Università degli Studi di Salerno
Dipartimento di Informatica

Tesi di Laurea Magistrale in
Informatica

A hybrid environment for the simulation and analisys of fake news' dynamics: model enhancement and experimentation

Relatore
Prof.ssa Delfina Malandrino
Prof. Rocco Zaccagnino

Candidato
Andrea Di Pierno
Matr. 0522501254

Anno Accademico 2022-2023

Abstract

Oggigiorno il rilevamento delle *fake news* è divenuto un argomento centrale nell'ambito della ricerca scientifica. Con la nascita delle *intelligenze artificiali* generative e degli strumenti che da esse derivano, è aumentata la facilità con cui le fake news possono essere create e diffuse in rete. La proliferazione di tali notizie espone la società a notevoli rischi quali la *disinformazione* e la manipolazione dell'opinione pubblica. Tali fattori, uniti alla facilità di accesso ad Internet ed ai social network, permettono alle fake news di diffondersi in breve tempo tra gli utenti. In questo ambito, lo studio delle reti sociali che vengono a crearsi tra gli utenti dei social network, diventa un elemento fondamentale per comprendere a fondo come le fake news nascano e si diffondano anche al di fuori dei gruppi di utenti che le promuovono. In questo lavoro di tesi, si vuole approfondire lo studio della diffusione delle fake news e dei meccanismi di contrasto sfruttando insieme approcci *model-driven* e *data-driven*, al fine di migliorare un modello di apprendimento che possa identificare la diffusione delle fake news in uno scenario realistico. A tal fine, verranno impiegati una simulazione ad agenti che rappresenta la rete sociale ed un super-agent che sfrutta tecniche di *Deep Reinforcement Learning* per imparare a riconoscere la diffusione delle fake news e le azioni da intraprendere per contrastarle. Al fine di rendere la simulazione più realistica e quindi migliorare le prestazioni del super-agent nel mondo reale, questo lavoro di tesi si basa sull'implementazione di caratteristiche fondamentali delle reti sociali ad oggi esistenti, unitamente alla modellazione di elementi della psicologia umana per rendere ancora più realistica l'interazione tra agenti. Tra le caratteristiche implementabili di maggiore interesse figurano gli schemi di following dei social network ed i bias cognitivi, che permettono un significativo avvicinamento della simulazione alle dinamiche del mondo reale.

Questa tesi è stata sviluppata in EDGE RESEARCH LAB

Indice

Capitolo 1

Introduzione

Negli ultimi anni i social network hanno avuto una crescita considerevole che ha cambiato il modo in cui le persone interagiscono tra loro, traslando molte interazioni dal mondo reale a quello digitale. Tutto ciò, unitamente alla facilità di creazione ed ai costi decisamente bassi per la diffusione, permette la generazione di una grandissima mole di informazioni a cui l'utenza viene continuamente sottoposta. Notizie, podcast, video e post sono solo alcuni esempi delle varie connotazioni che l'informazione può assumere al giorno d'oggi e che rappresentano forme semplici ed economiche per veicolare una notizia. Tuttavia, la facilità di diffusione delle informazioni ha ben presto destato l'attenzione di utenti malintenzionati, che manipolano le informazioni e ne diffondono una versione alterata, al fine di perseguire un obiettivo. Solitamente l'obiettivo riguarda l'ottenimento di denaro da parte degli utenti, tuttavia in alcuni casi si può anche arrivare a voler incitare una gran mole di persone ad avere comportamenti sovversivi nei confronti delle istituzioni, con il potenziale pericolo di destabilizzare intere comunità o addirittura interi stati. La limitazione della diffusione di tali contenuti, etichettati comunemente con *Fake News*, diventa dunque una sfida sempre più centrale nel mondo della ricerca scientifica, in cui da diversi anni si lavora all'implementazione di tecniche di riconoscimento e contrasto della diffusione di fake news. Grazie agli studi effettuati in quest'ambito, sappiamo che la diffusione delle informazioni sui social network è così rapida per via del fenomeno delle *echo chamber*, ovvero comunità di persone con la stessa opinione su un dato argomento che tendono autonomamente ad aggregarsi, permettendo una maggiore diffusione delle informazioni di cui condividono l'opinione. Per poter avere un'analisi precisa delle complesse dinamiche sociali che governano le modalità di diffusione delle notizie sui

social network è necessario avere un modello simulativo che astragga le caratteristiche principali delle reti sociali al fine di rappresentare al meglio i modelli comportamentali e sociali del mondo reale [1, 2, 3, 4]. Grazie alla versatilità ed alla potenza offerta dai modelli di intelligenza artificiale, inoltre, è possibile automatizzare le attività di analisi dei suddetti modelli di simulazione, ottenendo in questo modo un ambiente software in grado di fornire analisi precise delle dinamiche di diffusione delle fake news [5, 6, 7, 8, 9, 10, 4].

Questo lavoro di tesi si basa sullo studio e sul miglioramento di un ambiente software ibrido che combina gli approcci della simulazione basata su agenti a tecniche di Deep Reinforcement Learning (DRL). Gli agenti rappresentano gli utenti di un social network, mentre con le tecniche di DRL si implementa un super-agente che funge da autorità (amministrazione, moderazione, ecc.) e che permette di analizzare la rete ed intervenire, se necessario. All'interno della simulazione vi è una echo chamber, che permette la diffusione di una fake news all'interno della rete. Il super-agente ha l'obiettivo di contrastare la diffusione delle fake news e dunque applica delle azioni scelte sulla base del suo apprendimento per poter raggiungere il suo obiettivo. In questo scenario, il lavoro svolto ha l'obiettivo di migliorare la simulazione aggiungendo caratteristiche alla rete che possano avvicinare il modello a veri e propri scenari del mondo reale. In particolare, l'idea è quella di aggiungere dinamicità alla struttura della rete, in modo tale che possa meglio rappresentare un social network ed effettuare degli esperimenti per meglio comprendere l'impatto di queste aggiunte sulla simulazione.

Struttura della tesi. Questo lavoro è organizzato come segue:

- Capitolo 2: stato dell'arte ed analisi della simulazione;
- Capitolo 3: introduzione sulla simulazione ad agenti, sui concetti di base del Deep Reinforcement Learning e sull'ambiente ibrido di simulazione;
- Capitolo 4: descrizione dei meccanismi sociali aggiunti alla simulazione e degli adattamenti effettuati per renderli compatibili con l'ambiente ibrido.
- Capitolo 5: dettagli sugli esperimenti condotti e analisi dei risultati ottenuti.
- Capitolo 6: osservazioni finali e sviluppi futuri.

Capitolo 2

Stato dell'arte e analisi del lavoro svolto

In questo capitolo verrà analizzato il lavoro svolto finora, fornendo un background teorico sulle fake news. In particolare, verrà descritto il problema delle fake news (Sez. 2.1), l'ambiente ibrido di simulazione(Sez. 2.2) ed i suoi limiti. (Sez. 2.3)

2.1 Le fake news: tra scienza, sociologia e psicologia

Grazie all'accesso alla rete Internet, negli anni l'informazione online è divenuta sempre più importante, poichè rappresenta il modo più veloce ed economico per accedere alle notizie. La mancanza di controlli sulla veridicità delle notizie mista ad una crescita esponenziale del consumo di notizie online ha portato alla nascita a due fenomeni da cui si originano le fake news[11]:

- *Echo Chamber* [12]: è una rete che viene a crearsi tra più persone che condividono la stessa opinione su una data notizia, spesso dovuta a bias cognitivi [13]. Ciò permette una maggiore diffusione della notizia nella rete;
- *Diffusione di opinioni false*: poichè su internet vige la libertà di espressione, ognuno può esprimersi liberamente. A seconda delle motivazioni che spingono l'utente ad esprimersi, si può ricadere in due casistiche:
  - *Opinioni false intenzionali*: si ricade in questa casistica quando gli utenti cercano di manipolare l'opinione altrui attraverso la dif-

fusione di informazioni false, spesso anche per via della mancanza di misure attive di filtraggio dei contenuti condivisi. In tal caso, lo scompiglio creato potrebbe permettere una diffusione più rapida della notizia;

– *Opinioni false non intenzionali*: si diffondono quando un utente cerca di diffondere un'opinione non vera, magari attraverso metodi comunicativi atti ad ottenere un certo seguito, ma che finisce con il creare disinformazione in modo involontario [11].

Negli utlimi anni, l'interesse della comunità scientifica è quindi aumentato, nel tentativo di contrastare tale fenomeno e fornire strumenti in grado di effettuare una distinzione tra notizie vere e false. Alla base di questi studi vi è l'analisi dei fattori psicologici che spingono le persone a diffondere fake news. Dagli studi emerge che le fake news devono la loro diffusione alla reazione psicologica scatenata nel lettore, poichè il loro contenuto è più allettante rispetto alle normali news. Le fake news, infatti, propongono notizie esposte in maniera più semplice, riuscendo ad attirare maggiore attenzione anche da parte di persone poco scolarizzate o che non riescono a prestare particolare attenzione ai contenuti di cui stanno fruendo. Inoltre, in questo ambito giocano un ruolo fondamentale anche l'esposizione dell'utente alle news, i bias cognitivi ed una serie di comportamenti sociali.

L'esposizione dell'utente consiste nel numero di volte in cui egli vede la stessa notizia: più alto è questo valore, maggiore è la probabilità di credere al contenuto. Per quanto riguarda i bias cognitivi, invece, alcuni dei più importanti sono [14]:

- *Bias di conferma*: l'utente tende a credere più facilmente a contenuti che confermano la propria opinione [15];
- *Bias di attenzione*: una persona che legge di un avvenimento scioccante può cominciare a notare con maggiore frequenza fenomeni collegati a quell'avvenimento [16];
- *Bias emozionale*: l'utente tende ad evitare di credere a notizie che possono coinvolgerlo emotivamente, ad esempio provocando nervosismo o preoccupazione [17];
- *Bias di aspettativa*: l'utente tende a credere a notizie in linea con le proprie aspettative, nonostante le spiegazioni riportate siano insufficienti a dimostrare quanto accaduto [18].

Per ciò che concerne i comportamenti sociali, abbiamo:

- *Effetto gregge o bandwagon effect*: l'utente tende a percepire come veritiere le notizie credute da un vasto numero di persone, mentre tende a percepire come un attacco le notizie discordanti[19];
- *Effetto del falso consenso*: l'utente tende a credere che la propria opinione sia quella maggiormente accettata, percependo come ostili le opinioni contrastanti[20];
- *Naive realism*: l'utente tende a credere che la realtà percepita sia la realtà effettiva[21];
- *Normative social influence theory*: l'utente tende ad adeguarsi all'opinione del gruppo sociale di appartenenza o nel quale si identifica[22].

Tutti questi fattori insieme, contribuiscono alla diffusione delle fake news, alla creazione delle echo chamber e, in generale, alla manipolazione dell'opinione pubblica.

2.2 Ambiente ibrido di simulazione per le fake news

In questa sezione si offre una breve panoramica sulle motivazioni che hanno portato allo sviluppo dell'ambiente software oggetto di questo lavoro di tesi, passando poi alle possibili funzionalità da implementare per miglioralo.

2.2.1 Perchè un ambiente ibrido?

Per ambiente ibrido di simulazione intendiamo un ambiente software in grado di combinare i due approcci principalmente impiegati per lo studio di misure di contrasto alle fake news. Da una parte abbiamo una simulazione basata su agenti, dall'altra tecniche di Deep Reinforcement Learning (DRL) per poter addestrare un modello in grado riconoscere e contrastare attivamente la diffusione delle fake news.

Si rende necessaria l'ibridazione degli approcci per via dei limiti tecnici riscontrati in letteratura su entrambi i fronti. Un modello di DRL applicato ad una rete sociale esistente, fornisce scarsi risultati e quindi applica azioni correttive in modo errato. Numerosi studi che presentano un livello di accuratezza molto elevato in fase di addestramento del modello, finiscono col vedere questo valore crollare quando il modello viene applicato a reti sociali reali. Ciò accade per via della natura stessa delle fake news e delle reti sociali, oltre che dei modelli di IA in generale. Le fake news, infatti, non

parlano sempre dello stesso argomento, ma tendono a seguire gli argomenti trattati nelle news reali. Un esempio di questa tendenza è il passaggio da una circolazione di fake news che presentavano come tematica preponderante la pandemia da Covid-19 a fake news sulla guerra in Ucraina dal Febbraio 2022. In aggiunta a ciò, i modelli trattati in letteratura sono specializzati su una sola tipologia di fake news, rendendosi quindi non applicabili a scenari realistici, oppure addestrati su una tutte le fake news pubblicate fino ad un certo periodo, ma incapaci di apprendere come rilevare le fake news di nuova pubblicazione. Infine, se l'argomento principale delle fake news cambia, vi è anche un cambiamento nella struttura delle echo chambers, a cui vanno ad aggiungersi nuovi utenti sensibili all'argomento trattato, mentre altri le abbandonano.

I due approcci vengono combinati in modo da introdurre all'interno della simulazione il modello di DRL, permettendogli di analizzare lo stato della rete ed eseguire delle azioni. Poichè la simulazione è basata su agenti, il modello di DRL è anch'esso un agente, in particolare un super-agente che, a differenza degli altri agenti, è in grado di intraprendere azioni correttive al fine di contrastare la diffusione delle fake news.

2.3 Ambiente di simulazione: limiti e possibili sviluppi

Nonostante l'approccio ibrido rappresenti un passo in avanti per la ricerca scientifica, sussistono delle limitazioni che, se risolte, potrebbero migliorare ulteriormente la qualità della simulazione in termini di verosimiglianza con le reti sociali reali.

Per ciò che concerne il modello simulativo in essere, vi è la mancanza di caratteristiche peculiari delle reti sociali reali, che scaturiscono dal comportamento degli utenti. Di seguito, un breve elenco di queste ultime[4]:

- *Schemi di follow-up*: nei social network seguire un utente significa avere piacere nel vedere i contenuti che egli condivide. Ciò significa che un utente interessato alle fake news, seguirà altri utenti che ne condividono o che ne creano. Inoltre, considerando il fatto che gli interessi di una persona possono cambiare nel tempo, è possibile rendere questo comportamento dinamico per rendere lo scenario di simulazione ancora più realistico;

- *Crescita della rete*: come per tutti i prodotti e servizi, un social network ha un ciclo di vita. Inizialmente, molti utenti sono attratti dalla novità e si iscrivono, mentre dopo un certo periodo di tempo la crescita comincia a calare gradualmente. Questa caratteristica, oltre a rendere più realistica la simulazione, consente di introdurre un maggior numero di nodi nella rete, mentre nella simulazione originale il numero è sempre rimasto invariato;

- *Decrescita della rete*: dualmente alla crescita di un social network vi è la sua decrescita. Infatti, esiste un vero e proprio turnover di utenti che si iscrivono e abbandonano la piattaforma. Questa funzionalità risulta, oltretutto, molto utile per bilanciare il tasso di crescita, evitando che troppi nodi popolino la rete e che quindi i tempi di esecuzione di una simulazione diventino eccessivamente lunghi.

Per quanto riguarda i fattori psicologici che intervengono nella diffusione di una fake news, è possibile implementare i bias cognitivi, che possono rendere più realistico il processo di formazione delle echo chambers e di diffusione della notizia nella rete. Ciò permetterebbe anche una sperimentazione più approfondita, dando la possibilità di analizzare, ad esempio, i cambiamenti nei risultati della simulazione per tutte le possibili combinazioni di bias[4].

Per la sperimentazione, invece, è possibile introdurre l'esecuzione parallela di più esperimenti sulla stessa macchina, con l'obiettivo di sfruttare tutti i core della CPU e ridurre quindi i tempi complessivi di esecuzione della sperimentazione. Tuttavia, è importante considerare che la simulazione ha un consumo elevato di memoria RAM, dunque l'obiettivo è massimizzare il numero di esperimenti, cercando di evitare la saturazione della memoria[4].

Capitolo 3

Background scientifico metodologico della ricerca

In questo capitolo verrà esposto un quadro generale su aspetti di base che caratterizzano metodologie e tecnologie adoperate in questo lavoro di tesi; in particolare si espongono:

- introduzione alla simulazione basata su agenti (Sez. 3.1);
- introduzione al Deep Reinforcement Learning (Sez. 3.2);
- panoramica sul modello ibrido di simulazione, sulle tecnologie utilizzate e sul suo funzionamento (Sez. 3.3).

3.1 Overview sulle simulazioni Agent-based (ABM)

3.1.1 Breve introduzione alle scienze sociali computazionali

Le scienze sociali computazionali sono una disciplina nata dall'intersezione tra le scienze sociali ed la scienza dell'informazione. Nell'ambito della simulazione, è possibile combinare le teorie delle scienze sociali, tipicamente studiate su carta o realizzando equazioni, con ambienti software di simulazione che permettono di studiare in maniera approfondita l'interazione di diversi attori. Lo studio passa quindi da una forma cartacea ad una digitale, permettendo anche la simulazione di scenari artificiali impossibili da realizzare nel mondo reale.
La realizzazione di una simulazione implica la parametrizzazione di tutte le caratteristiche che si vogliono introdurre in essa. In poche parole, ogni informazione viene trasformata in parametri da fornire in input. La parte più

complessa della realizzazione di un modello simulativo è proprio la selezione e l'implementazione dei parametri, che per le simulazioni più complesse possono essere molto numerosi. Tuttavia, la simulazione ha il vantaggio di permettere anche agli altri ricercatori di studiare a partire dallo stesso modello, migliorando la velocità e la precisione con le quali viene portata avanti l'attività di ricerca.

3.1.2 Agent-based model

I modelli basati su agente sono una classe di modelli computazionali sviluppati per consentire la simulazione di interazioni tra agenti. L'obiettivo di una simulazione è valutare l'effetto dell'interazione tra gli agenti, dato un determinato contesto.

Un modello ad agenti deve possedere tre elementi di base, ovvero gli agenti, il contesto e le regole. Gli agenti rappresentano delle entità singole (persone, animali, cose) o collettive (organizzazioni, aziende, gruppi, ecc.) e posseggono dei parametri soggettivi grazie ai quali possono avere una caratterizzazione ben specifica. Il contesto della simulazione è l'ambiente in cui si svolge l'interazione tra gli agenti, possiede delle caratteristiche che possono avere effetto sugli agenti e può essere suddiviso in unità spaziali per poter, ad esempio, localizzare un determinato cluster di agenti. Le regole rappresentano invece un insieme di leggi che governano le interazioni tra gli agenti, definendo le modalità e i tempi delle interazioni stesse.

Di fondamentale importanza è il fatto che gli agenti interagiscono tra loro, ma non è detto che lo facciano tutti contemporaneamente o con tutti gli altri agenti. Infatti, solitamente un agente ha tra le sue caratteristiche una serie di vicini con cui può interagire.

3.2 Introduzione al Deep Reinforcement Learning

Il Deep Reinforcement Learning deriva dalla combinazione del Reinforcement Learning con il Deep Learning. In questa sezione verrà fornita una panoramica sul suo funzionamento e sulla tipologia di algoritmi scelti per la simulazione.

3.2.1 Il Reinforcement Learning (RL) in breve

Il Reinforcement Learning è una tecnica per la creazione di modelli di IA che consente di effettuare l'apprendimento dalle interazioni con degli elementi

in uno specifico contesto. Alla base di questa tecnologia vi è l'osservazione del comportamento di un agente che esegue delle azioni nell'ambiente in cui opera. Le azioni che l'agente decide di eseguire sono scelte sulla base delle ricompense (*reward*) assegnate ad ogni azione. Il valore della reward è stabilito da alcune condizioni associate all'ambiente in cui viene eseguita l'osservazione.
La differenza dalle tecniche di apprendimento supervisionato e non supervisionato sta proprio nel modo in cui l'agente si comporta, poiché nell'apprendimento supervisionato si cerca di classificare dati sulla base di dati in input già etichettati, mentre in quello non supervisionato l'obiettivo è classificare informazioni sulla base di dati non etichettati, attraverso l'identificazione di pattern o strutture.

3.2.2 Come funziona l'apprendimento per rinforzo

Come anticipato, nel RL l'agente sceglie le azioni da eseguire con lo scopo di raggiungere un determinato obiettivo, andando a modificare di conseguenza lo stato dell'ambiente e le azioni che verranno intraprese successivamente. Poiché ogni azione ha degli effetti non prevedibili, l'agente effettuerà un'analisi prendendo in esame tre parametri:

- *Policy*, ovvero l'associazione stato-azione. Una policy fornisce all'agente la migliore azione da eseguire in un determinato stato dell'ambiente;
- *Reward*, che quantifica la desiderabilità da parte dell'agente di essere in un certo stato. Il raggiungimento di una reward si ha con l'esecuzione nel tempo di determinate azioni da parte dell'agente, che ha come obiettivo la massimizzazione di questo parametro;
- *Value function*, ovvero la ricompensa ricevibile dall'agente nel lungo termine, a partire da un certo stato. Ogni stato, infatti, ha una value function che potrebbe indicare ad esempio la quantità totale di reward ottenibile dall'agente a partire da quello stato.

L'obiettivo di un agente di RL è quello di creare policy e value function tali da massimizzare la propria reward.

3.2.3 Introduzione alla tecnica del Deep Q-Learning

Il Q-Learning è un algoritmo di IA basato sul concetto di Q-Function. La funzione Q di una policy rappresenta il rendimento atteso o la somma delle reward ottenute dallo stato di partenza, applicando una certa azione, seguita

dall'applicazione di una policy.
Una funzione Q di una certa policy $\pi, Q^{\pi}(s, a)$ parte da uno stato s, applica un'azione a e successivamente una policy π.
Una funzione Q ottimale $Q^{*}(s, a)$ rappresenta il rendimento massimo che si può ottenere partendo da s, applicando un'azione ed una policy ottimale subito dopo, obbedendo all'equazione di Bellman sull'ottimalità:

$$Q^{*}(s, a) = \mathbb{E}[r + \gamma \ max_{a'} \ Q^{*}(s', a')]$$

Quindi il massimo ottenibile a partire da s ed applicando a, è dato dalla somma di tutte le reward r ed il valore di ritorno che si ottiene dall'applicazione della policy ottimale.
Per quanto riguarda il Q-Learning, l'idea è quella di utilizzare la suddetta equazione iterativamente per aggiornare i valori dell'algoritmo tramite la seguente funzione:

$$Q_{i+1}(s, a) \leftarrow \mathbb{E}[r + \gamma \ max_{a'} \ Q_i(s', a')]$$

Ciò viene fatto perché tale funzione converge verso quella ottimale [23].

Normalmente l'algoritmo rappresenta la Q-Function con una tabella contenente una entry per ogni combinazione stato-azione, ma ciò non risulta pratico per la quasi totalità dei problemi. In tal caso si addestra una funzione di approssimazione attraverso una rete neurale per stimare i valori (detti Q-Values) associati alle coppie stato-azione. Tale rete neurale prende in input dei parametri θ.
Questo algoritmo apprende seguendo la seguente policy greedy: $a = max_a Q(s, a; \theta)$
Per quanto riguarda l'ambiente e le interazioni con esso, la policy scelta è differente, di solito con tipologia $\epsilon - greedy$. Tale policy seleziona un'azione a caso con probabilità ϵ e la azione ottimale, di conseguenza, con probabilità $1 - \epsilon$

Experience Replay. La tecnica dell'Experience Replay consiste nell'inserire le transizioni di stato effettuate ad ogni passo in un *replay buffer* per poi usare queste informazioni in fase di calcolo della perdita temporale. In particolare, vengono utilizzate delle transizioni composte da *mini-batch* campionati dal buffer, per la computazione. I vantaggi ottenibili da questa tecnica riguardano la stabilità e l'efficienza del processo di Q-Learning

3.3 Il modello ibrido di simulazione

L'ambiente utilizzato per eseguire le simulazioni di questo progetto di tesi ibrida gli approcci *model-driven* e *data-driven* in un unico strumento. In questa sezione verranno analizzati i vari aspetti che caratterizzano questo ambiente.

3.3.1 Overview sul sistema e modello di funzionamento

L'ambiente di simulazione è composto da due elementi principali:

- *Simulazione ad agenti*, che si occupa di gestire il setup e l'avanzamento della simulazione, i basic-agents, il super-agent e le loro interazioni;
- *Moduli in Python*, che si occupano di orchestrare l'esecuzione della simulazione, fornire i parametri di esecuzione, collezionare informazioni da fornire in output e gestire gli algoritmi di DRL per l'addestramento del super agente.

Per la simulazione ad agenti viene utilizzato il tool NetLogo nella versione 6.2.0, uno strumento versatile che consente, attraverso delle API di interagire con la simulazione e modificare ogni parametro presente in essa, anche durante l'esecuzione di quest'ultima. L'interazione delle API con i moduli Python viene realizzata attraverso la libreria *PyNetlogo* [24]. I moduli scritti in Python si dividono in moduli per gestire la simulazione, i suoi parametri e le funzioni disponibili ed altri moduli necessari ai processi di reinforcement learning. Di seguito viene fornita una panoramica generale del sistema:

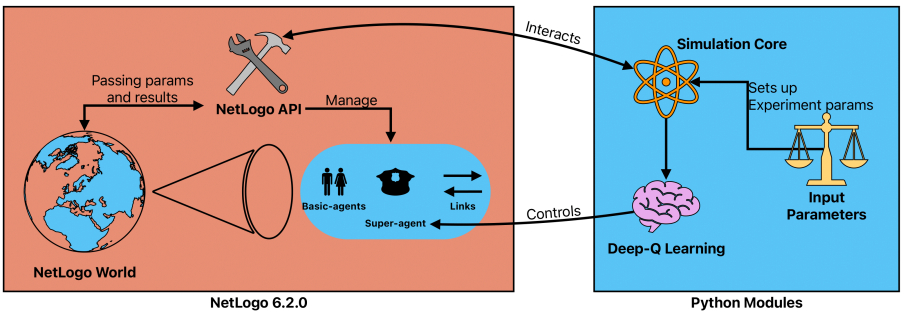


Figura 3.1: Abstract dell'architettura del sistema.

Com'è possibile vedere nella figura 3.1, il processo di simulazione comincia dal setup dei parametri da parte della persona che esegue la simulazione.

Una volta impostati i parametri (`Simulation parameters` ed `Environment Settings`) è necessario eseguire il modulo contenente il core della simulazione, che si occupa di prelevare i parametri precedentemente impostati, passarli in input a NetLogo e fare il setup della simulazione attraverso le API. Contemporaneamente viene inizializzato il processo di reinforcement learning per il super agente. Una volta completato il setup la simulazione comincia la sua esecuzione. Ogni passo della simulazione rappresenta un istante di tempo e viene detto *tick*, durante il quale NetLogo computa le interazioni tra agenti per quel preciso istante. Quando un tick viene completato, il core preleva alcuni dati dallo stato della simulazione (`Environment data`), necessari all'addestramento del super agente e li passa al modulo che gestisce il DRL, che si occuperà poi di eseguire i processi descritti nella sezione 3.2 e scegliere quindi la prossima azione da far eseguire all'agente, oltre che computare la reward per quanto eseguito nel tick appena concluso. Una volta completato questo processo, il comando per l'esecuzione dell'azione viene passato a NetLogo, che lo eseguirà nel tick successivo, reiterando poi questo procedimento fino alla fine dei tick stabiliti nei parametri di simulazione o fino al completamento della simulazione stessa.

3.3.2 Descrizione del modello agent-based

Di seguito viene fornita una descrizione dell'ABM utilizzato per simulare la diffusione di una fake news.

Obiettivi della simulazione

La simulazione ha come obiettivo quello di far circolare una fake news all'interno di una rete sociale e computare ciò che accade agli agenti esposti alla notizia. Nella simulazione sono presenti due tipologie di agenti: basi-agents e super-agent. Il super agente ha come obiettivo quello di eseguire interventi mirati a contrastare la diffusione della fake news e, chiaramente, non possiede un'opinione sulla notizia. I basic-agents, al contrario, possiedono un'opinione sulla notizia che può essere:

- *Opinione A*, a supporto della fake news e caratterizzata da un colore arancione per l'agente;
- *Opinione B*, contraria alla fake news, che presenta un colore blu per l'agente;
- *Opinione neutra*, caratterizzata da un colore grigio e che indica indecisione sulla notizia.

Stato degli agenti

Le variabili presenti nella simulazioni sono di due tipi, globali o di stato. Le variabili globali sono visibili a tutti gli agenti, mentre quelle di stato, appunto, riguardano lo stato di ogni singolo agente. Tra le variabili di stato troviamo:activation treshold (θ), betweenness, eigenvector, closeness, clustering, community, page-rank, in-degree, out-degree, degree), is-a-active, is-b-active, is-in-cluster, warning, reiterate, is-opinion-b-static, opinion-metric, received-a news-counter, received-b-news-counter. Il super-agent non possiede alcun tipo di variabile. Le variabili globali sono: eutral-agents, active-a-agents, active-b-agents, in-cluster-agents, total-number-agents, total-links, k, global-cascade, is-warning-active, is-reiterate-active.

Inizializzazione della simulazione

La simulazione inizia con la scelta del numero di nodi da generare, detto N, il numero di tick e la tipologia di rete sociale da implementare. La rete può avere tre tipologie:

- *Erdős-Rényi*, tipologia di rete che segue una distribuzione di *Poisson*, caratterizzata da un parametro k che indica il valore della distribuzione normale per la generazione dei link tra i nodi e la deviazione standard. In particolare, k serve per stabilire il numero medio di link che avranno i nodi;
- *Small world*, che possiede i parametri rewire-probability e neighborood-size, rispettivamente la probabilità di rewire (scollegamento di un link dal nodo a cui è connesso e collegamento casuale ad un altro nodo) ed il numero di link che i nodi devono avere;
- *Preferencial attachment*, che genera una rete di tipo *scale free*, ovvero una rete in cui i nodi vengono aggiunti uno alla volta e generano link con altri nodi in base al grado degli altri nodi nella rete: maggiore è il grado di un nodo, maggiore è la probabilità con cui un nuovo nodo genererà un collegamento con esso.

Echo chamber

L'inizializzazione della rete è seguita dalla creazione della *echo chamber*, ovvero un insieme di utenti caratterizzati da due proprietà:

- *Opinion Polarization* (P_o), ovvero la proprietà secondo la quale gli utenti condividono lo stesso punto di vista su un'opinione;

- *Network Polarization (P_n)*, implica maggior coesione tra gli utenti di una sottorete, rispetto al resto della rete.

Per la creazione di una echo chamber, attraverso la funzione *Echo Chamber Fraction (ECF)* viene scelta una frazione di nodi della rete che dovrà entrare a farne parte e si imposta il loro attributo `is-in-cluster` a *true*. Per ottenere il numero `c` di nodi per la echo chamber si esegue il seguente calcolo: `c` $= N \times ECF$ Successivamente si calcola il numero di archi come $E' = (c \times \frac{k}{2}) \times P_n$, con `k` che rappresenta il grado medio dei nodi ed `c` che rappresenta il numero di nodi. Si va quindi a creare link tra i nodi appartenenti alla echo chamber, aumentando così la `degree-centrality` media del cluster. A questo punto viene scelto un nodo casuale della echo chamber e si va ad impostare il parametro `is-a-active` a true e l'opinion metric, dopodichè si selezionano i suoi vicini e si reitera il processo. Dualmente, il processo viene rieseguito per i nodi di tipo B. A questo punto si va ad inizializzare la soglia di attivazione (`activation treshold`) dei nodi: per i nodi di tipo A viene impostato un valore di θ pari a $\Theta - P_o$, mentre per i nodi di tipo B viene impostato θ uguale al valore scelto all'inizio della simulazione.

Il super agente

In seguito alla creazione della echo chamber, viene inizializzato ed inserito il super-agent nella rete. Questo speciale agente non possiede attributi, ma può eseguire delle azioni: *warning*, *reiterate*, *static b nodes* ed una quarta azione per lasciare l'agente in attesa. Le prime tre azioni servono a contrastare la diffusione delle opinioni di tipo A.

La funzione `Go`

Completata l'inizializzazione del super agente, ha inizio la simulazione. All'inizio di ogni tick viene controllato se il super-agent abbia eseguito o meno azioni di contrasto, avendo premura che esegua solo e soltanto un'azione per tick. In seguito, si controlla se qualche agente abbia cambiato o meno opinione durante il tick considerato e, in caso positivo, si imposta come attivo per l'opinione A o B. Infine, si calcola per ogni nodo la frazione di vicini di tipo a e di tipo B, dopodichè si confronta la frazione predominante con la soglia di attivazione (θ) nel modo seguente:

$$\begin{cases} \max(fraction_a^{ba_i}, fraction_b^{ba_i}) > \Theta & \text{viene modificata l'}\textit{opinion-metric} \\ fraction_a^{ba_i} = fraction_b^{ba_i} & ba_i \text{ non verrà influenzato dai vicini} \end{cases}$$

Una volta eseguito questo calcolo, si eseguono eventualmente le azioni del super-agent e, una volta raggiunto il numero di tick definito nei parametri, si calcola la *global-cascade*.

3.3.3 Il design del modello in pillole

La costruzione del modello si basa su un'assunzione di base: la diffusione delle fake news comincia a partire da una echo chamber. La simulazione si basa sul fatto che nella rete esistano due tipologie di nodi in contrasto tra loro, ovvero i nodi con opinione A e B, nei quali l'andamento dell'opinione è determinato da una soglia di attivazione θ.
L'equilibrio nella rete è modificabile unicamente dalle azioni del super-agent. Tale equilibrio dipende dal contrasto tra le due tipologie di basic-agent, che vogliono far prevalere la propria opinione sugli altri.
In questo contesto, l'unico agente in grado di modificare il proprio comportamento in relazione all'ambiente circostante è il super agente, che sceglie quali azioni intraprendere grazie alle informazioni apprese tramite DRL ed allo stato attuale della rete. Lo scopo del super-agent ad ogni tick è quindi quello di prevedere la migliore azione possibile, grazie anche alla possibilità di ispezionare completamente lo stato della rete, a differenza dei basig-agents che possono conoscere solo le informazioni degli agenti vicini.
Gli eventi successivi alle azioni che il super-agent può intraprendere ed il modo in cui la rete e la echo chamber vengono inizializzate, contribuiscono a dare stocasticità alla simulazione, similmente a quanto avviene nel mondo reale in un social network, dove i basic-agents sono gli utenti ed il super-agent è rappresentato dal team di moderazione.
Infine, per modellare al meglio i vari aspetti della simulazione, sono presenti dei parametri da passare in input come da tabella 3.1.

3.3.4 Azioni e metriche di osservazione

Le azioni che l'agente può compiere sono necessarie a contrastare attivamente la diffusione delle fake news. La scelta dell'azione da intraprendere dipende dal processo illustrato nel paragrafo 3.2.3.

Warning

L'azione di warning può essere di due tipi: normale o globale. Un warning globale imposta a true l'attributo `warning` di tutti i *basic-agents*, mentre un warning normale lavora solo su un gruppo *basic-agents*. Il warning può avere effetto solo sui nodi di tipo A e quelli neutrali, infatti esistono due parametri

Parametro	Descrizione
`nb-nodes`	numero di nodi con cui inizializzare la simulazione
`edge-type`	tipologia di archi presenti nella rete, che possono essere `directed` o `undirected`
`network-type`	tipologia di rete da inizializzare, che può essere *Erdős–Rényi*, *Small World* o *Preferencial Attachment*
`k-value`	valore per l'inizializzazione di una rete *Erdős–Rényi* che indica il numero medio di collegamenti che ogni nodo deve avere
`std-dev`	deviazione standard per l'inizializzazione di una rete *Erdős–Rényi*
`rewire-prob`	probabilità per una rete di tipo *Small World* che un arco venga disconnesso da una delle sue estremità e riconnesso ad un altro nodo della rete scelto in modo casuale
`neighborhood-size`	numero di vicini di un nodo in una rete *Small World*
`ticks`	numero di unità temporali (*ticks*) da raggiungere per completare la simulazione
P_n	network polarization, implica che gli utenti della *echo chamber* sono più connessi tra loro, rispetto al resto del network
P_o	opinion polarization, indica la facilità con cui gli utenti di una *echo chamber* siano inclini a condividere le stesse vedute su un dato argomento
θ	soglia di attivazione per l'`opinion-metric-step` di un agente
`echo-chamber-fraction`	frazione del numero totale di nodi della rete che deve far parte della *echo chamber*
`initial-opinion-metric-value`	valore iniziale della opinion metric degli agenti

Tabella 3.1: Elenco dei parametri in input alla simulazione.

che vanno a regolare l'influenza di quest'azione sulle suddette tipologie di agenti. La procedura di warning è permanente, ovvero, una volta lanciata su un agente, mantiene i suoi effetti fino al termine della simulazione.
Il warning ha maggiore effetto, quando l'indecisione di un agente su un'opinione è più elevata, mentre ha effetti inferiori sugli agenti già polarizzati verso un'opinione. Nello specifico, il funzionamento del warning è il seguente: quando un agente viene influenzato da una notizia, controlla se ha warning attivi su di esso e, se l'esito del controllo è positivo, genera un numero reale compreso tra 0 e 1. Se tale valore è inferiore alla soglia di `warning-impact`, allora si aggiorna l'`opinion metric`, altrimenti la notizia non ha effetto sull'agente.

Reiterate

Similmente a quanto effettuato con la warning, un'azione di reiterate imposta la rispettiva variabile di stato a true nei basic-agents selezionati. Quando un agente viene influenzato da una notizia, controlla l'attributo `reiterate` e, se impostato a true, riceverà ad ogni tick una notizia di tipo B per un numero di tick pari al numero dei suoi vicini. Quando è attiva una reiterate, il processo di aggiornamento dell'`opinion metric` è diverso rispetto al normale, ovvero genera un numero reale compreso tra 0 e 1 e controlla se tale valore sia minore o uguale al parametro θ del nodo e, nel caso in cui dovesse esserlo, si procederà ad aggiornare l'`opinion metric` verso l'opinione B. Nel caso in cui il nodo dovesse passare all'opinione B prima dell'esaurimento dei tick residui, la procedura terminerà in anticipo.

Static B Agents

La Static B Agents esegue una forzatura sull'opinione degli agenti selezionati, costringendoli a mantenere l'opinione B per tutta la durata della simulazione, rendendoli di fatto immuni alle opinioni di tipo A. I nodi vengono selezionati in base ai parametri *page rank*, *degree* e *betweenness*, nello specifico vengono selezionati i nodi che presentano i valori maggiori. Il super-agent modifica l'`opinion metric` degli agenti selezionati a 0 e attiva il booleano `is-opinion-b-static` al fine di rendere permanente la modifica per tutta la durata della simulazione.

Metriche della simulazione

Le metriche di simulazione servono per poter valutare la simulazione appena eseguita o lo stato della simulazione durante l'esecuzione.

Opinion metric L'*opinion metric* di un nodo è un numero reale compreso tra 0 e 1 che viene modificato quando l'agente viene esposto a delle notizie o a delle azioni del super agente. Attraverso il parametro `opinion-metric-step` è possibile impostare di quanto debba variare l'*opinion metric* di un agente quando quest'ultimo viene influenzato.
L'*opinion-metric* si basa sulle seguenti soglie:

- *valori compresi tra 0 e 0.33*, opinione di tipo B;
- *valori compresi tra 0.34 e 0.65*, opinione neutrale;
- *valori compresi tra 0.66 e 1*, opinione di tipo A.

Global Cascade La *global cascade* è la frazione di nodi che hanno un'opinione di tipo A in un certo istante della simulazione.

Viralità La *viralità* rappresenta il numero di volte in cui la *global cascade* supera la soglia di 0.5 dopo un certo numero di simulazioni ed è espressa come un numero reale compreso tra 0 e 1. Maggiore è la viralità, maggiore è la facilità con cui la fake news ha proliferato durante la simulazione.

Global Opinion Metric Mean La *Global Opinion Metric Mean* rappresenta il valore medio dell'opinion metric di tutti i *basic-agent* nella simulazione.

Get Most Influent A Nodes La *Get Most Influent A Nodes* rappresenta la frazione dei nodi che supportano l'opinione di tipo A e che hanno maggiore influenza nella rete. I nodi restituiti da questa metrica vengono ordinati in base a *betweenness centrality*, *degree centrality* oppure *page rank*.

Capitolo 4

Analisi ed implementazione delle dinamiche sociali nel modello simulativo

In questo capitolo verranno analizzate le varie funzionalità implementate in questo lavoro di tesi. Nello specifico, il capitolo è così suddiviso:

- introduzione alle dinamiche delle reti sociali ed alla computazione parallela(Sez. 4.1);
- implementazione delle dinamiche sociali nel modello simulativo(Sez. 4.2);
- parallelizzazione delle simulazioni ed analisi del carico computazionale in parallelo (Sez. 4.3);
- limitazioni del modello e possibili sviluppi futuri(Sez. 4.4).

4.1 Introduzione alle dinamiche delle reti sociali ed alla computazione parallela

In questa sezione verrà fornita una breve panoramica sulle principali caratteristiche delle reti sociali e sui concetti di base della computazione parallela.

4.1.1 Introduzione alle dinamiche sociali nel modello simulativo

Le dinamiche sociali sono l'insieme dei comportamenti e delle interazioni che uno o più individui possono avere in un determinato contesto sociale. Tali dinamiche possono essere semplicisticamente suddivise in dinamiche individuali o di gruppo.
Nell'ambito di questo lavoro di tesi vengono considerate ed analizzate le dinamiche sociali inerenti alle reti sociali virtuali, ovvero ai comportamenti ed alle interazioni degli utenti dei social network. In particolare, sono state analizzate le dinamiche inerenti al *follow-up* ed allo *user-turnover* dei social network.

Follow-up Seguire una persona su un social network significa volersi esporre ai contenuti che quest'ultima pubblica. Solitamente, si segue una persona perché i contenuti che pubblica sono di nostro interesse, ma le motivazioni possono essere molteplici. Nel nostro caso, andremo a considerare soltanto la suddetta casistica, ovvero la condivisione di interessi su un certo tipo di contenuti, nello specifico sulla fake news che verrà fatta circolare nella rete. Nel mondo reale, una persona segue un'altro utente sui social network fintanto che quest'ultimo pubblicherà contenuti in linea con la sua opinione. Un certo numero di contenuti che si disallineano con gli interessi dei follower, possono portare questi ultimi a non voler più seguire la persona che li pubblica.
Tra gli obiettivi di questo lavoro di tesi rientra proprio mappare quest'ultimo aspetto, ovvero il possibile cambiamento dei collegamenti tra utenti, in seguito a determinati eventi.

User-turnover Ogni social network, al pari di ogni prodotto o servizio, ha un proprio ciclo di vita. Inizialmente, data la novità introdotta dall'avvento della nuova piattaforma, vi sarà un gran numero di utenti che effettueranno l'iscrizione e pochi che effettueranno la cancellazione del proprio account. Il *turnover di utenti* comincia quindi con un numero di iscrizioni molto superiore a quello delle disiscrizioni. Tuttavia, con il passare del tempo, con la nascita di nuovi social network e a meno di sostanziali novità che possano attirare l'attenzione dell'utenza, vi è la tendenza da parte degli utenti ad abbandonare sempre di più il social network. La mancanza di interesse, fa quindi aumentare le disiscrizioni e calare le iscrizioni, portandoci quasi ad invertire la situazione iniziale [25];

4.1.2 Storia e principi della computazione parallela in pillole

Il *calcolo parallelo* consiste nell'eseguire software simultaneamente su più processori o su più core dello stesso processore. L'obiettivo del calcolo parallelo è risparmiare risorse computazionali ed accelerare i tempi di esecuzione del software.
In passato, tutti i processori presentavano un'architettura *single-core*, dunque la CPU poteva gestire un solo processo per volta, attraverso un complesso sistema di scheduling che consentiva di ottimizzare l'esecuzione dei programmi. Un semplice esempio è la gestione dell'interfaccia grafica: all'utente sembra che il software venga eseguito contemporaneamente ad alcune sue azioni, come ad esempio il movimento del cursore. Ciò è dovuto al rapido *cambio di contesto* effettuato dalla CPU. Tuttavia, le nuove CPU dell'epoca tendevano ad essere sempre più piccole e, secondo la *legge di Moore*, con un numero di *transistor* che raddoppiava ogni due anni circa. Questo ha portato a problemi di *thermal noise*, ovvero il problema per cui troppi transistor in uno spazio troppo ridotto, generavano una quantità di calore eccessiva, tanto da poter arrivare a danneggiare la CPU stessa. Per tale motivo, la soluzione al problema fu quella di trovare un trade-off tra temperatura, potenza di calcolo e complessità del software.
Da ciò nacque la computazione parallela, che comporta una maggiore complessità nello sviluppo del software, che però è bilanciata da minori temperature e da una potenza di calcolo nettamente maggiore. La computazione parallela è quindi diventata lo standard de facto per la progettazione delle architetture dei nuovi processori. Le CPU di nuova generazione, infatti, sono dotate di numerosi core in grado di gestire diversi *thread*. I thread sono dei processi *lightweight*, ovvero istanze di sottoprocessi che condividono tra loro il codice, la memoria e le risorse messe a disposizione dal sistema operativo.

Implicazioni della computazione parallela sul software

Il calcolo parallelo, se correttamente sfruttato, può permettere di eseguire task complessi in minor tempo, attraverso la suddivisione della computazione tra i vari thread. Tuttavia, avere un'architettura *multi-core*, non basta per poter effettuare della computazione in parallelo. Il prezzo da pagare per una maggiore efficienza è la complessità aggiuntiva nello sviluppo del software, collegata alla gestione della computazione parallela.
Quando parliamo di computazione parallela, infatti, dobbiamo tener presente che il software va modificato ad-hoc per poter gestire l'accesso alla memoria, l'esecuzione dei task per ogni thread e la gestione dei dati che i thread devono

condividere tra loro per poter proseguire nel calcolo. Inoltre, un aspetto da non sottovalutare è la necessità che i thread in determinati momenti possano avere la gestione esclusiva di determinate risorse, in modo da evitare modifiche da parte di altri thread, che possano portare a condizioni di inconsistenza. Il fenomeno appena descritto è detto *interfogliamento* delle istruzioni, ed è una diretta conseguenza della computazione parallela. Un core, infatti, potrebbe eseguire più thread che si alternano tra loro nell'esecuzione delle istruzioni e che, ad esempio, potrebbero dover effettuare delle scritture in memoria, basandosi sui dati rilevati in precedenza con delle letture. Ricordiamo che i thread condividono la memoria. Ipotizziamo quindi una situazione come quella descritta in figura 4.1. Al termine della computazione, il valore salvato dal thread A viene perso e, alla successiva lettura, A leggerà un valore differente da quello che si aspettava di leggere, cosa che potrebbe causare eventuali comportamenti indesiderati nel programma.

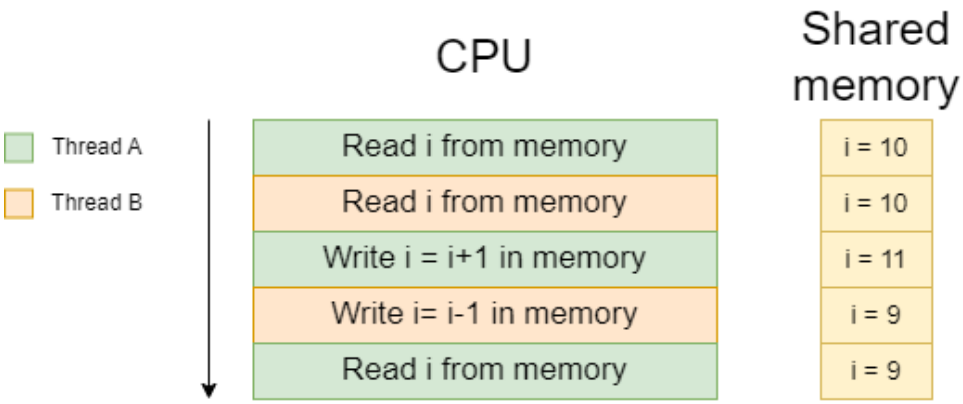


Figura 4.1: Esempio di interfogliamento tra due thread che accedono e modificano la stessa variabile condivisa

L'importanza della parallelizzazione delle simulazioni

Nell'ambito di questo lavoro di tesi la parallelizzazione è di fondamentale importanza per accelerare l'esecuzione degli esperimenti. Prima di qualsiasi modifica, infatti, un esperimento aveva una durata media di circa 7 ore. Le modifiche apportate al sistema vanno ad aumentare la complessità della simulazione e quindi ad allungare il tempo necessario per il completamento di un esperimento. Piuttosto che parallelizzare l'esecuzione di un singolo esperimento, l'idea è quella di eseguire più esperimenti insieme, andando

ad istanziare diverse simulazioni a seconda delle esigenze dell'utente. Di particolare importanza, tuttavia, è l'aumento delle risorse richieste dalla simulazione, sia in termini di carico sulla CPU che in termini di memoria principale richiesta.

4.2 Implementazione delle dinamiche sociali nel modello simulativo

In questa sezione verranno illustrate le modalità di pianificazione ed implementazione delle dinamiche sociali illustrate nella sezione 4.1.

4.2.1 Crescita e decrescita di una rete sociale

Nel listato 4.1 viene mostrata la funzione implementata per gestire la crescita della rete. Tale funzione innanzitutto controlla se il parametro *is_network_growing* è impostato a True (riga 4). Se il controllo ha esito negativo, l'algoritmo restituisce False (riga 30), altrimenti continua la sua esecuzione (riga 5). Se la simulazione è effettivamente iniziata, l'algoritmo preleverà il tick corrente e due array contenenti rispettivamente le soglie e le percentuali per la crescita (righe 11-12). Successivamente si effettuano dei controlli per evitare di essere oltre la fine degli array (righe 17-18), si calcolano gli agenti da aggiungere (righe 21-23) e, infine, si invia a *Netlogo* il comando per aggiungere gli agenti e si restituisce True (righe 25-26).

```
1  def grow(self):
2      is_network_growing = self.netlogo.get_growth()
3
4      if(is_network_growing):
5          #Controlliamo che la simulazione sia effettivamente cominciata
6          [...]
7
8          growth_ticks = self.params.getGrowthTicks()
9          growth_percentages = self.params.getGrowthPercentages()
10         index = 0
11
12         for tick_step in growth_ticks:
13             if(tick > tick_step):
14                 index += 1
15             elif(tick <= tick_step):
16                 break
17
18         if index >= len(growth_percentages):
19             index = len(growth_percentages) - 1
20
21         growth_percentage = growth_percentages[index]
```

```
        basic_agents = self.netlogo.get_total_agents()
        agents_to_add = int((basic_agents/100)*growth_percentage)

        self.netlogo.add_agents(agents_to_add)
        return True
    return False
```

Code Listing 4.1: Funzione di crescita per la rete

Dualmente, nel listato 4.2 viene mostrato l'algoritmo per la decrescita della rete sociale, il cui funzionamento è speculare a quello dell'algoritmo 4.1.

```
def leave(self):
    is_nodes_leaving = self.netlogo.get_leaving()

    if(is_nodes_leaving):
        #Controlliamo che la simulazione sia effettivamente cominciata
        [...]

        leave_ticks = self.params.getLeaveTicks()
        leave_percentages = self.params.getLeavePercentages()
        index = 0

        for tick_step in leave_ticks:
            if(tick > tick_step):
                index += 1
            elif(tick <= tick_step):
                break

        if index >= len(leave_percentages): # if the index is out of
    bounds, set it to the last index
            index = len(leave_percentages) - 1

        leave_percentage = leave_percentages[index]
        basic_agents = self.netlogo.get_total_agents()
        agents_to_remove = int((basic_agents/100)*leave_percentage)

        self.netlogo.remove_agents(agents_to_remove)
        return True
    return False
```

Code Listing 4.2: Funzione per la decrescita della rete

Per poter effettuare l'aggiunta e la rimozione dei nodi è stato necessario implementare le funzioni `add-agents` e `remove-agents` in Netlogo, di cui è presente il codice nel listato 4.3.

```
to add-agents [N]
 ;; Choosing the type of network
  if network = "Erdos Reny" [
     ER-RN N
  ]
```

```
6     if network = "Preferencial Attachment" [
7       P-A
8     ]
9     if network = "Small World" [
10      S-W
11    ]
12  end
13
14  to remove-agents [num-to-remove]
15    let agents-to-remove (n-of num-to-remove basic-agents)
16    ask agents-to-remove [ die ]
17  end
```

Code Listing 4.3: Funzioni per il controllo della crescita e della decrescita in Netlogo

Mentre la funzione `remove-agents` esegue il suo compito senza invocare altre funzioni, la funzione `add-agents` ha bisogno di chiamare la funzione specifica per il tipo di rete istanziato, al fine di aggiungere i nodi in modo corretto. Per fare ciò, la funzione di creazione degli agenti ha dovuto subire delle modifiche per poter distinguere gli agenti appena aggiunti, ai quali vanno poi inizializzati i vari parametri, da quelli già esistenti, che non devono subire modifiche. A tal fine, come è possibile vedere nel listato 4.4, i nuovi agenti vengono dapprima colorati in giallo per poter essere distinti dagli agenti già presenti, e solo successivamente all'inserimento dei parametri assumeranno la consueta colorazione grigia (agenti neutrali), prima di essere influenzati dalle notizie e passare ai colori blu e arancione.

```
1   to ER-RN [N]
2     create-basic-agents N [
3       setxy random-xcor random-ycor
4       set color yellow
5     ]
6
7     ask basic-agents with [color = yellow] [
8       let x random-normal k-value std-dev
9       if x < 0 [
10        set x 0
11      ]
12      ask n-of x basic-agents with [who != [who] of myself] [
13        ifelse links-to-use = "undirected" [
14          create-undirected-edge-with myself
15        ]
16        [
17          create-directed-edge-to myself
18        ]
19      ]
20    ]
21    ask basic-agents with [color = yellow] [
22      set-characteristics
```

```
23    ]
24
25    ask links [
26      set color grey - 1
27      set thickness 0.1
28    ]
29
30  end
```

Code Listing 4.4: Funzione per la creazione degli agenti in una rete di tipo Erdős-Rényi

Strategia di crescita e decrescita della rete

La rete presenta una strategia di crescita in linea con i dati riportati su Digital2023 [25]. I dati sulla crescita a disposizione, però, sono già consuntivi dei valori di decrescita. Per impostare i parametri di crescita e decrescita, quindi si imposta `grow_percentages[i]` come l'i-esimo valore di crescita presentato sommato ad un valore `x`, con `x` tale che sommato all'i-esimo valore del grafico, lo arrotonda alla decina superiore. `leave_percentages[i]` vale semplicemente `x`. In questo modo si riesce a bilanciare il sistema di crescita e decrescita della rete in base ai dati considerati. Di seguito un esempio:

- supponiamo di avere l'i-esimo valore di crescita pari all'8% nel grafico;
- `x` sarà uguale al 2%;
- il valore di `grow_percentages[i]` sarà 8% + `x` = 10%;
- `leave_percentages[i]` varrà `x`, ovvero 2%.

4.2.2 Simulazione degli schemi di follow-up attraverso il rewiring dei nodi

Al fine di introdurre gli schemi di follow-up, caratteristica fondamentale dei social network, sono state necessarie alcune modifiche alla simulazione. Innanzitutto, il parametro `rewire-prob` per l'inizializzazione delle reti Small World viene riutilizzato come la probabilità che un nodo subisca il rewire. In secondo luogo, per effettuare il rewire dei nodi, è necessario avere i loro dati e le informazioni sui collegamenti. Per tale motivo è stato necessario implementare delle funzioni per importare ed esportare lo stato completo della rete in un file *world.csv*, come mostrato nel listato 4.5. Al contrario delle operazioni di aggiunta e rimozione dei nodi, la rewire è stata implementata completamente in Python. Tale funzione, visibile nel listato 4.6, si occupa

di esportare lo stato della simulazione da Netlogo (riga 4), formattare il file per l'utilizzo con la libreria *pandas* (righe 19-31), scorrerlo fino alla sezione dedicata agli agenti (TURTLES), contarli e prelevare l'id del super-agente e l'id più alto dei *basic-agents* (righe 44-64) e, infine, di scorrere fino alla sezione dedicata ai collegamenti (LINKS) e, per ciascuno di loro, avviare la procedura di rewire (righe 66-88), facendo importare lo stato modificato a Netlogo immediatamente dopo (righe 91-94).
La procedura di rewire prevede la generazione di un numero random che, se inferiore alla *rewire_prob*, permette di generare un ulteriore numero compreso tra 0 e il *max_agent_id*, per poi sostituire il vecchio id nel collegamento con quello appena generato.

```
1  to export-data [filename]
2    export-world filename
3  end
4
5  to import-data [filename]
6    import-world filename
7  end
```

Code Listing 4.5: Funzioni per importare ed esportare lo stato della simulazione

```
1  def rewire(self):
2      is_rewiring_active = self.netlogo.get_rewire()
3      rewire_prob = self.netlogo.get_rewire_probability()
4      self.netlogo.export_network("world_{}.csv".format(self.proc_id))
5
6      if(not is_rewiring_active):
7          return False
8
9      # Input
10     data_file = "./netlogo/world_{}.csv".format(self.proc_id)
11
12     # Delimiter
13     data_file_delimiter = ','
14
15     # The max column count a line in the file could have
16     largest_column_count = 0
17
18     # Loop the data lines
19     with open(data_file, 'r') as temp_f:
20         # Read the lines
21         lines = temp_f.readlines()
22
23         for l in lines:
24             # Count the column count for the current line
25             column_count = len(l.split(data_file_delimiter)) + 1
26
27             # Set the new most column count
```

```
28              largest_column_count = column_count if
    largest_column_count < column_count else largest_column_count
29
30      # Generate column names (will be 0, 1, 2, ...,
        largest_column_count - 1)
31      column_names = [i for i in range(0, largest_column_count)]
32
33      # Read csv
34      df = pd.read_csv(data_file, header=None, delimiter=
        data_file_delimiter, names=column_names, low_memory=False)
35
36      begin_index = 0
37      end_index = 0
38      counting_agents = False
39      max_agent_id = 0
40      super_agent_id = 0
41      agent_ids = []
42
43      #read lines one by one
44      for index, row in df.iterrows():
45          if(row[0] == 'TURTLES'):
46              counting_agents = True
47              begin_index = index + 2 #skip "TURTLES" and header
48
49          if(counting_agents and index >= begin_index):
50              if(row[0] == 'PATCHES'):
51                  counting_agents = False
52                  break
53
54              value = int(row[0])
55              if(value > max_agent_id):
56                  max_agent_id = value
57                  #saving every existing id
58                  agent_ids.append(value)
59
60              if(row[8] == '{breed super-agents}'):
61                  super_agent_id = value
62
63      # print(len(agent_ids))
64      rewiring = False
65
66      for index, row in df.iterrows():
67          if(row[0] == 'LINKS'):
68              begin_index = index + 2 #skip "LINK" and header
69              rewiring = True
70
71          if(row[0] == 'PLOTS'):
72              rewiring = False
73              break
74
75          if(rewiring and index >= begin_index):
76              #calculate rewiring probability
77              if(random.random() <= rewire_prob):
78                  #generate a random between 0 and max_agent_id
79                  random_agent_id = random.choice(agent_ids)
80                  current_agent_id = int(row[0].split(' ')[1].split('}')
```

```
    [0])
81
82                  while random_agent_id == current_agent_id:
83                      random_agent_id = random.choice(agent_ids)
84
85                  if random_agent_id != super_agent_id:
86                      df.at[index, 1] = '{basic-agent ' + str(
    random_agent_id) + '}'
87                  else:
88                      df.at[index, 1] = '{super-agent ' + str(
    random_agent_id) + '}'
89
90      #remove header
91      df1 = df.tail(-1)
92      df1.to_csv("./netlogo/world_{}.csv".format(self.proc_id), index=
    False, header=False)
93
94      self.netlogo.import_network("world_{}.csv".format(self.proc_id))
95      return True
```

Code Listing 4.6: Funzione per il rewire dei nodi

4.3 Parallelizzazione delle simulazioni ed analisi del carico computazionale in parallelo

La parallelizzazione delle simulazioni, come anticipato nella sezione 4.1.2, si rende necessaria al fine di abbattere i tempi necessari per la sperimentazione. In questa sezione verranno analizzate sia la soluzione ideata che gli accorgimenti necessari per garantire il corretto funzionamento della simulazione.

Breve introduzione ad OpenMPI *OpenMPI* è un'implementazione di MPI (Message Passing Interface), un protocollo per il calcolo parallelo nei sistemi a memoria distribuita. Si tratta di uno strumento sviluppato e mantenuto da un consorzio di ricercatori pubblici e privati, oltre che da aziende partner. L'obiettivo di *OpenMPI* è quello di rappresentare la migliore implementazione di MPI.

4.3.1 Introduzione della computazione parallela nel sistema

Per l'implementazione della computazione parallela all'interno dell'ambiente di simulazione è stato necessario rivedere l'architettura del sistema. Lo strumento scelto per effettuare il calcolo parallelo è *OpenMPI*, la cui compatibilità con *Python* è garantita dalla libreria *mpi4py*, che astrae tutti i

passaggi per tradurre il codice *Python* in codice compatibile con *OpenMPI*. Nella figura 4.2 viene mostrata la nuova architettura del sistema.

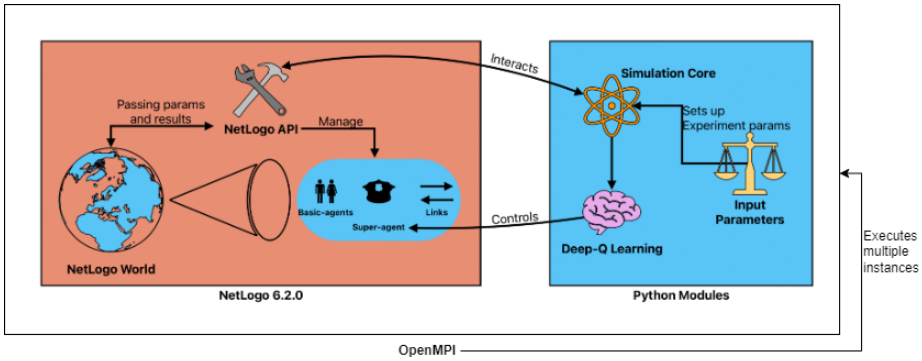


Figura 4.2: Abstract dell'architettura del sistema dopo l'aggiunta di OpenMPI.

Al fine di utilizzare *OpenMPI* nella simulazione, sono stati introdotte delle modifiche allo script di base. Tali modiiche sono visibili nel listato 4.7, che mostra un esempio di script per l'avvio di una simulazione in parallelo. L'inizializzazione della simulazione prevede anche l'inizializzazione dell'ambiente parallelo, creando i parametri necessari a gestire i vari processi. La simulazione prosegue poi normalmente, con l'unica accortezza di gestire le stampe a video ed il salvataggio dei file, inserendo il processo che le sta eseguendo nel messaggio o nel nome del file.

```
1
2  def main():
3      # Initialize MPI
4      comm = MPI.COMM_WORLD
5      rank = comm.Get_rank()
6      size = comm.Get_size() # Number of processes
7      plotTitle = ""
8
9      if size == 8:
10         if rank == 0:
11             params.setWarningWeight(int(params.getWarningWeight()/2))
12             plotTitle = "DeepQLearning with half Warning weight"
13         if rank == 1:
14            params.setWarningWeight(int(params.getWarningWeight()*2))
15            plotTitle = "DeepQLearning with double Warning weight"
16         if rank == 2:
17             params.setReiterateWeight(int(params.getReiterateWeight()
       /2))
18             plotTitle = "DeepQLearning with half Reiterate weight"
19         if rank == 3:
20              params.setReiterateWeight(int(params.getReiterateWeight()
       *2))
21              plotTitle = "DeepQLearning with double Reiterate weight"
```

```
22          if rank == 4:
23              params.setStaticWeight(int(params.getStaticWeight()/2))
24              plotTitle = "DeepQLearning with half Static weight"
25          if rank == 5:
26              params.setStaticWeight(int(params.getStaticWeight()*2))
27              plotTitle = "DeepQLearning with double Static weight"
28          if rank == 6:
29              params.setGoWeight(int(params.getGoWeight()/2))
30              plotTitle = "DeepQLearning with half Go weight"
31          if rank == 7:
32              params.setGoWeight(int(params.getGoWeight()*2))
33              plotTitle = "DeepQLearning with double Go weight"
34      else:
35          if rank == 0:
36              print("Unexpected number of processors")
37          return
38
39      #Inizializzazione della simulazione
40      [...]
41
42
43      #Avvio della simulazione
44      [...]
45      for episode in range(total_training_episodes):
46          [...]
47              if terminated:
48                  print("[" + str(rank) + ']Total training rewards: {}
    after n steps = {} with final reward = {}'.format(
    total_training_rewards, episode, reward))
49
50                  if steps_to_update_target_model >= 100:
51                      print("[" + str(rank) + ']Copying main network
    weights to the target network weights')
52                      target_model.set_weights(model.get_weights())
53                      steps_to_update_target_model = 0
54                  break
55
56
57      #Calcolo delle reward
58      [...]
59
60      #Testing della simulazione
61      [...]
62
63      total_testing_time = (time.time() - start_time)/60
64      print("[" + str(rank) + "] Training time %s minutes ---" %
    total_training_time)
65      print("[" + str(rank) + "] Testing time %s minutes ---" %
    total_testing_time)
66      print("[" + str(rank) + "] Totaltime {} minutes ---".format(
    total_training_time+total_testing_time))
67
68
69      rewards_to_plot = [[reward[0] for reward in rewards] for rewards
    in total_rewards]
70      df1 = pd.DataFrame(rewards_to_plot).melt()
```

```
71      df1.rename(columns={"variable": "episodes", "value": "reward"},
        inplace=True)
72      sns.set(style="darkgrid", context="talk", palette="rainbow")
73      sns.lineplot(x="episodes", y="reward", data=df1, ax=axis[0], label
        ="test rewards").set(
74          title= plotTitle
75      )
76      df2 = pd.DataFrame({"episodes": [i for i in range(
        total_test_episodes)], "global cascade": global_cascade})
77      sns.lineplot(ax = axis[1], x = "episodes", y = "global cascade",
        data = df2, label="global cascade").set(
78          title="Global cascade during test " + str(rank)
79      )
80      legend = plt.legend()
81      plt.savefig(savepath + "/process" + str(rank) + ".png")
82
83      env.close()
84      netlogo.kill_workspace()
85
86  if __name__ == '__main__':
87      main()
```

Code Listing 4.7: Esempio di simulazione con super-agente eseguibile in parallelo

Per ciò che concerne le simulazioni di test, i cui risultati verranno analizzati nel Capitolo 5, sono stati presi accorgimenti inerenti alla gestione dei parametri da passare in input alle varie istanze della simulazione, al salvataggio dei risultati ed alla gestione dei file contenenti lo stato della simulazione (per la funzione di *rewire*, ad esempio).

Per lanciare una simulazione è stato creato uno script che si occupa di eseguire il comando necessario all'avvio delle varie simulazioni tramite MPI, inserendo il numero di processo specificato dall'utente, come è possibile vedere dal listato 4.8.

```
1  import os
2
3  proc_num = '8'
4
5  os.execvp('mpiexec', ['mpiexec', '-np', proc_num, 'python', '
       deepq_simulation.py'])
```

Code Listing 4.8: Script per l'invocazione del comando mpirun

4.3.2 Analisi del consumo di risorse in parallelo

L'esecuzione di più simulazioni in parallelo ha un consumo di risorse computazionali molto più elevato, rispetto ad una normale simulazione. Ciò è ulteriormente incentivato dalle caratteristiche introdotte nella rete.

Le simulazioni effettuate utilizzando questo modello, infatti, sono *RAM expensive*, ovvero presentano un consumo molto elevato di memoria principale. Eseguendo le simulazioni in parallelo, non è più bastata una macchina domestica con 64 Gb di RAM, ma è stato necessario passare ad una macchina equipaggiata come descritto nella tabella 6.1. Dagli esperimenti effettuati, è emerso che il massimo numero di simulazioni eseguibili simultaneamente è pari a 10. Oltre questo limite, la RAM viene completamente saturata e le simulazioni si arrestano.

4.4 Limitazioni del modello e possibili sviluppi futuri

Nonostante le modifiche apportate al modello originale permettano di ottenere un sostanziale avvicinamento alle dinamiche del mondo reale ed un aumento di velocità nell'esecuzione degli esperimenti, possono essere aggiunti ulteriori elementi per rendere la simulazione ancora più realistica.

Bias cognitivi I bias sono una deviazione della razionalità nei processi cognitivi. Rappresentano una forma di adattamento mentale a situazioni di potenziale disagio. Nell'ambito delle fake news, i bias rappresentano un fattore fondamentale per la loro diffusione. Nello specifico, gli utenti tendono a manifestare forme di bias cognitivi a causa della preoccupazione per determinati eventi.
La modellazione e l'implementazione dei bias cognitivi nella simulazione implicherebbe un ulteriore avvicinamento alle dinamiche reali, consentendo simulazioni più accurate.

Criteri di rewire La funzione `rewire` consente di modificare i collegamenti tra i nodi, simulando i cambiamenti che avvengono nelle reti sociali reali, in cui gli interessi degli utenti cambiano e, di conseguenza, cambiano anche le persone da cui questi ultimi scelgono di ricevere informazioni. Al fine di migliorare ulteriormente la simulazione, potrebbero essere introdotti diversi criteri di selezione dei nodi da ricollegare ed ulteriori criteri per scegliere i nodi a cui collegarli. Ad esempio, potrebbe essere utilizzato come criterio la selezione randomica di uno dei 5 nodi con grado maggiore, al fine di simulare la capacità di un *influencer* di polarizzare l'opinione degli utenti.

Capitolo 5

Esperimenti

In questo capitolo verranno illustrati gli esperimenti condotti sul modello simulativo a cui sono state apportate le modifiche discusse nel Capitolo 4. Gli esperimenti eseguiti ricalcalno quelli con super agente presenti nella tesi magistrale del dott. Perfetto [4]. Lo scopo della sperimentazione è quello di confrontare i risultati ottenuti con il modello modificato con quelli ottenuti in passato, per evidenziare i cambiamenti ottenuti e gli eventuali miglioramenti.

La sperimentazione è stata svolta su una macchina equipaggiata come in Tabella 6.1, utilizzando *Visual Studio Code* come IDE e *OpenMPI* per la parallelizzazione degli esperimenti. In particolare, per la sperimentazione sono state eseguite 8 simulazioni in parallelo per ciascun esperimento.

5.1 Studio della diffusione virale delle fake news in un social network in presenza di una rete dinamica

In questa sezione verranno analizzati i risultati ottenuti con gli esperimenti eseguiti in parallelo sul modello a cui sono stati aggiunte le funzionalità di crescita, decrescita e modifica dei collegamenti tra nodi. In ogni test si fa uso del *super-agent* e si effettuano analisi su 100 episodi da 100 tick ciascuno. Grazie alla parallelizzazione delle simulazioni è stato possibile ottenere un incremento nella velocità di esecuzione degli esperimenti. In particolare, si ottiene una fattore di riduzione pari a $\frac{1}{n}$, con n che rappresenta il numero di esperimenti svolti in parallelo. In sintesi, è possibile eseguire n esperimenti nel tempo impiegato da un solo esperimento, fintanto che le risorse della macchina utilizzata non si esauriscono.

Per l'esecuzione degli esperimenti sono stati utilizzati i valori di default impiegati per la sperimentazione effettuata dal dott.Perfetto, come riportato in tabella 5.1

Attributo	**Valore Default**
`sa-delay`	2
`node-range-static-b`	0.05
`node-range`	0.10
`global-warning`	*True*
`choose-method`	*degree*
`warning-impact`	0.10
`warning-impact-neutral`	0.30
`growth-percentages`	[10, 15, 14, 26, 17, 10, 10, 16, 14, 5]
`leave-percentages`	[2, 3, 3, 5, 2, 2, 3, 3, 4, 2]

Tabella 5.1: Valori di default degli esperimenti in input al modello simulativo

Durante lo svolgimento dei test viene monitorata la *viralità*, il cui valore è inversamente proporzionale alla desiderabilità delle azioni intraprese dal *super-agent*.

Ogni test risponde ad un precisa domanda di ricerca. Di seguito verranno mostrati i test eseguiti.

5.1.1 Qual è l'impatto della *rete dinamica* sulla Viralità al variare della network polarization e credulonerìa?

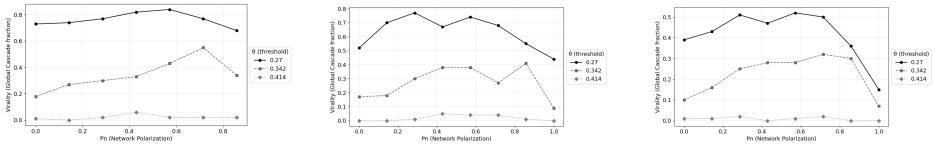
In questo test si valuta la diffusione delle fake news in termini di viralità (indicata con il valore V) al variare dei parametri P_n e θ, con il *super-agent* su rete dinamica.

In particolare, dalla Fig. 5.1 si può notare che il test viene eseguito con `sa-delay` fissato a 5 (a), 4 (b) e 2 (c).
I valori di credulonerìa considerati sono $\Theta = \{0.270, 0.342, 0.414\}$.
I risultati sono riportati in Fig. 5.1.

Con `sa-delay` fissato a 5, la viralità tende a crescere fino ad determinati valori di P_n, specificamente $P_n = 0.6$ per $\theta = 0.27$ e $P_n = 0.7$ per $\theta = 0.342$, mentre per $\theta = 0.414$ non sussistono forti cambiamenti nel trend. Per quanto concerne i valori di viralità, invece, vi è un aumento dei valori inversamente proporzionale alla soglia di θ. Ciò è dovuto al fatto che con θ più alto, gli

agenti risultano meno suscettibili a cambiare opinione.
Per `sa-delay` $= 4$, la situazione è simile al caso precedente, ma i valori a partire dai quali la curva di viralità tende a calare cambiano. Nello specifico, la viralità cala da $P_n = 0.6$ per $\theta = 0.27$ e da $P_n = 0.85$ per $\theta = 0.342$. Infine, per `sa-delay` $= 2$ i valori di viralità calano drasticamente a partire rispettivamente da $P_n = 0.75$ per $\theta = 0.27$ e da $P_n = 0.85$ per $\theta = 0.342$.

Dai test eseguiti si può notare che, in generale, al diminuire di sa-delay i valori di viralità diminuiscono. Specificamente, con sa-delay $= 2$ la curva di viralità non arriva mai al di sopra di 0.5, ovvero la soglia di viralità.



(a) Con `sa-delay` $= 5$. (b) Con `sa-delay` $= 4$. (c) Con `sa-delay` $= 2$.

Figura 5.1: Andamento medio della Viralità delle fake news, su rete dinamica, al variare di P_n e Θ.

Nella figura 5.2 vengono illustrate le differenze tra i test condotti senza rete dinamica e senza *super-agent* e quelli condotti su rete dinamica al variare di `sa-delay`. Come già precedentemente affermato, i valori di viralità risultano essere inversamente proporzionali a quelli di `sa-delay`. Rispetto alla rete statica, dunque, la rete dinamica introduce una maggiore possibilità di diffusione delle fake news, ma al contempo il *super-agent* riesce a contrastarne la diffusione.

5.1.2 Qual è l'impatto della *rete dinamica* sulla Viralità al variare della opinion polarization?

Questo esperimento permette un'analisi della *viralità* al variare dell'*opinion polarization* (P_o).

I risultati sono riportati in Fig. 5.3, in cui si evidenziano i cambiamenti nella viralità al variare della network polarization, con P_o fissato rispettivamente a 0.15 (a) e 0.27 (b). L'andamento degli esperimenti con $\theta = 0.27$ presenta valori di viralità superiori alla soglia di 0.5, che va al di sotto di tale soglia a partire rispettivamente da $P_n = 0.8$ e $P_n = 0.85$. Per $\theta = 0.342$

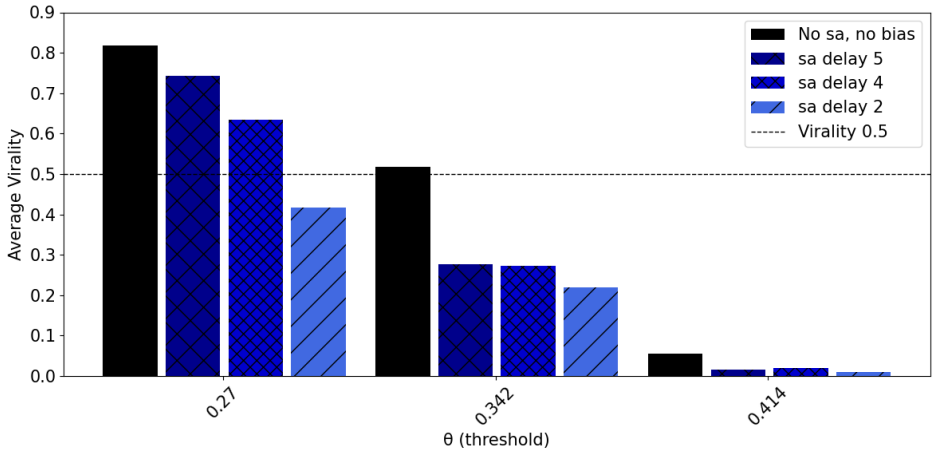
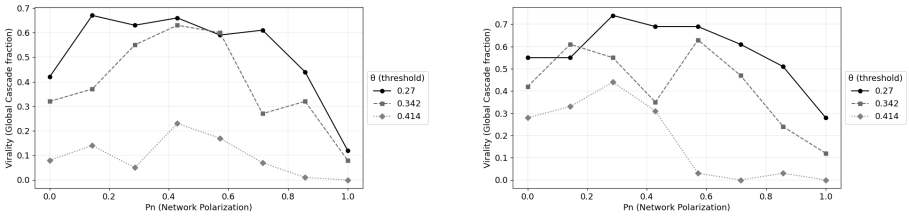


Figura 5.2: Media della Viralità raggiunta al variare di Θ (asse x) negli esperimenti senza super-agent ("No sa") e rete dinamica (no dynamic) e su rete dinamica.

i valori di viralità sono inizialmente superiori alla soglia di 0.5 e presentano una crescita più rapida, con l'unico crollo di tale valore per $P_n = 0.45$ nella Fig.5.3(b). In questo caso, la decrescita della curva risulta molto più repentina rispetto a $\theta = 0.27$. Infine, per $\theta = 0.414$ sono presenti valori di viralità sempre inferiori a 0.5, con valori molto bassi in Fig.5.3(a) e valori inizialmente più alti, che poi decrescono velocemente ed assumono valori di viralità inferiori allo 0.1 a partire da $P_n = 0.25$.



(a) $P_o = 0.15$. (b) $P_o = 0.27$.

Figura 5.3: Andamento medio della Viralità delle fake news, su rete dinamica, al variare di P_n e Θ.

Confrontati tra loro in Fig.5.4, i valori dei tre esperimenti mostrano che più alto è il valore di P_o, maggiore è la viralità raggiunta durante la simulazione. All'innalzarsi della soglia di credulonería, inoltre, i valori tendono a calare. La soglia di viralità $V = 0.5$ viene raggiunta solo con $\theta = 0.27$ e $P_o = 0.15$ oppure $P_o = 0.27$, risultando inferiore in tutte le altre casistiche.

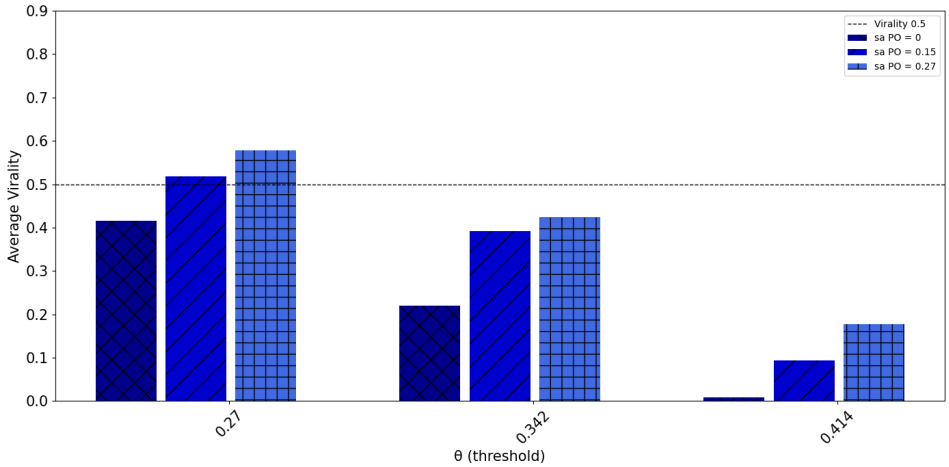


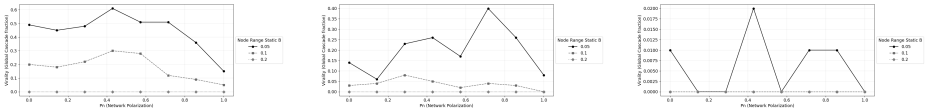
Figura 5.4: Media della Viralità raggiunta al variare di Θ (asse x) negli esperimenti senza super-agent ("No sa") e rete dinamica e con rete dinamica.

5.1.3 Qual è l'impatto della *rete dinamica* sulla Viralità al variare del parametro node-range-static-b?

In questo esperimento si analizza l'andamento della viralità al variare del parametro *node-range-static-b*, ovvero il parametro che permette di definire su quanti agenti avrà effetto l'azione *Static B Nodes*. I risultati sono riportati in Fig. 5.5.

Da quanto emerso dagli esperimenti, la viralità tende ad aumentare solo con la soglia più bassa 5.5(a), mentre con gli altri due valori ha una crescita inferiore o nulla. Da notare che nelle ultime due casistiche (Fig.5.5(b) e Fig.5.5(c)) i valori di viralità non superano mai la soglia di 0.5, ovvero la soglia che permette di considerare la notizia come virale. In particolare, il picco massimo di viralità raggiunto è pari a 0.4 con $\theta = 0.342$, mentre con $\theta = 0.414$ è pari a 0.02.
Il parametro *node-range-static-b* ha quindi un impatto molto significativo

sulla viralità, arrivando a contrastare agevolmente la diffusione delle fake news con valori molto bassi.



(a) Con $\Theta = 0.270$. (b) Con $\Theta = 0.342$. (c) Con $\Theta = 0.414$.

Figura 5.5: Andamento medio della Viralità delle fake news, su rete dinamica, al variare di P_n, θ e `node-range-static-b`.

Per ciò che concerne gli altri esperimenti effettuati, si evince chiaramente come la rete dinamica permetta di avere valori di viralità nettamente inferiori rispetto alle altre simulazioni. In particolare, i valori di viralità sono sempre molto al di sotto dello 0.5. Con l'aumentare di θ, inoltre, tali valori risultano sempre in decrescita.

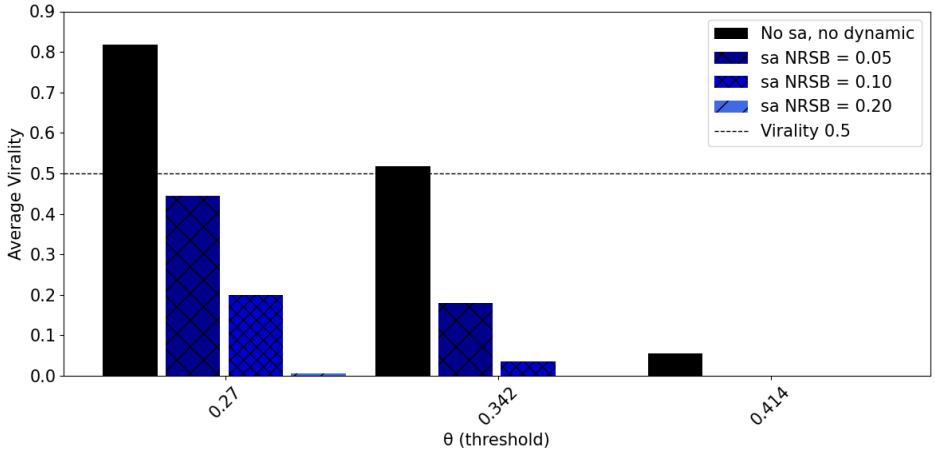


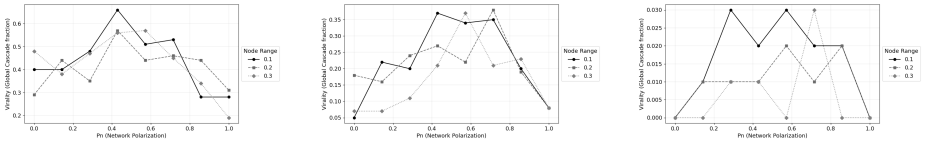
Figura 5.6: Media della Viralità raggiunta al variare di Θ (asse x) negli esperimenti senza super-agent ("No sa") e rete dinamica e con rete dinamica al variare di `node-range-static-b` *(NRSB).*

5.1.4 Qual è l'impatto della *rete dinamica* sulla Viralità al variare del parametro node-range?

In questa sezione si analizza, al variare di `node-range`, l'influenza della *rete dinamica* sulla Viralità.

Il parametro `node-range` permette di scegliere la frazione dei nodi su cui il *super-agent* interviene quando attiva l'operazione *Warning* (non globale) o *Reiterate*, se il *Warning*.
Per questo esperimento sono stati usati i seguenti valori: `node-range` = $\{0.10, 0.20, 0.30\}$.
I risultati sono riportati in Fig. 5.7.

Dai test eseguiti, emerge che il numero di nodi su cui agisce il *super-agent* è poco rilevante per l'abbattimento della viralità. In particolare, i risultati dimostrano che l'azione `Reiterate` ha una minore influenza sull'andamento della simulazione rispetto al `Warning`. Ciò è principalmente dovuto alla durata degli effetti della prima azione rispetto a quelli della seconda, i quali durano fino al termine della simulazione. Tuttavia, vi è un miglioramento dei valori di viralità raggiunti, quando sia `node-range` che θ crescono, arrivando a raggiungere valori di viralità molto prossimi allo zero (5.7(c)).



(a) Con $\Theta = 0.270$ (b) Con $\Theta = 0.342$ (c) Con $\Theta = 0.414$

Figura 5.7: Andamento medio della Viralità delle fake news, su rete dinamica, al variare di P_n e `node-range`.

Dal grafico a barre in Fig.5.8 si evince come la combinazione di *rete dinamica* e *super-agent* produca risultati molto migliori rispetto ai risultati ottenuti in passato. In particolare, i valori di viralità sono sempre inferiori alla soglia dello 0.5 e tendono a diminuire all'aumentare di θ, poichè gli agenti sono meno inclini a cambiare opinione.

5.1.5 Qual è l'impatto della *rete dinamica* sulla Viralità al variare dei parametri legati al warning?

In questa sezione si analizza l'impatto del *super-agent* su rete dinamica, al variare dei parametri legati all'azione *Warning*, cioè `warning-impact` e

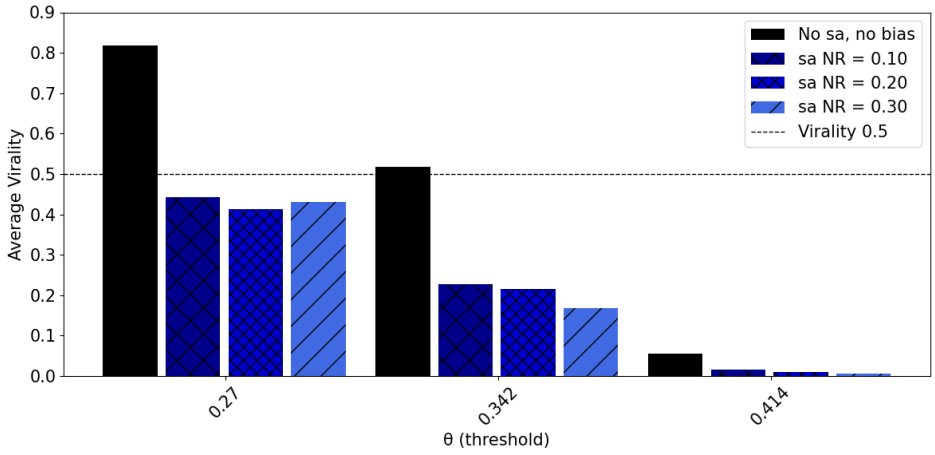


Figura 5.8: Media della Viralità raggiunta al variare di Θ *(asse x) negli esperimenti senza super-agent ("No sa") e rete dinamica e con rete dinamica al variare di* `node-range` *(NR).*

`warning-impact-neutral`, che definiscono il modo in cui l'azione riesce ad influenzare gli agenti.

Per questo esperimento sono stati utilizzati `warning-impact` $= \{0.1, 0.2\}$ e `warning-impact-neutral` $= \{0.3, 0.5\}$.

I test eseguiti sono 3, in particolare:

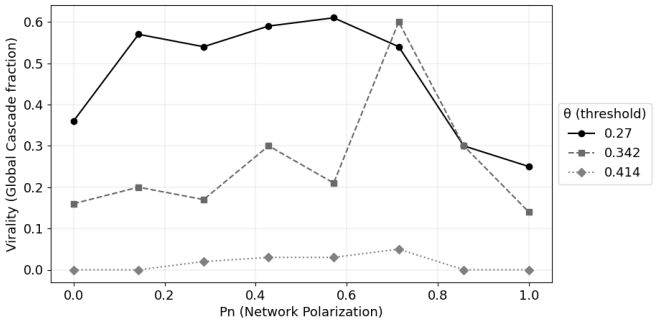
- `warning-impact` $= 0.2$ e `warning-impact-neutral` $= 0.3$
- `warning-impact` $= 0.1$ e `warning-impact-neutral` $= 0.5$
- `warning-impact` $= 0.2$ e `warning-impact-neutral` $= 0.5$

L'obiettivo dei test è analizzare quale dei due parametri abbia maggiore influenza sulla rete dinamica.
I risultati sono riportati in Fig. 5.9. Dal test mostrato in Fig.5.9(a) si può notare che il con $\theta = 0.27$ i valori di Viralità rimangono elevati, quasi sempre al di sopra di 0.5, mentre per gli altri due valori, la viralità tende a rimanere bassa, tranne per un picco con $\theta = 0.342$. Nei restanti due esperimenti (Fig.5.9(b) e Fig.5.9(c)), in cui il valore di `warning-impact-neutral` viene incrementato, i valori di viralità con $\theta = 0.27$ sono più bassi, mentre con $\theta = 0.342$ presentano un andamento più controllato. Ciò dimostra che su *rete dinamica* il parametro `warning-impact-neutral` ha maggiore influenza sulla viralità rispetto a `warning-impact`. Riuscire a controllare gli agenti

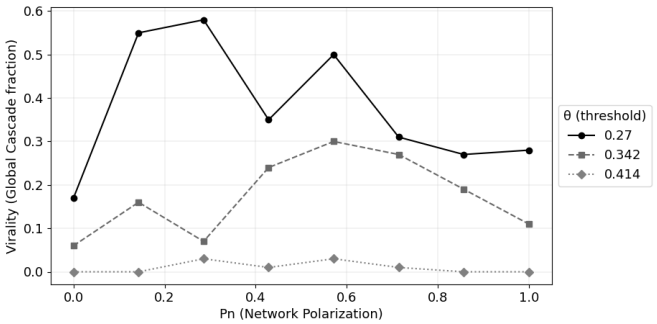
neutrali, dunque, risulta più efficace che riuscire a controllare gli agenti suscettibili alle fake news.

Dal grafico in Fig. 5.10 si può notare quanto affermato finora.

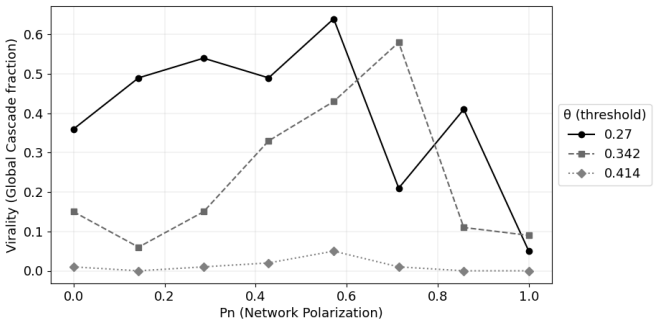
In conclusione, dagli esperimenti si evince che il parametro `warning-impact-neutral` ha una maggiore incidenza nel controllo delle fake news rispetto a `warning-impact`. Ciò è dato dal fatto che gli agenti aggiunti tramite la funzione di crescita sono sempre inizialmente neutrali.



(a) Con `warning-impact` $= 0.2$, `warning-impact-neutral` $= 0.3$.



(b) Con `warning-impact` $= 0.1$ e `warning-impact-neutral` $= 0.5$.



(c) Con `warning-impact` $= 0.2$ e `warning-impact-neutral` $= 0.5$.

Figura 5.9: Andamento medio della Viralità delle fake news su rete dinamica, in presenza di super-agent, al variare di P_n e Θ.

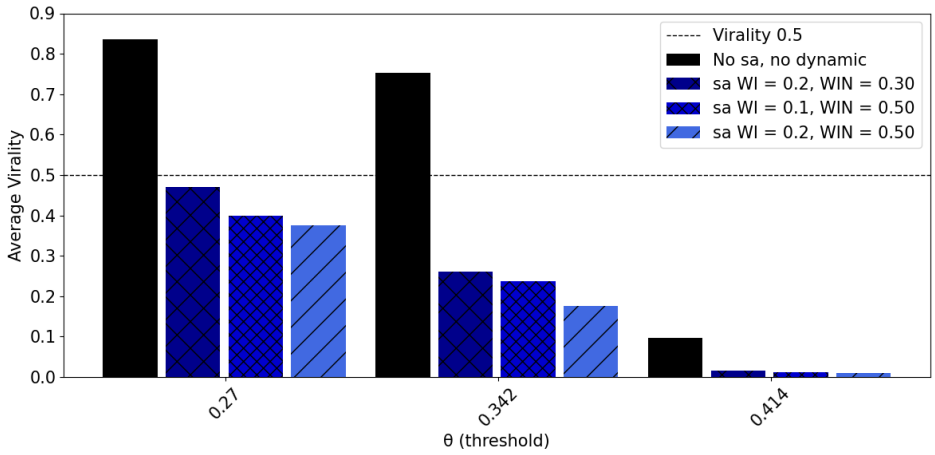


Figura 5.10: Media della Viralità raggiunta al variare di Θ (asse x) negli esperimenti senza super-agent ("No sa") e rete dinamica e con rete dinamica al variare di `warning-impact` *(WI) e* `warning-impact-neutral` *(WIN).*

Capitolo 6

Conclusioni

Grazie alla crescita dei social network avvenuta negli ultimi anni, gli utenti hanno avuto accesso ad una potente fonte di informazione, più veloce e maggiormente fruibile dei mezzi tradizionali. La larga diffusione di dispositivi smart ha ampliato il fenomeno, permettendo alle persone di informarsi anche attraverso le app dei vari social network installate sugli smartphone. Tuttavia, data la facilità con cui queste informazioni riescono a diffondersi, alcuni utenti malintenzionati hanno trovato una soluzione semplice e veloce per riuscire a diffondere fake news.
In questo scenario, l'assenza di un elemento che controlli la veridicità delle informazioni che circolano in rete, porta a problemi consistenti, tra cui la mancata gestione ed identificazione delle fake news e la possibilità per alcuni utenti di credere in questi contenuti. Quando un cospicuo numero di utenti con le stesse idee si raggruppa, nascono le cosiddette *echo chambers*, da cui poi vengono generate vere e proprie campagne di disinformazione.
Attraverso l'approccio ibrido tra sistemi *model-driven* e *data-driven* ha dimostrato come sia possibile creare uno scenario simulativo con un *super-agent* in grado di contrastare attivamente la viralità delle fake news tramite le azioni applicate sugli agenti di una rete sociale simulata. Ciò è stato possibile grazie ai meccanismi di Deep Reinforcement Learning che consentono al *super-agent* di apprendere dalle proprie azioni passate per prendere decisioni in tempo reale, basandosi sullo stato della simulazione.
In questo lavoro di tesi si è migliorato questo approccio andando ad implementare caratteristiche tipiche delle reti sociali reali ed un sistema di parallelizzazione degli simulazioni per velocizzare la fase di sperimentazione. In particolare, sono state implementate le funzionalità che permettono di simulare le *curve di crescita* delle reti sociali ed i sistemi di *follow-up* per

gli utenti dei social network. La parallelizzazione è stata eseguita attraverso *OpenMPI* ed ha permesso di eseguire più simulazioni contemporaneamente, abbattendo i tempi di esecuzione della sperimentazione, al costo di accorgimenti nella gestione della computazione parallela negli algoritmi già esistenti e quelli implementati.
I risultati ottenuti dimostrano che la *rete dinamica* presenta una maggiore difficoltà di gestione delle fake news per il *super-agent*, rispetto alla versione statica del modello simulativo. Ciò è dovuto dal maggior numero di agenti e dal fatto che i collegamenti tra questi ultimi variano nel tempo. Tuttavia, ciò permette di sperimentare il *super-agent* su uno scenario più realistico. Infine, la parallelizzazione ha abbattuto notevolmente i tempi di sperimentazione. Lo strumento è ora più efficace per la ricerca nell'ambito del contrasto delle fake news e rappresenta un metodo per sperimentare dinamiche complesse o irrealizzabili nel mondo reale.

6.1 Limitazioni e sviluppi futuri

Nonostante il modello implementato sia più ricco di altri (ad esempio, [13]), il lavoro svolto presenta ancora delle limitazioni. In particolare le limitazioni presenti si riferisco:

- al modello simulativo;
- al sistema di DRL;
- alla sperimentazione svolta.

Di seguito viene presentata una panoramica delle limitazioni e delle possibili migliorie apportabili.

- *Modello simulativo*: il modello implementato presenta alcune lacune nella gestione delle azioni degli agenti, in particolare i *basic-agents*. Questi ultimi, come affermato in [26], possono essere rappresentati da reti bayesiane ed avere *bias cognitivi* che ne modellino il comportamento nella simulazione, specificamente *bias* sull'autorevolezza delle fonti e sull'accuratezza delle informazioni ricevute. Stando a quanto affermato in [27], infatti, gli utenti tendono a considerare caratteristiche quali l'affidabilità e l'accuratezza delle fonti da cui scelgono di ricevere informazioni. I bias cognitivi, tuttavia, sono numerosi e possono essere implementati per sperimentare scenari ancora più realistici[14]. Tra i bias implementabili troviamo:

  - *Confirmation bias*;
  - *Emotional bias*;
  - *Congruence Bias*;
  - *Attentional bias*;
  - *Belief bias*;
  - *Selective exposure.*

  Al fine di replicare il funzionamento di una rete sociale reale, potrebbero essere sfruttati strumenti in grado di prelevare informazioni dai social network più usati, come ad esempio le *Twitter API*[1]. Lo schema di cambio collegamenti implementato, infine, present un criterio di modifica dei collegamenti completamente stocastico. L'implementazione di un sistema di rewiring basato su criteri quali l'interesse degli agenti per i contenuti pubblicati da una certa fonte possono avvicinare ulteriormente il modello alle dinamiche del mondo reale.

- *Sistema di DRL*: il sistema di Deep Reinforcement Learning potrebbe essere migliorto implementando un migliore sistema di *reward*. In particolare, fornendo un peso ad ogni azione eseguita in un particolare contesto, si può fornire un migliore addestramento al *super-agent* grazie ad un sistema di reward che si basa su tale peso. Ad esempio, eseguire un'azione di warning ha un peso contenuto, poichè è necessario semplicemente contattare l'agente interessato e segnalare l'azione eseguita su di esso. In uno scenario reale, tale azione potrebbe corrispondere all'avviso di rimozione dei contenuti attuato dalle piattaforme di social network. Un'azione *Static B Nodes*, invece, sarebbe molto più onerosa da applicare nel mondo reale, quindi avrebbe un peso sicuramente maggiore, basato anche sul numero di agenti nella rete. Un'altra possibile miglioria è permettere al *super-agent* di tenere traccia dell'evoluzione del network col passare dei *tick*, andando a ricreare la struttura della rete simulata in *Netlogo* con un grafo in *OpenAI Gymnasium*, fornendo quindi all'agente un addestramento più accurato ed un migliore calcolo delle *reward*. Un ulteriore miglioramento è l'implmentazione di nuove azioni, sulla base degli ulteriori problemi collegati alla diffusione delle fake news [28]. Infine, un miglioramento potrebbe essere ottenuto con un'operazione di tuning sia sulle azioni che sui parametri del*super-agent*.

[1]Twitter (ora rinominato X) ha interrotto l'accesso alle proprie API per la comunità dei ricercatori a partire da Marzo 2023, permettendo solo ai sottoscrittori di un abbonamento Enterprise di avere accesso completo alle API

- *Sperimentazione*: attraverso la discretizzazione del dominio dei parametri della simulazione, il numero di esperimenti eseguibili per il modello simulativo è pari a 10^{23}, mentre per il *super-agent* è pari a 10^3, grazie a ben 448 parametri modificabili. Il tutto permette un numero di esperimenti totale pari a $4,4 * 10^{25}$ [4], a cui si aggiungono i nuovi parametri per la gestione della rete dinamica. Per questa sperimentazione è stata utilizzata unicamente una rete di tipo *Erdős–Rényi*, con archi indiretti (e quindi informazioni scambiate in modo bidirezionale tra gli agenti). Sarebbe interessante, ai fini della ricerca scientifica, analizzare i comportamenti degli agenti con altre tipologie di rete, in combinazione con archi sia diretti che indiretti e con rete statica o dinamica.

Bibliografia

[1] Anna Gausen, Wayne Luk e Ce Guo.
≪Can we stop fake news? using agent-based modelling to evaluate countermeasures for misinformation on social media≫.
In: *International AAAI Conference on Web and Social Media (ICWSM). https://doi. org/10.36190*. 2021.

[2] Giancarlo Ruffo et al. ≪Surveying the research on fake news in social media: a tale of networks and language≫.
In: *arXiv preprint arXiv:2109.07909* (2021).

[3] Marcella Tambuscio et al. ≪Network segregation in a model of misinformation and fact-checking≫.
In: *Journal of Computational Social Science* 1 (2018), pp. 261–275.

[4] Ciro Perfetto. *Analisi e contrasto delle dinamiche di diffusione delle fake news. Integrazione euristica di modelli agent-based e deep reinforcement learning*. 2023.

[5] Roberto Abbruzzese et al. ≪Detecting influential news in online communities: an approach based on hexagons of opposition generated by three-way decisions and probabilistic rough sets≫.
In: *Information Sciences* 578 (2021), pp. 364–377.

[6] Nicola Capuano et al. ≪Content Based Fake News Detection with machine and deep learning: a systematic review≫.
In: *Neurocomputing* (2023).

[7] Mohamed K Elhadad, Kin Fun Li e Fayez Gebali.
≪Fake news detection on social media: a systematic survey≫.
In: *2019 IEEE Pacific Rim conference on communications, computers and signal processing (PACRIM)*. IEEE. 2019, pp. 1–8.

[8] Wenlin Han e Varshil Mehta. ≪Fake news detection in social networks using machine learning and deep learning: Performance evaluation≫. In: *2019 IEEE International Conference on Industrial Internet (ICII)*. IEEE. 2019, pp. 375–380.

[9] Rohit Kumar Kaliyar, Anurag Goswami e Pratik Narang. ≪FakeBERT: Fake news detection in social media with a BERT-based deep learning approach≫. In: *Multimedia tools and applications* 80.8 (2021), pp. 11765–11788.

[10] Syed Ishfaq Manzoor, Jimmy Singla et al. ≪Fake news detection using machine learning approaches: A systematic review≫. In: *2019 3rd international conference on trends in electronics and informatics (ICOEI)*. IEEE. 2019, pp. 230–234.

[11] Andrea Renda. *The legal framework to address" fake news": Possible policy actions at the EU level*. European Parliament, 2018.

[12] Walter Quattrociocchi, Antonio Scala e Cass R Sunstein. ≪Echo chambers on Facebook≫. In: *Available at SSRN 2795110* (2016).

[13] Petter Törnberg. ≪Echo chambers and viral misinformation: Modeling fake news as complex contagion≫. In: *PLOS ONE* 13.9 (set. 2018), pp. 1–21. DOI: 10.1371/journal.pone.0203958. URL: https://doi.org/10.1371/journal.pone.0203958.

[14] Giancarlo Ruffo et al. ≪Studying fake news spreading, polarisation dynamics, and manipulation by bots: A tale of networks and language≫. In: *Computer Science Review* 47 (2023), p. 100531. ISSN: 1574-0137. DOI: https://doi.org/10.1016/j.cosrev.2022.100531. URL: https://www.sciencedirect.com/science/article/pii/S157401372200065X.

[15] Raymond S. Nickerson. ≪Confirmation Bias: A Ubiquitous Phenomenon in Many Guises≫. In: *Review of General Psychology* 2.2 (1998), pp. 175–220. DOI: 10.1037/1089-2680.2.2.175. eprint: https://doi.org/10.1037/1089-2680.2.2.175. URL: https://doi.org/10.1037/1089-2680.2.2.175.

[16] J Mark G Williams et al.
Cognitive psychology and emotional disorders. Vol. 2.
Wiley Chichester, 1997.

[17] Lisa Feldman Barrett e Moshe Bar.
≪See it with feeling: affective predictions during object perception≫.
In: *Philosophical Transactions of the Royal Society B: Biological Sciences* 364.1521 (2009), pp. 1325–1334.

[18] Jacqueline P Leighton, Robert J Sternberg et al.
The nature of reasoning. Cambridge University Press, 2004.

[19] Richard Nadeau, Edouard Cloutier e J.-H. Guay.
≪New Evidence About the Existence of a Bandwagon Effect in the Opinion Formation Process≫.
In: *International Political Science Review* 14.2 (1993), pp. 203–213.
DOI: 10.1177/019251219301400204.
URL: https://doi.org/10.1177/019251219301400204.

[20] Lee Ross, David Greene e Pamela House.
≪The "false consensus effect": An egocentric bias in social perception and attribution processes≫. In: *Journal of Experimental Social Psychology* 13.3 (1977), pp. 279–301. ISSN: 0022-1031.
DOI: https://doi.org/10.1016/0022-1031(77)90049-X.
URL: https://www.sciencedirect.com/science/article/pii/002210317790049X.

[21] Lee Ross e Andrew Ward. ≪Naive Realism: Implications for Social Conflict and Misunderstanding≫. In: gen. 1996, pp. 103–135.

[22] C. Nathan DeWall e Brad J. Bushman.
≪Social Acceptance and Rejection: The Sweet and the Bitter≫. In: *Current Directions in Psychological Science* 20.4 (2011), pp. 256–260.
DOI: 10.1177/0963721411417545.
URL: https://doi.org/10.1177/0963721411417545.

[23] Volodymyr Mnih et al.
Playing Atari with Deep Reinforcement Learning. 2013.
arXiv: 1312.5602 [cs.LG].

[24] Marc Jaxa-Rozen e Jan H. Kwakkel.
≪PyNetLogo: Linking NetLogo with Python≫. In: *Journal of Artificial Societies and Social Simulation* 21.2 (2018), p. 4. ISSN: 1460-7425.
DOI: 10.18564/jasss.3668.
URL: http://jasss.soc.surrey.ac.uk/21/2/4.html.

[25] Matteo Starri. *Digital 2023*. Rapp. tecn.
Meltwater, We are social, 2023.

[26] Jan-Philipp Fränken e Toby Pilditch.
≪Cascades Across Networks Are Sufficient for the Formation of Echo Chambers: An Agent-Based Model≫. In: *Journal of Artificial Societies and Social Simulation* 24.3 (2021), p. 1. ISSN: 1460-7425.
DOI: 10.18564/jasss.4566.
URL: http://jasss.soc.surrey.ac.uk/24/3/1.html.

[27] Daniel Hawthorne-Madell e Noah D Goodman.
≪Reasoning about social sources to learn from actions and outcomes.≫
In: *Decision* 6.1 (2019), p. 17.

[28] Stephan Lewandowsky et al. ≪Misinformation and its correction: Continued influence and successful debiasing≫. In: *Psychological science in the public interest* 13.3 (2012), pp. 106–131.

[29] William Rand et al.
≪An Agent-Based Model of Urgent Diffusion in Social Media≫. In: *Journal of Artificial Societies and Social Simulation* 18.2 (2015), p. 1. ISSN: 1460-7425. DOI: 10.18564/jasss.2616.
URL: http://jasss.soc.surrey.ac.uk/18/2/1.html.

[30] Yimin Chen, Niall J Conroy e Victoria L Rubin.
≪Misleading online content: recognizing clickbait as" false news"≫.
In: *Proceedings of the 2015 ACM on workshop on multimodal deception detection*. 2015, pp. 15–19.

[31] Jens Koed Madsen, Richard M Bailey e Toby D Pilditch.
≪Large networks of rational agents form persistent echo chambers≫.
In: *Scientific reports* 8.1 (2018), pp. 1–8.

[32] Walter Quattrociocchi, Guido Caldarelli e Antonio Scala.
≪Opinion dynamics on interacting networks: media competition and social influence≫. In: *Scientific reports* 4.1 (2014), pp. 1–7.

[33] Tanveer Khan, Antonis Michalas e Adnan Akhunzada.
≪Fake news outbreak 2021: Can we stop the viral spread?≫ In: *Journal of Network and Computer Applications* 190 (2021), p. 103112. ISSN: 1084-8045.
DOI: https://doi.org/10.1016/j.jnca.2021.103112.
URL: https://www.sciencedirect.com/science/article/pii/S1084804521001326.

[34] Soroush Vosoughi, Deb Roy e Sinan Aral.
≪The spread of true and false news online≫.
In: *Science* 359.6380 (2018), pp. 1146–1151.
DOI: 10.1126/science.aap9559. URL:
https://www.science.org/doi/abs/10.1126/science.aap9559.

[35] Antonio F. Peralta, János Kertész e Gerardo Iñiguez.
Opinion dynamics in social networks: From models to data. 2022.
arXiv: 2201.01322 [physics.soc-ph].

[36] Claudio Castellano, Santo Fortunato e Vittorio Loreto.
≪Statistical physics of social dynamics≫.
In: *Rev. Mod. Phys.* 81 (2 mag. 2009), pp. 591–646.
DOI: 10.1103/RevModPhys.81.591.
URL: https://link.aps.org/doi/10.1103/RevModPhys.81.591.

[37] Michele Carillo et al.
≪Sociality, Sanctions, Damaging Behaviors: A Distributed
Implementation of an Agent-Based Social Simulation Model≫.
In: *Euro-Par 2013: Parallel Processing Workshops*.
A cura di Dieter an Mey et al.
Berlin, Heidelberg: Springer Berlin Heidelberg, 2014, pp. 595–604.
ISBN: 978-3-642-54420-0.

[38] Flaminio Squazzoni et al. ≪Computational models that matter during a global pandemic outbreak: A call to action≫. In: (2020).

[39] Volker Grimm et al. ≪The ODD Protocol for Describing Agent-Based and Other Simulation Models: A Second Update to Improve Clarity, Replication, and Structural Realism≫. In: *Journal of Artificial Societies and Social Simulation* 23.2 (2020), p. 7. ISSN: 1460-7425.
DOI: 10.18564/jasss.4259.
URL: http://jasss.soc.surrey.ac.uk/23/2/7.html.

[40] Sandersan Onie et al. ≪Investigating the Effects of Inhibition Training on Attentional Bias Change: A Simple Bayesian Approach≫.
In: *Frontiers in Psychology* 9 (2019). ISSN: 1664-1078.
DOI: 10.3389/fpsyg.2018.02782. URL: https:
//www.frontiersin.org/articles/10.3389/fpsyg.2018.02782.

[41] R. Kelly Garrett. ≪Echo chambers online?: Politically motivated selective exposure among Internet news users1≫. In: *Journal of Computer-Mediated Communication* 14.2 (gen. 2009), pp. 265–285.
ISSN: 1083-6101. DOI: 10.1111/j.1083-6101.2009.01440.x.
eprint: https://academic.oup.com/jcmc/article-

pdf/14/2/265/21491614/jjcmcom0265.pdf.
URL: https://doi.org/10.1111/j.1083-6101.2009.01440.x.

[42] Andrei Boutyline e Robb Willer.
≪The Social Structure of Political Echo Chambers: Variation in Ideological Homophily in Online Networks≫.
In: *Political Psychology* 38.3 (2017), pp. 551–569.
DOI: https://doi.org/10.1111/pops.12337. URL: https://onlinelibrary.wiley.com/doi/abs/10.1111/pops.12337.

[43] David O. Sears e Jonathan L. Freedman.
≪Selective Exposure to information: a critical review*≫.
In: *Public Opinion Quarterly* 31.2 (gen. 1967), pp. 194–213.
ISSN: 0033-362X. DOI: 10.1086/267513.
eprint: https://academic.oup.com/poq/article-pdf/31/2/194/5132577/31-2-194.pdf.
URL: https://doi.org/10.1086/267513.

[44] Damon Centola.
≪The Spread of Behavior in an Online Social Network Experiment≫.
In: *Science* 329.5996 (2010), pp. 1194–1197.
DOI: 10.1126/science.1185231. URL: https://www.science.org/doi/abs/10.1126/science.1185231.

[45] Michela Del Vicario et al.
≪Modeling confirmation bias and polarization≫.
In: *Scientific reports* 7.1 (2017), p. 40391.

[46] Xinyi Zhou e Reza Zafarani. ≪A Survey of Fake News: Fundamental Theories, Detection Methods, and Opportunities≫.
In: *ACM Comput. Surv.* 53.5 (set. 2020). ISSN: 0360-0300.
DOI: 10.1145/3395046. URL: https://doi.org/10.1145/3395046.

[47] Nigel Gilbert.
≪Agent-based social simulation: dealing with complexity≫.
In: *The Complex Systems Network of Excellence* 9.25 (2004), pp. 1–14.

[48] Yuxi Li. *Deep Reinforcement Learning: An Overview*. 2018.
arXiv: 1701.07274 [cs.LG].

[49] Greg Brockman et al. *OpenAI Gym*. 2016.
arXiv: 1606.01540 [cs.LG].

[50] Seth Tisue e Uri Wilensky.
≪Netlogo: A simple environment for modeling complexity≫.
In: *International conference on complex systems*. Vol. 21.
Citeseer. 2004, pp. 16–21.

[51] V. Grimm et al. ≪The ODD protocol for describing agent-based and other simulation models: A second update to improve clarity, replication, and structural realism≫. In: *Journal of Artificial Societies and Social Simulation* 23.2 (gen. 2020).
URL: http://eprints.bournemouth.ac.uk/33918/.

[52] Damian Tambini. ≪Fake news: public policy responses≫. In: (2017).

[53] Victoria L. Rubin et al. ≪A News Verification Browser for the Detection of Clickbait, Satire, and Falsified News≫.
In: *Journal of Open Source Software* 4.35 (2019), p. 1208.
DOI: 10.21105/joss.01208.
URL: https://doi.org/10.21105/joss.01208.

[54] Benjamin D. Horne e Sibel Adali. ≪This Just In: Fake News Packs a Lot in Title, Uses Simpler, Repetitive Content in Text Body, More Similar to Satire than Real News≫. In: *CoRR* abs/1703.09398 (2017).
arXiv: 1703.09398. URL: http://arxiv.org/abs/1703.09398.

[55] W. Phillips Davison. ≪The Third-Person Effect in Communication≫.
In: *Public Opinion Quarterly* 47.1 (gen. 1983), pp. 1–15.
ISSN: 0033-362X. DOI: 10.1086/268763.
eprint: https://academic.oup.com/poq/article-pdf/47/1/1/5382397/47-1-1.pdf.
URL: https://doi.org/10.1086/268763.

[56] Jonathan L. Freedman e David O. Sears.
≪Selective Exposure11The preparation of this paper was supported in part by NSF grants to the authors.≫ In: a cura di Leonard Berkowitz.
Vol. 2. Advances in Experimental Social Psychology.
Academic Press, 1965, pp. 57–97.
DOI: https://doi.org/10.1016/S0065-2601(08)60103-3.
URL: https://www.sciencedirect.com/science/article/pii/S0065260108601033.

[57] P. C. Wason.
≪On the Failure to Eliminate Hypotheses in a Conceptual Task≫.
In: *Quarterly Journal of Experimental Psychology* 12.3 (1960), pp. 129–140. DOI: 10.1080/17470216008416717.
URL: https://doi.org/10.1080/17470216008416717.

Appendice

A. Caratteristiche dell'ambiente di sviluppo

In questo lavoro di tesi è stata utilizzata una macchina equipaggiata come riportato in Tabella 6.1.

Componente	**Caratteristiche**
Processore	`Intel® Xeon(R) Gold 5218 CPU @ 2.30GHz × 32 core`
RAM	`256 GB (251,4 GB utilizzabile)`
Tipo di sistema	SO a 64 bit, processore basato su `x64`
Scheda Video	`NVIDIA Quadro RTX 4000`
Nome SO	`Ubuntu Server 21.10`
Modello Sistema	`ProLiant ML350 Gen10`
Archiviazione	`HPE NVMe 2048GB`

Tabella 6.1: Configurazione della macchina utilizzata per gli esperimenti

Di seguito viene mostrata l'esecuzione di una simulazione su *NetLogo*.

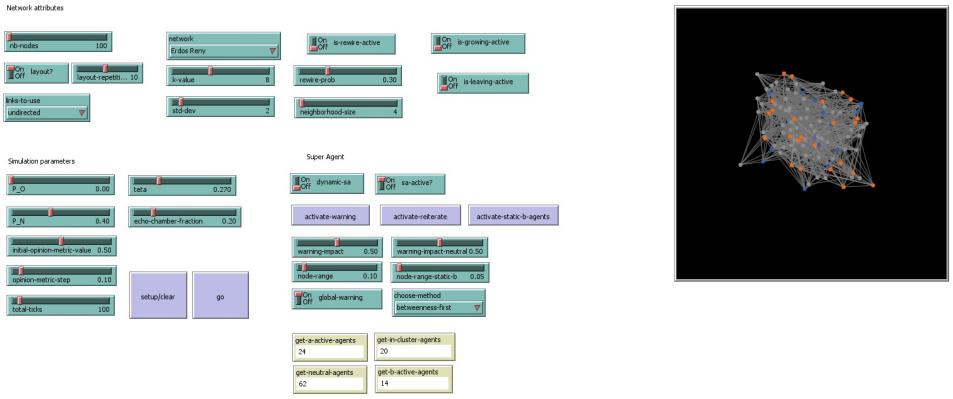


Figura 6.1: Simulazione di una rete sociale con diffusione di fake news su NetLogo