Relazione Progetto IIA 2023

Università degli Studi di Milano Bicocca, Riccardo Chimisso 866009 - Alberto Ricci 869271

Studio sull’applicazione di apprendimento per rinforzo alla gestione di un’intersezione semaforizzata.

Obiettivo

Si vuole confrontare per un particolare tipo di intersezione semaforizzata, riferita d'ora in poi con **2WSI** (**2 W**ay **S**ingle **I**ntersection), uno schema di gestione dei semafori a ciclo fisso con due diversi schemi di gestione controllati da agenti che hanno appreso per rinforzo.
In particolare, verranno quindi confrontati 3 schemi di controllo:

- Ciclo fisso: le fasi del semaforo sono fisse e si ripetono sempre uguali.
- Q-Learning: il semaforo è controllato da un agente che ha appreso per rinforzo usando la tecnica del Q-Learning, discussa in dettaglio più avanti.
- Deep Q-Learning: il semaforo è controllato da un agente che ha appreso per rinforzo usando la tecnica del Deep Q-Learning, discussa in dettaglio più avanti.

Per le varianti QL e DQL verranno sperimentate 2 diverse configurazioni con variazione del discount factor (gamma).
Ciascuno di questi modelli verrà addestrato con 2 diverse situazioni di traffico, prima una a basso traffico e poi una ad alto traffico. Successivamente i modelli verranno valutati sia rieseguendoli sulla stessa situazione di traffico sia eseguendoli sulla situazione di traffico non vista durante l'addestramento.
Questa scelta è motivata dal voler non solo confrontare i modelli tra di loro, ma anche verificare quanto i modelli ad apprendimento riescano a generalizzare, evitando l'overfitting, e quindi adattarsi a diverse situazioni di traffico.
La robustezza degli agenti è molto importante in quanto nella realtà è facile che un'intersezione semaforizzata sia soggetta a traffico variabile: basta pensare alla differenza tra orario di punta e notte, oppure mesi feriali e festivi.

Apprendimento per rinforzo

Reinforcement Learning

Il Reinforcement Learning è una tecnica di apprendimento che prevede l'apprendimento di un agente attraverso l'interazione con un ambiente dinamico. L'agente interagisce con l'ambiente in modo sequenziale, compiendo azioni e ricevendo una ricompensa (reward).
Lo scopo dell'agente è quello di massimizzare la ricompensa cumulativa che viene fornita dall'ambiente in risposta alle sue azioni.
L'RL si basa su un processo di apprendimento per esplorazione e sperimentazione, in cui l'agente deve scegliere le azioni da effettuare in modo da massimizzare la sua ricompensa. L'agente apprende dalla sua esperienza accumulando conoscenze e sviluppando strategie sempre più efficaci.
Lo scopo di un agente è quindi $max \sum_{t=0}^{T} R(s_t, a_t)$ dove T è il massimo di istanti temporali.
È da notare che un agente con tale scopo potrebbe trovarsi in indecisione nel caso di sequenze di azioni la cui ricompensa totale sia pari. Ad esempio date le sequenze di ricompensa $\langle 0, 0, 1 \rangle$ e $\langle 1, 0, 0 \rangle$ quale dovrebbe scegliere l'agente? Per decidere si introduce il discount factor γ per diminuire il peso che le ricompense future hanno rispetto alle più immediate, così che l'agente scelga la massimizzazione più veloce della ricompensa cumulativa. Il discount factor è $0 \leq \gamma \leq 1$ e la ricompensa al tempo t è data da: $R_t = r_t + \gamma r_{t+1} + \gamma^2 r_{t+2} + \ldots + \gamma^{T-t} r_T = \sum_{i=t}^{T} \gamma^{i-t} r_i = r_t + \gamma R_{t+1}$, dove r_i è la ricompensa per la transizione dell'istante di tempo i-esimo. Questa sommatoria altro non è che la serie geometrica, e in quanto tale converge sempre a un valore finito anche per $T = \infty$.

Q-Learning

Il Q-Learning è uno specifico algoritmo di Reinforcement Learning che si basa sulla costruzione di una tabella Q che indichi il valore di ricompensa per ogni possibile stato al compimento di una qualsiasi delle azioni possibili.
Per costruire tale tabella si utilizza una procedura iterativa in cui l'agente esplora l'ambiente eseguendo delle azioni più o meno casuali. In dettaglio, ad ogni passo, la tabella sarà aggiornata con: $Q[s][a] = Q[s][a] + \alpha(r + \gamma * max(Q[s']) - Q[s][a])$, dove s è lo stato attuale, a l'azione compiuta, r la ricompensa ottenuta, s' lo stato successivo, γ il discount factor, α il learning rate e $max(Q[x])$ restituisce la massima ricompensa ottenibile dallo stato x.
In questo modo la cella della tabella che rappresenta il valore atteso della ricompensa per il compimento dell'azione a nello stato s convergerà gradualmente all'effettivo valore.
Come anticipato, la scelta dell'azione da compiere sarà inizialmente casuale, finché non si decide che si è esplorato abbastanza. La politica spesso più usata a questo fine è la ε-greedy, dove data una ε iniziale che rappresenta la probabilità di compiere un'azione casuale, si diminuisce tale valore al progredire delle iterazioni fino a un minimo.
Il QL è una tecnica molto potente ed efficace, in grado di apprendere strategie di azione ottimali in una vasta gamma di applicazioni. Tuttavia, può essere sensibile al rumore, all'indeterminazione delle azioni e all'applicazione su ambienti continui. Inoltre, il QL richiede un grande quantità di memoria per memorizzare la tabella Q, specialmente quando l'ambiente ha uno spazio di stati (S) e di azioni (A) possibili ($\Theta(SA)$).

Deep Reinforcement Learning

Il Deep Reinforcement Learning è una tecnica di apprendimento automatico basata sull'RL, ma che si pone come obiettivo di sopperire alla problematica di quest'ultimo per spazi di stati e azioni molto grandi. Per farlo si utilizzano delle reti neurali profonde (Deep Neural Networks, da cui il nome) per approssimare i valori della Q table senza richiederne le stesse risorse in termini di memoria.

Deep Q-Learning

L'approccio più semplice per l'implementazione di una rete neurale profonda come approssimatore per la Q table consiste nell'utilizzare una rete neurale profonda ad ogni passo per ottenere la ricompensa attesa e aggiornare i pesi della rete neurale tramite il metodo del gradient descent rispetto alla ricompensa effettivamente ottenuta.
Questo approccio ha però lo svantaggio di non rispettare due condizioni importanti per la probabile convergenza di un metodo di apprendimento supervisionato come le reti neurali:

- Gli obiettivi (target) della rete neurale non sono stazionari, ovvero variano nel tempo, poiché è la rete stessa che ottiene i target attuali in base alle predizioni che essa stessa fa per i target futuri. Infatti la rete neurale stima i valori di Q, che rappresentano l'atteso guadagno futuro associato ad una coppia stato-azione, e questi valori vengono utilizzati per calcolare i target per l'aggiornamento dei pesi della rete stessa. Poiché la rete neurale si aggiorna utilizzando i suoi stessi output per stimare i target futuri, i target non sono fissi e variano continuamente nel tempo e questo rende la rete neurale instabile e prona alla non convergenza.
- Gli input alla rete neurale non sono indipendenti ed identicamente distribuiti poiché generati da una sequenza temporale con correlazione sequenziale e dipendono dalla Q table utilizzata dall'agente, che può cambiare nel tempo a seguito dell'esperienza maturata.

Deep Q-Network

L'approccio che prende il nome di Deep Q-Network cerca di arginare le problematiche del più semplice DQL tramite i seguenti due metodi:

- Per diminuire la non stazionarietà dei target si introduce una seconda rete neurale profonda, detta target network, che viene usata per stimare i target a cui deve convergere la rete principale, detta main network, in fase di addestramento. I pesi della target network vengono anch'essi aggiornati con il progredire dell'addestramento, ma con una frequenza molto minore rispetto a quella della main network. In questo modo, è possibile dividere l'addestramento in tanti piccoli problemi di apprendimento supervisionato che vengono presentati all'agente in maniera sequenziale. Questo non solo consente di aumentare la probabilità di convergenza, ma anche migliorare la stabilità del training, sebbene a costo di una velocità di convergenza minore, in quanto non vengono utilizzati i valori più aggiornati dei target.
- Per ridurre l'impatto della correlazione tra gli input viene adottata la tecnica Experience Replay, ovvero l'utilizzo di una struttura dati chiamata replay buffer all'interno della quale salvare dei campioni $(s,\ a,\ r,\ s')$ raccolti dall'agente durante l'apprendimento così da poterlo addestrare anche su dei gruppi di campioni selezionati casualmente dal replay buffer, che in questo modo permette di rendere gli input un po' più i.i.d. di quanto effettivamente siano. Inoltre questa tecnica consente di imparare di più dai singoli episodi, richiamare eventi rari e, in generale, fare un uso migliore dell'esperienza accumulata dall'agente.

Strumenti

SUMO

SUMO (Simulation of Urban MObility) è un simulatore open source di mobilità urbana.
SUMO consente agli sviluppatori di simulare il traffico veicolare e pedonale in un ambiente urbano, consentendo loro di testare e valutare soluzioni di mobilità come semafori intelligenti, veicoli autonomi, car pooling e molto altro ancora.
Il simulatore è altamente personalizzabile e consente agli utenti di definire le caratteristiche dei veicoli, delle strade e degli incroci, nonché delle condizioni meteorologiche e del traffico, per creare scenari realistici. Inoltre, SUMO offre diverse metriche di valutazione, come il tempo di percorrenza, il consumo di carburante, le emissioni di gas a effetto serra e il tempo di attesa di ciascun veicolo, che possono essere utilizzate per valutare le prestazioni dei sistemi di mobilità.
Viene utilizzato in questo progetto proprio come simulatore dell'ambiente, creata la rete stradale come 2WSI e le diverse situazioni di traffico da dover gestire.
SUMO fornisce anche un'API chiamata TraCI (Traffic Control Interface), un'interfaccia di controllo del traffico per consentire l'interazione tra il simulatore di traffico e gli agenti esterni, come ad esempio i sistemi di controllo del traffico intelligenti.
L'interfaccia messa a disposizione da TraCI è basata su socket che consente agli utenti di controllare i veicoli nel simulatore, modificare le caratteristiche della rete stradale e dei semafori, e ottenere informazioni sullo stato del traffico in tempo reale. Inoltre, TraCI consente anche la registrazione e la riproduzione di scenari di traffico per analizzare i risultati della simulazione. TraCI è supportato da una vasta gamma di linguaggi di programmazione, tra cui Python utilizzato in questo progetto.

Sumo-RL

Sumo-RL è un progetto open source basato su SUMO e TraCI per l'applicazione di algoritmi di RL ad ambienti di simulazione del traffico per la gestione di intersezioni semaforizzate. Fornisce un'interfaccia in Python semplice da utilizzare per la creazione di ambienti in SUMO e la loro gestione tramite algoritmi di RL. In particolare è possibile utilizzare un'implementazione già fatta del QL, inoltre è facilmente possibile integrare gli ambienti forniti con altre implementazioni di algoritmi di altre librerie, come ad esempio la DQN di Stabe Baselines 3, purché tali implementazioni accettino un ambiente Gymnasium (in caso di singolo semaforo) o Petting Zoo (in caso di multipli semafori).

Matplotlib

Matplotlib è una libreria in Python per la creazione di visualizzazioni statiche, animate o interattive. Poiché semplice da usare e già integrata con Jupyter Notebook, viene usata in questo progetto per creare e salvare grafici di varie metriche raccolte durante l'esecuzione degli algoritmi di RL.
Inizialmente si era anche pensato di permettere la visualizzazione in tempo reale della costruzione dei grafici delle metriche, ma, nonostante si fosse ottenuto tale risultato, si è scelto di escludere questa funzione poiché non valeva l'enorme rallentamento che ricadeva sulle simulazioni.

Stable Baselines 3

Stable Baselines 3 è un'altra libreria open source in Python che fornisce implementazioni di molteplici algoritmi di RL e DRL, di cui in questo progetto si è selezionata quella per DQN, e l'integrazione con ambienti di Gymnasium e Petting Zoo. È stato in realtà necessario installare una versione particolare di SB3 per garantire l'effettiva integrazione con ambienti Gymnasium.

Python, Anaconda e Jupyter Notebook

Python è stato il linguaggio scelto per questo progetto vista la sua grande adeguatezza nelle applicazioni di apprendimento automatico, e per via del fatto che sia Sumo-RL che Matplotlib sono scritti e facilmente integrabili in Python. Inoltre questo ha permesso l'utilizzo di Jupyter Notebook, che facilita l'esecuzione e la condivisione dello script principale del progetto, permettendo inoltre la visualizzazione dei grafici in real time (come detto però, questo è stato tolto) e la visualizzazione dei grafici completati (anche questa funzione in realtà rimossa per evitare output troppo lunghi, ma è facilmente riattivabile).
La versione di Python utilizzata è la $3.9.13$ tramite Anaconda, poiché SB3 utilizza Pytorch il quale a sua volta necessita di una versione non superiore alla 3.9 di Python. Inoltre tramite Anaconda e Pytorch è stato possibile eseguire le reti neurali direttamente sulle GPU NVIDIA a nostra disposizione.

Visual Studio Code

Visual Studio Code è stato l'IDE scelto per lo sviluppo del codice del progetto, dai file xml per la realizzazione degli ambienti SUMO, ai file Python e Jupyter Notebook per la scrittura, implementazione e esecuzione delle sperimentazioni. La scelta è ricaduta su questo editor poiché già conosciuto e con una facilissima integrazione di Git, Jupyter Notebook, Python e GitHub.

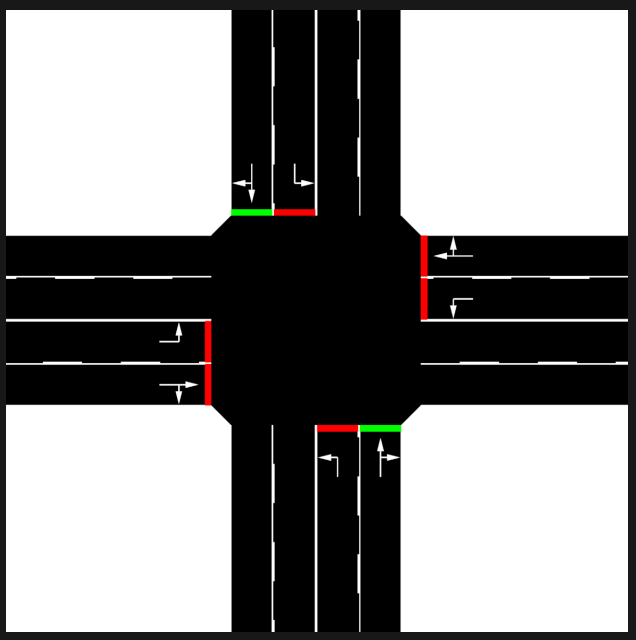
GitHub

Infine, GitHub è stato l'ultimo degli strumenti principali utilizzati nella realizzazione di questo progetto, permettendo una facile coordinazione tra sviluppi paralleli (questo più Git che GitHub) e uno spazio per salvare, pubblicare e condividere la repository del progetto, mantenendo comunque l'esclusiva proprietà della stessa. Inoltre ha reso facile la scelta di una licenza per la repository, nonché ha offerto la possibilità di visualizzare propriamente il README in Markdown.
Sebbene il README contenga la relazione stessa, è stata comunque caricata la relazione in PDF nella repository per permettere una più facile visualizzazione della stessa, con il vantaggio di visualizzare meglio l'indice, la divisione tra pagine (e quindi argomenti) e le formule Latex.
Inoltre la differenza principale tra la relazione e il README è che quest'ultimo è scritto in inglese.

Ambienti

Definizioni

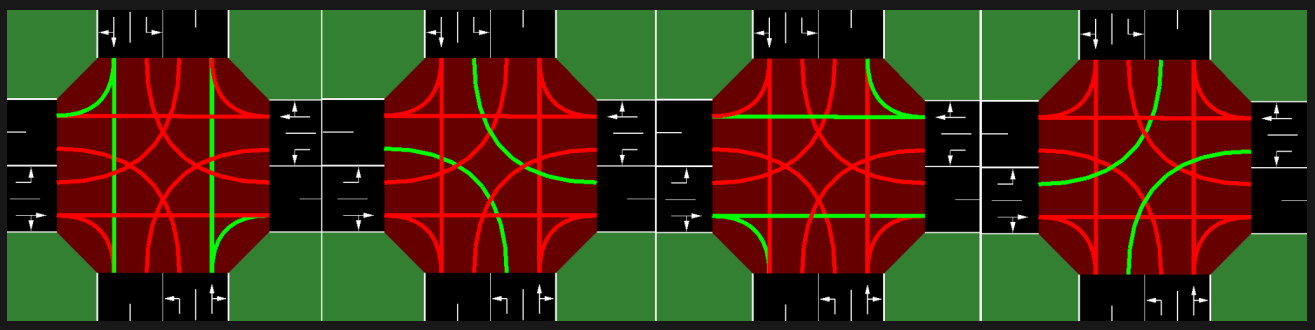
L'ambiente in cui gli agenti sono stati inseriti è una semplice intersezione con singolo semaforo che, come anticipato, si tratta della 2WSI raffigurata qui a destra. Questa rete stradale è stata definita nel file 2wsi.net.xml che, seguendo lo standard di SUMO, definisce le corsie, le luci semaforiche, i sensi di marcia, ecc.
All'interno dei file 2wsi-lt.rou.xml e 2wsi-ht.rou.xml sono invece presenti le definizioni per le situazioni di traffico, specificando quanti veicoli per unità temporale debbano esserci e che direzione debbano seguire.



Stati, Azioni e Ricompense

Per quanto riguarda la codifica di stati, azioni e ricompense si è scelto di utilizzare le definizioni fornite di base da Sumo-RL.

- Stato: ciascuno stato è rappresentato da un vettore `[phase_one_hot, min_green, lane_1_density, … ,lane_n_density, lane_1_queue, … ,lane_n_queue]` dove `phase_one_hot` è un vettore one-hot che codifica l'attuale fase verde attiva, `min_green` è un booleano che indica se sono già passati almeno `min_green` secondi di simulazione dall'attivazione della attuale fase verde, `lane_i_density` è il numero di veicoli in arrivo nella corsia i-esima fratto per la capacità totale della corsia, infine `lane_i_queue` è il numero di veicoli in coda nella corsia i-esima diviso la capacità totale della corsia.
- Azioni: ci sono 4 possibili azioni che corrispondono al cambiamento da una fase verde a un'altra come illustrato nella figura sotto.



- Ricompense: la funzione di ricompensa, chiamata Differential Waiting Time, è definita come il cambiamento cumulativo dei tempi di attesa dei veicoli (Total Waiting Time, twt), ovvero $r_t = twt_t - twt_{t+1}$ Questa misura indica quanto il tempo di attesa in risposta ad un'azione sia migliorato o peggiorato, forzando l'agente a cercare di compiere le azioni che portano alla diminuzione del twt, infatti se il twt diminuisce allo step successivo, la differenza porterà a un risultato positivo.

Configurazioni

L'ambiente di SUMO è stato configurato con i seguenti parametri per tutti gli esperimenti:

- 100000 secondi di simulazione.
- 10 secondi di simulazione tra un'azione e l'altra dell'agente. Questo, assieme ai secondi, porta il numero di passi compiuti dall'agente ad ogni run a 10000.
- 4 secondi di durata fissa per la fase gialla, ovvero la durata della fase in cui la luce semaforica passa da verde a gialla e poi a rossa.
- 10 secondi di durata minima per una fase verde.
- 50 secondi di durata massima per una fase verde.

Questi valori sono stati scelti dopo varie prove e dopo aver cercato dei valori comunemente utilizzati.

Per valutare gli agenti sono state selezionate le 4 metriche di sistema offerte da Sumo-RL:

- `system_total_stopped`, il numero totale di veicoli fermi (velocità ≤ $0.1\ m/s$) nel passo attuale.
- `system_total_waiting_time`, la somma di tutti i tempi di attesa di ciascun veicolo. Il tempo di attesa di un veicolo è definito come il tempo in secondi di simulazione che il veicolo passa a fermo dall'ultima volta che è stato fermo.
- `system_mean_waiting_time`, la media aritmetica di tutti i tempi di attesa dei veicoli.
- `system_mean_speed`, la media aritmetica delle velocità dei veicoli.

Per ciascuna run vengono generati dei file csv e dei grafici che rappresentano il valore delle metriche al variare dell'istante (passo, step) temporale. Più il valore della metrica è basso meglio è, tranne per la `system_mean_speed` per la quale invece è vero il contrario.
Infine vengono anche generati dei grafici per confrontare i risultati medi di ciascuna run.

Le 2 situazioni di traffico differenti sono omogenee (senza variazioni di traffico nel tempo) e l'unica differenza tra loro è che quella ad alto traffico ha 300 veicoli ogni ora, mentre quella a basso traffico ne ha la metà, 150.

Esperimenti e risultati

Iperparametri

In ciascuno degli esperimenti riportati di seguito le configurazioni degli agenti sono state queste:
Ciclo Fisso:

- nome: `fixedcycle`

QL-1:

- nome: `ql_1`
- α: 0.1
- γ: 0.75
- ε iniziale: 1
- ε finale: 0.01
- ε decadimento: 0.9996

QL-2:

- nome: `ql_2`
- α: 0.1
- γ: 0.9
- ε iniziale: 1
- ε finale: 0.01
- ε decadimento: 0.9996

DQL-1:

- nome: `dql_1`
- α: 0.1
- γ: 0.75
- ε iniziale: 1
- ε finale: 0.01
- ε decadimento: 0.99

DQL-2:

- nome: `dql_2`
- α: 0.1
- γ: 0.9
- ε iniziale: 1
- ε finale: 0.01
- ε decadimento: 0.99

È importante nota che l'ε decadimento per gli agenti QL indichi la costante moltiplicativa per cui la ε è moltiplicata ad ogni decisione presa dall'agente. Il valore di 0.9996 garantisce che per buona parte dell'addestramento la ε abbia un valore alto e quindi l'agente esplori.
Invece, per gli agenti DQL, l'ε decadimento indica la frazione di step in cui la ε vada dal valore iniziale a quello finale. Anche qui il valore di 0.99 garantisce che per la maggior parte degli step la ε assuma un valore altro e l'agente esplori.

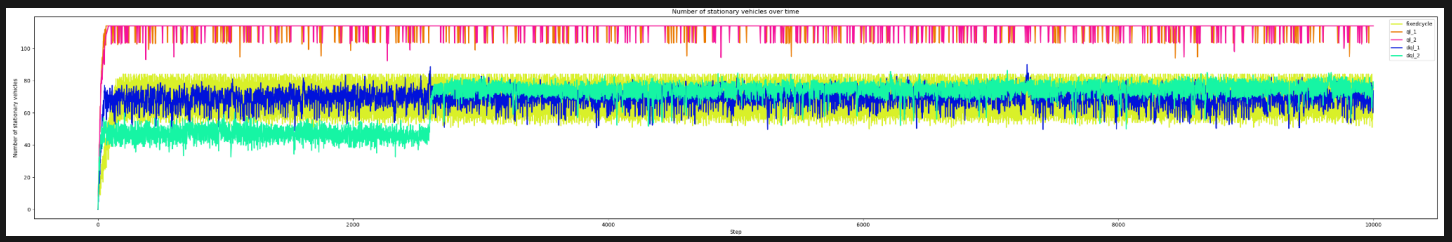
Infine, come traspare dai valori elencati sopra, le configurazioni sono tutte molto simili e cambiano per il tipo di modello utilizzato dall'agente, per il valore del discount factor o per il decadimento della probabilità di esplorazione. Ovviamente il modello a ciclo fisso non ha configurazioni di iperparametri perché non utilizza alcun modello di apprendimento automatico.

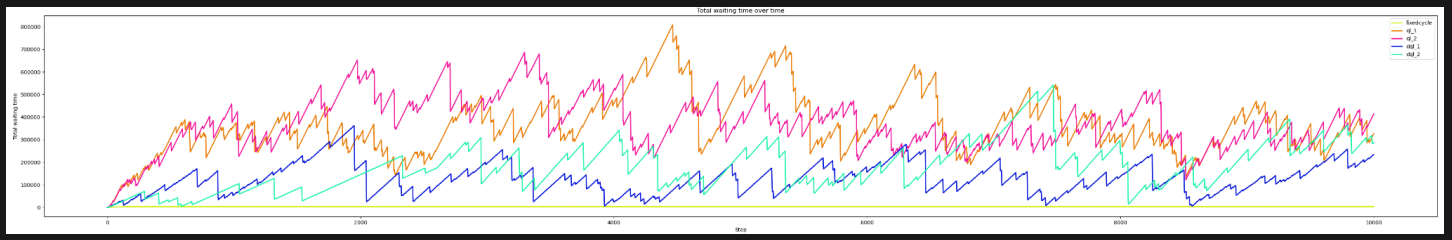
In questo documento vengono riportati i sunti dei risultati, è possibile visualizzare ogni grafico all'interno della repository sotto `experiments results`.

Inoltre, i grafici riportati qui rispecchiano la media di 3 diverse run.

Low Traffic - Low Traffic

In questo esperimento, abbreviato lt-lt, gli agenti sono stati addestrati utilizzando la situazione di traffico bassa specificata nel file 2wsi-lt.rou.xml per poi essere valutati sulla medesima situazione. I risultati non sono stati soddisfacenti in quanto l'agente fixedcycle si è comportato meglio di tutti gli altri. Ad esempio, ecco i grafici che indicano il numero totale di veicoli fermi e la somma totale dei tempi di attesa, entrambi al variare dello step:

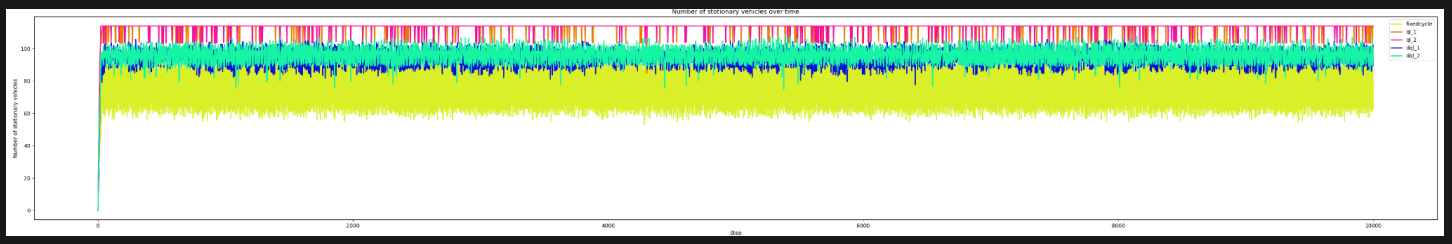


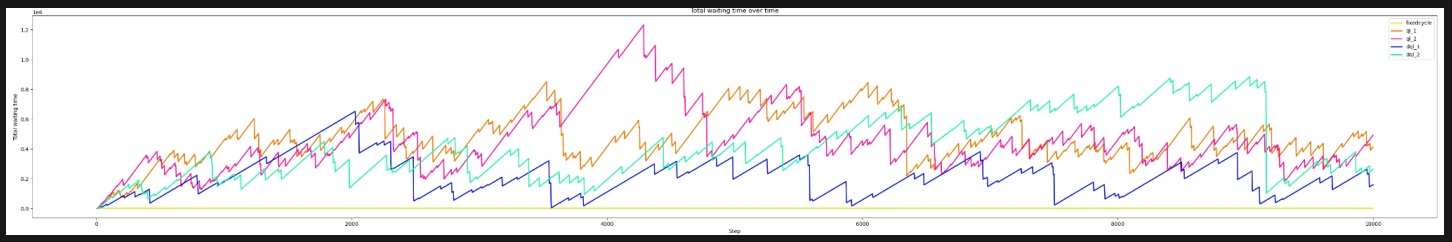


Come è possibile notare, specialmente dalla somma dei tempi di attesa, l'agente fixedcycle ha permesso una gestione del traffico notevolmente migliore rispetto agli agenti che hanno appreso per rinforzo. Un'altra nota interessante è che gli agenti QL, in ogni grafico, si sono comportati decisamente peggio rispetto rispetto a quelli DQL. È poi peculiare il primo quarto di run per l'agente dql_2 in quanto riesce a ottenere un numero di veicoli fermi minore rispetto al fixedcycle, nonostante successivamente questa situazione peggiori e vada a stazionare su un numero simile. Infine è interessante notare che, nonostante il numero di veicoli fermi sia molto simile se non migliore per gli agenti DQL rispetto al fixedcycle, i tempi di attesa complessivi sono fortemente peggiori.

Low Traffic - High Traffic

In questo esperimento, abbreviato lt-ht, gli agenti sono stati addestrati utilizzando la situazione di traffico bassa del file 2wsi-lt.rou.xml per poi essere valutati sulla situazione in 2wsi-ht.rou.xml. Risulta evidente come gli agenti addestrati su una situazione a basso traffico non siano riusciti ad adattarsi a una situazione ad alto traffico, infatti:

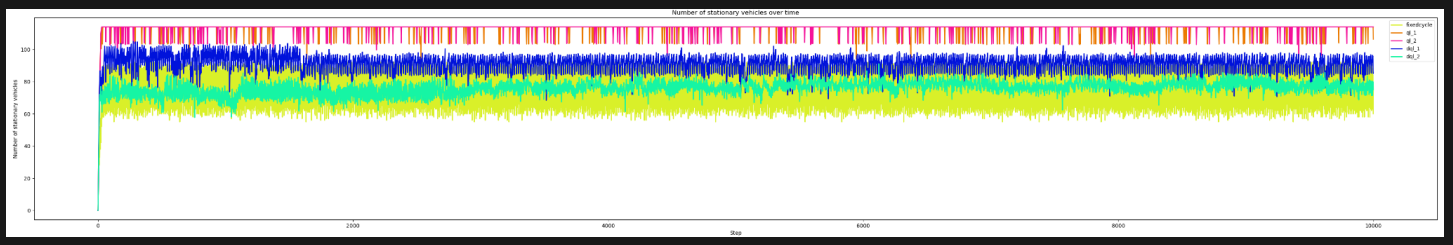


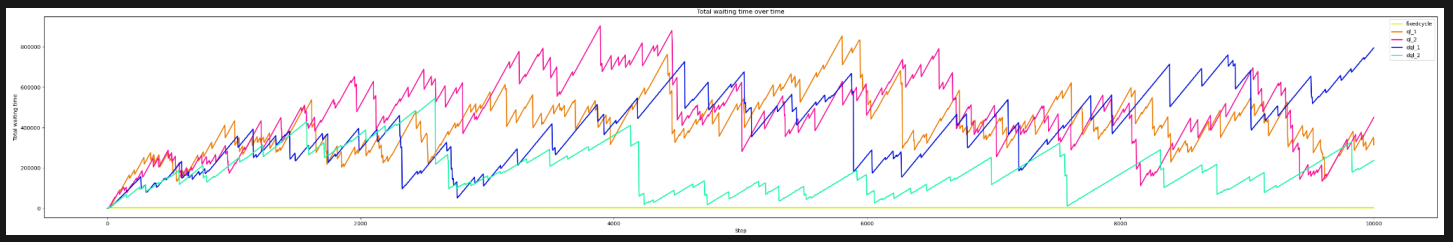


In questo esperimento, nonostante ogni modello abbia ottenuto punteggi peggiori, come ci si poteva aspettare dato il maggiore numero di veicoli, l'agente fixedcycle mantiene i valori migliori.
Nel numero di veicoli fermi, sebbene ancora gli agenti DQL si comportino meglio dei QL, questa volta non riescono ad ottenere dei valori simili al fixedcycle. Al contrario dell'esperimento precedente, qui l'agente dql_1 è spesso migliore rispetto al dql_2, anche se ottengono un punteggio molto simile per il numero di veicoli fermi. Infine nella somma dei tempi di attesa questa volta gli agenti QL, sebbene quasi sempre peggiori, non sempre si discostano di molto da quelli DQL.

High Traffic - High Traffic

In questo esperimento, abbreviato ht-ht, gli agenti sono stati addestrati utilizzando la situazione di traffico alta specificata nel file 2wsi-ht.rou.xml per poi essere valutati sulla stessa situazione.
I risultati non sono stati soddisfacenti poiché l'agente fixedcycle si è comportato meglio di ogni altro:

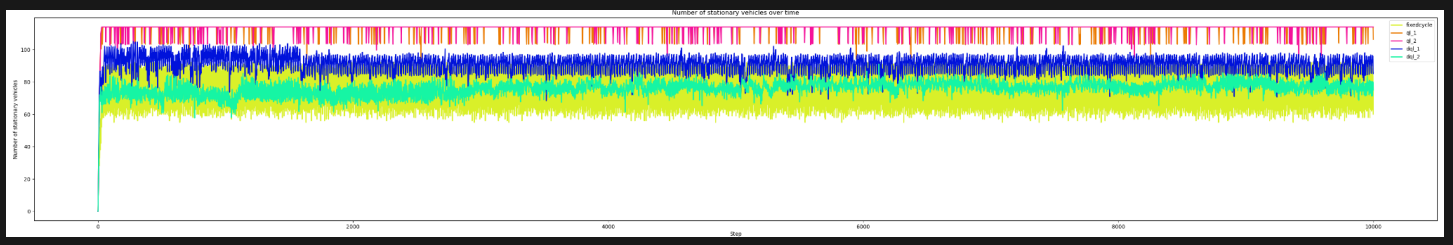


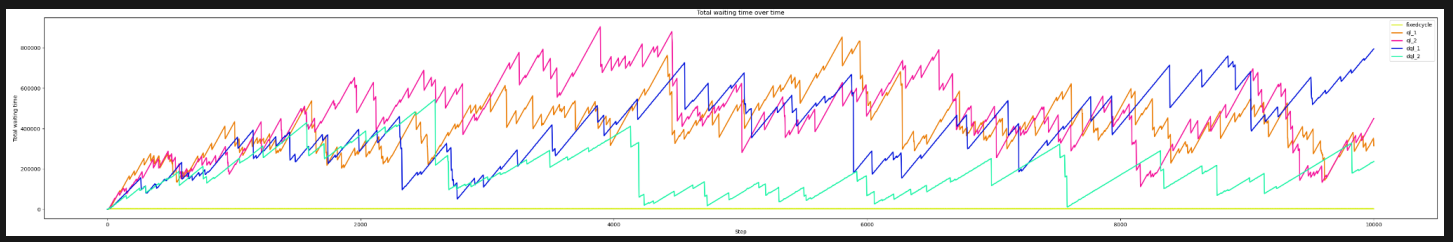


In questo esperimento l'agente dql_2 è quasi sempre migliore del dql_1, il quale invece ottiene risultati simili se non peggiori degli agenti QL. L'agente fixedcycle è qui il migliore in assoluto.

High Traffic - Low Traffic

In questo esperimento, abbreviato ht-lt, gli agenti sono stati addestrati utilizzando la situazione di traffico alta specificata nel file 2wsi-ht.rou.xml per poi essere valutati sulla situazione contenuta in 2wsi-lt.rou.xml. I risultati sono stati mediamente più soddisfacenti rispetto agli altri esperimenti, in quanto a tratti un agente DQL riesce a comportarsi leggermente meglio rispetto al fixedcycle:

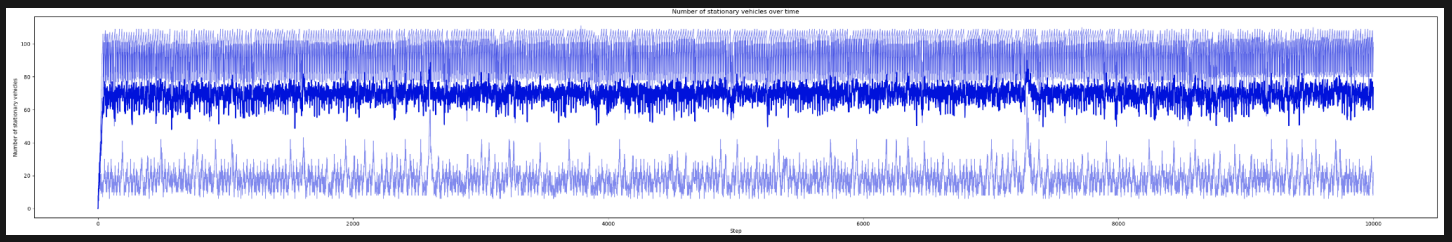


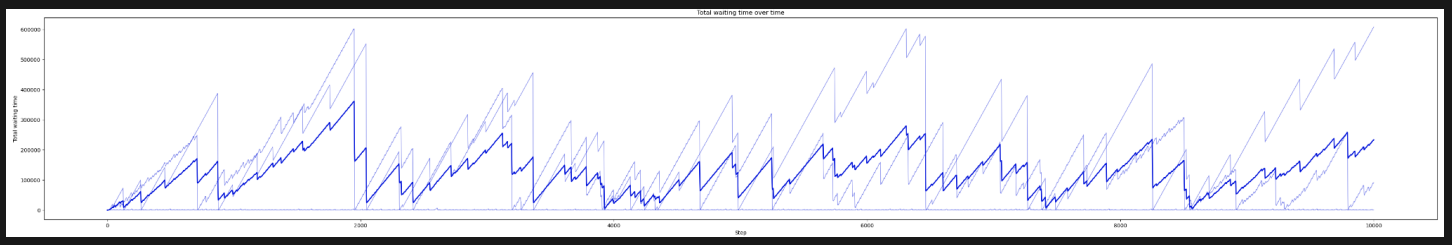


Si nota infatti come nel numero di veicoli fermi l'agente dql_2 ottenga un punteggio mediamente migliore rispetto al fixedcycle. Tuttavia solamente in alcuni brevi istanti riesce a raggiungere un punteggio comparabile a quello del fixedcyle per i tempi di attesa.
Anche qui, soprattuto, l'agente dql_1 non si comporta granché meglio rispetto ai QL.

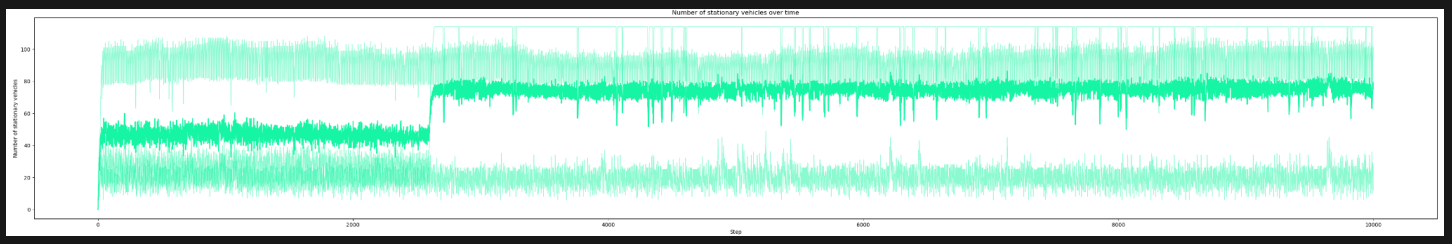
Considerazioni generali

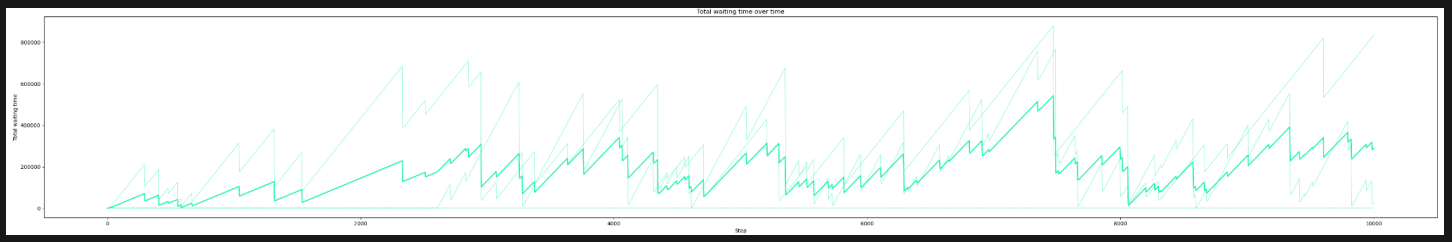
Per sintesi i risultati degli esperimenti riportati sopra, nonostante diano una buona panoramica generale, tralasciano alcuni dettagli interessanti ripetutisi in ognuno di essi.
Ad esempio gli agenti DQL, al contrario del fixedcycle e dei QL, hanno i risultati più eterogenei tra le 3 diverse run. Spesso infatti capita che ci sia una run ottenga un punteggio decisamente migliore, una run decisamente peggiore e l'ultima run un punteggio medio tra le altre due. Questo porta poi il confronto tra le medie a perdere il dettaglio di quelle run che hanno ottenuto dei punteggi particolarmente buoni. L'esperimento in cui questo fenomeno è più evidente è il lt-lt per l'agente dql_1, infatti dai seguenti grafici è possibile notare come una run in particolare abbia avuto un comportamento eccezionalmente ottimo, quasi sempre migliore anche del fixedcycle:





Inoltre è anche possibile vedere più nel dettaglio il motivo per cui la media dell'agente dql_2 nell'esperimento lt-lt abbia il primo quarto migliore rispetto agli altri:





C'è infatti una run che inizialmente ha un ottimo punteggio, tuttavia succede qualcosa per cui dopo circa 2600 step il numero di veicoli fermi aumenta di colpo e rimane più o meno fisso a 114. Investigare cosa sia successo attorno a quel momento potrebbe valerne la pena.

Anche in questi grafici si trova la stessa situazione descritta precedentemente, dove una run ha dei risultati comparabili e migliori rispetto al fixedcycle.

È sempre possibile confrontare i dati più nel dettaglio facendo riferimento ai file csv.

Conclusioni e possibili sviluppi

In conclusione, dai risultati degli esperimenti svolti non è stato possibile concludere che l'applicazione di agenti che hanno appreso per rinforzo dia un vantaggio significativo nella gestione del traffico per la 2WSI rispetto allo schema al ciclo fisso.
Tuttavia è evidente come il semplice modello QL non riesca a gestire il traffico, differentemente al DQL che invece in qualche caso ha ottenuto dei risultati molto buoni.
Nonostante con gli esperimenti fatti non si sia potuta affermare l'utilità dell'applicazione del DRL alla gestione del traffico, viste le run in cui gli agenti che utilizzavano tale modello hanno dimostrato una gestione più efficiente rispetto al ciclo fisso, è possibile che, aumentando il tempo di training, scegliendo diversi valori per gli iperparametri e/o dando la possibilità agli agenti di imparare su più episodi, si riesca ad ottenere e dimostrare un effettivo miglioramento e vantaggio nel loro utilizzo.

Come appena detto, quindi, una ricerca successiva più approfondita sugli agenti con DQL e i loro iperparametri potrebbe portare a conclusioni diverse e più soddisfacenti.
Si potrebbe ad esempio far sì che per ciascuna run di training abbia più episodi, oppure vagliare l'utilizzo di diverse combinazioni di iperparametri, o anche modificare la configurazione dell'ambiente.
Sicuramente poi si potrebbe, per diminuire le tempistiche richieste per l'addestramento e la valutazione, parallelizzare l'esecuzione dei diversi agenti.

Referenze

- P. Alvarez Lopez, M. Behrisch, L. Bieker-Walz, J. Erdmann, Yun-Pang Flötteröd, R. Hilbrich, L. Lücken, J. Rummel, P. Wagner, E. Wiessner, (2018).
  Microscopic Traffic Simulation using SUMO.
  IEEE Intelligent Transportation Systems Conference (ITSC).
  https://elib.dlr.de/124092/
- Lucas N. Alegre, (2019).
  SUMO-RL.
  GitHub repository.
  https://github.com/LucasAlegre/sumo-rl
- A. Raffin, A. Hill, A. Gleave, A. Kanervisto, M. Ernestus, N. Dormann, (2021).
  Stable-Baselines3: Reliable Reinforcement Learning Implementations.
  Journal of Machine Learning Research.
  http://jmlr.org/papers/v22/20-1364.html