POLITECNICO
MILANO 1863
SCHOOL OF INDUSTRIAL AND
INFORMATION ENGINEERING

Mobility as a Service: Data-driven operational and economic feasibility of an autonomous urban car-sharing

Thesis of Master Degree in
Mathematical Engineering - MMF Quantitative Finance

Author: **Stefano Provera**

Student ID: 945943
Advisor: Prof. Silvia Carla Strada
Co-advisors: Prof. Sergio Matteo Savaresi, Ing. Francesco Abbracciavento, Ing. Davide Penati
Academic Year: 2021-2022

Abstract

The world of mobility is moving towards three main mega-trends: the use of electric vehicles, the development of self-driving (autonomous) vehicles and the expansion of MaaS (Mobility as a Service) models; i.e. models of shared transportation, intended as a service for citizens, and no longer based on the use of private cars. Therefore, we can expect that electric and autonomous car-sharing services will take a huge role in the mobility of the future.

This work aims to study the feasibility of an autonomous car-sharing service, both from the operational perspective and from the economic one and has the unique feature of being based on real trips data made by private vehicles, hence it is also totally data driven.

Analyzing travel habits of a medium sized urban area inhabitants, and thanks to the implementation of a realistic simulator of this electric and autonomous car-sharing service, we found that almost all urban trips made by the inhabitants could be served by a fleet of robotaxis. The ratio between the number of autonomous urban robotaxis and that of private cars, with a high level of activity in the city, results, from simulations and optimization based on real data, to be 1:9. This means that trips of 9 private cars can be covered, almost entirely, by a single electric and autonomous vehicle. From this, a significant reduction of the parking areas and the possibility of dismissing one's owned car.

Carrying out simulations of this car-sharing service, we could also make considerations on the minimum fares to apply in order to reach the economic break-even. These fares vary between 0.19€/km and 0.21€/km, or similarly, between 0.07€/min and 0.08€/min, depending on the dimension of the simulated service. These prices would already be competitive compared to other shared mobility services, such as public transportation, and highly cheaper than traditional car-sharing services.

Furthermore, we observed that the efficiency of the service increases as the number of people joining the service increases. This leads, also, to lower costs for the user.

It is, therefore, desirable that as many people as possible switch to a form of shared mobility, like the one presented in this work.

Sommario

Il mondo della mobilità si sta muovendo verso tre mega-trend principali: l'utilizzo di veicoli elettrici, lo sviluppo di veicoli a guida autonoma e la diffusione di modelli "a servizio", cioè sistemi integrati di trasporto condiviso, intesi come servizio per i cittadini, e non più basati sull'uso di veicoli privati. Quindi, una tipologia di trasporto che è lecito attendersi prenderà il sopravvento in futuro, è quella del car-sharing elettrico ed autonomo.
Questo progetto di tesi studia la fattibilità di un servizio di car-sharing autonomo ed elettrico, sia dal punto di vista dell'efficienza del servizio, sia per quanto riguarda la fattibilità economica, e ha la caratteristica unica di essere totalmente basato su dati di percorrenza reali di veicoli privati.
Studiando le abitudini di percorrenza degli abitanti di un'area urbana di medie dimensioni, e grazie alla costruzione di un realistico simulatore di questo servizio di car-sharing elettrico ed autonomo, si è trovato che quasi tutti i viaggi urbani potrebbero essere serviti da una flotta di robotaxi. Il rapporto tra il numero di robotaxi e quello del numero di auto private, che hanno un alto livello di attività in città, risulta essere di 1:9 (risultato ottenuto a partire da simulazioni e ottimizzazioni basate su dati reali). Questo significa che i viaggi effettuati da 9 veicoli privati possono essere serviti quasi completamente da un singolo veicolo autonomo ed elettrico. Tutto ciò comporterebbe un significativo calo di aree dedicate ai parcheggi e la possibilià di dismettere la propria autovettura.
Svolgendo simulazioni di tale servizio di car-sharing, è stato possibile fare considerazioni sulle tariffe minime da attuare per raggiungere il break-even economico. Queste tariffe variano tra i 0.19-0.21 €/km, oppure tra i 0.07-0.08 €/min, a seconda della dimensione del servizio simulato. I prezzi sarebbero già competitivi rispetto ad altri servizi di mobilità condivisa, come i mezzi pubblici, e altamente più convenienti rispetto ai servizi di car-sharing tradizionali.
Inoltre si è osservato come più utenti usufruiscano del servizio di car-sharing elettrico

ed autonomo, migliore sia l'efficienza del servizio stesso. Tutto ciò comporta anche dei costi finali minori per l'utente. È, perciò, auspicabile che quante più persone possibili passino a una forma di mobilità condivisa.

Ringraziamenti

Durante il lavoro di tesi ho ricevuto un grande aiuto e supporto.
Vorrei inizialmente ringraziare la Prof.ssa Silvia Carla Strada, che, in virtù della sua esperienza e conoscenza nel settore, è stata una guida fondamentale nello sviluppo dei punti chiave di questo studio.
In secondo luogo vorrei ringraziare tutti coloro che hanno partecipato e hanno collaborato attivamente per la buona riuscita del progetto, quali il Prof. Sergio Matteo Savaresi, l'Ing. Abbracciavento, l'Ing. Penati e tutto il gruppo di ricerca MOVE.
La mia famiglia, inoltre, è stata un supporto importantissimo durante il periodo di tesi e in generale durante tutto il periodo universitario; non potrò mai ringraziarli a sufficienza.

Un ringraziamento speciale va a tutti i miei amici, che mi hanno aiutato nei momenti difficili e con cui ho condiviso tantissime belle esperienze: Albi, Dani e Lomba in primis, poi anche Ila, tutte le "persone perbene", quelli di "G4 gang" e "di nuovo ten".
Vorrei citare anche i miei amici da una vita, quali Gabry, Ario, Luca e Francesca, che mi hanno visto crescere e maturare come persona fin da quando ero un marmocchio incazzoso.
Vorrei inoltre ringraziare stemix, il mio alter ego, che ha portato un po' di creatività ed estro nel mondo rigido e logico di un ingegnere matematico.
Infine vorrei fare un ringraziamento alla musica in generale, unica ancora di salvezza in momenti cupi e bui; e in particolare ai ROS, band che ho potuto conoscere personalmente, e che da voce a disagi, emozioni e urgenze di una generazione, utilizzando parole e accordi che sento come se fossero miei.

Elenco delle figure

Elenco delle tabelle

Indice

1 | Introduzione

La maggior parte della popolazione mondiale vive attualmente in un ambiente urbano e si prevede che questo numero continuerà ad aumentare fino a toccare percentualemente l'80% nel 2050 [1].
La gestione dei trasporti e del traffico all'interno delle città è, di conseguenza, sempre più una sfida cruciale per il benessere e lo sviluppo economico della società.
I veicoli privati costituiscono ancora uno dei principali metodi di trasporto anche all'interno delle città, e sono tra le cause maggiori di traffico e inquinamento.
Il mondo della mobilità è responsabile, inoltre, di circa un quarto delle emissioni antropogeniche di gas serra.
Per rispondere a queste enormi criticità, il mondo della mobilità si sta muovendo verso tre principali direzioni o mega-trends, [2]:

- stimolo al passaggio ai veicoli elettrici
- sviluppo dei veicoli a guida autonoma "intelligenti"
- sviluppo di modelli "a servizio" (MaaS - Mobility as a Service) che trasformino l'auto da un bene di proprietà e di uso privato, ad uno condiviso, eventualmente in modo esclusivo.

Tutto questo si traduce nella necessità di sviluppare sistemi di mobilità intelligente e condivisa, per esempio sistemi di car-sharing elettrici e a guida autonoma, analizzandone la fattibilità e validandone le prestazioni, meglio se sulla base di dati (misurati ed oggettivi) della mobilità privata attuale.

La diffusione di una metodologia di trasporto condivisa può modificare in modo significativo il tasso di proprietà dei veicoli (ovvero il rapporto tra il numero dei veicoli rispetto al numero di persone), riducendo il numero di veicoli esistenti di un fattore di 8-12, dal momento che la maggior parte delle auto private vengono utilizzate meno

del 10% del tempo, [3], inoltre può anche migliorare il funzionamento del settore dei trasporti, poichè un elevato chilometraggio annuale dei veicoli crea un forte incentivo verso l'impiego di veicoli oggi costosi, ma puliti ed altamente efficienti, [4]. Servizi di trasporto condiviso dovrebbero essere anche economicamente più convenienti del trasporto privato, [4].
L'avvento della tecnologia a guida autonoma andrà ulteriormente ad accelerare l'adozione di questa modalità di trasporto, rendendola ancor più conveniente: i veicoli possono spostarsi per prelevare i clienti in autonomia (robotaxi), un servizio paragonabile a quello di un taxi senza il costo del conducente. La popolarità di servizi relativamente costosi (ma comunque più economici dei taxi tradizionali) come Uber, mostra il potenziale di questo tipo di modalità di trasporto, ovvero come con prezzi bassi la gente sia maggiormente incentivata verso questo tipo di mobilità.
Vantaggi di un sistema di car-sharing che utilizza veicoli a guida autonoma includono la migliore efficienza di guida, che porta a consumi inferiori di circa il 4/10%, eliminazione del tempo impiegato nella ricerca di parcheggio e un bisogno minore di aree di parcheggio nelle città, [5]. Inoltre si potrebbe utilizzare un veicolo adatto per ogni tipo di viaggio, andando a migliorare le prestazioni del servizio. Infine i veicoli a guida autonoma possono facilitare il processo di elettrificazione del settore dei servizi, in quanto i veicoli sono in grado di ottimizzare il loro state of charge (SOC) e il loro orario di ricarica, assicurando così un servizio affidabile per gli utenti.

1.1. Motivazione della tesi

In questo progetto di tesi viene effettuato uno studio pilota che cerca di unire i tre mega-trends precedentemente elencati, descrivendo e ottimizzando la fattibilità (sia in termini di efficienza del servizio che in termini economici) di un servizio di Car Sharing Elettrico, a Guida Autonoma e di tipo Free Floating (i veicoli possono essere noleggiati e rilasciati in qualsiasi punto all'interno di un'area designata). A differenza dello stato dell'arte in tale ambito, la forza di questo progetto è quella di essere basato totalmente su dati reali di percorrenza di veicoli privati.

Grazie alla collaborazione con Unipol, una delle più grande compagnie assicurative in Italia, si è avuto accesso a un enorme dataset contentente tutti i viaggi effettuati in 2 anni da una porzione importante di popolazione residente nella provincia di Brescia.

Questo ha abilitato uno studio completamente data-driven (cioè basato esclusivamente su dati), che, per sua natura, ha consentito di fare davvero poche assunzioni, così da avere risultati molto realistici e convincenti. In tal modo si è potuto rispondere alla seguente domanda, cardine di tutta questa ricerca:
Sarebbe fattibile, sia in termini di soddisfacimento della domanda di mobilità, che in termini economici, allo stato attuale, sostituire i veicoli privati con un servizio di car-sharing autonomo ed elettrico?
In caso di risposta affermativa:
Per quanti utenti sarebbe conveniente utilizzare un servizio di mobilità urbano condiviso, anzichè avere una macchina privata, e quindi dismettere quest'ultima?

Le due sfide principali per rispondere a queste domande sono state:

1. **Costruzione di un simulatore della mobilità privata urbana nell'area in studio.** Costruire un tool di simulazione in grado di analizzare una così grande mole di dati, simulando totalmente un servizio di car sharing coerente con essi, gestendo in maniera ottimale le chiamate, l'allocazione del parco auto e tutta la parte dovuta alla ricarica dei veicoli elettrici.
2. **Ottimizzazione del servizio di Sharing.** Studio dei metodi per rendere il più efficiente possibile il servizio di car-sharing, indivuando il valore ottimale delle variabili di input (numero di veicoli, numero di stazioni di ricarica e area del servizio), affinchè il servizio sia il più conveniente possibile dal punto di vista economico, lato gestore del servizio e lato cliente.

1.2. Risultati principali

Tutti i risultati ottenuti in questo lavoro possono essere così riassunti:

- Analizzando i dati di percorrenza di veicoli realmente presenti nella provincia di Brescia, si è potuto osservare come la migliore collocazione per una ipotetica area di servizio di Car Sharing sia una regione (circolare in questo studio) centrata nel centro storico di Brescia.
  Inoltre si è potuto constatare come un raggio ottimale per l'area del servizio di

Sharing sia compreso tra gli 8km e i 12km; in questa area circa il 10% dei veicoli presenti (valore dipendenete dalla grandezza della regione circolare) vengono considerati compatibili con lo sharing (ovvero almeno il 90% dei loro viaggi è effettuato all'interno dell'area considerata, cioè sono viaggi urbani).

- Considerando la distribuzione durante il giorno dei viaggi degli utenti sharing compatibili e attraverso opportune verifiche con il simulatore, si è trovato come il metodo più efficiente per utilizzare il maggior tempo possibile i veicoli sia impiegare politiche di ricarica differenziate per il giorno (7-23) e la notte (23-7).
- A seguito dell'ottimizzazione dal punto vista economico dei parametri per il servizio di sharing (numero veicoli presenti, numero di colonnine e raggio dell'area circolare del servizio) si è trovato come il raggio ottimale sia di 9km, utilizzando 2804 veicoli e con 986 stazioni di ricarica. Ipotizzando che un gestore voglia effettivamente offrire un servizio di sharing, per raggiungere il break-even e non essere quindi in perdita, è necessario stabilire una tariffa minima di 0.074 €/min, oppure di 0.201 €/km.
- Attraverso un'analisi dinamica degli utenti coinvolti nel servizio di sharing, si è trovato che, se anche solo la metà degli utenti sharing compatibili usassero effettivamente tale servizio, la tariffa minima lieviterebbe del 50% circa.
- Ipotizzando una tariffa di 0.30€/km, per il 95% degli utenti sharing compatibili sarebbe economicamente conveniente utilizzare tale servizio, invece di possedere un veicolo di proprietà (più è basso il chilometraggio annuo individuale, più è conveniente utilizzare veicoli in sharing).

1.3. Struttura della tesi

Il lavoro effettuato in questo progetto di tesi si articola nelle seguenti parti:

- Nel capitolo 2 vengono brevemente descritti gli studi correlati con l'oggetto della tesi e da cui si è partiti per sviluppare questo progetto di ricerca.
- Nel capitolo 3 vengono presentati i dati di percorrenza telematici dei veicoli privati, sui quali è basato il progetto (quindi il dataset contentente i viaggi realmente effettuati da veicoli in provincia di Brescia tra l'inizio del 2019 e la fine del 2020) e il modo in cui questi si sono ricavati dalle autovetture. Successivamente sono

spiegati i metodi attraverso cui i dati sono stati puliti e aggregati in maniera da essere utilizzabili per gli scopi di questa ricerca.

- Il capitolo 4 comincia con le prime analisi a partire dal dataset costruito nella sezione precedente. In particolare viene effettuato lo studio su come trovare la regione ottimale del servizio di sharing (collocazione e una prima stima sull'area) e in seguito vengono definiti e individuati gli utenti sharing compatibili (ovvero gli utenti target del nostro studio, cioè coloro i quali "viaggiano spesso" all'interno di una piccola area circoscritta e quindi sarebbero adatti a utilizzare un servizio di sharing). Successivamente si prosegue con uno studio riguardante la densità abitativa e il numero di automobili immatricolate all'interno provincia, in maniera tale da ottenere il rapporto tra veicoli nel dataset e quelli realmente esistenti nell'area considerata.

- Nel capitolo 5 viene presentata la struttura base del simulatore di car-sharing urbano autonomo, evidenziando tutte le assunzioni fatte e i meccanismi alla base del processo di simulazione. In seguito si procede con una analisi sulla convergenza del simulatore, ovvero per quanto tempo è necessario simulare affinchè la variabili output del servizio di sharing (per esempio la percentuale di chiamate perse, l'overhead, ecc) convergano a un valore costante.

- Il capitolo 6 tratta tutta la parte dello studio di tesi riguardante la mobilità elettrica del servizio di sharing autonomo. In primo luogo vengono definiti i veicoli e le stazioni di ricarica utilizzate nel processo di simulazione, e viene spiegata la politica utilizzata per ricaricare i veicoli. Successivamente si procede con l'ottimizzazione pre-simulazione di questa parte elettrica; ovvero lo studio sul posizionamento ottimale delle stazioni di ricarica all'interno della regione di copertura del servizio e l'analisi sulle politiche di ricarica che garantiscano la migliore efficienza del servizio.

- Nel capitolo 7 viene definito il problema generale di ottimizzazione: si tratta di un problema di minimizzazione dei costi, con vincoli che garantiscano una minima efficienza del servizio. Poichè le variabili utilizzate nel problema sono output di un processo di simulazione basato su dati reali, la funzione obiettivo è estremamente non lineare e quindi gli algoritmi classici per raggiungere convergenza non vi si adattano. Si procede dunque con una analisi dei vari metodi di convergenza utilizzabili in questo contesto specifico.

- Nel capitolo 8 vengono riassunti tutti i risultati ottenuti nel problema di ottimizzazione, inizialmente utilizzando un raggio compreso tra 8km e 12km e successivamente anche un raggio lievemente maggiore per poter verificare le stime di area ottimale di copertura a cui si era giunti nel capitolo 4. Successivamente viene svolto uno studio più approfondito sui risultati finali (output del processo di ottimizzazione), sia dal punto di vista economico che dell'efficienza del servizio stesso.
- Nel capitolo 9 vengono effettuate ulteriori analisi a posteriori sui risultati ottenuti, in particolare una analisi dinamica di sensitività sul numero di persone coinvolte nel servizio (ossia come cambiano le variabili output della simulazione, qualora un numero inferiore del previsto di utenti sharing compatibili utilizzi effettivamente il servizio) e una analisi sui costi/benefici garantiti se gli utenti sharing compatibili usassero tale servizio di sharing, invece di possedere un'automobile.

2 | Stato dell'arte nell'ambito del car-sharing elettrico ed autonomo

Il mondo della mobilità è in continuo cambiamento, specialmente nell'ultimo periodo storico: veicoli elettrici, a guida autonoma e sviluppo di sistemi a servizio sono sempre più oggetto di studio e ricerche.
Per lo sviluppo di questo progetto di tesi si sono analizzate in primo luogo le pubblicazioni che proponessero soluzioni per efficientare al massimo un servizio di car-sharing elettrico e condiviso, e in secondo luogo quelli riguardanti la parte di ottimizzazione del servizio dal punto di vista economico.

2.1. Efficientamento del servizio

Cocca et al. nel loro progetto di ricerca, [6], [7], [8], studiano il collocamento ottimale delle stazioni di ricarica per veicoli elettrici e la politica di restituzione dei veicoli, a partire da dati di percorrenza di servizi di sharing di tipo free-floating; i loro studi mostrano come uno dei metodi più efficienti sia posizionare le stazioni di ricarica dove finiscono più spesso i viaggi degli utenti (dove quindi i veicoli sono spesso parcheggiati, ma per poco tempo).
Altre ricerce si concentrano sui parametri di valutazione di un servizio di car-sharing autonomo di tipo free-floating, [9], in questo caso a partire dal servizio viene studiata la performance principalmente attraverso le seguenti metriche: dimensione della flotta dei veicoli, chilometraggio dei veicoli, tempo di attesa medio, e percentuale di viaggi serviti (o analogamente la percentuale di chiamate perse).
Javanshour et al., [10], a partire da avanzati simulatori di traffico stradale, simulano un servizio di car-sharing autonomo di tipo on-demand (un servizio analogo a quel-

lo oggetto di questo studio, eccetto che per le mobilità non di tipo elettrico); il loro obiettivo è studiare il ricollocamento ottimale dei veicoli durante l'arco della giornata in maniera da essere disponibili più velocemente per gli utenti; in particolare mostrano come un continuo riposizionamento dei veicoli, sebbene permetta di servire più clienti, non sia per nulla positivo in termini di congestione del traffico e causi un aumento totale dei chilometri percorsi dai veicoli (VKT) di circa il 77%. Questo tipo di strategia non è incluso nel seguente lavoro. Inoltre, analizzando la variabilità dell'eVKT(empty vehicle kilometres travelled- chilometraggio effettuato dei veicoli senza utenti a bordo), mostrano come questo valore diminuisca aumentando la dimensione della flotta.
Miao et al., [11], nel loro studio presentano un metodo di ottimizzazione in 2 step per l'efficientamento di un servizio di car-sharing autonomo e elettrico: il primo passo consiste nel trovare la collocazione ottimale della regione di copertura del servizio e nel secondo step vengono utilizzati algoritmi genetici per il posizionamento ottimale di tutte le infrastrutture di ricarica. (gli algortimi genetici sono dei particolari algoritmi euristici utilizzati per risolvere specifici problemi di ottimizzazione)
Xiang Huo et al., [12], portano avanti uno studio sulla allocazione ottimale dei veicoli elettrici nelle varie stazioni di ricarica. Questo è uno dei pochi studi ad essere data-driven, in particolare si basa su dati raccolti da un servizio di car-sharing elettrico esistente nella città di Pechino. Lo studio mostra come durante il corso della giornata ci siano picchi/minimi di richieste di veicoli e che quindi sia necessario frazionare la giornata e avere politiche di allocazione/ricarica differenziate a seconda dell'ora del giorno in cui ci si trova. Attraverso politiche più efficienti sostengono come si possa avere un aumento nei profitti giornalieri di circa il 10%.

2.2. Analisi economica

Iacubbucci et al., [13], nel loro lavoro studiano la fattiblità di un sistema di car-sharing autonomo ed elettrico, sia dal punto di vista della qualità del servizio che per quanto riguarda la performance economica, inoltre analizzano nel dettaglio l'interazione del sistema di car-sharing con la rete elettrica. Sebbene lo studio si basi su dati di viaggi ottenuti attraverso sondaggi e non su dati di percorrenza reali, nel loro case study ottengo risultati in linea con quelli avuti in questo lavoro di tesi; in particolare ottengono tariffe minime del servizio di circa 0.22-0.32 USD/km , e mostrano come si potrebbe ridurre il numero di veicoli in circolazione di un fattore 7-10.

In uno studio condotto nella città di Zurigo, [14], viene svolta un'analisi puramente finanziaria della fattiblità di un servizio di car-sharing a guida autonoma ed elettrico. La ricerca, basata su un simulatore di traffico del cantone di Zurigo, trova come la tariffa minima del servizio sia di circa 0.49-0.42 CHF/km, in base alla dimensione del parco auto; inoltre mostra come i costi del servizio tendano a diminuire all'aumentare della grandezza della flotta, sia perchè si hanno sconti maggiori all'acquisto della flotta, sia perchè il servizio diventa più efficiente.

Il presente progetto di tesi si propone inizialmente di armonizzare i metodi principali utilizzati nelle varie ricerche per fornire un servizio di sharing efficiente, e in seguito di ottimizzare in termini economici rispetto ai parametri in input del servizio (quali la dimensione, il numero di veicoli in sharing e il numero di stazioni di ricarica).
Ciò che contraddistingue questo lavoro da tutti quelli esistenti è il fatto di essere basato esclusivamente su dati di percorrenza di veicoli privati. La maggior parte degli studi in questo ambito si basa, infatti, su simulatori di traffico o dati di sondaggi, mentre la restante piccola parte è anch'essa costituita da ricerche "data-driven", ma a partire da dati di percorrenza di veicoli in sharing e non privati. Inoltre il dataset utilizato per questo progetto comprende più di cento milioni di viaggi, mentre gli altri studi "data-driven" si basano, al massimo, su centinaia di migliaia viaggi, garantendo quindi una ben minore significatività statistica e spazio/temporale.

3 | Il dataset telematico

Grazie alla collaborazione con UnipolSai, per questo progetto abbiamo potuto avere a disposizione un dataset contenente circa 115 milioni di viaggi effettuati tra Gennaio 2019 e Dicembre 2020 da 33170 veicoli immatricolati nella provincia di Brescia.
Tutti questi eventi sono stati raccolti in forma anonima da sensori GPS (comunemente denominati e-box) montati all'interno dei veicoli assicurati. Queste e-box, attive con continuità nel tempo, raccolgono costantemente dati sul comportamento del veicolo e periodicamente inviano i dati di percorrenza ai server Unipol.

3.1. Definizione dei viaggi

Il dataset consiste in un insieme di viaggi effettuati da diversi veicoli.
Ogni viaggio è caratterizzato dai seguenti parametri:

- **ID_driver** Codice (numerico) utilizzato per identificare il veicolo che ha raccolto i dati su questo viaggio
- **Latitude_start** Dato GPS sulla latitudine del punto di partenza del viaggio considerato
- **Longitude_start** Dato GPS sulla longitudine del punto di partenza del viaggio considerato
- **Latitude_end** Dato GPS sulla latitudine del punto di arrivo del viaggio considerato
- **Longitude_end** Dato GPS sulla longitudine del punto di arrivo del viaggio considerato
- **Distance** Distanza, espressa in metri, del viaggio considerato

- **Stop_duration** Durata temporale, espressa in secondi, tra la fine del viaggio precendente effettuato da questo veicolo e l'inizio di questo viaggio
- **Trip_duration** Durata temporale, espressa in secondi, del viaggio considerato
- **Time_start** Orario di inizio del viaggio (Data, Ora, Minuti e Secondi)
- **Time_end** Orario di termine del viaggio

In seguito, in figura 3.1, si trova l'esempio di un viaggio presente nel dataset, index rappresenta il codice identificativo del viaggio

index	id_driver	latitude_start	longitude_start	latitude_end	longitude_end
1336530	33532	45.545379	10.219446	45.559111	10.191827

distance	stop_duration	trip_duration	time_start	time_end
3.311	15379	620	2019-01-07 00:00:11	2019-01-07 00:10:31

Figura 3.1: Esempio di viaggio presente nel dataset, associato al veicolo con ID 33532

3.2. Pulizia dei dati e pre-elaborazione

I dati sui viaggi sono ottenuti aggregando eventi di tipo primario (raw events)(la cui descrizione è oltre gli scopi di questo lavoro) e i sensori GPS non sempre sono precisi, perciò alcuni viaggi non sono effetivamente affidabili e utilizzabili per la seguente ricerca e devono essere rimossi.
In particolare si procede con il rimuovere i viaggi con le seguenti caratteristiche:

- **Viaggi a distanza 0** Tutti i viaggi caratterizzati dall'avere una distanza uguale a 0 ovviamente non sono rilevanti e sono errori dovuti ad un accorpamento non corretto dei "raw events"
- **Viaggi troppo lenti o troppo veloci** I viaggi con una velocità media inferiore ai 5km/h oppure superiore ai 150km/h
- **Viaggi con un tempo di percorrenza eccessivamente alto o basso** I viaggi con una durata inferiore ai 2 minuti, oppure aventi una durata maggiore di 16 ore.
- **Viaggi non coerenti** I viaggi la cui durata (Trip_duration) non è uguale alla differenza temporale tra Time_start e Time_end (considerando comunque

un minimo di tolleranza (20%) dovuta all'incertezza nella creazione dei viaggi). In particolare quei viaggi che non rispettano la seguente condizione:

$$Trip\_duration \cdot 0.8 < (Time\_end - Time\_start) < Trip\_duration \cdot 1.2$$

(Time_end-Time_start) viene convertito in secondi per effettuare il confronto

Inoltre si procede con l'individuazione di tutti gli utenti poco affidabili (coloro i quali hanno più del 40% dei viaggi che rispecchiano almeno una delle condizioni sopra elencate) e la successiva rimozione di tutti i viaggi da loro effettuati.
Il risultato finale è un dataset comprendente circa 74 milioni di viaggi effettuati da 27777 utenti.

Figura 3.2: Dataset pre e post pulizia

4 | Identificazione degli utenti target del car-sharing urbano

A valle del processo di data cleaning, come descritto nel capitolo precedente, si può cominciare lo studio dell'applicazione specifica di questo progetto di tesi.
Il primo passo è quello di studiare il comportamento degli utenti presenti all'interno del dataset, in maniera tale da identificare una collocazione e una dimensione ottimale dell'area di copertura del servizio di car-sharing urbano.

4.1. Brescia case study: selezione dell'area di copertura del servizio di car-sharing

Come nel caso di un servizio di sharing di tipo free-floating tradizionale, l'area di copertura del servizio delimita quella regione entro cui i veicoli possono essere noleggiati e devono essere rilasciati alla fine del viaggio. Il viaggio stesso può in parte essere effettuato anche all'esterno dell'area di copertura, l'importante è che il punto di partenza e il punto di arrivo siano all'interno dell'area considerata.
Il primo step è identificare il centro del servizio. Si procede analizzando la distribuzione delle partenze dei viaggi presenti nel dataset: dalla heatmap in Fig. 4.1 è chiaro come l'area in cui vengono effettuati la maggior parte dei viaggi sia quella della città di Brescia (come era lecito attendersi data la maggior densità abitativa). Il centro del servizio di car-sharing è quindi fissato nel centro storico di Brescia, in particolare nelle coordinate del Palazzo del Broletto (45.53936 N, 10.22218 E).
Poichè la zona con la maggior densità di viaggi ha una forma tendenzialmente circolare, così come l'area che delimita il comune di Brescia si è optato per definire un'area di copertura circolare del servizio di car-sharing.
Il raggio dell'area del servizio di sharing rimane ancora un grado di libertà, ma si può

cominciare a inviduarne il limite massimo. Come enunciato nel capitolo precedente, i viaggi nel dataset sono stati effettuati da cittadini residenti nella provincia di Brescia, quindi non avrebbe senso considerare un'area di copertura del servizio oltre i confini della provincia. Si è trovato così il limite superiore per il raggio, che è di 26km. In Fig. 4.2 viene evidenziata la collocazione della area di copertura, fino al raggio massimo possibile del servizio.

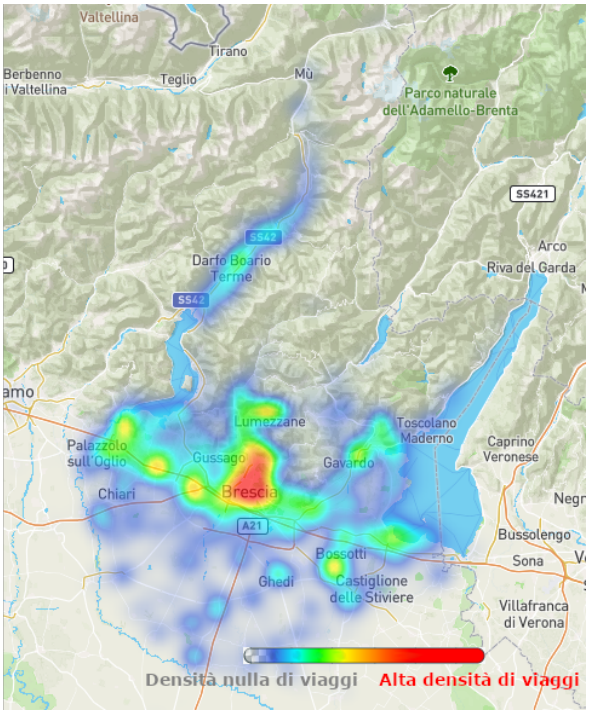


Figura 4.1: Heatmap rappresentante la densità di partenze dei viaggi nel dataset

Figura 4.2: Posizionamento della area di copertura del servizio

4.1.1. Criteri di selezione degli utenti compatibili con lo sharing

Definita l'area ottimale del servizio, anche se con un raggio ancora variabile, è bene individuare quali utenti presenti all'interno dell'area abbiano determinate abitudini di percorrenza compatibili con un servizio di sharing.
Un utente viene definito "sharing-compatibile" se rispecchia le seguenti condizioni:

- Il domicilio dell'utente è all'interno dell'area di copertura del servizio

- Almeno il 90% dei viaggi di tale utente è effettuato all'interno dell'area di copertura del servizio

Essendo, per la privacy, il dataset anonimizzato, i dati sulla posizione della casa dell'utente sono stati ottenuti aggregando tutti i viaggi effettuati da questo specifico utente e verificando l'esistenza di un punto di partenza/arrivo frequente.
Per viaggio effettuato all'interno dell'area si intende che i punti di inizio e fine del viaggio sono all'interno di tale regione di copertura, ammettendo temporanee uscite dall'area del servizio.

4.1.2. Stima della dimensione dell'area di copertura

Si procede quindi a studiare la variabilità del numero di utenti target in funzione dell'area del servizio, in maniera tale da fare considerazioni utili per dare una prima stima sulla dimensione del servizio.

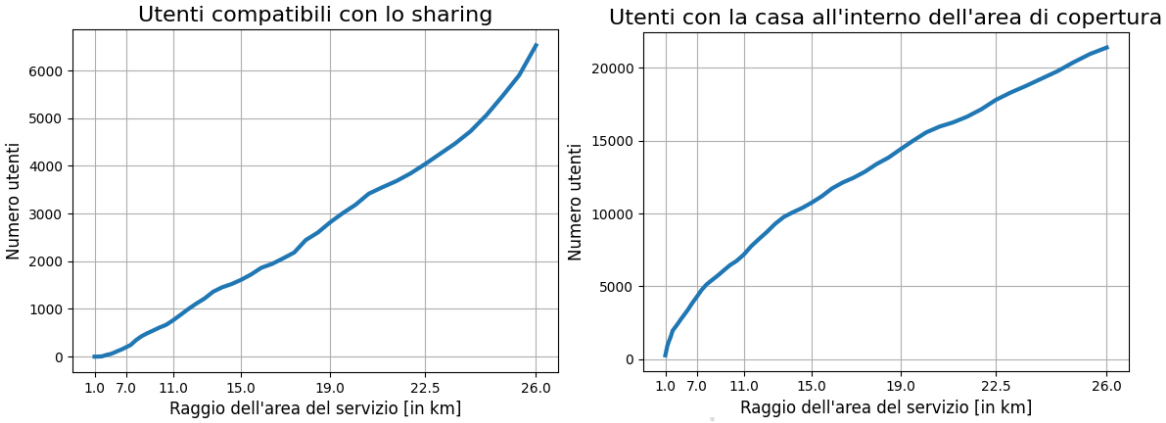


Figura 4.3: Analisi di sensitività degli utenti coinvolti in funzione del raggio del servizio

Come intuibile e come si nota in Fig. 4.3, il numero di utenti target cresce all'aumentare del raggio del servizio. Poichè questi grafici da soli non forniscono informazioni sufficienti per poter dare una prima stima sulla dimensione dell'area si è deciso di analizzarne anche la derivata. Il valore della derivata è calcolato come un semplice rapporto incrementale (rapporto incrementale rispetto all'area del servizio, quindi rispetto al raggio elevato al quadrato).

$$V(r) = \frac{N(r + \Delta r) - N(r)}{\pi \cdot \Delta r^2} \tag{4.1}$$

V(r) rappresenta il valore della derivata, mentre N(r) rappresenta il numero di utenti.

Il grafico riferito al numero di utenti compatibili con lo sharing (Fig. 4.4 a sinistra) si mantiene abbastanza costante (con numerose oscillazioni), mentre per quello riferito al numero di utenti con la casa all'interno dell'area di copertura del servizio (Fig. 4.4 a destra) si nota come ci sia un drastico calo della derivata nei primi chilometri che porta poi a una discesa più graduale.

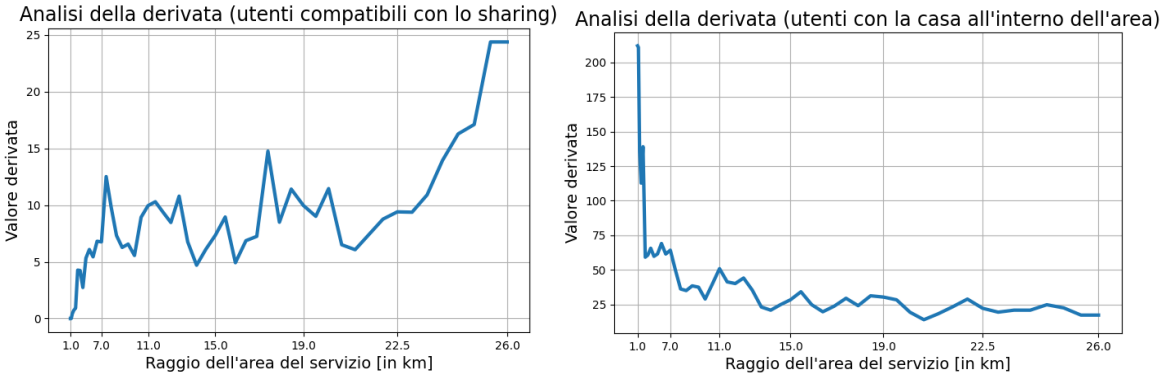


Figura 4.4: Analisi della derivata del numero di utenti in funzione del raggio dell'area di copertura

Si è deciso quindi di optare per un raggio compreso tra i 6 e i 15km, per i seguenti motivi.

- Il grafico della derivata nel caso dell'analisi degli utenti con la casa all'interno dell'area mostra come dopo 15 km il valore della derivata si abbassi ulteriormente e quindi si vadano ad considerare progressivamente sempre meno utenti.
- I due grafici riguardanti il numero di utenti sharing compatibili indicano come sia infattibile creare un servizio unicamente entro il perimetro della città (meno di 6km di raggio), in quanto troppi pochi utenti sarebbero considerabili sharing compatibili (si nota bene come il valore della derivata sia estremamente basso prima dei 6km). Quindi è necessario considerare un raggio maggiore che includa, almeno in parte, anche l'area metropolitana.
- La heatmap 4.1 mostra come la concentrazione maggiore di viaggi sia entro un raggio di circa 8km dal centro di Brescia.

Una stima più accurata di questo parametro (il raggio del servizio) verrà presentata nei prossimi capitoli.

4.2. Brescia "case study": analisi del centro urbano ed dell'area metropolitana

Si procede, dunque, analizzando in dettaglio l'area compresa tra i 6km di raggio e i 15km.
In primo luogo si può fornire una stima sulla precentuale di utenti target in funzione del raggio. Fissato un raggio, il valore della percentuale è semplicemente stimato come numero di utenti compatibili con lo sharing fratto il numero di utenti che abitano all'interno dell'area. Come si può notare in Fig. 4.5, questo valore varia da un minimo di 3% a un massimo del 15%.

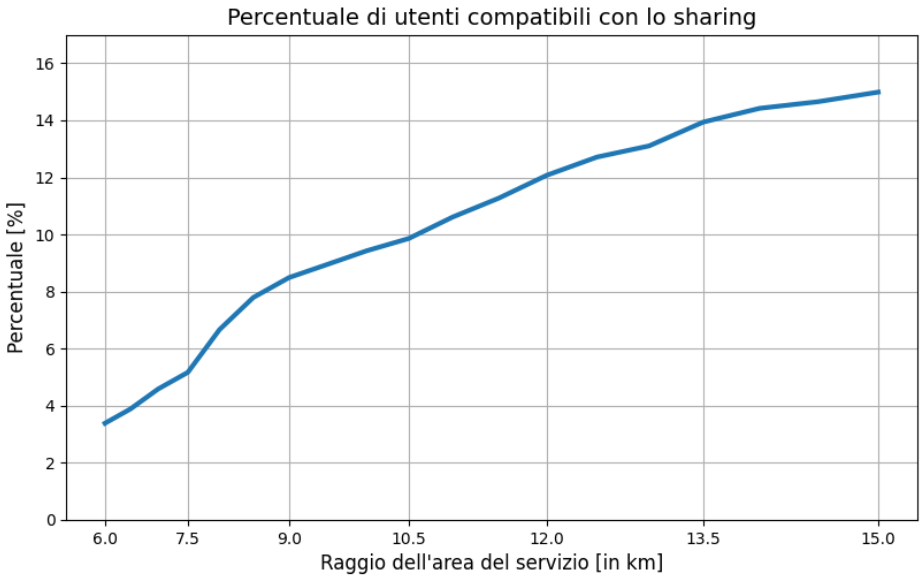


Figura 4.5: Percentuale di utenti compatibili lo sharing in funzione del raggio dell'area del servizio

Questo grafico ci suggerisce inoltre come per un raggio inferiore agli 8km la percentuale di utenti target compatibili con lo sharing sia molto bassa.

4.2.1. Studio dei veicoli immatricolati nell'area

Il passo successivo è quello di stimare il rapporto tra il numero di veicoli esistenti nell'area e quelli presenti nel dataset, così da poter fornire un valore sul numero reale di utenti che potrebbero effettivamente usufruire del servizio di sharing oggetto di questo studio.
Inizialmente è necessario studiare la densità abitativa presente a Brescia e nell'area metropolitana. Dopo aver analizzato la densità abitativa di tutti i paesi presenti all'interno dei 15km di raggio considerato, tenendo anche conto della loro distanza dal centro del servizio, si è provveduto facendo una stima della densità abitativa come riportato nella seguente tabella (Tab. 4.1).

Densità abitativa nell'area metropolitana di Brescia

Distanza dal centro del servizio	**Densità abitativa [abitanti/km^2]**
Da 0 a 6km	2170
Da 6 a 10km	950
Da 10 a 15km	600

Tabella 4.1: Densità abitativa area metropolitana di Brescia

Nella provincia di Brescia sono presenti 814516 autovettore, [15], e inoltre risiedono nella provincia circa 1250000 abitanti, questo equivale a un rapporto di 0.65 auto per abitante. In Fig. 4.6 si può vedere l'andamento del numero dei veicoli immatricolati, a partire dalle stime sulla densità abitativa.

A questo punto si può procedere trovando il rapporto tra i veicoli realmente esistenti nell'area di analisi e quelli presenti nel dataset (Fig. 4.7): tale rapporto verrà indicato nell'equazione seguente con ρ:

$$\rho(r) = \frac{I(r)}{U(r)} \tag{4.2}$$

dove I(r) rappresenta il numero di veicoli immatricolati, mentre U(r) il numero di utenti con la casa all'interno dell'area (Fig. 4.3 sulla destra).
Questo rapporto è intorno a 45-40, perciò si può affermare come i veicoli presenti nel dataset (cioè quelli dotati di box Telematica) rappresentino effettivamente 2.5%

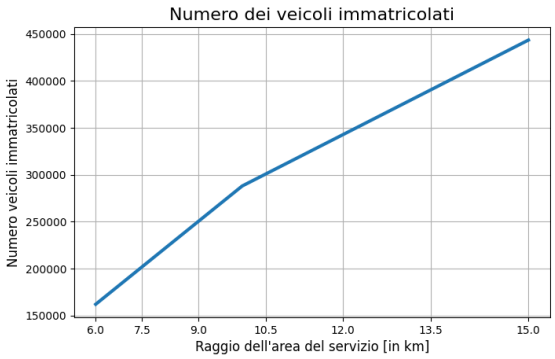


Figura 4.6: Numero dei veicoli immatricolati in funzione del raggio del servizio

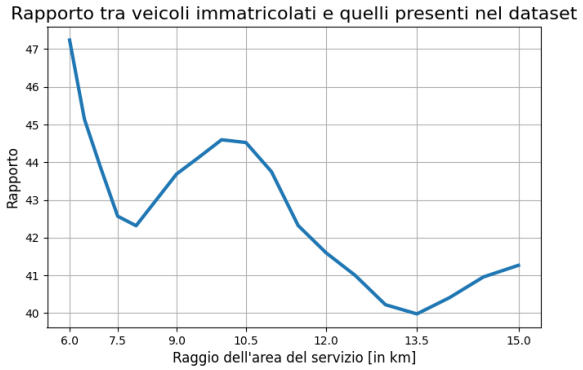


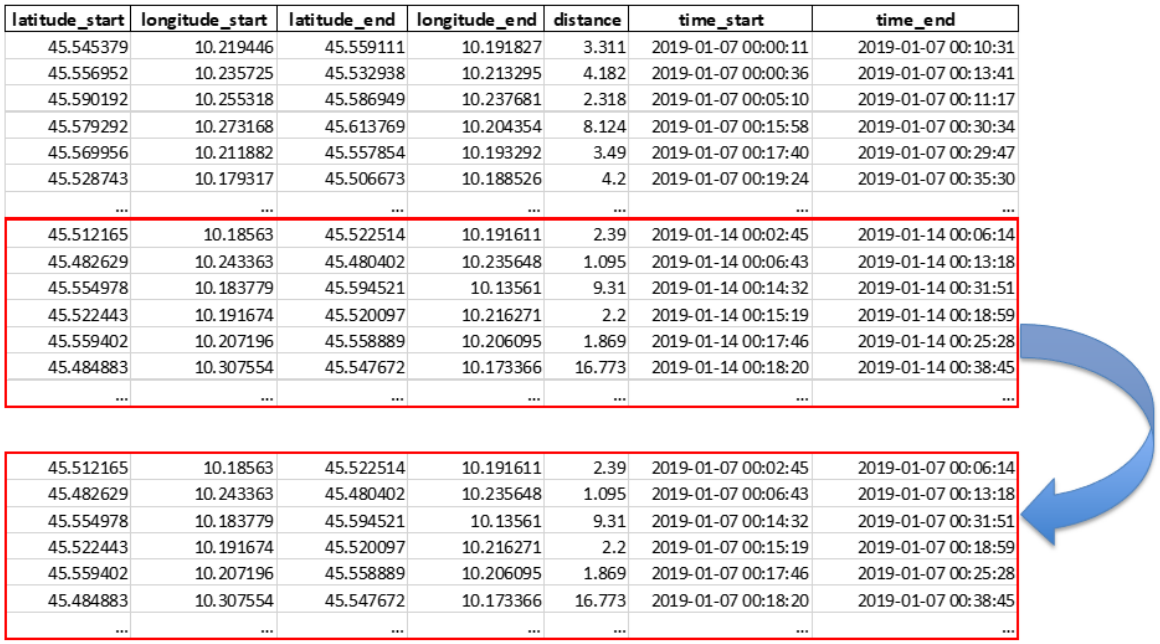
Figura 4.7: Rapporto tra veicoli immatricolati e quelli presenti nel dataset

dei veicoli realmente esistenti nell'area metropolitana di Brescia (2.5% viene trovato considerando il reciproco del rapporto ρ, cioè $1/\rho$).

4.2.2. Costruzione del dataset rappresentante il numero reale di veicoli immatricolati

In questa sezione viene descritto come il dataset di partenza venga proiettato per "costruire" la base di dati di tutti i veicoli realmente esistenti nell'area, la quale potrà poi essere utilizzata come input del simulatore del servizio di sharing.
Per prima cosa, per ogni raggio, si considerano tutti gli utenti sharing compatibili (Fig. 4.3 a sinistra) e si estraggono tutti i loro viaggi effettuati all'interno dell'area di copertura (ricordando come si intende che il punto di partenza e di arrivo debbano essere interni all'area). In seguito i viaggi vengono ordinati cronologicamente in funzione dell'orario di partenza (il parametro time_start).
Il dataset di partenza utilizzato in questo progetto comprende 2 anni di dati, a partire da Gennaio 2019 fino a Dicembre 2020, questo ci permette di avere a disposizione 104 settimane di dati.
Come visto precedentemente, i veicoli presenti nel dataset rappresentano solo il 2.5% dei veicoli realmente esistenti nella provincia di Brescia. L'idea è quella di accorpare più settimane di dati fra di loro (e unire quindi viaggi di settimane differenti), in modo da poter successivamente simulare il servizio di sharing considerando tutti i possibili utenti nella provincia, e non solamente quelli nel dataset.
Si procede quindi con il modificare i viaggi presenti nel dataset nella seguente maniera:

la data di inizio e quella di fine viaggio vengono cambiate in maniera tale da essere sovrapponibili con la prima settimana dell'anno del 2019. Viene quindi modificato solamente il giorno in cui è effettuato il viaggio, mantenendo comunque inalterato il giorno della settimana (Fig. 4.8).
Infine i viaggi vengono riordinati cronologicamente rispetto all'orario di partenza, come visibile in (Fig.4.9).



latitude_start	longitude_start	latitude_end	longitude_end	distance	time_start	time_end
45.545379	10.219446	45.559111	10.191827	3.311	2019-01-07 00:00:11	2019-01-07 00:10:31
45.556952	10.235725	45.532938	10.213295	4.182	2019-01-07 00:00:36	2019-01-07 00:13:41
45.590192	10.255318	45.586949	10.237681	2.318	2019-01-07 00:05:10	2019-01-07 00:11:17
45.579292	10.273168	45.613769	10.204354	8.124	2019-01-07 00:15:58	2019-01-07 00:30:34
45.569956	10.211882	45.557854	10.193292	3.49	2019-01-07 00:17:40	2019-01-07 00:29:47
45.528743	10.179317	45.506673	10.188526	4.2	2019-01-07 00:19:24	2019-01-07 00:35:30
...	...	...	...	...	...	...
45.512165	10.18563	45.522514	10.191611	2.39	2019-01-14 00:02:45	2019-01-14 00:06:14
45.482629	10.243363	45.480402	10.235648	1.095	2019-01-14 00:06:43	2019-01-14 00:13:18
45.554978	10.183779	45.594521	10.13561	9.31	2019-01-14 00:14:32	2019-01-14 00:31:51
45.522443	10.191674	45.520097	10.216271	2.2	2019-01-14 00:15:19	2019-01-14 00:18:59
45.559402	10.207196	45.558889	10.206095	1.869	2019-01-14 00:17:46	2019-01-14 00:25:28
45.484883	10.307554	45.547672	10.173366	16.773	2019-01-14 00:18:20	2019-01-14 00:38:45
...	...	...	...	...	...	...
45.512165	10.18563	45.522514	10.191611	2.39	2019-01-07 00:02:45	2019-01-07 00:06:14
45.482629	10.243363	45.480402	10.235648	1.095	2019-01-07 00:06:43	2019-01-07 00:13:18
45.554978	10.183779	45.594521	10.13561	9.31	2019-01-07 00:14:32	2019-01-07 00:31:51
45.522443	10.191674	45.520097	10.216271	2.2	2019-01-07 00:15:19	2019-01-07 00:18:59
45.559402	10.207196	45.558889	10.206095	1.869	2019-01-07 00:17:46	2019-01-07 00:25:28
45.484883	10.307554	45.547672	10.173366	16.773	2019-01-07 00:18:20	2019-01-07 00:38:45
...	...	...	...	...	...	...

Figura 4.8: Modifica dell'orario dei viaggi

latitude_start	longitude_start	latitude_end	longitude_end	distance	time_start	time_end
45.545379	10.219446	45.559111	10.191827	3.311	2019-01-07 00:00:11	2019-01-07 00:10:31
45.556952	10.235725	45.532938	10.213295	4.182	2019-01-07 00:00:36	2019-01-07 00:13:41
45.512165	10.18563	45.522514	10.191611	2.39	2019-01-07 00:02:45	2019-01-07 00:06:14
45.590192	10.255318	45.586949	10.237681	2.318	2019-01-07 00:05:10	2019-01-07 00:11:17
45.482629	10.243363	45.480402	10.235648	1.095	2019-01-07 00:06:43	2019-01-07 00:13:18
45.554978	10.183779	45.594521	10.13561	9.31	2019-01-07 00:14:32	2019-01-07 00:31:51
45.522443	10.191674	45.520097	10.216271	2.2	2019-01-07 00:15:19	2019-01-07 00:18:59
45.579292	10.273168	45.613769	10.204354	8.124	2019-01-07 00:15:58	2019-01-07 00:30:34
45.569956	10.211882	45.557854	10.193292	3.49	2019-01-07 00:17:40	2019-01-07 00:29:47
45.559402	10.207196	45.558889	10.206095	1.869	2019-01-07 00:17:46	2019-01-07 00:25:28
45.484883	10.307554	45.547672	10.173366	16.773	2019-01-07 00:18:20	2019-01-07 00:38:45
45.528743	10.179317	45.506673	10.188526	4.2	2019-01-07 00:19:24	2019-01-07 00:35:30
...	...	...	...	...	...	...

Figura 4.9: Dataset riproiettato e riordinato

Il numero di settimane che vengono aggregate fra di loro è funzione del rapporto ρ trovato nella sezione precedente(Fig. 4.7), ed è quindi dipendente dal raggio del servizio.

Attraverso questa tecnica, si è quindi fatta una riproiezione su un dataset che rappresenta fittiziamente, ma in maniera statisticamente molto significativa, i viaggi effettuati in una settimana anche da tutti gli utenti della provincia, e non solamente quelli dotati di box telematica.
A questo punto sorge la prima semplificazione fatta in questo progetto di tesi. Chiaramente mettere insieme più settimane di viaggi fatte dagli stessi utenti non è equivalente a considerare i viaggi effettuati da tutti gli utenti target nella regione di analisi: infatti i viaggi effettuati nelle varie settimane dagli stessi utenti iniziano e finiscono spesso nella stessa località. Tuttavia, per mantenere lo studio completamente "data-driven" e non fare alcuna assunzione, si è deciso di non modificare in alcun modo i punti di partenza e di arrivo dei viaggi, perchè saranno elementi chiave per impostare, nel simulatore, i luoghi di chiamata e rilascio dei veicoli del servizio di robotaxi.

Il dataset, così creato, rappresenta l'insieme di tutti viaggi che il servizio di car-sharing dovrebbe servire. Questo sarà l'input principale del simulatore.
I viaggi sono ordinati cronologicamente secondo il parametro time_start, in quanto questo verrà utilizzato per identificare l'orario di chiamata di un ipotetico utilizzatore del servizio.
L'orario di inizio e termine della corsa non saranno rappresentati effettivamente dai parametri time_start e time_end, in quanto dal momento della chiamata, al momento in cui il robotaxi arriva a prelevare il cliente, passa un certo lasso di tempo. Per esprimere la durata di una certa corsa nel servizio verrà quindi utilizzato il parametro trip_duration.
Il parametro distance rappresenta la distanza del viaggio nel servizio, mentre i parametri riferiti alla posizione iniziale e finale del viaggio rappresentano il luogo di chiamata e di rilascio del veicolo al termine della corsa.

5 | Il simulatore del servizio di car-sharing urbano autonomo

Una volta creata l'adeguata base di dati di percorrenza complessiva, è stato necessario costruire un tool in grado di simulare in maniera realistica il servizio di car-sharing urbano e autonomo. Questo è uno dei maggiori contributi di tale progetto di tesi.
La prima versione del simulatore, descritta in questo capitolo, non gestisce la parte di ricarica dei veicoli elettrici (ipotizzando per semplicità batterie di capacità infinita) che verrà introdotta nel capitolo successivo, con tutte le ottimizzazioni necessarie per garantire un servizio il più efficiente possibile.

5.1. Struttura del simulatore

I parametri che identificano in maniera univoca un servizio di car-sharing autonomo sono (attualmente) due:

1. **Il raggio dell'area di copertura del servizio (R)** Ricordando come il centro del servizio di car-sharing sia stato già fissato nel capitolo precedente, così come la forma dell'area (circolare), grazie a questo valore è possibile identificare unicamente la posizione e dimensione dell'area di copertura.
   Le corse servite devono quindi iniziare e terminare all'interno di questa regione ben delimitata.

2. **Numero dei veicoli in sharing (N)** Il parametro in questione fornisce indicazioni sulla dimensione della flotta dei veicoli in sharing. Tali veicoli sono i robotaxi che andranno a servire le corse all'interno del servizio.

Il simulatore deve essere strutturato in maniera tale da ricevere questi due parametri in input e simulare efficaciemente un servizio di car-sharing urbano e autonomo.
Nell'implementazione del processo di simulazione si è seguita una logica tale da rispec-

chiare un servizio di car-sharing nella maniera più realistica possibile.
La logica alla base è, quindi, la seguente:

1. Utilizzando il parametro R in input, viene definita l'area di copertura e quindi viene creato il dataset alla base del processo di simulazione del servizio, come descritto nella sezione 4.2.2

2. Il parametro N in input viene utilizzato per inizializzare N veicoli in sharing. In particolare gli N veicoli si trovano inizialmente nei punti di partenza dei primi N viaggi presenti nel dataset appena creato.

3. Ora si inizia a scorrere questo dataset a partire dalla prima corsa (si ricorda come le corse siano ordinate cronologicamente secondo time_start).
   Si considera time_start come se fosse il parametro indicante l'ora della chiamata effettuata dall'utente. Il luogo della chiamata è identificato da latitude_start e longitude_start.

4. Una volta effettuata la chiamata si seleziona il veicolo attualmente disponibile più vicino. Se il tempo di attesa affichè il veicolo arrivi è superiore ai 20 minuti la chiamata è considerata persa, altrimenti il veicolo si avvia autonomamente per andare a recuperare l'utente.
   Se non vi sono veicoli attualmente disponibli la chiamata è considerata persa.

5. Se la chiamata è valida l'utente può quindi effettuare il viaggio. Il veicolo rimarrà ovviamente occupato dal momento della chiamata fino al termine del viaggio, ossia dall'orario di time_start, fino all'orario identificato da:
   time_start + tempo necessario per il robotaxi per andare nel punto della chiamata + durata della corsa (trip_duration).

6. Terminata la corsa lo stato del veicolo diventa nuovamente disponibile. Tale robotaxi rimane fermo nelle coordinate identificate dai parametri latitude_end e longitude_end.

7. Si procede in questa maniera a simulare ricorsivamente tutti i viaggi presenti nel dataset creato.

A valle del processo di simulazione, è possibile estrapolare delle statistiche utili a descrivere in maniera sintetica l'efficienza del servizio di car-sharing appena simulato.

Tali statistiche, che vengono poi fornite dal simulatore come variabili output, sono le seguenti:

- **Il tempo di attesa medio di un utente** Per ogni corsa effettuata nel servizio si tiene traccia del tempo di attesa, ossia la durata temporale che decorre dal momento in cui è stata effettuata la chiamata, al momento in cui il robotaxi arriva a prelevare l'utente.
  Viene poi fatta una media artimetica di tutti questi valori.
- **La frazione di chiamate perse** La frazione di chiamate che il servizio non è riuscito a servire. Questo valore viene trovato come rapporto tra il numero di corse che il servizio appena simulato non è riuscito a servire e il numero totale di viaggi presenti nel dataset.
- **La distanza percorsa dai veicoli in sharing in media al giorno** Per ogni veicolo si tiene conto della distanza percorsa durante il processo di simulazione, questa distanza viene poi normalizzata rispetto a una giornata in servizio. Infine si fa una media aritmetica di tali valori.
- **La frazione di overhead** Questa statistica viene calcolata nel seguente modo: si sommano inizialmente tutte le distanze percorse senza persone a bordo dai robotaxi (per esempio per andare a recuperare un utente per la chiamate effettuata) e analogamente si sommano le distanze totali percorse dai vari robotaxi. Il rapporto tra queste due sommatorie descrive quindi la frazione di overhead generata dal servizio nel suo complesso.

5.2. Assunzioni per la simulazione

In questa sezione sono enunciate tutte le assunzioni fatte durante la creazione del simulatore, in maniera da chiarire anche la logica presentata nella sezione precedente.

- Quando i veicoli viaggiano senza utente a bordo si assume che si muovano con una velocità fissa di 15km/h, quando invece stanno effetuando un corsa con un utente a bordo, che guida, seguono la stessa velocità media descritta dal viaggio (distance/trip_duration).
- Per trovare la distanza del veicolo dal luogo in cui è effettuata la chiamata si

procede utilizzando la formula di Haversine:

$$d = 2r \cdot arcsin(\sqrt{sin^2(\frac{\phi_2 - \phi_1}{2}) + cos(\phi_1)cos(\phi_2)sin^2(\frac{\lambda_2 - \lambda_1}{2})}) \tag{5.1}$$

Dove r rappresenta il raggio della terra, ϕ_1, ϕ_2 rappresentano la latitudine dei due punti (espresse in radianti), e λ_1 e λ_2 rappresentano la longitudine dei due punti (espresse in radianti). La distanza d viene quindi moltiplicata per un fattore 1.5 (dovuto al fatto di avere strade irregolari che non collegano in linea retta i due punti) [16].

- Si assume che al termine della corsa il veicolo sia parcheggiato in prossimità della posizione in cui il viaggio è terminato.

5.3. Analisi sulla convergenza del simulatore

Una volta definita l'architettura del simulatore si può procedere con un'analisi sulla convergenza delle variabili output, ossia per quanto tempo è necessario simulare affinchè tali variabili convergano a un determinato valore.
Si decide per esempio di simulare un servizio con 2800 autovetture in sharing e un raggio dell'area di copertura di 9km. Questi valori sono stati presi arbitrariamente, ma in maniera tale da rispecchiare un servizio che sia abbastanza efficiente (numero di chiamate perse sufficientemente basso, tempo di attesa medio per gli utenti non eccessivo, ecc). Si è comunque verificato come simulando il servizio con parametri in input diversi da 2800 veicoli e 9km di raggio, si giunga alle medesime considerazioni.

I grafici in Fig. 5.1 mostrano come la variabili in output alla simulazione varino in funzione della durata della simulazione stessa.
L'inizio della simulazione è un lunedì mattina alle 00:00.

Il primo picco che si nota nei grafici del tempo di attesa medio e delle chiamate perse è dovuto alla grande concentrazione di viaggi che vi sono durante le prime ore di punta della giornata, ovvero verso le 8-10, mentre durante le ore serali e quelle notturne ci sono molte meno richieste da servire e quindi tendenzialmente tutte le chiamate riescono ad essere soddisfatte in maniera efficiente.
Dopo due giorni tutte le variabili tendono a stabilizzarsi verso un valore definito, tut-

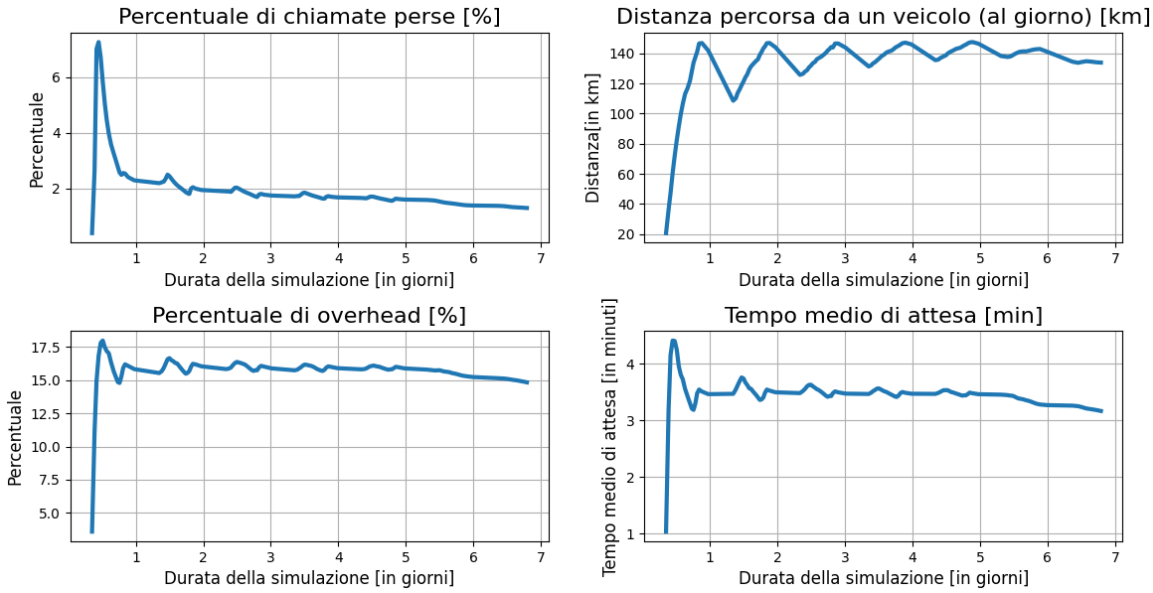


Figura 5.1: Analisi sulla convergenza delle variabili in uscita dal simulatore

tavia dopo il quinto giorno, ovvero dopo il venerdì, si nota come tutti i grafici abbiano una leggera flessione e diminuiscano lievemente. Questo è dovuto al fatto che nei weekend tutti i viaggi per recarsi o tornare dal posto di lavoro non ci sono e quindi il servizio di sharing ha molti meno viaggi da servire.

Per avere delle stime approssimative sull'efficienza del servizio basterebbe simulare un paio di giorni di viaggi del dataset riproiettato, però per avere risultati più accurati è necessario simulare una settimana intera, in maniera da considerare anche i weekend. Per i risultati finali ottenuti nel Capitolo 8 si è quindi optato per simulare sempre su una settimana intera.

6 | Integrazione della mobilità elettrica nel servizio di car-sharing urbano autonomo

Il passo successivo è quello di introdurre il tema della mobilità elettrica all'interno del servizio di robotaxi e quindi del processo di simulazione.

6.1. Definizione dei veicoli elettrici ed autonomi per il car-sharing e delle stazioni di ricarica

Il veicolo elettrico medio di riferimento in questo studio si assume essere una Volkswagen ID 3 con le seguenti caratteristiche:

- Capacità Batteria: 60 kWh
- Consumo medio in città (C_c): 0.2 kWh/km
- I costi di assicurazione, la tassa di circolazione e i costi di mantenimento si assumono essere 1000 €/anno
- Prezzo: 35000 €

Figura 6.1: Volkswagen ID 3

L'ipotesi aggiuntiva sarà, ovviamente, che questa vettura sia con livello L4 di autonomia, in maniera da essere utilizzabi-

le come robotaxi.
L'evoluzione del livello della batteria della macchina (State of charge - SOC) durante l'utilizzo, cioè il processo di scarica è descritto dalla seguente equazione:

$$SOC(t + \Delta t) = SOC(t) + C_c \cdot d(\Delta t) \tag{6.1}$$

SOC rappresenta lo stato di carica dalla batteria [in kWh], C_c il consumo medio, come indicato sopra, e $d(\Delta t)$ la distanza percorsa [in km] dal veicolo nell'intervallo di tempo Δt .

In ogni stazione di ricarica è presente un numero variabile di colonnine, con le seguenti proprietà:

- La potenza in uscita dalle colonnine (P_c) è di 13kW
- Si assume una efficienza (η_c) nel processo di ricarica del 97.5%

Figura 6.2: Colonnine di ricarica in una stazione

L'evoluzione del SOC durante il processo di ricarica è descritto della seguente equazione:

$$SOC(t + \Delta t) = SOC(t) + \eta_c P_c \Delta t \tag{6.2}$$

SOC rappresenta lo stato di carica dalla batteria [in kWh], η_c e P_c come definito sopra e Δt è l'intervallo di tempo espresso in ore in cui il veicolo è collegato a una colonnina e si sta quindi ricaricando.

Il numero di colonnine totali diventa quindi un parametro in input della simulazione (insieme al raggio e al numero di veicoli).

6.2. Meccanismo di ricarica all'interno del simulatore

Una volta definiti i veicoli e le stazioni di ricarica si può procedere con la presentazione della logica utilizzata per la gestione dei veicoli durante il processo di ricarica.
Si ipotizza di descrivere un veicolo in sharing:

1. La macchina è in servizio con un certo SOC.
2. Al termine di una corsa il SOC diventa minore di una certa soglia predefinita (che nel proseguo si indicherà con low_thres): $SOC(t) < low\_thres$. Lo stato della macchina diventa OCCUPATO, e il veicolo non può servire altre corse.
3. (a) Se non vi sono colonnine di ricarica disponibili, il veicolo rimane fermo nella sua posizione finchè non si libera una colonnina.
   (b) Se vi sono colonnine di ricarica disponibili si seleziona quella più vicina e si indirizza la macchina verso quella stazione.
4. La macchina rimane nello stato OCCUPATO mentre si trova in carica.
5. Quando il SOC diventa maggiore di un'altra soglia predefinita (che si indicherà con $high\_thres$) $SOC(t) > high\_thres$, lo stato del veicolo torna ad essere DISPONIBILE, ma il veicolo continua ad occupare la stazione e a ricaricarsi.
6. La colonnina di ricarica viene quindi liberata nelle due seguenti casisitiche:
   (a) Una chiamata viene effettuata a quel veicolo in carica, che ora è DISPONIBILE: il veicolo inizia quindi a muoversi verso il punto della chiamata.
   (b) Il veicolo raggiunge il 100% di SOC: si assume in questo caso che la colonnina venga liberata, e il veicolo rimanga fermo nella stessa posizione dove si stava precedentemente ricaricando.
7. La macchina è quindi tornata in servizio e si ricomincia dal punto 1.

Al fine di decidere quale valore assegnare a low_thres e $high\_thres$ si procede nella seguente maniera.
La soglia low_thres deve essere scelta in maniera tale da impedire che un veicolo rimanga senza carica, in particolare il veicolo deve essere in grado di fare una corsa

completa e poi di andare a ricaricarsi autonomamente.
Simulando, per esempio, un servizio con raggio 12km, uno dei casi più estremi che possono capitare durante il processo di simulazione si può notare in Fig. 6.3.

Sulla sinistra si può vedere la corsa più lunga effettuata da un veicolo in sharing (36km), mentre sulla destra il tragitto per andare in una delle stazione di ricarica più lontane (29km). Il totale dei due viaggi equivale a 65km che corrispondono a 13 kWh (utilizzando la formula 6.1), ossia il 21.7% della carica totale del robotaxi.

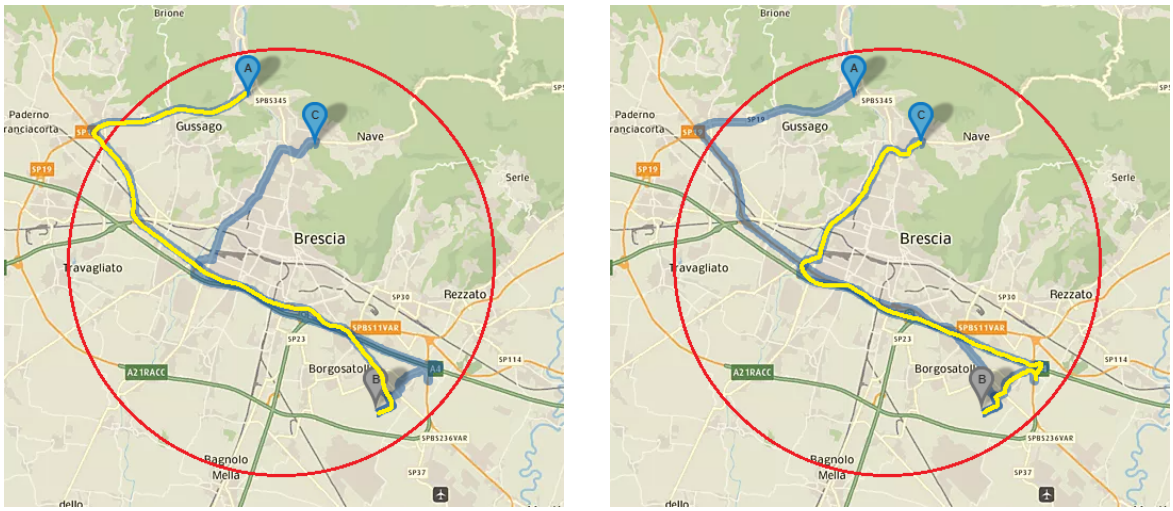


Figura 6.3: Esempio di percorso effettuato da un veicolo in sharing: sulla sinistra l'ultima corsa effettuata, sulla destra il tragitto per andare a ricaricarsi

Considerando un minimo di margine di robustezza, dovuto al fatto che nelle successive analisi di sensitività sul raggio si utilizzerà anche un raggio del servizio fino ai 15km, si è deciso di porre questa soglia al 25% della carica totale.

Per quanto riguarda $high\_thres$ la questione è meno complessa e vi è più libertà di scelta, la scelta di questa soglia verrà presa nella sezione 6.4.

6.3. Metodo per il posizionamento ottimale delle stazioni di ricarica

Sorge, quindi, il problema di trovare una collocazione ottimale delle stazioni di ricarica all'interno del servizio di car-sharing. Più precisamente, dato un numero di colonnine

in input, si vuole definire la maniera ottimale di posizionare tali colonnine all'interno dell'area di analisi.

Per trovare una soluzione a questo problema si è deciso di prendere spunto dalla ricerca di Cocca et al., [8], [7], che evidenzia come uno dei metodi migliori per avere un posizionamento efficiente si ottenga a partire dalla distribuzione spaziale delle chiamate effettuate nel servizio. La procedura utilizzata in questo studio è la seguente:

- Si considera l'area di copertura del servizio.
- L'area in analisi viene divisa in sottoregioni quadrate di lato 500m, come visibile in Fig. 6.4.
- Per ogni sottoregione quadrata si conta il numero di viaggi che partono in quell'area, in maniera da creare una distribuzione (visibile graficamente in Fig. 6.5) (dove il parallelepipedo è più alto vi è un numero maggiore di partenze).

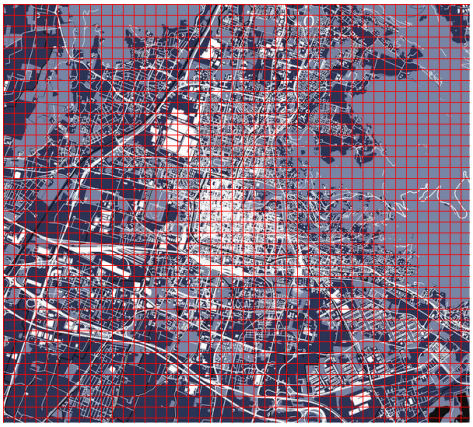


Figura 6.4: Esempio di suddivisione della città di Brescia in sottoregioni quadrate con lato 500m

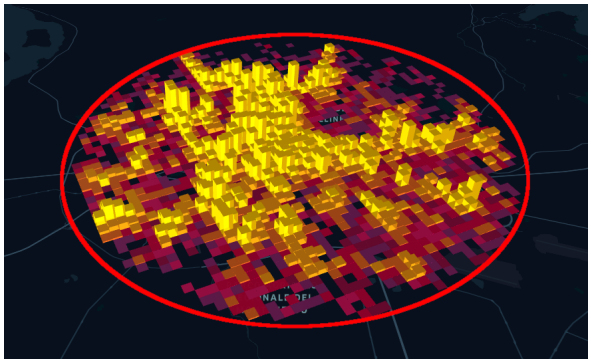


Figura 6.5: Distribuzione delle partenze dei viaggi nelle sottoregioni di lato 500m, all'interno dell'area con raggio 12km

- Tutte le sottoregioni vengono ordinate in base al numero di partenze (da quella con più partenze a quella con meno).
- A questo punto si procede ricorsivamente a partire dalla prima sottoregione. La stazione viene posizionata esattamente nel centro della sottoregione e il numero di colonnine presenti in quella stazione è fissato in maniera tale da rispec-

chiare la distribuzione delle partenze (secondo la formula che verrà descritta successivamente 6.3).

- Si procede così a considerare le successive sottoregioni con più partenze e a disporre le colonnine.
- La procedura si interrompe quando il numero totale di colonnine collocate nell'area di copertura raggiunge il numero di colonnine dato in input al processo di simulazione.

Il problema successivo è stabilire quante colonnine possano essere poste al massimo in una stazione di ricarica. Si vuole fare in maniera di non occupare con i veicoli in carica più dello 0.1% di suolo pubblico. Considerando che una sottoregione ha lato 500m, la sua area totale è di $250000m^2$. Lo 0.1% di tale valore equivale a $250m^2$.

Figura 6.6: Dimensioni del veicolo in sharing

Uno dei veicoli in sharing (Volkswagen ID 3) occupa un'area di circa $7.70m^2$, considerando anche la colonnina di ricarica si può assumere che lo spazio totale occupato da un veicolo in carica sia di circa $10m^2$. Il limite massimo di colonnine presenti in una stazione di ricarica sarebbe quindi di 25, però per rimanere un minimo più conservativi si è optato per mettere un limite massimo di 20.

Si può dunque definire le funzione utilizzata per trovare il numero di colonnine in una stazione di ricarica:

$$N\_colonnine = \left\lceil \frac{N\_partenze}{Max\_N\_partenze} \cdot 4 \right\rceil \cdot 5 \tag{6.3}$$

In particolare tale numero è il rapporto tra il numero di partenze in quella sottoregione, e il massimo numero di partenze fra le varie sottoregioni, moltiplicato per 4 e arrotondato per eccesso. Infine tale numero è moltiplicato per 5, in modo che il numero massimo di colonnine sia di 20, come definito poco sopra. Il numero di colonnine presenti in ogni stazione di ricarica può assumere quindi solo 4 differenti valori:

5,10,15,20.
Questo passaggio è stato svolto al fine di evitare che ci sia un numero eccessivo di stazioni di ricarica con troppe poche colonnine. Situazione che si presenta in caso di aree di copertura grandi($\geq$ 12km di raggio) con un numero elevato di colonnine da collocare.

Un esempio di distribuzione di 2000 colonnine in un'area di copertura di raggio 12km è visibile in Fig. 6.7.

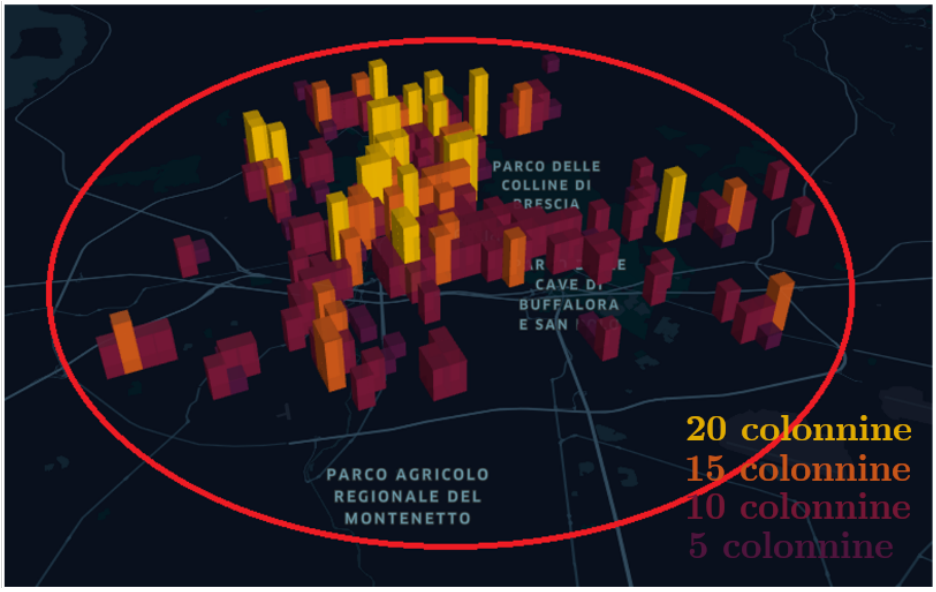


Figura 6.7: Esempio di collocamento delle stazioni di ricarica, 2000 colonnine totali e raggio di 12km

6.4. Vincoli di ricarica differenziati per il giorno e la notte e studio degli effetti

Per la seguente analisi si è deciso di prendere come esempio una simulazione del processo di sharing con i seguenti parametri in input: 3200 veicoli, 1500 colonnine di ricarica e 10km come raggio del servizio.
Tali valori sono stati presi arbitrariamente, ma in maniera tale da rispecchiare un servizio di car-sharing che non sia eccessivamente funzionale. Questo poichè le differenze di efficienza del servizio, utilizzando politiche di ricarica differenti, si notano in maniera più apprezzabile.

Se per esempio si fosse considerato un servizio di sharing con parametri in input molto grandi (>5000 veicoli, >2000 colonnine per 10km di raggio) i parametri di efficienza del servizio (quali la percentuale di chiamate perse , o il tempo di attesa medio, ecc) risulterebbero essere molto bassi, rimanendo quasi invarianti considerando politiche di ricarica differenti, rendendo quindi molto difficile capire quale sia il caso migliore.

Analizzando la distribuzione media nel tempo dei viaggi presenti nel dataset (visibile in Fig. 6.8) si è notato come la maggior parte dei viaggi siano concentrati durante il giorno, mentre come è lecito aspettarsi durante la notte non vi sono quasi chiamate.

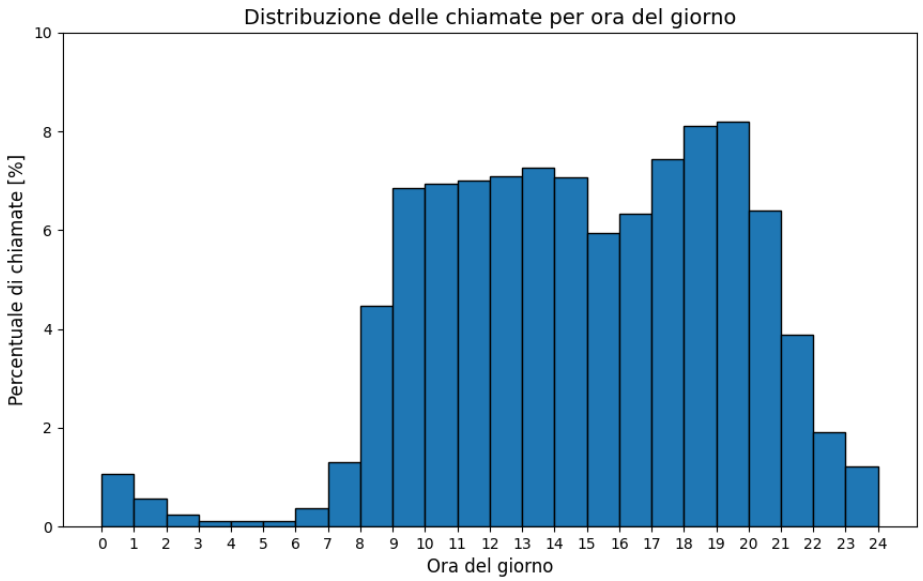


Figura 6.8: Distribuzione temporale, in media, delle chiamate nel servizio di sharing

Tale considerazione suggerisce come sia necessario avere politiche di ricarica differenziate per il giorno e la notte, in maniera tale da sfruttare le ore notturne, in cui la presenza di chiamate è estremamente ridotta, per ricaricare completamente il parco auto. In questa maniera i veicoli sarebbero completamente carichi al mattino e pronti a servire i clienti durante il giorno senza necessità di ricaricarsi.
Considerando la distribuzione di chiamate, Fig. 6.8, si sono decise le fasce orarie che indicano il giorno e la notte. Il giorno va dalle 7 alle 23, mentre la notte dalle 23 alle 7 (periodo in cui la percentuale di chiamate effettuate è più basso).
Per verificare tale intuizione, e per stabilire quale sia effettivamente la politica migliore di ricarica si sono individuati 4 casi base, come visibile in tabella 6.1.

Le 4 casistiche considerate

Casistiche	Low_thres day	High_thres day	Low_thres night	High_thres night
Caso 1	25%	30%	25%	30%
Caso 2	25%	60%	25%	60%
Caso 3	25%	60%	60%	90%
Caso 4	25%	30%	60%	90%

Tabella 6.1: Politiche di ricarica differenziate tra il giorno e notte, casistiche studiate con la definizione delle soglie

I primi 2 casi sono sono a soglia fissata sia durante il giorno che la notte, mentre i casi 3 e 4 hanno soglie differenziate per il giorno e la notte.
Il caso 1, avendo una soglia superiore ($high\_thres$) molto bassa, permette di effettuare ricariche anche di breve durata. Il caso 2 invece ha una soglia superiore parecchio più alta, che impedisce ai veicoli di effettuare ricariche troppo corte. Il caso 3 ha soglie uguali al caso 2 durante il giorno, mentre durante le notte le soglie sono fissate a livelli alti, così da "obbligare" i veicoli ad andare a ricaricarsi. Il caso 4 ha soglie durante il giorno uguali al caso 1 (quindi molto permissive), mentre di notte ha le soglie fissate agli stessi livelli del caso 3.
In primo luogo, andare a visualizzare graficamente come evolve lo stato della batteria dei veicoli in servizio, può aiutare a comprendere quale sia la politica migliore. In Fig. 6.9 si può notare l'evoluzione del SOC di un veicolo in sharing nelle 4 casistiche appena menzionate.

Nel caso 1 è chiaro che il veicolo effettui troppe ricariche e di breve durata, inoltre sono visibili dei tempi morti durante la notte (zone in cui il grafico è piatto) che sarebbero momenti ideali in cui mandare il veicolo a ricaricare.
Nel caso 2 vi è lo stesso problema del caso 1, in cui ci sono momenti in cui il grafico è piatto durante la notte. Inoltre data la soglia superiore al 60% i veicoli effettuano ricariche lunghe anche durante il giorno, spesso in orari di punta, che sono i momenti in cui c'è bisogno di avere più veicoli disponibili possibili, in maniera da soddisfare la grande richiesta di chiamate.

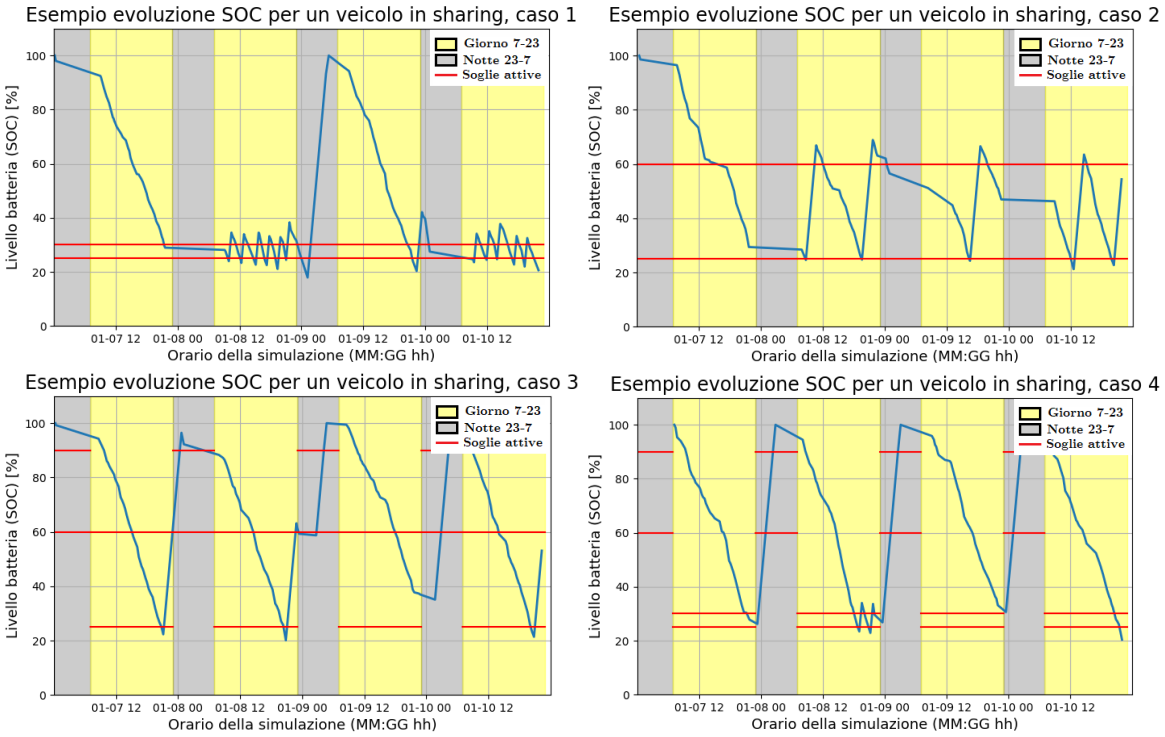


Figura 6.9: Esempi di evoluzione SOC nei 4 casi

Nei casi 3 e 4 si nota come avere politiche di ricarica differenziate per il giorno e la notte permetta ai veicoli di essere al massimo della carica (o poco di meno) all'inzio della giornata (ore 7:00). Questo permette ai veicoli di effettuare la loro "giornata lavorativa" spesso senza necessità di ricaricarsi, così da essere in grado di servire tutte le corse della giornata. Il caso 4 in particolare, avendo soglie molto permissive durante il giorno, permette di servire qualche viaggio in più nelle ultime ore dalla sera, rispetto al caso 3.

A questo punto è bene riepilogare i risultati delle simulazioni, sempre nelle 4 casistiche esempio, in maniera da confermare le considerazione fatte precedentemente.

Le 4 casistiche considerate: Risultati simulazione

Casi	Chiamate perse [%]	Distanza media, al giorno per veicolo [km]	Overhead [%]
Caso 1	11.50%	216.28	19.76%
Caso 2	12.84%	213.12	19.71%
Caso 3	7.44%	218.08	18.24%
Caso 4	7.69%	215.18	17.82%
Casi analizzati	**Tempo di attesa medio [min]**	**Numero di ricariche medio al giorno, per veicolo**	**Batteria mediamente ricaricata per ogni ricarica effettuata [%]**
Caso 1	5.71	3.44	16.25%
Caso 2	6.01	1.23	47.21%
Caso 3	4.86	0.97	62.91%
Caso 4	4.65	1.08	54.00%

Tabella 6.2: Le 4 casistiche considerate, risultati della simulazione

La Tab. 6.2 riassume gli indici più importanti per valutare la performance del servizio di sharing. É ora chiaro come l'introduzione di una politica differenziata durante la notte porti evidenti vantaggi, nei casi 3 e 4 si riesce ad avere un servizio più efficiente, con una percentuale minore di chiamate perse e un tempo di attesa medio per i clienti più basso.
Si è quindi deciso di utilizzare per tutti i risultati finali, presenti nel Cap. 8, la politica di ricarica utilizzata nel caso 4. Sebbene la percentuale di chiamate perse sia lievemente più alta del caso 3, il fatto di avere una politica meno stringente durante il giorno permette di avere un tempo di attesa medio inferiore e una percentuale di overhead minima.

6.5. Modifica nella struttura del simulatore

La struttura base del simulatore viene perciò modificata, aggiungendo tutte le considerazioni e assunzioni trattate in questo capitolo.
I parametri che identificano in maniera univoca un servizio di car-sharing autonomo ed elettrico diventano quindi tre: oltre al raggio dell'area di copertura del servizio (R) e il numero di veicoli in sharing (N) (come descritto nella sezione 5.1) si aggiunge anche il numero di colonnine di ricarica (S). Quest'ultimo parametro indica il numero di colonnine di ricarica che sono presenti all'interno dell'area di copertura, cioè quelle colonnine dove i robotaxi possono andare a ricaricarsi mentre sono in servizio. Il collocamento delle S colonnine all'interno dell'area segue il procedimento descritto nella sezione 6.3.
Di conseguenza il numero di variabili input del simulatore diventano tre.

7 | Ottimizzazione "simulator in the loop" del servizio di car-sharing urbano, autonomo ed elettrico

Nei capitoli precedenti è stato illustrato come si è arrivati a sviluppare un simulatore che rispecchi in maniera realistica un servizio di car-sharing urbano, autonomo ed elettrico.
A questo punto è necessario affrontare la seconda sfida principale di questo progetto di tesi, ossia l'ottimizzazione del servizio di car-sharing. Come precedentemente enunciato, la variabili che descrivono in maniera univoca il servizio sono tre: il numero di veicolo in sharing (N), il numero di colonnine di ricarica (S) e il raggio dell'area di copertura (R). È perciò necessario trovare quali siano i valori ottimali di queste tre variabili.

7.1. Il problema di ottimizzazione

L'efficientamento del servizio in sè, dal punto di vista operativo, è stato illustrato nel capitolo 6. Il problema di ottimizzazione trattato successivamente segue, perciò, una logica economica. In particolare si tratta di un problema di minimizzazione dei costi, con vincoli che garantiscano una minima efficienza e operatività del servizio.

7.1.1. Definizione delle variabili indipendenti del problema

Le variabili indipendenti sono quelle rispetto a cui bisogna ottimizzare e sono le stesse che vengono utilizzate come input dal simulatore di car-sharing, ossia N, S e R. Esse indicano, rispettivamente, il numero di veicoli nel servizio, il numero di colonnine di ricarica e il raggio dell'area di copertura.

7.1.2. Definizione delle variabili dipendenti del problema

Per definire il problema di ottimizzazione nella sua interezza è necessario specificare delle altre variabili, che sono output del simulatore e servono per valutare l'efficienza del servizio. Poichè queste variabili sono output del simulatore, dipendono dalle variabili input N,S ed R. In particolare:

- $x_1(N, S, R)$ indica la percentuale di chiamate perse
- $x_2(N, S, R)$ indica la distanza percorsa mediamente [in km] al giorno da un veicolo in sharing
- $x_3(N, S, R)$ indica la percentuale totale di distanza percorsa in overhead (ovvero senza utente a bordo)
- $x_4(N, S, R)$ indica il tempo di attesa medio per ogni chiamata (in min)

Queste quattro variabili sono, perciò, le stesse descritte nella sezione 5.1, e vengono ricavate nella stessa maniera.

7.1.3. Definizione della funzione obiettivo

Poichè si è optato per un problema di minimizzazione dei costi, il primo passo per definire la funzione obiettivo è trovare i costi principali in un servizio di car-sharing. Prendendo spunto dalla ricerca di Richter et al., [14], e sfruttando parte delle loro assunzioni, si è visto come i costi principali in un servizio di sharing di questo tipo siano tre:

- **Costo di acquisto e mantenimento della flotta di veicoli:** $C_1(N)$
  - Il costo di acquisto di un veicolo in sharing, come definito nel Cap. 6 è assunto essere pari a 35000€(costo_veicolo), ma date le grandi dimensioni

della flotta si assume uno sconto all'acquisto del 30% (fattore_sconto).

- Si assume che i veicoli abbiano una periodo di attività di 5 anni. In questo arco di tempo, un veicolo in sharing percorre in media circa 300000km.
- Tutti i vari costi di manutenzione (pulizia del veicolo, riparazioni eventuali, costi di assicurazione, ecc) si assumono essere 1000€/anno (costo_mantenimento) per veicolo.

- **Costo totale dell'energia necessaria per ricaricare i veicoli:** $C_2(N, S, R)$.
  - Il costo al kWh si assume essere pari al prezzo dell'energia in Lombardia che è di circa 0.3€/kWh (costo_energia)
- **Costo per il collocamento e il mantenimento delle stazioni di ricarica:** $C_3(S)$
  - Il costo delle colonnine di ricarica è estremamente variabile, e varia principalmente in funzione della rapidità con cui i veicoli vengono ricaricati, e quindi in base alla potenza in uscita dalla stessa. 13 kW è una potenza medio-bassa per essere una colonnina pubblica e quindi anche il suo prezzo si mantiene basso.
    Si assume, quindi, un costo di 5000€(costo_colonnina) per l'acquisto di ogni colonnina di ricarica.
  - Il noleggio del suolo pubblico per creare un'area di parcheggio si assume essere 500€/anno (noleggio_suolo) per ogni colonnina.

Il costo totale del servizio di sharing è dato dalla somma di questi 3 termini. Inoltre, poichè la vita utile dei veicoli è assunta pari a 5 anni, si è deciso di definire i costi totali su questo orizzonte temporale.
La funzione obiettivo $F(N, S, R)$ è quindi uguale al costo totale C_{tot} del servizio nei 5 anni.

$$F(N, S, R) = C_{tot} = C_1(N) + C_2(N, S, R) + C_3(S) \tag{7.1}$$

dove i termini C_1, C_2, C_3 sono così descritti:

$$C_1(N) = N \cdot [costo\_veicolo \cdot (1 - fattore\_sconto) + costo\_mantenimento \cdot 5]$$
$$C_2(N,S,R) = energia\_consumata(N,S,R) \cdot costo\_energia$$
$$C_3(S) = S \cdot [costo\_colonnina + noleggio\_suolo \cdot 5]$$

energia_consumata(N,S,R) rappresenta la quantità totale di energia utilizzata per ricaricare la flotta nei 5 anni di servizio dei veicoli (espressa in kWh); questo valore è una statistica output del processo di simulazione e dipende perciò da tutte e 3 la variabili input del simulatore.

Il dataset utilizzato in input per simulare il servizio, come visto nel Cap. 3, è formato da circa 45 settimane di dati, condensate in una settimana. Poichè la funzione obiettivo è definita su 5 anni, è chiaro come sia necessario fare l'assunzione che la distribuzione dei viaggi effettuati dagli utenti rimanga costante su questo periodo.

7.1.4. Definizione del problema

Dopo aver introdotto tutti i termini necessari, si può definire il problema completo di ottimizzazione:

$$\begin{aligned} \min_{N,S} \quad & F(N,S,R) \\ \text{s.t.} \quad & x_1(N,S,R) < 5\% \\ & x_2(N,S,R) > 100 \\ & x_3(N,S,R) < 30\% \\ & x_4(N,S,R) < 5 \end{aligned} \tag{7.2}$$

I vincoli sulle variabili (x_i i=1,..,4) sono stati posti in maniera tale da assicurare una efficienza minima al servizio.

È importante notare come la funzione obiettivo (rappresentate i costi totali del servizio) venga minimizzata solamente rispetto alle variabili N (numero dei veicoli in sharing) ed S (numero colonnine di ricarica), nonostante essa dipenda anche da R(raggio dell'area di copertura). Questo poichè è chiaro come un'area di copertura ristretta implichi costi minori (principalmente dovuto ad un acquisto di una flotta di dimensione ridotta).

Tuttavia anche gli eventuali ricavi sarebbero minori, in quanto ci sarebbero meno clienti da servire, rispetto a un'area di copertura più estesa.
Per ovviare a questo problema si è deciso di svolgere diverse ottimizzazioni separate, ognuna effettuata mantenendo un raggio del servizio fissato. Infine nel capitolo 8 si sono valutate altre statistiche output del simulatore (in particolare il prezzo delle tariffe minime che garantirebbero il break-even economico) per stabilire quale sia il raggio ottimale del servizio.

7.1.5. Schema del processo di ottimizzazione

Dal momento che le variabili utilizzate nei vincoli e la funzione obiettivo del problema di ottimizzazioni assumono valori che sono output del simulatore, è bene chiarire le schema di fondo utilizzato nel processo di ottimizzazione "simulator in the loop", in Fig. 7.1 è possibile visualizzare la logica seguita.

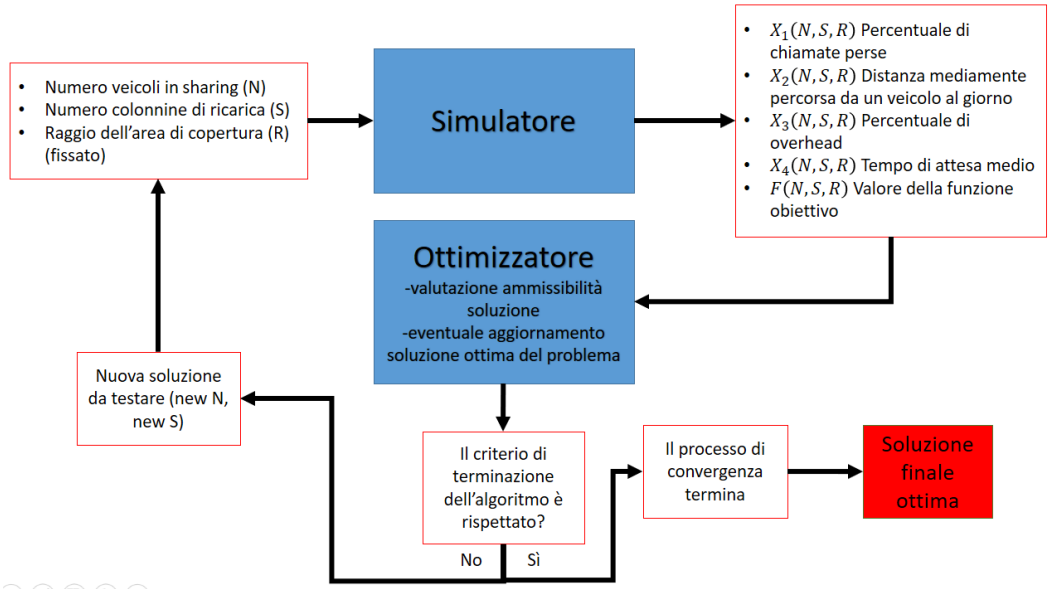


Figura 7.1: Schema del processo di ottimizzazione "simulator in the loop"

Le variabili indipendenti N,S ed R vengono fornite come input al simulatore, che restituisce i valori delle variabili dipendenti e della funzione obiettivo. In questa maniera è possibile valutare la bontà della soluzione attuale utilizzando la definizione del problema di ottimizzazione.
A questo punto, se il criterio di terminazione dell'algoritmo iterativo è rispettato, il processo di convergenza termina e l'ottimo attuale risulta essere anche l'ottimo globa-

le, altrimenti l'algoritmo di convergenza stabilisce un ulteriore soluzione da testare in maniera di muoversi verso il punto di ottimo globale.

È quindi necessario individuare un algoritmo iterativo efficiente per questo tipo di problema.

7.2. Analisi di convergenza del problema di ottimizzazione

Una volta che è stato definito il problema di ottimizzazione sorge un quesito molto importante, ovvero quale metodo iterativo sia da utilizzare per raggiungere la convergenza.
La prima difficoltà da affrontare è il fatto di avere variabili rispetto a cui ottimizzare di tipo discreto (numero di veicoli nel servizio, numero totale delle colonnine), questo impedisce di utilizzare i metodi iterativi più comuni (come il metodo del gradiente coniugato, o il metodo di Newton), poichè fanno uso delle derivate.
Un'altra difficoltà è dovuta al fatto che molte delle variabili utilizzate sono variabili output di un processo di simulazione basato su dati reali. Questo causa non linearità sia nella funzione obiettivo che nei vincoli. Tutto ciò comporta, inoltre, la presenza di numerosi minimi locali.
Un altro fattore importante da considerare è che il processo di simulazione è computazionalmente molto dispendioso (circa un'ora per simulare il servizio di sharing, nel caso di aree di copertura grandi). È quindi necessario trovare un metodo iterativo che converga nel numero minore possibile di passi.

7.2.1. Algoritmo "grid-search"

Il primo algoritmo analizzato è il metodo "grid-search". Questo processo iterativo utilizza il seguente schema per arrivare a convergenza:

1. Inizialmente vengono definiti i limiti inferiori e superiori delle varibili che si vogliono ottimizzare (il numero di veicoli e il numero di colonnine); si trova così il dominio in cui si vuole ottimizzare.

2. Il dominio viene diviso in una griglia, come visibile in Fig. 7.2.

3. La funzione obiettivo viene valutata in tutti i punti della griglia. Si trova così

l'ottimo attuale.

4. A questo punto si procede allo step successivo con un raffinamento della griglia.
5. Viene determinato un nuovo dominio attorno al punto di ottimo (i nuovi limiti inferiori e superiori sono definiti dai punti più vicini all'ottimo analizzati nello step precedente, come evidente in Fig. 7.2).
6. Si crea una nuova griglia nel nuovo dominio e si ritorna al punto 3.
7. L'algoritmo termina quando entrambe le differenze tra i limiti superiori e inferiori del dominio diventano minori dei parametri utilizzati per dividere il dominio in una griglia.
   Per esempio in Fig. 7.2 il parametro utilizzato per dividere il dominio del numero di colonnine ad ogni step è 5. Quindi la condizione per terminare l'algoritmo riferito a questa variabile è che:

$$lim\_sup(numero\_colonnine) - lim\_inf(numero\_colonnine) < 5 \qquad (7.3)$$

Questo algoritmo si adatta molto bene alla tipologia di problema di ottimizzazione studiata, tuttavia richiede di valutare la funzione obiettivo in un numero veramente alto di punti (circa 300 prima di raggiungere convergenza nel caso in Fig. 7.2). Poichè ogni valutazione della funzione obiettivo richiede una simulazione del servizio è chiaro come questo metodo sia difficilmente attuabile all'atto pratico, nel caso di aree di copertura con grandi raggi richiederebbe circa 2 settimane di tempo computazionale per raggiungere la convergenza.
Si procede quindi a valutare un algoritmo alternativo e più efficiente in grado di trovare il valore ottimo nel problema studiato.

7.2.2. Algoritmo di Nelder Mead

L'algoritmo di Nelder Mead è un metodo di ottimizzazione non lineare di funzioni definite su un dominio a n dimensioni (n=2 nel caso specifico). È un metodo di ricerca euristica, che non fa uso delle derivate, ma si basa sul concetto di simplesso, un particolare tipo di politopo a n+1 vertici in un dominio a n dimensioni (un triangolo nel piano nel caso specifico). Il procedimento che utilizza questo algoritmo è il seguente:

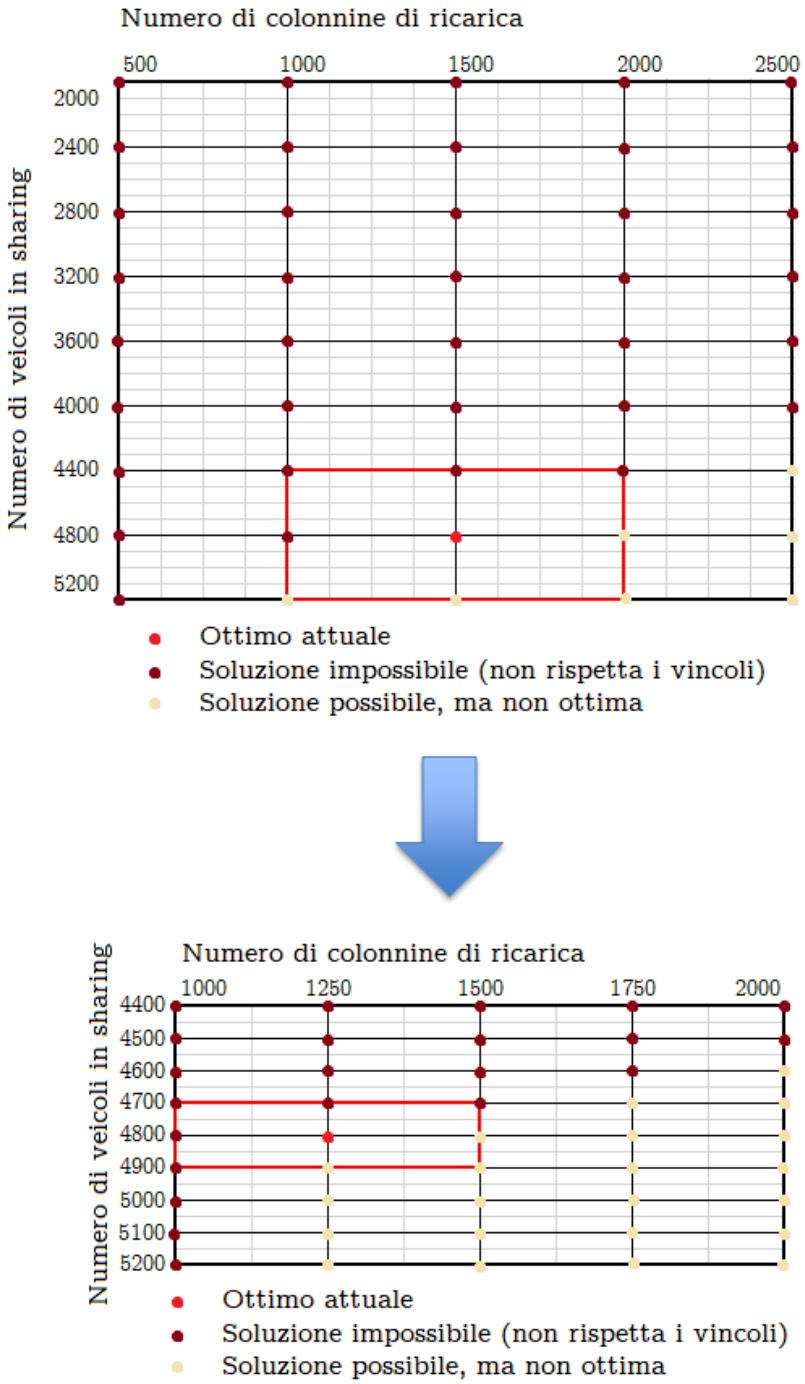


Figura 7.2: Metodo di risoluzione "grid-search", esempio con un raggio di 11km

1. Si comincia da un politopo arbitrariamente fissato all'interno del dominio di ricerca (condizione iniziale).

2. La funzione obiettivo viene valutata nei vertici del politopo.

3. I vertici vengono ordinati, secondo il valore della funzione obiettivo (da minimizzare in questo lavoro), ad esempio $F(x_3) \leq F(x_2) \leq F(x_1)$

4. Viene calcolato il baricentro (che si indicherà con x_0) riferito agli n-1 vertici migliori (quindi rispetto a x_3 e x_2).

5. A questo punto il politopo viene modificato attraverso uno dei seguenti metodi (visibili graficamente in Fig. 7.3):

   (a) **Riflessione** Viene calcolato il punto riflesso (Il punto peggiore viene riflesso rispetto al baricentro) $x_r = x_0 + \alpha(x_0 - x_1)$
   Se il punto riflesso è migliore del penultimo punto peggiore, ma non più valido del punto migliore, per esempio se $F(x_3) \leq F(x_r) < F(x_2)$, allora si determina il nuovo simplesso sostituendo x_1 con x_r e si torna al punto 2.

   (b) **Espansione** Se il punto riflesso calcolato al punto 5a è il migliore (cioè se $F(x_r) < F(x_3)$), allora viene calcolato il punto espanso $x_e = x_0 + \gamma(x_r - x_0)$. Se il punto espanso è migliore del punto riflesso $F(x_e) < F(x_r)$, allora viene sostituito il punto espanso al vertice peggiore del politopo e si ritorna al punto 2, altrimenti se $F(x_r) < F(x_e)$ viene sostituito il punto riflesso al vertice peggiore del politopo e si torna egualmente al punto 2.

   (c) **Contrazione** In questo caso si ha certamente che $F(x_r) > F(x_2)$ (cioè il punto riflesso è peggiore del penultimo punto migliore).
   Viene quindi determinato il punto di contrazione $x_c = x_0 + \rho(x_0 - x_1)$. Se il punto di contrazione è migliore del punto peggiore ($F(x_c) < F(x_1)$), si determina il nuovo simplesso sostituendo il vertice peggiore (x_1) con il punto di contrazione x_c e si torna al punto 2. Altrimenti si procede con l'ultima tipologia di modifica del simplesso.

   (d) **Riduzione** Tutti i punti del simplesso, tranne il migliore, vengono modificati nella seguente maniera $x_i = x_3 + \sigma(x_i - x_3)\ \forall i \in \{1, 2\}$ e si torna nuovamente al punto 2.

6. Durante il processo per raggiungere convergenza il politopo tende inizialmente ad allargarsi ed espandersi, in maniera tale da "muoversi rapidamente" verso il punto di ottimo, e quando si avvicina ad esso inizia a contrarsi e ridursi di

dimensione. Idealmente l'algoritmo continuerebbe indefinitamente a ridurre la dimensione del politopo intorno al punto di ottimo, perciò è necessario trovare un criterio di terminazione. Il criterio utilizzato per bloccare il processo iterativo è basato sulla deviazione standard della funzione obiettivo valutata nei vertici del politopo. In particolare l'algoritmo termina quando std($F(x_1)$,$F(x_2)$,$F(x_3)$) $< \epsilon$.

Quando si presenta questa situazione la funzione obiettivo calcolata nei vari punti è praticamente uguale. Questo significa che i vertici del politopo sono estremamente vicini tra di loro, e quindi anche estremamente vicini al punto di ottimo (se uno di essi non lo è già), con un margine di tolleranza dato da ϵ.

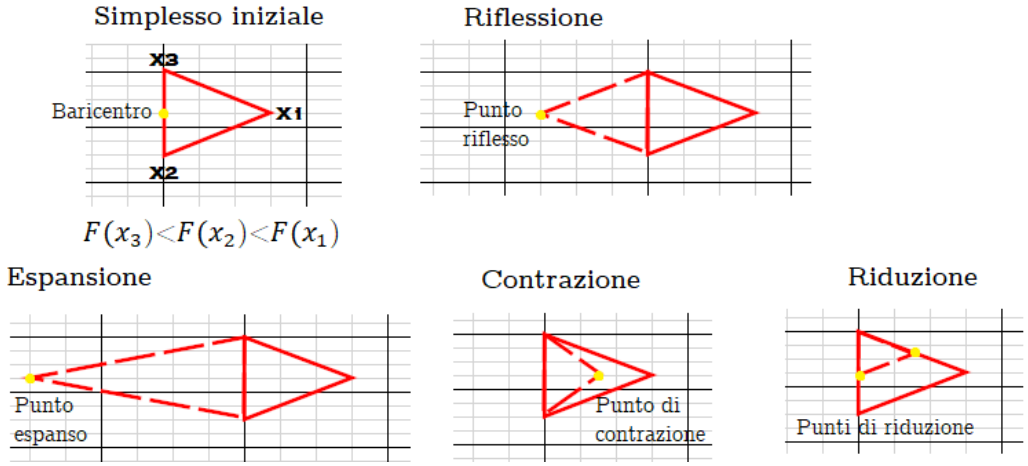


Figura 7.3: Esempio di modifica del simplesso nell'algoritmo di Nelder Mead

In questo progetto si sono utilizzati i seguenti valori dei coefficienti elencati precedentemente: $\alpha = 1$, $\gamma = 2$, $\rho = \frac{1}{2}$, $\sigma = \frac{1}{2}$, inoltre ϵ è stato posto a 0.1.

Durante il processo di modifica del simplesso succede spesso che i nuovi punti trovati non siano a coordinate intere, ma con numeri decimali. Poichè il simulatore riceve in input valori interi per il numero di colonnine e il numero di veicoli, in questi casi le coordinate vengono arrotondate all'intero più vicino.

Questo metodo si adatta anch'esso molto bene al problema in questione, in quanto il simplesso, modificandosi, riesce a non rimanere "intrappolato" nei piccoli, ma numerosi, minimi locali presenti. Inoltre, poichè non fa utilizzo delle derivate, si adatta facilmente a problemi in cui le variabili rispetto a cui ottimizzare sono di tipo discreto. Il numero di punti in cui l'algoritmo deve valutare la funzione obiettivo prima di raggiungere convergenza è circa 50 (valore dipendente dal simplesso iniziale), quindi

estremamente più veloce e competitivo dal punto di vista computazionale rispetto al metodo "grid-search".
L'unica criticità di questo metodo, poichè è un metodo euristico, è quella di essere dipendente dalla condizione iniziale. In caso di simplesso iniziale non appropriato l'algoritmo converge a ottimi locali, ma non globali. Per ovviare a questo problema, nei risultati finali illustrati nel Cap. 8, si sono svolte diverse ottimizzazioni sullo stesso problema, utilizzando simplessi iniziali differenti, in maniera da trovare il politopo iniziale che premettesse di raggiungere l'ottimo globale.
Si è notato, comunque, che considerando condizioni iniziali differenti l'algoritmo in tutti i casi converge a valori molto simili; la funzione obiettivo calcolata nei vari punti di ottimo trovati varia di un $\pm 1\%$ circa. Inoltre, poichè il lavoro di tesi si basa su dati reali, facendo anche assunzioni, una variazione sui risultati di 1% influisce in maniera irrilevante sulle considerazioni svolte nei successivi capitoli.

8 | Risultati dell'ottimizzazione "simulator in the loop"

Nel seguente capitolo si presentano tutti i risultati ottenuti come output del problema di ottimizzazione.
Fissato inizialmente il raggio, cioè la dimensione dell'area di copertura, vengono trovati i valori ottimali del numero di veicoli in sharing e del numero di colonnine di ricarica da posizionare.

8.1. Risultati con il raggio dell'area di copertura del servizio fissato

Come già enunciato precedentemente la funzione obiettivo è la somma di tutti i costi presenti nel servizio. Se l'ottimizzazione prendesse come input anche il valore del raggio tenderebbe sempre al raggio più piccolo possibile, per minimizzare al massimo il costo del servizio. Perciò si è optato per svolgere ottimizzazioni parallele a raggio fissato, e poi successivamente si sono confrontati i vari risultati, e fatte altre considerazioni al fine di stabilire quale sia il raggio ottimale.

8.1.1. Analisi con l'area di copertura del servizio ridotta

Si procede quindi ad analizzare i risultati considerando un'area del servizio ristretta, più in linea con la dimensione di altri servizi di sharing esistenti. In particolare si considera un raggio dell'area di copertura del servizio che varia tra gli 8km e i 12km, i risultati dell'ottimizzazione sono visibili in Tab. 8.1.

Risultati finali con raggio del servizio 8-12km

Raggio dell'area del servizio	**Numero di veicoli**	**Numero di colonnine di ricarica**	**Rapporto veicoli/colonnine**
8 km	2049	693	2.96
9 km	2804	986	2.84
10 km	3780	1043	3.62
11 km	4873	1217	4.00
12 km	5857	1792	3.27
Chiamate perse [%]	**Distanza mediamente percorsa al giorno per veicolo [in km]**	**Chilometraggio medio dei veicoli dopo 5 anni[in km]**	**Overhead totale [%]**
4.94%	128.82	295792	21.14%
4.98%	139.92	318419	19.80%
4.91%	140.34	317377	19.30%
4.98%	138.48	309307	18.29%
4.96%	143.99	318829	17.75%
Tempo di attesa medio [in min]	**Tempo medio in servizio al giorno per veicolo [HH:mm]**	**Velocità media dei viaggi effettuati [km/h]**	**Costo totale del servizio nei 5 anni[in milioni di €]**
3.94	5:59	21.33	102.48
3.73	6:21	22.03	144.17
3.61	6:11	22.68	190.66
3.38	5:57	23.25	241.46
3.35	5:59	24.06	298.36

Tabella 8.1: Risultati dell'ottimizzazione con area di copertura ridotta

Il numero di veicoli e il numero di colonnine, visibili in tabella, sono i risultati che si sono ottenuti ottimizzando la cifra di merito del problema di ottimizzazione definito

nel capitolo precedente. Si può notare, inoltre, come il vincolo più restrittivo sia quello riguardante la percentuale di chiamate perse: infatti gli altri vincoli del problema di ottimizzazione vengono soddisfatti più largamente, mentre i valori di percentuale di chiamate perse sono molto vicini al limite massimo del 5%.
Dalla Tab. 8.1 sembra che in termini di efficienza del servizio sia meglio avere il raggio il più grande possibile, in quanto la percentuale di overhead e il tempo di attesa medio diminuiscono all'aumentare della dimensione dell'area di copertura. Tuttavia questa non è una considerazione troppo indicativa, in quanto è evidente che una dimensione maggiore dell'area permetta di effettuare viaggi più lunghi e quindi l'overhead sia minore in termini percentuali. Inoltre dimensioni maggiori permettono di considerare molti più utenti target, quindi i viaggi sono meglio distribuiti all'interno dell'area e questo permette di avere tempi di attesa mediamente più bassi. Per valutare quale sia il miglior raggio è necessario valutare anche altre statistiche output della simulazione, secondo cui il raggio più grande non fornisce i risultati migliori: ad esempio il tempo medio in servizio dei veicoli e altre considerazioni dal punto di vista economico, come verrà spiegato successivamente. Si è deciso, comunque, nella prossima sezione, di verificare cosa succeda con raggi ancora più grandi, per valutare se effettivamente sia conveniente un servizio più esteso.
Poichè la maggior parte dei servizi di sharing ha una tariffa basata sulla durata temporale del noleggio, e non sul chilometraggio percorso, è chiaro come sia opportuno fare considerazione anche sul tempo in servizio dei veicoli. In questo caso il raggio migliore sembra essere 9km, in quanto il tempo medio in servizio dai veicoli è quello più alto.
A questo punto è necessario fare anche considerazioni dal punto di vista economico, in quanto, l'ottimizzazione è stata effettuata in questo senso. Nella Tab. 8.2, si possono vedere le tariffe minime che bisognerebbe impostare, per assicurarsi di arrivare a break-even dopo 5 anni di servizio.
In particolare le tariffe minime vengono calcolate come il costo totale del servizio nei 5 anni (valore visibile nella tabella precedente 8.1) diviso i chilometri totali percorsi in servizio dai veicoli in sharing, oppure diviso il tempo totale in servizio dei vari veicoli sempre nei 5 anni.
Le assunzione effettuate per il calcolo di questa tariffa sono, quindi, che la distribuzione nel tempo delle chiamate rimanga invariata nei 5 anni, come già esplicitato alla fine del Cap. 7, e che il valore totale dei veicoli e delle colonnine dopo 5 anni sia uguale a zero. Ossia si ipotizza che i veicoli, visto il chilometraggio estremamente elevato in 5

anni, si deprezzino totalmente e abbiano un valore nullo alla fine del periodo utile in cui sono in servizio; lo stesso ragionamento si applica per le colonnine di ricarica.

Tariffa minima per raggiungere il break-even in 5 anni

Raggio dell'area del servizio	Costo al chilometro [in €]	Costo al minuto [in €]
8 km	0.2144	0.0762
9 km	0.2013	0.0739
10 km	0.1969	0.0744
11 km	0.1958	0.0759
12 km	0.1939	0.0777

Tabella 8.2: Tariffe minime del servizio per andare a pareggio in 5 anni

Come si può notare, aumentando il raggio del servizio, il costo al chilometro è progressivamente più basso. Questo è lecito attenderselo in quanto in un'area più grande i viaggi sono mediamente più lunghi e veloci, quindi non è un parametro troppo indicativo per la scelta del raggio ottimale.
Il costo al minuto più basso si ottiene, invece, con un raggio di 9km.
Dal momento che l'area di copertura con raggio di 9km fornisce buoni parametri di performance del servizio (come il tempo di utilizzo più elevato dei veicoli) e inoltre è quello che permette una tariffazione al minuto più bassa, questo è da considerarsi come raggio ottimale dell'area di copertura del servizio di robotaxi proposto.

8.1.2. Analisi con l'area di copertura del servizio ampliata

Nella sezione precedente si è visto come il raggio ottimale sia di 9km, tuttavia anche il raggio di 12km fornisce risultati interessanti. Si è deciso, perciò, di svolgere l'ottimizzazione anche considerando un'area del servizio lievemente maggiore, fino ai 15km di raggio (limite superiore da considerare per il raggio, come stabilito nel Cap. 4). Nella seguente Tab. 8.3 vi sono i risultati ottenuti.

Risultati finali con raggio del servizio 13-15km

Raggio dell'area del servizio	Numero di veicoli	Numero di colonnine di ricarica	Rapporto veicoli/colonnine
13 km	8438	2007	4.20
14 km	9374	2141	4.38
15 km	10324	2417	4.08
Chiamate perse [%]	**Distanza mediamente percorsa al giorno per veicolo [in km]**	**Chilometraggio medio dei veicoli dopo 5 anni[in km]**	**Overhead totale [%]**
4.85%	133.03	293653	17.32%
4.75%	143.42	313416	16.48%
4.77%	147.22	322132	16.55%
Tempo di attesa medio [min]	**Tempo medio in servizio al giorno per veicolo [HH:mm]**	**Velocità media dei viaggi effettuati [km/h]**	**Costo totale del servizio nei 5 anni[in milioni di €]**
3.35	5:21	24.08	438.03
3.06	5:49	24.59	489.53
3.14	5:56	24.78	544.96

Tabella 8.3: Risultati dell'ottimizzazione con area di copertura estesa

Valutando un raggio più grande valgono le stesse considerazioni a cui si è giunti precedentemente, cioè si può notare come l'overhead generato e il tempo di attesa medio siano più bassi, rispetto a quelli ottenuti con un'area di copertura più piccola. Mentre invece i tempi medi di utilizzo dei veicoli sono sempre subottimali.
Per quanto riguarda la parte economica, nella seguente Tab. 8.4 si possono vedere le tariffe minime, calcolate nella stessa maniera della sezione precedente. Le tariffe al chilometro sono in linea con quelle ottenute per raggi inferiori, mentre le tariffe minime al minuto sono considerevolmente più elevate. Per questo motivo si è deciso

di mantenere 9km come raggio ottimale del servizio di car-sharing.

Tariffa minima per raggiungere il break-even in 5 anni

Raggio dell'area del servizio	Costo al chilometro [in €]	Costo al minuto [in €]
13 km	0.2138	0.0884
14 km	0.1995	0.0818
15 km	0.1964	0.0812

Tabella 8.4: Tariffa minima del servizio (area del servizio allargata)

8.2. Risultati finali con l'area di copertura del servizio ottimizzata

La regione di copertura ottimizzata è quindi di 9km di raggio, considerando quindi un servizio con 2804 veicoli e 986 colonnine di ricarica.
La prima considerazione che si può fare è trovare quante automobili potrebbero essere dismesse se effettivamente si passasse ad una forma di servizio di car-sharing di questo tipo. Il numero di veicoli in analisi, con un raggio di 9km, sono 575. Considerato che si sono accorpate 44 settimane di dati per la simulazione del servizio (secondo il grafico in Fig. 4.7 sulla destra), il numero reale di utenti con veicolo privato che potrebbero usufruire del servizio è di 25300.
Il rapporto tra il numero di auto private e il numero di veicoli in sharing è quindi di 9.02. Si potrebbe quindi dismettere 8 veicoli su 9.

8.2.1. Analisi degli spostamenti dei veicoli in sharing

Successivamente si sono analizzati in dettaglio gli spostamenti dei veicoli in sharing, e più in particolare la distribuzione della lunghezza dei viaggi effettuati dai veicoli (visibile in Fig. 8.1), e la distribuzione della durata dei viaggi stessi (Fig. 8.2).

Come è visibile in entrambi i grafici la maggior parte dei viaggi sono brevi, il 50% dei viaggi non supera i 3.15km (valore mediano). Questo è dovuto al criterio con cui sono stati selezionati gli utenti sharing compatibili nel Cap. 4: infatti si sono scelti

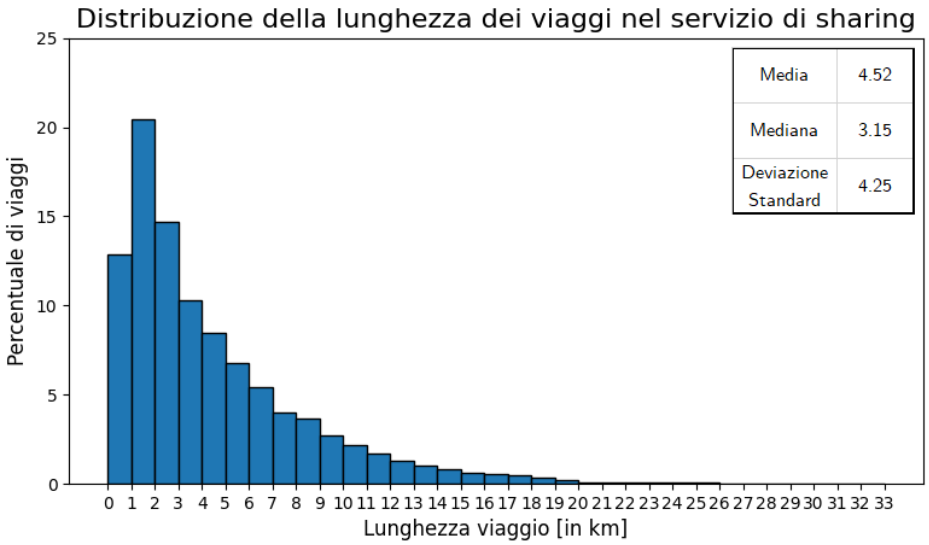


Figura 8.1: Distribuzione della lunghezza dei viaggi nel servizio ottimizzato

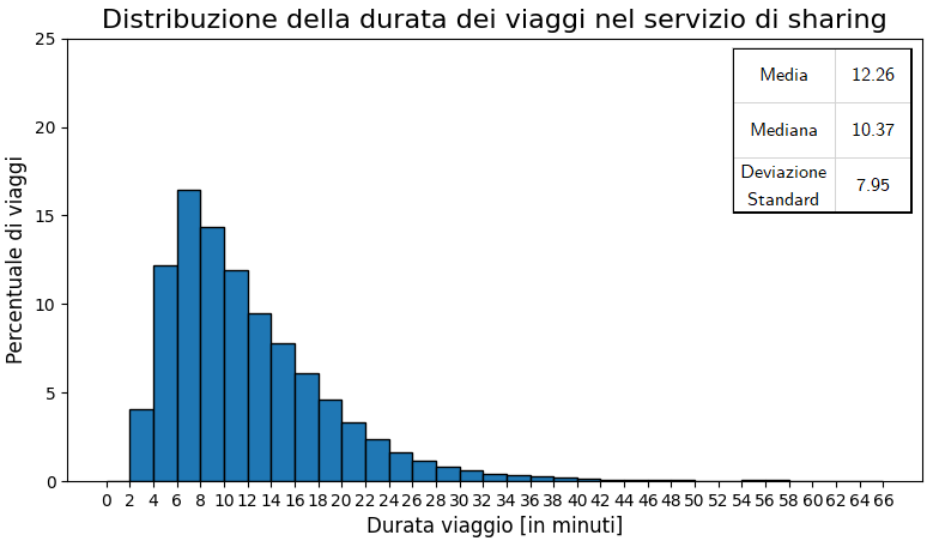


Figura 8.2: Distribuzione della durata dei viaggi nel servizio ottimizzato

solamente gli utenti che avessero almeno il 90% dei viaggi con inizio e fine all'interno dell'area di copertura. Questi utenti effettuano viaggi molto corti e di breve durata, e perciò anche la distribuzione dei viaggi nel servizio rispecchia questa caratteristica.
Una distribuzione di questo tipo (con forte asimmetria verso i valori più bassi) è comunque molto simile alle distribuzioni dei viaggi eseguiti dai veicoli in servizio in car-sharing esistenti, come visibile nel lavoro di Cocca et al., [8].
Si sarebbe potuti essere più permissivi nel processo di selezione degli utenti target e dei

viaggi (per avere una distribuzione con viaggi mediamente più lunghi), tuttavia uno dei principali obiettivi di questo progetto di tesi è proprio quello di studiare il passaggio da una mobilità urbana di tipo privato a una mobilità condivisa, intesa come servizio per i cittadini, e per valutare concretamente la fattibilità di una mobilità di questo tipo è stato necessario essere particolarmente stringenti con i parametri di selezione, affinchè i risultati fossero veramente significativi.

8.2.2. Analisi dinamica dell'overhead

Dopo aver controllato la distribuzione dei viaggi effettuati, è interessante valutare l'overhead generato dai viaggi, in funzione della distanza del viaggio effettuato. In maniera da fare considerazioni su un possibile piano tariffario per questo servizio di car-sharing autonomo ed elettrico. Si richiama come per overhead si intenda la distanza percorsa dai veicoli senza nessun utente a bordo. Al fine di trovare il grafico in Fig. 8.3 si è proceduto nel seguente modo:

- Inizialmente si sono suddivisi tutti i viaggi effettuati nel servizio in gruppi basati sulla distanza del viaggio stesso (da 0km a 1km, da 1km a 2km, ecc)
- Per ogni gruppo si è calcolata la distanza totale percorsa servendo un utente, e la distanza totale percorsa senza utente a bordo (ossia la somma delle distanze percorse dai veicoli per andare nel punto di chiamata dell'utente, prima di servire la corsa)
- Per ogni gruppo si è trovata la percentuale di overhead come rapporto tra la distanza percorsa senza utente a bordo e la distanza totale percorsa (con e senza utente a bordo)
- Il grafico viene poi costruito interpolando lineramente i vari punti

Come evidenziato in Fig. 8.3, i viaggi che hanno una distanza molto piccola (minore di 0.8km) generano un overhead maggiore del 50%, questo significa che mediamente i veicoli devono percorrere una distanza maggiore per andare nel luogo di chiamata, rispetto alla distanza percorsa poi nel viaggio stesso.
La linea rossa in figura rappresenta il limite superiore (30%) per l'overhead totale del servizio utilizzato come vincolo nel problema di ottimizzazione, questa linea viene attraversata dal grafico a 2.5km. Quindi se il servizio di car-sharing dovesse servire sempre corse più corte di 2.5km, l'overhead totale generato dal servizio stesso sarebbe

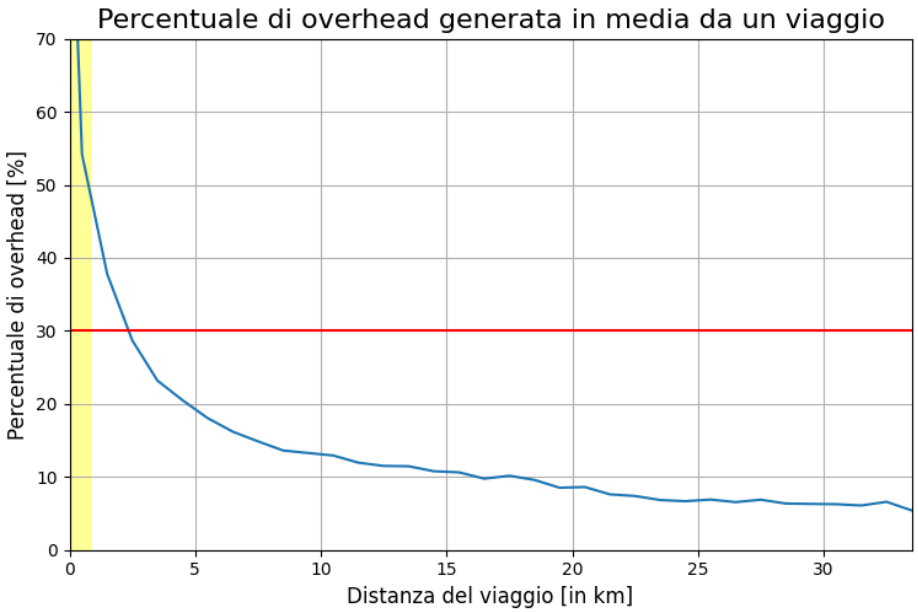


Figura 8.3: Overhead generato dai viaggi nel servizio di car-sharing

maggiore del 30% e perciò non rispetterebbe i vincoli imposti dal problema di ottimizzazione.
In termini economici i viaggi più corti sono quelli più svantaggiosi, poichè comportano una distanza maggiore percorsa senza utente a bordo. Di conseguenza nei sistemi in sharing è opportuno introdurre una tariffazione progressiva a fasce.

8.2.3. Analisi economica e proposta di tariffazione

A seguito delle considerazione appena fatte, si è deciso di proporre un sistema tariffario a 2 fasce (divise rispetto alla lunghezza del viaggio o in base alla durata dello stesso). Al fine di garantire che il servizio sia profittevole e con un margine di tolleranza rispetto al numero di utenti target effettivamente coinvolti nel servizio, le tariffe proposte sono state ottenute aumentando di circa il 50% le tariffe minime ottenute nella sezione precedente. In Tab. 8.5 è possibile vedere tali tariffe.
Questi risultati sono in linea con i valori ottenuti in altri studi su questo argomento. Dandl et al., [17], stimano un costo al chilometro per un sistema di sharing (equivalente a quello in questo progetto di tesi) di circa 0.25-0.27€/km. Bosch et al., [18], trovano un prezzo del servizio di circa 0.37€/km.

I prezzi proposti rimangono ancora notevolmente più bassi (almeno il 40% in meno) rispetti ai costi dei servizi di car-sharing esistenti e questa è una conclusione molto significativa. Si sono prese come riferimento le tariffe di Share-now a Milano, che partono da un minimo di 0.19€/min [19].

Proposta di tariffazione

Tariffa basata sulla distanza percorsa	**Tariffa basata sulla durata del viaggio**
Primi 3 km del viaggio 0.35€/km	Primi 10 minuti del viaggio 0.13€/minuto
Dopo i 3 km 0.30€/km	Dopo i 10 minuti 0.10€/minuto

Tabella 8.5: Proposta di tariffazione basata sugli output dell'ottimizzazione con il simulatore

Il prezzo medio di un viaggio è di circa 1.48€ per entrambe le tariffe proposte (ricordando che un viaggio effettuato nel servizio è mediamente lungo 4.5km). Questo valore è anche molto competitivo rispetto ai prezzi dei mezzi pubblici. Nell'area urbana in analisi, cioè Brescia e area metropolitana il costo di un biglietto ordinario è di 1.40€. I ricavi per settimana, a vantaggio del gestore del servizio di sharing, sarebbero di poco più di 900000€ in entrambe la casistiche.
A questo punto si può analizzare la distribuzione dei costi del servizio: in Fig. 8.4 si nota come il costo principale per creare un servizio di questo tipo sia dato dal prezzo di acquisto iniziale della flotta (in termini assoluti si tratta di 74.12 milioni di €) e si tratta di quasi il 50% del costo totale del servizio nei 5 anni.

I costi di mantenimento del servizio sono abbastanza limitati, mentre occupa una percentuale importante il costo dell'energia necessaria per ricaricare i veicoli nei 5 anni di servizio. In particolare ogni settimana si spenderebbero 207000€ per ricaricare i veicoli.

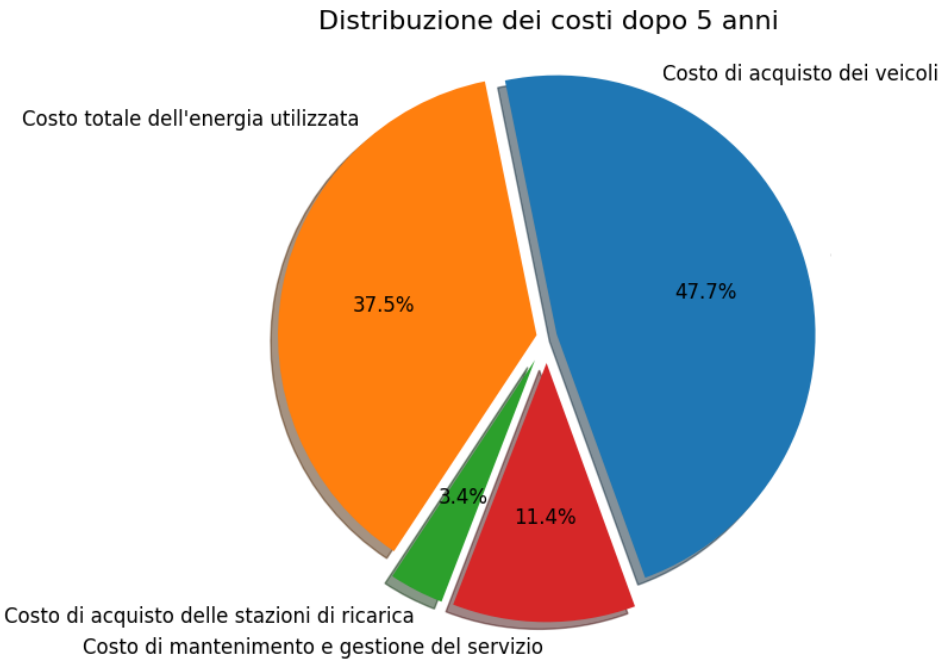


Figura 8.4: Distribuzione dei costi

9 | Mobility As A Service: dall'auto privata al car-sharing urbano, autonomo ed elettrico

Il progetto di tesi assume che tutti gli utenti target identificati nel capitolo 4 passino istantaneamente al servizio di car-sharing e si liberino immediatamente della propria autovettura privata. Nella realtà di tutti i giorni è lecito attendersi che il processo di transizione da un sistema di mobilità privato ad uno condiviso sia molto lento e graduale. Inoltre la ragione primaria che potrebbe spingere gli utenti ad abbandonare la propria autovettura in favore di un servizio condiviso è quella economica. È necessario quindi controllare se sia effettivamente vantaggioso per gli utenti target utilizzare un servizio di car-sharing elettrico ed autonomo come descritto in questo lavoro.

9.1. Studio sulla progressiva transizione degli utenti dall'auto privata al car-sharing urbano, autonomo ed elettrico

Si procede quindi ad analizzare uno scenario di passaggio graduale al servizio di MaaS, ipotizzando che in principio un numero molto inferiore di utenti (rispetto a quelli definiti compatibili con lo sharing) dismettano la propria vettura e per spostarsi in città usino robotaxi.
Si considera sempre un raggio del servizio di 9km.
Due scenari differenti vengono analizzati:

1. **I parametri del servizio vengono riottimizzati al variare del numero di utenti**
   In funzione del numero di utenti che usufruiscono del servizio di car-sharing si trovano il numero ottimale di veicoli e delle colonnine di ricarica.
   Questa situazione simula un servizio che viene costruito gradualmente, aggiungendo progressivamente veicoli e colonnine di ricarica, in base al numero effettivo di utenti che decidono di passare a questa forma di trasporto condiviso.

2. **I parametri del servizio non vengono riottimizzati**
   Le variabili del problema di ottimizzazione rimangono dunque fissate.
   Questa situazione simula un servizio costruito subito per tutti gli utenti target, e analizza cosa succeda qualora poi effettivamente meno persone del previsto ne usufruiscano.

Viene quindi studiato il primo caso; i risulati dell'ottimizzazione sono visibili in Tab. 9.1.

Come si nota dalla tabella il servizio perde efficienza qualora vi siano meno utenti coinvolti: infatti minore è il numero di utenti che usufruiscono del servizio maggiore è il tempo di attesa medio e la percentuale di overhead. La stessa considerazione si può fare studiando i grafici in Fig. 9.1, che rappresentano l'evoluzione della tariffa minima (basata sia sulla distanza percorsa nei viaggi, sia sulla durata dei viaggi stessi).

Aumento progressivo degli utenti che utilizzano il servizio

Numero utenti che usufruiscono del servizio	Utenti che usufruiscono del servizio(rispetto al numero totale di utenti target) [%]	Numero dei veicoli	Numero delle stazioni di ricarica	Chiamate perse [%]	Tempo di attesa medio [min]	Overhead totale [%]
2433	10%	344	233	4.99%	4.82	24.14%
4865	20%	659	243	4.99%	4.53	23.32%
7785	30%	947	391	4.97%	4.22	21.88%
10217	40%	1238	396	4.92%	4.13	21.83%
12650	50%	1547	404	4.89%	3.99	21.59%
15083	60%	1761	633	4.93%	3.82	20.34%
17515	70%	1967	700	4.91%	3.80	20.14%
20435	80%	2377	727	4.94%	3.77	20.22%
22867	90%	2563	956	4.99%	3.75	19.78%
25300	100%	2804	986	4.98%	3.73	19.80%

Tabella 9.1: Aumento graduale degli utenti coinvolti nel servizio ed ottimizzazione ad ogni step

In generale è chiaro come sia necessario che in futuro più persone possibili passino a questa forma di mobilità condivisa, una volta che essa sia esistente ed avviata. Maggiore è il numero di utenti coinvolti migliore è l'efficienza del servizio e i costi di conseguenza diminuiscono.

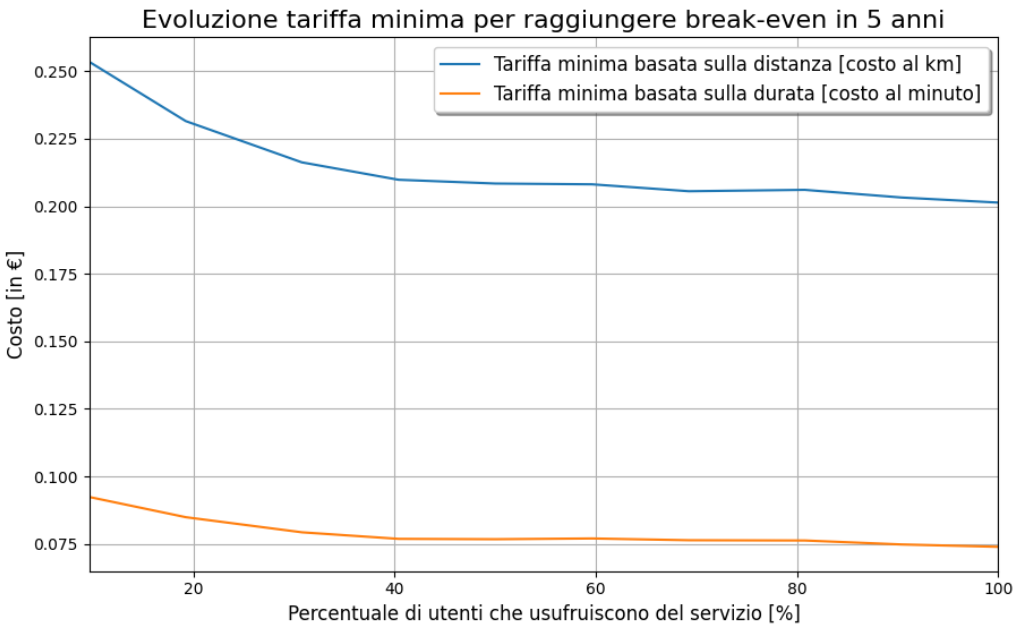


Figura 9.1: Evoluzione tariffa minima nel caso di servizio sempre ottimizzato

Tuttavia è interessante notare come la tariffa minima aumenti relativamente poco anche considerando meno utenti: se il 40% degli utenti usufruissero del servizio i costi minimi al chilometro aumenterebbero solo del 5% circa, passando a 0.210 €/km. Lo stesso ragionamento vale per il costo al minuto che passarebbe da 0.0739 €/min a 0.0769 €/min, sempre considerando il 40% degli utenti totali.

Si procede ora ad analizzare il secondo caso: il numero di veicoli e il numero di colonnine rimangono fissati e si procede a simulare il servizio considerando progressivamente meno utenti e quindi meno viaggi.

Aumento progressivo degli utenti che utilizzano il servizio (parametri del servizio fissati)

Numero utenti che usufruiscono del servizio	Utenti che usufruiscono del servizio(rispetto al numero totale di utenti target) [%]	Numero dei veicoli	Numero delle stazioni di ricarica	Chiamate perse [%]	Tempo di attesa medio [in minuti]	Overhead totale [%]
2433	10%	2804	986	0.00%	0.20	1.72%
4865	20%	2804	986	0.00%	0.57	4.28%
7785	30%	2804	986	0.03%	1.03	6.83%
10217	40%	2804	986	0.10%	1.39	8.84%
12650	50%	2804	986	0.24%	1.74	10.79%
15083	60%	2804	986	0.44%	2.08	12.64%
17515	70%	2804	986	0.74%	2.40	14.26%
20435	80%	2804	986	2.13%	2.91	16.54%
22867	90%	2804	986	3.62%	3.29	18.09%
25300	100%	2804	986	4.98%	3.73	19.80%

Tabella 9.2: Aumento graduale degli utenti coinvolti nel servizio a numero di veicoli e colonnine prefissato (cioè senza rieseguire l'ottimizzazione)

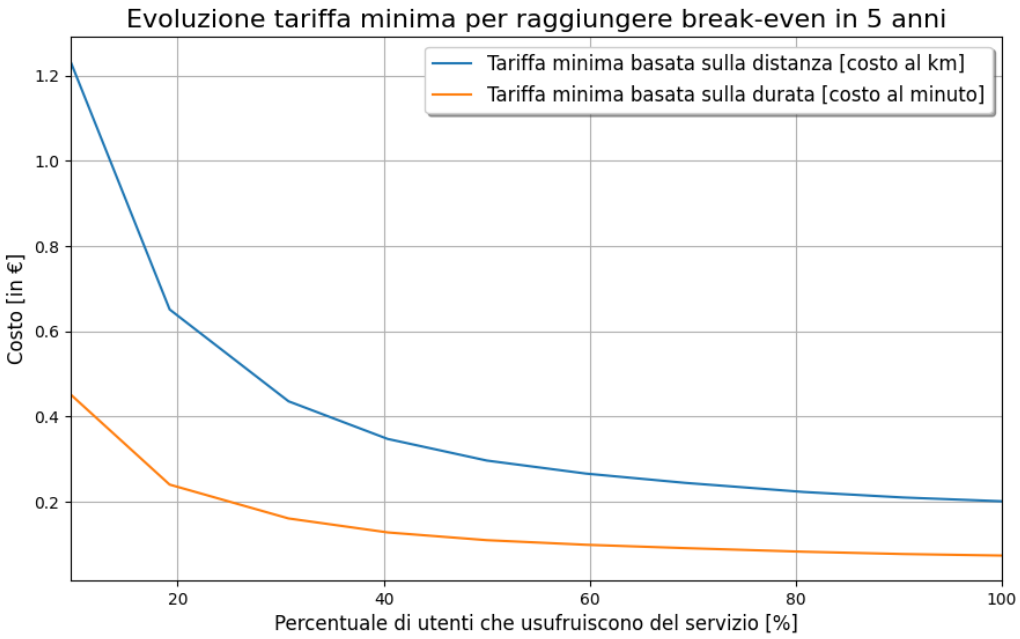


Figura 9.2: Evoluzione tariffa minima nel caso di servizio con parametri fissati

In questa situazione, poichè veicoli e colonnine sono già fissati, è chiaro che considerando meno utenti ci sia un servizio migliore per gli utenti (come visibile in Tab. 9.2); infatti i tempi di attesa diminuiscono con pochi utenti e vi è una quasi certezza di essere serviti per un corsa, anche durante le ore di punta del traffico. Tuttavia dal punto di vista economico lo scenario è altamente sconveniente.
In Fig. 9.2 si vede chiaramente come la tariffa minima lieviti enormemente considerando pochi utenti. Qualora vi fossero solo il 35% degli utenti previsti il servizio dovrebbe costare più del doppio.
In termini di fattibilità economica è quindi necessario mantenere sempre il servizio con le variabili ottimizzate, in quanto questo permette di contenere i costi e i prezzi delle tariffe, mantenendo comunque l'efficienza.

9.2. Analisi costi-benefici della dismissione dell'auto privata urbana

Le valutazioni economiche sono un aspetto importante nelle decisioni delle persone. Ci si è quindi posti la domanda: è effettivamente conveniente, per gli utenti target in analisi, passare da un servizio di mobilità privato a uno condiviso come quello studiato

in questa tesi?

9.2.1. Scelta del veicolo di riferimento

Il primo passo per rispondere alla domanda in questione è decidere un veicolo tipo di un cittadino abitante nell'area metropolitana di Brescia. Si considera quindi una piccola "city car" con motore a combustione interna: una Peugeut 208. In seguito vi sono i principali costi in cui incorre il proprietario dell'autovettura:

Figura 9.3: Peugeot 208

- **Costo di acquisto dell'automobile** 16900 €
- **Consumo urbano** 18km/L
- **Costo di assicurazione** 341.9 €/anno
- **Tassa di circolazione** 191 €/anno
- **Prezzo medio della benzina** 1.8 €/L

Poichè il chilometraggio annuale dei veicoli degli utenti target non è elevato come quello dei veicoli in sharing, non si può fare la stessa assunzione di un deprezzamento totale dopo 5 anni. Si procede, quindi, con l'introduzione di un modello che valuti il deprezzamento di un'autovettura nel corso del tempo, in funzione del chilometraggio annuale. I modelli più comuni in letteratura, per valutare tale svalutazione, seguono un andamento esponenziale, [20], [21]: si è deciso perciò di seguire questa linea, proponendo la seguente funzione di deprezzamento:

$$V(j) = V_0 \cdot \alpha(k)^j \tag{9.1}$$

V(j) rappresenta il valore del veicolo dopo j anni, V_0 è il prezzo di acquisto del veicolo nuovo, $\alpha(k)$ è un parametro che rappresenta il tasso di deprezzamento del veicolo (varia tra 0.8 e 0.86 in funzione del chilometraggio annuale (k)).
In Fig. 9.4 si può vedere un esempio di evoluzione del valore di un'automobile. Maggiore è la distanza percorsa dal veicolo, più veloce è il deprezzamento dello stesso. Si

può notare, inoltre, come dopo 5 anni il valore del veicolo sia inferiore al 50% del valore all'acquisto, qualunque sia il chilometraggio medio annuale.

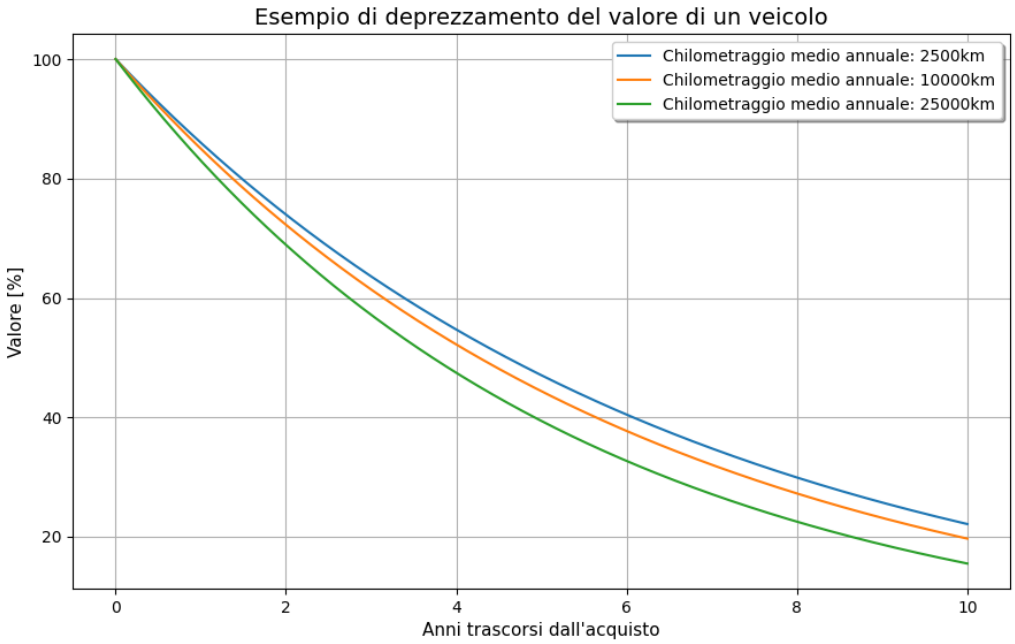


Figura 9.4: Deprezzamento di un veicolo privato

9.2.2. Analisi in funzione del chilometraggio annuale effettuato dagli utenti compatibili con il car-sharing

Una volta che sono state definite tutte le assunzioni nel paragrafo precedente, si può calcolare il costo al chilometro (sempre in funzione del chilometraggio annuale) in cui incorre un cittadino, utilizzando un veicolo privato.
Poichè nella funzione obiettivo nel Cap. 7 si sono definiti i costi su un orizzonte temporale di 5 anni, si è deciso, anche in questo caso, di calcolare il costo al chilometro per il proprietario dell'autovettura nei primi 5 anni in cui la possiede. I costi totali sostenuti sono i seguenti:

$$Costi = [V(0) - V(5)] \cdot costo\_acquisto + \quad \textbf{Deprezzamento veicolo}$$

$$\frac{k \cdot 5}{consumo\_urbano} \cdot prezzo\_benzina + \quad \textbf{Costo della benzina acquistata}$$

$$(tassa\_circolazione + prezzo\_assicurazione) \cdot 5 \quad \textbf{Costi di mantenimento}$$

dove V(0) rappresenta il valore percentuale del veicolo all'istante 0, cioè all'acquisto, mentre V(5) è il valore percentuale del veicolo dopo 5 anni, secondo la funzione 9.1. k rappresenta invece il chilometraggio annnuale medio effettuato. Gli altri parametri sono gli stessi definiti ad inizio sezione.

Il costo al chilometro è, perciò, calcolato come i costi totali sostenuti, diviso i chilometri totali percorsi nei 5 anni (cioè $k \cdot 5$).

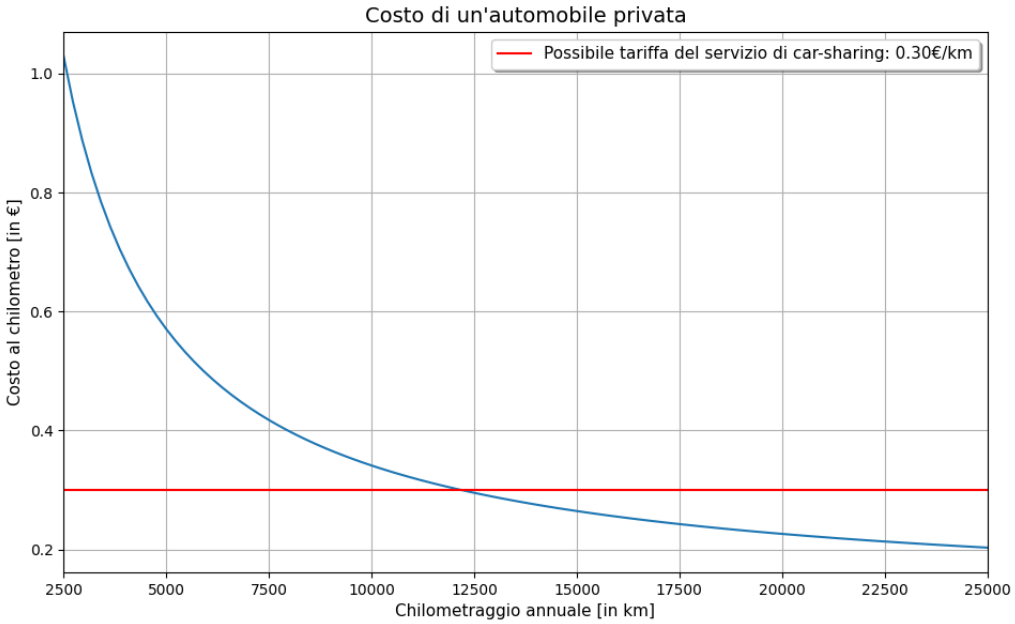


Figura 9.5: Costo al km di un veicolo privato

Come si nota del grafico in Fig. 9.5, minore è il chilometraggio percorso annualmente dalle vetture, maggiore è il loro costo al chilometro. È quindi chiaro come sia oppurtuno che i veicoli vengano sfruttati il più possibile nel corso del loro periodo di vita, al fine di garantire dei costi minori, e un utilizzo più efficiente delle risorse. Il passaggio da una forma di mobilità di tipo privato ad una condivisa assicurerebbe che i veicoli vengano utilizzati al pieno delle loro potenzialità (si ricorda come i veicoli in sharing studiati percorrano circa 60000 km all'anno).
Sempre considerando lo stesso grafico, gli utenti che percorrono meno di 12000 km l'anno avrebbero un costo al chilometro maggiore di 0.30 €(valore identificativo di una possibile tariffa del servizio di car-sharing). Per tali persone sarebbe economicamente conveniente utilizzare un servizio di sharing, come quello studiato in analisi, anzichè avere un veicolo di proprietà.

Si procede, dunque, a valutare la distribuzione del chilometraggio annuale mediamente percorso con auto private dagli utenti target, quelli cioè identificati come sharing compatibili. In questo caso, per valutare il chilometraggio totale, si sono considerati solo i viaggi effettuati da questi utenti all'interno dell'area di copertura di raggio 9 km (cioè che iniziano e finiscono all'interno di quest'ultima). Si ricorda, inoltre, come tali utenti eseguono più del 90% dei viaggi all'interno dell'area di copertura, quindi il numero di viaggi che vengono scartati è molto ridotto.

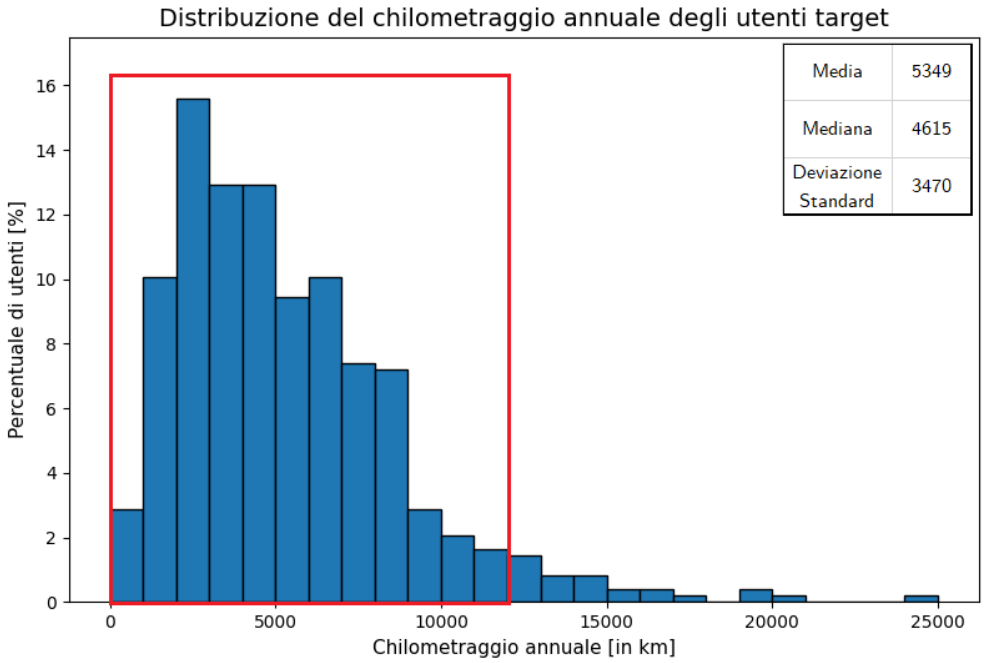


Figura 9.6: Distribuzione del chilometraggio annuale degli utenti target

In Fig. 9.6 viene evidenziata la porzione di utenti che percorrono mediamente meno di 12000 chilometri l'anno. Questa porzione rappresenta il 95% degli utenti target totali. Per loro sarebbe quindi conveniente utilizzare questa forma di moblità condivisa, anzichè possedere un veicolo privato.
Dunque, per la grande maggioranza degli utenti target è anche economicamente conveniente passare da una forma di mobilità di tipo privato a una forma di trasporto condiviso, come quella studiata in questo progetto di tesi.

10 | Conclusioni

Affrontando questa ricerca ci si era posti la seguente domanda: **Sarebbe fattibile, sia in termini di efficienza che in termini economici, allo stato attuale, sostituire i veicoli privati, utilizzati per la mobilità prevalentemente urbana, con un servizio di car-sharing autonomo ed elettrico?**
Lo sviluppo di una mobilità di tipo condiviso è ancora in fase abbastanza embrionale, quindi per molti cittadini è ancora necessario utilizzare veicoli privati per muoversi.
Tuttavia, come individuato in questa ricerca, c'è una porzione di cittadini per cui, già allo stato attuale, sarebbe pienamente fattibile rimpiazzare i propri veicoli privati con robotaxi in un servizio di car-sharing autonomo ed urbano. Questi utenti target compatibili con lo sharing rappresentano circa il 10% dei cittadini totali. Per essi, nel 95% dei casi è anche conveniente dal punto di vista economico usufruire del servizio.
Si è trovato, inoltre, come utilizzando questa forma di mobilità condivisa si potrebbe ridurre il numero di veicoli circolanti in area metropolitana di un fattore 9, riducendo enormemente l'area destinata ai parcheggi nelle città.
Dal punto di vista dell'efficienza si è visto come sia sempre attuabile la costruzione di un servizio di car-sharing urbano elettrico ed autonomo. Considerando un qualsiasi raggio dell'area di copertura il problema di ottimizzazione ha sempre restituito una soluzione e i vincoli di minima efficienza del servizio sono sempre stati rispettati.

Si procede quindi con il riassumere tutti gli specifici contributi di analisi di dati di percorrenza, di ottimizzazione e simulazione di questa tesi.
Il presente lavoro si basa su di un dataset di viaggi (circa 115 milioni) realmente percorsi da 33170 veicoli nella provincia di Brescia, tra il Gennaio 2019 e il Dicembre 2020. La forza di tale studio è quella di essere totalmente data-driven. Inoltre, questo progetto è unico nel suo genere in quanto si basa su viaggi percorsi da veicoli privati, e non da veicoli in servizi di sharing esistenti, come in altri articoli in letteratura. In particolare, grazie alla presenza in vari veicoli di box telematiche dotate di sensori GPS, è stato possibile monitorare costantemente gli spostamenti effettuati da un gran

numero di cittadini residenti nella provincia di Brescia. I dati sugli spostamenti sono stati poi aggregati in viaggi, in maniera tale da essere utilizzabili per la analisi svolte in questo progetto.
Il primo step è stato individuare e motivare una possibile area di copertura del servizio. Analizzando la distribuzione dei viaggi nel dataset, si è visto come la regione di copertura ottimale sia circolare, con centro del servizio nel centro di Brescia.
Successivamente, è stato necessario identificare gli utenti per cui sarebbe sensato dismettere i propri veicoli privati e passare ad un servizio di car-sharing autonomo. Questi utenti rappresentano il 3%-15%(in base alla dimensione del servizio di car-sharing) dei cittadini totali residenti nella area di copertura del servizio.
Una volta che sono stati identificati gli utenti target e i loro viaggi, si è costruito il simulatore del servizio di car-sharing. La scrittura di un programma in grado di rispecchiare fedelmente un servizio di car-sharing autonomo è stata la sfida più grande in questo progetto di tesi, e anche quella che ha richiesto un tempo maggiore. L'introduzione della parte di mobilità elettrica e delle relative infrastrutture di ricarica, in particolare, ha richiesto un grande sforzo e lavoro.
Sempre riguardo alla parte di mobilità elettrica, è stato necessario valutare delle politiche di ricarica che rendessero il più efficiente possibile il servizio. Analizzando la distribuzione temporale delle chiamate effettuate, e procedendo simulando il servizio di car sharing in diversi casi, si è notato come sia conveniente avere politiche di ricarica differenziate per il giorno (7-23) e la notte (23-7). In particolare è opportuno ricaricare completamente i veicoli in servizio durante la notte (periodo in cui il numero di viaggi da servire è estremamente basso), così da avere la flotta totalmente carica durante il giorno.
Dopo aver reso quanto più efficiente possibile il servizio di car-sharing, si è proceduto con l'ottimizzazione dal punto vista economico dei parametri del servizio (numero veicoli presenti, numero di colonnine e raggio dell'area circolare del servizio). Si è trovato come il raggio ottimale sia di 9km, utilizzando 2804 veicoli e con 986 stazioni di ricarica. Per raggiungere il break-even point economico e non essere quindi in perdita, è necessario stabilire una tariffa minima di 0.074€ al minuto, oppure di 0.201€ al chilometro. In generale si è visto che per qualsiasi raggio compreso tra gli 8km e i 15km il servizio di car-sharing elettrico ed autonomo è realizzabile, e le tariffe minime rimangono comunque abbastanza simili al caso migliore ottimizzato.
Il prezzo di un viaggio effettuato nel servizio si è stimato essere mediamente di 1.48€, tale valore è in linea con i prezzi dei mezzi pubblici nella città di Brescia (biglietto

ordinario di 1.40€). Inoltre per la grande maggioranza degli utenti target (il 95%) sarebbe economicamente conveniente passare a questa forma di car-sharing urbano e autonomo e dismettere i propri veicoli.
Attraverso un'analisi dinamica di progressiva e graduale crescita degli utenti che scelgono di usufruire del servizio si è capito come sia necessario valutare nella maniera più precisa possibile il numero di utenti target, al fine di mantenere i costi del servizio contenuti. Inoltre è chiaro come più utenti usufruiscano del servizio di car-sharing, maggiore sia l'efficienza del servizio stesso.
I viaggi nel servizio che hanno una percorrenza minore di 0.8km generano un overhead maggiore del 50%, cioè in media è maggiore la distanza percorsa dai robotaxi, senza utente a bordo, per andare nel luogo di chiamata, rispetto alla distanza del viaggio stesso. Intuitivamente sarebbe quindi opportuno avere un piano di tariffazione a fasce, in base alla lunghezza/durata del viaggio da effettuare (così come è in tutti i servizi di car-sharing attualmente esistenti).

10.1. Discussioni e sviluppi futuri

- Sarebbe possibile ottimizzare ulteriormente la collocazione delle colonnine di ricarica seguendo tecniche meta-euristiche presentate in alcuni studi, come ad esempio: in [8].
  Inoltre le politiche di ricarica si potrebbero lievemente perfezionare, dividendo in più sezioni la giornata, anzichè solo in due come presentato in questo lavoro.

- Al fine di efficientare ulteriormente il servizio di sharing sarebbe possibile introdurre una politica di riallocazione dei veicoli come suggerito per esempio nel lavoro di Weikl et al. [22].

- Nel Cap. 4 i criteri di selezione degli utenti target, definiti compatibili con un servizio di sharing, sono posti in maniera logica, ma abbastanza arbitraria. Sarebbe possibile svolgere una analisi dinamica in base al variare di questi criteri. In particolare, in questo progetto, viene utilizzato il vincolo che il 90% dei viaggi effettuati dall'utente sia all'interno dell'area di copertura del servizio, per identificare gli utenti target.
  Si potrebbero valutare i cambiamenti nell'efficienza e nel costo del servizio variando tale parametro.

- Nel progetto di tesi si assume che la distribuzione dei viaggi nel servizio di car-sharing rimanga invariata nel tempo (in particolare tutte le settimane si ripetono con la stessa distribuzione di chiamate della prima), questa è chiaramente una limitazione. Si potrebbero studiare le politiche ottimali per far fronte a una variazione nel tempo di tale distribuzione. Per esempio si potrebbe analizzare come reagire di fronte a improvvisi picchi di domanda, o come cercare di contenere i costi dovuti ad un utilizzo minore del servizio (ad esempio in periodi di vacanze).

Bibliografia

[1] Ambiente urbano, dagli spazi urbani agli ecosistemi urbani, https://www.eea.europa.eu/it/articles/ambiente-urbano, 2010.

[2] Sergio Savaresi. Auto a guida autonoma: una nuova era per la mobilità pubblica (e per l'ambiente), https://www.agendadigitale.eu/smart-city/auto-a-guida-autonoma-una-nuova-era-per-la-mobilita-pubblica-e-per-lambiente/, 2022.

[3] Daniel J Fagnant and Kara M Kockelman. The travel and environmental implications of shared autonomous vehicles, using agent-based model scenarios. *Transportation Research Part C: Emerging Technologies*, 40:1–13, 2014.

[4] Jeffery B Greenblatt and Susan Shaheen. Automated vehicles, on-demand mobility, and environmental impacts. *Current sustainable/renewable energy reports*, 2(3):74–81, 2015.

[5] James M Anderson, Kalra Nidhi, Karlyn D Stanley, Paul Sorensen, Constantine Samaras, and Oluwatobi A Oluwatola. *Autonomous vehicle technology: A guide for policymakers*. Rand Corporation, 2014.

[6] Michele Cocca, Danilo Giordano, Marco Mellia, and Luca Vassio. Free floating electric car sharing in smart cities: Data driven system dimensioning. In *2018 IEEE International Conference on Smart Computing (SMARTCOMP)*, pages 171–178, 2018.

[7] Michele Cocca, Danilo Giordano, Marco Mellia, and Luca Vassio. Free floating electric car sharing: A data driven approach for system design. *IEEE Transactions on Intelligent Transportation Systems*, 20(12):4691–4703, 2019.

[8] Michele Cocca, Danilo Giordano, Marco Mellia, and Luca Vassio. Free floating electric car sharing design: Data driven optimisation. *Pervasive and Mobile Computing*, 55:59–75, 2019.

[9] Hussein Dia and Farid Javanshour. Autonomous shared mobility-on-demand: Melbourne pilot simulation study. *Transportation Research Procedia*, 22:285–296, 2017.

[10] Farid Javanshour, Hussein Dia, Gordon Duncan, Rusul Abduljabbar, and Sohani Liyanage. Performance evaluation of station-based autonomous on-demand car-sharing systems. *IEEE Transactions on Intelligent Transportation Systems*, 2021.

[11] Hongzhi Miao, Hongfei Jia, Jiangchen Li, and Tony Z Qiu. Autonomous connected electric vehicle (acev)-based car-sharing system modeling and optimal planning: A unified two-stage multi-objective optimization methodology. *Energy*, 169:797–818, 2019.

[12] Xiang Huo, Xinkai Wu, Ming Li, Nan Zheng, and Guizhen Yu. The allocation problem of electric car-sharing system: A data-driven approach. *Transportation Research Part D: Transport and Environment*, 78:102192, 2020.

[13] Riccardo Iacobucci, Benjamin McLellan, and Tetsuo Tezuka. Modeling shared autonomous electric vehicles: Potential for transport and power grid integration. *Energy*, 158:148–163, 2018.

[14] Maximilian A Richter, Johannes Hess, Christoph Baur, and Raphael Stern. Exploring the financial implications of operating a shared autonomous electric vehicle fleet in zurich. *Journal of Urban Mobility*, 1:100001, 2021.

[15] Veicoli - pubblico registro automobilistico, http://dati.istat.it/index.aspx?datasetcode=dcis_veicolipra, 2022.

[16] Henrik Becker, Francesco Ciari, and Kay W Axhausen. Modeling free-floating car-sharing use in switzerland: A spatial regression and conditional logit approach. *Transportation Research Part C: Emerging Technologies*, 81:286–299, 2017.

[17] Florian Dandl and Klaus Bogenberger. Comparing future autonomous electric taxis with an existing free-floating carsharing system. *IEEE Transactions on Intelligent Transportation Systems*, 20(6):2037–2047, 2018.

[18] Patrick M Bösch, Felix Becker, Henrik Becker, and Kay W Axhausen. Cost-based analysis of autonomous mobility services. *Transport Policy*, 64:76–91, 2018.

[19] Share_now milano, https://www.share-now.com/it/it/milan/, 2022.

[20] Ryan Logtenberg, James Pawley, and Barry Saxifrage. Comparing fuel and maintenance costs of electric and gas powered vehicles in canada. *2 Degrees Institute*, 2018.

[21] Kenneth Lebeau, Philippe Lebeau, Cathy Macharis, and Joeri Van Mierlo. How expensive are electric vehicles? a total cost of ownership analysis. In *2013 World Electric Vehicle Symposium and Exhibition (EVS27)*, pages 1–12. IEEE, 2013.

[22] Simone Weikl and Klaus Bogenberger. Relocation strategies and algorithms for free-floating car sharing systems. *IEEE Intelligent Transportation Systems Magazine*, 5(4):100–111, 2013.

A | Appendice

In questa appendice è possibile visualizzare il codice utilizzato per svolgere più ottimizzazioni parallele. Vi è definita la classe simulatore e poi il main con le funzioni necessarie a far partire il processo di ottimizzazione.

```
import pandas as pd
import numpy as np
import os
import bisect
import time
from tqdm import tqdm
import multiprocessing as mp
import datetime
from math import radians, cos, sin, asin, sqrt, exp
import scipy.optimize as sc_opt

class Simulator:
    ### definition simulator of autonomous and electric sharing service
    def __init__(self, path, range_to_test, radius, n_charging_stations):
        # Simulation data import
        self.path = path  ### path of the trip dataset
        self.df = pd.read_parquet(self.path) ### trip dataset to simulate
        # the service
        self.df.time_start = pd.to_datetime(self.df.time_start)
        self.df.time_end = pd.to_datetime(self.df.time_end)
        self.df = self.df.sort_values(by='time_start') ## sort by
        # time_start = time of the calls in the service
        self.df = self.df.reset_index()
        self.df.distance = self.df.distance / 1000 ## from meter
        # to kilometer
        self.df_list = []

        #read df containg the distribution of trips' starts,
        # used in order to place recharging stations
        self.zones_trips = pd.read_parquet(
            r"...\Distribuzione_partenze15.0.parquet",
            engine='fastparquet')

        ### order zones by the number of trips' starts
        self.zones_trips.sort_values('Starting_point',
                                     ascending=False,inplace=True)
```

```
self.maximum_number_stations = 20  # keep divisible by 4
# (or change code in InitializeRechargingStations)
self.InitializeRechargingStations(radius,
             n_charging_stations) # Place charging stations
self.every_station_occupied = False

# final_df contains information about the vehicles in the service
self.final_df=pd.DataFrame(columns=['index','latitude', 'longitude',
             'busy_until', 'distance', 'overhead',
             'dead_time', 'tot_calls', 'time_waiting_recharge',
                              'time_recharging','tot_recharges'])


self.EARLY = -1 ## used in order to setup an early stop
# in the simulation process

self.RANGE_TO_TEST = range_to_test  #number of vehicles in
# the service
# lists containing the vehicles in the service
self.free_cars = []
self.busy_cars = []
self.free_cars_recharging = []
# Set speed for an autonomous vehicle
self.L3VehicleSpeed = 15 # km/h

self.N_PARALLEL = 1 #one simulation at time per optimization

# Max time the user is willing to wait for the car to arrive.
self.USER_WAIT_FOR_CAR = 20 # [min]

# Parameters Car
self.battery_capacity = 60 #in kwh
self.efficiency_battery = 0.975
self.consumption = 0.2 # in kwh/km
self.recharging_power = 13 #in kw


### thresholds used in the charging policy of the service
self.threshold_send_recharge_day = 25
self.threshold_car_free_day = 30
self.threshold_send_recharge_night = 60
self.threshold_car_free_night = 90
## initialize thresholds and nighttime
self.threshold_send_recharge = self.threshold_send_recharge_day
self.threshold_car_free = self.threshold_car_free_day
self.night = False



# List of all the user wait time
self.user_wait_time = []
# List of the number of free vehicles at each iteration
```

```
        self.free_cars_v = []
        # List of all the skipped calls
        self.skipped_calls = []

    def InitializeRechargingStations(self, radius, N_charging_stations):
        ###
        # This function takes as input the radius of the coverage area
        # and the total number of the charging stations to place and
        # create a dataframe with the initialized stations
        ###

        #select just subregions inside the coverage area
        lat_centre_radius = 45.53936
        #taken from charging_places, always fixed
        lon_centre_radius = 10.22218 # coordinates of Brescia city centre
        rows_to_drop = []
        for index, row in self.zones_trips.iterrows():
            distance = haversine(lon_centre_radius,lat_centre_radius,
             (row['West']+row['East'])/2,(row['North']+row['South'])/2)
            if distance>radius:
                rows_to_drop.append(index)

        self.zones_trips.drop(rows_to_drop,inplace=True)

        # find maximum number of starting points
        max_starting_points = self.zones_trips['Starting_point'].max()
        counter_placed_stations = 0
        parameter_division = 1


        #now the number of charging station in the selected position
        # is proportional to the number of trips starting from there

        self.charging_stations = pd.DataFrame(columns=['latitude',
               'longitude','available_stations','occupied_stations'])
        for index,row in self.zones_trips.iterrows():
            if row['Starting_point']<max_starting_points/1.5 and \
                    row['Starting_point']>=max_starting_points/2:
                parameter_division = 4/3
            if row['Starting_point']<max_starting_points/2 and \
                    row['Starting_point']>=max_starting_points/4:
                parameter_division = 2
            if row['Starting_point']<max_starting_points/4:
                parameter_division = 4
            if counter_placed_stations < N_charging_stations:
                self.charging_stations.loc[len(self.charging_stations)] = \
                [(row['North']+row['South'])/2,(row['West']+row['East'])/2,
                self.maximum_number_stations/parameter_division,0]
                counter_placed_stations += \
                    self.maximum_number_stations/parameter_division
                if counter_placed_stations > N_charging_stations:
                    self.charging_stations.loc[len
                                (self.charging_stations)-1] = \
```

```
                    [(row['North'] + row['South']) / 2,
                     (row['West'] + row['East']) / 2,
                    self.maximum_number_stations / parameter_division
                     - counter_placed_stations + N_charging_stations, 0]
            else:
                break
```

```
def FreeChargingStation(self, used_car,**kwargs):
    ###this function is used when a car leaves the charging station

    time_req = kwargs.get('time_req', None)
    ## find the charging zone where the car was recharging and free it
    latitude=used_car[0]
    longitude=used_car[1]
    index_selected_station = self.charging_stations.index[
        (self.charging_stations['latitude']==latitude) &
        (self.charging_stations['longitude']==longitude)]
    self.charging_stations.loc[index_selected_station,
                               'available_stations'] += 1
    self.charging_stations.loc[index_selected_station,
                               'occupied_stations'] -= 1
    self.every_station_occupied = False
    if time_req!=None:
        #now update State of Charge until the last moment
        time_delta = (time_req - used_car[12]).total_seconds()
        used_car[7] = self.newSOC(used_car[7], True,
                                  delta_time=time_delta)
        # update tot_time recharging and other parameters
        # of the vehicle
        used_car[10] = used_car[10] + time_delta / 3600
        used_car[12] = time_req
        used_car[13].append(used_car[7])
        used_car[14].append(used_car[12])
    #car no longer recharging
    used_car[8] = False

    return used_car

def SendToChargingStation(self, chosen_vehicle,actual_time):
    ### when a vehicle needs to recharge, this function finds
    # the closest available station and send the car there

    #find the closest charging station available
    lats = self.charging_stations.loc[self.charging_stations
                ['available_stations']>0,'latitude'].tolist()
    lons = self.charging_stations.loc[self.charging_stations
                ['available_stations']>0,'longitude'].tolist()

    if len(lats) == 0:
```

```
            self.every_station_occupied=True
            return [], False  ## not currently available stations

        vehicle_pos = [chosen_vehicle[0],chosen_vehicle[1]]
        distances = 1.5 * haversine_distance(lats, lons,
             vehicle_pos) #find distances of the stations from the car

        # Choose the closest station
        index_min = min(range(len(distances)), key=distances.__getitem__)
        index_selected_station = \
            self.charging_stations.index[(self.charging_stations['latitude']
                                          ==lats[index_min]) &
                                         (self.charging_stations['longitude']==
                                          lons[index_min])]
        self.charging_stations.loc[index_selected_station,'available_stations'] -= 1
        self.charging_stations.loc[index_selected_station, 'occupied_stations'] += 1

        # now update all data of the current vehicle
        distance_to_station = distances[index_min]
        time_to_station = 3600*distance_to_station/self.L3VehicleSpeed #this
        # is a time in seconds

        chosen_vehicle[7]=self.newSOC(chosen_vehicle[7], charging=False,
                            tot_distance=distance_to_station)
        newtime=actual_time+pd.to_timedelta((time_to_station), unit='s')

        chosen_vehicle[13].append(chosen_vehicle[7])
        chosen_vehicle[14].append(newtime)
        chosen_vehicle[0] = lats[index_min]
        chosen_vehicle[1] = lons[index_min]
        chosen_vehicle[3] = chosen_vehicle[3]+distance_to_station
        chosen_vehicle[4] = chosen_vehicle[4]+distance_to_station
        chosen_vehicle[8] = True
        chosen_vehicle[11] = chosen_vehicle[11]+1
        chosen_vehicle[12] = newtime
        chosen_vehicle[15] = newtime+datetime.timedelta(seconds=900)

        return chosen_vehicle, True


    def InitializeCars(self, N_sharcars):

        ## this function initializes all the vehicles in the service
        ## here below all the tracked statistics during the simulation process

        # 0,        1,         2,          3,        4,        5,         6,
        # Latitude, longitude, busy_until, distance, overhead, dead_time, tot_calls,
        # 7,                8,               9,                              10,              11,
        # State of Charge,  Charging Status, time_waiting_free_stations, time_recharging, tot_charges,
        # 12,                  13,             14,                 15,               16,
        # last_check_recharge, SOC evolution ,time_soc_change,  new_need_check  ,#id_car

        ########### use list  as containers of cars
```

```
        self.free_cars=[]
        self.busy_cars=[]
        self.free_cars_recharging = []
        #initialize all the cars
        for i in range(N_sharcars):
            self.busy_cars.append([self.df.iloc[i].latitude_start,
                                   self.df.iloc[i].longitude_start,
                                   self.df.iloc[i].time_start, 0, 0, 0, 0, 100, False,
                                   0, 0, 0,self.df.iloc[0].time_start,[100],[self.df.iloc[0].time_start],
                                   self.df.iloc[i].time_start,i])



    def FreeCarAvailable(self, row, request_time, request_pos):

        ## this function finds the nearest available car to che place of the call and simulate the trips

        # Compute the distances for all the free cars
        lats = [i[0] for i in self.free_cars]
        lons = [i[1] for i in self.free_cars]
        soc = [i[7] for i in self.free_cars]
        keys = [i[16] for i in self.free_cars]
        for i in self.free_cars_recharging:
            lats.append(i[0])
            lons.append(i[1])
            soc.append(i[7])
            keys.append(i[16])
        distances = 1.5 * haversine_distance(lats, lons, request_pos)
        # Choose the closest vehicle and send it to the user
        distance_to_user = distances.min()
        indices = [i for i, x in enumerate(distances) if x == distance_to_user]
        if len(indices)>1: #two or more available cars at the same charging station
            #let's find the car with greatest soc
            soc_same_place = [soc[i] for i in indices]
            chosen_vehicle = indices[np.argmax(soc_same_place)]
        else:
            chosen_vehicle = np.argmin(distances)

        # This car has to go to the user
        dt_to_user = 3600*distance_to_user/self.L3VehicleSpeed #this is a time in seconds

        if dt_to_user/60 > self.USER_WAIT_FOR_CAR: # If the user has to wait too much, the call is dropped
            self.skipped_calls.append(1)
        else:
            self.user_wait_time.append(dt_to_user)
            self.skipped_calls.append(0)
            # Now the user has the car and can use it
            dt_trip = (row.time_end - row.time_start).delta/1e9
            # now check whether the car was recharging or not
            N_free_cars = len(self.free_cars)
            if chosen_vehicle<N_free_cars:
                used_car = self.free_cars.pop(chosen_vehicle)
            else:
```

```
            used_car = self.free_cars_recharging.pop(chosen_vehicle-N_free_cars)

        if used_car[8] == True: #charging=true --> I free the charging stations
            used_car = self.FreeChargingStation(used_car,time_req=request_time)
            dead_time = 0

        else:
            # save the dead time (time when the vehicle was unused)
            dead_time = (request_time - used_car[2]).delta / 1e9



        # Now update the car statistics
        new_soc = self.newSOC(used_car[7],charging=False,tot_distance=row.distance+distance_to_user)
        new_time = request_time + pd.to_timedelta((dt_to_user + dt_trip), unit = 's')
        used_car[13].append(new_soc)
        used_car[14].append(new_time)
        used_car[15]=new_time

        used_car[0] = row.latitude_end
        used_car[1] = row.longitude_end
        used_car[2] = new_time
        used_car[3] = used_car[3] + row.distance + distance_to_user
        used_car[4] = used_car[4] + distance_to_user
        used_car[5] = used_car[5] + dead_time/3600
        used_car[6] = used_car[6] + 1
        used_car[7] = new_soc
        used_car[8] = False
        used_car[11] = used_car[11]
        used_car[15] = new_time

        self.insertOrderedVehicle(used_car,busy_vehicle=True) ## insert the vehicle in the busy_car list

    def NoFreeCar(self, row, request_time, request_pos):

        # There are no free cars
        self.skipped_calls.append(1)



    def CheckFreeVehicles(self, request_time):
        # This function updates all the vehicles' statistics until
        # 'request time'(time of the most recent call in the simulation process)
        # and than check if there are free vehicles available to serve the call
        #### implementation using list

        # counter needed in order to know the number of elements(vehicles) modified, so that it is possible
        # to reorder the list at the end of the loop
        counter_modified_vehicles = 0
        i = 0
        while i < len(self.busy_cars): ##update vehicles in busy_car list
            vehicle = self.busy_cars[i]  # lists are copied by reference!!!!
            if vehicle[15] > request_time:
```

```
            # new_need_check > request time
            # no more need to check other cars
            break
        param_car = vehicle  # [copied by reference] the elements in self.busycars are modified as well
        if param_car[8] == True and param_car[12] < request_time:
            # the vehicle is recharging, and its SOC has to be updated

            time_delta = (request_time - param_car[12]).total_seconds()
            delta_second_check = 900 ## the SOC level of the cars is updated
            # every 900 seconds when recharing
            if time_delta > 0:
                param_car[12] = request_time  # update new last_check_recharge
                param_car[7] = self.newSOC(param_car[7], True, delta_time=time_delta)
                param_car[10] = param_car[10] + time_delta / 3600  # time spent recharging, in hours

                #keep track of soc evolution of the vehicle
                param_car[13].append(param_car[7])
                param_car[14].append(param_car[12])

                param_car[15] = param_car[12]
                if param_car[7] == 100:  # the car is completely charged, leave the charging station
                    param_car[15] = param_car[12]
                    param_car = self.FreeChargingStation(param_car)
                    car = self.busy_cars.pop(i)
                    i -= 1  # with pop one element removed in the list
                    self.free_cars.append(car)
                elif param_car[7] > self.threshold_car_free: # the SOC level is now over
                    # the high_threshold, but the car still remains in charge

                    param_car[15] = param_car[12] + datetime.timedelta(seconds=delta_second_check)

                    #move vehicle from busy_car list to free_car_recharging list
                    car = self.busy_cars.pop(i)
                    i -= 1  # with pop one element removed in the list
                    self.insertOrderedVehicle(car, busy_vehicle=False)
                else: #the car remains busy, update new time when it has to be checked again
                    param_car[15] = param_car[12] + datetime.timedelta(seconds=delta_second_check)
                    counter_modified_vehicles += 1

        else:

            if param_car[2] <= request_time: # the vehicle has finished a trip
                if param_car[7] > self.threshold_send_recharge:
                    # soc greater than lower threshold = 25%/60%,
                    # vehicle is free and remains in the position where the trip ended

                    #move car from busy list to free_cars list
                    car = self.busy_cars.pop(i)
                    i -= 1
                    self.free_cars.append(car)
                else:
                    # else the car must be sent to recharge
                    if self.every_station_occupied == True:
```

```
                        param_car[15] = param_car[15] + datetime.timedelta(
                            seconds=600)  # every station occupied, wait 10 minutes
                        counter_modified_vehicles += 1

                    else:
                        #send the car to the charging station
                        used_car, available = self.SendToChargingStation(param_car, request_time)
                        if available == True:
                            # there's at least one station free
                            # this is needed to deal with the limit case of the car going to
                            # occupy the last available station

                            # update time waiting to be sent to charging station
                            used_car[9] = used_car[9] + (request_time -
                                                         used_car[2]).total_seconds() / 3600
                            counter_modified_vehicles += 1
                        else:
                            # the previous car occupied the last station, no more free
                            param_car[15] = param_car[15] + datetime.timedelta(
                                seconds=600)  # every station occupied, wait 10 minutes
                            counter_modified_vehicles += 1

        i += 1
    ## reorder the elements modified in the list
    self.reOrderFrontVehicles(counter_modified_vehicles, True)

    # now update free cars recharging
    counter_modified_vehicles = 0
    i = 0
    while i < len(self.free_cars_recharging):
        vehicle = self.free_cars_recharging[i]
        if vehicle[15] > request_time:
            # new_need_check > request time
            # no more need to check other cars
            break
        param_car = vehicle
        if param_car[12] < request_time:
            # if we passed the last check
            time_delta = (request_time - param_car[12]).total_seconds()
            delta_second_check = 900  # next update after 15 minutes
            if time_delta > 0:
                param_car[12] = request_time  # update new last_check_recharge
                param_car[7] = self.newSOC(param_car[7], True, delta_time=time_delta)
                param_car[10] = param_car[10] + time_delta / 3600  # time spent recharging, in hours
                param_car[13].append(param_car[7])
                param_car[14].append(param_car[12])
                if param_car[7] == 100:  # the car is completely charged, leave the charging station
                    param_car = self.FreeChargingStation(param_car)
                    car = self.free_cars_recharging.pop(i)
                    i -= 1
                    self.free_cars.append(car)
                else:
                    param_car[15] = param_car[12] + datetime.timedelta(seconds=delta_second_check)
```

```
                    counter_modified_vehicles += 1
            i += 1

        self.reOrderFrontVehicles(counter_modified_vehicles, False)

        ### now update free vehicles
        counter_updated_vehicles = 0
        if self.night == True and self.every_car_to_recharge == False:
            # when night comes, recharge policy change-> send free cars under thres to recharge
            i = 0
            while i < len(self.free_cars):
                vehicle = self.free_cars[i]
                if vehicle[7] < self.threshold_send_recharge:
                    vehicle[15] = request_time
                    car = self.free_cars.pop(i)
                    i -= 1
                    self.insertOrderedVehicle(car, True)
                    counter_updated_vehicles += 1
                i += 1
            if counter_updated_vehicles == 0:
                self.every_car_to_recharge = True

    def reOrderFrontVehicles(self,number_modified_cars,busy_list):
        #takes as input the number of car in the front of the list that must be reordered

        #reorder the list
        if busy_list == True: #busy_car_list
            list_mod_cars=[]
            while number_modified_cars>0:
                #remove cars from the list and add again in the correct position
                list_mod_cars.append(self.busy_cars.pop(0))
                number_modified_cars -=1
            while len(list_mod_cars)>0:
                vehicle = list_mod_cars.pop(0)
                self.insertOrderedVehicle(vehicle,True)

        if busy_list == False: #free_recharging_car_list
            list_mod_cars = []
            while number_modified_cars > 0:
                #remove cars from the list and add again in the correct position
                list_mod_cars.append(self.free_cars_recharging.pop(0))
                number_modified_cars -= 1
            while len(list_mod_cars) > 0:
                vehicle = list_mod_cars.pop(0)
                self.insertOrderedVehicle(vehicle, False)




    def insertOrderedVehicle(self,vehicle,busy_vehicle):

        ## this function inserts the vehicle passed as input in the correct position in the list,
```

```
        # according to new_need_check parameter, so the list remains always ordered.


        if busy_vehicle==True: #busy_car_list
            if len(self.busy_cars)==0:
                self.busy_cars.append(vehicle)
                return
            new_check = vehicle[15]
            keys = [car[15] for car in self.busy_cars]
            index = bisect.bisect(keys, new_check) #find the index of the correct position
            self.busy_cars.insert(index, vehicle)


        if busy_vehicle == False: #free vehicle recharging list
            if len(self.free_cars_recharging) == 0:
                self.free_cars_recharging.append(vehicle)
                return

            new_check = vehicle[15]

            keys = [car[15] for car in self.free_cars_recharging]
            index = bisect.bisect(keys, new_check)
            self.free_cars_recharging.insert(index, vehicle)


    def newSOC(self, old_soc, charging, *args, **kwargs):
        #this function gives as output the new State of Charge of the car in percentage terms
        delta_time_in_sec = kwargs.get('delta_time',None) #in seconds
        distance = kwargs.get('tot_distance',None) #in km
        new_soc = 0
        if charging == True: #tha car is recharging
            charge_battery = self.efficiency_battery * self.recharging_power * delta_time_in_sec/3600
            pct_charge = charge_battery/self.battery_capacity
            new_soc = old_soc + pct_charge*100 ## new soc level
            if new_soc>100:
                new_soc=100
        else: #i'm using the car->discharging
            pct_used_power = self.consumption * distance/self.battery_capacity
            new_soc = old_soc - pct_used_power*100
            if new_soc<0:
                new_soc=0
                print('the charge wouldn\'t be enuogh to complete the trip')

        return new_soc




    def createData(self, N):
    #create summary output of the simulation
        v0, v1, v2, v3, v4, v5, v6, v7 = [], [], [], [], [], [], [], []
        self.evolution_soc=[]
```

```
        self.time_soc=[]

        for car in self.free_cars+self.free_cars_recharging+self.busy_cars:
            v0.append(car[16])
            v1.append(car[0])
            v2.append(car[1])
            v3.append(car[2])
            v4.append(car[3])
            v5.append(car[4])
            v6.append(car[5])
            v7.append(car[6])
            self.final_df.loc[len(self.final_df)] = [car[16],car[0],car[1],car[2],car[3],
                                                     car[4],car[5],car[6],car[9],car[10],car[11]]
            self.evolution_soc.append(car[13])
            self.time_soc.append(car[14])

    def Simulation(self, N):
        # This function performs the simulation process
        self.InitializeCars(N)

        users_wait_time, free_cars, skipped_calls = [], [], []

        # data of a trip
        info_col = ['time_start', 'latitude_start', 'longitude_start', 'time_end',
                    'latitude_end', 'longitude_end', 'distance']

        # scroll through the dataframe of the trips
        for index, row in tqdm(self.df[info_col].iterrows(), total = len(self.df)):

            # When and where the call has been received
            request_time = row.time_start
            request_pos = np.array([row.latitude_start, row.longitude_start])

            #### changes in recharging thresholds
            if self.night == False and (request_time.hour>=23 or request_time.hour<7): #night came
                self.threshold_car_free = self.threshold_car_free_night
                self.threshold_send_recharge = self.threshold_send_recharge_night
                self.night = True
                self.every_car_to_recharge = False #needed in check free vehicles

            if self.night == True and (request_time.hour>=7 and request_time.hour<23): #morning glory
                self.threshold_car_free = self.threshold_car_free_day
                self.threshold_send_recharge = self.threshold_send_recharge_day
                self.night=False

            # Find the free vehicle at this moment in time
            self.CheckFreeVehicles(request_time)
            self.free_cars_v.append(len(self.free_cars)+len(self.free_cars_recharging))

            # Look for the cars and see if one is free
            if (len(self.free_cars)+len(self.free_cars_recharging)) > 0:
                # Free vehicles are available, find the closest to the user
                self.FreeCarAvailable(row, request_time, request_pos)
```

```
            else:
                # There are no free vehicles!
                self.NoFreeCar(row, request_time, request_pos)


        ## here the simulation is completed, let's create some summary data and send it as outputs

        time_span_simulation = [self.df.time_start.min(),self.df.time_end.max()]

        self.createData(N)
        return self.final_df, self.user_wait_time, self.free_cars_v, self.skipped_calls, \
               time_span_simulation, self.evolution_soc,self.time_soc

    def Run(self):

        ## this function runs the simulation
        ## a bit useless but remains from a previous version of the simulator
         if self.N_PARALLEL == 1:
            res = [self.Simulation(self.RANGE_TO_TEST[0])]

         return res

    def ShowInfo(self,path):

        ## this function displays info of the current simulation and save them also in a txt file
        print('--- Simulator Settings ---')
        print('Dataset imported from: {}'.format(self.path))
        print('Desired range of analysis: {}'.format(self.RANGE_TO_TEST))
        if self.EARLY != -1:
            print('Early stop: {}'.format(self.EARLY))
        if self.N_PARALLEL != -1:
            print('Parallel processes: {}'.format(self.N_PARALLEL))
        print('\n--- Dataset Info ---')
        print('Simulated trips: {}'.format(len(self.df)))
        print('Simulation start: {}'.format(self.df.time_start.min()))
        print('Simulation end: {}'.format(self.df.time_end.max()))
        print(self.df.time_end.idxmax())
        print('Time span: {}'.format(self.df.time_end.max() - self.df.time_start.min()))
        appendNewLine(path, '--- Simulator Settings ---')
        appendNewLine(path, 'Dataset imported from: ' + str(self.path))
        appendNewLine(path,' ')
        appendNewLine(path,'--- Dataset Info ---')
        appendNewLine(path,'Simulated trips: '+ str(len(self.df)))
        appendNewLine(path,'Simulation start: '+ str(self.df.time_start.min()))
        appendNewLine(path, 'Simulation end: ' + str(self.df.time_start.max()))
        appendNewLine(path,'Time Span: '+str(self.df.time_end.max() - self.df.time_start.min()))
        appendNewLine(path, ' ')




    def SetupEarlyStop(self, EARLY):
```

```
        ## used to stop early the simulation (until the EARLYth trip)
        self.df = self.df[:EARLY]
        self.EARLY = EARLY


    def SetupBeginSimulation(self, date):
        ## setup beginning of the simulation by date
        initial_date = pd.Timestamp(datetime.datetime.strptime(date, "%Y%m%d %H:%M:%S"))
        self.df = self.df.loc[(self.df['time_start'] >= initial_date)]

    def SetupEndSimulation(self, date):
        ## setup end of the simulation by date
        final_date = pd.Timestamp(datetime.datetime.strptime(date, "%Y%m%d %H:%M:%S"))
        self.df = self.df.loc[(self.df['time_end'] <= final_date)]

    def MultiplicationData(self, times):
        # this function puts together more weeks of trip data
        pd.options.mode.chained_assignment = None  # default='warn' ## avoid some useless warnings

        begin_first_week = self.df.iloc[0].time_start
        end_first_week = begin_first_week + datetime.timedelta (days=7)
        df_list = []
        df_list.append(self.df.loc[(self.df['time_start'] >= begin_first_week) &
                                   (self.df['time_end'] <= end_first_week)])
        for i in range(times-1):
            df_temporary = self.df.loc[(self.df['time_start'] >= begin_first_week +
                                        datetime.timedelta(days=7*(i+1)) ) &
                                       (self.df['time_end'] <= end_first_week +
                                        datetime.timedelta(days=7*(i+1)))]

            ## here all the trips in a specific week are modified in order
            # to be overlapping with the first week
            df_temporary['time_start'] = df_temporary['time_start'].\
                apply(lambda x: x-datetime.timedelta(days=7*(i+1)))
            df_temporary['time_end'] = df_temporary['time_end'].\
                apply(lambda x: x - datetime.timedelta(days=7 * (i+1)))
            df_list.append(df_temporary)

        ## concatenate all the weeks of trips and order by time_start
        self.df = pd.concat(df_list).sort_values(by= 'time_start').reset_index()
        pd.options.mode.chained_assignment = 'warn'  # default='warn'

### here we are outside of the simulator class
# Useful Functions

def haversine_distance(lats, lons, request_pos):
    # compute haversine distance

    # convert decimal degrees to radians
    lat1, lon1 = np.radians(lats), np.radians(lons)
    lat2 = np.radians([request_pos[0] for _ in range(len(lats))])
    lon2 = np.radians([request_pos[1] for _ in range(len(lats))])
```

```
    # haversine formula
    dlon = lon2 - lon1
    dlat = lat2 - lat1
    a = np.sin(dlat/2)**2 + np.cos(lat1) * np.cos(lat2) * np.sin(dlon/2)**2
    c = 2 * np.arcsin(np.sqrt(a))
    r = 6371
    return c*r


def haversine(lon1, lat1, lon2, lat2):
    #haversine distance but with different inputs
    # convert decimal degrees to radians
    lon1, lat1, lon2, lat2 = map(radians, [lon1, lat1, lon2, lat2])

    # haversine formula
    dlon = lon2 - lon1
    dlat = lat2 - lat1
    a = sin(dlat/2)**2 + cos(lat1) * cos(lat2) * sin(dlon/2)**2
    c = 2 * asin(sqrt(a))
    r = 6371 # Radius of earth in kilometers. Use 3956 for miles
    return c * r


def evaluation_opt_function(df,user_wait,skipped_calls,time_span,n_cars,n_stations,path):

    ## function used to evaluate the objective function value
    ### parameters used in the objective function
    cost_car = 35000 #euros
    discount = 0.30
    lifetime = 5 #in years
    maintenance = 1000 #euros per year
    price_energy = 0.30 #euros per Kwh
    station_cost = 5000
    parking_rental = 500 #euros per year

    ## compute useful statistics (dependent variables) of the current simulation
    frac_skipped_calls = sum(skipped_calls)/len(skipped_calls)
    time_sim = time_span[1]-time_span[0]
    time_sim = time_sim.total_seconds()/(24*3600)
    avg_distance_per_day_per_car = df['distance'].sum()/(len(df)*time_sim)
    trips_per_day = df['tot_calls'].sum()/(len(df)*time_sim) #per car
    frac_overhead = df['overhead'].sum()/(df['distance'].sum()+df['overhead'].sum())
    avg_wait = sum(user_wait) / len(user_wait) /60 #wait in minutes

    ## compute costs
    fleet_cost = n_cars*(cost_car*(1-discount)+maintenance*lifetime)
    energy_cost = df['time_recharging'].sum() * 13 * (365*lifetime*price_energy)/time_sim
    charging_stations_cost = n_stations*(station_cost+parking_rental*lifetime)

    ## objective function value
    function_val = (fleet_cost+energy_cost+charging_stations_cost)/1000000 #in million of euro

    ## let's check if the current solution respects the constraints of the optimization problem
```

```
    ### variables used in order to manage the solutions which don't respect the constraints
    base_millions=40
    multiplicative_factor=function_val/base_millions

    if frac_skipped_calls>0.05:
        ### add an exponential penalty in function value if the constraint is not respected
        function_val = function_val + exp((frac_skipped_calls - 0.05) * 100) * multiplicative_factor
        appendNewLine(path, 'Too many skipped calls')
    if avg_distance_per_day_per_car < 150:
        ### penalty if the constraint is not respected
        function_val = function_val + 1000
        appendNewLine(path, 'Not enough km per day per car')
    if frac_overhead > 0.3:
        ### penalty if the constraint is not respected
        function_val = function_val + 1000
        appendNewLine(path, 'Too much overhead')
    if avg_wait > 6:
        ### penalty if the constraint is not respected
        function_val=function_val+(avg_wait-6)*10
        appendNewLine(path, 'Average wait too big')

    return function_val


def objective_fun(x,radius):
    ### This function perform an iteration in the optimization process
    # and returns as output the value of the objective function

    global numero_iterazioni #total number of iterations

    # path where to save output of this process
    path_save_data = r'...\Risultato_radius'+str(radius)+'.txt'

    n_cars = x[0] #cars
    n_charging_stations = x[1]

    integer_n_cars = [int(round(n_cars,0))] #number of vehicles
    index_closest_radius = find_closest_index_radius(radius)
    radius_considered = range_radius[index_closest_radius] #radius of the coverage area
    integer_n_stations = float(round(n_charging_stations,0)) #number of the charging stations

    #setup the simulation class
    sim = Simulator(paths[index_closest_radius], integer_n_cars, radius_considered, integer_n_stations)

    #setup simulation parameters
    begin_simulation_date = "20190107 00:00:00"  # 'YYYYMMDD hh:mm:ss' format string
    sim.SetupBeginSimulation(begin_simulation_date)
    sim.MultiplicationData(multiplication_data_variable[index_closest_radius])
    end_simulation_date = "20190114 00:00:00"  # 'YYYYMMDD hh:mm:ss' format string
    sim.SetupEndSimulation(end_simulation_date)
    if numero_iterazioni==0:
```

```
        sim.ShowInfo(path_save_data)

    #run simulation
    result = sim.Run()

    #collect results of the simulation
    res = result[0]
    df = res[0]
    user_wait = res[1]
    free_cars = res[2]
    skipped_calls = res[3]
    time_span = res[4]
    evolution_soc = res[5]
    time_soc = res[6]



    numero_iterazioni+=1

    ## print variables in order to keep track of the optimization evolution
    print('Iterazione numero: ',numero_iterazioni)
    print('N AUTO: ', integer_n_cars,'____', n_cars)
    print('RAGGIO: ', range_radius[index_closest_radius],'____', radius)
    print('N Stazioni: ', integer_n_stations)
    val_function = evaluation_opt_function(df, user_wait, skipped_calls, time_span,
                                           integer_n_cars[0],integer_n_stations,path_save_data)
    appendNewLine(path_save_data, 'N AUTO: '+str(integer_n_cars) + '____'+ str(n_cars)+' '+
                  'N STAZIONI: '+str(integer_n_stations)+' '+'Objective fun Value :'+str(val_function)+
                  'Iterazione numero: '+str(numero_iterazioni))
    print('Valore funzione ottimo: ', val_function)
    print()

    return val_function


def find_closest_index_radius(raggio):
    ## find the corresponding index in range_radius, given a radius value
    index = 0
    while raggio>range_radius[index]:
        index += 1
    if raggio == range_radius[index]:
        return index
    else:
        dist1 = raggio - range_radius[index-1]
        dist2 = range_radius[index] - raggio
        if dist1<dist2:
            return index-1
        else:
            return index

def run_parallel_optimizations(indexes_radius_to_test):

    ### function used to run parallel optimizations, given different radius of the coverage area
```

```
        t_s = time.time()
        N_PARALLEL = mp.cpu_count()
        if len(indexes_radius_to_test)>N_PARALLEL:
            print('more parallel simulations than the number of cores')


        p = mp.Pool(len(indexes_radius_to_test))
        print('Running {} simulations in parallel'.format(len(indexes_radius_to_test)))
        res = p.map(minimization_function,indexes_radius_to_test)
        p.close()
        t_e = time.time()
        print('Done in {:.1f}s'.format(t_e - t_s))
        return res



def appendNewLine(file_name, text_to_append):
    # this function appends text as a new line at the end of file
    # Open the file in append & read mode ('a+')
    with open(file_name, "a+") as file_object:
        # Move read cursor to the start of file.
        file_object.seek(0)
        # If file is not empty then append '\n'
        data = file_object.read(100)
        if len(data) > 0:
            file_object.write("\n")
        # Append text at the end of file
        file_object.write(text_to_append)



def minimization_function(local_index_radius):
    ### This function starts the optimization process (in this case minimization of the objective function)
    actual_solution = [2000, 1200] ## initial point (not needed if the initial simplex is specified)

    ### here the initial simplex for 3 (10km,11km,12km of radius) parallel
    # optimizations is set (add for more parallel simulations)
    if local_index_radius == 8:
        #10 km
        simplesso_iniziale = np.array([[3700, 1300], [3790, 1000], [3740, 1122]])
    if local_index_radius == 12:
        #11km
        simplesso_iniziale = np.array([[4900, 1200], [4800, 1400], [4830, 1350]])
    if local_index_radius == 16:
        #12km
        simplesso_iniziale = np.array([[5700, 2400], [5750, 1900], [5800, 1800]])

    bounds_cars = (100, 20000)
    bounds_n_charging = (50, 10000)
    bounds = [bounds_cars, bounds_n_charging]
    radius = range_radius[local_index_radius]
```

```
    ## call the optimization function, using Nelder-Mead algorithm
    res = sc_opt.minimize(objective_fun, actual_solution, args=(radius,),method='Nelder-Mead', bounds=bounds,
                          options={'initial_simplex':simplesso_iniziale})
    return res


#### GLOBAL VARIABLES

DATA_DIR = r'...\dfs_brescia_radius' ## directory containg the trips dataset, one dataset for each
# radius of the coverage area


## this code was used for optimization for a radius of the coverage area from 8 km to 12km
##(with little modifications it was adapted for bigger radius)
range_radius = np.linspace(8,12,17).tolist()
range_radius = [round(i,2) for i in range_radius]
files = os.listdir(DATA_DIR)  #all files in the directory
paths = [os.path.join(DATA_DIR, file) for file in files] #create all paths, list format
paths = paths[9:]+paths[:9] #reorder the paths


multiplication_data_variable = [43,43,43,44,44,44,44,45,45,45,44,44,43,43,42,42,42]
##used to unify more weeks of data

numero_iterazioni=0

### finally main of the code
if __name__ == '__main__':
    optimal_val = 100000 ### initialize obective function to a huge value
    miglior_numero_cars = 0
    miglior_raggio = range_radius[0]
    miglior_index = 0
    converged = False
    ### here select a list of indexes (referred to range_radius list) w.r.t
    # which perform parallel optimizations
    indici_radius = [8,12,16]

    results = run_parallel_optimizations(indici_radius) ## run parallel optimizations
    # on different cores of the computer
```