastico [38]. Altri studi enfatizzano come l'uso dei network di contatto, che contengono i percorsi attraverso cui la malattia si trasmette, nel rappresentare la natura dell'interazione umana sia più preciso della procedura di tracing adoperata dai modelli aggregati [7]. Per stimare l'utilità del contact tracing nei modelli SIR e SIS in termini di malattie sessualmente trasmissibili (MST) è stata utilizzata l'approssimazione *pairwise* e sono state condotte simulazioni completamente stocastiche, che hanno consentito l'analisi

della relazione tra l'efficienza del contact tracing e il numero riproduttivo di base (R0). Combinando delle reti clusterizzate generate da un computer all'interno del modello, si scopre che il clustering contribuisce alla distruzione delle possibili relazioni ed è osservata la riduzione del requisito minimo di efficienza del contact tracing, contrariamente a quanto previsto [7]. In un altro studio, vengono formulati due modelli di trasmissione dell'HIV con misure di controllo e prevenzione sulla base di modelli di differential infectivity e di staged-progression per valutare l'efficacia del contact tracing e dello screening casuale [39]. Il numero riproduttivo e l'equilibri endemico sono derivati dalla stima dell'impatto di diversi livelli di programmi d'intervento. L'efficacia dello screening casuale e del contact tracing varia tra i due modelli, il che ci ricorda l'importanza dell'eziologia della trasmissione della malattia, che non può essere trascurata quando si misura l'efficienza di programmi di prevenzione. Il contributo inoltre indica la possibilità di integrare il costo in termini economici come un'ulteriore misura di valutazione.

Oltre all'utilizzo degli strumenti di modellazione tradizionali per meglio comprendere le dinamiche dell'infezione ed i programmi di contact tracing, sono stati condotti molti sforzi nell'investigazione della trasmissione della malattia in corrispondenza dell'utilizzo del contact tracing all'interno delle reti sociali. Nonostante ciò, ci sono molti problemi critici circa la simulazione del contact tracing all'interno dei network, in quanto ci si basa unicamente sul riportare in maniera autonoma le proprie relazioni personali, richiede un dispendioso processo di raccolta dati, presenta delle assunzioni idealizzate della topologia della rete di trasmissioni, difficoltà nel rappresentare le dinamiche delle interazioni umane, che includono il rompersi e il formarsi di relazioni più o meno forti all'interno del network, oltre la dicotomia dello 0-1, presente o assente [34].

La modellazione basata sugli agenti, anche conosciuta come modellazione basata sugli individui, è stata utilizzata in molti campi ed applicazioni, inclusi il controllo del traffico aereo, il business process management, l'investigazione del comportamento dei consumatori e la diffusione delle epidemie. Con lo scoppio dell'epidemia da Covid-19,

sono molti i modelli ad agenti nati per soddisfare la necessità di comprendere meglio la malattia e definire in modo preciso misure di contenimento efficaci anche da un punto di vista economico [44][45]. Tali modelli, infatti, strizzano un occhio di riguardo al trade-off economico delle misure di contenimento.

In questa tesi si sono utilizzate tali tecniche di modellazione ad agenti per rappresentare la trasmissione delle malattie infettive. Il modello ad agenti utilizza questi ultimi per rappresentare le persone con uno specifico status sociale e diversi livelli di rischio nel contrarre Covid-19. Scenari differenti sono associati a design differenti del modello e a differenti rappresentazioni dell'epidemia, risultando in implementazioni differenti dell'ambiente all'interno del modello, dei pattern di contatto degli individui e delle caratteristiche degli agenti.

Capitolo 3: Il modello

Il modello in [10][42][43] è un modello computazionale ad agenti ideato in seguito alla prima ondata europea di SARS-CoV-2 per valutare l'efficacia di diverse misure di contenimento ed è stato utilizzato sul territorio francese [10] e newyorkese [42], scritto interamente in linguaggio C++.

Esso costruisce una rete di individui e li fa interagire per un predeterminato numero di giorni, permettendo molte combinazioni di misure di contenimento diverse. Esso infatti permette di abilitare, al soddisfare di determinate condizioni, il distanziamento sociale, il tracciamento e l'isolamento solo per alcune attività e per varie fasce d'età.

Il modello dev'essere utilizzato specificando il numero di agenti e di location. Nelle simulazioni svolte si sono utilizzati 500000 agenti su un'unica location.

3.1 Le librerie Intel oneAPI

Il software fa uso dell'Intel oneAPI Base Toolkit, utilizzando due componenti di tale pacchetto: Thread Building Blocks (oneTBB) e Math Kernel Library (oneMKL).

3.1.1 Thread Building Blocks

La libreria oneTBB è utilizzata per l'ottimizzazione e la velocizzazione del programma in multithread, consentendo la facile creazione di applicazioni multithread anche per coloro che non sono esperti di threading. Essa, infatti, enfatizza una programmazione scalabile ed in parallelo, rendendo possibile l'accesso ad una stessa risorsa o collection per thread diversi tramite la divisione di quest'ultima in pezzi più piccoli, anziché dividere il programma in diversi blocchi funzionali eseguiti ciascuno su di un unico

thread. TBB, inoltre, ottimizza l'utilizzo delle risorse computazionali della CPU, mappando il parallelismo logico di quest'ultima con i thread presenti nel programma [50].

3.1.2 Math Kernel Library

La libreria oneMKL viene utilizzata dal software per l'implementazione e il bilanciamento dell'accuratezza e delle performance della matematica vettoriale, nonché per l'implementazione di tecniche di RNG (Random Number Generator, generatore di numeri casuali), in particolare tramite l'utilizzo della libreria VSL (Vector Statistical Library) di RNGs ad alte performance [51].

3.2 Generazione del mondo

Innanzitutto, il modello inizializza il generatore di numeri random:

```
stream.resize(nThreads);
vslNewStream(&(stream[0]), VSL_BRNG_MCG31, 200882);

for (int i = 1; i < nThreads; ++i)
{
        vslCopyStream(&(stream[i]), stream[0]);

}
for (int i = 0; i < nThreads; ++i)
{
        vslLeapfrogStream(stream[i], i, nThreads);
        RandomGenerator.emplace_back(vlsRandGenerator(stream[i]));
}
```

Dove *nThreads* è il numero di core logici:

```
int nThreads(std::thread::hardware_concurrency());
```

Si noti che il codice presente in [43] si riferisce alla versione 2020 di TBB, e tale variabile viene inizializzata utilizzando la funzione *default_num_thread()* della classe deprecata in oneTBB 2021 *task_scheduler_init*, che però ha lo stesso output di *std::thread::hardware_concurrency()*, ossia il numero di core logici [52][14].

Inserito il numero di agenti come dato di ingresso, il modello inizializza il mondo come privo di social distancing e di tracking; dopodiché esso genera e dispone su di una griglia quadrata, rappresentata con un unico vettore (*_families*), le famiglie, sulla base del numero medio di individui (*p(AvgHousehold)*) per famiglia e del numero di agenti inserito (*_size*), ed attribuisce ad ognuna una posizione all'interno della stessa, utilizzando l'ottimizzazione computazionale in multithread fornita dal *task_group* di TBB.

```
long int n = static_cast<int>(sqrt(_size / p(AvgHousehold)));
_size = n;
_families.reserve(n*n);

int j(0), i(0);
tbb::task_group g;
while (j*i < n*n)
{
        for (int k = 0; k < nThreads; ++k)
        {
                vlsRandGenerator * rnd = &(RandomGenerator[k]);
                g.run(
                        [=] {
                                        _families.emplace_back(i, j, *rnd, p);
                                }
                );
                j++;
                if (j > n && i <= n) { i++; j = 0; }
                if (j*i == n * n) break;

        }
        g.wait();

}
```

dove *rnd* è il *RandomGenerator* precedentemente creato e *p* è un'istanza della classe *Parameters* istanziata nel **main**, che contiene un vettore *variables* con tutte le variabili caratteristiche della simulazione su un determinato territorio, che vengono specificate preventivamente e caricate a runtime da file csv in fase di setup.

```
while (getline(myfile, line).good()) {

        if (line == "") { continue; }
        variables[i] = stof(line);
        ++i;
}
```

Le famiglie sono a loro volta divise in 4 tipologie: *Single*, cioè una famiglia composta da un solo individuo adulto; *CoupleOnly*, cioè una famiglia composta unicamente da una coppia etero adulta e priva di figli; *CoupleWithChildren* è invece una famiglia composta da una coppia e da un numero variabile di bambini; *SingleParent* è infine una famiglia composta da un solo genitore e da un numero variabile di figli. Tale tipologia è scelta casualmente in base alla distribuzione delle corrispondenti categorie sul territorio reale, quindi ad ogni categoria è associata una probabilità ed ovviamente la somma delle probabilità associate alle tipologie fa 1, ossia una famiglia non può non far parte di una di queste categorie.

```
float r = rnd();
int type(0);

for (int i = 0; i < 5; ++i)
{
        r -= p(familyType, i);
        if (r <= 0)
        {
                type = i+1;
                break;
        }
}
```

dove *rnd()* corrisponde a

```
vsRngUniform(VSL_RNG_METHOD_UNIFORM_STD, stream, size, r, 0, 1);
```

ossia corrisponde alla generazione di un numero uniformemente casuale tra 0 ed 1.

- Un nucleo familiare composto da una sola persona viene generato partendo dal definire casualmente il sesso della stessa sulla base della proporzione dei nuclei familiari di maschi e femmine single all'interno della popolazione, ed in base al sesso pescato, il modello definirà l'età dell'individuo casualmente sulla base di una CDF polinomiale di quinto grado;

```
sex = rnd() > p(familySingleMale, 6) ? 1 : 0;
age = p.getPrecalculated(sex == 0 ? familySingleMale :
familySingleFemale, rnd());
_members.emplace_back(age, sex, rnd, p, this);
```

La funzione *getPrecalculated()*, così definita

```
int Parameters::getPrecalculated(int table, float p) const
{
        unsigned int a(static_cast<unsigned int>(p * _precision + 0.5));
        a = a >= _precision ? 99 : a;
        switch (table)
        {
        case familySingleMale:
                return familySingleMaleTable[a];
                break;
        case familySingleFemale:
                return familySingleFemaleTable[a];
                break;
        case familyCoupleOnly:
                return familyCoupleOnlyTable[a];
                break;
        default:
                return 0;
        }
}
```

ritorna un elemento in una posizione casuale da 0 a 99 di una delle tre tabelle che descrivono la distribuzione per età (da 0 a 99) di una famiglia composta da un uomo single, da una donna single, o dell'età della donna in una famiglia composta da una coppia etero. Tali tabelle sono calcolate ciascuna tramite la funzione *precalculate()*, utilizzata subito dopo la lettura dei parametri da file csv., così definita.

```
std::array<int, Parameters::_precision> Parameters::precalculate(int var)
{
	std::array<int, _precision> result = {};
	float r[_precision];
	float age;
	for (int i = 20; i < 100; i++)
	{
		r[i] = 0;
		for (int j = 0; j < 6; j++)
		{
			age = static_cast<float>(i / 100.0f);
			r[i] += variables[var + j] *
			 static_cast<float>(pow(age, 5-j));
		}
	}
	int j(0);
	for (int i = 20; i < _precision; i++)
	{
		while (j <= static_cast<int>(r[i] * _precision + 0.5)) { if (j >=
		_precision) break;  result[j] = i; j++; }
	}
	if (j>0) for (int i = j; i < _precision; i++) result[i] = result[j-1];
	return result;
}
```

Essa, prendendo in ingresso i valori della CDF polinomiale, genera per ciascun'età il valore di probabilità ad essa associato e ritorna il risultato in un array, che viene conservato nelle tabelle. Il parametro *_precision* è invece definito staticamente pari a 1000.

- Un nucleo familiare composto unicamente da una coppia etero senza figli viene generato a partire da una CDF polinomiale di quinto grado che descrive l'età della donna in una coppia di questo tipo all'interno della popolazione. A partire da essa, verrà ricavata l'età dell'uomo aggiungendovi la differenza con l'età dell'uomo, ricavata casualmente a partire da una distribuzione normale;

```
age = p.getPrecalculated(familyCoupleOnly, rnd());
_members.emplace_back(age, 1, rnd, p, this);
age = static_cast<int>(age + rnd.normal(p(AvgCoupleDistance, 0),
p(AvgCoupleDistance, 1)));
_members.emplace_back(age, 0, rnd, p, this);
```

Dove la funzione *normal()* del RNG genera un numero random da una distribuzione normale.

```
float vlsRandGenerator::normal(float alpha, float sigma) const
{
        float _r[1];
        vsRngGaussian(VSL_RNG_METHOD_UNIFORM_STD, stream, 1, _r,
alpha, sigma);
        return _r[0];
}
```

- Le coppie che hanno figli vengono generate a partire dall'età del primo figlio, definita casualmente a partire dalla media. Il numero di ulteriori figli è definito casualmente a partire da una distribuzione di Poisson e, per ciascuno di essi, l'età sarà calcolata a partire da una distribuzione normale che definisce la differenza d'età tra due fratelli, mentre il genere sarà definito in maniera completamente casuale (cioè con la stessa probabilità tra maschio e femmina). Infine, l'età della madre verrà calcolata aggiungendo all'età del primo figlio, l'età a cui la madre stessa lo ha partorito, calcolata pescando un valore random da una distribuzione gaussiana che descrive a che età le madri hanno il loro primo figlio all'interno della popolazione. L'età del padre, invece, viene definita come nel caso di un nucleo familiare composto da una coppia senza figli, ossia aggiungendo all'età della donna la differenza d'età con l'uomo, ricavata casualmente da una distribuzione gaussiana;

```
ageChild = static_cast<int>(rnd() * p(familyUnder20, 1));
childrenAges.push_back(ageChild);

numChildren = rnd.poisson(p(nChildren));

for (int i = 0; i < numChildren - 1; ++i)
{
        ageNextChild = static_cast<int>(rnd.normal(p(AvgSiblingDistance, 0),
        p(AvgSiblingDistance, 1)));
        ageChild = ageNextChild > 0 ? ageChild + ageNextChild : ageChild;
        childrenAges.push_back(childrenAges[0] + ageNextChild);
```

```
}

age = static_cast<int>(ageChild + rnd.normal(p(AvgAgeFirstChild, 0),
p(AvgAgeFirstChild, 1)));
_members.emplace_back(age, 1, rnd, p, this);

age = static_cast<int>(age + rnd.normal(p(AvgCoupleDistance, 0),
p(AvgCoupleDistance, 1)));
_members.emplace_back(age, 0, rnd, p, this);

for (size_t i = 0; i < childrenAges.size(); ++i)
{
        _members.emplace_back(childrenAges[i] < 0 ? 0 : childrenAges[i], rnd()
        < 0.5 ? 1 : 0, rnd, p, this);
}
```

- Nel caso di nuclei familiari composti da un unico genitore con dei figli, la famiglia verrà composta partendo dall'età del primo figlio, come nel caso delle coppie che hanno figli precedentemente discusso. La differenza risiede nel fatto che, essendo un unico genitore, il suo sesso sarà definito casualmente, con una proporzione che prevede il 65% di individui maschi, in questi casi.

```
age = static_cast<int>(ageChild + rnd.normal(p(AvgAgeFirstChild, 0),
p(AvgAgeFirstChild, 1)));
_members.emplace_back(age, (rnd() < 0.65 ? 0 : 1), rnd, p, this);
```

Ogni famiglia ha inoltre una variabile *unsigned int _socialReach*, che descrive il raggio entro cui una famiglia ha rapporti con altre famiglie all'interno di una società, che viene inizializzata prendendo casualmente un valore da una distribuzione di Poisson.

```
        _socialReach = rnd.poisson(p(AvgFriends));
```

Ad ogni agente sono inoltre assegnate un numero variabile (da 0 a 5) di comorbidità, termine che indica la coesistenza di più patologie diverse, sulla base della probabilità di averle data una certa età ed un certo sesso.

```
int index(5);
if (age < 35) index = 0;
else if (age < 45) index = 1;
else if (age < 55) index = 2;
else if (age < 65) index = 3;
else if (age < 75) index = 4;

for (int i = 0; i < nComorbidities; ++i)
{
        if (rnd() < p(ComorbidityRate, index + sex * 6 + i * 6 * 2)) _list[i] =
        true; else _list[i] = false;
}
```

Dove *_list[]* è un array di lunghezza *nComorbidities*, pari a 5.

Ogni comorbidità corrisponde, per il modello, a 5 anni in più per l'individuo. L'assegnazione di tale età, avviene al momento in cui l'individuo viene infettato, tramite la riga di codice

```
age = age + ind->getNumberofComorbidities() * 5;
```

Dopo aver generato tutti gli individui con un'età che va dagli 0 ai 99 anni

```
individuals::individuals(int age, int sex, vlsRandGenerator & rnd, const Parameters
& p, family * f)
        if (_age > 99) _age = 99;
```

Ed averli raggruppati in famiglie, il modello genera delle relazioni di amicizia tra le famiglie a partire dalla quantità di famiglie amiche delle stesse, ossia il loro *_socialReach*. Per farlo, il modello utilizza le classi *task_group* e *concurrent_vector* di TBB per smistare le operazioni sui core logici ed ottimizzare i

tempi. Si inizia col popolare per pezzi un *concurrent_vector* di una *struct socialPlace* appena creata, con tutte le famiglie

```
tbb::task_group g;


struct socialPlace {
	int x;
	int y;
	family * f;
	int reach;
} places;
int static const piecesSize(1000);
for (int pieces = 0; pieces < static_cast<int>(_families.size() / piecesSize) + 1;
pieces++)
{

	tbb::concurrent_vector<socialPlace> socialWorld;

	for (int i = pieces* piecesSize; i < ((pieces*piecesSize + piecesSize +
	100)>_families.size()? _families.size() : (pieces * piecesSize +
	piecesSize+100)); i++)
	{
		places.f = &_families[i];
		places.reach = static_cast<size_t>(_families[i].getSocialReach());
		socialWorld.push_back(places);
	}
```

Si noti che il ciclo for superiore non si è chiuso; esso, infatti, si chiuderà solo alla fine del processo di creazione dei rapporti di amicizia tra le famiglie.

Il vettore appena creato *socialWorld* viene quindi mescolato casualmente.

```
int seed = RandomGenerator[0].uniform();
std::shuffle(socialWorld.begin(), socialWorld.end(),
std::default_random_engine(seed));
```

Vengono poi assegnati casualmente i valori *x* e *y* del *socialPlace*, ossia i suoi spazi tramite la generazione di *nThread* valori casuali da 0 a 10 per un certo numero di volte, determinato dal numero di core logici e dal numero delle famiglie presenti.

```
int * spaces(RandomGenerator[0](10, vlsRandGenerator::size));
size_t pos(0);
int rIndex(0);
for (size_t i = 0; i < socialWorld.size(); i++)
{
	socialWorld[i].x = spaces[rIndex];
	socialWorld[i].y = spaces[rIndex + 1];
	if (pos >= socialWorld.size()) break;
	rIndex++;
	if (rIndex + 1 >= vlsRandGenerator::size) { spaces =
RandomGenerator[0](10, vlsRandGenerator::size); rIndex = 0; }
}
```

Quindi, sulla base dei valori di *_socialReach* ricavati da Poisson e sulla base dei valori di *spaces* trovati in modo uniformemente random tra 0 e 10, il modello aggiunge agli amici di ciascuna famiglia le famiglie che rientrano nel suo *_socialReach* tramite un ciclo while.

```
while (i < socialWorld.size())
{

	for (int k = 0; k < nThreads; ++k)
	{

		vlsRandGenerator * rnd = &(RandomGenerator[k]);
		g.run(
			[=] {

			int socialReach(socialWorld[i].reach);
			int socialReachSquares((socialReach - socialReach % 10) /
10 + 1);
			std::vector<family *> f;

			int i_int(static_cast<int>(i)),
			socialSize_int(static_cast<int>(socialSize));

			int startx(i_int % socialSize_int - socialReachSquares); startx
			= startx < 0 ? 0 : startx;
			int starty((i_int - i_int % socialSize_int) / socialSize_int -
			socialReachSquares); starty = starty < 0 ? 0 : starty;
			int xCells(0), yCells(0);
			for (size_t x = static_cast<size_t>(startx); x <
static_cast<size_t>(startx) + socialReachSquares; x++)
			{
				xCells++;
				for (size_t y = static_cast<size_t>(starty); y <
static_cast<size_t>(starty) + socialReachSquares; y++)
```

```
                                {
                                        yCells++;
                                        size_t pos(x + y * socialSize);
                                        if (pos == i) continue; // skip self
                                        int dist(distance(socialWorld[i].x,
                                        socialWorld[i].y, (xCells - 1) * 10 +
                                        socialWorld[pos].x, (yCells - 1) * 10 +
                                        socialWorld[pos].y));
                                        if (dist > socialReach) continue;
                                        if (socialWorld[pos].reach >= socialReach)
                                        {
                                                socialWorld[i].f-
                                                >addFriend(socialWorld[pos].f, true);
                                                 f.push_back(socialWorld[pos].f);
                                        }
                                }
                        }
                        socialWorld[i].f->saveFriends(f);
                });
                i++;
                if (i >= socialWorld.size()) break;
        }
        g.wait();
}
```

Il modello quindi, chiuso il for, definisce tutte le persone tra 20 e 65 anni che lavorano, generando valori casuali a partire dalla proporzione dei lavoratori in quella fascia d'età all'interno della popolazione, e le distribuisce, tra aziende piccole e non, in base alla proporzione di aziende con meno di 10 dipendenti (piccole) presenti nella popolazione d'interesse.

```
float empRate(p(employmentRate));
for (size_t i = 0; i < _families.size(); ++i)
{
        std::vector<individuals *> tempInd(_families[i].getIndividuals());
        for (std::vector<individuals *>::iterator j = tempInd.begin(); j < tempInd.end();
++j)
        {
                int age((*j)->getAge());
                if (20 < age && age <= 65 && RandomGenerator[0]() < empRate)
_employables.push_back((*j));
        }
}
```

```
int seed = RandomGenerator[0].uniform();
std::shuffle(_employables.begin(), _employables.end(),
std::default_random_engine(seed));
size_t
smallCompanies(static_cast<size_t>(p(ProportionsmallCompanies)*_employables.size()));
size_t employees(_employables.size());
for (size_t i = 0; i < smallCompanies; i = i + 2)
{
        for (size_t j = i; j < i + 2; j++) if (j < employees) _employables[i]->setWork(2,
        static_cast<int>(i), static_cast<int>(j));
}
for (size_t i = smallCcompanies; i < employees; i = i + 5)
{
        for (size_t j = i; j < i + 5; j++) if (j < employees) _employables[i]->setWork(5,
        static_cast<int>(i), static_cast<int>(j));
}
```

Infine, vengono create dal modello le scuole sulla base del numero di bambini e della media grandezza di una classe ed il numero di posti letto in terapia intensiva sulla base della densità degli stessi.

```
_schools = estimateSectorSize((p(familyType,1) + p(familyType, 2))*p(nChildren)
* _families.size() / (20*p(schoolClassSize)));
_shopping = estimateSectorSize(p(nShopping) * _families.size());

int location::estimateSectorSize(float a)
{
        float avgDist = sqrt(_families.size() / a);
        return static_cast<int>(avgDist+0.5);
}
```

3.3 Il paziente zero

La simulazione necessita la creazione di un individuo già infetto. Ogni giorno, per l'intera durata scelta, tale agente, noto come paziente zero, può infettare uno o più agenti. Tale comportamento simula l'effetto delle infezioni provenienti dall'estero nella realtà. L'agente non infetta necessariamente ogni giorno, ma decide ogni quanto farlo sulla base di una distribuzione di Poisson centrata nella media di delay tra un'infezione dall'estero ed un'altra; infatti, escluso il primo giorno, in cui solo una persona verrà infettata dal paziente zero, il numero di persone infettate da esso

sarà determinato pescando un valore random da una distribuzione di Poisson centrata nella media di contaminazioni che avvengono dall'estero.

```
individuals patientZero;
patientZero.setPatientZero();
int i(0);
while (i < max_days)
{
	int nInfected =
	(i==0?1:RandomGenerator[0].poisson(p(AvgInternationalContaminated)));
	int * r0 = RandomGenerator[0](static_cast<int>(_list.size()), nInfected);

	for (int j = 0; j < nInfected; ++j)
	{

		int n = _list[r0[j]].getSize();
		int r1 = RandomGenerator[0].uniform()%n;
		individuals * contaminatedInd = _list[r0[j]].getHead(r1);

		link l = foreignLink(contaminatedInd);
		patientZero.addLink(l, i);

	}

	i += RandomGenerator[0].poisson(p(AvgDelaiInternational));

}


foreignLink::foreignLink(individuals * i1) :link(i1, 36000000, 1, lForeign)
{
	_frequency = { 1,1,1,1,1,1,1 };

}
```

La durata di tali contatti rende l'infezione quasi sicura, come si vedrà in seguito.

3.4 La simulazione

Quando un individuo viene infettato, egli genererà tutte le altre connessioni (link). Tali link descrivono un contatto tra due persone, avvenuto per una certa durata e mantenendo una certa distanza.

```
link::link(individuals * i1, int duration, int distance, int type);
```

I link possono avere luogo una sola volta (come tutti quelli correlati al paziente zero) o essere ricorrenti, e quindi verificarsi con una determinata cadenza o in determinati giorni della settimana.

```
void individuals::addLink(link l)
{
	_reccurentLink.push_back(l);
}
void individuals::addLink(link l, int simTime)
{
	l.setSimTime(simTime);
	_oneTimeLink.push_back(l);
}

class link
{
public:
	...
protected:
	...
	std::array<int, 7> _frequency; //giorni della settimana

...

}
```

Tutti gli agenti genereranno delle connessioni con la propria famiglia, giornaliere e frequenti, a distanza ravvicinata:

```
void individuals::contaminated(vlsRandGenerator & rnd, const Parameters & p, int t, float reducedDiagnosis)
{
	_disease.contaminated(this, rnd, p, t, reducedDiagnosis);
	if (!(*getFamily()).linkCreated()) (*getFamily()).createLink();
}

void family::createLink()
{
	_size = _members.size();
	for (size_t i = 0; i < _size; ++i)
	{
		for (size_t j = i + 1; j < _size; ++j)
		{
			link l1 = sameRoofIntrafamilialLink(&_members[j]);
			_members[i].addLink(l1);
			link l2 = sameRoofIntrafamilialLink(&_members[i]);
```

```
                        _members[j].addLink(l2);
                }
        }

        _isLinked = true;
}

sameRoofIntrafamilialLink::sameRoofIntrafamilialLink(individuals * i1):
        intrafamilialLink(i1, lIntrafamilialHousehold)
{
        _frequency = { 1,1,1,1,1,1,1 };
}

intrafamilialLink::intrafamilialLink(individuals * i1, int type)
        :link(i1, 360, 1, type)
{
}
```

Gli individui più giovani di 20 anni avranno unicamente dei link con i propri compagni di scuola, intesi come tutte le persone della stessa età, distribuendole per il numero di scuole calcolato nella generazione del mondo.

```
if (age > 20)
{
        createShoopingLink((*i), simTime, RandomGenerator, p);
        createWorkLink((*i), RandomGenerator, p);
        createTransportLink((*i), simTime, RandomGenerator, p);
        createFriendLink((*i), simTime, RandomGenerator, p);
        createEventLink((*i), simTime, RandomGenerator, p);
}

else
{
        createSchoolLink((*i), RandomGenerator, p);

}


void location::createSchoolLink(individuals * ind, std::vector<vlsRandGenerator>
& RandomGenerator, const Parameters & p)
{
        // Se ha uno School Link non fare nulla
        if (ind->hasLink(lSchool)) return;

        int age((*ind).getAge());
        std::array<int, 2> position = ind->getFamily()->getPosition();
```

```
        // Vettore contenente i compagni di classe
        tbb::concurrent_vector<individuals *> classmates;
        // Troviamo il suo quartiere
        int x((position[0] - position[0] % _schools) / _schools);
        int y((position[0] - position[0] % _schools) / _schools);
        int maxPos = static_cast<int>(_families.size()),
fsize(static_cast<int>(sqrt(maxPos)));
        tbb::task_group g;
        int col(0);
        while (col < _schools)
        {
                for (int k = 0; k < nThreads; ++k) {
                        vlsRandGenerator * rnd = &(RandomGenerator[k]);
                        g.run(
                                [=, &classmates] {
                                        int row(0);
                                        while (row < _schools)
                                        {
                                                int pos = (col +  y* _schools)*fsize +
x * _schools + row;
                                                if (pos < maxPos) {
                                                        //persone della stessa età
                                                        std::vector<individuals *>
familyMembers = _families[pos].getIndividuals(age);
                                                        for (std::vector<individuals
*>::iterator i = familyMembers.begin(); i < familyMembers.end(); ++i)
                                                        {
                                                        classmates.push_back(*i);
                                                        }
                                                }
                                                else break;
                                                row++;
                                        }
                                });
                                col++;
                                if (col >= _schools) break;
                }
                g.wait();
        }

        struct linksToAdd
        {
                individuals * ind;
                std::vector<link> l;
        };
        std::vector<linksToAdd> lta;

        //Ora avviene il linking
        for (size_t i = 0; i < classmates.size(); ++i)
        {
                linksToAdd a;
```

```
                a.ind = classmates[i];
                for (size_t j = 0; j < classmates.size(); ++j)
                {
                        if (i != j) a.l.push_back(schoolLink(classmates[j]));
                }
                lta.push_back(a);
        }
        int i(0);
        while (i < lta.size())
        {
                for (int k = 0; k < nThreads; ++k) {
                        g.run(
                                [=, &lta] {
                                        lta[i].ind->addLink(lta[i].l);
                                });
                        i++;
                        if (i >= lta.size()) break;
                }
                g.wait();
        }
}
```

Questi link avranno lo stesso tempo di contatto dei link con la famiglia, una distanza maggiore, ma una frequenza minore, non prevedendo i giorni festivi.

```
schoolLink::schoolLink(individuals * i1)
        :link(i1, 360, 2, lSchool)
{
        _frequency = { 1,1,1,1,1,0,0 };
}
```

Gli individui al di sopra dei 20 anni avranno invece molti più tipi di link:

- *Shopping*: creato a partire dalla media del numero di persone incontrate quando si fa la spesa e da una distribuzione gaussiana che descrive la frequenza, in termini discreti di giorni, con cui un individuo fa la spesa; tale link ha sia una durata che una distanza basse;

```
shoppingLink::shoppingLink(individuals * i1)
         :link(i1, 10, 1, lShooping)
{
         _frequency = { 1,1,1,1,1,1,1 };
}
```

```
void location::createShoopingLink(individuals * ind, int simTime,
std::vector<vlsRandGenerator> & RandomGenerator, const Parameters & p)
{
	// Solo gli adulti
	if (ind->getAge() < 20) return;
	// Troviamo il quartiere
	std::array<int, 2> position = ind->getFamily()->getPosition();
	int x((position[0] - position[0] % _shopping) / _shopping);
	int y((position[0] - position[0] % _shopping) / _shopping);
	int maxPos = static_cast<int>(_families.size()),
fsize(static_cast<int>(sqrt(maxPos)));
	// Shopping
	// Si assume di incontrare (AvgShoppingEncounters) persone, di andare
(AvgShoppingFreq) volte a settimana.
	// Finestra di contaminazione che ritorna i giorni dall'infezione all'ultimo
	aggiornamento di stato della malattia
	std::array<int, 2> time(ind->getContaminationWindow());
	int startTime(time[0]),
nEncounters(static_cast<int>(p(AvgShoppingEncounters)));
	tbb::task_group g;
	// Attraversa la finestra di contaminazione e crea i link
	while (startTime < time[1])
	{
		for (int k = 0; k < nThreads; ++k) {
			vlsRandGenerator * rnd = &(RandomGenerator[k]);
			startTime += (*rnd).poisson(p(AvgShoppingFreq));
			if (startTime >= time[1]) break;
			g.run(
				[=] {
					// Prende coordinate random
					int * coord = (*rnd)(_shopping, 2 *
nEncounters);
					for (int i = 0; i < nEncounters; ++i)
					{
					int * coord = (*rnd)(_shopping, 2 *
nEncounters);
						int pos = (coord[i] +y *
_shopping)*fsize + x * _shopping + coord[i+ nEncounters];
						if (pos < maxPos) {
							individuals *
ptrInd(_families[pos].getHead());
							link l = shoppingLink(ptrInd);
							ind->addLink(l, startTime);
							}
					}
				});
		}
		g.wait();
	}
}
```

- *Work*: creato a partire dal numero di impiegati definiti in fase di generazione del modello, esso ha una durata di contatto maggiore di tutti quelli visti finora, ma una distanza ed una frequenza pari al link scolastico;

```
void location::createWorkLink(individuals * ind, std::vector<vlsRandGenerator> &
RandomGenerator, const Parameters & p)
{
	// check if already has link
	if (ind->hasLink(lWork)) return;

	// Check if employed
	std::array<unsigned int, 3> work(ind->getWork());
	if (work[0] == 0) return;

	struct linksToAdd
	{
		individuals * ind;
		std::vector<link> l;
	};
	std::vector<linksToAdd> lta;
	tbb::task_group g;
	//Let's link them up
	for (size_t i = work[1]; i < work[2]+1; ++i)
	{
		linksToAdd a;
		a.ind = _employables[i];
		for (size_t j = work[1]; j < work[2]+1; ++j)
		{
			if (i != j) a.l.push_back(workLink(_employables[j]));
		}
		lta.push_back(a);
	}
	int i(0);
	while (i < lta.size())
	{
		for (int k = 0; k < nThreads; ++k) {
			g.run(
				[=, &lta] {
				lta[i].ind->addLink(lta[i].l);
			});
			i++;
			if (i >= lta.size()) break;
		}
		g.wait();
	}
}
```

```
workLink::workLink(individuals * i1)
	:link(i1, 480, 2, lWork)
{
	_frequency = { 1,1,1,1,1,0,0 };
}
```

- *Transport*: tale link è creato considerando se l'individuo appena infettato è o meno un lavoratore. Se non lo è, e quindi l'agente utilizza i trasporti pubblici unicamente per uscire per altri scopi, viene considerata la media di incontri nei trasporti pubblici non negli orari di punta e la frequenza con cui un individuo compie tali viaggi, calcolata a partire da una distribuzione di Poisson centrata nella media.

```
int nEncounters(static_cast<int>(p(AvgTransportEncounters)));
tbb::task_group g;
int startTime(time[0]);
int size = static_cast<int>(_families.size());
nEncounters = static_cast<int>(p(AvgTransportEncountersLow));
while (startTime  < time[1])
{
	for (int k = 0; k < nThreads; ++k) {
		vlsRandGenerator * rnd = &(RandomGenerator[k]);
		startTime += (*rnd).poisson(p(AvgTripGoingOutFreq));
		g.run([=] {
			// Prende coordinate random
			int * coord = (*rnd)(size, nEncounters);
			for (int i = 0; i < nEncounters; ++i)
			{
				std::vector<individuals *>
ptrInds(_families[coord[i]].getIndividuals());
				for (std::vector<individuals *>::iterator j =
ptrInds.begin(); j < ptrInds.end(); ++j)
				{
					link l = gointOutTransportLink(*j);
					ind->addLink(l, startTime);
				}
			}
		});
	}
	g.wait();

}
```

Nel caso in cui l'agente lavori viene considerata anche la media del numero di persone incontrare quando si utilizzano trasporti pubblici in orari di punta per creare un diverso tipo di link.

```
bool work(ind->getWork()[0] != 0);
if (work)
{
	int startTime = time[0];
	// Crea link per tutto il periodo di contaminazione
	while (startTime < time[1])
	{
		for (int k = 0; k < nThreads; ++k) {
			vlsRandGenerator * rnd = &(RandomGenerator[k]);
			startTime++;
			if (startTime >= time[1]) break;
			g.run([=] {
				// Prende coordinate random
				int * coord =
(*rnd)(static_cast<int>(_employables.size()), nEncounters);
				for (int i = 0; i < nEncounters; ++i)
				{
					individuals * ptrInd(_employables[coord[i]]);
					if (ind != ptrInd)
					{
						link l = workTransportLink(ptrInd);
						ind->addLink(l, startTime);
					}
				}
			});
		}
		g.wait();
	}
}
```

Un'altra differenza tra i due tipi di link è che per i lavoratori non è attivo nei giorni festivi;

```
goingOutTransportLink::goingOutTransportLink(individuals * i1) :
transportLink(i1, lGoingOutTransport)
{
	_frequency = { 1,1,1,1,1,1,1 };
}
workTransportLink::workTransportLink(individuals * i1) : transportLink(i1,
lWorkTransport)
{
	_frequency = { 1,1,1,1,1,0,0 };
}
```

- *Friend*: questo tipo di link è generato a partire dalle connessioni tra famiglie definite in fase di generazione del modello. Esso considera una distribuzione poissoniana che descrive la frequenza con cui un individuo

incontra i suoi amici e sulla base di un'altra distribuzione di Poisson centrata nel numero medio di amici per incontro crea l'intero set di amici.

```
void location::createFriendLink(individuals * ind, int simTime,
std::vector<vlsRandGenerator> & RandomGenerator, const Parameters & p)
{
	if (ind->hasLink(lFriend)) return;
	// Prendi il tempo di contaminazione
	std::array<int, 2> time(ind->getContaminationWindow());
	int startTime(time[0]);
	int nFriends(ind->getFamily()->getTotalFriends());
	if (nFriends == 0) return;
	tbb::concurrent_vector<family *> familyFriends(ind->getFamily()-
>getFriends());
	std::vector<individuals *> members(ind->getFamily()->getIndividuals());
	tbb::task_group g;
	int prev(startTime);
	while (startTime < time[1])
	{
		for (int k = 0; k < nThreads; ++k) {
			vlsRandGenerator * rnd = &(RandomGenerator[k]);
			if (startTime == time[0]) startTime +=
(*rnd).poisson(p(AvgFriendEncountersFreq)/2);
			else startTime += (*rnd).poisson(p(AvgFriendEncountersFreq));
			std::shuffle(familyFriends.begin(), familyFriends.end(),
std::default_random_engine((*rnd).uniform()));
			g.run([=] {
				int n((*rnd).poisson(p(AvgFriendperEncounters)));
				std::vector<individuals *> f;
				// Seleziona gli amici con cui si è entrati in contatto
durante l'incontro
				for (int i = 0; i < (n > nFriends ? nFriends : n); ++i)
				{
					if (familyFriends[i]->someIsSick(startTime)) {}
					else
					{
						std::vector<individuals *>
fm(familyFriends[i]->getIndividuals());
						for (std::vector<individuals *>::iterator j =
fm.begin(); j < fm.end(); ++j)
						{
							f.push_back(*j);
						}
					}
				}
				struct linksToAdd
				{
					individuals * ind;
					std::vector<link> l;
```

```
                                        int time;
                                };
                                std::vector<linksToAdd> lta;
                                // Linking degli amici
                                // 1. Aggiunge gli amici ai membri della famiglia
                                for (size_t i = 0; i < members.size(); i++)
                                {
                                        linksToAdd a;
                                        a.time = startTime;
                                        a.ind = members[i];
                                        for (size_t j = 0; j < f.size(); j++)
a.l.push_back(friendLink(f[j]));
                                        lta.push_back(a);
                                }
                                // 2. Aggiunge I membri della famiglia agli amici
                                for (size_t i = 0; i < f.size(); i++)
                                {
                                        linksToAdd a;
                                        a.time = startTime;
                                        a.ind = f[i];
                                        for (size_t j = 0; j < members.size(); j++)
a.l.push_back(friendLink(members[j]));
                                        lta.push_back(a);
                                }
                                // 3. Salvataggio
                                for (size_t i = 0; i < lta.size(); i++) for (size_t j = 0; j <
lta[i].l.size(); j++) lta[i].ind->addLink(lta[i].l[j], lta[i].time);
                        });
                }
                g.wait();
        }
}
```

Il link di tipo friend ha una durata e una distanza basse e può avvenire in ogni giorno della settimana.

```
friendLink::friendLink(individuals * i1)
        :link(i1, 180, 1, lFriend)
{
                _frequency = { 1,1,1,1,1,1,1 };

}
```

- *Event*: gli eventi sono assunti essere delle occasioni a cui partecipa ogni membro della famiglia con età compresa tra i 15 ed i 75 anni. La frequenza con cui un individuo partecipa agli eventi è pescata in modo casuale da una distribuzione di Poisson con lambda pari alla media di eventi a settimana a

cui partecipa un individuo, mentre il numero di incontri che avvengono in tale evento è definito da una distribuzione di Poisson basata sulla media di persone che si incontrano ad un evento.

```
void location::createEventLink(individuals * ind, int simTime,
std::vector<vlsRandGenerator> & RandomGenerator, const Parameters & p)
{
	// Prende la finestra di contaminazione
	std::array<int, 2> time(ind->getContaminationWindow());
	int startTime(time[0]);
	int freq(RandomGenerator[0].poisson(p(AvgEventsPerIndividualPerWeek))); if
(freq == 0) return;
	int delayEvents = static_cast<int>(365 / 2) / freq;
	startTime = delayEvents - startTime;
	int
nEncounters(RandomGenerator[0].poisson(p(AvgEventPersons)/p(AvgHousehold)));
	tbb::task_group g;
	while (startTime < time[1])
	{
		for (int k = 0; k < nThreads; ++k) {
			vlsRandGenerator * rnd = &(RandomGenerator[k]);
			startTime += delayEvents;
			if (startTime >= time[1]) break;
			g.run([=] {
				// Prende coordinate random
				int * coord = (*rnd)(static_cast<int>(_families.size()),
nEncounters);
				for (int i = 0; i < nEncounters; ++i)
				{
					std::vector<individuals *>
ptrInds(_families[coord[i]].getIndividuals());// Assume che gli eventi siano una cosa di
famiglia
					for (std::vector<individuals *>::iterator j =
ptrInds.begin(); j < ptrInds.end(); ++j)
					{
						int age(ind->getAge());
						if (15 < age && age < 75) {
							link l = eventLink(*j);
							ind->addLink(l, startTime);
						}
					}
				}
			});
		}
		g.wait();
	}
}
```

Tale tipo di link può avvenire in ogni giorno della settimana, per una durata bassa ed a una distanza maggiore rispetto alla media, cioè pari a 2.

```
eventLink::eventLink(individuals * i1)
	:link(i1, 120, 2, lEvent)
{
	_frequency = { 1,1,1,1,1,1,1 };
}
```

La simulazione vera e propria non è altro che una routine che si ripete fino alla fine dei giorni preimpostati. Per ogni giorno, il modello scorre tutte le location (nel nostro caso ne è una), controlla prima i casi che si sviluppano all'interno della location quel giorno, poi controlla quelli che vengono dall'esterno, ossia le contaminazioni internazionali rappresentate dal paziente zero vengono attivate in questa circostanza, non appena si arriva al giorno specificato casualmente in fase di setup. Il modello eventualmente attiva determinate misure di contenimento al raggiungimento di condizioni determinate dall'utilizzatore.

```
// Run simulation
for (int i = 0; i < max_days; i++)
{
	for (int j = 0; j < _list.size(); j++)
	{
		// First look at the non imported cases
		_list[j].nextDay(i, RandomGenerator, p);

		// Then imported
		std::vector<individuals *> internationalContamination =
_list[j].incidence(&patientZero, i, RandomGenerator[0], p);
		_list[j].addNewlyInfected(internationalContamination, i,
RandomGenerator, p);
	}
}
```

Per ogni individuo infetto all'interno della popolazione, per ogni individuo sano a cui egli è linkato in quel giorno della settimana, il modello controlla se tale link genera o meno un'infezione dell'individuo sano sulla base della distanza,

della durata di contatto e del rischio di contaminazione, considerando, se attivo, il coefficiente di distanziamento sociale (0.15 se attivo, 1 se disattivo).

```
void location::nextDay(int simTime, std::vector<vlsRandGenerator> &
RandomGenerator, const Parameters & p)
{
	std::vector<individuals *> contaminations;
	int index(-1);
	for (std::vector<individuals *>::iterator i = _infected.begin(); i <
_infected.end(); i++)
	{
		index++;
		std::array<int, 2> ContaminationWindow = (*i)-
>getContaminationWindow();
		if (ContaminationWindow[1] < simTime)
		{
			(*i)->clearLinks();
			continue;
		}
		std::vector<individuals *> contaminationsFromAPatient =
incidence(*i, simTime, RandomGenerator[0], p);

		for (std::vector<individuals *>::iterator j =
contaminationsFromAPatient.begin(); j < contaminationsFromAPatient.end(); j++)
		{
			contaminations.push_back(*j);
		}
	}
	addNewlyInfected(contaminations, simTime, RandomGenerator, p);
}
std::vector<individuals*> location::incidence(individuals * ind, int simTime,
vlsRandGenerator & rnd, const Parameters & p)
{
	return ind->contaminates(simTime,rnd,p, _IsolationFlags,
socialDistanceFlag);
}
```

Dove la funzione *contaminates* della classe *individuals* è:

```
std::vector<individuals * > individuals::contaminates(int simTime, vlsRandGenerator
& rnd, const Parameters & p, std::array<int, nLinks> _IsolationFlags, bool
socialDistanceFlag)
{
	std::vector<individuals * > newInfected;
	individuals * ind;
	std::array<int, 2> linkProperties;
	// Giorno della settimana
	int f(simTime % 7);
```

```
        // Recurrent Link
        for (tbb::concurrent_vector<link>::iterator i = _reccurentLink.begin(); i <
_reccurentLink.end(); ++i)
        {
                // Get connected individual
                ind = i->connected(simTime,f);
                if (ind == nullptr) continue;
                if (ind->getAge() >= _IsolationFlags[i->getType()] && !ind-
>wasTestedPositive()) continue;
                // Get link properties
                linkProperties = i->getProperties();
                if (_disease.contaminates(simTime, ind, rnd, p, linkProperties[0],
linkProperties[1],false, socialDistanceFlag))
                {
                        (*i).contaminated();
                        newInfected.push_back(ind);
                }
        }
        // One time at date of simTime
        for (tbb::concurrent_vector<link>::iterator i = indexOneTime; i <
_oneTimeLink.end(); i++)
        {
                // Now sorted by time
                if (i->getTime() > simTime) break;

                // Get connected individual
                ind = i->connected(simTime, f);
                if (ind == nullptr) {
                        indexOneTime = i; continue;
                }
                if (ind->getAge() >= _IsolationFlags[i->getType()] && !ind-
>wasTestedPositive()) {
                        indexOneTime = i; continue;
                }
                // Get link properties
                linkProperties = i->getProperties();
                if (_disease.contaminates(simTime, ind, rnd, p, linkProperties[0],
linkProperties[1], false, socialDistanceFlag))
                {
                        newInfected.push_back(ind);
                }
                indexOneTime = i;
        }
        infectedIndividuals += static_cast<int>(newInfected.size());
        return newInfected;
}
```

Quindi, la funzione cardine che viene richiamata al momento dell'infezione per controllare se essa avviene è *_disease.contaminates*, in cui l'oggetto *_disease*

dell'agente infetto ed infettante "prova ad infettare" un altro *individual* e la *_disease* di quest'ultimo. Se questa condizione sarà verificata, ossia se l'infezione avverrà, l'individuo viene aggiunto al vettore *newInfected*, che successivamente sarà processato dalla funzione *addNewlyInfected*. La funzione *contaminates*, che tra le altre cose tiene conto anche della distanza e la durata dei singoli link, è la seguente.

```
bool disease::contaminates(int simTime, individuals * ind, vlsRandGenerator & rnd, const Parameters & p, int distance, int time, bool hospitalLink, bool _socialDistancing)
{
	// Check if patients is isolated, dead or in the hospital, or cured
	if ((!hospitalLink && hospitalized(simTime)) || isolated(simTime) || dead(simTime) || cured(simTime)) return false;

	// If connection is already sink nothing to do
	if (ind->wasInfected(simTime)) return false;

	// Gradual infectiosity until time of clinical onset
	float contagiosity(1);
	if (simTime < _symptomatic) contagiosity = static_cast<float>(1.0f / pow(2, _symptomatic - simTime));
	// Reduced contagiosity of asymptomatics
	if (!_hasSymptoms) contagiosity = 0.01f;

	// Check if individuals is contaminated
	if (rnd() > (_socialDistancing ? p(socialDistancing) : 1) * p(contaminationRisk) * time * contagiosity / (distance * distance)) return false;

	// Set contamination
	(*ind).contaminated(rnd, p, simTime);
	return true;
}
void individuals::contaminated(vlsRandGenerator & rnd, const Parameters & p, int t)
{
	contaminated(rnd, p, t, 1);
}
void individuals::contaminated(vlsRandGenerator & rnd, const Parameters & p, int t, float reducedDiagnosis)
{
	_disease.contaminated(this, rnd, p, t, reducedDiagnosis);
	if (!(*getFamily()).linkCreated()) (*getFamily()).createLink();
}
```

La funzione *disease::contaminated* è quella che effettivamente stabilisce le caratteristiche dell'infezione appena avvenuta:

```
void disease::contaminated(individuals const * ind, vlsRandGenerator & rnd, const
Parameters & p, int t)
{
	contaminated(ind, rnd, p, t, 1);
}
void disease::contaminated(individuals const * ind, vlsRandGenerator & rnd, const
Parameters & p, int t, float reducedDiagnosis)
{
	_infected = t;
	_isolated = INT_MAX;
	// Set severity (estimated severe in diagnosed case only)
	float r = rnd();
	int age((*ind).getAge());
	age = age + ind->getNumberofComorbidities() * 5; // add 5 year per
comorbidities

	// Pr Severe & Critical
	float ageSevere = age < p(prSevereAgeTh) ? p(prSevereAgeTh) / 100.0f :
static_cast<float>(age) / 100.0f;
	float s0(((p(prAsymp, 0) - (p(prAsymp, 1))*p(prDiagnosed))/10) * age +
p(prAsymp, 1)*p(prDiagnosed)), s1((1/p(PrDeath))*p(prCritical,0) * exp
((static_cast<float>(age)/100.0f) * p(prCritical, 1))), s2(p(prSevere, 0) * ageSevere
* ageSevere + p(prSevere, 1) * ageSevere + p(prSevere, 2));
	// Symptoms
	_symptomatic = static_cast<int>(t + rnd.weibull(p(delaySymptomatic, 0),
p(delaySymptomatic, 1)));
	// Asympt
	s0 = s0 < 0 ? 0 : s0;
	r -= s0;
	if (r < 0) { // Asymptomatic
	}
	else
	{
		// Diagnosed
		_diagnosed = static_cast<int>(_symptomatic +
rnd.weibull(p(delayDiagnosis, 0), p(delayDiagnosis, 1)));
		// Adjust for diagnosed
		float denominator(1);
		r = rnd();
		r = r - s1 * denominator;
		if (r < 0) _severity = 2;
		else if (r < s2 * denominator) _severity = 1;
	}
	// Symptoms
	_hasSymptoms = _severity == 0?rnd() < (p(prSymptomatic)*(1-s1-
s2)):true; // Only mild patient are symptomatic
```

```
        // Hospitalized
        if (_severity > 0) _hospitalized = static_cast<int>(t +
rnd.weibull(p(delayHospitalized, 0), p(delayHospitalized, 1)));
        if (_hospitalized < _diagnosed) _diagnosed = _hospitalized;
        // ICU
        if (_severity == 2) { _icu = _hospitalized+2; }
        // Dead
        // For now we will assume that only critical case die (Based on China CDC
Report)
        if (_severity == 2)
        {
                if (rnd() < p(PrDeath) * p(il9))
                {
                        _dead = static_cast<int>(_icu +
rnd.weibull(p(delayICUDeath, 0), p(delayICUDeath, 1)));
                }
                else
                {
                        _cured = static_cast<int>(_icu +
rnd.weibull(p(delayRecovery, 0), p(delayRecovery, 1)));
                }
        }
        // Cured
        if (_severity < 2) _cured =
static_cast<int>((_severity==0?_symptomatic:_hospitalized) +
+rnd.weibull(p(delayRecovery, 0), p(delayRecovery, 1)));
        if (_cured < _diagnosed) _diagnosed = INT_MAX;
}
```

Al momento dell'infezione vengono dunque definite tutte le caratteristiche della stessa. Innanzitutto viene calcolata la probabilità di essere sintomatici; in caso affermativo, viene definito il momento in cui il paziente diventerà sintomatico, ossia dopo quanti giorni inizierà a presentare sintomi. Tali giorni sono calcolati con una distribuzione di Weibull dai parametri di scala e di forma preimpostati sulla base dei dati raccolti. I giorni di distanza dalla diagnosi vengono calcolati allo stesso modo, cioè aggiungendo al giorno trovato nel passaggio precedente un valore pescato da una distribuzione di Weibull che descrive il delay in giorni tra quando si diventa sintomatici e quando si riceve la diagnosi. Il modello presenta tre gradi di severità della malattia: non severo, severo e critico; solo gli agenti che sviluppano una malattia critica possono morire, all'interno del modello. Conoscendo inoltre la probabilità che un caso sintomatico diventi

critico o severo e la probabilità che un sintomatico muoia, si riesce a determinare in modo random la severità della malattia di tale agente sulla base dell'età. In caso il paziente risulti severo o critico, esso viene ricoverato dopo un totale di giorni ricavato casualmente dalla distribuzione di Weibull che descrive il delay che intercorre tra quando si riceve la diagnosi e quando si viene ospedalizzati.

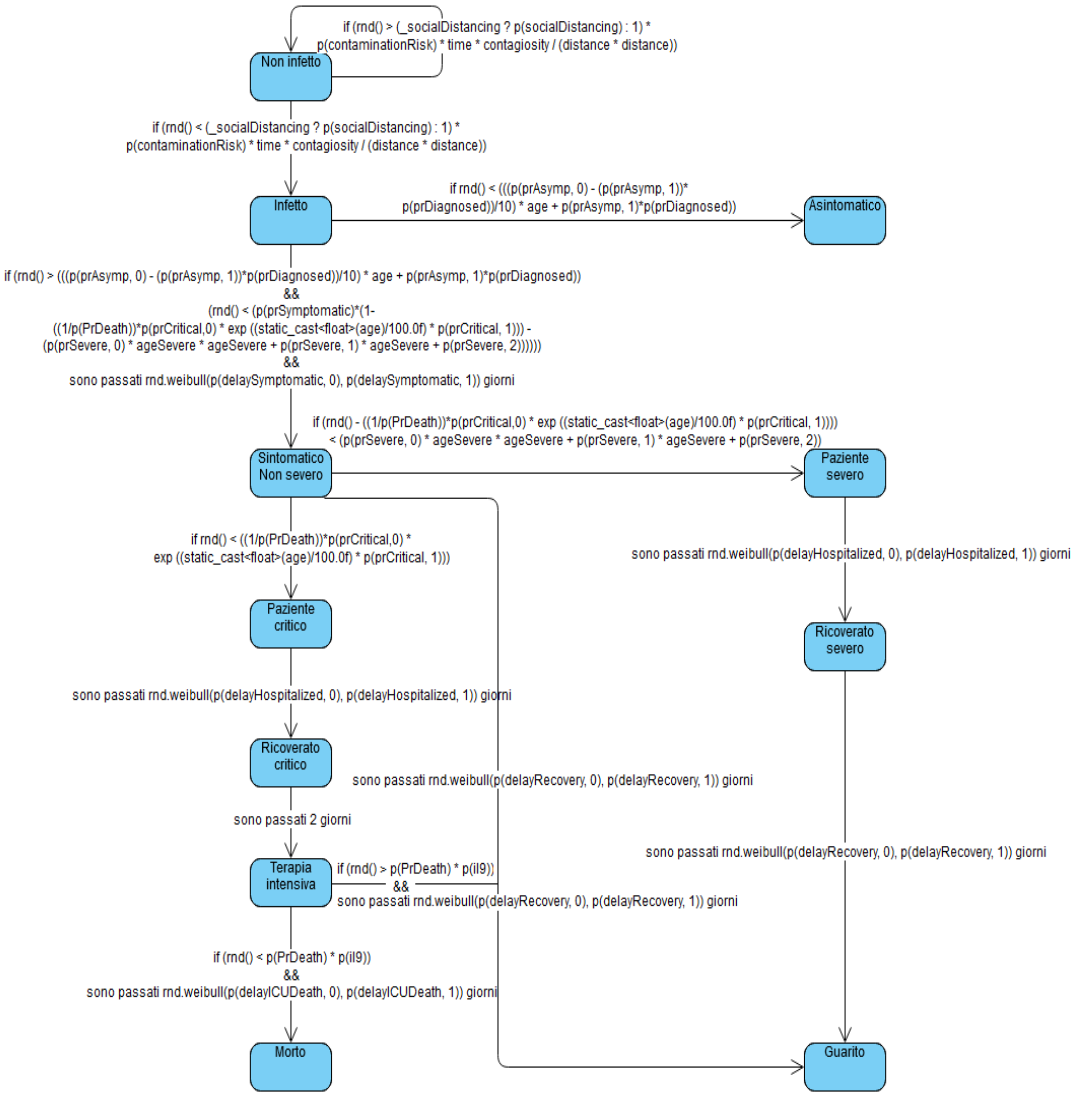

Figura 9: State flow chart di un individuo appena infettato attraverso un qualsivoglia link.

Infine, se il caso è critico il paziente è messo in terapia intensiva, dove, conoscendo la probabilità con cui si muore in terapia intensiva, si stabilisce il

giorno in cui il paziente morirà o verrà definitivamente curato con l'utilizzo, ancora una volta di due distribuzioni di Weibull distinte che descrivono, reciprocamente, il delay che intercorre tra l'entrata in terapia intensiva e la morte o il delay tra l'entrata in terapia intensiva e l'avvenuta guarigione; se il caso non è critico, viene calcolato il giorno di guarigione pescando un valore da una distribuzione di Weibull descrittiva del delay che intercorre tra l'entrata in ospedale (per casi severi) o l'avvenuta diagnosi (per casi non severi) e la guarigione.

In Figura 9 è riassunto questo processo stocastico di assegnazione dei parametri della progressione della malattia a partire da un contatto con un individuo qualsiasi su di un qualsivoglia link, che può, come evidenziato dal primo passaggio dello state flow chart, passare o meno l'infezione. Al cambiare del link, infatti, cambiano le variabili *distance* e *duration*, che appaiono nella formula che descrive la probabilità con cui si viene infettati.

Al momento dell'infezione di un individuo per qualsiasi mezzo, dunque, il modello già definisce quale dovrà essere la sua storia, che tipo di malattia svilupperà l'agente, con quale severità e dopo quanti giorni verrà diagnosticato, ospedalizzato se severo, spostato in terapia intensiva se critico e conoscerà anche dopo quanti giorni l'individuo guarirà o morrà. Dunque, al momento dell'infezione, viene aggiunto il contributo unitario dell'agente ai rispettivi counter, divisi per giorni e che tengono conto del numero di infetti, di diagnosticati, di ricoverati, di persone in terapia intensiva, di persone con bisogno di andare in terapia intensiva, di persone curate ed, infine, di persone morte. Questo avviene nella funzione *addNewlyInfected*, richiamata al termine di *NextDay*, in cui viene passato come argomento il vettore di nuovi infetti trovato con la funzione *incidence*, dunque, il software eseguirà la funzione *addNewlyInfected*:

```
void location::addNewlyInfected(std::vector<individuals*> infected, int
```

```
simTime, std::vector<vlsRandGenerator> & RandomGenerator, const
Parameters & p)
{
	// For each newly infected individuals
	std::vector<individuals*>::iterator i = infected.begin();
	while (i < infected.end())
	{
		int age((*i)->getAge());
		int sex((*i)->getSex());
		// Create links
		if (age > 20)
		{
			createShoopingLink((*i), simTime,
RandomGenerator, p);
			createWorkLink((*i), RandomGenerator, p);
			createTransportLink((*i), simTime,
RandomGenerator, p);
			createFriendLink((*i), simTime,
RandomGenerator, p);
			createEventLink((*i), simTime,
RandomGenerator, p);
		}
		else
		{
			createSchoolLink((*i), RandomGenerator, p); //
Check if school are isolated
		}
		// sort the oneTime
		(*i)->sortOneTimeLink();
		// Tracking
		std::array<std::array<int, 2>, 4> track = (*i)-
>getTracking();
		// Count
		_incidence[sex * 100 + age][simTime]++;
		if (track[0][0] < max_days) _diagnosed[sex * 100 +
age][track[0][0]]++;
		// ICU
		if (track[2][0] != INT_MAX) // Needs ICU
		{
			for (int i = track[2][0]; i < (track[2][1] <
max_days ? track[2][1] : max_days); i++) _needICU[sex * 100 +
age][track[2][0]]++;
			if (_ICUAvailable[track[2][0]] > ICUBeds) // No
beds available
			{
				(*i)->noICU(track[2][0]);
				track = (*i)->getTracking(); // Reload with
death
			}
			else
			{
```

```
                                        for (int i = track[2][0]; i < (track[2][1] <
max_days ? track[2][1] : max_days); i++) _ICU[sex * 100 + age][i]++;
                                        for (int i = track[2][0]; i < (track[2][1] <
max_days ? track[2][1] : max_days); i++) _ICUAvailable[i]++;
                                    }
                            }
                            if (track[1][0] != INT_MAX) for (int i = track[1][0]; i <
(track[1][1] < max_days? track[1][1]: max_days); i++) _hospitalized[sex * 100
+ age][i]++;
                            if (track[3][0] < max_days) _cured[sex * 100 +
age][track[3][0]]++;
                            if (track[3][1] < max_days) _dead[sex * 100 +
age][track[3][1]]++;

                            _infected.push_back(*i);
                              i++;
        }
}
std::array<std::array<int, 2>, 4> individuals::getTracking() const

{
        return _disease.getTracking();
}
std::array<std::array<int, 2>, 4> disease::getTracking() const
{
        int end(_dead > _cured ? _cured : _dead);
        return std::array<std::array<int, 2>, 4>({ { {_diagnosed, end
},{_hospitalized, end},{_icu, end},{_cured, _dead }} });
}
// _diagnosed = giorno in cui è stata diagnosticata la malattia, _cured giorno in
cui è stata curata, _dead in cui è morto il paziente, _hospitalized giorno in cui
l'individuo è stato ricoverato, _icu giorno in cui l'individuo è andato in terapia
intensiva. Altrimenti INT_MAX
```

Ogni giorno, inoltre, se il tracking è attivo, il modello esegue test anche su individui asintomatici e, per ogni agente infetto, esamina tutti i contatti (link) che quest'ultimo ha avuto dal momento dell'infezione e controlla che ognuno degli individui con cui ha avuto contatti abbia o meno contratto la malattia, eseguendo la diagnosi il giorno successivo.

```
void location::enableTracking(int i)
{
        trackingCases = true;
        if (i == 1) testOnlySymptomatic = false;
}
```

```
void location::trackCases(int simTime, std::vector<vlsRandGenerator>&
RandomGenerator, const Parameters & p)
{
	if (!trackingCases) return; // Stop tracking,ex. if too many cases

	tbb::task_group g;
	size_t i(trackPosition), j(trackPosition);
	std::vector<size_t> jpos;
	for (int k = 0; k < nThreads; ++k) {
		jpos.push_back(trackPosition);
	}
	while (i < _infected.size())
	{
		for (int k = 0; k < nThreads; ++k) {

			vlsRandGenerator * rnd = &(RandomGenerator[k]);

			jpos[k] = i;

			g.run(
				[=, &jpos] {
				trackPosition = i;
				std::array<int, 3> d((_infected[i])->getDiagnosed());
// Get infection and diagnosed date
				if (d[0] == 1 || d[2] == INT_MAX)
				{
					return; // Already tracked or never
diagnosed;
				}
				else if (d[2] > simTime)
				{
					jpos[k] = (i > jpos[k]) ? jpos[k] : i;
					return; // Not yet diagnosed, may need to
come back to him
				}
				else
				{
					// Isolate individual
					_infected[i]->isolate(simTime);
					_infected[i]->setTracked();

					// Look for contacts
					int delay(1);
					std::vector<individuals*> contacts =
(_infected[i])->trackedInfected(d[1]+delay, simTime, _CanTrack, _IsolationFlags);
// Get all between infection date and diagnosis

					// For each contact, let test at +delay of
diagnosis
					for (std::vector<individuals*>::iterator j =
contacts.begin(); j < contacts.end(); j++)
```

```
                                        {
                                                (*j)->test(d[2] + delay,
testOnlySymptomatic, true, (*rnd), p);
                                        }
                                }
                        });
                        g.wait();
                                i++;
                                if (i >= _infected.size()) break;
                        }
        }
        trackPosition = i;
        for (int k = 0; k < nThreads; ++k) { if (jpos[k] != 0) trackPosition =
(trackPosition > jpos[k]) ? jpos[k] : trackPosition; }

}
```

Dove *trackedInfected* restituisce il vettore di individui che un individuo ha incontrato fino al giorno corrente:

```
std::vector<individuals*> individuals::trackedInfected(int s, int e,
std::array<bool, nLinks> canTrack, std::array<int, nLinks> _IsolationFlags)
{
        std::vector<individuals*> contacts;
        individuals * ind;
        // Recurrent Link
        for (tbb::concurrent_vector<link>::iterator i = _reccurentLink.begin(); i
< _reccurentLink.end(); ++i)
        {
                // Get connected individual
                ind = i->contactBetween(s, e);
                if (ind == nullptr) continue;
                contacts.push_back(ind);
        }

        // One time at date of simTime
        for (tbb::concurrent_vector<link>::iterator i = _oneTimeLink.begin(); i
< _oneTimeLink.end(); i++)
        {
                int ltime(i->getTime());
                // Now sorted by time
                if (ltime < s) continue;
                if (ltime > e) break;

                // Get connected individual
                if (!canTrack[i->getType()]) continue;
                ind = i->getPair();
                if (ind == nullptr) continue;
                contacts.push_back(ind);
```

```
        }

        return contacts;
}

individuals * link::contactBetween(int s, int e) const
{
        if (!_active) return nullptr;
        if (_simTime == -1) // Regular contacts
        {
                for (int i = s; i < e; i++)
                {
                        if (_frequency[(i % 7)] == 1) return _pair;
                }
                return nullptr;
        }
        else if (s <= _simTime && _simTime <= e) return _pair; // OneTime
        return nullptr;
}
```

Al momento della diagnosi di un individuo, questo viene messo in isolamento e vengono messi in isolamento tutti gli agenti testati positivi alla PCR, ossia non possono più contaminare nessuno tramite i suoi link. La funzione di test infatti viene chiamata con il parametro *isolate = true* e con *symptoms = true* se il tracking è abilitato. Le funzioni test e isolate sono riportate di seguito:

```
bool individuals::test(int simTime, bool symptoms, bool isolate,
vlsRandGenerator & rnd, const Parameters & p)
{
        if ((symptoms && _disease.isSymptomatic(simTime)) || !symptoms)
        {
                if (rnd() < p(PCRSensitivity) /*&& rnd() < p(prSuccessTrack)*/)
return _disease.test(simTime, isolate);
        }
        return false;
}

bool disease::test(int simTime, bool isolate)
{
        if (dead(simTime) || cured(simTime)) return false;
        if (simTime > _infected)
        {
                _diagnosed = simTime + 1;
                if (isolate) _isolated = simTime;
                return true;
        }
        return false;
```

```
}
void individuals::isolate(int simTime)
{
        _disease.setIsolate(simTime);
}
void disease::setIsolate(int t)
{
        _isolated = t;
}
```

L’utilizzatore del modello deve quindi specificare quali misure di prevenzione e controllo intende adottare all’interno del modello.

Capitolo 4: Simulazioni sul territorio campano

Utilizzando il modello appena introdotto, allo scopo di testare l'efficacia e capire il funzionamento di tale modello, sono state svolte 20 simulazioni della prima ondata di Covid-19 utilizzando dei parametri propri del territorio campano.

4.1 I dati

I parametri utilizzati sono sintetizzati nelle seguenti tabelle. I dati sono stati ricavati da [40][41].

Parametro	**Valore ricercato**	**Valori calcolati**	**Fonte**
Sesso in un nucleo familiare single	3676 uomini, 4884 donne	0.4294392523	ISTAT nazionale 2019
Struttura familiare	557 single, 377 solo coppia, 916 coppia con bambini, 311 genitore singolo	0.257751041 single, 0.174456270 coppia, 0.423877835 coppia con bambini, 0.143914854	ISTAT regionale 2019

		genitore singolo	
Età del primo figlio	307 di anni <5, 462 di anni 6-13, 284 di anni 18-24, 1288 di anni >25	23.41	ISTAT nazionale 2019
Numero di figli[1]	361 uno, 412 due, 143 tre e più	1.91812	ISTAT regionale 2019
Età della donna alla nascita del primo figlio	31.02 media 5.34 deviazione		ISTAT regionale 2017, Deviazione assunta
Numero medio di persone in un nucleo[2]**	8562 1 membro, 6967 2 membri, 4954 3 membri, 3872 4 membri, 1023 5 membri, 337 6 o più membri	2.26544	ISTAT nazionale 2019
Grandezza media di una classe scolastica	totale iscritti 960507, totale classi 50672	18.95538	ISTAT regionale 2019
Proporzione di piccole aziende	699884 totali, 669478 con meno di 10 dipendenti	0.956556	ISTAT regionale 2018

[1] Per il calcolo del numero di figli è stata fatta la media pesata considerando 143 "tre e più" come 143/2 tre, 143/4 quattro, 143/8 cinque, etc.
[2] Per il calcolo del numero di persone per nucleo familiare è stata fatta la media pesata considerando 337 "6 o più membri" come 337/2 sei, 337/4 sette, 337/8 otto, etc.

Tasso di occupazione tra i 20 e i 65 anni	710299.09 lavoratori da 15 anni in su, 3541087 popolazione Campania 20-65	0.2005878675 (assunto 0.5 per il lavoro nero)	ISTAT regionale 2017, ISTAT regionale 2020
Densità posti letto in terapia intensiva	5.8 per 100'000 abitanti	0.000058	AGENAS

Parametro	**Età uomo single**	**Età donna single**	**Età della donna nella coppia**
Valore ricercato	1207 di anni 20-44, 1369 di anni 45-64, 1100 di anni 65-99	685 di anni 20-44, 1253 di anni 45-64, 2946 di anni 65-99	23 di anni 15-34, 58 di anni 35-54, 106 di anni 55-64, 190 di anni 65-99
Valori calcolati	0.19 0 0.20 0.0131338 0.21 0.0262677 0.22 0.0394015 0.23 0.0525354 0.24 0.0656692 0.25 0.078803 0.26 0.0919369 0.27 0.105071 0.28 0.118205 0.29 0.131338 0.30 0.144472 0.31 0.157606	0.19 0 0.20 0.00561016 0.21 0.0112203 0.22 0.0168305 0.23 0.0224406 0.24 0.0280508 0.25 0.0336609 0.26 0.0392711 0.27 0.0448812 0.28 0.0504914 0.29 0.0561016 0.30 0.0617117 0.31 0.0673219	0.19 0 0.20 0.0040672 0.21 0.00813439 0.22 0.0122016 0.23 0.0162688 0.24 0.020336 0.25 0.0244032 0.26 0.0284704 0.27 0.0325376 0.28 0.0366048 0.29 0.040672 0.30 0.0447392 0.31 0.0488064

	0.32 0.17074	0.32 0.072932	0.32 0.0528736
	0.33 0.183874	0.33 0.0785422	0.33 0.0569408
	0.34 0.197008	0.34 0.0841523	0.34 0.061008
	0.35 0.210141	0.35 0.0897625	0.35 0.0687003
	0.36 0.223275	0.36 0.0953726	0.36 0.0763926
	0.37 0.236409	0.37 0.100983	0.37 0.0840849
	0.38 0.249543	0.38 0.106593	0.38 0.0917772
	0.39 0.262677	0.39 0.112203	0.39 0.0994695
	0.40 0.275811	0.40 0.117813	0.40 0.107162
	0.41 0.288945	0.41 0.123423	0.41 0.114854
	0.42 0.302078	0.42 0.129034	0.42 0.122546
	0.43 0.315212	0.43 0.134644	0.43 0.130239
	0.44 0.328346	0.44 0.140254	0.44 0.137931
	0.45 0.346967	0.45 0.153081	0.45 0.145623
	0.46 0.365588	0.46 0.165909	0.46 0.153316
	0.47 0.384208	0.47 0.178737	0.47 0.161008
	0.48 0.402829	0.48 0.191564	0.48 0.1687
	0.49 0.42145	0.49 0.204392	0.49 0.176393
	0.50 0.440071	0.50 0.217219	0.50 0.184085
	0.51 0.458692	0.51 0.230047	0.51 0.191777
	0.52 0.477312	0.52 0.242875	0.52 0.199469
	0.53 0.495933	0.53 0.255702	0.53 0.207162
	0.54 0.514554	0.54 0.26853	0.54 0.214854
	0.55 0.533175	0.55 0.281357	0.55 0.242971
	0.56 0.551795	0.56 0.294185	0.56 0.271088
	0.57 0.570416	0.57 0.307013	0.57 0.299204
	0.58 0.589037	0.58 0.31984	0.58 0.327321
	0.59 0.607658	0.59 0.332668	0.59 0.355438
	0.60 0.626279	0.60 0.345495	0.60 0.383554

	0.61 0.644899	0.61 0.358323	0.61 0.411671
	0.62 0.66352	0.62 0.371151	0.62 0.439788
	0.63 0.682141	0.63 0.383978	0.63 0.467905
	0.64 0.700762	0.64 0.396806	0.64 0.496021
	0.65 0.709311	0.65 0.41404	0.65 0.510421
	0.66 0.717861	0.66 0.431274	0.66 0.52482
	0.67 0.726411	0.67 0.448508	0.67 0.539219
	0.68 0.73496	0.68 0.465742	0.68 0.553619
	0.69 0.74351	0.69 0.482976	0.69 0.568018
	0.70 0.75206	0.70 0.500211	0.70 0.582418
	0.71 0.760609	0.71 0.517445	0.71 0.596817
	0.72 0.769159	0.72 0.534679	0.72 0.611216
	0.73 0.777709	0.73 0.551913	0.73 0.625616
	0.74 0.786258	0.74 0.569147	0.74 0.640015
	0.75 0.794808	0.75 0.586381	0.75 0.654415
	0.76 0.803358	0.76 0.603615	0.76 0.668814
	0.77 0.811907	0.77 0.620849	0.77 0.683213
	0.78 0.820457	0.78 0.638084	0.78 0.697613
	0.79 0.829007	0.79 0.655318	0.79 0.712012
	0.80 0.837556	0.80 0.672552	0.80 0.726412
	0.81 0.846106	0.81 0.689786	0.81 0.740811
	0.82 0.854656	0.82 0.70702	0.82 0.75521
	0.83 0.863205	0.83 0.724254	0.83 0.76961
	0.84 0.871755	0.84 0.741488	0.84 0.784009
	0.85 0.880305	0.85 0.758722	0.85 0.798408
	0.86 0.888854	0.86 0.775956	0.86 0.812808
	0.87 0.897404	0.87 0.793191	0.87 0.827207
	0.88 0.905954	0.88 0.810425	0.88 0.841607
	0.89 0.914503	0.89 0.827659	0.89 0.856006

	0.90 0.923053 0.91 0.931603 0.92 0.940152 0.93 0.948702 0.94 0.957252 0.95 0.965801 0.96 0.974351 0.97 0.982901 0.98 0.99145 0.99 1 1 1	0.90 0.844893 0.91 0.862127 0.92 0.879361 0.93 0.896595 0.94 0.913829 0.95 0.931064 0.96 0.948298 0.97 0.965532 0.98 0.982766 0.99 1 1 1	0.90 0.870405 0.91 0.884805 0.92 0.899204 0.93 0.913604 0.94 0.928003 0.95 0.942402 0.96 0.956802 0.97 0.971201 0.98 0.985601 0.99 1 1 1
CDF polinomiale[3]	17.743824870958x^5 -47.54140429285x^4 45.299341880017x^3 -18.67404172359x^2 4.6555485175631x -0.45880771178557	5.3177736029607x^5 -18.42458270995x^4 22.98384147608x^3 -11.62698567296x^2 3.0581167655023 x -0.296163697476534	35.3075857712395x^5 -106.82215899298x^4 119.377654254708x^3 -59.430227975168x^2 13.7671949411599x -1.17547758835396

Tabelle 1 e 2: Dati relativi alla popolazione campana

La simulazione è stata eseguita prevedendo nessuna misura di lockdown o distanziamento sociale fino al presentarsi delle prime morti [20][47] (ordinanza dell'11/03/2020). Quando queste si presentano, il modello attiva misure di distanziamento sociale e di lockdown (ossia isola ogni link) per tutte le età per la durata di due mesi e, dopo questo periodo, il lockdown viene rilasciato unicamente per i lavoratori; contemporaneamente viene attivato il testing ed il tracking degli asintomatici [47] (ordinanze n° 40-41-42) [48] (dpcm del 17/05/2020 e del 18/05/2020). Il modello è stato utilizzato usando come parametri "*C:\MyDirectory 1 500000*", ossia su una località di 500000 agenti.

[3] Per il calcolo della CDF polinomiale a partire dal set di dati è stato usato il metodo di regressione polinomiale.

```
for (int i = 0; i < max_days; i++)
{
	for (int j = 0; j < _list.size(); j++)
	{
		// First look at the non imported cases
		_list[j].nextDay(i, RandomGenerator, p);
		// Then imported
		std::vector<individuals *> internationalContamination =
_list[j].incidence(&patientZero, i, RandomGenerator[0], p);
		_list[j].addNewlyInfected(internationalContamination, i,
RandomGenerator, p);
		float dead(_list[j].getNumberDead(i));
		_list[j].trackCases(i, RandomGenerator, p);
		// Containement
		//MIO
		if (dead > 0.1 && stopContainement == INT_MAX)
		{
			stopContainement = i;
			_list[j].containement(0, lEvent);
			_list[j].containement(0, lFriend);
			_list[j].containement(0, lWorkTransport);
			_list[j].containement(0, lGoingOutTransport);
			_list[j].containement(0, lWork);
			_list[j].containement(0, lSchool);
			_list[j].containement(0, lForeign);
			_list[j].disableTracking();
			_list[j].setSocialDistancing(true);
		}
		if (i == stopContainement + 60){
			_list[j].containement(INT_MAX, lWorkTransport);
			_list[j].containement(INT_MAX, lWork);
			_list[j].enableTracking(true);
			_list[j].setTrackingCapability(lWorkTransport, true);
			_list[j].setTrackingCapability(lGoingOutTransport, true);
			_list[j].setTrackingCapability(lShooping, true);
		}

	}
}
```

Dove *getNumberDead()* è stata definita come:

```
float location::getNumberDead(int simTime) const
{
	float a = 0.0f;
	for (int j = 0; j < simTime + 1; j++) for (int i = 0; i < 200; i++) a +=
_dead[i][j];
	return 100000.0f * a / population;
}
```

I risultati delle 20 simulazioni sono quindi stati sintetizzati applicando alle curve risultanti la media aritmetica punto per punto. L'esito delle simulazioni si presenta quindi così.

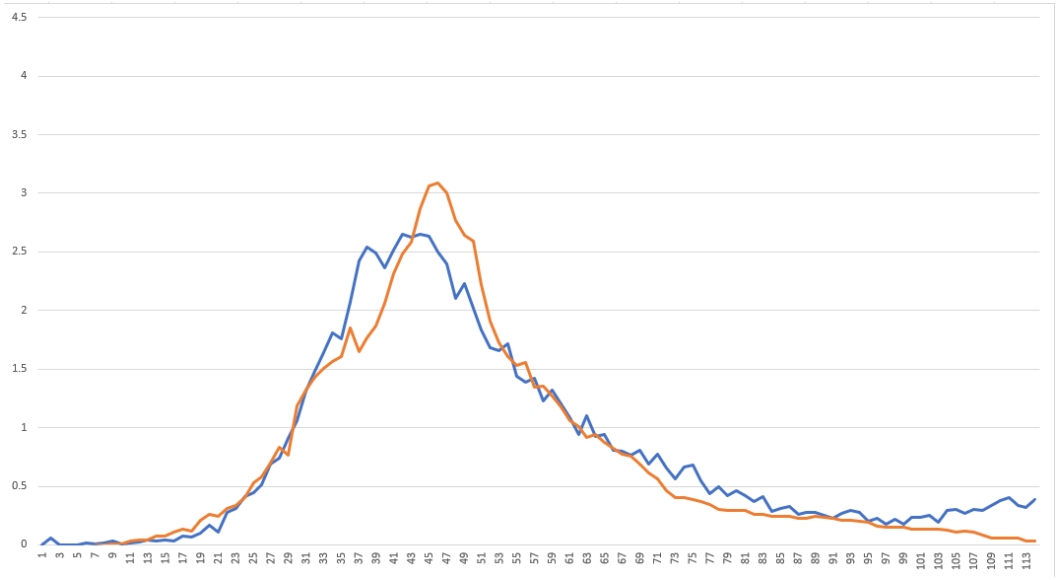

Figura 10: in arancione la media mobile su 7 periodi della curva dei diagnosticati giornalieri in Campania; in blu le previsioni del modello.

I valori sono espressi in numero di persone per ogni 100000 individui. Questo significa che il valore calcolato dal modello di incidenza reale della malattia sulla popolazione raggiunge il valore di 100 persone ogni 100000.

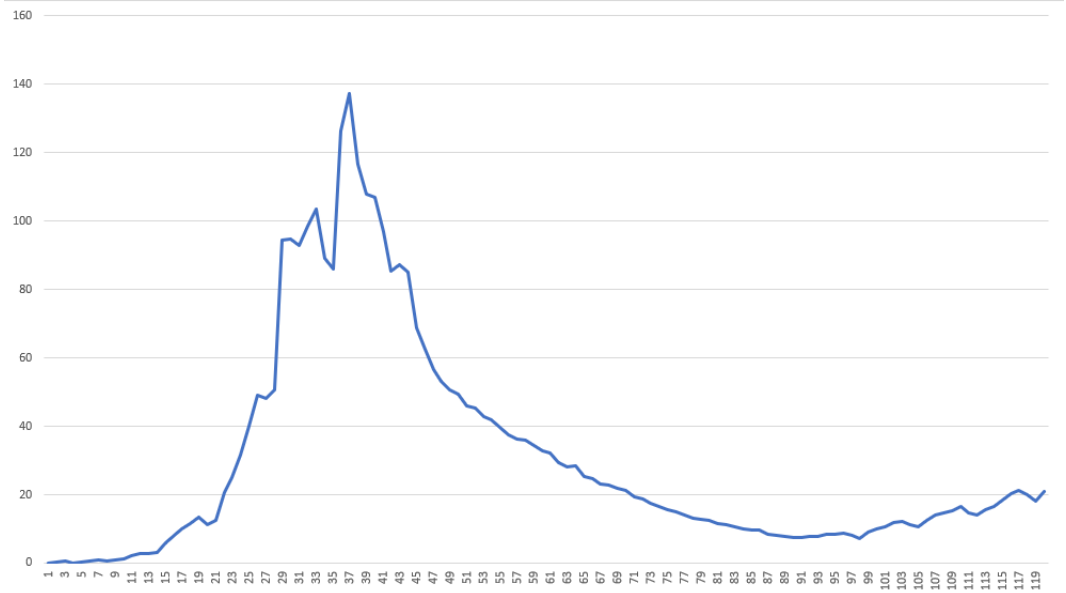

Figura 11: incidenza prevista dal modello

Invece, le morti non risultano coincidere, risultando in un volume molto maggiore di morti all'interno del modello rispetto ai dati prelavati dalla realtà [20].

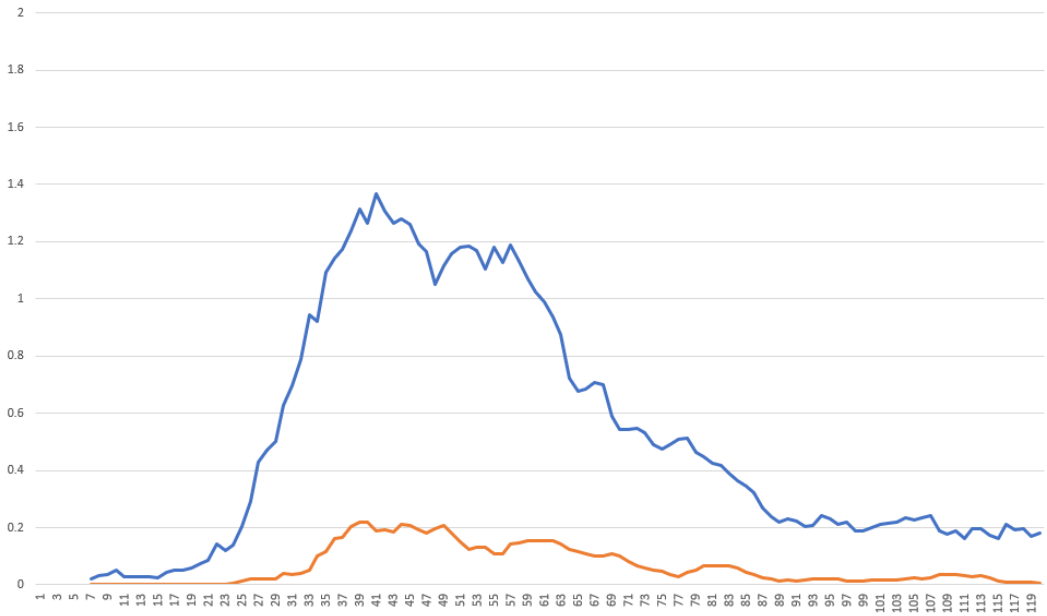

Figura 12: in arancione la media mobile su 7 periodi della curva dei morti giornalieri in Campania; in blu le previsioni del modello.

4.2 Provincia di Napoli

L'analisi fatta considerando la regione Campania può essere estesa alla Provincia di Napoli senza necessità di cambiare parametri. Infatti, anche recuperando gli unici parametri utili e più dettagliati per la provincia da [40], essi risultano circa pari a quelli della Campania:

Proporzione di piccole aziende	363060 totali, 345802 con meno di 10 dipendenti	0.952465	ISTAT provincia 2018

Tasso di occupazione tra i 20 ed i 65 anni	407507.91 lavoratori da 15 anni in su, 1879014 popolazione 20-65	0.21687 (assunto 0.5 per il lavoro nero)	ISTAT provincia 2017, ISTAT provincia 2020

Tabella 3: dati relativi alla popolazione napoletana

Quindi, il confronto tra i diagnosticati dal modello ed i diagnosticati – unico dato presente per la provincia di Napoli - durante la prima ondata si presenta come segue.

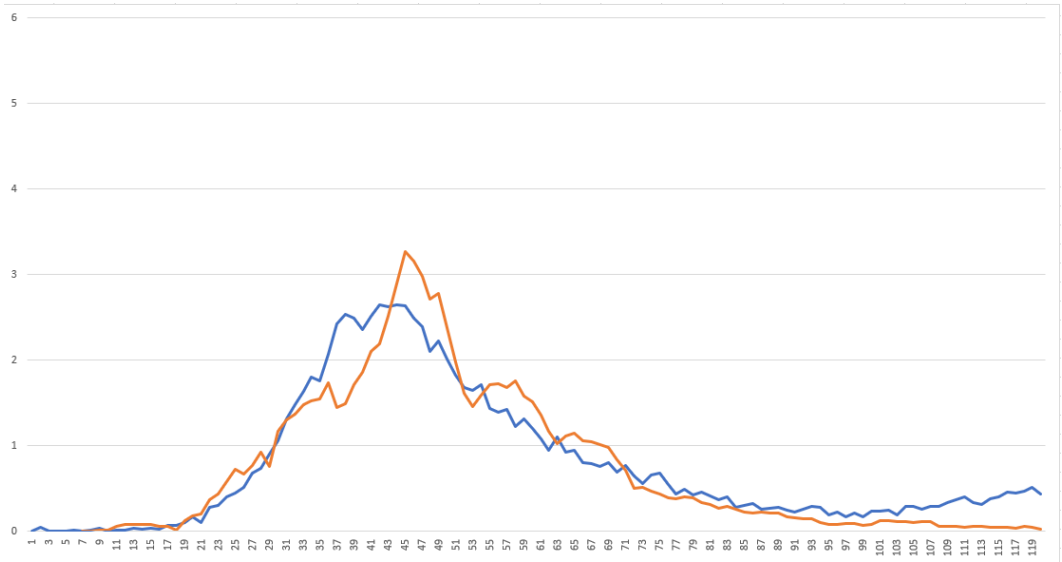

Figura 13: in arancione la media mobile su 7 periodi della curva dei diagnosticati giornalieri nella provincia di Napoli; in blu le previsioni del modello.

Conclusioni

Il modello ad agenti stocastico risulta essere molto più preciso di un modello compartimentale standard nelle sue previsioni sulle diagnosi e l'incidenza del virus. Tuttavia, le morti non risultano coincidere in numero, risultando essere molto più elevate nel modello.

Un'importante nota riguarda la data di creazione di tale modello, posteriore alla prima ondata e con lo scopo di esaminare misure alternative al lockdown per le successive [10]. Pertanto, sono considerati al suo interno parametri molto più dettagliati di quelli cui erano a conoscenza gli operatori sanitari ed il personale medico durante la prima ondata dell'epidemia, nonché è considerata molto più ampia la capacità di disporre di tamponi e mascherine. Basti pensare che nei primissimi giorni, in Campania, venivano utilizzati 10 tamponi al giorno [20], le mascherine chirurgiche si trovavano difficilmente ed inizialmente non erano obbligatorie, inoltre non si effettuavano test su asintomatici o con sintomi lievi.

Tale modello e sue versioni aggiornate e modificate (non presenti in [43]) vengono tuttora usate anche per definire e studiare delle strategie di vaccinazione, oltre che di contenimento [49].

Bibliografia

[1] World Health Organization Covid-19, https://covid19.who.int/, 22/02/2021

[2] FORRESTER Jay W. "***World Dynamics***." Cambridge: Wright-Allen Press, pp. 17-30, 1971.

[3] KERMACK William Ogilvy, MCKENDRICK Anderson Gray. "***A contribution to the mathematical theory of epidemics***." Proc. R. Soc. Lond. A115. pp. 700–721. 1927.

[4] BRAUER Fred. "***Mathematical epidemiology: Past, present, and future, Infectious Disease Modelling.***" Volume 2, Issue 2, pp. 113-127. May 2017.

[5] VICKERS David, OSGOOD Nathaniel "***A unified framework of immunological and epidemiological dynamics for the spread of viral infections in a simple network-based population***", Theoretical Biology and Medical Modelling volume 4, Article number: 49, 2007

[6] RAHMANDAD Hazhir, STERMAN John "***Heterogeneity and Network Structure in the Dynamics of Diffusion: Comparing Agent-Based and Differential Equation Models***", Management Science, Volume 54, Issue 5. 1 May 2008. Pp. 998-1014.

[7] EAMES K. T. D., KEELING M. J. "***Contact tracing and disease control***". Proc. R. Soc. Lond. B. Vol. 270: pp. 2565–2571. 2003.

[8] HUERTA Ramon and TSIMRING Lev S. "***Contact tracing and epidemics control in social networks***". Phys. Rev. E 66, Vol. 66, Iss. 5, article number 056115, 19 November 2002

[9] HYMAN James, LI Jia M. STANLEY E.Ann, "***Modeling the impact of random screening and contact tracing in reducing the spread of HIV***". Mathematical Biosciences Volume 181, Issue 1, Pages 17-54, January 2003

[10] HOERTEL Nicolas, BLACHIER Martin, BLANCO Carlos, OLFSON Mark, MASSETTI Marc, SANCHEZ RICO Marina, LIMOSIN Frédéric, LELEU Henri "***A stochastic agent-based model of the SARS-CoV-2 epidemic in France***". *Nat Med* **26,** pp. 1417–1421 (2020).

[11] STERMAN John. "***Business dynamics: Systems thinking and modeling for a complex world.***". Mc-Graw Hill. Pp. 191-199. 2000.

[12] System Dynamics Society. https://systemdynamics.org/what-is-system-dynamics/, 22/02/2021

[13] CHOISY M, J.-GUEGAN F., ROHANI P. "Mathematical Modeling of Infectious Diseases Dynamics", ENCYCLOPEDIA OF INFECTIOUS DISEASES: MODERN METHODOLOGIES, chapter 22, editor Michel Tibayrenc, 2006.

[14] Documentazione libreria thread https://en.cppreference.com/w/cpp/thread/thread/hardware_concurrency, 22/02/2021

[15] HUBBS Christian, "***Social Distancing to Slow the Coronavirus***", https://towardsdatascience.com/social-distancing-to-slow-the-coronavirus-768292f04296, 22/02/2021

[16] MWALILI S., KIMATHI M., OJIAMBO, V. et al. "***SEIR model for COVID-19 dynamics incorporating the environment and social distancing***." BMC Res Notes vol. 13, 352 (2020)

[17] GRIMM Veronika, MENGEL Friederike, SCHMIDT Martin "***Extensions of the SEIR Model for the Analysis of Tailored Social Distancing and Tracing Approaches to Cope with COVID-19***", medRxiv. 20078113. 24/04/2020

[18] FALCO Ivanoe, DELLA CIOPPA Antonio, SCAFURI U., TARANTINO Ernesto, "***Coronavirus Covid--19 spreading in Italy: optimizing an epidemiological model with dynamic social distancing through Differential Evolution***". arXiv:2004.00553v3. 01/04/2020

[19] LIU Y, GAYLE AA, WILDER-SMITH A, ROCCKLOY J. "***The reproductive number of COVID-19 is higher compared to SARS coronavirus***." J Travel Med. 27(2): 021. 2020 Mar 13.

[20] Repository della protezione Civile, https://github.com/pcm-dpc/COVID-19, 22/02/2021

[21] Ministero della salute, http://www.salute.gov.it/portale/nuovocoronavirus/dettaglioNotizieNuovoCoronavirus.jsp?lingua=italiano&menu=notizie&p=dalministero&id=4763, 22/02/2021

[22] GARNETT GP, ANDERSON RM. "***Sexually transmitted diseases and sexual behavior: insights from mathematical models.***" J Infect Dis. 174 Suppl 2: S150-161. 1996 Oct

[23] GRENFELL BT, BJORNSTAD ON, KAPPEY J. "***Travelling waves and spatial hierarchies in measles epidemics***". Nature. 414(6865): 716-723. 2001 Dec 13

[24] TIAN Yuan, ALAWAMI Fatima, AL-AZEM Assaad, OSGOOD Nathaniel, HOEPPNER Vernon, DUTCHYN Chris, 2011/12/01, pp. 1362-1373 "***A System Dynamics model of tuberculosis diffusion with respect to contact tracing investigation***" 10.1109/WSC. 6147857 Proceedings - Winter Simulation Conference. 2011.

[25] OSGOOD Nathaniel, "***Representing Heterogeneity in Complex Feedback System Modeling: Computational Resource and Error Scaling***", 2004/01/01

[26] OSGOOD Nathaniel, MAHAMOUD Aziza, LICH Kristen, TIAN Yuan, AL-AZEM Assaad, HOEPPNER Vernon, "***Estimating the Relative Impact of Early-Life Infection Exposure on Later-Life Tuberculosis Outcomes in a Canadian Sample***", Research in Human Development vol. 8, pp. 26-47. 2011/01/01

[27] GILBERT Nigel, "***Agent-Based Models***", The Centre for Research in Social Simulation. 2007/01/01

[28] WOOLDRIDGE Michael, JENNINGS Nicholas R. "***Intelligent Agents: Theory and Practice***", Knowledge Engineering Review, vol. 10(2), pp. 115-152. 1995.

[29] DARLEY V. and OUTKIN A.V. "***A NASDAQ Market Simulation: Insights on a Major Market from the Science of Complex Adaptive Systems***.", World Scientific Publishing Co Pte Ltd, pp. 39-46. 2007

[30] BONABEAU Eric "***Agent-based modeling: Methods and techniques for simulating human systems***", Proceedings of the National Academy of Sciences. 99 (suppl

3) pp. 7280-7287, May 2002.

[31] KHALIL Khaled, ABDELAZIZ Mohamed, NAZMY T.T., SALEM Adel, "***An agent-based modeling for pandemic influenza in Egypt***", INFOS2010 - 2010 7th International Conference on Informatics and Systems, pp. 1-7, 2010/04/30

[32] NEWMAN M. J. "***The structure and function of complex networks. SIAM Review Society for Industrial and Applied Mathematics***", Vol. 45 ,No. 2 , pp. 167–256, 2003

[33] PEREZ L., DRAGICEVIC S., "***An agent-based approach for modeling dynamics of contagious disease spread.***", Int J Health Geogr vol. 8, 50 (2009).

[34] KEELING Matt J and EAMES Ken T.D, "***Networks and epidemic models***", J. R. Soc. Interface. 2: pp. 295–307. 2005.

[35] VAZQUEZ-PROKOPEC G.M., STODDARD S.T., PAZ-SOLDAN V. et al., "***Usefulness of commercially available GPS data-loggers for tracking human movement and exposure to dengue virus***". Int J Health Geogr vol. 8, 68 (2009).

[36] JAFFRY S.W. and TREUR J. "***Agent-Based and Population-Based Simulation: A Comparative Case Study for Epidemics***". Proc. of the 22th European Conference on Modelling and Simulation, ECMS '08. European Council on Modeling and Simulation, , pp. 123-130, 2008.

[37] RILEY S., FRASER C., DONNELLY C.A., GHANI A.C., ABU-RADDAD L.J., HEDLEY A.J., LEUNG G.M., HO L.M., LAM T.H., THACH T.Q., CHAU P., CHAN K.P., LO S.V., LEUNG P.Y., TSANG T., HO W., LEE K.H., LAU E.M., FERGUSON N.M., ANDERSON R.M. "***Transmission dynamics of the etiological agent of SARS in Hong Kong: impact of public health interventions***". Science. 300(5627): 1961-1966. 2003 Jun 20.

[38] MULLER Johannes, KRETZSCHMARB Mirjam, DIETZ Klaus, "***Contact tracing in stochastic and deterministic epidemic models***". Elsevier Mathematical Biosciences vol. 164 pp.39-64 (2000)

[39] HYMAN James, LI Jia, STANLEY E., "***Modeling the impact of random screening and contact tracing in reducing the spread of HIV***". Mathematical biosciences, Vol. 181 pp.17-54, 2003/02/01

[40] Dati ISTAT, http://dati.istat.it, 22/02/2021

[41] AGENAS, Agenzia Nazionale per i Servizi Sanitari Regionali, https://www.agenas.gov.it/covid19, 22/02/2021

[42] HOERTEL N, BLACHIER M, BLANCO C, OLFSON M, MASSETTI M, LIMOSIN F, LELEU H. "***Facing the COVID-19 epidemic in NYC: a stochastic agent-based model of various intervention strategies.***" medRxiv [Preprint]: 2020.04.23.20076885. 2020 Apr 28

[43] Github Repository of stochastic model, https://github.com/henrileleu/covid19, 22/03/2021

[44] SILVA Petronio C. L., BATISTA Paulo V. C., LIMA Helder S., ALVES Marcos A., GUIMARAES Frederico G., SILVA Rodrigo C. P., "***COVID-ABS: An agent-based model of COVID-19 epidemic to simulate health and economic effects of social distancing interventions***". Chaos Solitons Fractals. Vol. 139: 110088. 2020 Oct

[45] MELLACHER Patrick, "***COVID-Town: An Integrated Economic-Epidemiological Agent-Based Model***" MPRA Paper 103661, University Library of Munich, Germany. 2020.

[46] MOLLALO Abolfazl, VAHEDI Behzad, RIVERA Kiara M., "***GIS-based spatial modeling of COVID-19 incidence rate in the continental United States***", Science of The Total Environment, Volume 728, 138884, ISSN 0048-9697. 2020.

[47] Ordinanze Regione Campania, https://www.regione.campania.it/regione/it/la-tua-campania/coronavirus-kyxz/ordinanze-del-presidente-della-regione-campania, 22/02/2021

[48] Osservatorio sulle fonti – Lista DPCM, https://www.osservatoriosullefonti.it/emergenza-covid-19/fonti-governative/decreti-del-presidente-del-consiglio-dei-ministri/2997-emcov-dpcm-elenco, 22/02/2021

[49] HOERTEL Nicola, BLACHIER Martin, LIMOSIN Frédéric, SANCHEZ-RICO Marina, BLANCO Carlos, OLFSON Mark, LUCHINI Stéphane, SCHWARZINGER Michaël, LELEU Henri, "***Optimizing SARS-CoV-2 vaccination strategies in France: Results from a stochastic agent-based model***", medRxiv. 21249970. 17/01/2021.

[50] Intel oneAPI Thread Building Blocks,

https://software.intel.com/content/www/us/en/develop/tools/oneapi/components/onetbb.html, 22/02/2021

[51] Intel oneAPI Math Kernel Library, https://software.intel.com/content/www/us/en/develop/tools/oneapi/components/onemkl.html, 22/02/2021

[52] Documentazione di Thread Building Blocks https://www.threadingbuildingblocks.org/docs/doxygen/a00150.html#aa0c0d46ec09bedc1ee876aa8aa87c3d1, 22/02/2021