Simulazione di sistemi

Filippo Speziali

12 maggio 2025

Indice

1 Introduzione:

In questo studio è stata analizzata una rete stradale semplice, composta da quattro intersezioni regolamentate da semafori. L'analisi si è focalizzata sull'osservazione di diverse metriche, valutando l'impatto della sincronizzazione semaforica e delle variazioni nel flusso di traffico in ingresso. Per la modellazione della rete, nonché per la simulazione del comportamento dei veicoli e dei semafori, è stato utilizzato **SUMO**[2]. Le analisi sono state poi condotte in Python, sfruttando la libreria **TraCI**[3] per l'interazione e il controllo della simulazione 1.1. Il codice completo del progetto si trova nella pagina github[1]. La struttura della rete è dettagliata nella sezione 1.2, mentre il comportamento dei veicoli e dei flussi è illustrato nella sezione 1.3. La metodologia sperimentale è descritta nella sezione 2, e i risultati ottenuti sono riportati nella sezione 3.

1.1 SUMO e TraCI

SUMO (Simulation of Urban MObility) è un software open-source per la simulazione del traffico veicolare, utilizzato per modellare reti stradali complesse, semafori e comportamenti dei veicoli. TraCI (Traffic Control Interface) è un'interfaccia che permette di controllare dinamicamente le simulazioni SUMO in tempo reale, ad esempio modificando i percorsi dei veicoli, i tempi dei semafori o raccogliendo dati statistici. TraCI mette a disposizione un' API che consente di dialogare con SUMO attraverso diversi linguaggi tra cui Python, con una integrazione client-server. SUMO opera come server responsabile dell'esecuzione della simulazione vera e propria, mantenendo attivi il modello stradale e la dinamica del traffico. Python svolge il ruolo di client, inviando istruzioni in tempo reale e ricevendo feedback costanti sullo stato della simulazione.

Lo pseudocodice (Alg. 1) mostra la logica di base dell'interazione tra TraCI e SUMO. Questo codice si può trovare nel file `simulator.py` il quale contiene la classe che

gestisce la simulazione. La funzione che esegue una singola run avvia la simulazione e si connette ad essa tramite TraCI. Durante l'esecuzione, viene impostata una configurazione specifica dei semafori e, a ogni step della simulazione, le metriche prestazionali vengono aggiornate utilizzando la classe `ManageTrackers` che verrà approfondita nella sezione 2.1.

Algorithm 1 Gestione simulazione con SUMO e TraCI

function SIMULATION_RUN($configuration = 0, gui = False$)
    Avvia SUMO con flag random
    Connetti Tramite TraCI
    $Trackers \leftarrow$ `ManageTrackers()`
    Configura semafori con $configuration$
    $step \leftarrow 0$
    while $step < simulation\_steps$ **do**
        Esegui passo di simulazione
        if $step > warm\_up$ **then**
            $Trackers$.UPDATE() ▷ Aggiorna metriche prestazionali (Alg. 3)
        end if
        $step \leftarrow step + 1$
    end while
    Chiudi connessione TraCI
    return lista di `Tracker` ▷ lista di metriche prestazionali per diverse sezioni della rete
end function

Sempre all'interno della classe `Simulator` (in `simulator.py`) È presente una funzione che permette di chiamare la singola run di simulazione per *numero_di_run* volte (Alg. 2). La funzione ritorna una lista con le metriche salvate le quali saranno analizzate da una classe apposita per calcolarne le stime puntuali e gli intervalli di confidenza 2.2.

Algorithm 2 Esecuzione di una configurazione specifica della simulazione

function SPECIFIC_CONFIG($configuration$)
    observations $\leftarrow$ lista vuota
    for $i \leftarrow 0$ **to** $numero\_di\_run$ **do**
        risultato $\leftarrow$ SIMULATION_RUN($configuration, gui = False$) ▷ (Alg. 1)
        Aggiungi risultato a observations
    end for
    return observations
end function

1.2 La rete stradale:

La rete stradale oggetto di studio è composta da una strada principale a due corsie per senso di marcia, con orientamento tangenziale. Lungo il suo tracciato, ogni 200 metri, si trova un'intersezione con una strada trasversale a una corsia per senso di marcia. Complessivamente, ci sono quattro di queste intersezioni. Il traffico è organizzato in

due flussi distinti: uno lungo la direzione tangenziale e uno lungo quella trasversale, senza possibilità di svolta per i veicoli. Ogni incrocio è regolato da un semaforo con un ciclo fisso di 15 secondi di verde e 15 secondi di rosso.

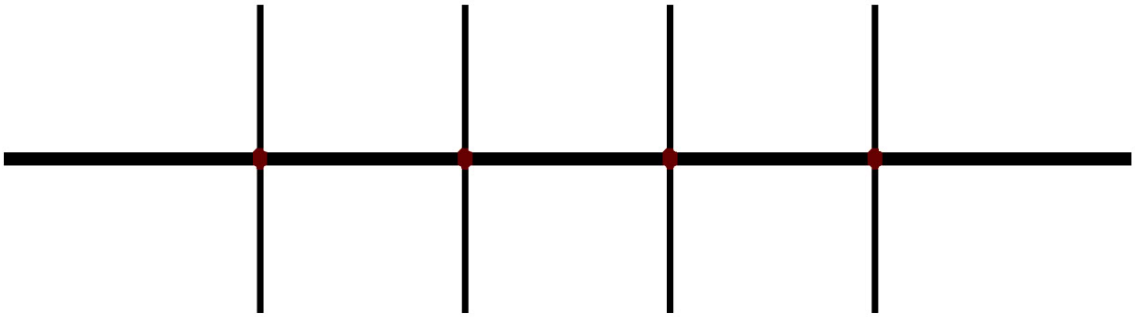
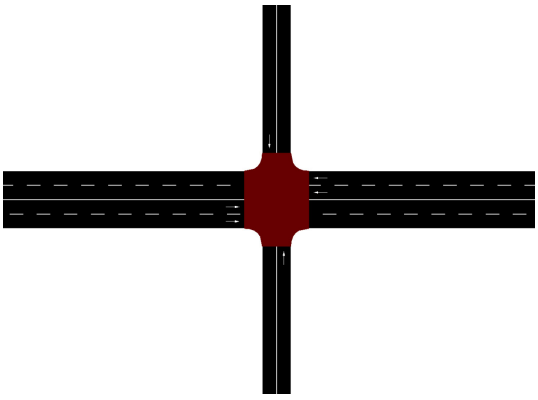
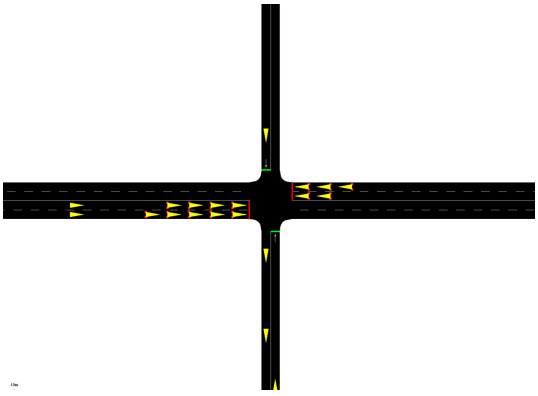


Figura 1: Network architecture



(a) Junction



(b) TLS example

Questa rete stradale è stata modellata utilizzando NETEDIT, uno strumento grafico fornito da SUMO che permette di definire nodi, connessioni, corsie, semafori e altri elementi della rete con un'interfaccia visuale intuitiva. La rete così creata è stata salvata in formato XML con il nome `lightSync.net.xml`, che viene successivamente utilizzato per avviare le simulazioni.

1.3 Veicoli e flussi

Lo studio si basa su un modello semplificato in cui tutti i veicoli appartengono a un'unica categoria e si muovono a velocità costante, senza variazioni dovute ad accelerazioni o decelerazioni. Il limite di velocità è fissato a 50 km/h e qualsiasi fattore di casualità nella guida è stato eliminato. Questa scelta consente di isolare l'effetto specifico della gestione semaforica.

Il file delle rotte (`lightSync.rou.xml`) genera, in modo stocastico e continuo, flussi

di veicoli. Gli intervalli tra le partenze sono campionati da una distribuzione esponenziale, il che fa sì che il numero di veicoli avviati in un intervallo di tempo fisso segua una legge di Poisson, con una media pari all'inverso del parametro esponenziale. SUMO impiega un generatore di numeri casuali basato sull'algoritmo Mersenne Twister.

Nel modello sono impiegati due flussi distinti: il flusso maggiore, indicato con L, per la direzione tangenziale e un flusso pari a L/3 per quella trasversale. Sono stati condotti diversi esperimenti variando i valori di L.

2 Descrizione degli esperimenti

In questo studio sono stati eseguiti diversi esperimenti, variando sia la sincronizzazione dei semafori sia il flusso in ingresso L. Per ciascuna configurazione sono state realizzate **20 run** di simulazione. Ogni run di simulazione è composto da **21000 step** di cui **6000** sono di **warm up** per eliminare il transiente iniziale (3). La configurazione di partenza è quella in cui tutti i semafori risultano perfettamente sincronizzati, ovvero con offset nullo:

$$\phi_0 = \phi_1 = \phi_2 = \phi_3 = 0$$

Definendo T come il ciclo completo del semaforo (15 secondi di rosso più 15 di verde), si è poi introdotto un *offset* incrementale che varia da 1 fino T/2 incluso. Viene applicato alla fase dei semafori in questo modo:

$$\phi_0 = 0$$

$$\phi_n = \phi_(n-1) + offset$$

Utilizzando T = 30 sono state esaminate **16 configurazioni**.
Ognuna delle 16 configurazioni è stata ripetuta su **3 valori del flusso L** diversi. Il valore di L corrisponde al parametro λ della distribuzione esponenziale.

$$L_0 = 0.3,\ L_1 = 0.4,\ L_2 = 0.5$$

Oltre al monitoraggio del flusso nelle direzioni longitudinale e trasversale, è stata analizzata l'utilizzazione del primo servente. Questa metrica è definita come il rapporto tra il tempo in cui è presente almeno una richiesta attiva e il tempo totale di osservazione. Un valore prossimo a 1 indica una condizione di saturazione del sistema.

2.1 Metriche osservate

In TraCI sono state implementate diverse funzioni per monitorare l'andamento dei veicoli e registrare specifiche metriche. Per farlo è stata implementata la classe `ManageTrackers` che gestisce durante la simulazione vari oggetti `Tracker`. Ogni tracker è associato ad una specifica parte della rete e tiene traccia di diverse metriche, tra cui il **tempo medio di permanenza nel sistema**, il **tasso di arrivo** e il **throughput in uscita** dei veicoli. La classe è estendibile per monitorare diversi serventi o sottoreti. I tracker utilizzati per questo progetto sono 3. I primi due tengono traccia rispettivamente del flusso tangenziale e di quello trasversale. Successivamente è stato anche

monitorato il primo semaforo semplicemente per rilevare con quale valore di lambda il suo tasso di utilizzazione si avvicina a 1, in modo da determinare su quali valori è possibile lanciare la simulazione senza arrivare a congestione.

La classe `ManageTrackers` gestisce l'aggiornamento di tutti i tracker attraverso la funzione update (Alg. 3). Questa funzione viene invocata ad ogni step di simulazione da (Alg. 1).

Algorithm 3 Aggiornamento dello stato dei tracker

```
1:  procedure UPDATE
2:      T ← T + 1
3:      vehicles_data ← tutti i risultati dei veicoli presenti nella simulazione
4:      for all (vehicle_id, speed, edge) ∈ vehicle_data do
5:          for all tracker do
6:              tracker.T ← T
7:              tracker.PROCESS_VEHICLE(vehicle_id, edge, speed)        ▷ (Alg. 4)
8:          end for
9:      end for
10: end procedure
```

Per ogni tracker viene chiamata ad ogni step una funzione che preso in input l'identificativo di un veicolo e la corsia in cui sta viaggiando con la sua velocità tiene traccia delle metriche di simulazione (Alg. 4). Un veicolo è considerato entrato nel sistema non appena raggiunge la prima coda (linea 2) oppure, se la coda è vuota, nel momento in cui viene servito (linea 3). Viene salvato il tempo di entrata. Se invece il veicolo inizia a percorrere la strada di uscita viene rimosso dal sistema e viene salvato il tempo totale di permanenza. Vengono anche aggiornati i numeri di arrivi e i numeri di completamenti. Dopo aver simulato una certa configurazione per T passi viene calcolato il throughput e il tasso di arrivo nel seguente modo:

$$\text{Throughput} = \frac{N_{\text{completamenti}}}{T}, \qquad \text{Arrival\_Rate} = \frac{N_{\text{arrivi}}}{T}$$

Algorithm 4 Elaborazione dei dati di un veicolo

```
1: procedure PROCESS_VEHICLE(vehicle_id, current_edge, vehicle_speed)
2:     in_queue ← current_edge ∈ incoming_routes and vehicle_speed < 0.1
3:     no_queue ← current_edge ∈ incoming_junction      ▷ il veicolo sta attraversando l'incrocio
4:     /* Caso in cui il veicolo è entrato nel sistema */
5:     if ((in_queue or no_queue) and vehicle_id ∉ system) then
6:         arrivals ← arrivals + 1
7:         system[vehicle_id] ← T                          ▷ Salva il tempo di arrivo
8:     end if
9:     /* Caso in cui il veicolo è uscito dal sistema */
10:    if current_edge ∈ outgoing_routes then
11:        depart_time ← system.pop(vehicle_id, None)
12:        if depart_time ≠ None then
13:            completed ← completed + 1
14:            /* Tempo di uscita - tempo di entrata = tempo nel sistema */
15:            lifetime.append(T − depart_time)
16:        end if
17:    end if
18: end procedure
```

2.2 Stime puntuali e intervalli di confidenza

Successivamente sono state calcolate le stime puntuali tra cui la **media**, la **varianza** e gli **intervalli di confidenza**. Fissata una configurazione e un valore di L si hanno M osservazioni. Sia X la variabile casuale corrispondente ad una certa metrica osservata si hanno quindi $X_1, ..., X_M$ osservazioni diverse. Per stimare la media si può calcolare:

$$\hat{\mu} = \bar{X} = \frac{1}{M} \sum_{m=1}^{M} X_m \tag{1}$$

La varianza viene calcolata invece attraverso la formula della varianza campionaria contente la correzione di Bessel per avere uno stimatore non distorto.

$$s^2 = \tilde{\sigma}^2 = \frac{1}{M-1} \sum (X_m - \bar{X})^2 \tag{2}$$

A questo punto la variabile casuale:

$$t = \frac{\hat{\mu} - \mu}{s/\sqrt{M}} \tag{3}$$

ha una distribuzione di probabilità t-student con $M-1$ gradi di libertà ed è poissibile calcolare degli intervalli di confidenza. È stato scelto un livello di confidenza del 95%. È possibile calcolare l'errore marginale e gli intervalli nel seguente modo:

$$t_{\text{crit}} = t_{M-1}\left(1 - \frac{\alpha}{2}\right) \quad \Rightarrow \quad \text{errore} = t_{\text{crit}} \cdot \frac{s}{\sqrt{M}} \tag{4}$$

$$CI = [\hat{\mu} - Errore, \hat{\mu} + Errore] \tag{5}$$

Le stime sono state calcolate utilizzando la libreria `numpy` e la libreria `scipy`. Il calcolo e salvataggio delle stime viene implementato nella classe `Statistics`. Un esempio di come il codice implementa le stime si trova nell'algoritmo (Alg. 5).

Algorithm 5 Calcolo delle statistiche su un array di osservazioni

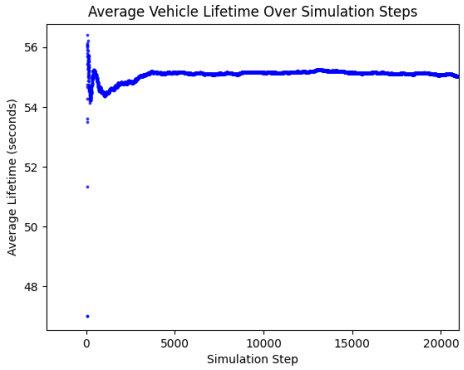
Require: `observations`: array contenente i valori della variabile aleatoria X per ogni run

$$X_1, X_2, ..., X_N$$

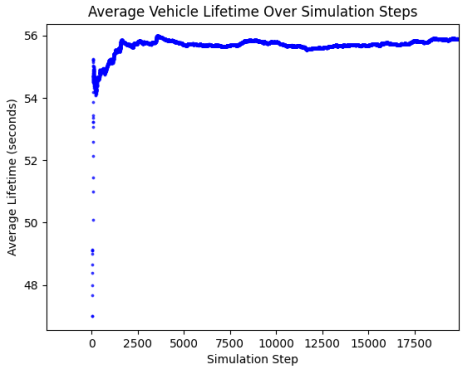
Require: `N`: numero di run

CONFIDENCE $\leftarrow 0.95$
$\alpha \leftarrow 1 - \text{CONFIDENCE}$
function STATS_ON_OBSERVATIONS(observations)
    sample_mean $\leftarrow$ `np.mean(observations)`
    sample_variance $\leftarrow$ `np.var(observations, ddof=1)`
    var_mean $\leftarrow$ sample_variance / `N`
    $t_{\text{crit}} \leftarrow$ `st.t.ppf(1 - `α` / 2, N - 1)`
    margin_error $\leftarrow t_{\text{crit}} \cdot \sqrt{\text{var\_mean}}$
    ci_lower $\leftarrow$ sample_mean $-$ margin_error
    ci_upper $\leftarrow$ sample_mean $+$ margin_error
    return intervallo di confidenza [*ci_lower*, *ci_upper*]
end function

Le stime sono fortemente influenzate dal transiente iniziale. Per attenuarne l'effetto, l'andamento della media è stato analizzato tramite il grafico corrispondente. Sulla base di queste osservazioni, sono stati scelti 6000 step di simulazione come fase di warm-up. In figura 3 sono riportati due diversi andamenti.



(a) $L_0 = 0.3$



(b) $L_0 = 0.4$

Figura 3: Average vehicle lifetime over simulation steps

3 Risultati

Di seguito vengono presentati i risultati, suddivisi in base al valore di L utilizzato. In conclusione, verranno svolte alcune considerazioni su come il lifetime vector varia in relazione ai diversi flussi.

3.1 Risultati relativi al flusso L_0

Scegliendo $L_0 = 0.3$ il sistema è perfettamente in grado di gestire correttamente tutte le richieste. La tabella 1 mostra i risultati dell'utilizzazione del primo servente, preso come esempio. Si osserva una utilizzazione media del 59%.

Configuration	Mean	Variance	CI Lower	CI Upper	Margin of Error
config_0	0.5916	3.5142e-05	0.5889	0.5944	2.7744e-03
config_1	0.5943	2.9125e-05	0.5918	0.5968	2.5258e-03
config_2	0.5960	2.6490e-05	0.5936	0.5984	2.4088e-03

Tabella 1: Utilizzazione sul primo servente tangenziale

La tabella 2 mostra i risultati del lifetime vector sul flusso trasversale. Si osserva come, nonostante le modifiche all'offset dei semafori, il lifetime medio rimanga invariato.

Configuration	Mean	Variance	CI Lower	CI Upper	Margin of Error
config_0	10.2977	5.05e-03	10.2644	10.3309	0.0333
config_1	10.2895	3.63e-03	10.2613	10.3177	0.0282
config_2	10.3037	6.83e-03	10.2650	10.3424	0.0387
config_3	10.3159	3.54e-03	10.2881	10.3438	0.0279
config_4	10.3037	3.27e-03	10.2769	10.3304	0.0268
config_5	10.2874	6.14e-03	10.2507	10.3241	0.0367
config_6	10.2808	6.18e-03	10.2440	10.3176	0.0368
config_7	10.2797	4.31e-03	10.2489	10.3104	0.0307
config_8	10.3049	4.60e-03	10.2731	10.3366	0.0317
config_9	10.3099	3.10e-03	10.2839	10.3360	0.0261
config_10	10.2956	3.75e-03	10.2670	10.3243	0.0287
config_11	10.2924	4.83e-03	10.2599	10.3249	0.0325
config_12	10.2742	3.30e-03	10.2474	10.3011	0.0269
config_13	10.2892	4.50e-03	10.2578	10.3206	0.0314
config_14	10.2902	7.99e-03	10.2483	10.3320	0.0418
config_15	10.3012	3.16e-03	10.2749	10.3275	0.0263

Tabella 2: Lifetime sul flusso trasversale

La tabella 3, invece, mostra il lifetime sul flusso tangenziale. Si osserva che il lifetime diminuisce all'aumentare dello shift dei semafori. utilizzando l'offset 15 si raggiunge il lifetime minimo, causato dal fenomeno dell'onda verde che si sviluppa in entrambe le direzioni sul flusso tangenziale.

Configuration	Mean	Variance	CI Lower	CI Upper	Margin of Error
config_0	100.0820	4.04e-03	100.0523	100.1118	2.97e-02
config_1	100.0970	1.42e-02	100.0413	100.1527	5.57e-02
config_2	89.4409	1.54e-02	89.3827	89.4990	5.82e-02
config_3	86.5019	1.94e-02	86.4367	86.5671	6.52e-02
config_4	81.7522	3.62e-02	81.6631	81.8412	8.91e-02
config_5	77.0760	3.75e-02	76.9854	77.1666	9.06e-02
config_6	76.7103	2.15e-02	76.6417	76.7789	6.86e-02
config_7	74.8088	4.70e-03	74.7767	74.8408	3.21e-02
config_8	72.0587	8.82e-03	72.0147	72.1026	4.40e-02
config_9	70.6932	8.07e-03	70.6512	70.7353	4.20e-02
config_10	70.0840	7.37e-03	70.0438	70.1242	4.02e-02
config_11	63.7012	1.21e-02	63.6496	63.7528	5.16e-02
config_12	59.5446	1.46e-02	59.4879	59.6012	5.66e-02
config_13	57.3000	1.39e-02	57.2448	57.3551	5.51e-02
config_14	55.6752	1.55e-02	55.6170	55.7334	5.82e-02
config_15	55.0669	9.47e-03	55.0214	55.1125	4.56e-02

Tabella 3: Lifetime sul flusso tangenziale

3.2 Risultati relativi al flusso L_1

Scegliendo come valore del flusso $L_1 = 0.4$ i risultati dell'utilizzazione del primo servente sono i seguenti.

Configuration	Mean	Variance	CI Lower	CI Upper	Margin of Error
config_0	0.7049	6.0732e-05	0.7012	0.7085	3.6473e-03
config_1	0.7056	6.8414e-05	0.7017	0.7095	3.8711e-03
config_2	0.7076	5.3762e-05	0.7042	0.7111	3.4316e-03

Tabella 4: Utilizzazione sul primo servente tangenziale

Il lifetime sulla direzione trasversale rimane invariato rispetto agli esperimenti fatti su L_0. Quello relativo al flusso tangenziale, pur essendo simile, risulta leggermente maggiore a causa di un carico piu elevato.

Configuration	Mean	Variance	CI Lower	CI Upper	Margin of Error
config_0	100.7964	1.47e-02	100.7396	100.8531	5.68e-02
config_1	100.8721	2.04e-02	100.8052	100.9390	6.69e-02
config_2	92.7803	3.30e-02	92.6952	92.8653	8.50e-02
config_3	90.2480	3.89e-02	90.1557	90.3403	9.23e-02
config_4	86.1292	4.61e-02	86.0287	86.2297	9.58e-02
config_5	80.5811	4.36e-02	80.4833	80.6788	9.77e-02
config_6	80.2653	3.91e-02	80.1727	80.3578	9.26e-02
config_7	76.8845	3.51e-02	76.7967	76.9722	8.77e-02
config_8	73.6078	2.28e-02	73.5370	73.6785	7.07e-02
config_9	72.0984	3.21e-02	72.0145	72.1823	8.39e-02
config_10	70.8504	1.14e-02	70.8004	70.9005	5.00e-02
config_11	65.8579	2.38e-02	65.7856	65.9302	7.23e-02
config_12	61.6281	1.62e-02	61.5686	61.6876	5.95e-02
config_13	58.6182	1.34e-02	58.5641	58.6724	5.41e-02
config_14	56.6146	1.11e-02	56.5653	56.6638	4.93e-02
config_15	55.8782	1.71e-02	55.8171	55.9394	6.12e-02

Tabella 5: Lifetime sul flusso tangenziale

3.3 Risultati relativi al flusso L_2

Scegliendo $L_2 = 0.5$ l'utilizzazione aumenta notevolmente come riportato in tabella 6.

Configuration	Mean	Variance	CI Lower	CI Upper	Margin of Error
config_0	0.8321	7.0131e-05	0.8281	0.8360	3.9193e-03
config_1	0.8326	9.1121e-05	0.8281	0.8371	4.4675e-03
config_2	0.8307	7.3432e-05	0.8267	0.8347	4.0105e-03

Tabella 6: Utilizzazione sul primo servente tangenziale

Nuovamente il flusso sulla direzione trasversale non cambia in funzione dell'offset dei semafori. In tabella 7 alcuni esempi.

Configuration	Mean	Variance	CI Lower	CI Upper	Margin of Error
config_0	11.2289	3.94e-03	11.1995	11.2582	0.0294
config_5	11.2512	1.40e-02	11.1958	11.3066	0.0554
config_10	11.2603	6.90e-03	11.2214	11.2992	0.0389
config_15	11.2448	5.27e-03	11.2108	11.2788	0.0340

Tabella 7: Lifetime sul flusso trasversale

Di seguito sono riportati i risultati relativi al flusso tangenziale. Si osserva che, aumentando il flusso, i valori medi ottenuti risultano più elevati.

Configuration	Mean	Variance	CI Lower	CI Upper	Margin of Error
config_0	103.5888	1.94e-01	103.3824	103.7952	2.064e-01
config_1	103.5348	1.88e-01	103.3321	103.7375	2.027e-01
config_2	97.2601	6.95e-02	97.1368	97.3835	1.234e-01
config_3	95.5353	2.26e-01	95.3127	95.7580	2.226e-01
config_4	91.5123	3.02e-01	91.2549	91.7697	2.574e-01
config_5	86.2755	3.02e-01	86.0185	86.5326	2.570e-01
config_6	85.9049	1.86e-01	85.7032	86.1065	2.017e-01
config_7	81.5475	2.94e-01	81.2936	81.8013	2.539e-01
config_8	77.3403	1.45e-01	77.1619	77.5188	1.785e-01
config_9	76.3386	2.19e-01	76.1197	76.5575	2.189e-01
config_10	73.6887	2.96e-01	73.4339	73.9435	2.548e-01
config_11	69.6426	1.34e-01	69.4712	69.8139	1.713e-01
config_12	65.9826	1.06e-01	65.8303	66.1348	1.522e-01
config_13	62.3121	1.89e-01	62.1089	62.5154	2.032e-01
config_14	59.7556	2.25e-01	59.5336	59.9775	2.220e-01
config_15	58.5429	2.83e-01	58.2942	58.7917	2.488e-01

Tabella 8: Lifetime sul flusso tangenziale

3.4 Confronto diretto sui flussi in ingresso

La figura 4 illustra l'andamento del lifetime medio per flusso in funzione delle configurazioni. Per ogni valore di L la configurazione migliore è quella con un offset di 15 secondi.

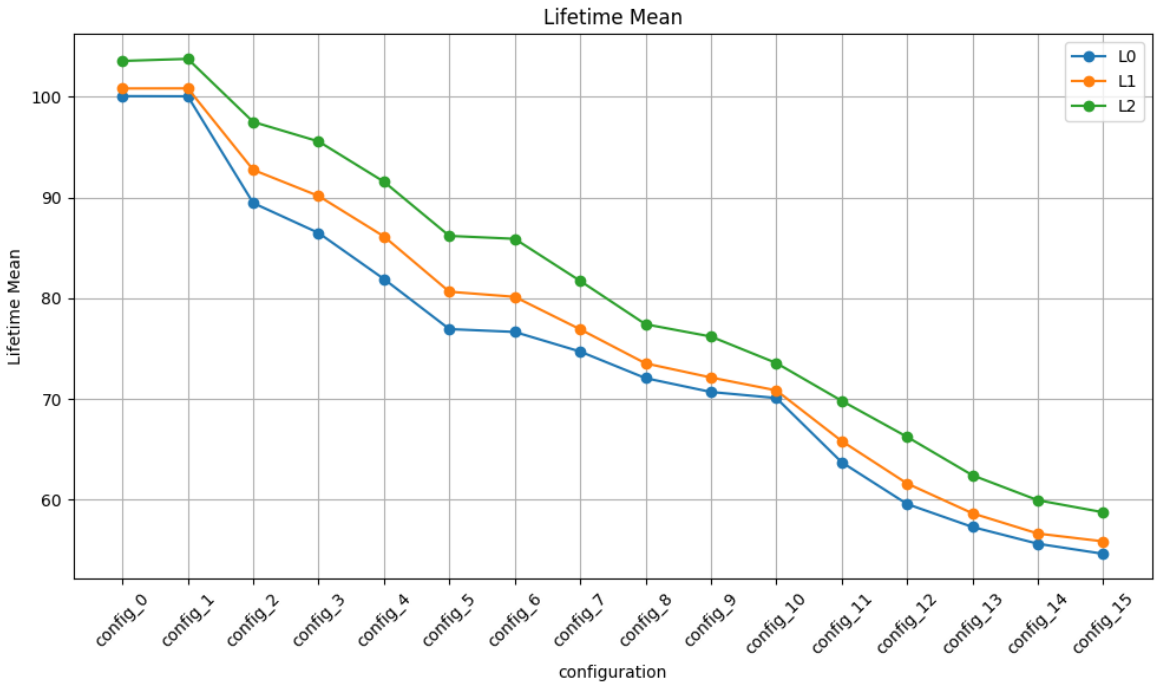


Figura 4: Lifetime mean al variare delle configurazioni sui diversi flussi

Scegliendo un flusso maggiore di L_2, come ad esempio $L_3 = 0.6$, l'utilizzazione del primo servente raggiunge il valore 1, indicando un carico critico per la rete. Per questo motivo, sono stati considerati solo i precedenti valori di L. Fissata la configurazione migliore sono stati variati diversi nuovi valori del flusso, con lo stesso numero di run e di step di simulazione. La figura 5 mostra come, superata una certa soglia, anche i valori del throughput si stabilizzino e non crescano ulteriormente. Questo comportamento indica che la rete ha raggiunto la propria capacità massima di servizio e non è in grado di gestire un flusso di arrivi maggiore con maggiore efficienza. La tabella 9. riporta i valori numerici del grafico 5

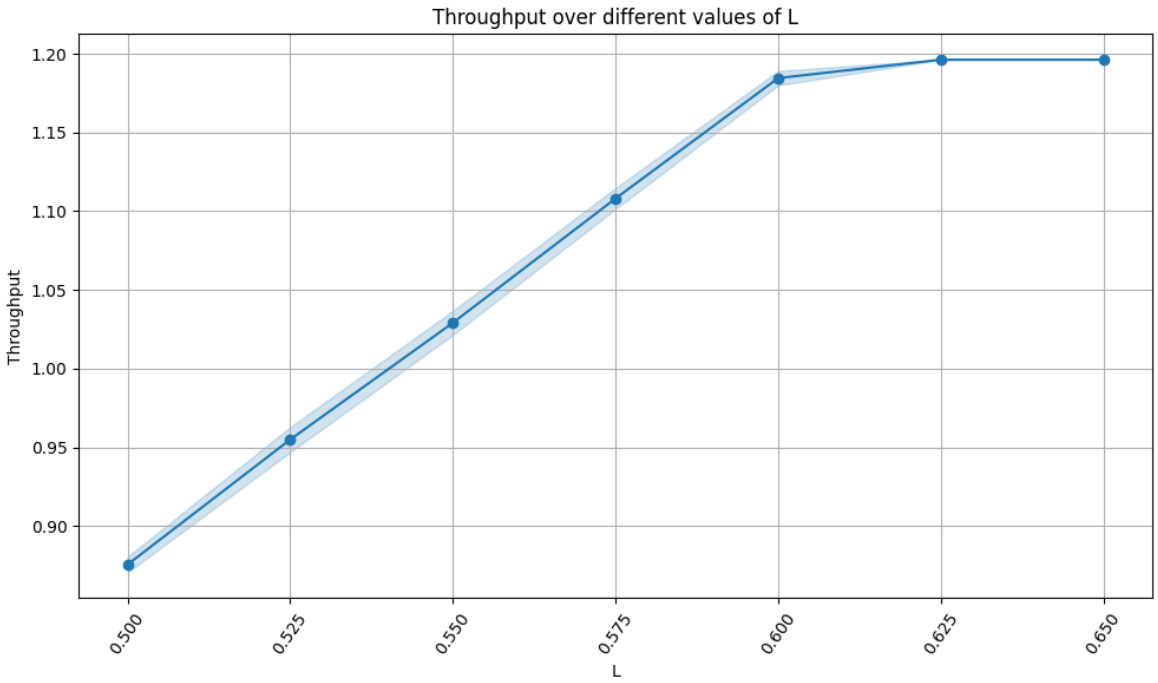


Figura 5: Throughput $offset = 15$ al variare di L

Lambda	Mean	Variance	CI Lower	CI Upper	Margin of Error
0.5	0.8756	1.13e-04	0.8706	0.8805	4.97e-03
0.525	0.9549	2.90e-04	0.9470	0.9629	7.97e-03
0.55	1.0291	2.68e-04	1.0214	1.0367	7.66e-03
0.575	1.1080	2.06e-04	1.1013	1.1147	6.71e-03
0.6	1.1845	9.59e-05	1.1799	1.1891	4.58e-03
0.625	1.1962	5.19e-32	1.1962	1.1962	1.07e-16
0.65	1.1962	5.19e-32	1.1962	1.1962	1.07e-16

Tabella 9: Risultati del throughput in funzione del parametro L

Per questo motivo si osserva anche un aumento del lifetime medio. La figura 6 mostra come all'aumentare del carico, man mano che ci si avvicina alla saturazione della rete, il lifetime cresce più rapidamente. In particolare, sono stati considerati valori di L fino a 0.575, per i quali l'utilizzazione del primo servente raggiunge il 95%.

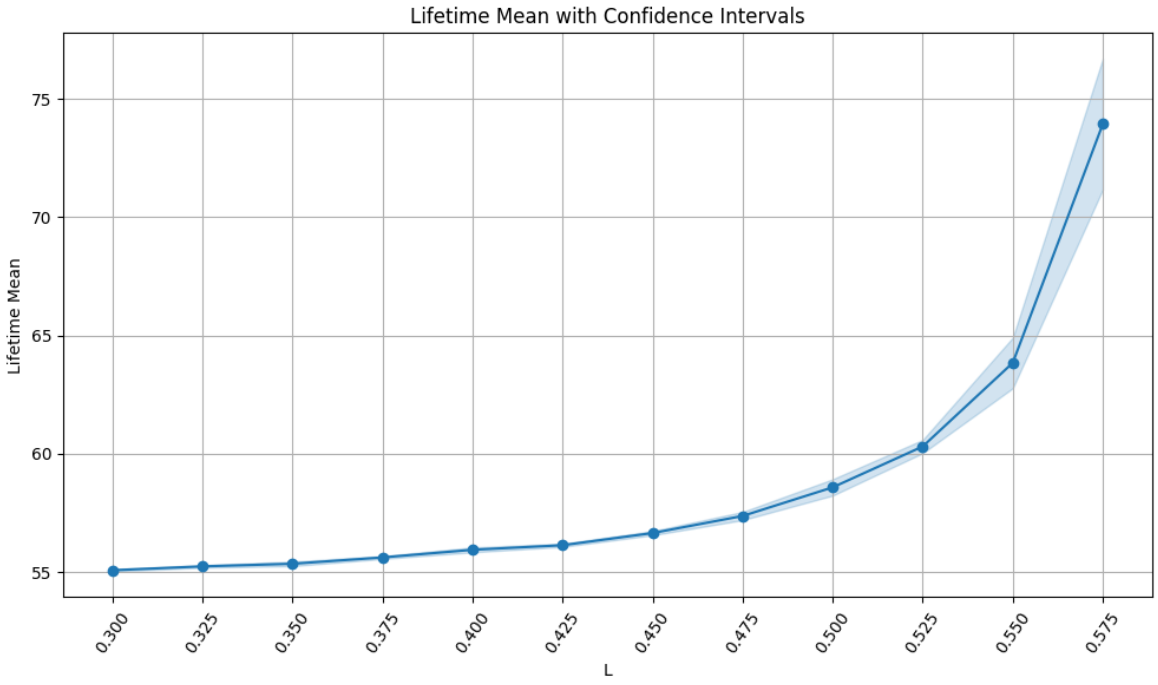


Figura 6: Lifetime mean con sincronizzazione $offset = 15$ al variare di L

4 Conclusioni

L'analisi condotta ha permesso di valutare l'impatto della sincronizzazione semaforica e della variazione del flusso di traffico sulle prestazioni della rete stradale studiata. I risultati ottenuti mostrano come l'ottimizzazione dell'offset semaforico possa ridurre significativamente il tempo medio di permanenza nel sistema per i veicoli in transito nella direzione tangenziale. In particolare, si è osservato che l'introduzione di un offset pari a 15 secondi ha favorito il fenomeno dell'onda verde, migliorando il deflusso del traffico. Questo avviene perchè 15 secondi è esattamente il tempo che un veicolo impiega a passare da un incrocio al successivo. Se un veicolo era fermo al rosso, una volta partito raggiunge il semaforo successivo esattamente quando esso diventa verde. Questo riduce il numero complessivo di fermate e quindi il tempo medio di percorrenza nella direzione tangenziale.
D'altra parte, il flusso trasversale non viene influenzato da questo fenomeno. L'analisi delle metriche di utilizzazione e del throughput hanno evidenziato che, con l'aumento del flusso in ingresso, il sistema si avvicina progressivamente alla saturazione, con conseguente aumento dei tempi di attesa e della congestione.
La sincronizzazione degli offset semaforici ha consentito ai veicoli, mantenendo una velocità costante, di attraversare gli incroci incontrando sempre la fase verde. In altre parole, l'offset di 15 secondi è stato scelto in modo tale da corrispondere al tempo necessario per percorrere la distanza tra le intersezioni, permettendo così una progressione continua del flusso.

Riferimenti bibliografici

[1] Simulation code. URL https://github.com/CiumbaSpe/Traffic_Light_Sync.

[2] Sumo documentation. URL https://sumo.dlr.de/docs/index.html.

[3] Traci documentation. URL https://sumo.dlr.de/docs/TraCI.html.