



METODI E TECNICHE DI SIMULAZIONE

nxn imitate if better networks

PRESENTATA DA:

Nicol Alesi





Imitate if better

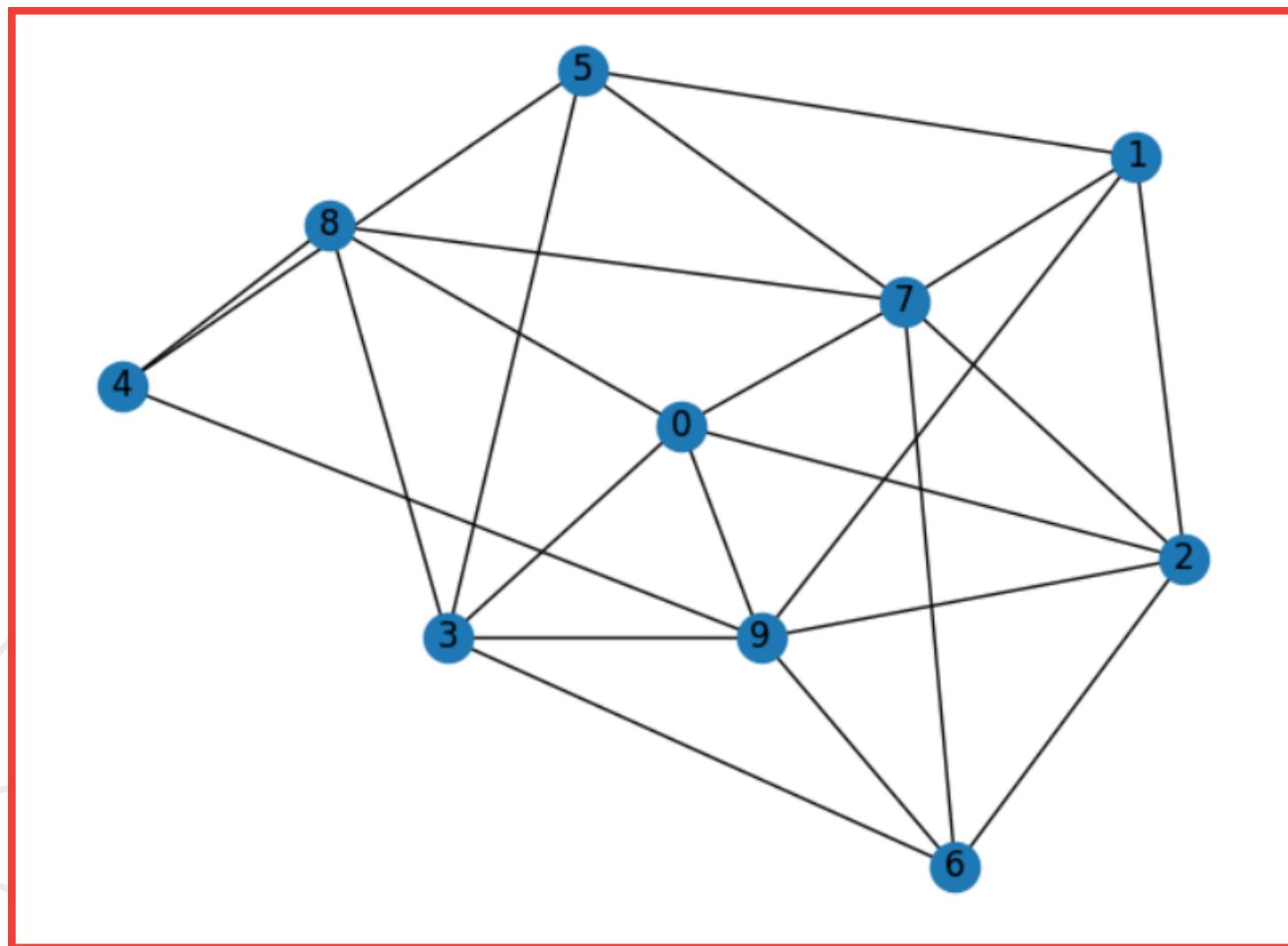
L'agente sceglie di cambiare strategia se l'agente che ha accanto ha un payoff migliore;

Collegando i giocatori in una rete Erdős-Rényi, questi tendono a scegliere la strategia non ottimale;

Verifichiamo cosa accade con altri tipi di network.

Erdős-Rényi

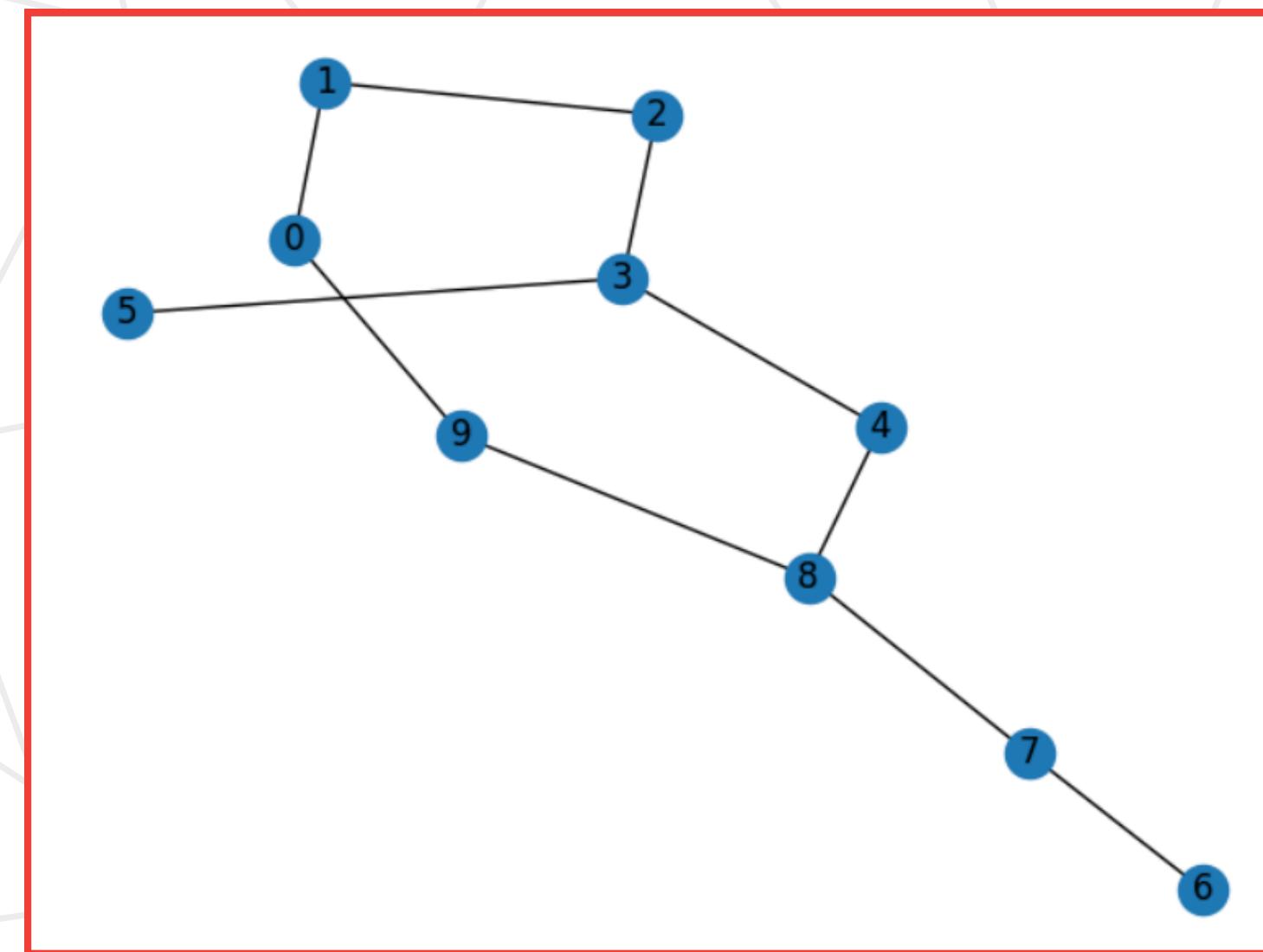
Il modello di Erdős-Rényi crea una rete collegando i nodi a caso, senza seguire nessuna regola precisa. Ogni coppia di nodi ha la stessa probabilità di essere collegata, quindi la rete è molto uniforme. Questo tipo di rete è semplice, ma non rappresenta bene le reti reali come quelle sociali o biologiche. Nel nostro codice ha il parametro "prob_link".



Watts Strogatz

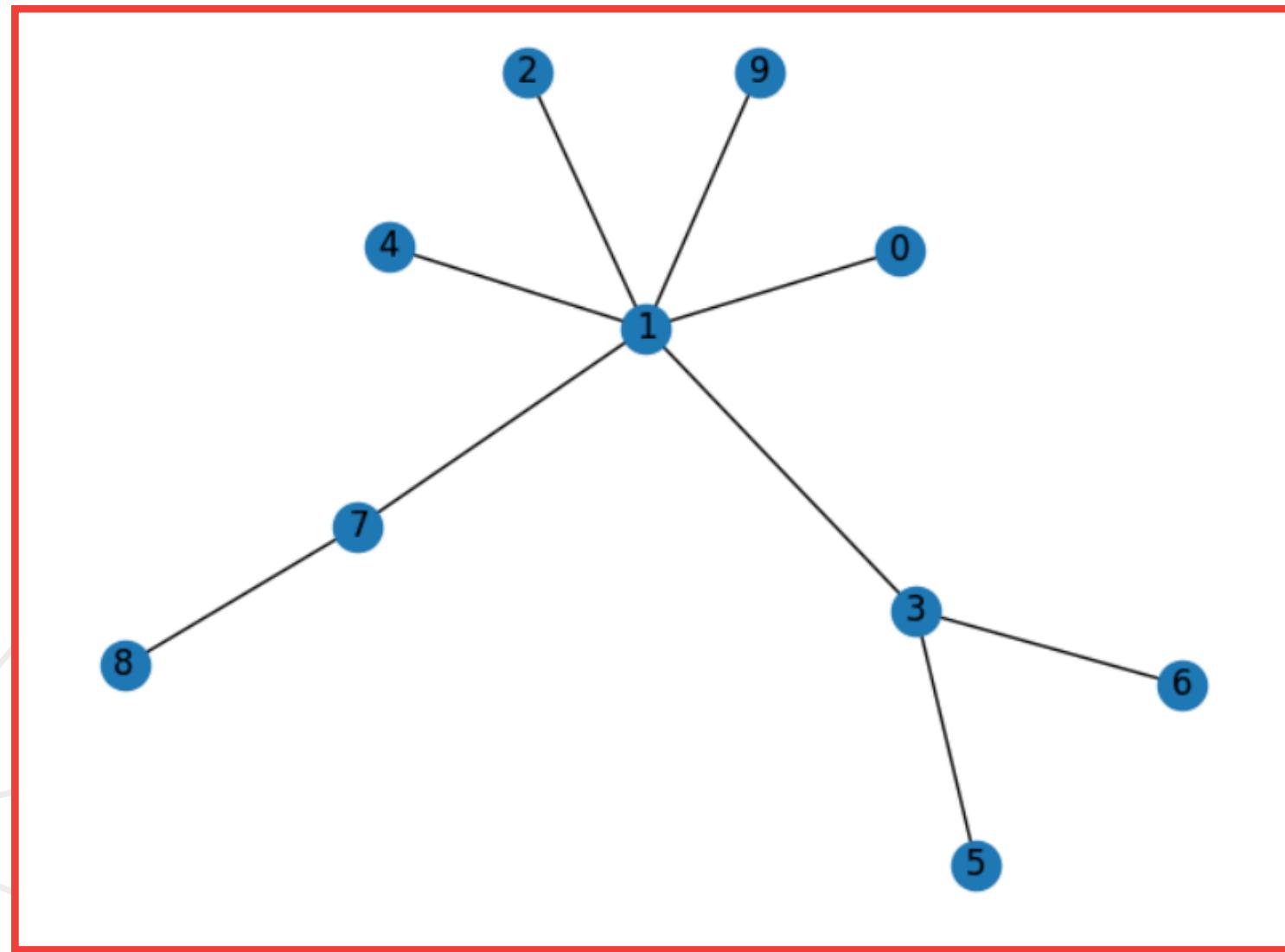
La rete Watts-Strogatz è un modello di rete "small-world" in cui i nodi sono inizialmente connessi ai vicini più prossimi (come in un reticolo), ma alcuni collegamenti vengono riassegnati in modo casuale per introdurre scorciatoie, creando una struttura che combina alta coesione locale e brevi distanze globali tra i nodi.

Nel nostro codice ha due parametri, il grado medio e la probabilità di rewiring.



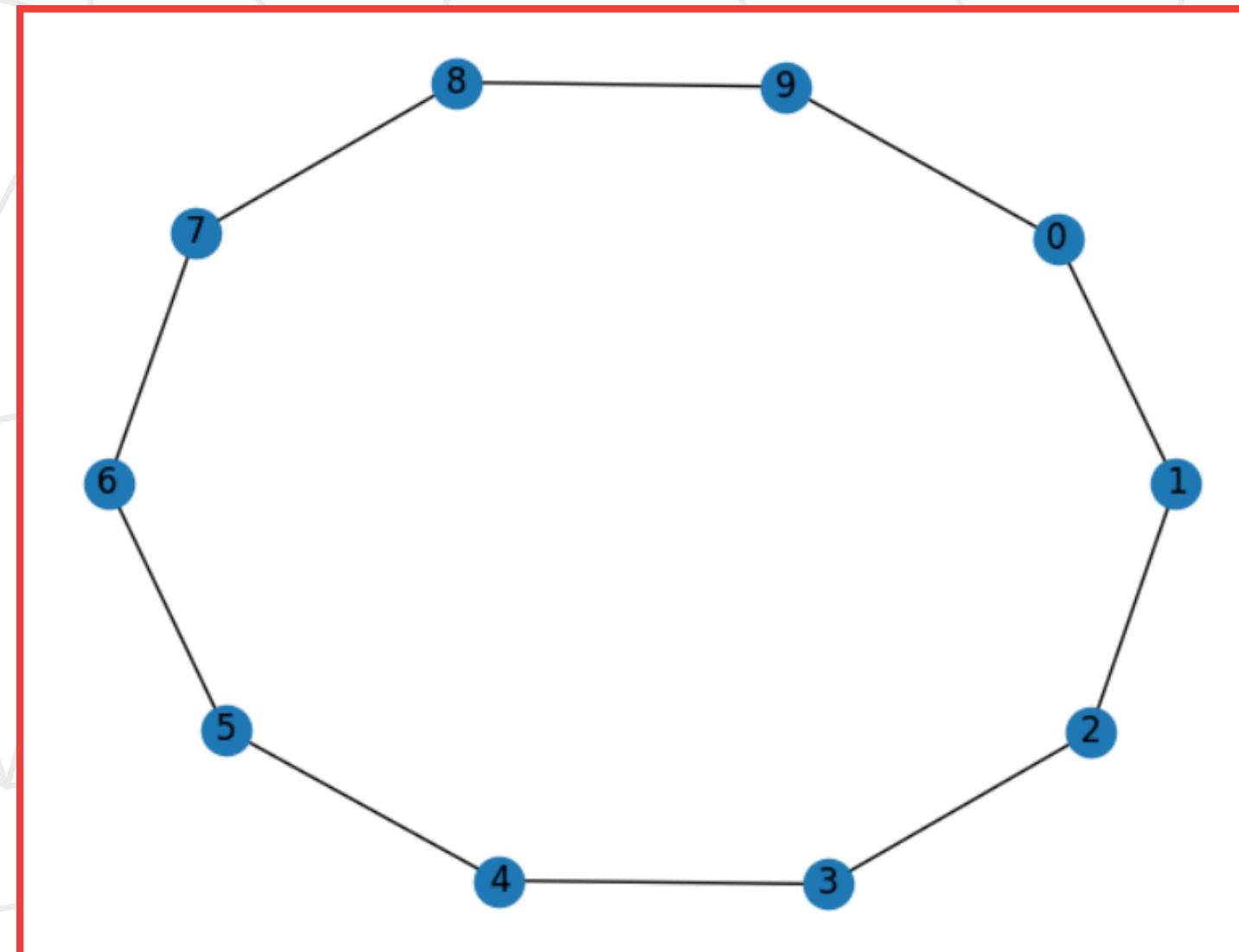
Preferential attachment

Il modello a preferential attachment (Barabási-Albert) costruisce la rete aggiungendo un nodo alla volta, che si collega più facilmente a nodi già molto connessi. In questo modo, i nodi più popolari diventano sempre più popolari. Il risultato è una rete con pochi nodi molto collegati (hub) e tanti nodi con pochi collegamenti, come succede nel web o nei social. Accetta il parametro grado minimo dei nodi.



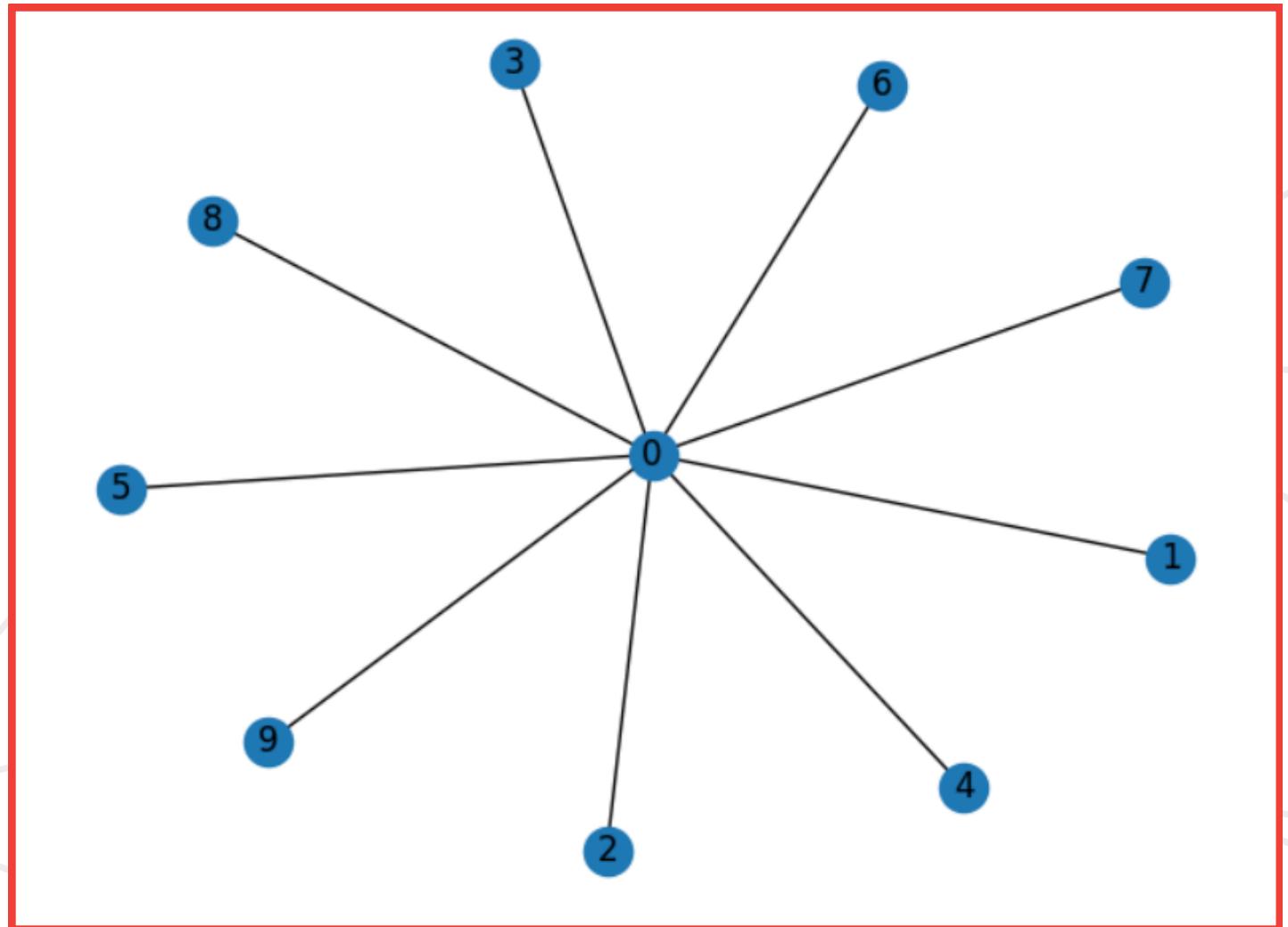
Ring network

Una ring network è una rete in cui ogni nodo è collegato solo ai suoi vicini diretti, formando un anello chiuso. Tutti i nodi hanno lo stesso numero di connessioni e la rete è molto ordinata. Però, per raggiungere un nodo lontano, bisogna passare attraverso molti altri, quindi le distanze sono lunghe.



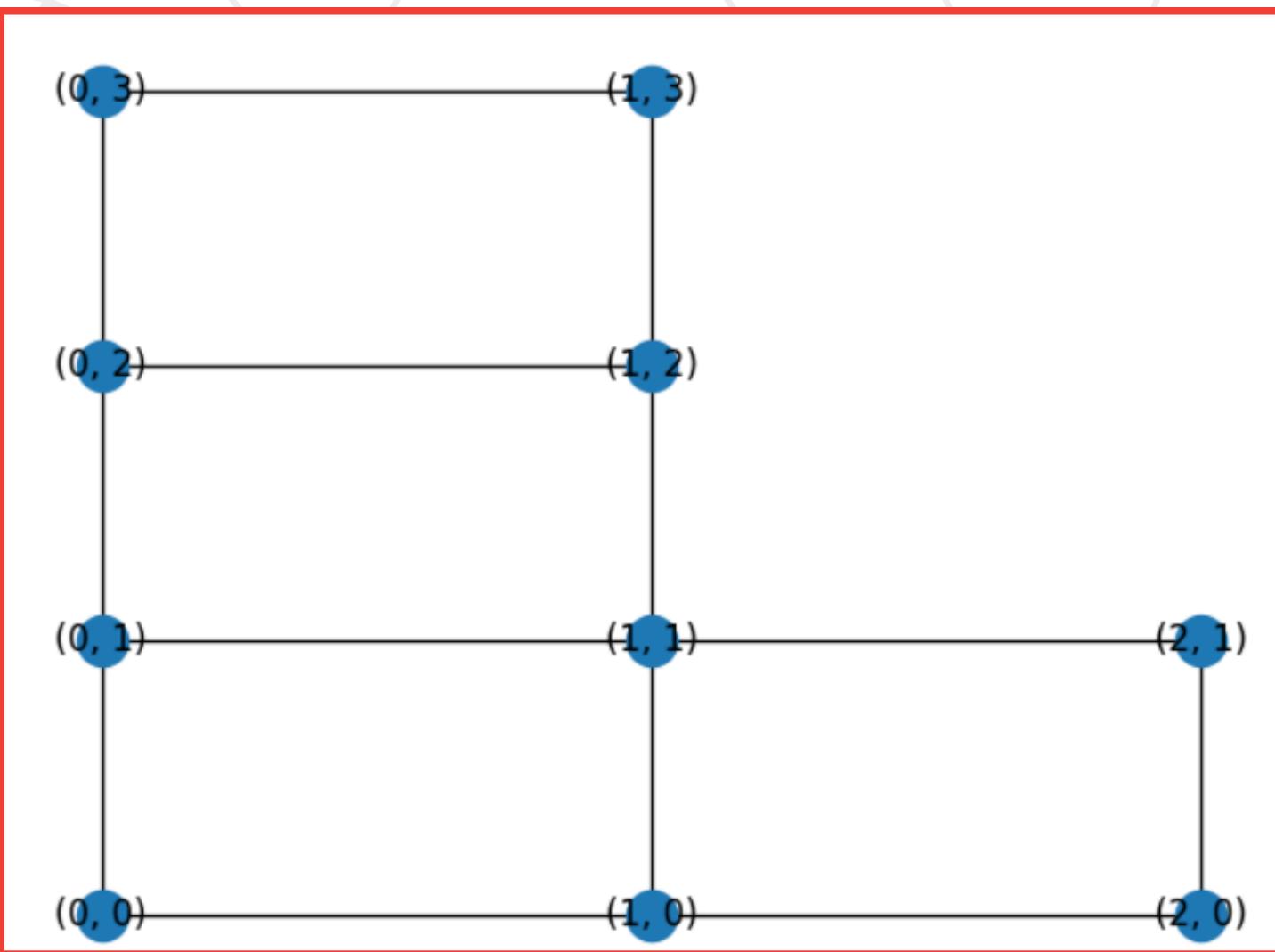
Star network

Una star network ha un nodo centrale collegato direttamente a tutti gli altri, mentre i nodi periferici non si collegano tra loro. Questo rende facile comunicare con il centro in un solo passaggio, ma se il nodo centrale si rompe, l'intera rete si blocca. È una struttura semplice ma fragile.



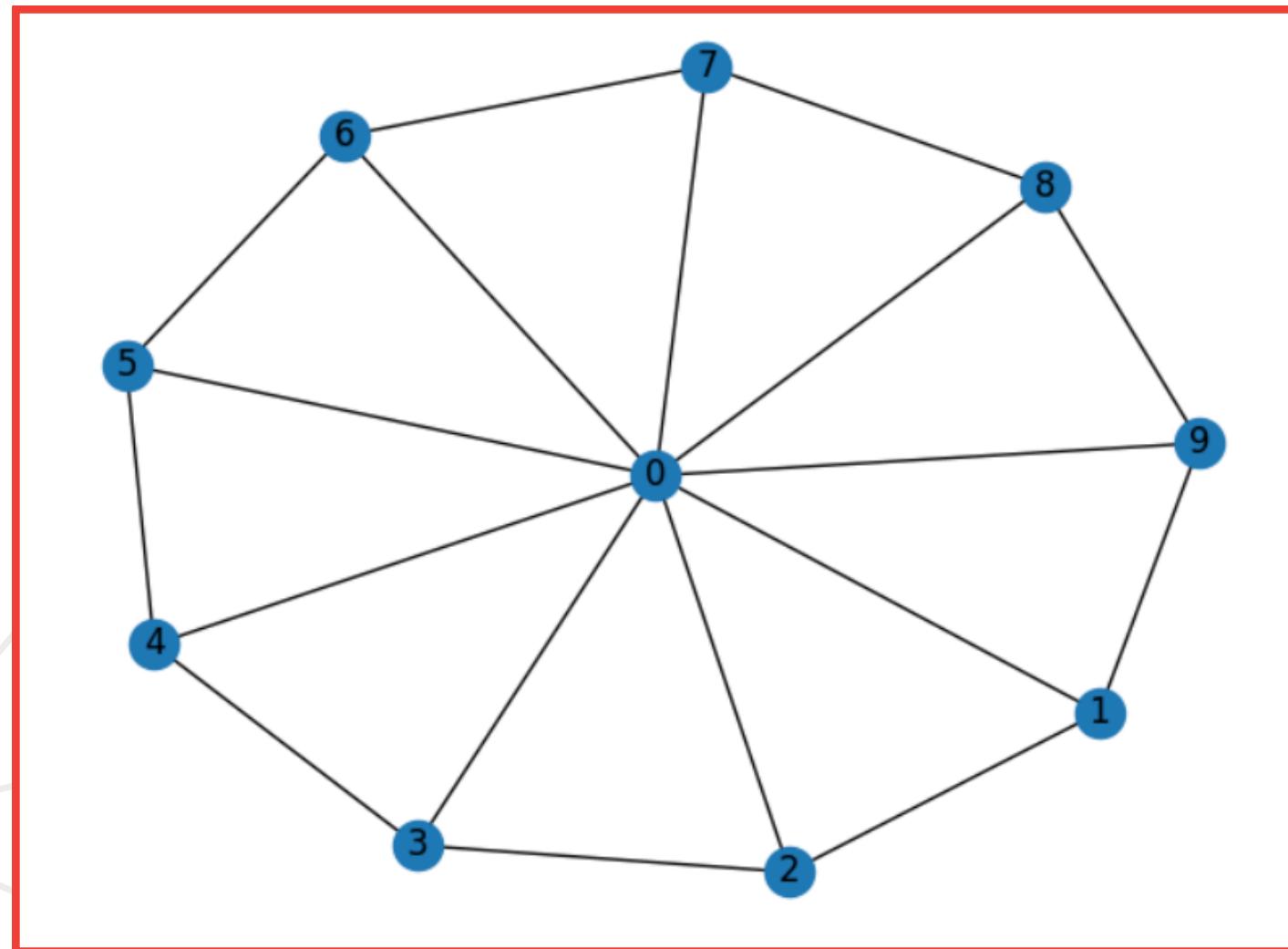
Grid 4 neighbors

Una grid a 4 vicini è una rete dove i nodi sono disposti in una griglia, e ogni nodo è collegato ai suoi 4 vicini immediati: sopra, sotto, a sinistra e a destra. Non ci sono connessioni diagonali. Questo tipo di rete è ordinata e permette di navigare facilmente tra i nodi, ma le connessioni sono limitate rispetto a reti più complesse.



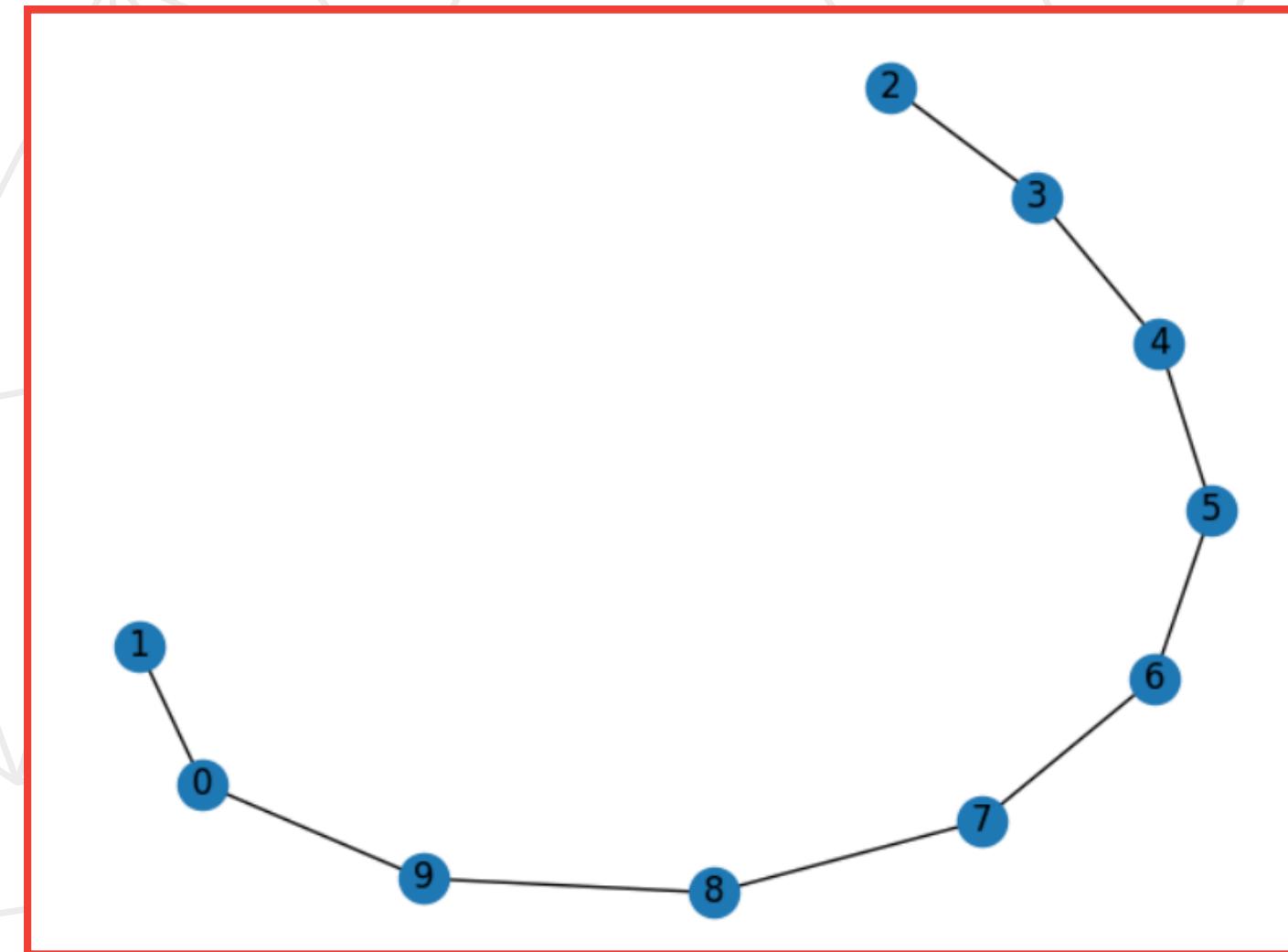
Wheel network

Una wheel network è simile a una star network, ma con una differenza: oltre al nodo centrale che è collegato a tutti gli altri, anche i nodi esterni sono collegati tra loro, formando un anello. In questo modo, ogni nodo esterno ha due connessioni, una al nodo centrale e una agli altri nodi della periferia. Questo aumenta la connettività della rete rispetto a una semplice star.



Path network

Una path network è una rete in cui i nodi sono disposti in una sequenza lineare, dove ogni nodo è collegato solo al nodo precedente e al nodo successivo. Non ci sono connessioni tra nodi più distanti. In questa rete, per spostarsi da un nodo all'altro, bisogna attraversare tutti i nodi intermedi. È una rete molto semplice e con una struttura chiara.



Model

```
''' Classe del modello, prende in input molti parametri per poter essere compatibile con la gestione dei dati
class nxnImitationModel(mesa.Model):
    """A model with some number of agents."""
    def __init__(self, n, seed=None, network_type="wheel_network", prob_revision=0.1, payoffs=payoffs,
                 initial_probs=initial_probs, edges=edges, avg_degree=avg_degree, prob_rewire=prob_rewire,
                 noise=noise, prob_link=prob_link):
        super().__init__(seed=seed)
        self.num_agents = n
        self.running = True
        self.prob_revision = prob_revision
        self.initial_probs = initial_probs
        self.payoffs=payoffs
        self.edges=edges
        self.avg_degree=avg_degree
        self.prob_rewire=prob_rewire
        self.noise=noise
        self.prob_link=prob_link
        self.grid=Network(n, network_type,edges,avg_degree,prob_rewire,prob_link)

        assert len(self.grid.G.nodes) == self.num_agents, f"Mismatch: {len(self.grid.G.nodes)} != {self.num_agents}!"

    # Metodo per la creazione degli agenti
    agents = nxnImitationAgent.create_agents(model=self, n=n)

    # Posizionamento degli agenti nella rete
    for node, agent in zip(self.grid.G.nodes, agents):
        print(f"Posizionamento agente {agent} nel nodo {node} con strategia {agent.strategy}")
        self.grid.place_agent(agent, node)

    # Dati da monitorare
    self.datacollector = mesa.DataCollector(
        agent_reporters={
            "Strategy": "strategy",
            "Payoff": "payoff",
            "Wealth": "wealth",
        }
    )
    # Step del modello, viene chiamato ad ogni iterazione della simulazione
    def step(self):
        self.datacollector.collect(self)
        self.agents.shuffle_do("update_payoff")
        self.agents.shuffle_do("update_strategy")
```

Agent

```
class nxnImitationAgent(mesa.Agent):
    def __init__(self, model):
        super().__init__(model)
        self.wealth = 1
        self.steps_not_given = 0
        self.strategy_after_revision = None
        self.payoffs=model.payoffs
        self.wealth = 1
        self.initial_probs = model.initial_probs

        strategies=list(self.payoffs.keys())

        if self.initial_probs is None:
            self.initial_probs = [1/len(strategies)] * len(strategies)

        self.strategy = self.random.choices(strategies,weights=self.initial_probs,k=1)[0]

    def update_payoff(self):
        neighbors = self.model.grid.get_neighbors(self.pos, include_center=False)
        if len(neighbors) > 0:
            for other in neighbors:
                # Calcola i payoff in base alle strategie correnti
                payoff_self, payoff_other = self.payoffs[self.strategy][other.strategy]

                # Aggiungi il payoff a entrambi gli agenti
                self.wealth = payoff_self
                other.wealth = payoff_other

    def update_strategy(self):
        if self.random.random() < self.model.prob_revision: # L'agente può aggiornare la strategia
            strategies = list(self.payoffs.keys())

            # 1. Con probabilità "noise", cambia a una strategia casuale
            if self.random.random() < self.model.noise:
                self.strategy = self.random.choice(strategies)

            # 2. Altrimenti, imita un vicino migliore
            else:
                neighbors = self.model.grid.get_neighbors(self.pos, include_center=False)
                if len(neighbors) > 0:
                    other = self.random.choice(neighbors)
                    if other.wealth > self.wealth:
                        self.strategy = other.strategy
```

Network App

```
, Classe che si occupa della costruzione della rete, in base al tipo di modello scelto
class Network(NetworkGrid):

    def __init__(self, num_nodes, network_model_type, edges, avg_degree, prob_rewire, prob_link):
        self.num_nodes = num_nodes
        self.network_model_type = network_model_type
        self.edges=edges
        self.avg_degree=avg_degree
        self.prob_rewire=prob_rewire
        self.prob_link=prob_link

        super().__init__(self.build_network()) # Passa la rete costruita al NetworkGrid

    def build_network(self):
        """Costruisce una rete in base al tipo di modello"""
        # Usa un dizionario per mappare la rete alla funzione
        model_functions = {
            "erdos_renyi": self.erdos_renyi,
            "watts_strogatz": self.watts_strogatz,
            "preferential_attachment": self.preferential_attachment,
            "ring_network": self.ring_network,
            "star_network": self.star_network,
            "grid_4_neighbors": self.grid_4_neighbors,
            "wheel_network": self.wheel_network,
            "path_network": self.path_network
        }

        # Richiamiamo il metodo per la creazione del grafo scelto
        if self.network_model_type in model_functions:
            graph=model_functions[self.network_model_type](self.num_nodes)
            self.pos=nx.spring_layout(graph)
            return graph
        else:
            raise ValueError("Unknown network model type: {}".format(self.network_model_type))

    # Da qui in poi sono definiti i vari metodi per la creazione dei grafi, ognuno con le proprie
    def erdos_renyi(self, num_nodes):
        """Crea un grafo Erdős-Rényi"""
        graph = nx.erdos_renyi_graph(n=num_nodes, p=self.prob_link)
```

```
, a qui in poi è definita la GUI, che permette di interagire con il modello e di visualizzarne
class App:

    def __init__(self, root):
        self.root = root
        self.root.title("Simulazione Strategica")
        self.strategy_colors = None # Colori assegnati alle strategie

        self.num_agents = tk.IntVar(value=100)
        self.network_type = tk.StringVar(value="erdos_renyi")
        self.steps = tk.IntVar(value=50)
        self.prob_revision = tk.DoubleVar(value=0.1)
        self.payoffs_input = tk.StringVar(value=str(payoffs_default))
        self.initial_distribution = tk.StringVar(value="None")
        self.edges = tk.IntVar(value=1)
        self.noise = tk.DoubleVar(value=0.1)
        self.avg_degree = tk.IntVar(value=4)
        self.prob_rewiring = tk.DoubleVar(value=0.3)
        self.prob_link = tk.DoubleVar(value=0.5)
        self.show_animation = tk.BooleanVar(value=True) # Default: mostra animazione

        self.frame_canvas = ttk.Frame(self.root)
        self.frame_canvas.pack(side=tk.RIGHT, fill=tk.BOTH, expand=True)

    # Canvas dei grafici
    self.figure = plt.Figure(figsize=(8, 4), dpi=100)
    self.canvas = FigureCanvasTkAgg(self.figure, master=self.frame_canvas)
    self.canvas.get_tk_widget().pack(side=tk.TOP, fill=tk.BOTH, expand=True)

    # Canvas della rete
    self.network_figure = plt.Figure(figsize=(8, 4), dpi=100)
    self.network_canvas = FigureCanvasTkAgg(self.network_figure, master=self.frame_canvas)
    self.network_canvas.get_tk_widget().pack(side=tk.TOP, fill=tk.BOTH, expand=True)

    self.setup_widgets()
```

Differenze con precedenti codici

Classe Modello

- Creazione del network
- Verifica che la lunghezza del network e il numero degli agenti coincidano
- Collocazione degli agenti nel network

Classe Agente

- Generalizzazione della matrice
- Individuazione dei vicini in base al network creato
- Wealth come variabile di supporto

Classe Network

- Possibilità di creare qualsiasi network, non uno solo
- Funzioni per la creazione di ogni network con i propri specifici parametri, possibilità di plot esplicativo

Classe App

- Totalmente nuova rispetto ai precedenti
- Necessità che nel modello vengano passati parametri che non sarebbero normalmente strettamente necessari
- Plot del network
- Plot delle strategie
- Plot animati
- Plot statici
- Possibilità di input dinamici e continuativi, velocizzando i test

Interfaccia grafica

Numero Agenti
50

Tipo di Rete
star_network

Passi di simulazione
50

Probabilità di Revisione
0.02

Rumore
0.1

Distribuzione iniziale
[0.7,0.3]

Per Erdos
0.5

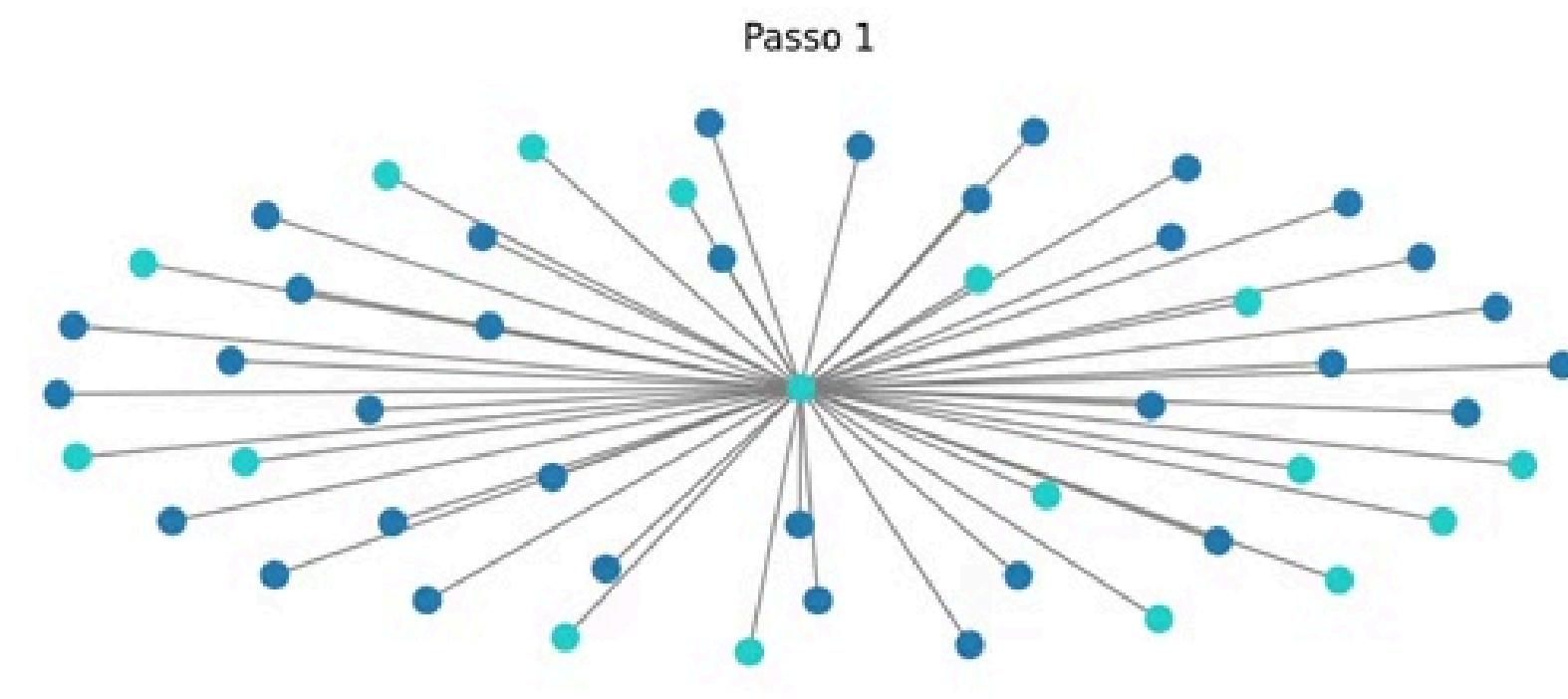
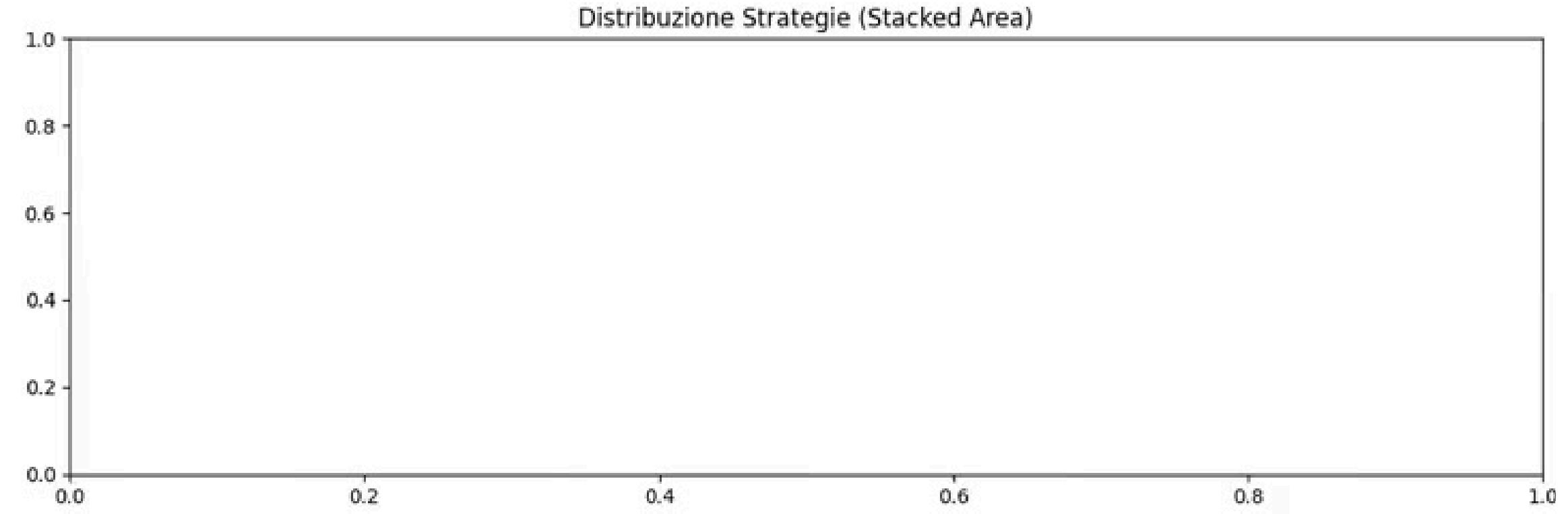
Per preferential attachment
Scegli il grado
1

Per small-world network
Grado medio
4

Probabilità rewiring
0.3

Matrice dei Payoff (formato dizionario)
{'A': {'A': (1, 1), 'B': (0, 0)}, 'B': {'A': (0, 0), 'B': (2, 2)}}

Mostra Animazioni



Numero Agenti
20

Tipo di Rete
ring_network ▾

Passi di simulazione
150

Probabilità di Revisione
0.1

Rumore
0.03

Distribuzione iniziale
[0.7,0.3]

Per Erdos
0.02

Per preferential attachment
Scegli il grado
1

Per small-world network
Grado medio
2

Probabilità rewiring
0.25

Matrice dei Payoff (formato dizionario)

```
A': {'A': (1, 1), 'B': (0, 0)}, 'B': {'A': (2, 2)}
```

Mostra Animazioni

Esegui Simulazione

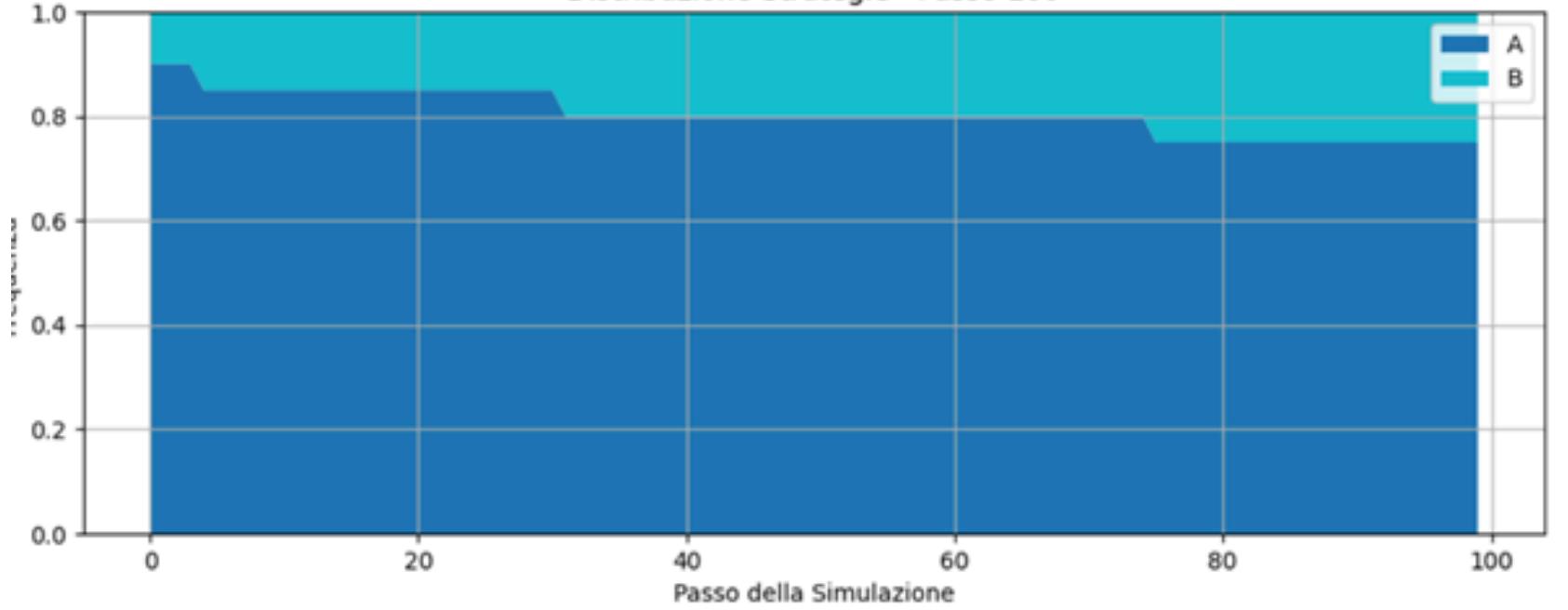
Sample runs

Eseguiamo delle prove seguendo i parametri richiesti nel nostro capitolo:

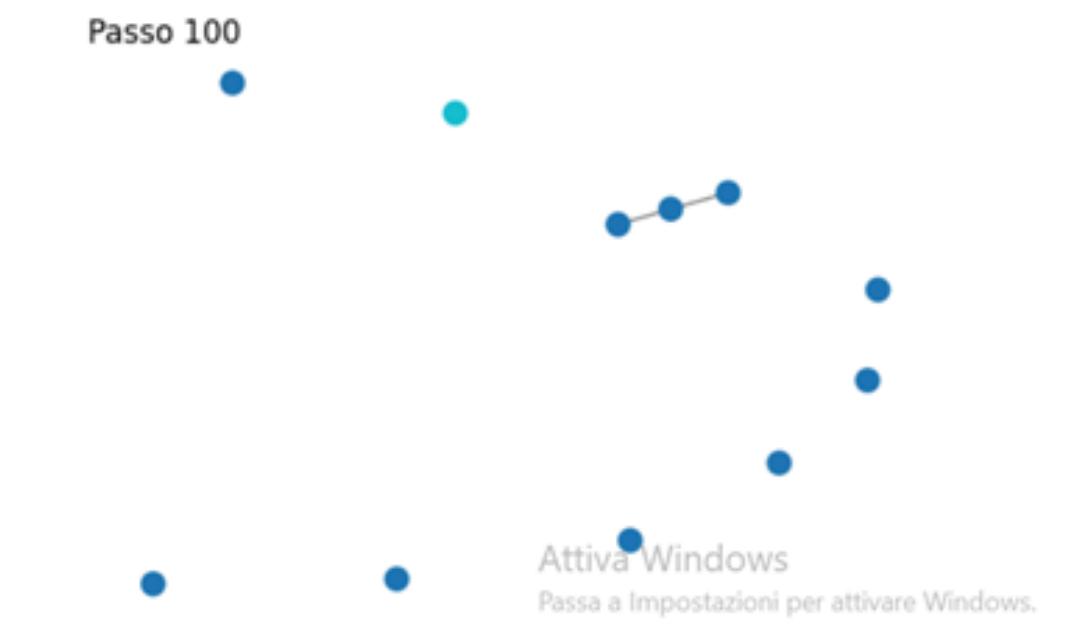
- Probabilità di revisione: 0.1
- Rumore: 0.03
- Distribuzione iniziale: [0.7,0.3]
- Tutte le reti
 - Erdos Renyi con prob_link= 0.02
 - Watts-Strogatz con grado medio =2 e probabilità di rewiring = {0,0.25,0.5,0.75,1}
 - Preferential-attachment con grado minimo = 1
 - Ring
 - Star
 - Path

		Player 2	
		Player 2 chooses A	Player 2 chooses B
		1, 1	0, 0
Player 1	Player 1 chooses A	1, 1	0, 0
	Player 1 chooses B	0, 0	2, 2

Distribuzione Strategie - Passo 100



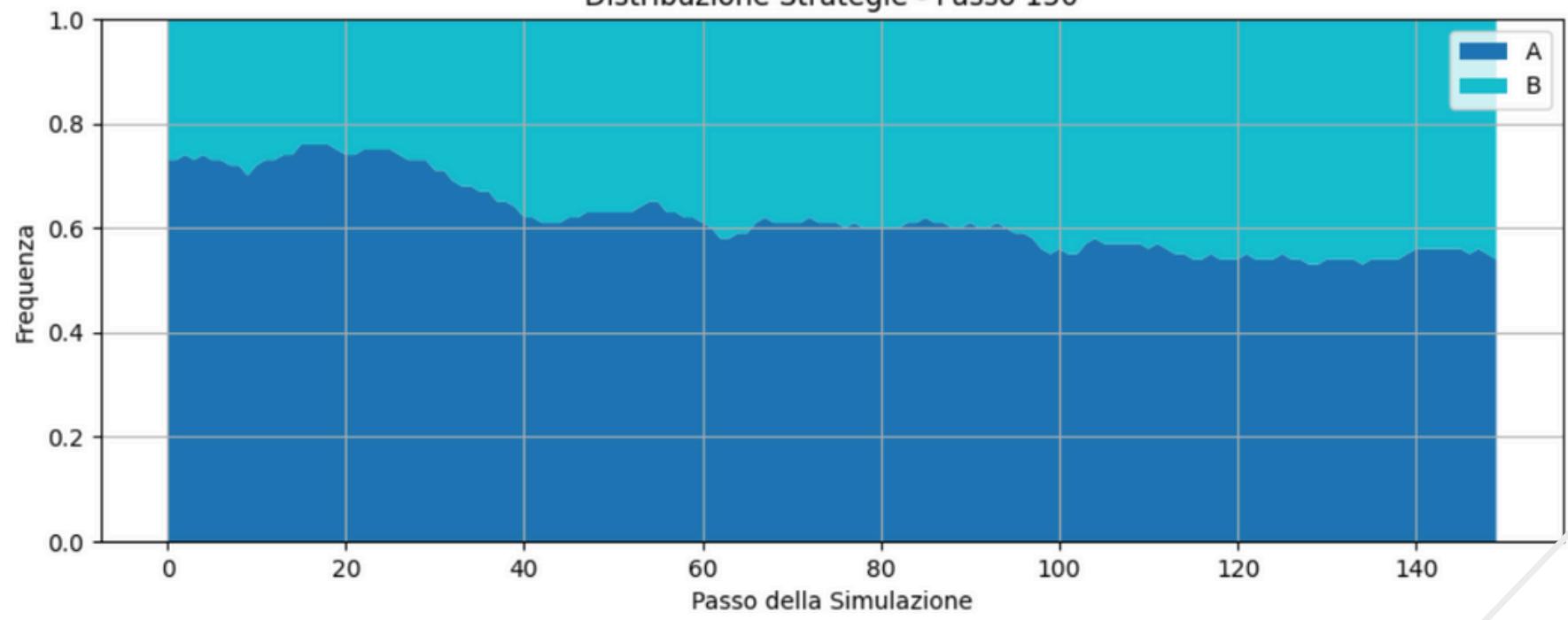
Passo 100



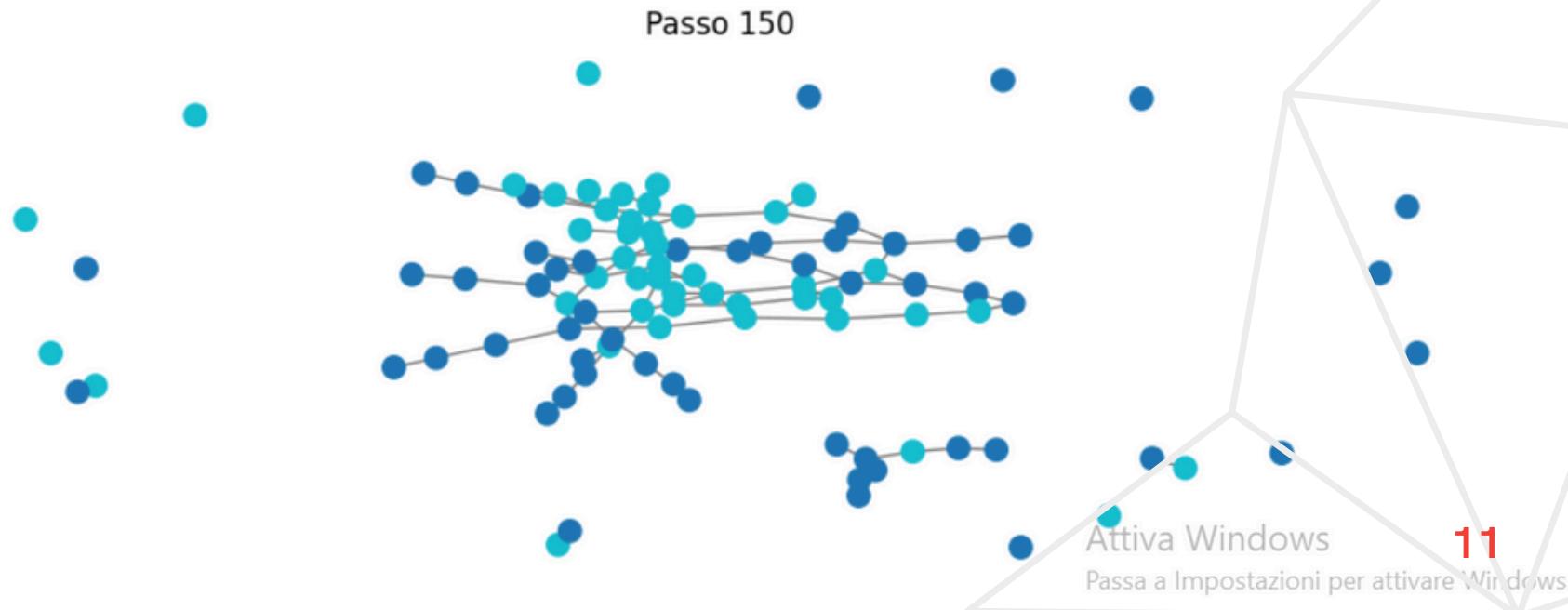
- Pochi agenti = bassa probabilità di connessione = mantenimento della strategia non ottimale
- Molti agenti = alta probabilità di connessione = maggiore diffusione della strategia ottimale
- Se non connessi tra loro l'unica ragione di cambiamento di strategia è l'irrazionalità, quindi il rumore
- Vale lo stesso se cambia la matrice dei payoff

Sample runs: Erdos Renyi

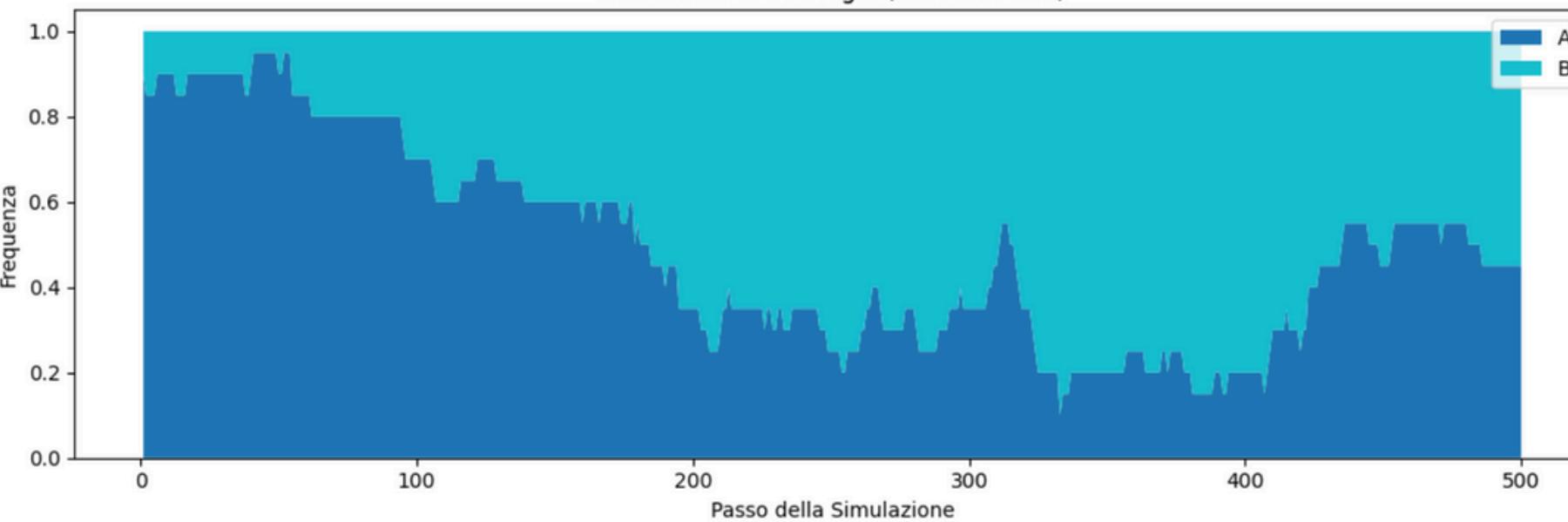
Distribuzione Strategie - Passo 150



Passo 150

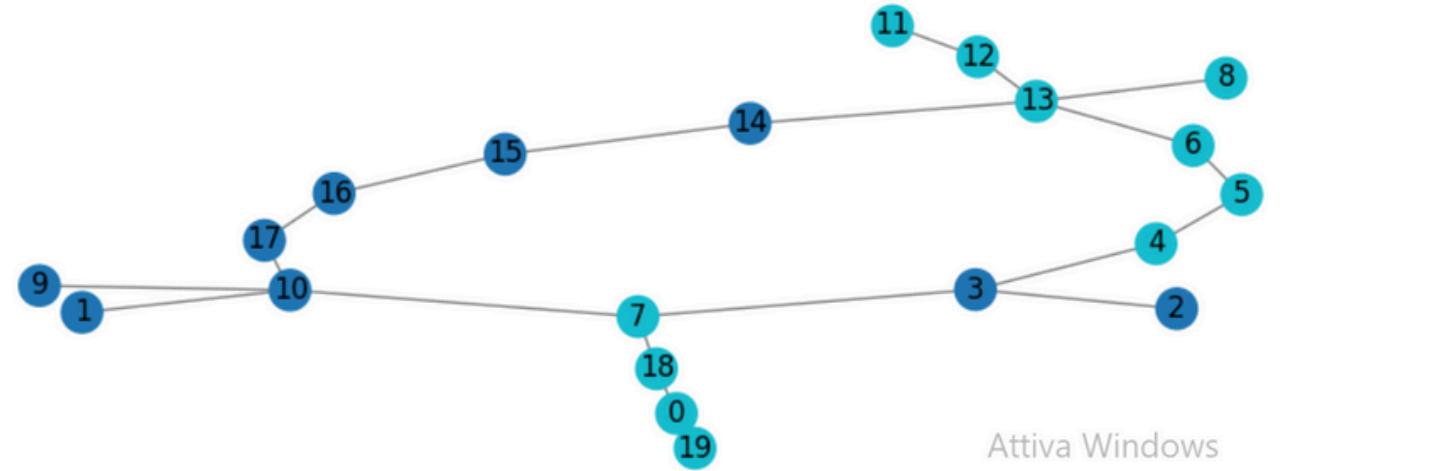


Distribuzione Strategie (Stacked Area)



Sample runs: Watts-strogatz

Rete degli Agenti



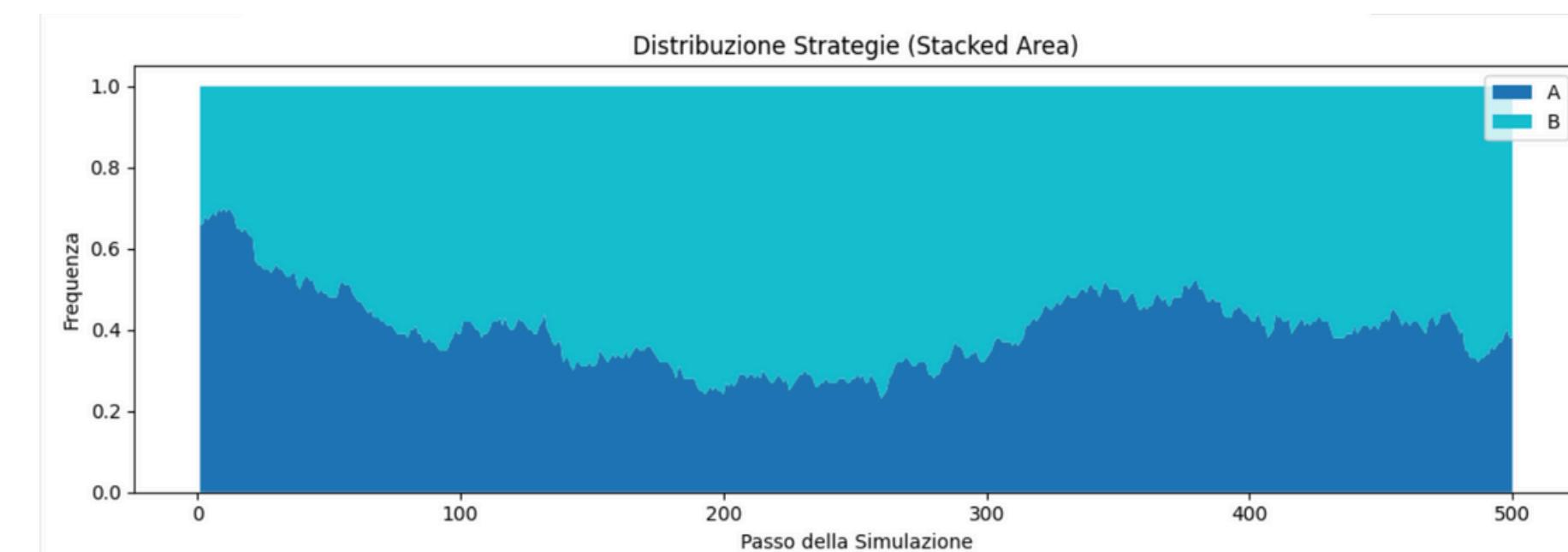
Sia con pochi che con molti agenti la due strategie a lungo termine tendono a stabilizzarsi, ciò accade anche nel caso in cui ce ne sia una terza.

Probabilmente perché nelle comunità molto vicine gli utenti tendono ad influenzarsi continuamente ed in fretta l'uno con l'altro.

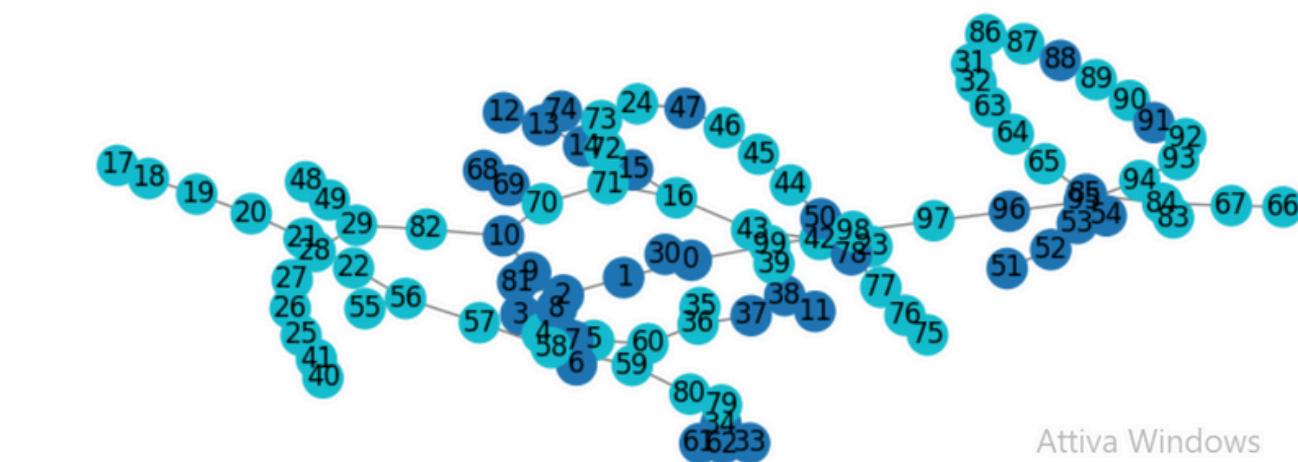
Cambiando il rewiring, se rimane compreso tra 0 e 1, si diffonderà leggermente di più la strategia ottimale e tendono agli estremi ad essere simili le strategie.

Aumentando il grado si diffonde maggiormente la strategia ottimale.

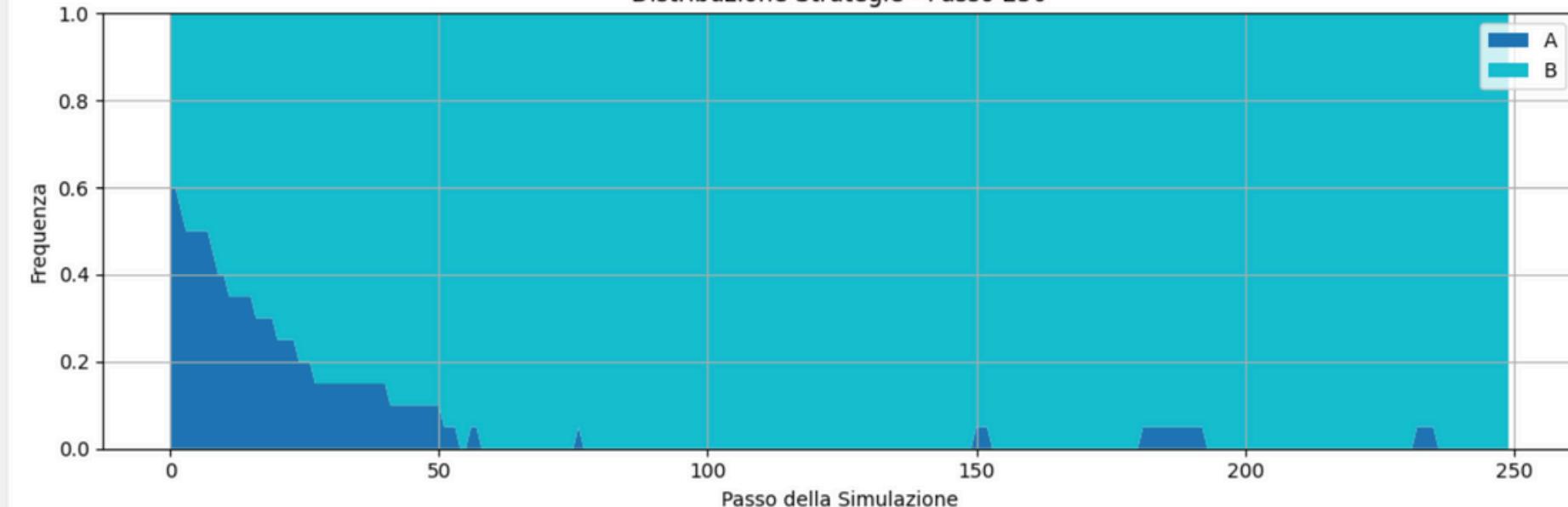
Distribuzione Strategie (Stacked Area)



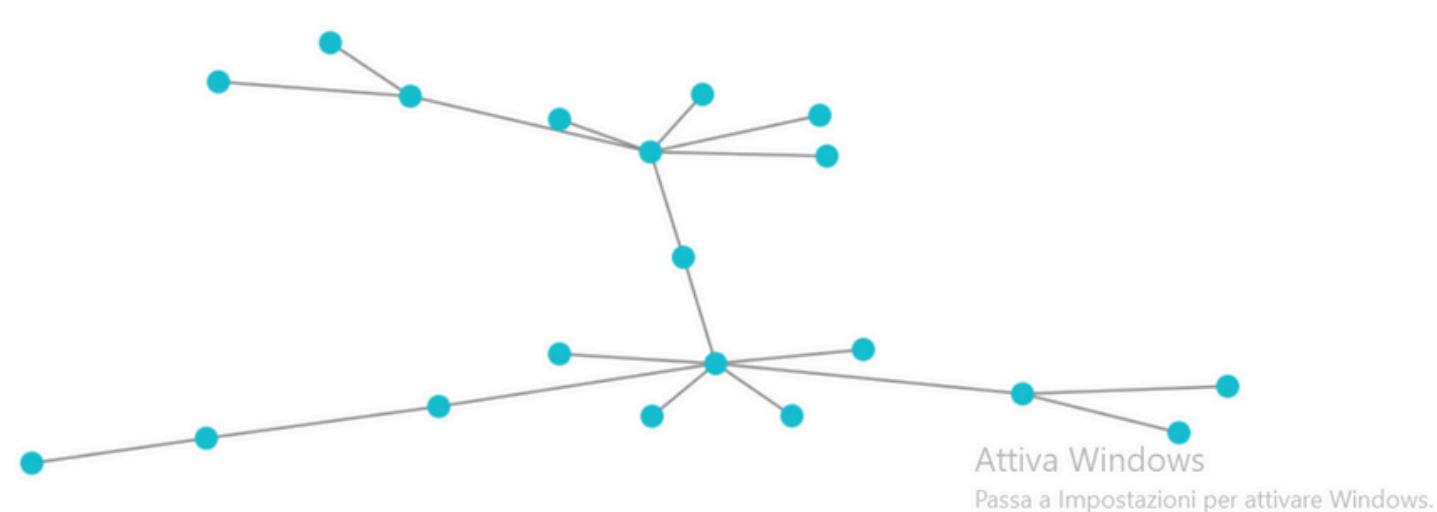
Rete degli Agenti



Distribuzione Strategie - Passo 250

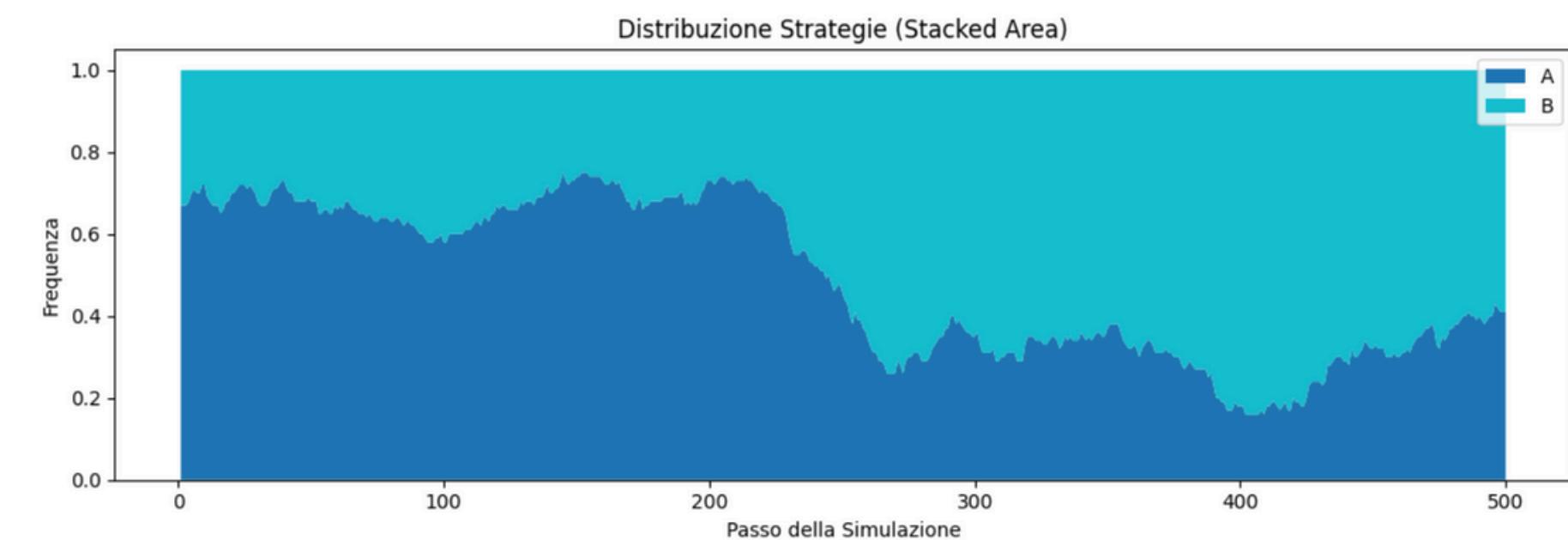


Passo 250

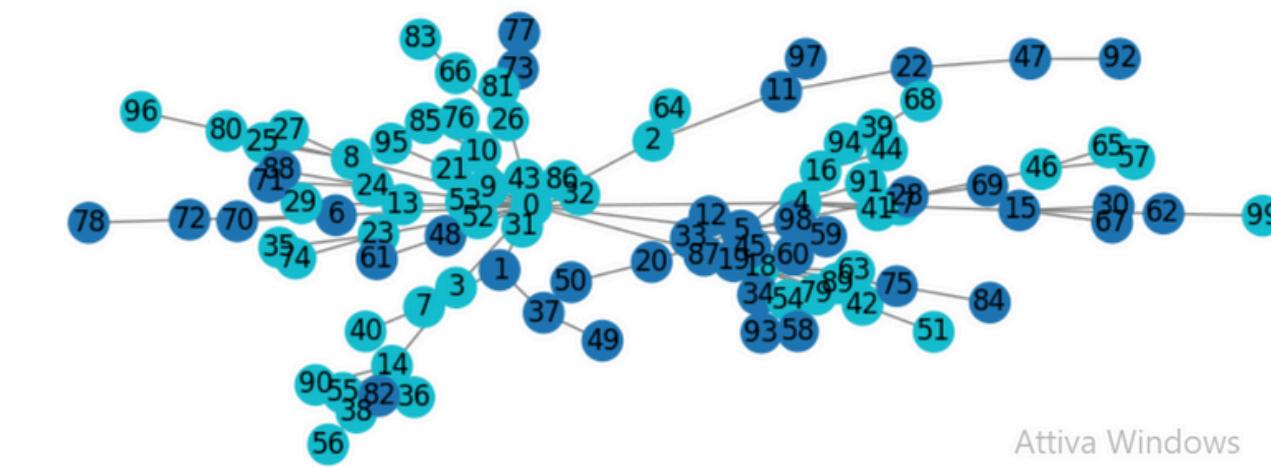


- Con pochi agenti e quindi un numero di hub limitato (max 30 agenti), gli hub tendono ad avere una forte influenza portando una strategia a prevalere
- Con molti agenti e quindi molti hub è più difficile che uno prevalga sull'altro, spesso si influenzano e le tendenze cambiano, ma difficilmente una strategia prevale incontrastata
- Sensibile alla distribuzione di probabilità

Sample runs: Preferential attachment

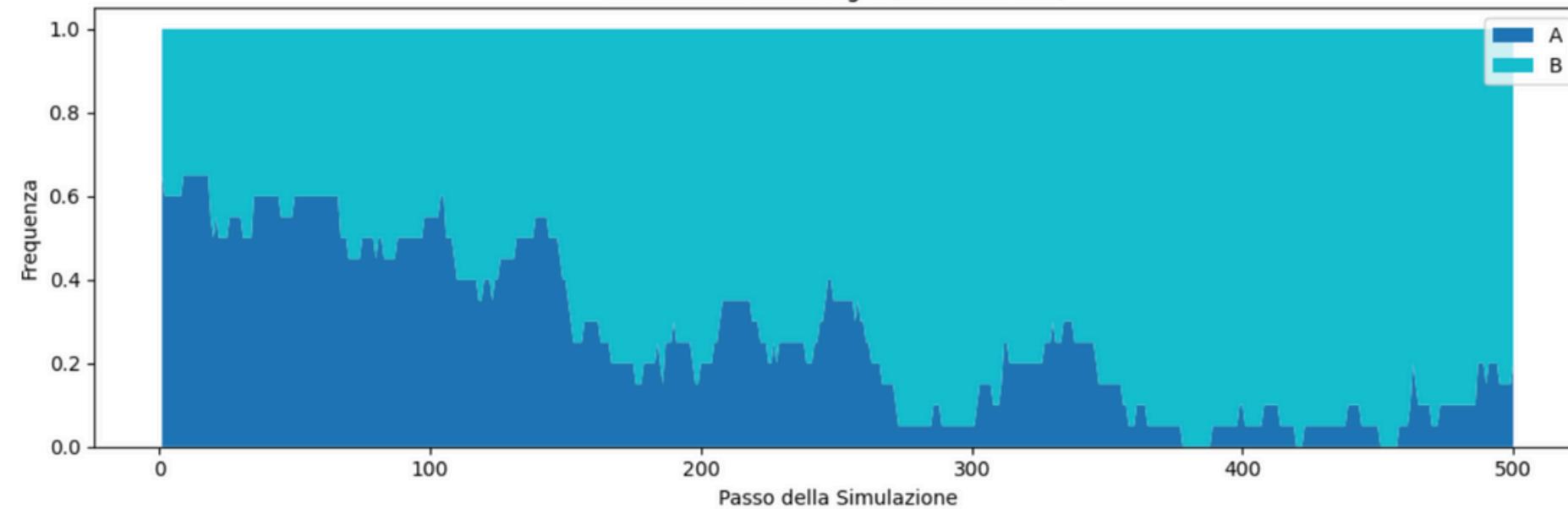


Rete degli Agenti

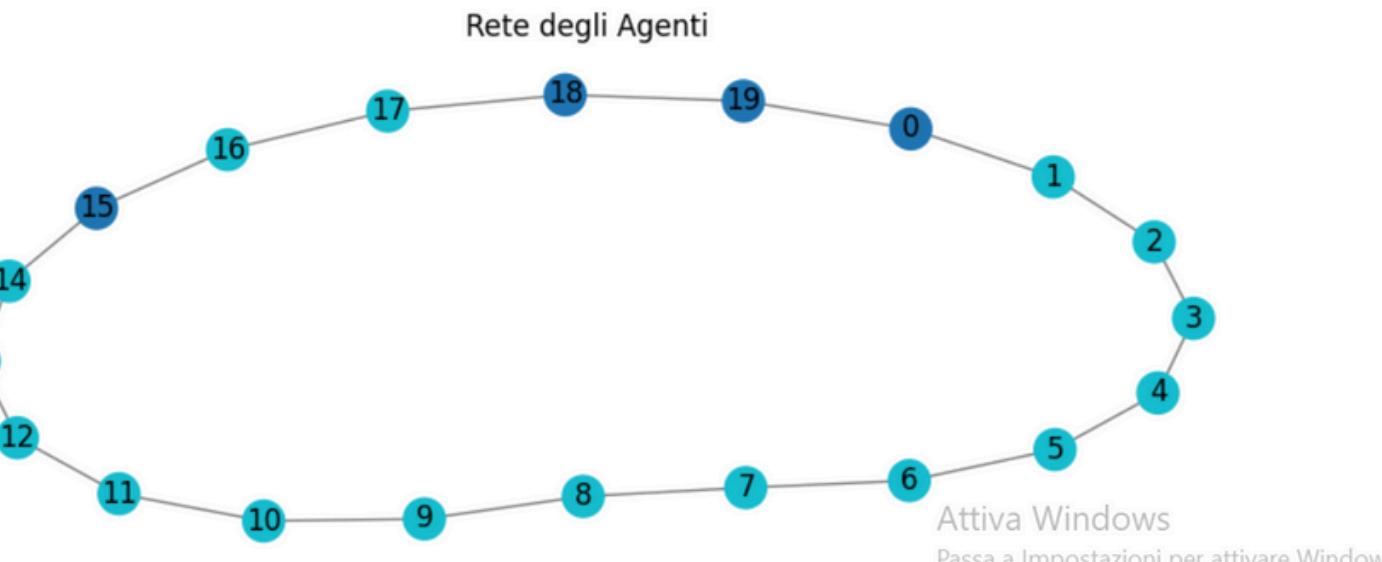


Attiva Windows
Passa a Impostazioni per attivare Windows.

Distribuzione Strategie (Stacked Area)

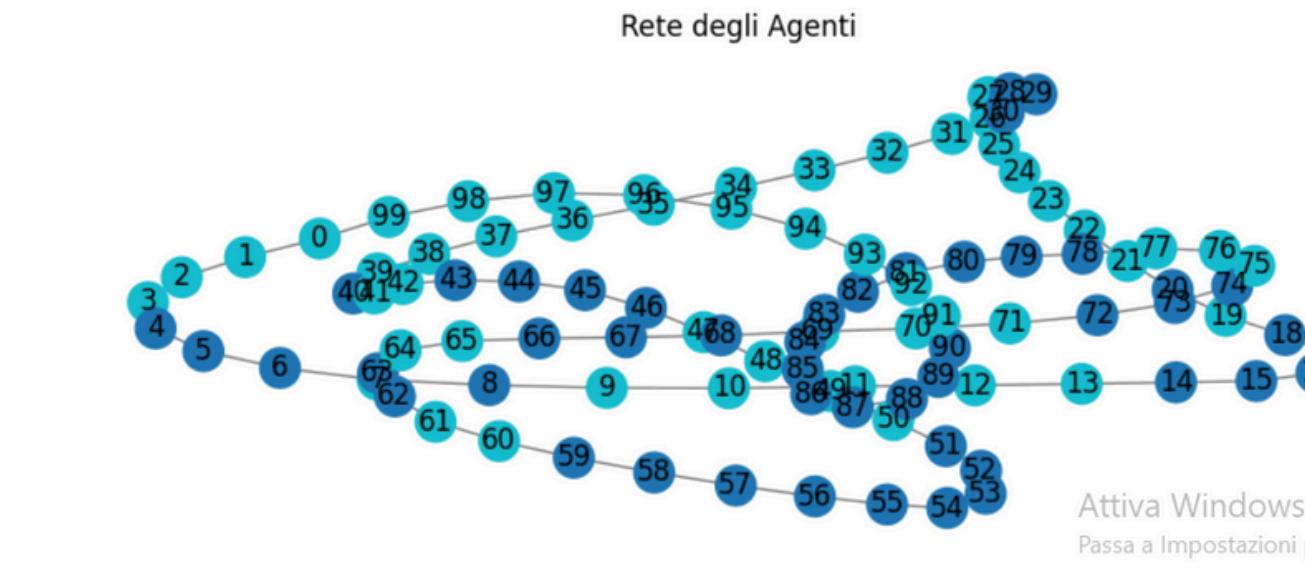
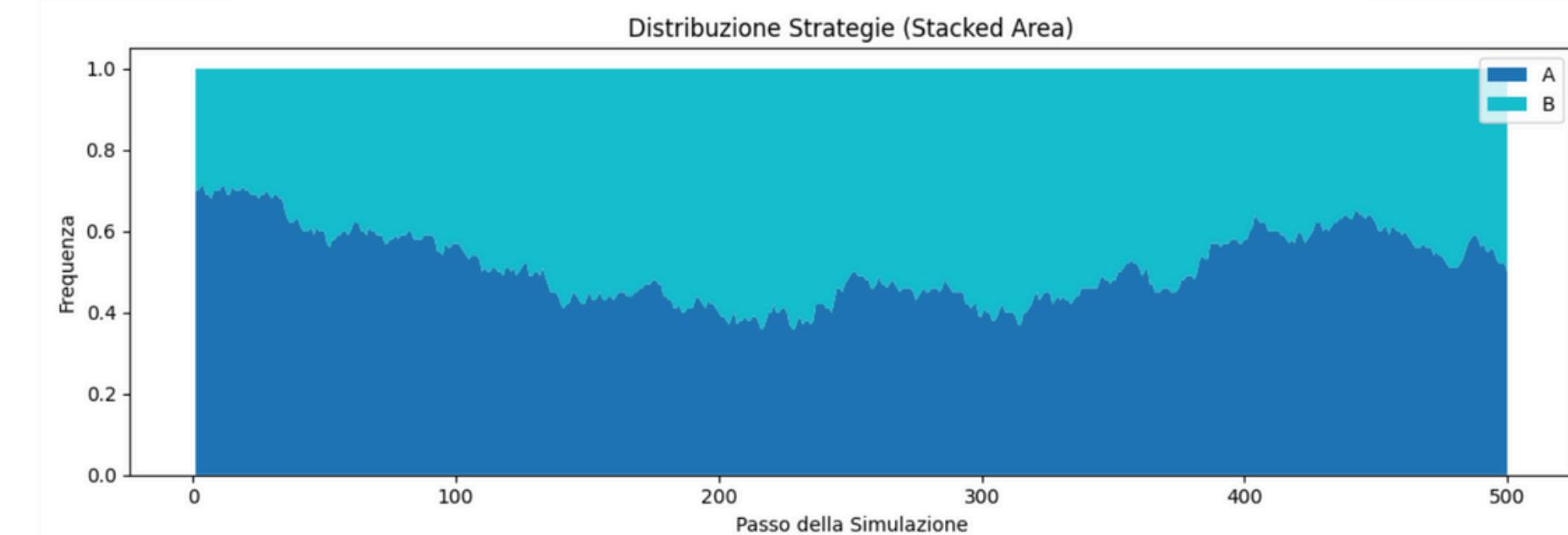


Rete degli Agenti

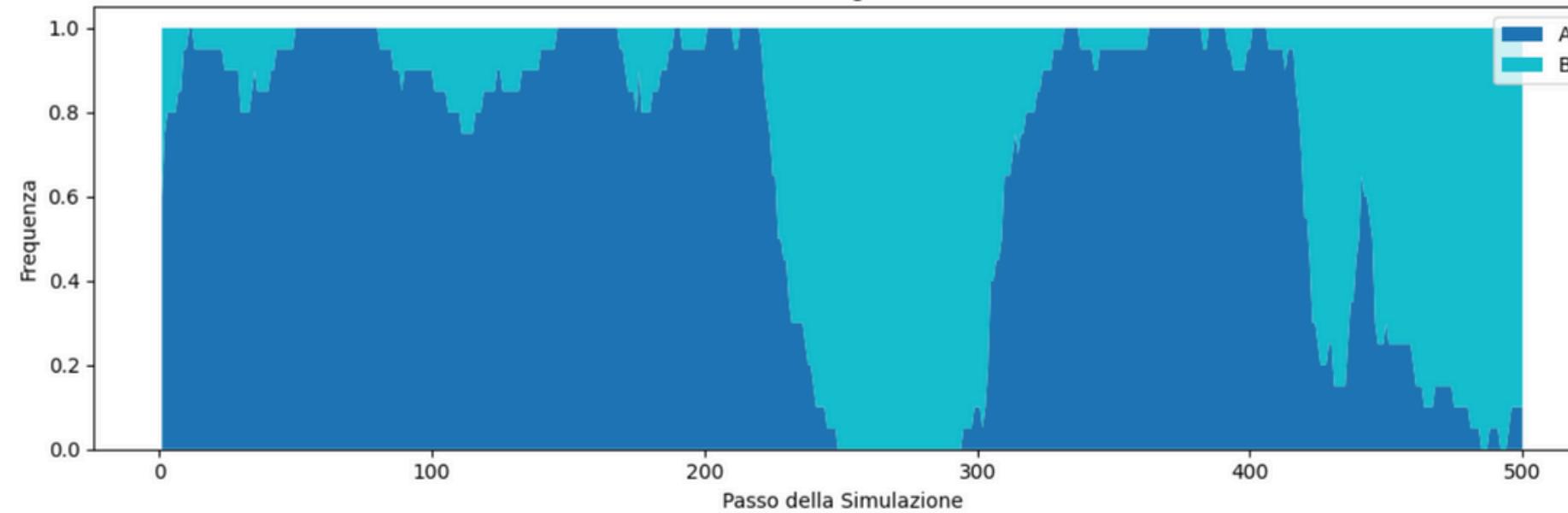


- La diffusione della conoscenza è estremamente lenta, anche in ambienti con soli 20 agenti sono necessari 200 passi perché vinca la strategia vantaggiosa, ed è raro che accada
- Estremamente sensibile alla distribuzione iniziale di probabilità
- "Ristagno" di idee, per questa ragione probabilmente rimane stabile

Sample runs: Ring network

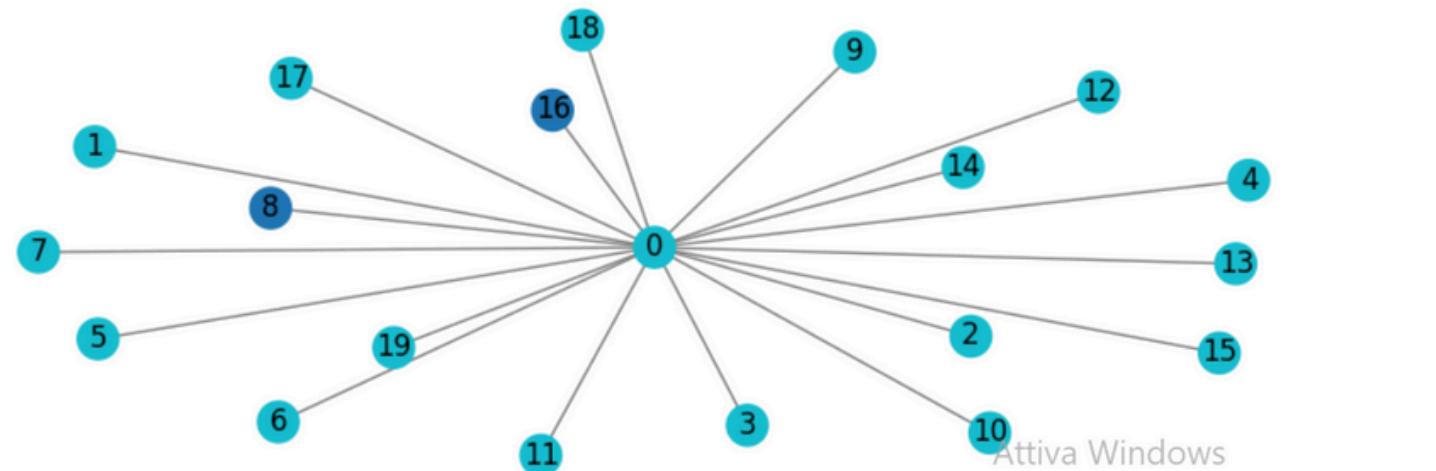


Distribuzione Strategie (Stacked Area)



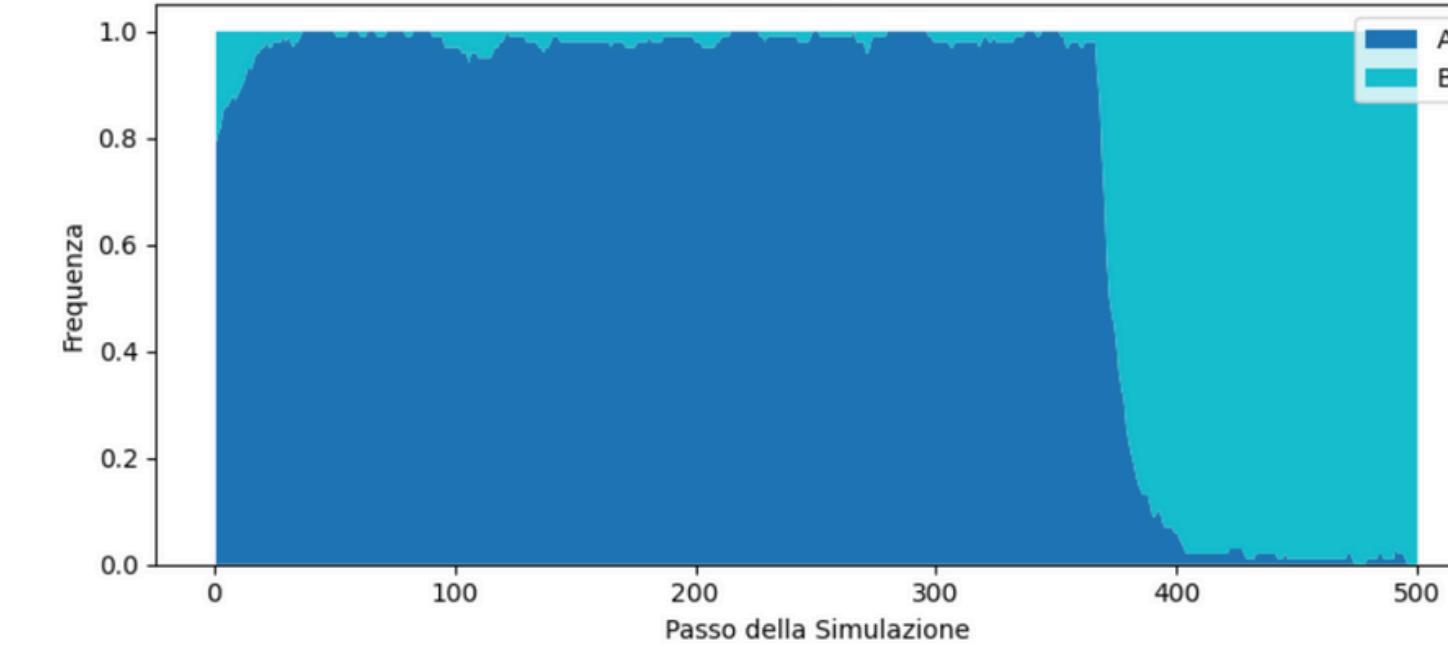
Sample runs: Star network

Rete degli Agenti

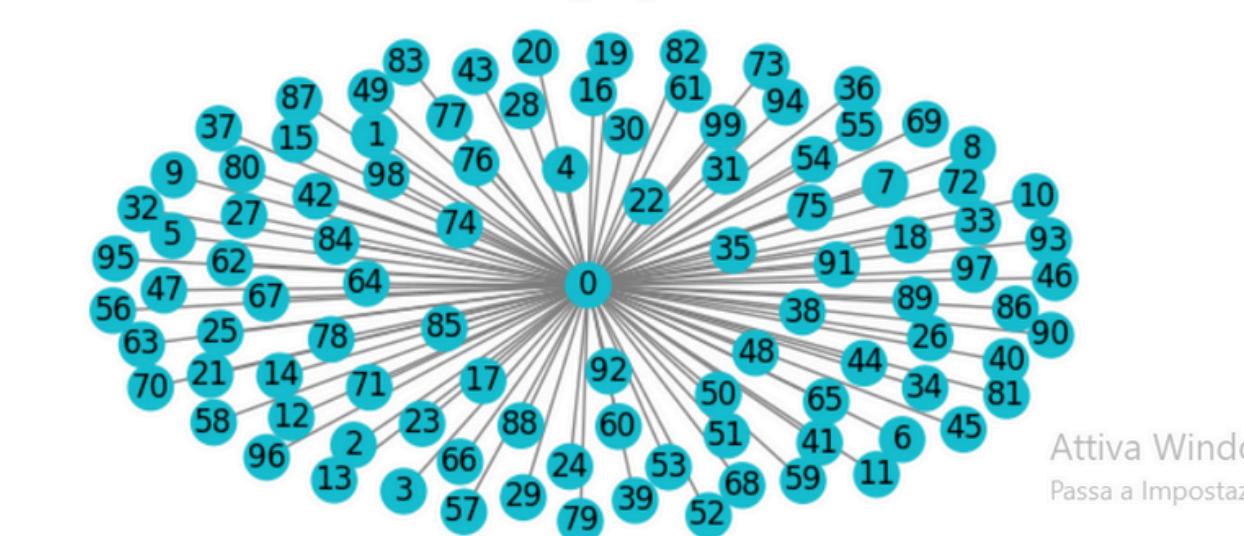


- Prima arriva la strategia ottimale al centro prima vincerà e soppianterà l'alternativa
- Reti più numerose di agenti aumentano la velocità in cui questo accade
- Forte influenza e probabilità di cambiare strategia
- Non è detto che le tendenze siano stabili nel tempo

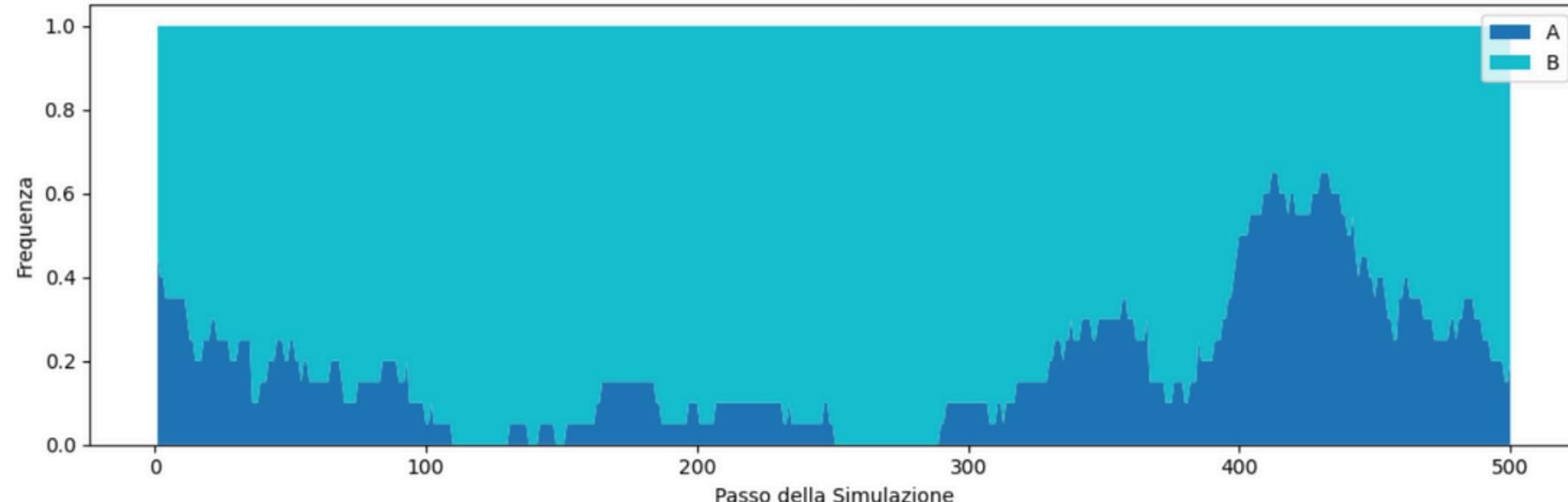
Distribuzione Strategie (Stacked Area)



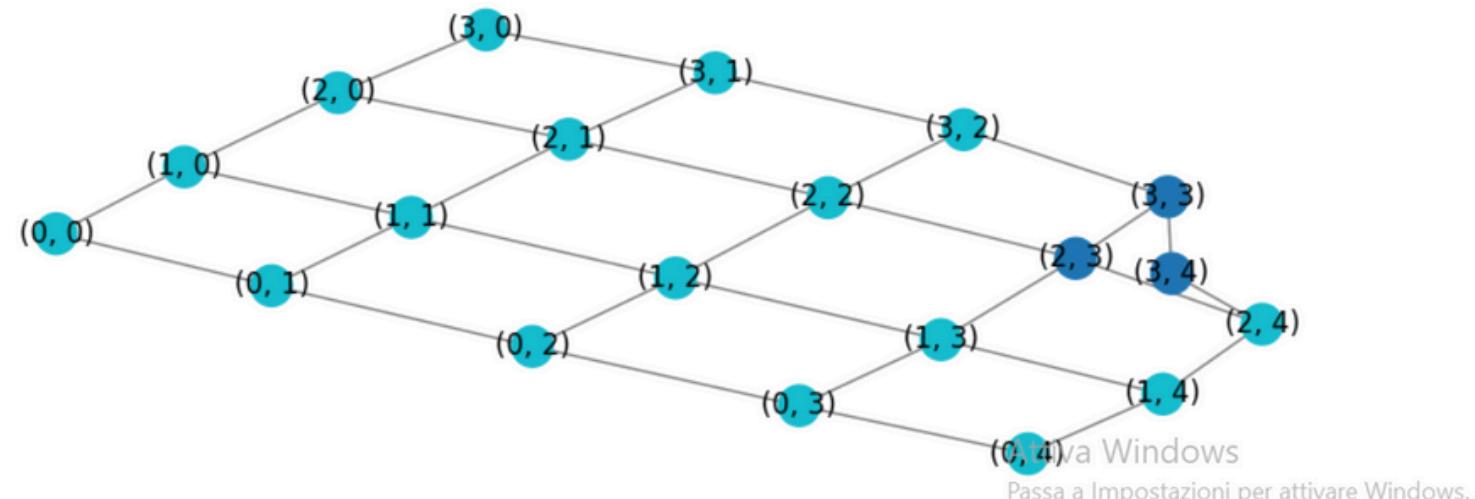
Rete degli Agenti



Distribuzione Strategie (Stacked Area)

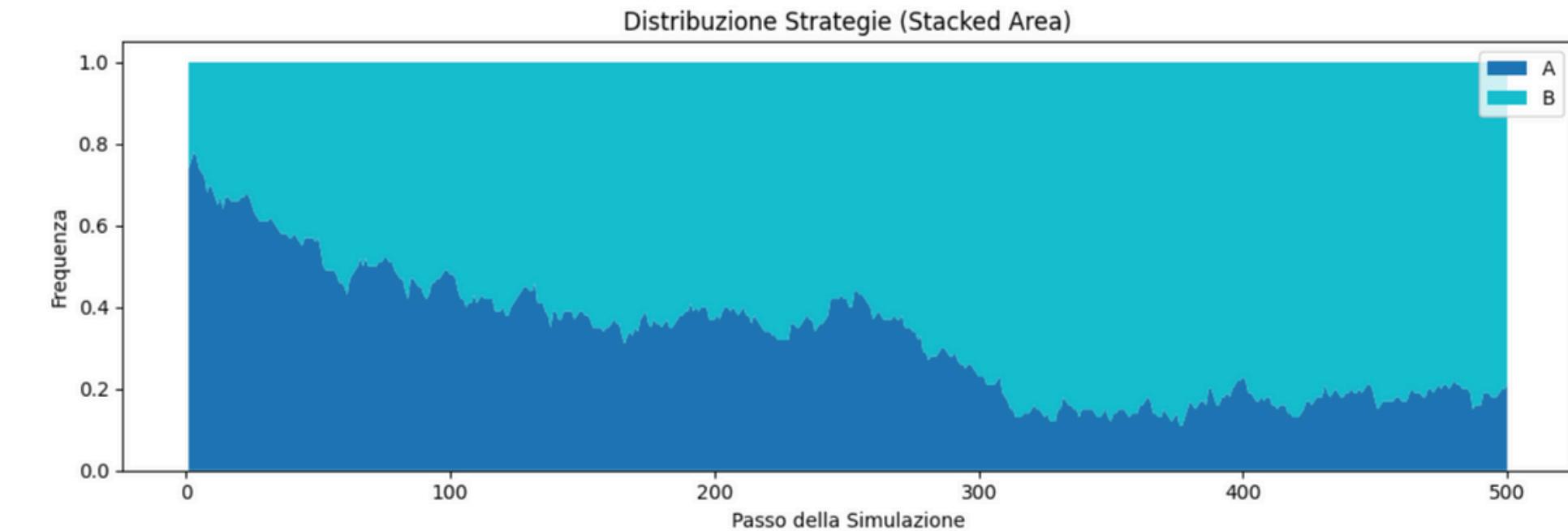


Rete degli Agenti

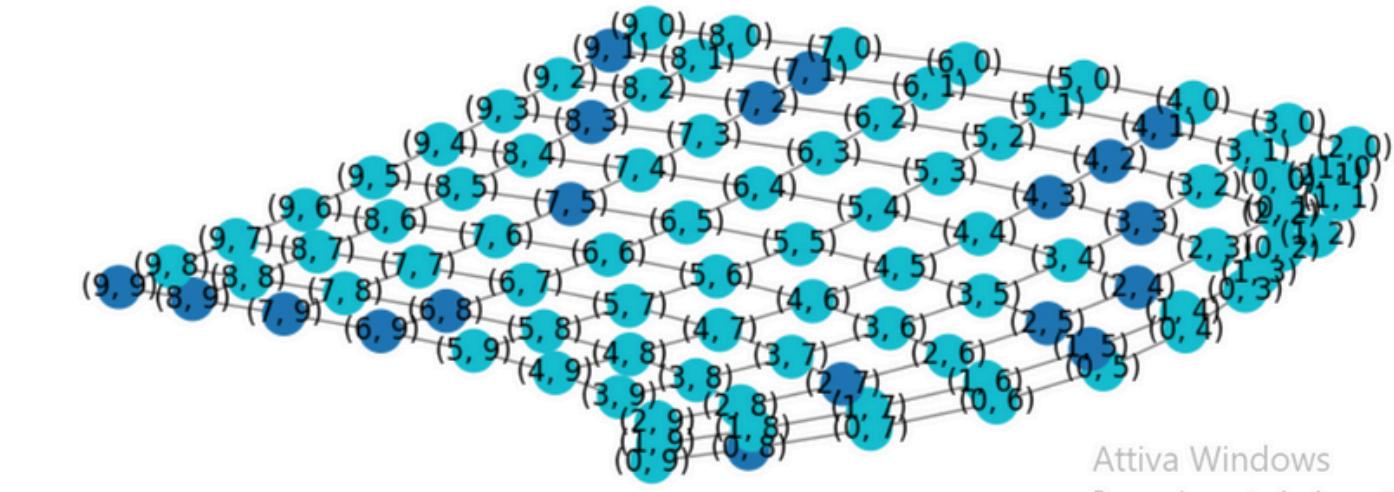


- Forte influenza di un'agente sull'altro, soprattutto nelle reti piccole
- Distanze brevi
- Influenza diretta limitata a pochi nodi
- Tendenze locali

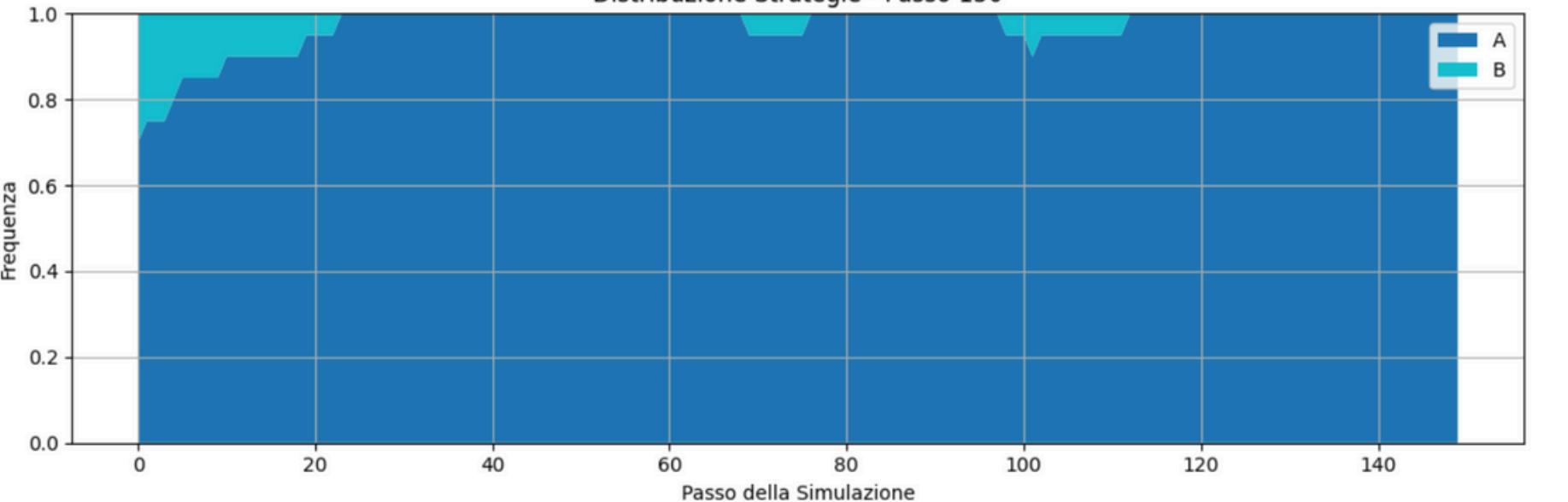
Sample runs: grid 4 neighbors



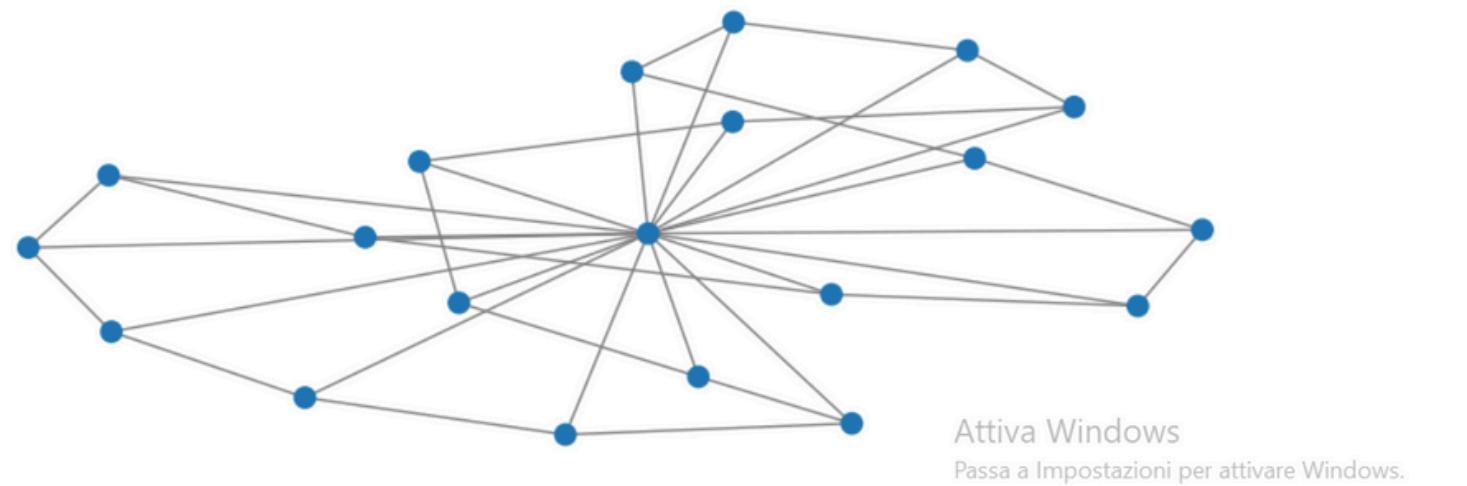
Rete degli Agenti



Distribuzione Strategie - Passo 150



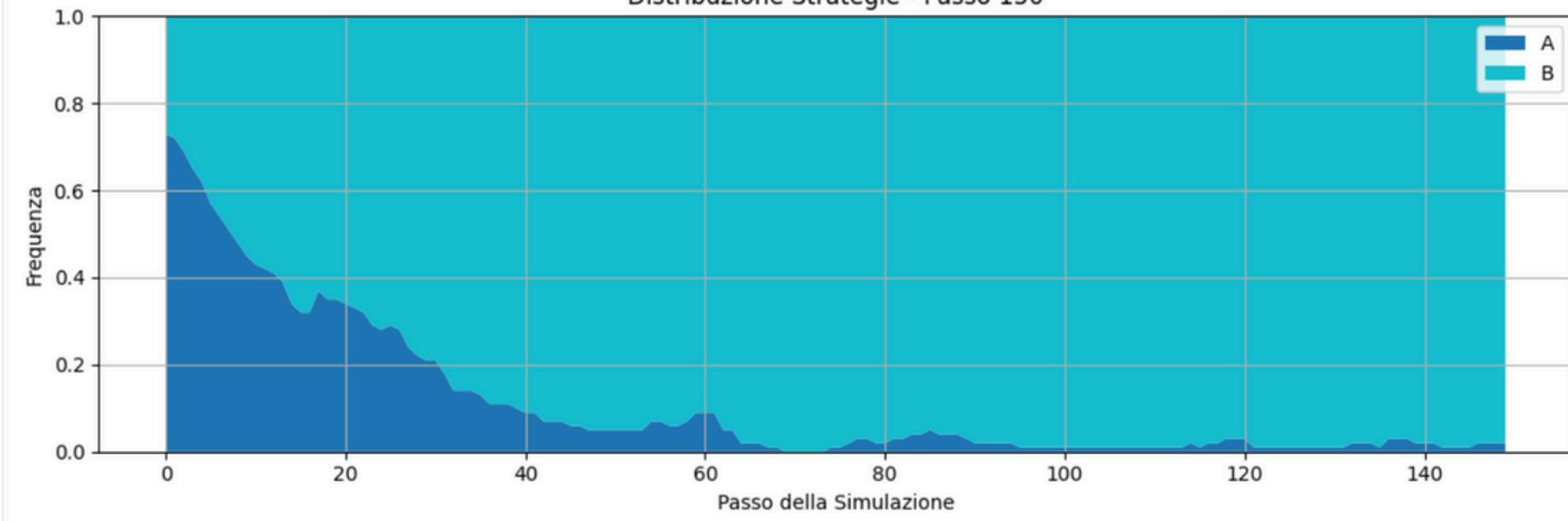
Passo 150



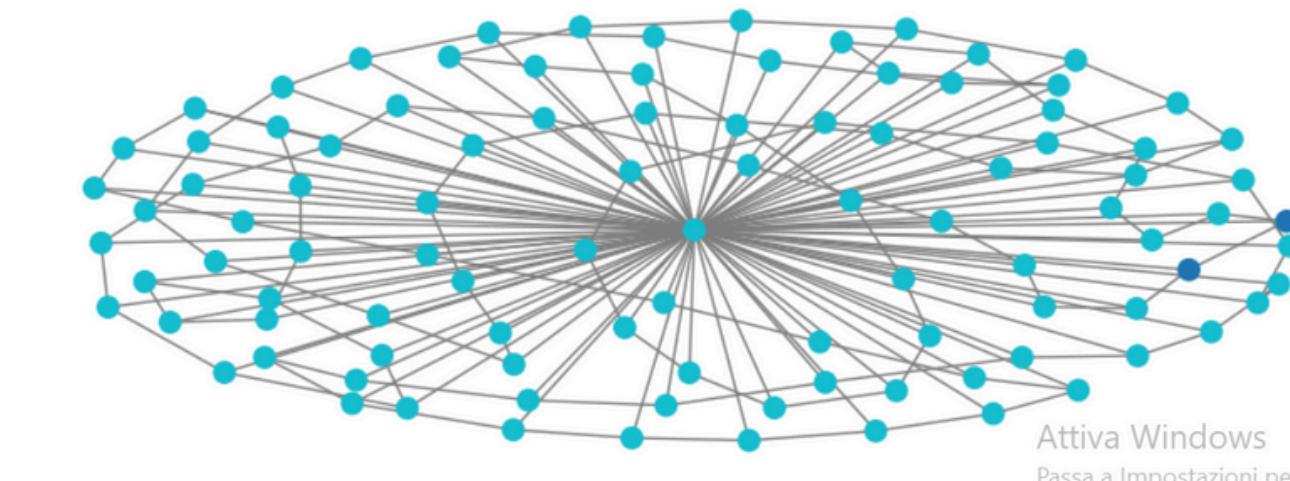
- Rete che tende ad essere estremamente polarizzata
- Dipende fortemente dalla strategia presente nel nodo centrale
- Molto veloce nella diffusione di conoscenza

Sample runs: Wheel network

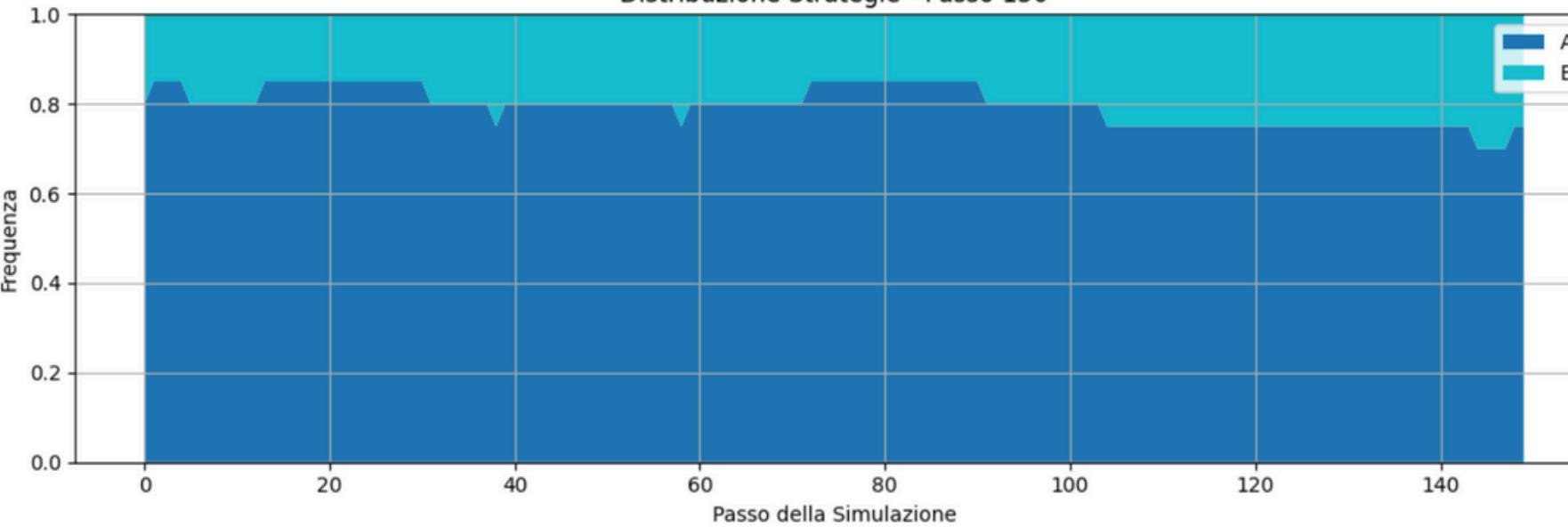
Distribuzione Strategie - Passo 150



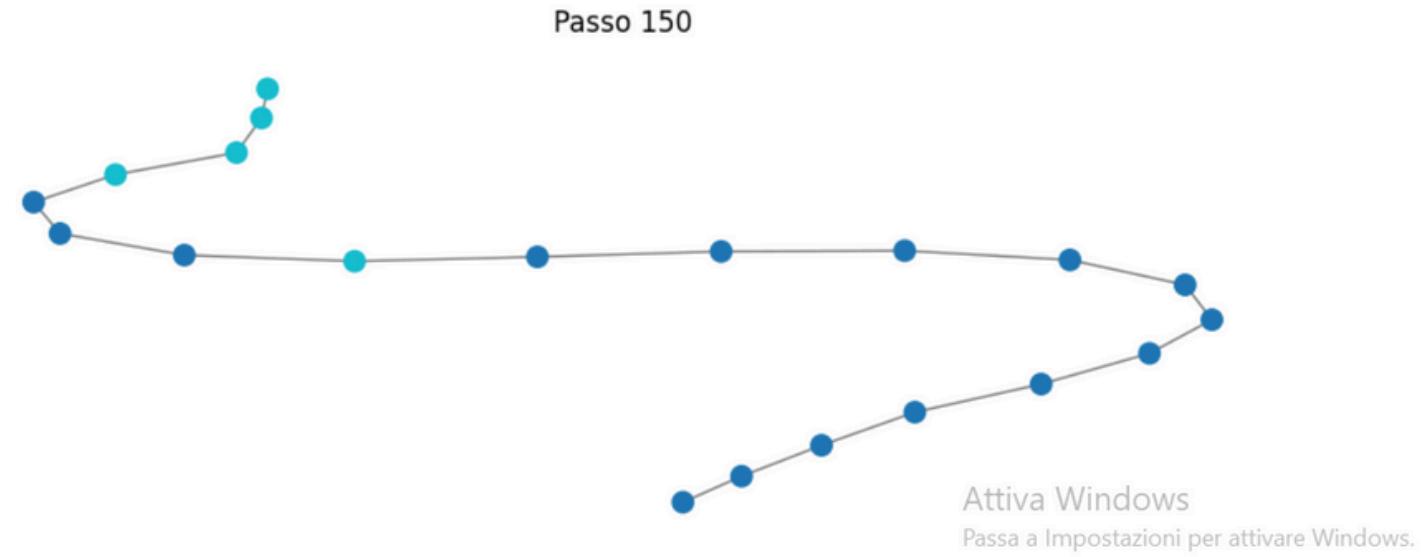
Passo 150



Distribuzione Strategie - Passo 150



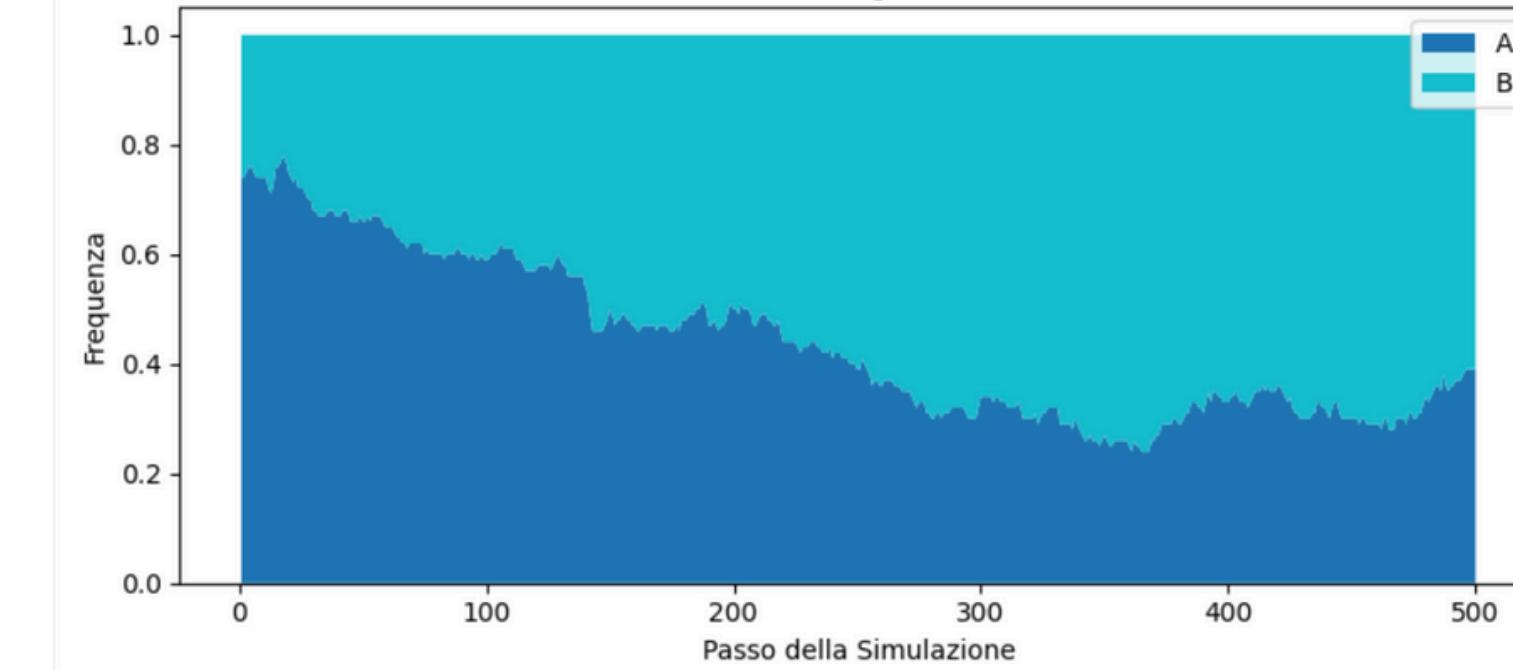
Passo 150



- Con un buon numero di agenti (100) e un grande numero di step (500) tenderà a prevalere la strategia ottimale, ma impiegherà molto tempo
- Con pochi agenti è difficile che prevalga la strategia ottimale, probabilmente dipende estremamente da dove sono collocati gli agenti che adottano la strategia ottimale
- La diffusione di conoscenza è estremamente lenta

Sample runs: Path network

Distribuzione Strategie (Stacked Area)



Rete degli Agenti



Conclusioni

Rete	Caratteristica	Output
Erdos Renyi	Estremamente dipendente dalla probabilità delle connessioni	Tendenzialmente vince la strategia non ottimale
Watts-Strogatz	Nel micromondo le connessioni sono molto vicine quindi tendono ad influenzarsi facilmente	Tendono a cambiare tendenza, ma la strategia ottimale tendenzialmente vince
Preferential-attachment	Gli hub influenzano la diffusione della strategia vincente	Vince la strategia che arriva prima agli hub
Ring network	Diffusione strategie lenta, molto sensibile alla diffusione di probabilità, una strategia difficilmente annulla l'altra	La strategia ottimale ha una buona diffusione, con molti agenti non maggioritaria

Conclusioni

Rete	Caratteristica	Output
Star network	La vittoria della strategia dipende dal nodo centrale, quella che conquista per prima questo modo vince, ma le tendenze cambiano facilmente	La vittoria di una strategia tende a dipendere dalla distribuzione iniziale
Grid 4 neighbors	Pochi vicini, distanze brevi, le tendenze locali tenderanno ad essere simili	Cambiano facilmente tendenza
Wheel network	Fortemente polarizzata, veloce diffusione della conoscenza	Vince la strategia che arriva prima al nodo centrale
Path network	Diffusione della conoscenza estremamente lenta, con molti step può vincere, ma impiega molto tempo, dipende anche in quale parte del network si trovano i nodi con la strategia ottimale	Tende a vincere la strategia più diffusa, nel nostro caso quella non ottimale

In generale

Reti con cluster

- Cambiano tendenza facilmente
- Molto polarizzate
- Veloce diffusione di informazioni
- Percorsi più brevi

Reti distribuite

- Difficoltà a cambiare tendenza
- Lenta diffusione di informazioni
- Zone con strategie uguali
- Percorsi più lunghi

Grazie per l'attenzione

E-mail

nicol.alesi@studio.unibo.it

Matricola

0001122827

