



UNIVERSITÀ DEGLI STUDI DI MILANO - BICOCCA
Dipartimento di Informatica, Sistemistica e
Comunicazione
Corso di Laurea in Informatica

Curriculum Learning applicato a tecniche di apprendimento per rinforzo nel controllo del traffico veicolare

Relatore: Prof. Vizzari Giuseppe

Tesi di Laurea di:
Sergio Vittorio Zambelli
Matricola 861172

Anno Accademico 2023-2024

Indice

1	Introduzione	2
2	Approcci e strumenti abilitanti	4
2.1	Apprendimento per Rinforzo	4
2.1.1	Q-learning	7
2.2	Apprendimento Profondo per Rinforzo	9
2.2.1	Deep Q-Network	10
2.3	Stable baselines	11
2.4	Curriculum Learning	11
2.5	SUMO	13
2.5.1	SUMO-RL	13
2.6	Agente a ciclo fisso	14
3	RL per il controllo delle luci semaforiche	15
3.1	Modello dell'ambiente	15
3.1.1	MDP	15
3.1.2	Flusso di veicoli	16
3.2	Integrazione con Stable Baseline	19
3.3	Processo di addestramento	20
3.3.1	Curriculum	21
4	Sperimentazione e risultati	23
4.1	Primi passi	23
4.2	Curriculum applicato	24
4.3	Ottimizzazione	26
4.4	Risultati	28
4.4.1	Addestramento	28
4.4.2	Test e risultati	32
4.5	Conclusioni	42
4.5.1	Sviluppi futuri	45
4.5.2	Applicazione di altri algoritmi	45
A	Appendice	50
A.1	Distribuzione Uniforme	50
A.2	Distribuzione di Weibull	50
A.3	T-Test	51
A.3.1	Test t di Welch	52

1 Introduzione

Il controllo del traffico è un problema molto importante dei nostri giorni, poiché le situazioni di grande congestione sono comuni nei centri cittadini e difficili da gestire. La sfida principale consiste nel regolare il traffico in una rete di incroci mediante l'uso di semafori, in modo da rendere più agevole il flusso dei veicoli.

Questo elaborato affronta il problema del controllo del traffico tramite agenti semaforici utilizzando tecniche di Apprendimento Profondo per Rinforzo, combinando la potenza delle reti neurali con l'efficacia dell'Apprendimento per Rinforzo e mettendole a confronto con l'approccio classico degli agenti ciclici, agenti programmati per alternare il flusso di traffico in ciascuna direzione secondo intervalli di tempo fissi e regolari, risultando quindi poco reattivi alle variazioni del traffico.

Un'altra sfida affrontata in questo lavoro è stata quella di creare agenti intelligenti capaci di convergere a comportamenti ottimali nel minor tempo possibile. Utilizzando la libreria SUMO per la simulazione dell'ambiente e la libreria Stable Baselines 3 per l'implementazione degli algoritmi di Apprendimento per Rinforzo, sono stati strutturati due diversi tipi di addestramento dell'agente: il primo classico e il secondo mediante l'utilizzo del Curriculum Learning, una tecnica che, come evidenziato da Bengio, Jérôme Louradour, Collobert et al. [5], dovrebbe ridurre il tempo richiesto dall'addestramento.

La fase di addestramento ha confermato che l'agente addestrato con un processo curriculare può raggiungere la convergenza a un comportamento ottimale in un tempo significativamente inferiore rispetto all'addestramento classico. Inoltre, la fase di test dei due modelli, ottenuti tramite tecniche di addestramento diverse, ha dimostrato che entrambi i modelli possiedono la stessa capacità di minimizzare il flusso di traffico. Il Curriculum Learning si conferma quindi un ottimo mezzo per ridurre l'utilizzo di risorse durante la fase di addestramento.

Una fase di confronto tra l'agente intelligente e l'agente a ciclo fisso ha invece rivelato differenze significative nelle loro prestazioni a seconda delle condizioni di traffico. In particolare, l'agente a ciclo fisso ha dimostrato di essere più performante in condizioni di traffico elevato, riuscendo a gestire efficacemente grandi volumi di veicoli e mantenendo il flusso del traffico più fluido. Tuttavia, quando il traffico è basso o moderato, l'agente a ciclo fisso ha mostrato prestazioni inferiori rispetto all'agente intelligente, evidenziando come i due approcci abbiano punti di forza diversi.

In Tabella 1.1 viene riportata la notazione utilizzata.

Tutte le sperimentazioni descritte in questo elaborato sono state condotte utilizzando una workstation Dell Precision 7540 equipaggiata con un processore

Notazione	Significato
RL	Apprendimento per Rinforzo
DL	Apprendimento Profondo
DRL	Apprendimento Profondo per Rinforzo
MDP	Processo Decisionale di Markov
ML	Apprendimento Automatico
DQN	Deep Q-Network

Tabella 1.1: Tabella che riporta la notazione utilizzata all'interno dell'elaborato.

Intel Core i9-9880H e una scheda grafica NVIDIA TU117GLM. Il codice necessario per replicare i risultati ottenuti è disponibile in una repository GitHub pubblica: <https://github.com/sergiozambelli/rl-traffic-lights>.

2 Approcci e strumenti abilitanti

In questo capitolo esamineremo gli approcci adottati per la realizzazione dell'elaborato e gli strumenti utilizzati. Inoltre, tratteremo gli argomenti teorici necessari per comprendere i risultati ottenuti.

2.1 Apprendimento per Rinforzo

Rielaborando quanto presentato in Kaelbling, Littman e Moore [2] l'Apprendimento per Rinforzo è una tecnica di Apprendimento Automatico il cui obiettivo è programmare agenti capaci di apprendere interagendo con un ambiente dinamico tramite un processo di tentativi ed errori. Una rappresentazione della gerarchia degli approcci all'interno dell'apprendimento automatico è riportata in Figura 2.1.

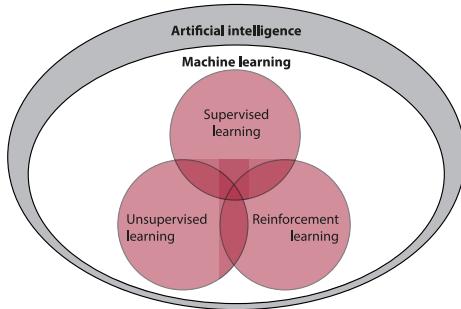


Figura 2.1: Tecniche di Apprendimento Automatico[11]

I due componenti principali all'interno di un problema di Apprendimento per Rinforzo sono l'agente e l'ambiente. L'agente è colui che prende le decisioni ed è la soluzione al problema; l'ambiente, invece, è la rappresentazione del problema stesso. La caratteristica principale dell'Apprendimento per Rinforzo risiede nell'interazione tra agente e ambiente: l'agente modifica l'ambiente attraverso le sue azioni, mentre l'ambiente risponde attivamente a tali azioni. Questo ciclo di azione e reazione rende il sistema dinamico.

L'agente, che prende le decisioni, è definito da un algoritmo di Apprendimento per Rinforzo e, nella maggior parte dei casi, ha un ciclo di vita che consiste di tre stati interconnessi, come rappresentato in Figura 2.2:

- L'agente interagisce con l'ambiente ottenendo una sua rappresentazione e dei dati

- L'agente valuta il proprio comportamento sulla base dei dati ottenuti
- L'agente migliora il proprio comportamento per ottenere risultati migliori nel lungo periodo.

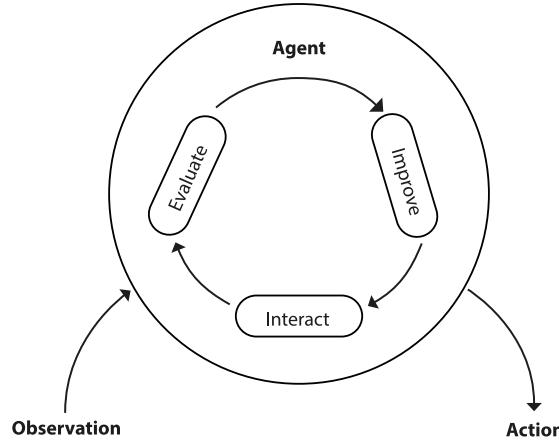


Figura 2.2: Rappresentazione del ciclo di vita di un agente all'interno di un problema di Apprendimento per Rinforzo. L'immagine è presa da Morales [11], pagina 36.

L'ambiente è una rappresentazione del problema reale in termini di Apprendimento per Rinforzo. La formulazione più adatta per questo contesto è quella dei Markov Decision Process, una struttura matematica per la rappresentazione di problemi decisionali complessi con incertezza. I problemi di Apprendimento per Rinforzo sono infatti tipicamente complessi e caratterizzati da incertezza, poiché:

- L'agente deve prendere decisioni in base a stati parzialmente osservabili e variabili nel tempo
- Le conseguenze delle azioni dell'agente non sono sempre prevedibili con certezza
- L'agente deve trovare un compromesso tra l'esplorazione di nuove strategie e lo sfruttamento delle conoscenze acquisite.

Un MDP è definito da una quintupla $(S, A, T, \mathcal{R}, \mathcal{E})$:

- S è lo spazio degli stati dell'agente, ossia l'insieme di tutti gli stati in cui l'agente può trovarsi
- A è lo spazio delle azioni dell'agente, ossia l'insieme di tutte le azioni che lo stato può intraprendere
- $T(s, a, s')$ rappresenta la funzione di transizione da stato a stato, che restituisce la probabilità che eseguendo l'azione a nello stato s si finisca nello stato s'

- $\mathcal{R}(s, a, s')$ è la funzione di ricompensa, che restituisce la ricompensa ricevuta durante la transizione allo stato s' dopo aver eseguito l'azione a nello stato s
- $\mathcal{E} \subseteq S$ è l'insieme degli stati finali, che una volta raggiunti impediscono qualsiasi futura azione o ricompensa terminando la simulazione.

All'interno di un problema di Apprendimento per Rinforzo, le interazioni tra agente e ambiente si ripetono per numerosi cicli come rappresentato in Figura 2.3.

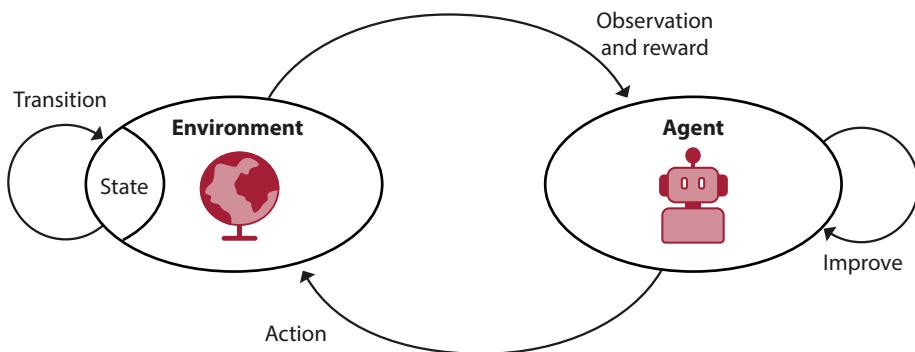


Figura 2.3: Rappresentazione del ciclo di vita di un agente in un problema di Apprendimento per Rinforzo. L'ambiente fornisce all'agente delle osservazioni che permettono di valutare quale azione intraprendere. Una volta eseguita l'azione, l'ambiente si modifica e invia all'agente una ricompensa. L'immagine è stata presa da Morales [11], pagina 9.

A ogni iterazione, l'ambiente mette a disposizione dell'agente un numero di azioni variabile a seconda dello stato attuale; la scelta dell'agente modifica l'ambiente tramite una funzione di transizione che porta dallo stato attuale al successivo. A questo punto, l'ambiente fornisce un segnale di ricompensa secondo la funzione di ricompensa e offre una nuova osservazione. Il segnale di ricompensa definisce l'obiettivo del problema, ossia il comportamento desiderato, poiché guida l'agente nel processo di apprendimento. L'agente utilizzerà la ricompensa ottenuta per valutare quale sarà l'azione migliore da intraprendere nel prossimo ciclo.

Quanto esposto spiega il motivo per cui si parla di un processo di tentativi ed errori: l'agente, inizialmente privo di informazioni sull'ambiente eccetto quelle immediatamente osservabili, deve provare diverse azioni per acquisire maggiori conoscenze sull'ambiente. Queste azioni, inizialmente scelte casualmente, porteranno inevitabilmente a errori, segnalati attraverso ricompense negative. Continuando ad agire, l'agente apprenderà quali azioni portano a ricompense negative e quali a ricompense positive, creando una rappresentazione gerarchica delle azioni basata sulla loro positività o negatività. Questo processo è definito come esplorazione (exploration). Il processo di agire sulla base delle conoscenze acquisite è invece definito come sfruttamento (exploitation) e permette all'agente di avvicinarsi al suo obiettivo, ossia un comportamento ottimale, basandosi su quanto ottenuto dalla fase di esplorazione. Il bilanciamento tra il tempo dedicato a questi due processi contrapposti è un argomento di studio rilevante, poiché

influisce significativamente sul tempo di computazione di un algoritmo, specialmente in ambienti molto complessi o ad alta dimensionalità. Esistono varie tecniche per gestire il dilemma dell'esplorazione e dello sfruttamento, una delle quali è di particolare interesse poiché sarà implementata da uno degli algoritmi utilizzati nell'elaborato. Questa strategia è chiamata decaying epsilon-greedy e consiste nel massimizzare l'esplorazione inizialmente, per poi concentrarsi sullo sfruttamento, come spiegato in Morales [11]. In un approccio epsilon-greedy standard, si sceglie sulla base di un fattore epsilon se effettuare l'azione con il valore stimato maggiore (sfruttamento) o se esplorare facendo un'azione casuale (esplorazione). L'aggiunta di un fattore di decadimento per il valore della variabile epsilon porta all'approccio di decaying epsilon-greedy: man mano che la simulazione avanza, si riduce progressivamente l'esplorazione, dando sempre più spazio allo sfruttamento della conoscenza acquisita. Questo permette all'agente di imparare inizialmente esplorando diverse possibilità e successivamente di ottimizzare il suo comportamento basandosi sulle esperienze raccolte. In 2.4 viene riportata un'implementazione possibile dell'approccio decaying epsilon-greedy lineare, preso da Morales [11].

```

def lin_dec_epsilon_greedy(env,
                           init_epsilon=1.0,
                           min_epsilon=0.01,
                           decay_ratio=0.05,
                           n_episodes=5000):
    <...>

    name = 'Line-greedy {} {} {}'.format(init_epsilon, min_epsilon,
                                         decay_ratio)
    for e in tqdm(range(n_episodes), desc='Episodes for: ' + name,
                  leave=False):
        # Epsilon decade linearmente sulla base del numero di iterazioni
        # effettuate
        decay_episodes = n_episodes * decay_ratio
        epsilon = 1 - e / decay_episodes
        epsilon *= init_epsilon - min_epsilon
        epsilon += min_epsilon
        epsilon = np.clip(epsilon, min_epsilon, init_epsilon)
        # Un numero randomico viene paragonato a epsilon; se il numero
        # estratto è maggiore allora si sfrutta, sennò si esplora
        if np.random.random() > epsilon:
            action = np.argmax(Q)
        else:
            action = np.random.randint(len(Q))

    <...>

    return name, returns, Q, actions

```

Figura 2.4: Possibile implementazione di un approccio decaying epsilon-greedy fornito in Morales [11].

2.1.1 Q-learning

Q-learning è un algoritmo di Apprendimento per Rinforzo che permette all'agente di apprendere una strategia ottimale all'interno di un MDP basandosi

sulle conseguenze delle sue azioni. L'agente, infatti, prova un'azione in un certo stato e poi valuta le conseguenze basandosi sia sulla ricompensa immediata che sulla stima del valore dello stato di partenza. Viene in seguito rielaborato quanto presentato in Watkins e Dayan [1].

Il problema

Consideriamo un agente che si muove all'interno di un ambiente rappresentato tramite un MDP.

Il compito dell'agente è determinare una politica ottimale, che massimizza la ricompensa totale scontata; per ricompensa scontata, intendiamo che le ricompense ricevute s passi nel futuro valgono meno delle ricompense ricevute ora, di un fattore γ^s ($0 < \gamma < 1$), come rappresentato in Figura 2.5.

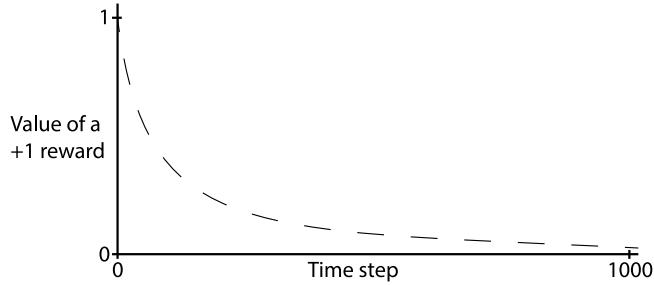


Figura 2.5: Grafico della ricompensa scontata: ad ogni iterazione, il valore della ricompensa diminuisce tendendo a zero. L'immagine è presa da Morales [11], pagina 58.

Preso una politica π , il valore associato a uno stato x del sistema è

$$V^\pi(x) \equiv \mathcal{R}_x(\pi(x)) + \gamma \sum_y P_{xy}[\pi(x)] V^\pi(y).$$

poichè l'agente si aspetta di ricevere la ricompensa $\mathcal{R}_x(\pi(x))$ immediatamente dopo aver eseguito l'azione che π suggerisce, e poi si sposta con probabilità $P_{xy}[\pi(x)]$ in uno stato dal valore $V^\pi(y)$. La teoria della Programmazione Dinamica ci assicura che esiste almeno una politica ottimale π^* tale per cui

$$V^*(x) \equiv V^{\pi^*}(x) = \max_a \left\{ \mathcal{R}_x(a) + \gamma \sum_y P_{xy}[a] V^{\pi^*}(y) \right\}$$

è l'azione migliore che un agente possa fare dallo stato x . Il compito dell'algoritmo $Q - learning$ è di determinare una π^* senza conoscere inizialmente i valori di $\mathcal{R}_x(a)$ e $P_{xy}[a]$.

L'approccio

Per una politica π , definiamo i valori Q come:

$$Q^\pi(x, a) = \mathcal{R}_x(a) + \gamma \sum_y P_{xy}[\pi(x)] V^\pi(y).$$

In altre parole, il valore Q è la ricompensa scontata attesa per eseguire l'azione a nello stato x e seguire successivamente la politica π . L'obiettivo di Q-learning è stimare i valori Q per una politica ottimale. Definiamo questi valori come $Q^*(x, a) \equiv Q^{\pi^*}(x, a), \forall x, a$.

Dal momento che $V^*(x) = \max_a Q^*(x, a)$ e che se a^* è l'azione in cui si raggiunge il massimo, allora una politica ottimale può essere formata come $\pi^*(x) \equiv a^*$. Qui risiede l'utilità dei valori Q : se un agente riesce ad apprenderli, può facilmente decidere cosa è ottimale fare. Nel Q-learning, l'agente si muove seguendo una sequenza di operazioni:

- osserva il suo stato corrente x_n ,
- seleziona ed esegue un'azione a_n ,
- osserva lo stato successivo y_n ,
- riceve una ricompensa immediata r_n , e
- modifica i suoi valori Q_{n-1} usando un fattore di apprendimento α_n , secondo:

$$Q_n(x, a) = \begin{cases} (1 - \alpha_n)Q_{n-1}(x, a) + \alpha_n[r_n + \gamma V_{n-1}(y_n)] & \text{se } x = x_n \text{ e } a = a_n \\ Q_{n-1}(x, a) & \text{altrimenti} \end{cases}$$

dove

$$V_{n-1}(y) \equiv \max_b \{Q_{n-1}(y, b)\}$$

è l'azione migliore che l'agente pensa di poter fare dallo stato y . I valori iniziali Q , $Q_0(x, a)$, per tutti gli stati e le azioni sono assunti dati. Questa successione di azioni converge a una politica ottimale[1].

2.2 Apprendimento Profondo per Rinforzo

Rielaborando quanto presentato in Morales [11], con Apprendimento Profondo per Rinforzo si intende un approccio ai problemi di Apprendimento per Rinforzo che utilizza tecniche di approssimazione di funzioni non lineari a più strati, tipicamente reti neurali (Apprendimento Profondo).

L'Apprendimento Profondo per Rinforzo ha ampliato l'applicabilità dell'Apprendimento per Rinforzo a problemi di dimensioni maggiori rispetto al passato, aumentando notevolmente le potenzialità di tale approccio. Questo è dovuto al fatto che le reti neurali profonde, strumento principale del DL, permettono di trovare automaticamente rappresentazioni compatte e ridotte di dati ad alta dimensionalità, come testi, immagini o audio. In questo modo si riduce la necessità di elaborare manualmente i dati, un'operazione costosa in termini di risorse e che richiede la partecipazione di esperti del settore.

Naturalmente, l'utilizzo di queste tecniche comporta anche alcuni aspetti negativi. Le reti neurali, infatti, sono "data-hungry", ossia richiedono grandi quantità di dati per raggiungere una convergenza ottimale. Inoltre, risultano

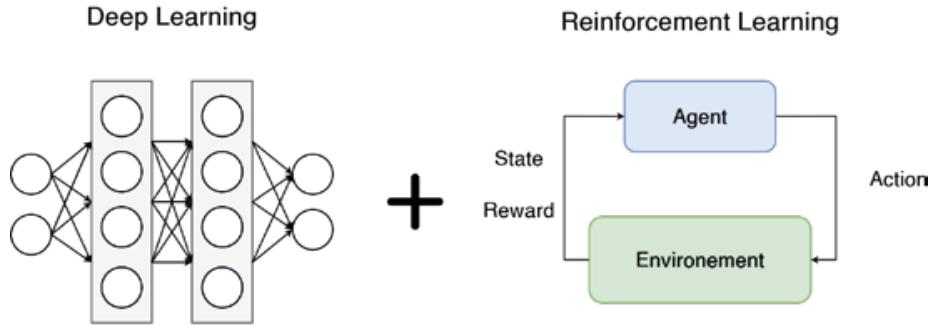


Figura 2.6: Rappresentazione della struttura di un agente che utilizza un algoritmo di Apprendimento Profondo per Rinforzo. Illustrazione presa da Vidali, Crociani, Vizzari et al. [10].

difficili da interpretare, poiché le logiche applicative rimangono nascoste all'interno dei layer di neuroni. Tuttavia, questi aspetti negativi non sono sufficienti per dissuadere dall'uso di questo approccio, data la loro potenza nell'approssimazione di funzioni e le loro prestazioni, che spesso risultano essere le migliori possibili.

Questo progresso ha reso possibile l'applicazione dell'Apprendimento per Rinforzo a problemi precedentemente intrattabili, come riportato in Arulkumaran, Deisenroth, Brundage et al. [6], portando a grandi successi come la creazione di AlphaGo, un sistema di Apprendimento per Rinforzo Profondo in grado di sconfiggere il campione mondiale di Go.

2.2.1 Deep Q-Network

Deep Q-Network (DQN), introdotto da DeepMind, è stato uno degli algoritmi più popolari dell'Apprendimento Profondo per Rinforzo, poiché pioniere di ricerche innovative nella storia dell'Apprendimento per Rinforzo. DQN è stato infatti il primo algoritmo a ottenere prestazioni superumane su una piattaforma di benchmark di Atari.

Come spiegato in Morales [11], DQN nasce con l'obiettivo di superare i limiti imposti da un classico problema che si incontra nel campo dell'Apprendimento per Rinforzo: la violazione di due assunzioni molto comuni nelle tecniche di ottimizzazione, ovvero l'assunzione che i dati siano indipendenti e identicamente distribuiti, e l'assunzione che la funzione obiettivo sia stazionaria.

A differenza di quanto accade nell'Apprendimento Supervisionato, il dataset non è noto e modificabile a priori: i dati sono spesso generati mentre l'algoritmo apprende. Questo crea una correlazione tra i dati generati in sequenza, compromettendone l'indipendenza e, man mano che la politica migliora, cambia il processo di generazione dei dati nell'ambiente, violando così l'assunzione che i dati siano identicamente distribuiti.

Per quanto riguarda la stazionarietà, questa viene compromessa dal fatto che la funzione obiettivo si modifica a ogni passo del processo di ottimizzazione.

DQN implementa l'algoritmo Q-Learning con quattro principali aggiunte volte a risolvere questi problemi:

- La funzione Q viene approssimata tramite una rete neurale profonda convoluzionaria
- Viene aggiunto uno spazio di memoria dedicato a salvare i risultati degli ultimi step, chiamato memoria di replay
- I dati di addestramento vengono prelevati da mini-batch estratti casualmente dalla memoria di replay, piuttosto che solo dai risultati dell'ultimo passaggio effettuato (tecnicamente chiamata *experience replay*); in questo modo si massimizza la possibilità di avere a che fare con dati indipendenti e identicamente distribuiti
- Il valore Q dello stato successivo viene valutato usando parametri più longevi all'interno della rete; ciò rende la valutazione della funzione obiettivo più stabile.[7]

2.3 Stable baselines

Stable Baselines è una raccolta open source di implementazioni di algoritmi di Apprendimento per Rinforzo e Apprendimento Profondo per Rinforzo, sviluppata in PyTorch[12].

La scelta di Stable Baselines in questo elaborato è stata motivata da diversi fattori. Prima di tutto, la semplicità di implementazione e utilizzo rappresenta un vantaggio significativo: Stable Baselines offre infatti interfacce intuitive e ben documentate, facilitando l'integrazione degli algoritmi.

Un altro motivo fondamentale per la scelta di Stable Baselines è la sua compatibilità con SUMO-RL. Questa compatibilità consente di utilizzare Stable Baselines per sviluppare e testare algoritmi di controllo del traffico, sfruttando la potenza dell'Apprendimento per Rinforzo.

La versione scelta è stata Stable Baselines 3, l'ultima versione disponibile al momento dello sviluppo.

2.4 Curriculum Learning

Il Curriculum Learning è un approccio sistematico e incrementale all'addestramento di algoritmi di Apprendimento Automatico che organizza il percorso di apprendimento in modo strutturato anziché casuale. Invece di presentare esempi in ordine randomico, tale metodologia parte con esempi semplici e progressivamente introduce casi sempre più complessi e articolati. L'obiettivo è facilitare l'apprendimento del modello guidandolo attraverso un processo di aumento graduale della difficoltà sino a raggiungere il comportamento desiderato, come rappresentato in Figura 2.4.

Questa strategia si ispira al modo in cui gli esseri umani apprendono nuove competenze. Il sistema educativo della nostra società si basa proprio su questo

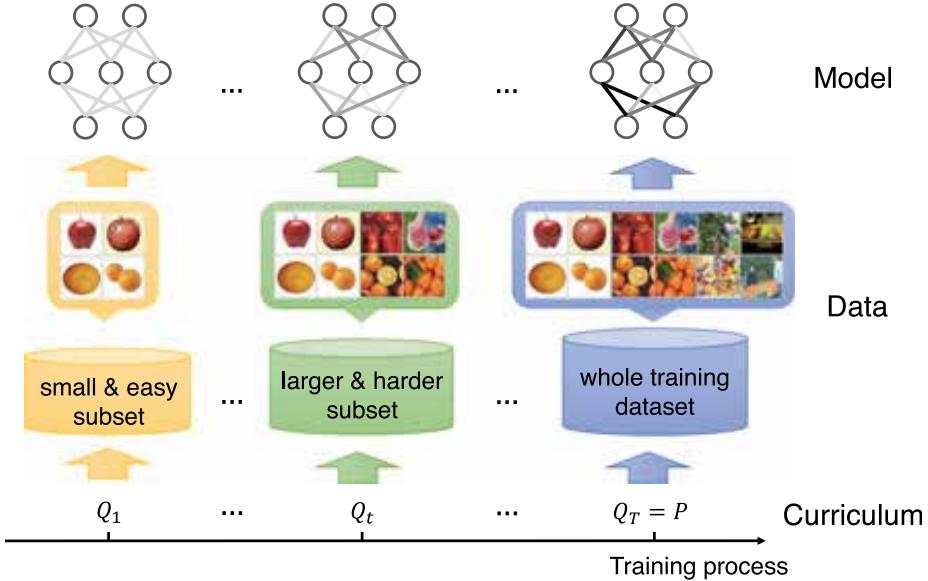


Figura 2.7: Illustrazione del processo di Curriculum Learning. Inizialmente, il modello viene addestrato su un sottoinsieme piccolo e semplice di dati, quindi su sottoinsiemi sempre più grandi e complessi, fino ad arrivare all’intero set di dati di addestramento. Questo approccio graduale aiuta il modello a imparare in modo più efficace e robusto. Immagine presa da Bengio, Jérôme Louradour, Collobert et al. [5].

principio: la conoscenza di nuove astrazioni viene costruita su nozioni acquisite in precedenza. Questo processo guidato permette agli esseri umani di aumentare significativamente la velocità di apprendimento e di acquisire competenze in modo più efficace.

In seguito viene riportata la formalizzazione di Curriculum Learning proposta in Bengio, Jérôme Louradour, Collobert et al. [5].

Sia z una variabile casuale che rappresenta un caso di training per l’apprendimento. Sia $P(z)$ il comportamento obiettivo che si vuole l’agente impari. Sia $0 \leq W_\lambda(z) \leq 1$ il peso applicato all’esempio z al passo λ nella sequenza di training secondo curriculum, con $0 \leq \lambda \leq 1$, e $W_1(z) = 1$. Al passo λ abbiamo

$$Q_\lambda(z) \propto W_\lambda(z)P(z) \quad \forall z$$

tale che

$$\int Q_\lambda(z)dz = 1.$$

Abbiamo quindi

$$Q_1(z) = P(z) \quad \forall z.$$

Consideriamo una sequenza monotonicamente crescente di valori di λ , a partire da $\lambda = 0$ e terminando a $\lambda = 1$.

Chiamiamo la corrispondente sequenza di distribuzioni Q_λ un curriculum se l’entropia di queste distribuzioni aumenta

$$H(Q_\lambda) < H(Q_{\lambda+\epsilon}) \quad \forall \epsilon > 0$$

e $W_\lambda(z)$ è monotonicamente crescente in λ , i.e.,

$$W_{\lambda+\epsilon}(z) \geq W_\lambda(z) \quad \forall z, \forall \epsilon > 0.$$

Questa definizione descrive il Curriculum Learning come una successione di casi di test in cui l'entropia del sistema aumenta, ovvero l'incertezza e la variabilità nelle situazioni di test diventano sempre maggiori. In altre parole, il modello inizia con esempi semplici e prevedibili e, gradualmente, affronta esempi sempre più complessi.

Inoltre, il peso associato alla distribuzione finale deve mantenere un valore positivo crescente. Questo significa che la situazione di addestramento deve tendere a diventare sempre più simile alla situazione obiettivo: man mano che l'addestramento procede, gli esempi proposti al modello devono avvicinarsi sempre più alle condizioni reali che il modello dovrà affrontare nell'applicazione pratica.

Empiricamente, il Curriculum Learning ha dimostrato di essere particolarmente efficace nell'ambito degli algoritmi di Apprendimento per Rinforzo. Questa strategia sembra infatti permettere agli agenti di apprendere più rapidamente e in modo più stabile. Come presentato in Bengio, Jérôme Louradour, Collobert et al. [5], l'applicazione di questa tecnica a diversi problemi classici ha evidenziato miglioramenti significativi nel tempo di convergenza degli agenti a un comportamento ottimale.

2.5 SUMO

SUMO, "Simulation of Urban MObility", è un software di simulazione del traffico open source progettato per gestire grandi reti in modo microscopico: ogni veicolo è modellato singolarmente, ha il proprio percorso e si muove individualmente lungo i tracciati generati. Le simulazioni sono largamente customizzabili e vengono fornite varie metriche di valutazione, quali il tempo di attesa dei veicoli, la loro velocità media o il numero di veicoli fermi per secondo. Il software include anche una API basata su socket chiamata TraCI (Traffic Control Interface), che consente l'interazione in tempo reale tra la simulazione e un agente esterno. Inoltre, offre un editor visivo (NetEdit) che facilita la creazione degli elementi statici del modello, come strade, corsie, intersezioni e giunzioni stradali[8].

In questo elaborato, SUMO è stato utilizzato per la realizzazione dell'ambiente di simulazione, per la gestione del flusso di veicoli e per l'acquisizione delle metriche di valutazione durante la fase di testing.

2.5.1 SUMO-RL

SUMO-RL è una progetto open source che fornisce un'interfaccia per istanziare ambienti di Apprendimento per Rinforzo specifici per problemi di controllo dei semafori, basandosi su SUMO e TraCI. Gli ambienti generati sono compatibili con gymnasium.Env e supportano le più comuni librerie di algoritmi di Apprendimento per Rinforzo, come stable baselines e RLLib, permettendo di modificare facilmente sia l'insieme degli stati che le funzioni di reward adottate[9].

In questo elaborato SUMO-RL è stato usato per implementare algoritmi di Apprendimento per Rinforzo tramite Stable Baselines 3.

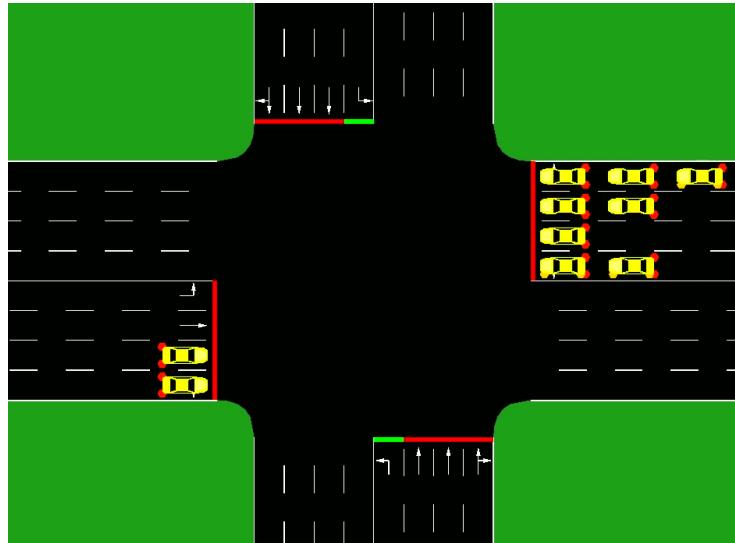


Figura 2.8: Rappresentazione di un agente a ciclo fisso che gestisce il controllo del traffico a un incrocio. In questo scenario, l'agente consente una svolta che nessun veicolo deve effettuare (indicata dal tratto verde sulla linea di stop), creando una situazione di congestione che potrebbe essere evitata con una scelta più opportunistica.

2.6 Agente a ciclo fisso

Per agente a ciclo fisso si intende un agente che esegue ciclicamente e costantemente tutte le azioni possibili, una dopo l'altra. Questo tipo di agente non mostra un comportamento adattivo o intelligente e rappresenta bene la gestione classica del controllo del traffico tramite l'utilizzo di semafori.

Il problema principale di questo tipo di agente è che non si adatta alla realtà del traffico, il che può portare a perdite di tempo e alla formazione di lunghe code, come illustrato nella Figura 2.8. In questo caso, tutto il flusso veicolare è interrotto per permettere una svolta che nessun veicolo deve effettuare.

L'idea di utilizzare agenti intelligenti per questo tipo di problema è proprio quella di sviluppare agenti capaci di comportarsi in modo opportunistico rispetto alle condizioni del traffico, garantendo sempre un'azione efficace e ottimizzata.

3 RL per il controllo delle luci semaforiche

Le tecniche di Apprendimento per Rinforzo sono ideali per affrontare il problema del controllo del traffico tramite luci semaforiche.

L’agente opera in un ambiente che rappresenta un incrocio, agendo a intervalli di tempo discreti t . In ogni passo, l’agente osserva lo stato s_t dell’incrocio e sceglie un’azione a_t , che corrisponde a una delle possibili configurazioni del semaforo. In risposta a questa azione, l’agente riceve una ricompensa r_t , che utilizza per valutare l’efficacia della sua decisione. Attraverso questo processo, l’agente impara a ottimizzare il controllo semaforico, riducendo al minimo i tempi di attesa e migliorando il flusso del traffico.

Questo capitolo dell’elaborato è dedicato alla descrizione della realizzazione dell’ambiente, dell’agente e della loro interazione.

3.1 Modello dell’ambiente

L’ambiente di studio è un’intersezione stradale a quattro corsie per senso di marcia, di cui due dedicate unicamente all’andare dritto, una alla svolta a sinistra e una sia all’andare dritto sia alla svolta a destra. In Figura 3.1 è mostrato l’ambiente creato con SUMO per simulare questo incrocio, basato sul modello di riferimento descritto in Alegre [9].

La scelta dell’incrocio è motivata dalla ricerca di un ambiente di test che sia sufficientemente realistico e complesso da richiedere una gestione accurata delle luci semaforiche. Questo incrocio, infatti, include manovre complesse come la svolta a sinistra, che blocca le restanti tre corsie del verso opposto, necessitando così di una pianificazione attenta.

3.1.1 MDP

L’ambiente è associato a un MDP implementato utilizzando la libreria SUMO-RL e strutturato nel seguente modo:

- Ogni stato è rappresentato da un vettore del tipo: $[phase_one_hot, min_green, lane_1_density, \dots, lane_n_density, lane_1_queue, \dots, lane_n_queue]$, dove:
 - $phase_one_hot$ descrive la luce verde attiva tramite codifica one-hot.
 - min_green è un valore booleano che indica se sono trascorsi almeno min_green secondi dall’ultimo cambio di fase (di default, 5 secondi).

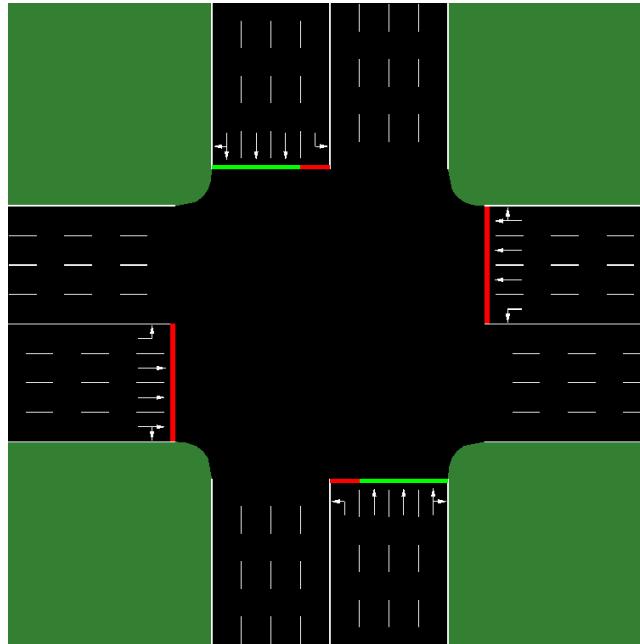


Figura 3.1: Incrocio stradale ricreato all'interno di SUMO

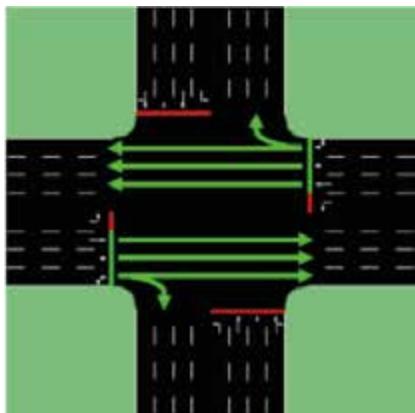
- $lane_i_density$ è il numero di veicoli presenti nella i-esima corsia diviso per la capacità massima della corsia.
- $lane_i_queue$ è il numero di veicoli in coda nella i-esima corsia diviso per la capacità massima della corsia.
- Le azioni possibili sono 4 e sono rappresentate in Figura 3.2
- La funzione di ricompensa scelta è Differential Waiting Time, definita come $r_t = twt_t - twt_{t+1}$, dove

$$twt_t = \sum_{i=1}^n wti_{i,t} \quad (3.1)$$

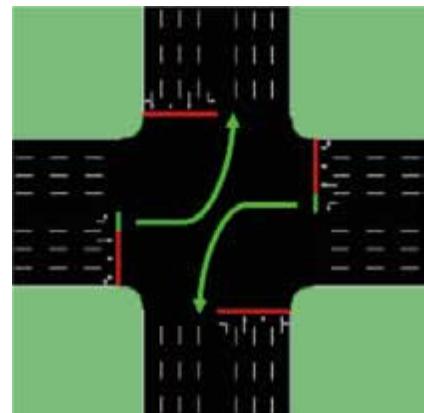
twt_t rappresenta il tempo totale di attesa al tempo t (Total Waiting Time), ossia il tempo cumulativo che le macchine passano in coda. L'ambiente genera una reward negativa se l'azione scelta dall'agente peggiora la situazione attuale. L'obiettivo dell'agente è quindi minimizzare la reward negativa.

3.1.2 Flusso di veicoli

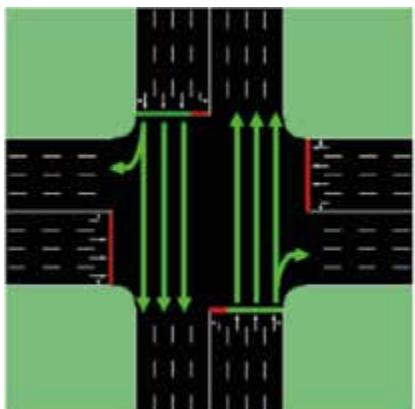
Nella simulazione, l'ambiente viene popolato da veicoli tramite un file che specifica il numero di veicoli e i percorsi che ognuno seguirà. La logica di generazione è gestita dalla seguente classe:



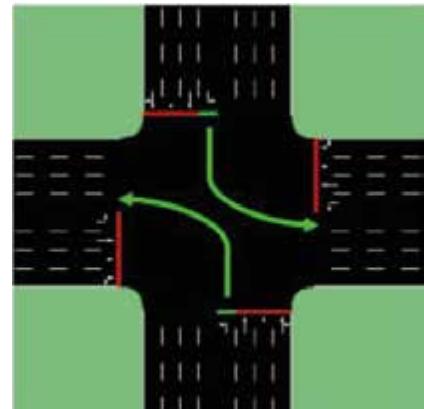
(a) Asse Est-Ovest senza svolta



(b) Asse Est-Ovest con svolta



(c) Asse Nord-Sud senza svolta



(d) Asse Nord-Sud con svolta

Figura 3.2: Rappresentazione dei percorsi possibili che un veicolo può seguire all'interno dell'ambiente, ciascuno corrispondente a una specifica azione. Immagine presa da Vidali, Crociani, Vizzari et al. [10].

```

class TrafficGenerator:
    def __init__(self, n_cars_generated, seconds, type, route_path):
        self._n_cars_generated = n_cars_generated
        self._seconds = seconds
        self._type = type # balanced, NS or EW
        self._route_path = route_path

    def generate_routefile(self, seed, n, function, multi_staged=False):
        # To obtain replayability
        np.random.seed(seed)
        # Max seconds for stop
        max_stop = seconds_stop

        if function == "uniform":
            # Flow and stops generation
            car_gen_steps = uniform(self._seconds, self._n_cars_generated)
            stop_time = uniform_array(self._n_cars_generated, max_stop)
        elif function == "weibull":
            car_gen_steps = weibull(self._seconds, self._n_cars_generated)
            stop_time = weibull_array(self._n_cars_generated, max_stop, points,
                                      min, max)
        elif function == "bimodal":
            car_gen_steps = bimodal(self._seconds, self._n_cars_generated)
            stop_time = bimodal(max_stop, self._n_cars_generated, sorted=False)

        if multi_staged:
            generate_xml_multi_stage(car_gen_steps, self._route_path +
                                      "routes.rou" + str(n) + ".xml", stop_time)
        else:
            generate_xml(self._type, car_gen_steps, self._route_path +
                        "routes.rou" + str(n) + ".xml", stop_time)

```

Ogni generazione della simulazione richiede la specifica di alcuni parametri: la durata della simulazione in secondi, il numero di veicoli da generare, il tipo di traffico da simulare e il nome del file di output. La generazione è basata su un seme, rendendo la simulazione riproducibile, e produce un flusso di traffico secondo una distribuzione di probabilità arbitraria. A ciascun veicolo viene associato un valore di fermo, espresso in secondi, che indica quanto tempo il veicolo dovrà restare fermo prima di uscire dalla simulazione. Questo valore di fermo è generato in modo simile al flusso di traffico, con valori più alti in situazioni di traffico intenso, simulando l'impatto del traffico non solo sull'incrocio ma anche sulle aree circostanti, impedendo ai veicoli di uscire liberamente dalla simulazione; il risultato finale viene raffigurato in Figura 3.3. Alla fine del processo, viene generato un file XML contenente i dati necessari, con ogni veicolo associato al suo tempo di fermo.

Le funzioni di generazione possono operare in due modalità: con o senza multi-staged. Un ambiente di traffico multi-staged prevede che il flusso oscilli tra tre stati in modo randomico: stato bilanciato, stato NS e stato WE. Nello stato bilanciato, il traffico si distribuisce equamente tra l'asse Nord-Sud e l'asse Est-Ovest; negli altri due stati, il traffico predilige una delle due direzioni con una probabilità distribuita secondo una gaussiana con valore medio 0.7 (70% del traffico) e deviazione standard 0.1. Nel caso senza multistage, la generazione del traffico dipenderà dal tipo di stato specificato come argomento.

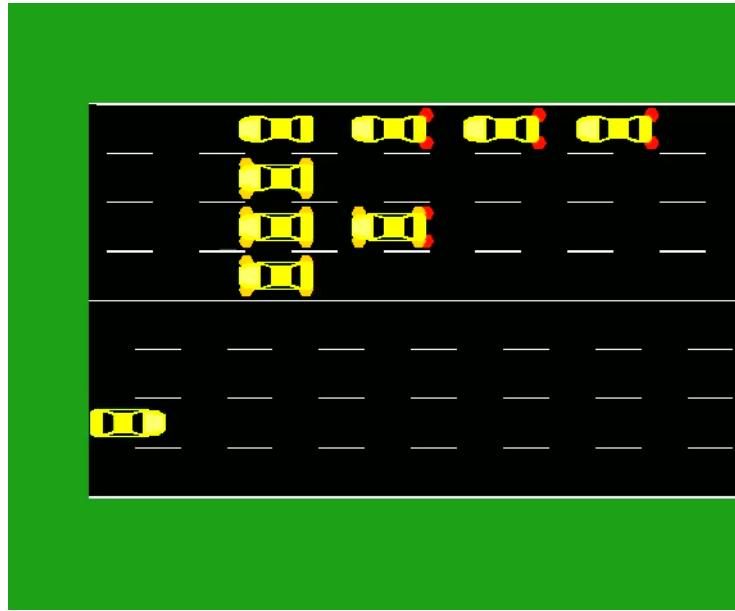


Figura 3.3: Illustrazione del periodo di attesa che i veicoli devono osservare prima di uscire dalla simulazione, al fine di rappresentare il traffico esterno all'incrocio in cui si svolge la simulazione.

3.2 Integrazione con Stable Baseline

L’agente è stato dotato di un algoritmo di Apprendimento per Rinforzo utilizzando la libreria Stable Baselines 3 (come approfondito in 2.3). L’algoritmo scelto è stato il Deep Q-Learning (DQN) (spiegato in 2.2.1).

Come riportato nella documentazione della libreria, l’implementazione fornita di DQN si basa su Q-Learning e incorpora diversi accorgimenti per stabilizzare l’apprendimento utilizzando una rete neurale profonda. Questi accorgimenti includono:

- Utilizzo di un replay buffer.
- Utilizzo di una rete separata per la funzione obiettivo.
- Utilizzo della tecnica di *gradient clipping*, che previene il problema dei gradienti esplosivi durante la retropropagazione ponendo un limite superiore alla loro norma.

L’implementazione adottata in questo elaborato mantiene gli iperparametri utilizzati nella prima realizzazione di DQN presentata in Mnih, Kavukcuoglu, Silver et al. [4]. Questo approccio permette un’istanziazione e un utilizzo del modello molto rapidi e agili, evidenziando una delle principali caratteristiche di Stable Baselines: fornire modelli di apprendimento pronti all’uso.

```
from stable_baselines3 import DQN
import sumo_rl
```

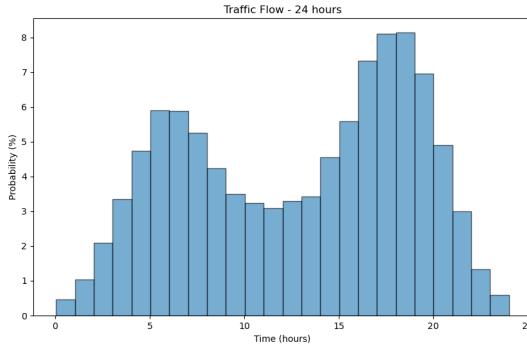


Figura 3.4: Probabilità di generazione di un veicolo durante le 24 ore

```
# Creazione di un ambiente usando SUMO-RL
env = gym.make('sumo-rl-v0',
               net_file='environment.net.xml',
               route_file="routes.rou.xml",
               use_gui=False,
               num_seconds=max_seconds)

# Istanza di un modello usando stable_baselines3
model = DQN(
    policy="MlpPolicy",
    env=env,
    verbose=0)
```

3.3 Processo di addestramento

L’obiettivo del processo di addestramento è ottenere un modello capace di gestire in modo ottimale una situazione di traffico verosimile nell’arco delle 24 ore. Il flusso di traffico è rappresentato in Figura 3.4.

Questa configurazione, ottenuta tramite composizione di due Gaussiane, riflette una tipica giornata in un centro cittadino: si osserva un picco di traffico durante le ore mattutine, dovuto all’afflusso dei lavoratori, seguito da una fase di relativa calma nel primo pomeriggio. Successivamente, il flusso di traffico aumenta nuovamente in serata, correlato alle attività ricreative.

Il singolo processo di addestramento segue i seguenti passaggi:

- Istanziazione del modello
- Generazione del file di flusso (come spiegato in 3.1.2)
- Istanziazione dell’ambiente di simulazione tramite SUMO-RL
- Avvio della simulazione
- Raccolta delle ricompense (reward) ottenute dal modello

Questo ciclo viene ripetuto fino a quando il modello non converge verso un comportamento ritenuto ottimale. Il criterio di convergenza utilizzato per l'addestramento del modello in questo elaborato è il seguente:

```

def isFinished(monitor_path):
    # Lettura dei dati
    data = pd.read_csv(monitor_path, delimiter=",", header=None)
    r_values = data[0]
    total_data_points = len(r_values)

    # Numero minimo di campionamenti
    if total_data_points < min_points:
        return False

    y = np.exp(np.linspace(0, linspace_max, points) / scale_factor)
    y_normalized = y / np.max(y)

    gamma = 1 - y_normalized[total_data_points-1]

    if gamma > 0:
        last_gamma_values = r_values[-math.ceil(gamma*total_data_points):]
        mean_last_gamma = np.mean(last_gamma_values)
        mean_last_gamma_modded = mean_last_gamma * gamma
        if math.ceil(mean_last_gamma_modded) == 0:
            return True

    return False

```

Questa funzione permette di considerare le ultime ricompense ottenute, applicarvi una trasformazione (i dati vengono moltiplicati per una variabile γ) e calcolarne il valore medio. Se questo valore medio è compreso fra 0 e 1, si ritiene che il modello sia giunto a convergenza.

La trasformazione e la dimensione della finestra di valutazione sono entrambe legate al valore γ , che corrisponde a una esponenziale normalizzata. Con l'aumentare delle iterazioni, il valore di γ decresce, riducendo sia la finestra di valutazione che i valori ottenuti dopo la trasformazione.

Questo metodo permette di interrompere il processo di training prima che inizi a durare troppo a lungo, giungendo a un compromesso tra risultati e costo. L'allenamento infatti è un processo che consuma molte risorse e, se esposto senza filtri a situazioni rumorose, potrebbe continuare inutilmente per un lungo periodo.

3.3.1 Curriculum

Il processo di training tramite curriculum prevede la selezione di una serie di simulazioni diverse, ciascuna con condizioni di generazione differenti e di difficoltà crescente. Il modello viene addestrato su queste simulazioni in modo sequenziale, iniziando da quella più semplice e passando alla successiva una volta raggiunta la convergenza.

La scelta dei vari step è la parte più laboriosa, poiché è necessario che rispettino i principi del curriculum learning (come esposto in 2.4): l'entropia del siste-

ma deve aumentare da una simulazione all'altra, avvicinandosi progressivamente all'obiettivo finale.

I parametri scelti sui quali basarsi per definire una metrica di "difficoltà" del singolo passaggio sono:

- Numero di veicoli generati all'interno della simulazione
- Distribuzione di probabilità con cui vengono generati i veicoli
- Lunghezza della simulazione
- Generazione multi stage o non multi stage

Il numero di veicoli, insieme alla lunghezza della simulazione, può essere riassunto in un parametro relativo alla densità di veicoli nel tempo. Per quanto riguarda la distribuzione di probabilità della generazione dei veicoli, sono state proposte due distribuzioni: uniforme e di Weibull (presentate in Appendice A). La densità e la distribuzione dei veicoli influenzano direttamente la difficoltà delle simulazioni, aumentando la complessità del traffico che l'agente deve gestire.

La generazione multi stage consente di esporre l'agente, all'interno di una singola simulazione, a condizioni di traffico diverse, permettendogli di adattarsi a variazioni nel flusso di traffico. Al contrario, nella generazione non multi stage, l'agente converge prima su una situazione di traffico specifica e successivamente sulle altre, seguendo un ordine sequenziale.

4 Sperimentazione e risultati

L’obiettivo della sperimentazione è quello di utilizzare algoritmi implementati tramite Stable Baselines 3 e applicarli al problema del controllo del traffico, in modo che l’agente possa gestire le fasi semaforiche puntando a un’ottimizzazione delle code di veicoli. L’algoritmo scelto è stato DQN poiché rappresenta uno dei più importanti algoritmi nella storia del Deep Reinforcement Learning, essendo una delle prime implementazioni di reti neurali a supporto di algoritmi di Apprendimento per Rinforzo. Il processo che ha portato ai risultati finali si è evoluto in diversi passaggi, analizzati in seguito.

4.1 Primi passi

Il primo passaggio è consistito nell’addestrare il modello in un ambiente di difficoltà intermedia per analizzare il tempo di convergenza. Considerando che DQN è un algoritmo progettato per mantenere stabilità durante l’apprendimento, vi era la possibilità che la convergenza richiedesse un significativo dispendio di risorse. L’ambiente di simulazione era caratterizzato dai seguenti parametri:

- Distribuzione di Weibull con picco costante
- Flusso di 5000 veicoli
- 8 ore simulate all’interno di una singola iterazione
- Flusso di traffico bilanciato su ciascun asse dell’incrocio
- 75% delle macchine proseguono senza svolta, 25% svoltano a sinistra

e in questo primo esperimento la convergenza ha richiesto circa 49 ore; il valore della ricompensa lungo il periodo di addestramento viene visualizzato in Figura 4.1.

Tale risultato ha fatto presagire che il tempo di convergenza in un ambiente finale più complesso avrebbe richiesto un impiego di risorse elevato. Si è quindi deciso di applicare una tecnica di addestramento che potesse aumentare la velocità di convergenza; la scelta è ricaduta sul Curriculum Learning che, secondo quanto esposto in Bengio, Jérôme Louradour, Collobert et al. [5], avrebbe potuto rendere la convergenza più rapida.

In Figura 4.2 sono riportati invece i risultati ottenuti dal modello all’interno dell’ambiente di test. Si osserva che i dati ottenuti presentano un elevato livello di rumore e che le prestazioni del modello sono variabili, in particolare nei casi di traffico limitato. Questo risultato è coerente con il fatto che il modello sia

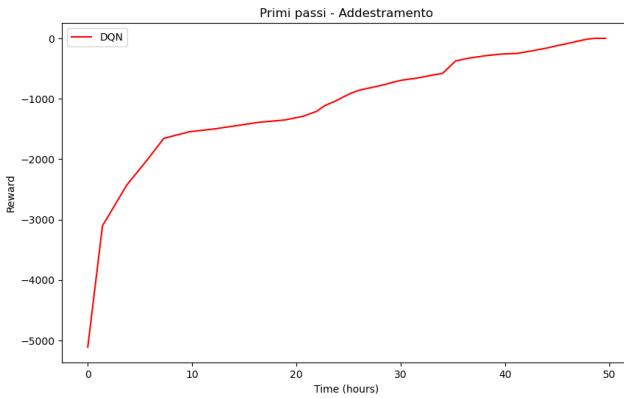


Figura 4.1: Andamento della ricompensa durante il periodo di addestramento. La convergenza è stata raggiunta dopo circa 49 ore. I valori della ricompensa, registrati utilizzando TensorBoard, sono stati campionati ogni 22400 step.

stato allenato in una sola condizione di densità di traffico, e quindi non abbia sviluppato flessibilità nel gestire densità variabili. Un vantaggio nell'utilizzare un approccio come il Curriculum Learning è anche quello di esporre l'agente a condizioni di densità diverse, rendendolo più adatto a situazioni generali e meno specializzato.

4.2 Curriculum applicato

Per ottenere un curriculum efficace, è stato necessario individuare simulazioni meno impegnative rispetto a quella finale, ma che potessero progressivamente avvicinarsi ad essa. Sono state individuate due distribuzioni per la generazione del flusso di veicoli. La più semplice e lontana dalla distribuzione finale è risultata essere la distribuzione uniforme, ideale per i primi step del curriculum. La seconda distribuzione individuata è stata la distribuzione di Weibull, rappresentativa dei picchi presenti nella simulazione obiettivo.

In seguito, è stato studiato il rapporto tra tempo di simulazione e numero di veicoli generati, in modo da individuare condizioni di basso e alto traffico: situazioni con minore congestione risultano più semplici per il modello e quindi più adatte ai primi step del curriculum.

Il primo approccio al curriculum ha quindi preso la seguente configurazione:

- Distribuzione uniforme, basso traffico, 1 ora di simulazione
- Distribuzione di Weibull, basso traffico, 3 ore di simulazione
- Distribuzione di Weibull, alto traffico, 5 ore di simulazione

mantenendo il resto dei parametri uguali a quanto fatto nello step precedente.

In aggiunta, ogni fase della simulazione è stata segmentata in tre sottofasi: traffico bilanciato, traffico Est-Ovest (EW) e traffico Nord-Sud (NS). Questa

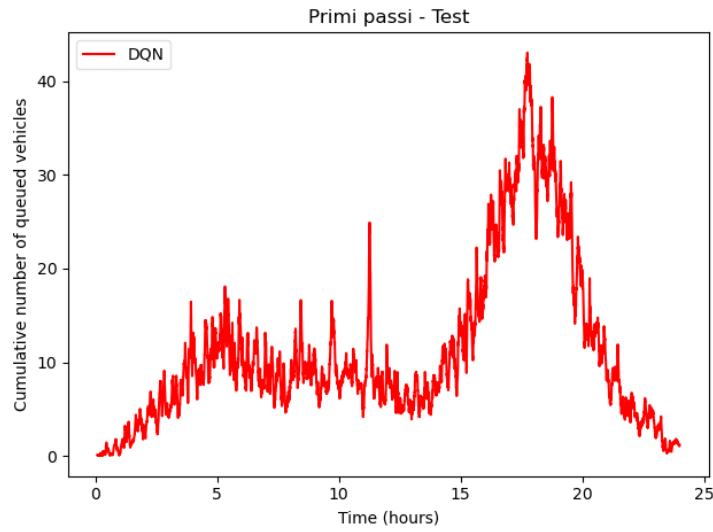


Figura 4.2: Risultati della fase di test utilizzando il modello ottenuto dopo la prima fase di addestramento. Il modello mostra un elevato livello di rumore e prestazioni variabili, soprattutto in condizioni di traffico limitato. I dati sono stati sottoposti a una media mobile con una finestra di 50 elementi per visualizzare meglio l'andamento generale.

configurazione ha permesso al modello di confrontarsi con condizioni di traffico sia bilanciate che sbilanciate sui vari assi, consentendogli di imparare un approccio più robusto e adattabile alle diverse situazioni.

Il curriculum completo ha quindi preso la seguente configurazione:

- Distribuzione uniforme, basso traffico, 1 ora di simulazione
 - Traffico bilanciato
 - Traffico Nord-Sud
 - Traffico Est-Ovest
- Distribuzione di Weibull, basso traffico, 3 ore di simulazione
 - Traffico bilanciato
 - Traffico Nord-Sud
 - Traffico Est-Ovest
- Distribuzione di Weibull, alto traffico, 5 ore di simulazione
 - Traffico bilanciato
 - Traffico Nord-Sud
 - Traffico Est-Ovest

Il modello ha completato l'addestramento in un tempo significativamente inferiore rispetto alle precedenti stime: 158 minuti, ovvero 2 ore e 38 minuti.

L'analisi della ricompensa ottenuta durante le iterazioni, riportata in Figura 4.3, mostra che solo il primo step di training è stato realmente utile per il modello. Questo è giustificato dal fatto che i parametri individuati non sono cambiati abbastanza da mettere alla prova il modello, che già nel primo step ha imparato un comportamento rivelatosi efficace. In Figura 4.4 si osserva come effettivamente gli ultimi step la ricompensa sia costante sul valore 0.

Di fronte a questo risultato, in cui il tempo di addestramento è diminuito significativamente come previsto, l'attenzione si è spostata sul trovare step di test più significativi. L'obiettivo è incentivare l'apprendimento nei passaggi successivi, migliorando ulteriormente le prestazioni del modello e valutando se l'applicazione del Curriculum Learning ai problemi di regolazione del traffico tramite semafori possa comportare anche un miglioramento delle prestazioni complessive.

4.3 Ottimizzazione

Il processo di ottimizzazione è stato eseguito per individuare le configurazioni di addestramento più favorevoli per l'agente, garantendo un apprendimento costante lungo tutto il percorso. L'obiettivo è che questo approccio conduca a un modello con un apprendimento più profondo e, di conseguenza, una maggiore efficienza.

Per proporre all'agente condizioni più impegnative, sono state introdotte le seguenti modifiche:

- **Configurazione multi-staged:** Questa configurazione introduce un cambiamento dinamico delle condizioni di traffico all'interno di una singola simulazione, come approfondito nella sezione 3.1.2.
- **Introduzione dell'aleatorietà nella distribuzione di Weibull:** Il picco della distribuzione varia da una simulazione all'altra, garantendo una maggiore variabilità nelle condizioni sperimentate dall'agente.
- **Gestione probabilistica del comportamento dei veicoli:** Nella generazione dei veicoli, la probabilità di avanzare o svoltare non è più costante. Questa probabilità è ora gestita da una distribuzione gaussiana, rendendo il comportamento dei veicoli più imprevedibile e realistico.

Dopo numerosi tentativi per individuare le condizioni ottimali, si è giunti alla seguente configurazione, che rispecchia il curriculum finale adottato:

- Distribuzione uniforme, 400 veicoli generati, 1 ora di simulazione
 - Traffico bilanciato
 - Traffico Nord-Sud
 - Traffico Est-Ovest
- Distribuzione uniforme, 1500 veicoli generati, 1 ora di simulazione, multi-staged
- Distribuzione Weibull, 2250 veicoli generati, 1 ora di simulazione, multi-staged

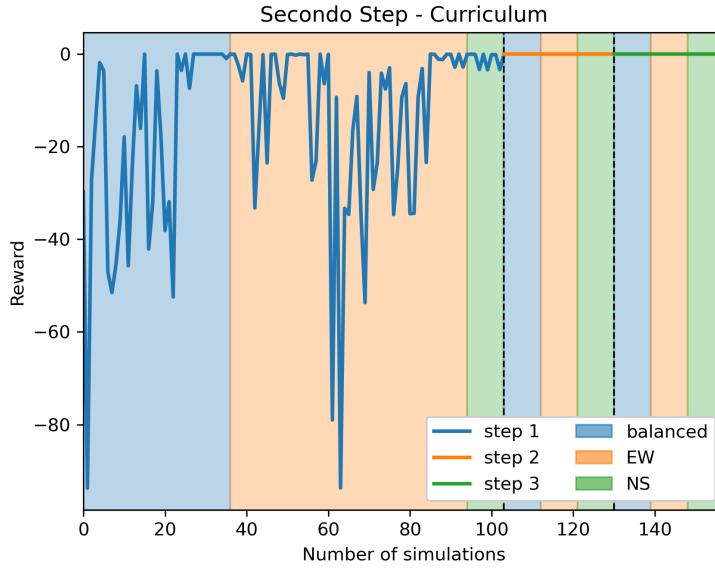


Figura 4.3: Grafico della ricompensa ottenuta per il secondo passaggio. Le linee rappresentano i vari step con colori distinti. Ogni step è separato da una linea tratteggiata verticale. I sottostep sono evidenziati con colori di sfondo differenti, che si ripetono per ciascuno step.

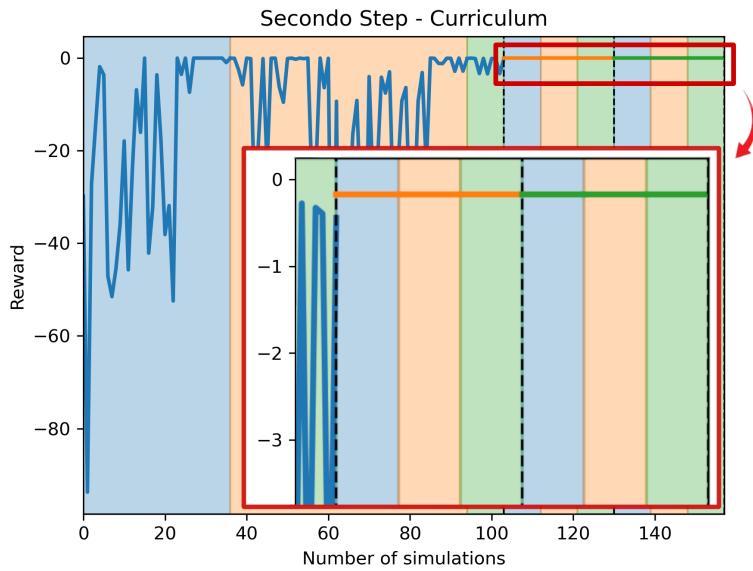


Figura 4.4: Zoom sul grafico per visualizzare gli ultimi due step finali tramite un nuovo plot con un range dell'asse delle ordinate inferiore. Si può osservare chiaramente come la reward sia nulla. Questo indica che l'agente non sta commettendo errori, segnalando una fase di non apprendimento.

- Distribuzione Weibull, 6000 veicoli generati, 3 ore di simulazione, multi-staged
- Distribuzione Weibull, 8000 veicoli generati, 5 ore di simulazione, multi-staged
- Distribuzione Weibull, 10000 veicoli generati, 7 ore di simulazione, multi-staged

Questa configurazione rispetta le ipotesi di un processo curriculare: l'entropia del sistema aumenta gradualmente, partendo da un comportamento semplice, ossia la gestione di una situazione di traffico a basso volume con condizioni uniformi, e progredendo verso un comportamento più complesso, avvicinandosi alla funzione obiettivo illustrata in Figura 3.4.

4.4 Risultati

4.4.1 Addestramento

I processi di addestramento eseguiti sono stati due: uno ha visto un agente DQN addestrato sul percorso curriculare esposto nella sezione 4.3 e l'altro un agente DQN con uguali parametri addestrato solo sull'ultimo step del percorso curriculare, ossia:

- Distribuzione Weibull, 10000 veicoli generati, 7 ore di simulazione, multi-staged

Il primo dei due a convergere è stato l'agente sottoposto al percorso curriculare, dopo 3 ore e 35 minuti. In Figura 4.5 possiamo osservare l'andamento della ricompensa durante l'addestramento: risulta evidente come il modello impari molto durante i primi step, che vanno a creare la base della conoscenza su cui si basa il comportamento del modello. Dopo i primi tre step l'agente si trova invece a fare errori dal valore assoluto alto all'inizio dei nuovi step, che però riassorbe subito modulando le sue conoscenze pregresse sul nuovo ambiente e raggiungendo velocemente un comportamento ottimale. Il modello è quindi molto reattivo al cambiamento e capace di gestirlo con prontezza.

Nel grafico 4.6 sono mostrati singolarmente gli isolamenti dei vari step, permettendo di visualizzare i vari andamenti della funzione di ricompensa. Come suggerito dall'immagine completa, il primo step è quello in cui l'agente commette più errori e, di conseguenza, impara di più. Dal quarto step in poi, gli errori diventano sporadici e vengono rapidamente corretti. La ragione per cui la ricompensa negativa è significativamente più alta negli ultimi step rispetto ai primi è dovuta alla maggiore durata delle simulazioni e al maggior numero di veicoli generati. Questo porta a code di veicoli in attesa fisiologicamente più lunghe rispetto a simulazioni con un numero inferiore di veicoli e tempi di simulazione più brevi. Di conseguenza, le ricompense negative sono maggiori, poiché dipendono dal numero totale di veicoli in coda, come meglio trattato nella sezione 3.1.1.

L'agente non sottoposto al processo curriculare, invece, è stato fermato prima della convergenza per eccessivo consumo di risorse. In Figura 4.7 è riportato

l'andamento della reward del modello nelle prime 36 ore e 3 minuti. Si nota come durante l'esecuzione migliori effettivamente l'andamento della reward, sembrando raggiungere più stabilità, ma non ci sia ancora modo di prevedere una convergenza prossima del modello.

Questi risultati confermano quanto ci si aspettava: il Curriculum Learning rende effettivamente più veloce la convergenza del modello in modo molto sensibile, raggiungendo una diminuzione misurabile in ordini di grandezza: 12900 secondi contro 129780 secondi.

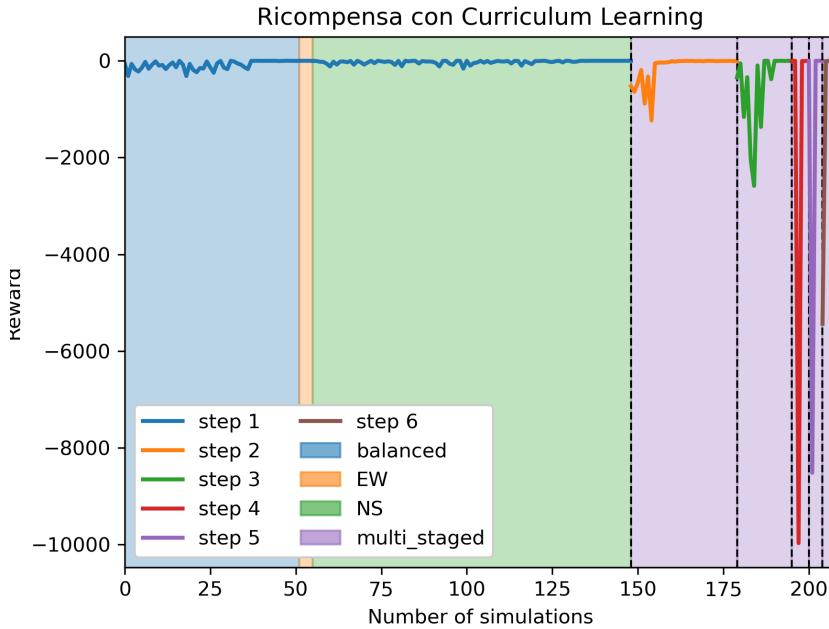


Figura 4.5: Grafico della ricompensa ottenuta durante il processo di addestramento dell'agente DQN sottoposto al processo curriculare descritto in 2.4.

È però da notare come il Curriculum Learning richieda una fase di studio impegnativa per ottenere una configurazione utile, che permetta all'agente di convergere. Molte sperimentazioni effettuate con altre configurazioni, infatti, sono risultate più instabili di quella ottenuta attualmente. Vengono riportati degli esempi in Figura 4.8 e Figura 4.9; il primo esempio rappresenta la funzione di reward per il curriculum:

- Distribuzione uniforme, 400 veicoli generati, 1 ora di simulazione
 - Traffico bilanciato
 - Traffico Nord-Sud
 - Traffico Est-Ovest
- Distribuzione uniforme, 1500 veicoli generati, 1 ora di simulazione, multi-staged

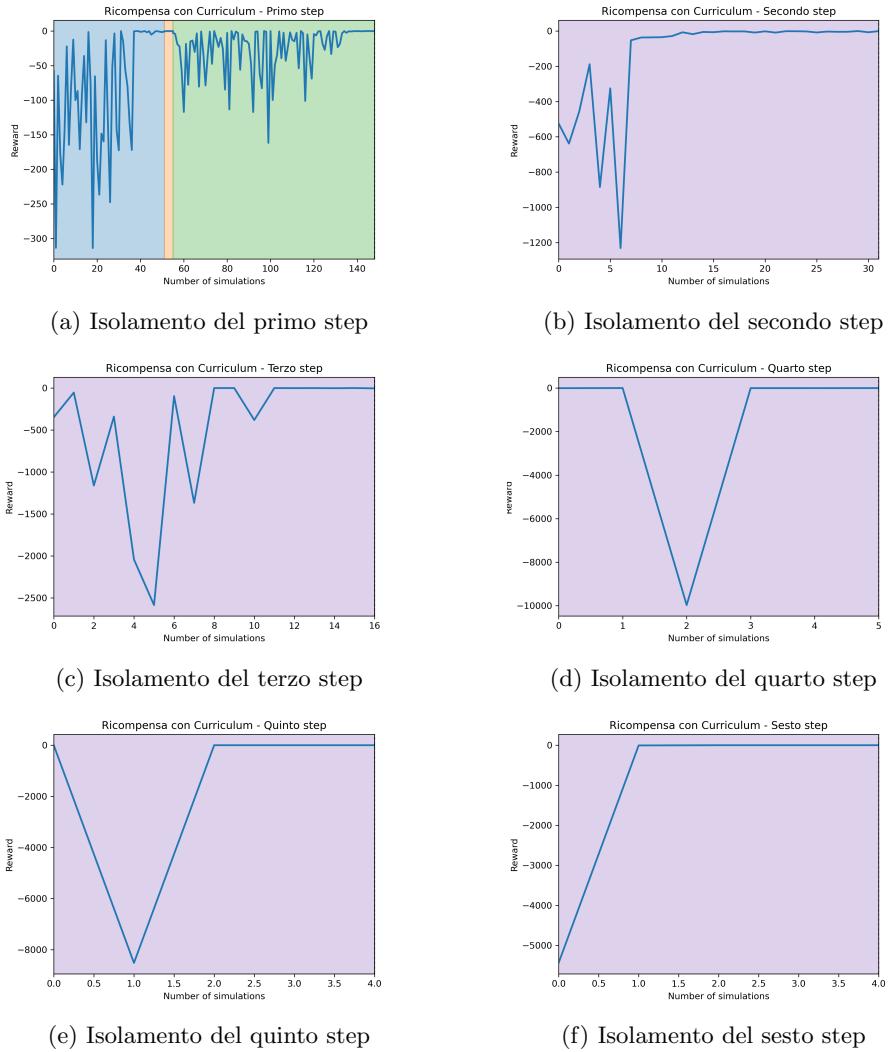


Figura 4.6: Isolamento dei singoli step che compongono l’addestramento, con rappresentazione della ricompensa ottenuta.

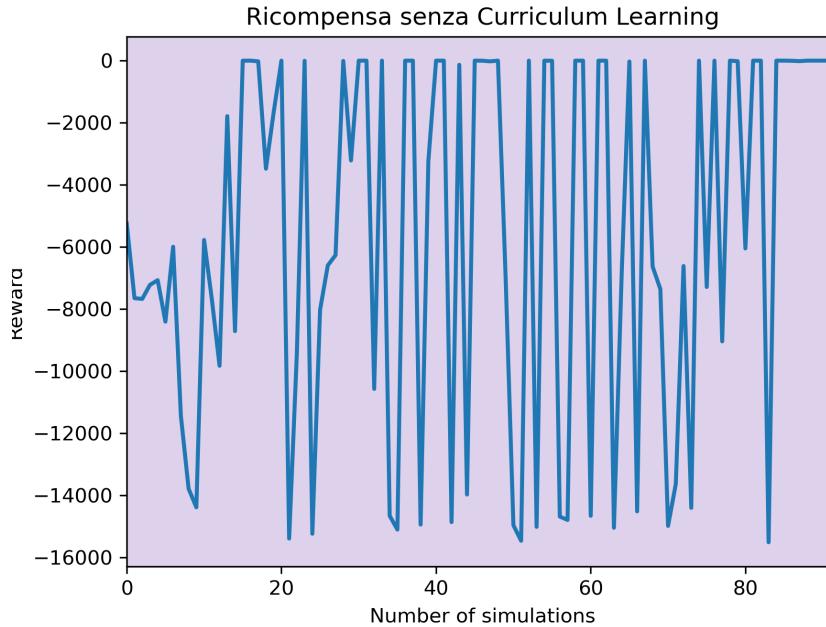


Figura 4.7: Grafico della ricompensa ottenuta durante il processo di addestramento dell’agente DQN addestramento unicamente su 4.4.1.

- Distribuzione Weibull, 2250 veicoli generati, 1 ora di simulazione, multi-staged
- Distribuzione Weibull, 6000 veicoli generati, 3 ore di simulazione, multi-staged
- Distribuzione Weibull, 10000 veicoli generati, 5 ore di simulazione, multi-staged
- Distribuzione Weibull, 14000 veicoli generati, 7 ore di simulazione, multi-staged

per un addestramento durato 22 ore e 40 minuti, il secondo esempio rappresenta la funzione di reward per il curriculum:

- Distribuzione uniforme, 400 veicoli generati, 1 ora di simulazione
 - Traffico bilanciato
 - Traffico Nord-Sud
 - Traffico Est-Ovest
- Distribuzione uniforme, 1500 veicoli generati, 1 ora di simulazione, multi-staged
- Distribuzione Weibull, 2250 veicoli generati, 1 ora di simulazione, multi-staged

- Distribuzione Weibull, 6000 veicoli generati, 3 ore di simulazione, multi-staged
- Distribuzione Weibull, 9000 veicoli generati, 5 ore di simulazione, multi-staged
- Distribuzione Weibull, 11000 veicoli generati, 7 ore di simulazione, multi-staged

per un addestramento durato 55 ore e 34 minuti. Risulta evidente come in entrambi i casi il modello sia arrivato a condizioni di instabilità durante l'addestramento che non hanno permesso la convergenza. Questa instabilità può essere giustificata dal fatto che i passaggi del percorso curriculare non sono abbastanza coerenti tra loro: i primi step non forniscono al modello gli strumenti necessari per gestire le situazioni successive.

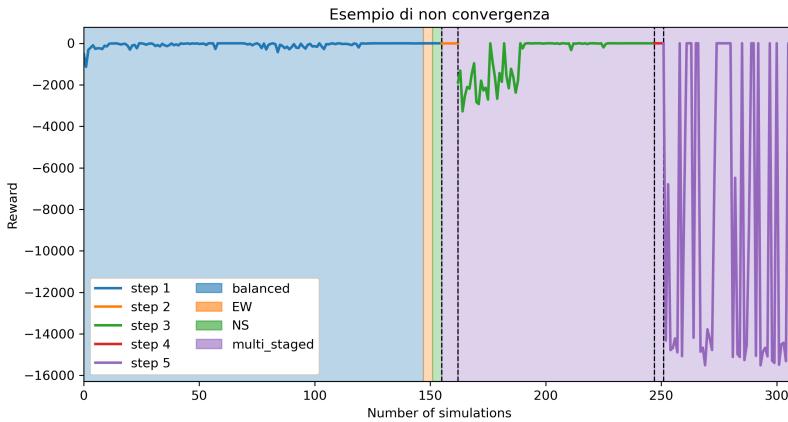


Figura 4.8: Esempio di simulazione in cui l'agente non riesce a raggiungere la convergenza a un comportamento ottimo a causa della complessità eccessiva della struttura del processo curriculare. Il processo in esame è descritto in 4.4.1.

4.4.2 Test e risultati

La fase di test ha messo a confronto i due modelli ottenuti durante la fase di addestramento: uno sottoposto a Curriculum Learning e uno non sottoposto a Curriculum Learning. La simulazione di test è definita dai seguenti parametri:

- Il numero di veicoli generati è pari a 20000
- Il flusso dei veicoli è generato tramite la distribuzione bimodale raffigurata in Figura 3.4
- Il flusso dei veicoli è variabile e suddiviso fra fasi di flusso bilanciato e fasi di flusso sbilanciato lungo l'asse Est-Ovest, come rappresentato in Figura 4.10.

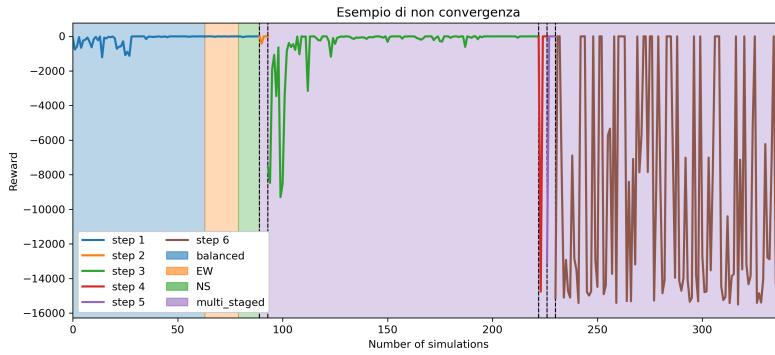


Figura 4.9: Esempio di simulazione in cui l’agente non riesce a raggiungere la convergenza a un comportamento ottimo a causa della complessità eccessiva della struttura del processo curriculare. Il processo in esame è descritto in 4.4.1.

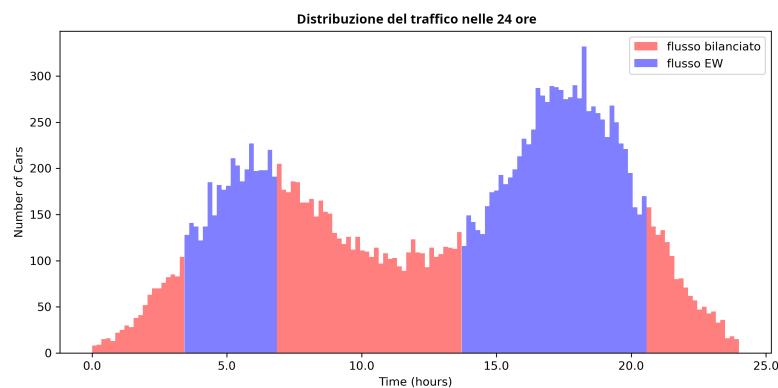


Figura 4.10: Distribuzione del traffico lungo le 24 ore; il traffico è diviso in fasi di flusso bilanciato e fasi di flusso sbilanciato lungo l’asse Est-Ovest, per simulare il movimento del traffico verso il centro cittadino.

In Figura 4.11 e in Figura 4.12 sono riportati i risultati dei singoli test, mentre in Figura 4.13 viene mostrato il risultato ottenuto durante la fase di test comparativa tra i due modelli di apprendimento. Entrambi i modelli dimostrano una capacità equivalente nella gestione del flusso, confermando le aspettative iniziali. Nello specifico, il processo curriculare ha permesso una convergenza più rapida verso un comportamento ottimale, raggiungendo la stessa efficienza di un agente addestrato con metodi tradizionali. Questo risultato evidenzia l'efficacia dell'addestramento curriculare nel migliorare i tempi di apprendimento senza compromettere le prestazioni finali.

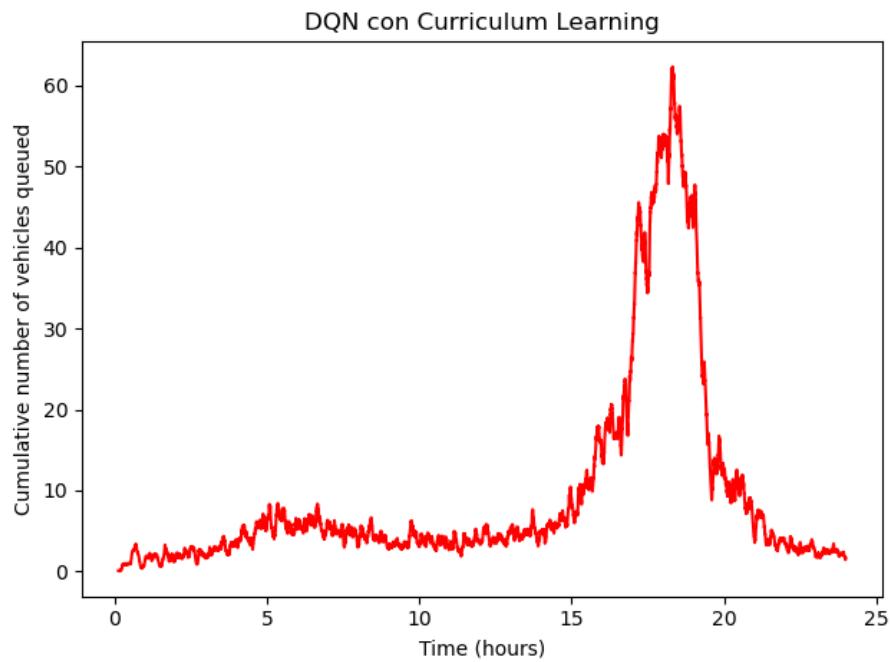


Figura 4.11: Rappresentazione del numero totale di veicoli in coda nel corso della simulazione utilizzando il modello addestrato tramite percorso curriculare. Si osserva che la distribuzione bimodale, mostrata in Figura 3.4, è rispettata. L'agente gestisce efficacemente il primo picco di traffico, ma mostra difficoltà durante il secondo picco. I dati sono stati sottoposti a una media mobile con finestra di 50, consentendo una visualizzazione più chiara dell'andamento complessivo.

Oltre ai due modelli menzionati, è stato testato nello stesso ambiente un agente privo di algoritmi di apprendimento. Questo agente si basa unicamente sulla ripetizione ciclica delle azioni possibili seguendo un ciclo fisso.

Il comportamento dell'agente a ciclo fisso risulta molto interessante: questo tipo di agente mostra infatti una variabilità estremamente elevata, arrivando in alcuni casi a eguagliare o superare le prestazioni di un agente intelligente. Questo comportamento è probabilmente dovuto alla natura aleatoria delle mosse dell'agente a ciclo fisso: se per caso l'agente agisce in modo coerente con la generazione del traffico, allora la gestione del flusso sarà efficace. Al contrario, se

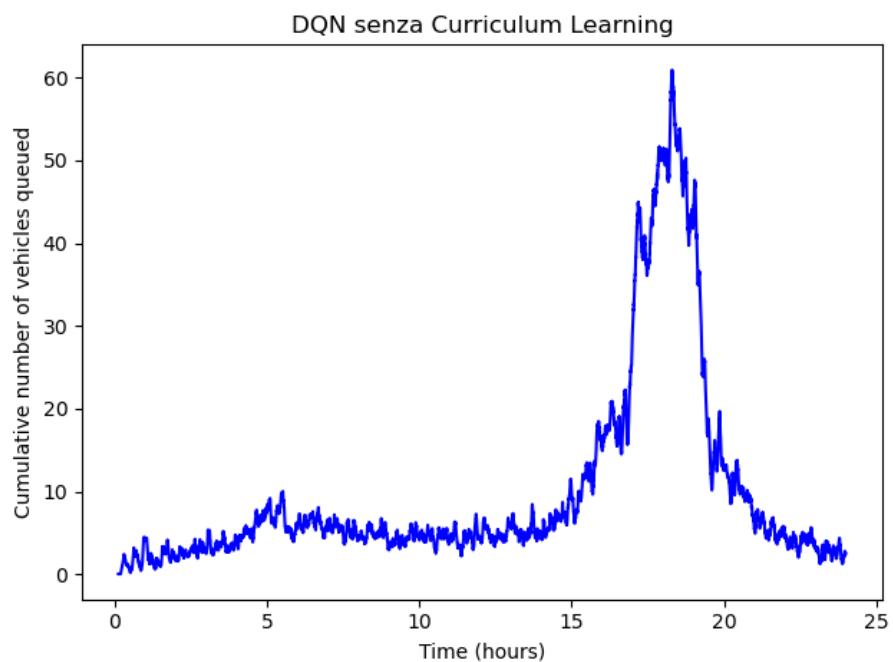


Figura 4.12: Rappresentazione del numero totale di veicoli in coda nel corso della simulazione utilizzando il modello addestrato tramite percorso classico. Si osserva che la distribuzione bimodale, mostrata in Figura 3.4, è rispettata. I dati sono stati sottoposti a una media mobile con finestra di 50, consentendo una visualizzazione più chiara dell'andamento complessivo.

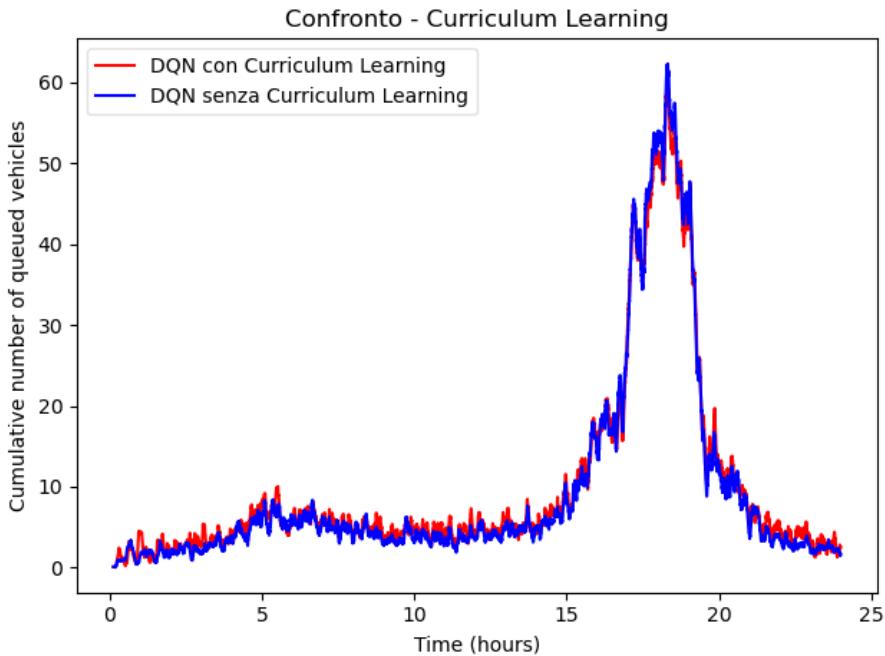


Figura 4.13: Confronto tra i modelli ottenuti dai due diversi percorsi di addestramento. I risultati mostrano chiaramente prestazioni molto simili.

le azioni dell'agente non sono allineate con il flusso, il traffico potrebbe aumentare, portando alla formazione di lunghe code. I risultati ottenuti sono riportati in Figura 4.14. Questo comportamento non si verifica invece per l'agente DQN, come mostrato in Figura 4.15, il quale sembra mantenere prestazioni costanti lungo le varie iterazioni. Tale deduzione è confermata anche dalle medie mobili e dalle deviazioni standard dei due agenti su un numero uguale di training, come mostrato rispettivamente nelle Figure 4.17 e 4.16. Si nota come per l'agente DQN la deviazione standard sia fortemente ridotta rispetto a quella mostrata dall'agente a ciclo fisso.

Il confronto tra il comportamento medio dell'agente a ciclo fisso e quello dell'agente dotato di un algoritmo di Apprendimento Automatico è rappresentato in Figura 4.20. Il grafico suggerisce che l'agente a ciclo fisso performa meglio in condizioni di traffico intenso, mantenendo un numero cumulativo di veicoli in coda più basso. Tuttavia, in situazioni di traffico ridotto, la performance dell'agente a ciclo fisso sembra essere inferiore rispetto all'agente con Apprendimento Automatico.

Per confermare queste intuizioni, è stato condotto uno studio sulle due funzioni, calcolandone media e deviazione standard nei pressi dei picchi di traffico. Si è scelto di studiare separatamente i due picchi per confrontare le effettive performance in condizioni di basso traffico (primo picco) e alto traffico (secondo picco). La divisione effettuata viene riportata in Figura 4.21.

Una volta calcolate la media e la deviazione standard per entrambi i modelli, riportate rispettivamente in Tabella 4.1 e Tabella 4.2, è stato applicato

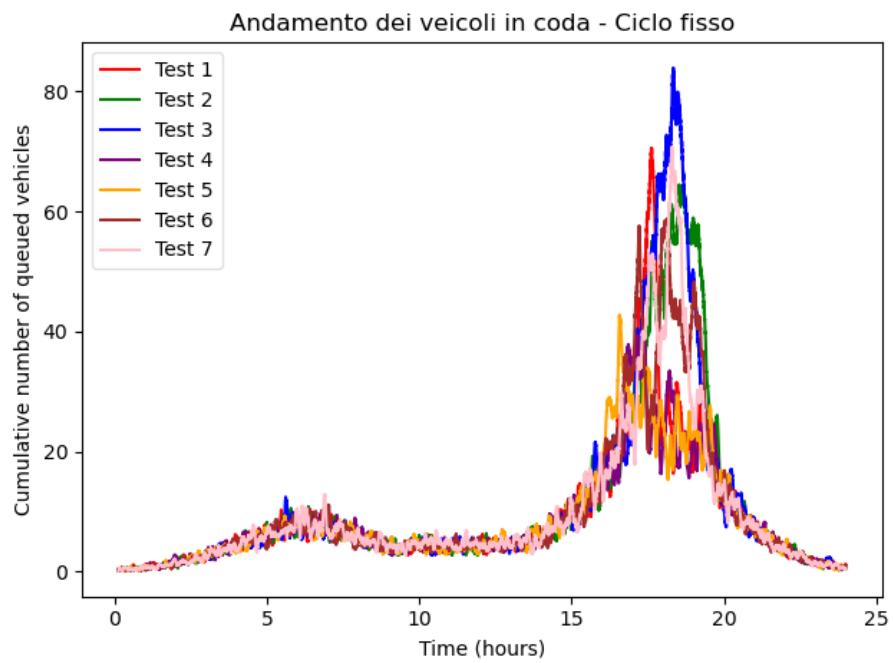


Figura 4.14: Risultati di 7 iterazioni di test utilizzando l'agente a ciclo fisso.

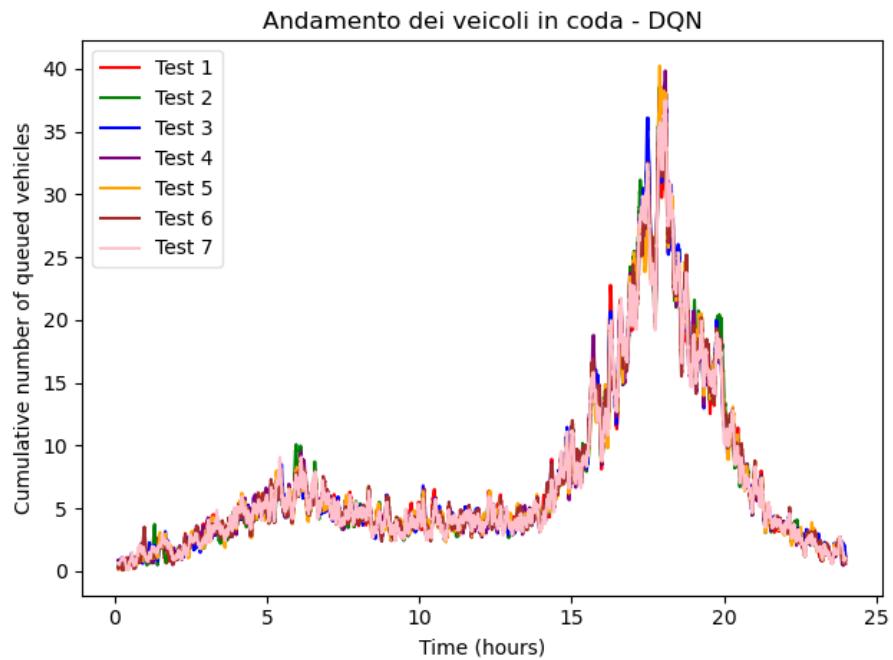


Figura 4.15: Risultati di 7 iterazioni di test utilizzando l'agente DQN. Il grafico mostra una notevole stabilità delle performance.

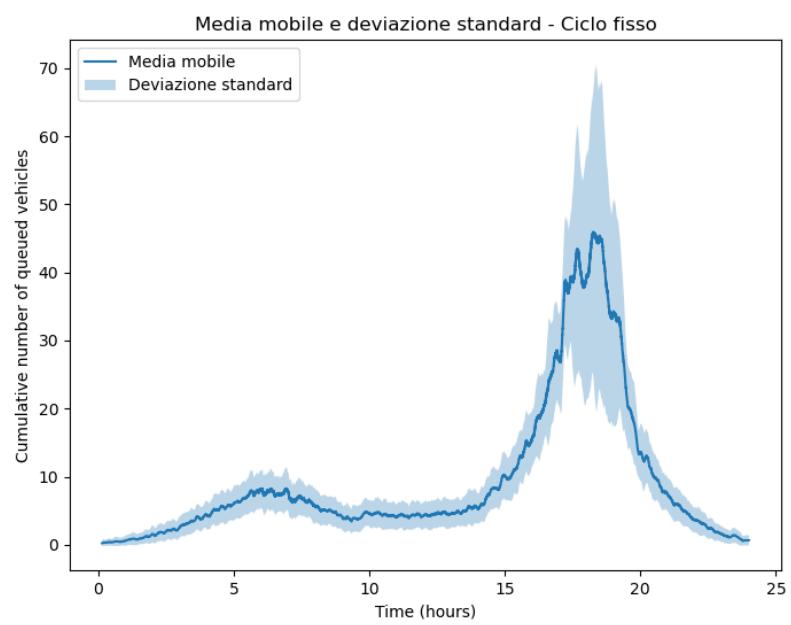


Figura 4.16: Grafico che mostra i valori medi ottenuti utilizzando l'agente a ciclo fisso insieme alla deviazione standard. Per facilitare la visualizzazione dei dati è stata applicata una media mobile di finestra pari a 100 elementi.

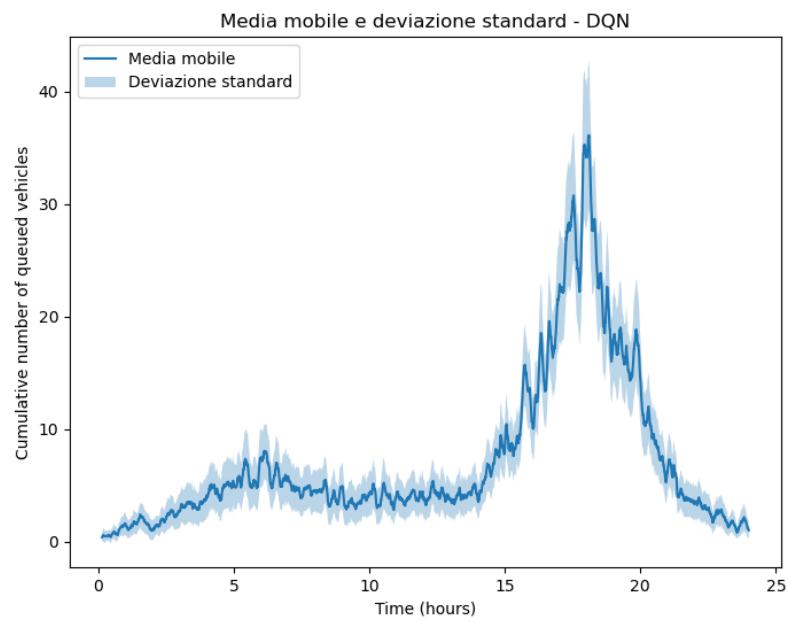


Figura 4.17: Grafico che mostra i valori medi ottenuti utilizzando l'agente con DQN insieme alla deviazione standard. Per facilitare la visualizzazione dei dati è stata applicata una media mobile di finestra pari a 100 elementi.

un test statistico per determinare se le medie delle performance dei due modelli fossero significativamente diverse. Utilizzando un test t di Welch (approfondito in Appendice A), è stato ottenuto un p-value estremamente vicino a zero, come mostrato in Tabella 4.3, il che ci permette di confermare che le differenze tra le medie sono statisticamente significative. Il test è stato effettuato poichè le ipotesi sono valide: le distribuzioni sono normali negli intervalli scelti, sono indipendenti e hanno devianze diverse.

Condizione	Media	Deviazione Standard
Traffico Moderato	4.9268	2.8242
Traffico Intenso	26.0777	17.8480

Tabella 4.1: Media e Deviazione Standard delle performance dell’agente DQN.

Condizione	Media	Deviazione Standard
Traffico Moderato	5.9286	2.3512
Traffico Intenso	24.4743	13.1747

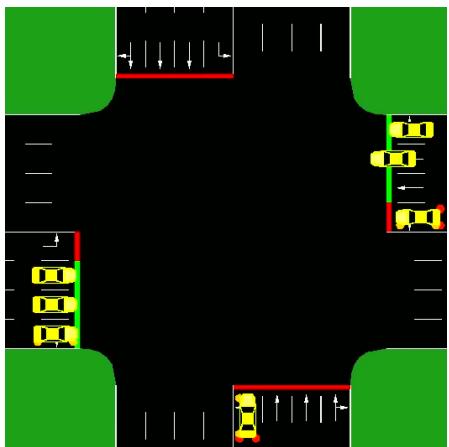
Tabella 4.2: Media e Deviazione Standard delle performance dell’agente a Ciclo Fisso.

Condizione	t-value	p-value
Traffico Moderato	17.9173	1.6182×10^{-70}
Traffico Intenso	4.7506	2.0610×10^{-6}

Tabella 4.3: Risultati del t-test per il confronto degli agenti a Ciclo Fisso e DQN

Questo porta a confermare la nostra tesi: l’algoritmo DQN mantiene un numero di veicoli in coda inferiore durante le fasi di traffico moderato, mentre l’agente a ciclo fisso performa meglio in condizioni di traffico intenso. Tale risultato è stato confermato anche da Vidali, Crociani, Vizzari et al. [10] in un altro studio.

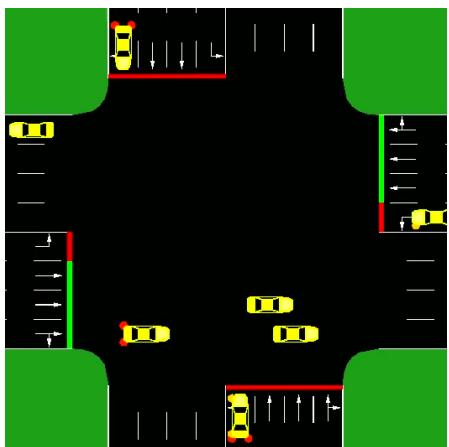
In Figura 4.19 e Figura 4.18 sono presentate due serie di screenshot che illustrano il paragone di comportamento tra l’agente ciclico e l’agente intelligente. Si osserva chiaramente come l’agente intelligente effettui scelte ottimizzate e opportunistiche basate sul traffico, mentre l’agente a ciclo fisso generi condizioni di traffico non necessarie e compia azioni che non favoriscono il passaggio dei veicoli.



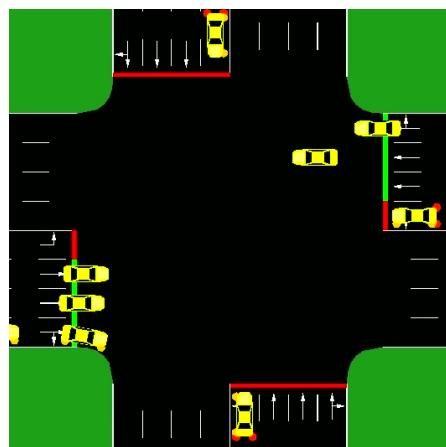
(a) DQN al tempo t_0



(b) DQN al tempo $t_0 + 5$



(c) DQN al tempo $t_0 + 10$



(d) DQN al tempo $t_0 + 15$

Figura 4.18: Raccolta di screenshot che illustrano le interazioni dell'agente intelligente con l'ambiente. L'agente, basato su DQN, mostra un comportamento più opportunistico e adattivo.

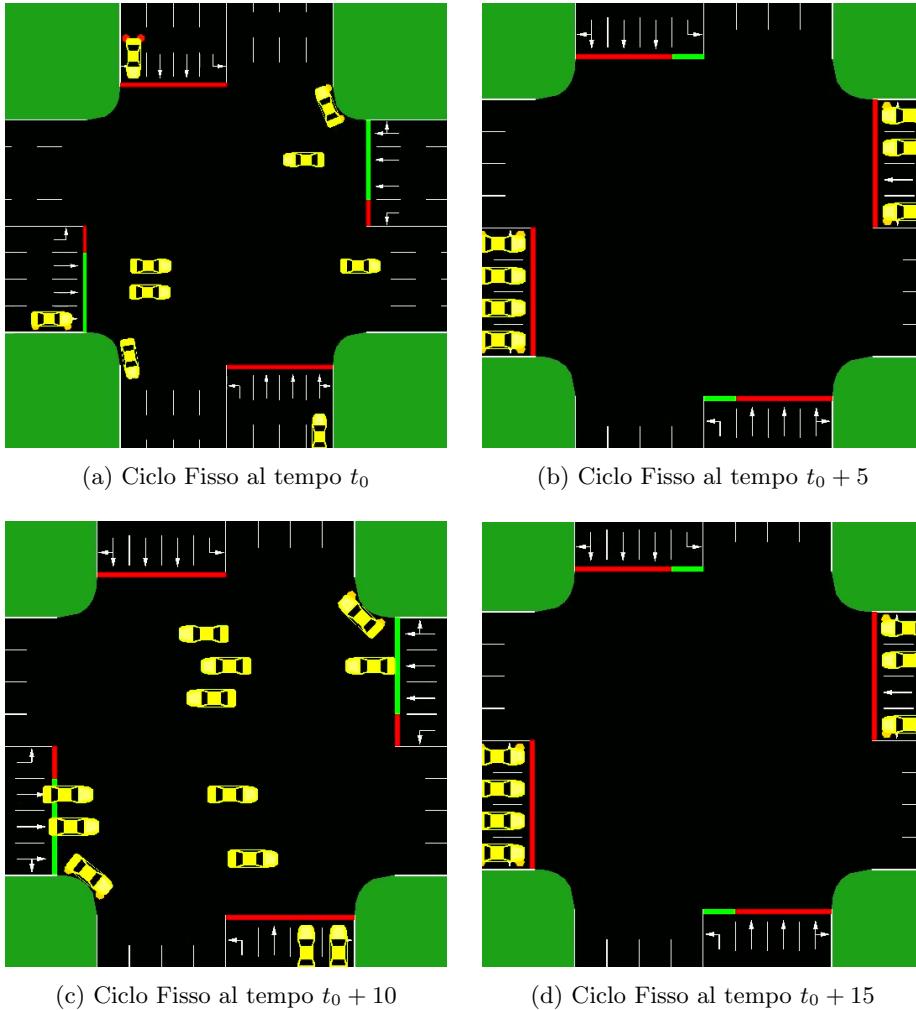


Figura 4.19: Raccolta di screenshot che illustrano le interazioni dell’agente a ciclo fisso con l’ambiente. L’agente tende a generare code con azioni non ottimali.

4.5 Conclusioni

In conclusione, l’obiettivo inizialmente prefissato di studiare l’efficacia del Curriculum Learning in questo contesto applicativo è stato raggiunto con successo. I risultati dimostrano che il Curriculum Learning migliora significativamente il tempo di convergenza e produce un agente con prestazioni comparabili a quelle di un agente addestrato tramite un processo classico.

Per quanto riguarda invece il confronto tra l’agente ciclico e l’agente intelligente, i risultati mostrano che entrambi gli approcci hanno i loro lati positivi. L’agente intelligente performa meglio in condizioni di traffico limitato, ma perde efficacia in condizioni di traffico intenso, dove l’agente ciclico sembra invece ottenere prestazioni superiori.

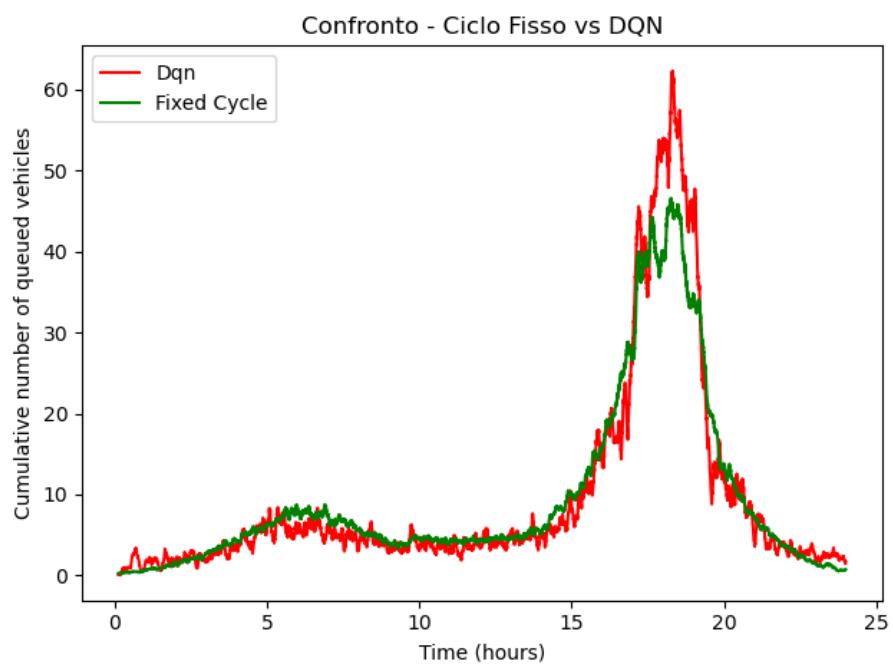


Figura 4.20: Confronto tra l'agente a ciclo fisso e l'agente con DQN; il grafico mostra la media dei valori ottenuti dai due approcci. Si osserva che, durante le fasi di traffico intenso, l'agente a ciclo fisso tende a mantenere un numero cumulativo di veicoli in coda più basso rispetto all'agente con DQN.

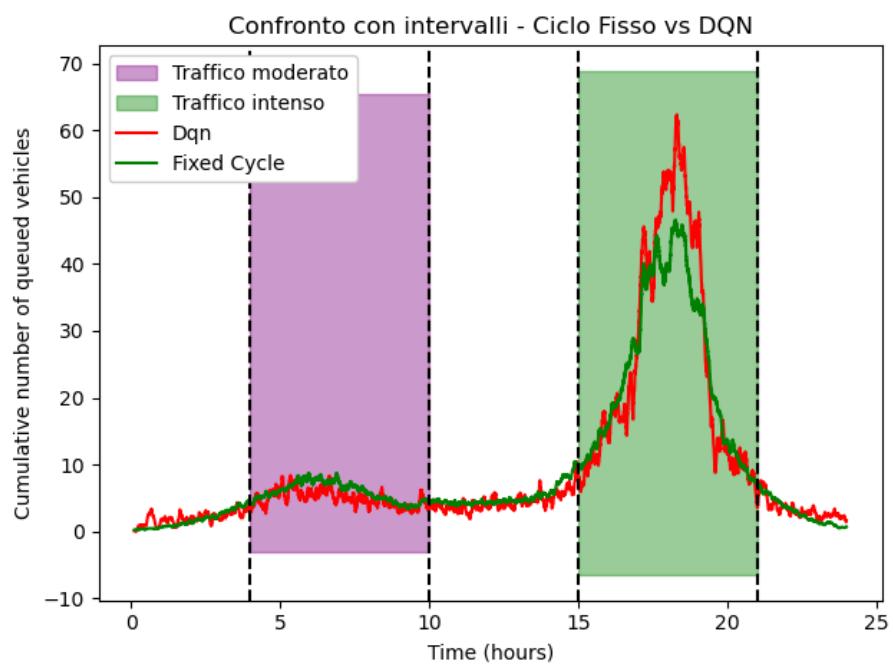


Figura 4.21: Confronto tra l'agente a ciclo fisso e l'agente con DQN. Il grafico mostra i due intervalli di tempo utilizzati per l'analisi statistica delle prestazioni dei due sistemi, evidenziando i periodi di traffico moderato e intenso.

Questi risultati sono particolarmente promettenti, poiché indicano che il Curriculum Learning può essere una strategia efficace nella creazione di agenti capaci di gestire il problema del traffico in città sovraffollate, come le metropoli contemporanee. L'adozione di tali agenti può contribuire a ridurre i costi associati all'addestramento, rendendo queste soluzioni più appetibili dal punto di vista del consumo di risorse.

La ricerca ha tuttavia messo in luce alcuni punti critici principali: in certi contesti, un agente a ciclo fisso risulta più efficace rispetto a un agente intelligente, e l'approccio curriculare richiede un investimento significativo di tempo preliminare per la sua realizzazione. Per affrontare queste sfide, studi successivi potrebbero sviluppare metodologie di addestramento capaci di creare agenti più performanti in situazioni di alto traffico e programmi per automatizzare la creazione di addestramenti curriculari su misura.

4.5.1 Sviluppi futuri

Gli sviluppi futuri di questa ricerca possono essere articolati in diversi ambiti, ciascuno con il potenziale di migliorare ulteriormente l'efficacia e l'efficienza degli agenti semaforici basati sulle tecniche di Apprendimento Profondo per Rinforzo. Di seguito sono elencati alcuni possibili sviluppi futuri:

- Esplorare l'uso delle nuove formule di generazione studiate in Yin, Li, Zhang et al. [3] per arricchire l'addestramento con nuovi contesti
- Continuare lo studio del Curriculum Learning applicato a diversi algoritmi di Apprendimento per Rinforzo, come approfondito in seguito, per comprendere meglio come strutturare l'apprendimento curriculare
- Ottimizzare gli iperparametri dell'ambiente di addestramento attraverso un processo sistematico, al fine di selezionare le condizioni ottimali per ciascuna delle fasi affrontate, migliorando l'efficacia dell'addestramento
- Testare i modelli ottenuti da questo studio in contesti multi-agente, dove due agenti collaborano per gestire il traffico di due incroci collegati, valutando così le capacità di cooperazione e coordinazione tra agenti

4.5.2 Applicazione di altri algoritmi

Considerati i risultati ottenuti utilizzando l'algoritmo DQN, è stato naturale estendere l'analisi applicando altri algoritmi, mantenendo la stessa struttura dell'approccio curriculare finale descritto in 4.3, con l'obiettivo di ottenere risultati analoghi. In questo studio preliminare sono stati scelti due algoritmi nello specifico, A2C e PPO, la cui implementazione è stata estremamente agevole grazie alle interfacce comuni degli algoritmi fornite da Stable Baselines 3. Questa sperimentazione sembra suggerire che lo stesso percorso non è ugualmente efficace per tutti gli algoritmi. Tale risultato è osservabile in Figura 4.22, Figura 4.23 e Figura 4.24; dove il comportamento dei due algoritmi in esame risulta estremamente diverso da quello di DQN: i primi step non sembrano fornire spunti di apprendimento al modello, mentre gli ultimi step iniziano a metterlo in difficoltà, portando però a situazioni instabili con errori molto rilevanti in valore assoluto. Questo è il motivo per cui il processo di addestramento è stato concluso prima di raggiungere la convergenza.

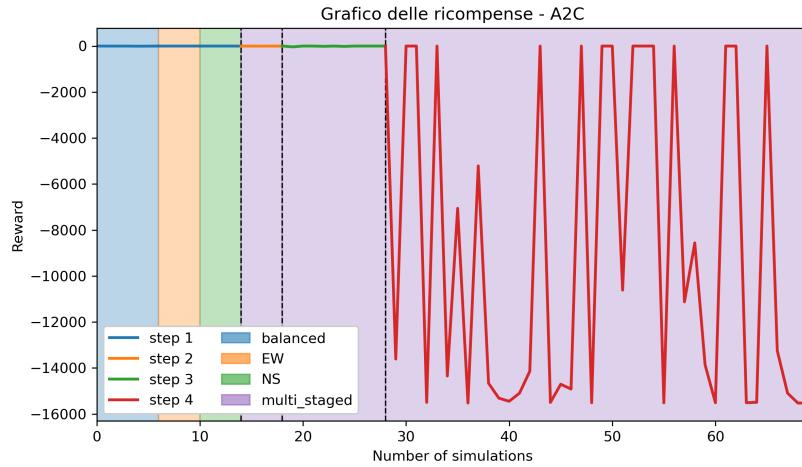


Figura 4.22: Grafico della ricompensa ottenuta da un algoritmo A2C applicato al processo curriculare descritto in 4.3.

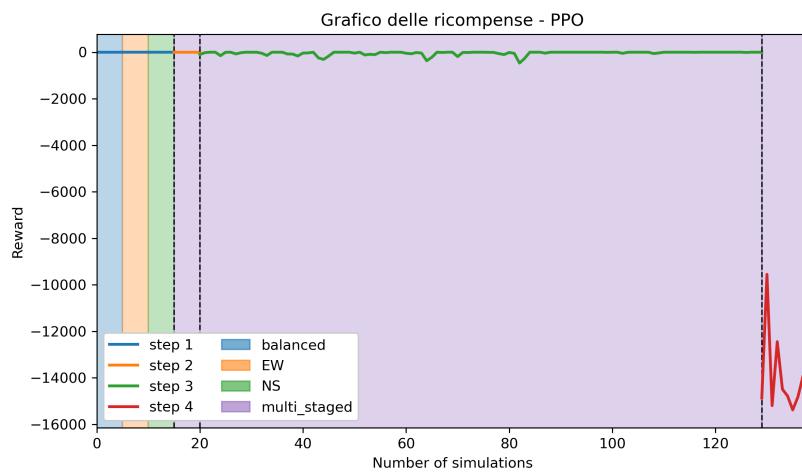


Figura 4.23: Grafico della ricompensa ottenuta da un algoritmo PPO applicato al processo curriculare descritto in 4.3.

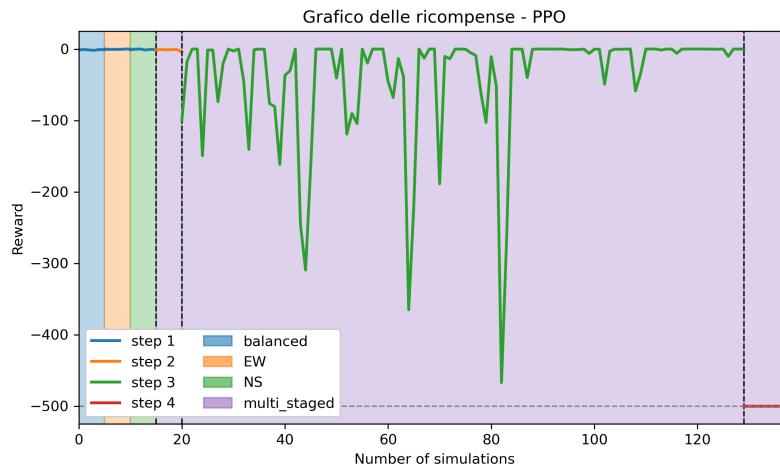


Figura 4.24: Grafico della ricompensa ottenuta da un algoritmo PPO applicato al processo curriculare descritto in 4.3, con un limite minimo impostato a -500 per consentire l'osservazione dell'andamento della ricompensa nei primi tre step. Quando la ricompensa si sovrappone alla linea di minimo grigia, significa che in quella sezione del grafico la ricompensa assume valori inferiori al minimo.

Questo potrebbe essere giustificato dal fatto che ogni algoritmo di apprendimento per rinforzo ha un proprio metodo di apprendimento. Sorge quindi la domanda se possa esistere un percorso curriculare comune che permetta a ogni algoritmo di apprendere efficacemente una task, o se sia necessario uno studio specifico basato sulla famiglia di algoritmi per identificare un percorso favorevole per ciascuno.

Uno studio più approfondito e applicato a diversi contesti potrebbe portare a una migliore comprensione di come adattare i percorsi curriculari alle specificità di ciascun algoritmo, ottimizzando così il processo di apprendimento per una varietà di metodi di apprendimento per rinforzo.

Bibliografia

- [1] C. J. Watkins e P. Dayan, «Q-learning,» *Machine learning*, vol. 8, pp. 279–292, 1992.
- [2] L. P. Kaelbling, M. L. Littman e A. W. Moore, «Reinforcement Learning: A Survey,» en, *Journal of Artificial Intelligence Research*, vol. 4, pp. 237–285, mag. 1996, ISSN: 1076-9757. DOI: 10.1613/jair.301. indirizzo: <https://www.jair.org/index.php/jair/article/view/10166> (visitato il 15/07/2024).
- [3] S. Yin, Z. Li, Y. Zhang, D. Yao, Y. Su e L. Li, «Headway distribution modeling with regard to traffic status,» in *2009 IEEE Intelligent Vehicles Symposium*, 2009, pp. 1057–1062. DOI: 10.1109/IVS.2009.5164427.
- [4] V. Mnih, K. Kavukcuoglu, D. Silver et al., *Playing Atari with Deep Reinforcement Learning*, arXiv:1312.5602 [cs], dic. 2013. DOI: 10.48550/arXiv.1312.5602. indirizzo: <http://arxiv.org/abs/1312.5602> (visitato il 15/07/2024).
- [5] Y. Bengio, Jérôme Louradour, R. Collobert e J. Weston, «Curriculum Learning,» 2016, pp. 41–48. indirizzo: <https://dl.acm.org/doi/10.1145/1553374.1553380>.
- [6] K. Arulkumaran, M. P. Deisenroth, M. Brundage e A. A. Bharath, «Deep reinforcement learning: A brief survey,» *IEEE Signal Processing Magazine*, vol. 34, n. 6, pp. 26–38, 2017.
- [7] M. Roderick, J. MacGlashan e S. Tellex, *Implementing the Deep Q-Network*, arXiv:1711.07478 [cs], nov. 2017. DOI: 10.48550/arXiv.1711.07478. indirizzo: <http://arxiv.org/abs/1711.07478> (visitato il 15/07/2024).
- [8] P. Alvarez Lopez, M. Behrisch, L. Bieker-Walz et al., «Microscopic Traffic Simulation using SUMO,» in *2019 IEEE Intelligent Transportation Systems Conference (ITSC)*, IEEE, nov. 2018, pp. 2575–2582. indirizzo: <https://elib.dlr.de/127994/>.
- [9] L. N. Alegre, *SUMO-RL*, <https://github.com/LucasAlegre/sumo-rl>, 2019.
- [10] A. Vidali, L. Crociani, G. Vizzari e S. Bandini, «A Deep Reinforcement Learning Approach to Adaptive Traffic Lights Management,» en, 2019.
- [11] M. Morales, *Grokking Deep Reinforcement Learning*. Manning Publications, 2020, ISSN: 1098-6596 eprint: arXiv:1011.1669v3, ISBN: 978-85-7811-079-6.

- [12] A. Raffin, A. Hill, A. Gleave, A. Kanervisto, M. Ernestus e N. Dormann, «Stable-Baselines3: Reliable Reinforcement Learning Implementations,» *Journal of Machine Learning Research*, vol. 22, n. 268, pp. 1–8, 2021. indirizzo: <http://jmlr.org/papers/v22/20-1364.html>.

A Appendice

La seguente appendice presenta delle informazioni sugli strumenti matematici utilizzati nell'elaborato.

A.1 Distribuzione Uniforme

La distribuzione uniforme è una distribuzione di probabilità che ha un insieme di valori equamente probabili. È definita da due parametri, a e b , che rappresentano gli estremi dell'intervallo in cui la variabile casuale può assumere valori. La funzione di densità di probabilità della distribuzione uniforme continua è data da:

$$f(x) = \begin{cases} \frac{1}{b-a} & \text{per } a \leq x \leq b \\ 0 & \text{altrimenti} \end{cases}$$

Le proprietà principali della distribuzione uniforme sono:

- **Media:** $\mu = \frac{a+b}{2}$

- **Varianza:** $\sigma^2 = \frac{(b-a)^2}{12}$

- **Funzione di ripartizione:** $F(x) = \begin{cases} 0 & \text{per } x < a \\ \frac{x-a}{b-a} & \text{per } a \leq x \leq b \\ 1 & \text{per } x > b \end{cases}$

A.2 Distribuzione di Weibull

La distribuzione di Weibull è una distribuzione di probabilità continua, utilizzata spesso per modellare il tempo di vita di un prodotto o la durata fino al guasto di un componente. È definita da due parametri, k (il parametro di forma) e λ (il parametro di scala). La funzione di densità di probabilità della distribuzione di Weibull è data da:

$$f(x; k, \lambda) = \begin{cases} \frac{k}{\lambda} \left(\frac{x}{\lambda}\right)^{k-1} e^{-(x/\lambda)^k} & \text{per } x \geq 0 \\ 0 & \text{per } x < 0 \end{cases}$$

Le proprietà principali della distribuzione di Weibull sono:

- **Media:** $\mu = \lambda \Gamma(1 + \frac{1}{k})$, dove Γ è la funzione Gamma.

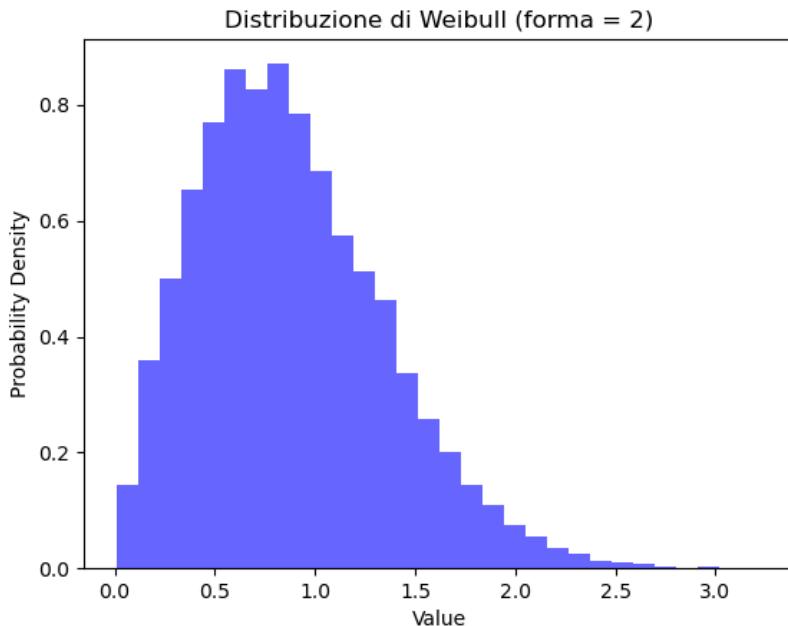


Figura A.1: Rappresentazione della distribuzione di Weibull con parametro di forma pari a 2. Si osserva un picco iniziale che cresce rapidamente, seguito da una diminuzione graduale.

- **Varianza:** $\sigma^2 = \lambda^2 \left[\Gamma(1 + \frac{2}{k}) - (\Gamma(1 + \frac{1}{k}))^2 \right]$
- **Funzione di ripartizione:** $F(x; k, \lambda) = \begin{cases} 1 - e^{-(x/\lambda)^k} & \text{per } x \geq 0 \\ 0 & \text{per } x < 0 \end{cases}$

La scelta di questa distribuzione è stata fatta perché rappresenta fedelmente il flusso tipico del traffico: il picco corrisponde ai momenti di massima congestione, che si dissolve gradualmente dopo essere emerso rapidamente. In particolare, è stata scelta una distribuzione di Weibull con parametro di forma pari a 2, come illustrato in Figura A.1.

A.3 T-Test

Il test t è una tecnica statistica utilizzata per confrontare le medie di due gruppi e determinare se esiste una differenza statisticamente significativa tra di esse. Può essere a uno o a due campioni.

Le assunzioni da rispettare sono:

- I dati devono seguire una distribuzione normale.
- Le osservazioni devono essere indipendenti.
- Per i t-test a due campioni, le varianze dei due gruppi devono essere uguali (per il test di Student) o possono essere diverse (per il test di Welch).

L'esecuzione si compone di questi step:

1. Formulare le ipotesi:

- Ipotesi nulla (H_0): Non c'è differenza significativa tra le medie dei gruppi.
- Ipotesi alternativa (H_a): C'è una differenza significativa tra le medie dei gruppi.

2. Calcolare il valore del t: Utilizzare le formule specifiche per il tipo di t-test che si sta eseguendo.

3. Determinare i gradi di libertà: Utilizzare le formule specifiche per il tipo di t-test che si sta eseguendo.

4. Calcolare il p-value: Confrontare il valore calcolato del t con una distribuzione t di Student per ottenere il p-value.

5. Confrontare il p-value con il livello di significatività (α): Se il p-value è minore di α (solitamente 0.05), si rifiuta l'ipotesi nulla.

A.3.1 Test t di Welch

Il test t di Welch è una variante del test t di Student che non assume l'uguaglianza delle varianze tra i due gruppi. È particolarmente utile quando i gruppi hanno varianze diverse e dimensioni campionarie diverse.