Università
Ca'Foscari
Venezia

Tesi di Laurea Triennale in
Ingegneria Fisica

Sviluppo di un ambiente di simulazione per la validazione di algoritmi di ottimizzazione della ricarica dei veicoli elettrici

Laureando:
Manuel Martini

Matricola:
895594

Relatore:
Prof. Marco Salvatore Nobile

Anno Accademico:
2024-25

Dedico questo lavoro ai miei genitori, per avermi sempre permesso di esprimere il mio potenziale ed avermi sostenuto in ogni scelta.

Abstract

La diffusione crescente dei veicoli elettrici rende sempre più necessaria la progettazione di infrastrutture di ricarica efficienti e scalabili. In particolare, le stazioni DC Fast Charging (DCFC), pur offrendo tempi di ricarica ridotti, introducono importanti sfide gestionali legate alla distribuzione della potenza, alla saturazione della rete e alla variabilità della domanda.

In questo lavoro viene sviluppato un ambiente simulativo in linguaggio Python per la modellazione e la validazione di algoritmi di allocazione ottimale della potenza in un'infrastruttura EVCI modulare. Il framework integra una componente di ottimizzazione basata su programmazione lineare intera mista (MILP) e una simulazione temporale deterministica ispirata ai principi della simulazione a eventi discreti. La struttura della rete è composta da moduli di potenza (MPU), punti di ricarica (PU) e veicoli elettrici (EV), con gestione dinamica delle connessioni e delle richieste.

I risultati ottenuti mostrano la capacità del sistema di adattarsi a scenari variabili, ottimizzando la distribuzione delle risorse e riducendo lo spreco energetico. Il framework si propone come strumento utile per l'analisi e la progettazione di strategie di controllo per infrastrutture di ricarica intelligenti.

Parole chiave: ricarica rapida, veicoli elettrici, MILP, allocazione della potenza, simulazione temporale


ARIM VNIVERSITAS
PAX TIBI MARCE
EVA NGE LISTA MEUS

Indice

Introduzione

L'ottimizzazione delle infrastrutture di ricarica per veicoli elettrici (EVCI) rappresenta una sfida centrale nella transizione verso una mobilità sostenibile e decarbonizzata. L'aumento progressivo del numero di veicoli elettrici impone lo sviluppo di soluzioni intelligenti in grado di gestire in modo efficiente le risorse energetiche disponibili, ridurre i tempi di attesa e garantire stabilità alla rete elettrica.

In particolare, le stazioni di ricarica ad alta potenza in corrente continua (DC Fast Charging, DCFC) introducono complessità notevoli nella distribuzione dinamica della potenza, a causa della loro elevata domanda istantanea e della necessità di coordinare molteplici moduli e dispositivi di ricarica in tempo reale.

Il presente lavoro si propone di sviluppare un ambiente simulativo completo, implementato in linguaggio Python, che consenta di modellare il comportamento di un'infrastruttura di ricarica DCFC, stimare in modo dinamico la potenza richiesta dai veicoli connessi e validare algoritmi di allocazione ottimale delle risorse energetiche. L'obiettivo è quello di offrire uno strumento versatile e modulare, utile sia per analisi teoriche che per applicazioni pratiche nel contesto della progettazione e del controllo delle EVCI.

La simulazione proposta integra una struttura hardware virtuale costituita da moduli di potenza (MPU), punti di ricarica (PU) e veicoli elettrici (EV), e permette di testare scenari complessi in cui variano nel tempo le connessioni, le richieste di potenza e la disponibilità delle risorse. Il cuore del sistema è costituito da un algoritmo di ottimizzazione basato su programmazione lineare intera mista (MILP), che gestisce l'allocazione della potenza in condizioni vincolate, con l'obiettivo di massimizzare l'efficienza complessiva del sistema.

A completamento del modello, è stata sviluppata una componente di simulazione temporale ispirata ai principi della simulazione a eventi discreti, con lo scopo di riprodurre l'evoluzione del sistema nel tempo e analizzare le prestazioni in condizioni dinamiche e realistiche.

Il lavoro si inserisce in un contesto di ricerca attuale e rilevante, offrendo una base concreta per l'analisi e il perfezionamento di strategie di ricarica intelligente applicabili a

scenari futuri di mobilità elettrica diffusa.

Focus su DC Fast Charging e Battery Energy Storage

Le stazioni di ricarica veloce in corrente continua (DCFC, Direct Current Fast Charging) rappresentano un elemento chiave nella diffusione dei veicoli elettrici, poiché permettono di ridurre significativamente i tempi di ricarica rispetto alle tradizionali colonnine in corrente alternata. Questo tipo di stazioni di ricarica può essere collocato lungo autostrade o in contesti urbani, dove esiste il bisogno di ricaricare rapidamente il veicolo elettrico. Tuttavia, l'elevata potenza richiesta da queste infrastrutture pone sfide importanti sia dal punto di vista tecnico che gestionale: sovraccarichi di rete, distribuzione efficiente delle risorse, costi di picco elevati.

Per mitigare questi problemi, una delle soluzioni più promettenti è l'integrazione di sistemi di accumulo energetico (BESS, Battery Energy Storage System), che consentono di stabilizzare il carico sulla rete prelevando energia nei momenti di bassa richiesta e restituendola nei picchi. L'accoppiata DCFC + BESS apre a nuovi scenari di controllo intelligente, in cui le risorse vengono allocate dinamicamente sulla base della domanda istantanea e dello stato di carica dei veicoli.

In questo contesto, diventa fondamentale disporre di strumenti di simulazione e ottimizzazione che permettano di progettare, testare e confrontare strategie di allocazione e gestione della potenza, in condizioni realistiche e scalabili.

Panoramica della Letteratura

Negli ultimi anni, l'ottimizzazione delle EVCI ha ricevuto crescente attenzione da parte della comunità scientifica, in particolare nei contesti ad alta potenza come le stazioni di ricarica in corrente continua (DCFC) e nei sistemi che includono dispositivi di accumulo energetico (BESS) e fonti rinnovabili.

Sebbene la letteratura sulle reti di stazioni di ricarica sia ancora limitata, i lavori esistenti includono studi su stazioni posizionate vicino alle autostrade o in contesti urbani, con focus su code e tempi di attesa.

I contributi rilevanti per la tesi sono quelli in cui l'ottimizzazione dello scheduling delle risorse è centralizzata. Un esempio è il lavoro di Bayram et al. [1] (2013), che affronta il problema dell'allocazione efficiente delle risorse energetiche e del routing dei clienti in una rete di stazioni di ricarica rapida per veicoli elettrici (EV). Gli autori propongono

un framework basato su modelli stocastici e ottimizzazione per migliorare la qualità del servizio (QoS) e ridurre l'impatto sulla rete elettrica.

Nel presente lavoro, proponiamo un ambiente di simulazione sviluppato in Python che integra: da un lato, la modellazione accurata della struttura modulare dell'EVCI e la sua ottimizzazione tramite programmazione lineare intera mista (MILP), dall'altro, l'evoluzione temporale e dinamica dei carichi, ispirata all'approccio a eventi discreti. L'obiettivo è quello di testare e validare algoritmi per la distribuzione ottimale della potenza tra moduli e veicoli elettrici, tenendo conto di vincoli realistici come la disponibilità degli switch, l'efficienza dei convertitori e la variabilità della domanda.

Questo framework consente anche di esplorare scenari differenti variando parametri chiave come la potenza massima di rete, la configurazione delle connessioni, i coefficienti di penalizzazione energetica e le politiche di allocazione, offrendo uno strumento utile sia per lo sviluppo algoritmico che per la progettazione operativa di stazioni di ricarica.

Obiettivi del Progetto

L'obiettivo principale di questo lavoro è lo sviluppo di un ambiente di simulazione completo per l'infrastruttura di ricarica dei veicoli elettrici, con particolare attenzione alla fase di allocazione della potenza in scenari ad alta variabilità.

Il sistema proposto è in grado di:

- simulare in modo realistico una rete di ricarica con architettura modulare;
- stimare dinamicamente la potenza richiesta dai veicoli;
- allocare in modo ottimale la potenza disponibile tramite algoritmi di programmazione lineare intera mista (MILP);
- simulare il flusso di potenza nell'infrastruttura con logiche ispirate alla simulazione a eventi discreti, attraverso l'uso di un algoritmo iterativo che tiene conto delle condizioni di saturazione;
- visualizzare e analizzare i risultati delle simulazioni, monitorando indicatori di performance come efficienza, spreco di risorse e stabilità delle connessioni.

L'ambiente è stato implementato in linguaggio Python, con strutture modulari e orientate agli oggetti, in modo da favorire la scalabilità, la modifica e il riutilizzo del codice in contesti futuri.

Struttura della Tesi

Il presente elaborato è strutturato in cinque capitoli principali, oltre all'introduzione e alla bibliografia finale.

- **Capitolo 1 – Infrastruttura di Ricarica dei Veicoli Elettrici (EVCI)**: descrive in dettaglio i componenti dell'infrastruttura (MPU, PU, EV), le assunzioni modellistiche e le modalità di interconnessione elettrica tra gli elementi.
- **Capitolo 2 – Algoritmi per il funzionamento efficiente dell'infrastruttura**: presenta gli algoritmi utilizzati per la gestione delle connessioni, la stima della potenza richiesta e la strategia di allocazione della potenza tramite programmazione lineare.
- **Capitolo 3 – Simulazione Temporale**: descrive la logica con cui è stato costruito l'ambiente simulativo, distinguendo tra il flusso di potenza e il flusso informativo, e spiegando come viene modellata la dinamica temporale del sistema.
- **Capitolo 4 – Risultati delle Simulazioni e Data Visualization**: riporta i risultati ottenuti dall'esecuzione delle simulazioni, accompagnati da grafici esplicativi e commenti sulle performance del sistema.
- **Capitolo 5 – Conclusioni**: fornisce una sintesi dei risultati raggiunti, riflessioni critiche sul lavoro svolto e spunti per futuri sviluppi.

1 | Infrastruttura di Ricarica dei Veicoli Elettrici (EVCI)

1.1. Composizione dell'EVCI

In questa parte studiamo più nello specifico il modello di EVCI che si vuole sviluppare. In seguito verranno elencati gli elementi che compongono l'infrastruttura, prestando attenzione ai dettagli di maggiore interesse per il nostro modello e alle assunzioni applicate ad essi.

La potenza espressa in kilowatt (kW) erogata dalla rete viene portata ai diversi elementi dell'infrastruttura. Le connessioni tra gli elementi sono sempre mediate da degli switch (interruttori on-off).

Seguendo il "flusso" della potenza in corrente continua, gli elementi che compongono la rete elettrica EVCI sono, in ordine:

Rete di distribuzione Sistema trifase che eroga corrente alternata AC. Per semplicità, interpretiamo il comportamento complesso dell'AC come un equivalente in continua DC attraverso l'uso del concetto di valore efficace o valore RMS (Root Mean Square), quindi sostanzialmente "traduciamo" una situazione in cui la potenza fluttua nel tempo in un equivalente costante che ha lo stesso effetto medio.

Chiamiamo $P_{\text{AC MAX}}$ l'equivalente DC del massimo di potenza che la rete può fornire.

La Rete è connessa ad ogni Unità di Potenza (MPU) attraverso dei semplici interruttori on-off. Di conseguenza, la potenza di rete viene distribuita tra le diverse MPU connesse in un certo istante.

MPU (Modular Power Unit: Unità di Potenza Modulare) Raddrizzatore di segnale che riceve in entrata una potenza in corrente AC e fornisce in uscita una potenza in corrente DC. Anche in questo caso, coerentemente con le assunzioni fatte per la Rete, la potenza in entrata viene rappresentata come l'equivalente in

corrente DC della (vera) potenza in corrente AC in entrata dalla rete. Siano quindi P_i la potenza in entrata (DC), P_o la potenza in uscita, η l'efficienza dell'MPU ($\eta \in [0, 1]$) tale che

$$P_o = \eta \cdot P_i$$

Chiamiamo P_n la potenza nominale della unità di potenza, ovvero la massima potenza erogabile in uscita da essa.

In figura 1.1 si vede il tipico andamento dell'efficienza di un rettificatore in funzione della potenza erogata P_o, rispetto alla potenza nominale P_n (corrispondente a 100% output power in fig. 1.1). In generale, a regimi di potenza bassa anche l'efficienza è relativamente bassa; al crescere della potenza, l'efficienza raggiunge un massimo (vicino a 1) e si stabilizza su valori alti. La curva di efficienza, ossia la funzione $\eta(P_o)$, viene fornita dal costruttore ed è diversa per ogni raddrizzatore.

Per semplicità, nel nostro modello considereremo l'efficienza η come indipendente da P_o e P_n, ma piuttosto assumerà un valore fisso per ogni MPU.

Ogni MPU può essere connessa a una sola delle Unità di Spina Elettrica (PU) alla volta, attraverso uno switch di posizione, ovvero un interruttore in grado di collegare elettricamente una linea di uscita dell'MPU a una sola tra più linee di ingresso disponibili sulle PU. Questi switch non permettono connessioni multiple simultanee: possono attivare solo una connessione alla volta, scegliendola tra più possibili alternative.

PU (Plug Unit: Unità di Spina Elettrica) Bus di segnale di potenza DCconnesso ad ogni MPU dell'EVCI attraverso degli switch di posizione. La potenza in uscita dalla PU, quindi, è la somma delle potenze erogate dalle MPU connesse in quell'istante, cioè con l'interruttore di posizione chiuso sulla PU in questione. Definiamo P_{DEL} (dall'inglese Delivered Power) la potenza trasportata dalla PU.

In uscita dalla PU c'è un altra connessione con interruttore on-off che connette la PU all'eventuale veicolo elettrico collegato con una spina elettrica (dall'inglese Plug appunto) per la ricarica. Le batterie del veicolo, quindi, assorbono tutta la potenza P_{DEL} presente sulla plug unit connessa.

EV (Electric Vehicle: Veicolo Elettrico) Veicolo elettrico dotato di batterie per l'immagazzinamento dell'energia.

Quando un veicolo elettrico si connette per la ricarica, esso è un elemento statico. Quindi, ai fini del nostro modello, l'unica parte rilevante del veicolo sono le sue

batterie.

Chiamiamo E_n l'energia nominale espressa in kilowattora (kWh) delle batterie dell'EV, SoC lo Stato di Carica complessivo delle batterie ($SoC \in [0, 1]$), P_{REQ} la massima potenza che può assorbire in ricarica (dall'inglese Required Power).

Nell'ambito dell'infrastruttura, l'EV è il consumatore finale del servizio: ci interessa soddisfare la sua richiesta di potenza in modo rapido ed efficiente al fine di completarne la ricarica. Nel protocollo di comunicazione tra EV ed EVCI non sempre avviene la trasmissione dell'informazione sul parametro P_{REQ} dall'EV verso l'EVCI; di conseguenza ci siamo posti nel peggiore dei casi: nella nostra infrastruttura nessun EV connesso per la ricarica comunica la massima potenza assorbibile. Dovrà esistere (e quindi dovremo implementare) un algoritmo che stima la potenza richiesta dal veicolo in base ad altri dati noti: chiamiamo P_{EV} questa stima della massima potenza richiesta. Il funzionamento dell'algoritmo di stima della potenza richiesta da ogni veicolo verrà spiegato in sezione 2.1.3. D'altra parte, assumiamo che il fornitore del servizio EVCI conosca lo stato di carica SoC delle batterie del veicolo, per semplicità.

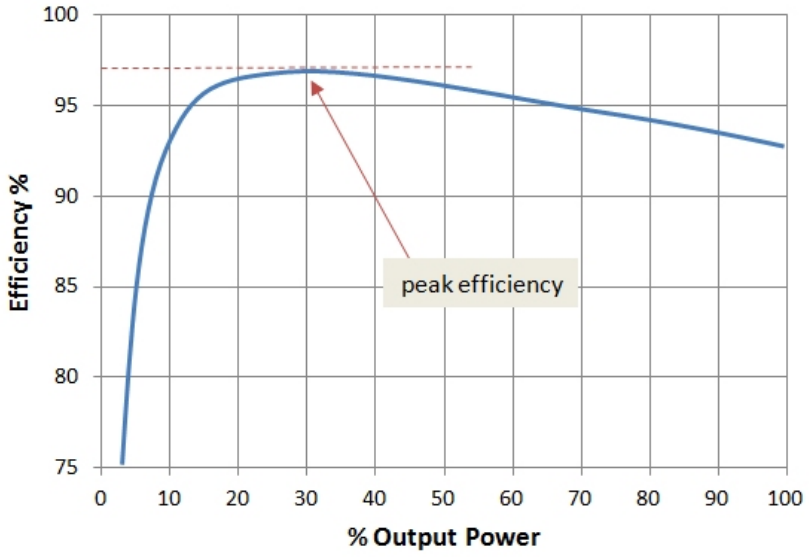


Figura 1.1: Curva di efficienza di un raddrizzatore. Fonte: `https://www.e-education.psu.edu/eme812/node/738`

Come detto, talvolta gli switch possono danneggiarsi. Sia x_{ij}^{avl} la disponibilità dello switch che collegano la MPU i alla PU j dell'infrastruttura: 1 se lo switch è in funzione, 0 se lo switch è in disfunzione (rotto, danneggiato). Si viene a formare quindi una matrice

$\boldsymbol{x^{avl}}$ della disponibilità delle connessioni. Se gli switch sono tutti funzionanti, allora la matrice è composta da tutti 1 (numero 1).

Elenco dei Paramateri dell'EVCI

Riassumiamo i parametri menzionati nelle descrizioni degli elementi dell'infrastruttura:

- $P_{\text{AC MAX}}$: massimo di potenza erogata dalla Rete (DC).
- P_n: potenza nominale di una MPU.
- η: efficienza di una MPU.
- P_{DEL}: potenza trasportata dalla PU all'EV connesso.
- E_n: energia nominale delle batterie di un EV
- SoC: Stato di Carica complessivo delle batterie di un EV
- P_{REQ}: potenza massima richiesta dal EV in ricarica
- P_{EV}: stima della potenza richiesta dal EV
- x_{ij}^{avl}: disponibilità della connessione tra MPU i e PU j

1.2. Topologia della Rete Elettrica

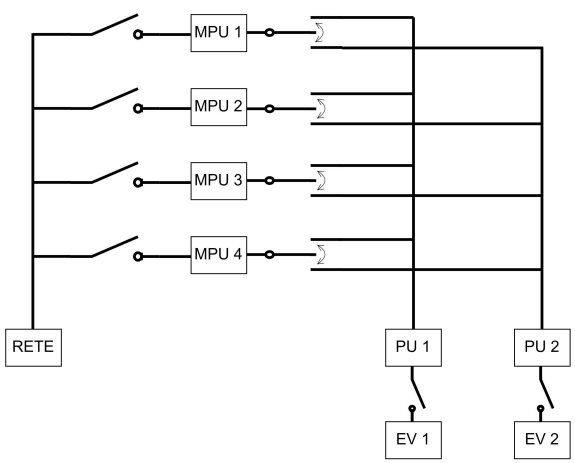
Il nostro modello di EVCI, in base a quanto detto sopra, è una rete elettrica in regime continuo DC, come quella nelle figure 1.2. La rete elettrica è composta da nodi, ossia i punti di connessione (Rete, MPU, PU, EV), e rami di connessione, ossia i cavi elettrici e gli interruttori switch che connettono i nodi. Assumiamo che i rami abbiano resistenza nulla, e quindi che non ci siano perdite resistive lungo essi.

Spesso useremo le lettere M per indicare il numero di MPU e N per indicare il numero di PU.

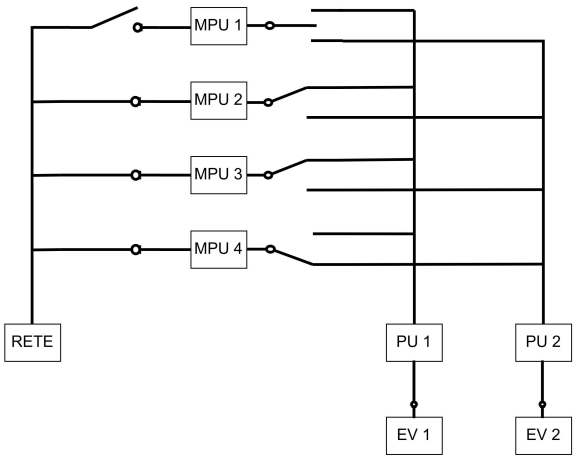
Inoltre, useremo gli indici i per indicare le posizioni delle MPU e j per le posizioni delle coppie PU-EV nell'infrastruttura. Gli indici sono tali che:

$$i \in \{1, 2, \ldots, M\} \subseteq \mathbb{N}$$

$$j \in \{1, 2, \ldots, N\} \subseteq \mathbb{N}$$



(a) Rete elettrica EVCI "spenta" (interruttori aperti), con 4 MPU, 2 PU, e 2 EV pronti alla ricarica.



(b) Rete elettrica EVCI in funzione: EV 1 si alimenta da MPU 2 e 3; EV 2 da MPU 4. MPU 1 è inattiva.

Figura 1.2: Confronto tra rete EVCI disattivata e in funzione in uno scenario con 4 MPU, 2 PU e 2 EV.

Esempio numerico

Nella figura 1.2b è stato riportato un esempio di una rete elettrica di una EVCI in funzione. Forniamo dei dati d'esempio numerici così che il lettore possa comprendere la modellizzazione matematica. Supponiamo che i dati riferiti alla figura 1.2b siano:

- Massima potenza di rete: $P_{\text{AC MAX}} = 700$ kW
- Efficienze delle MPU: $\eta_i = 1$ per $i = 1, 2, 3, 4$ (perdita dei converter delle MPU non considerata)
- Potenze nominali delle MPU: $P_{\text{n}\,1} = 40$ kW, $P_{\text{n}\,2} = 80$ kW, $P_{\text{n}\,3} = 40$ kW, $P_{\text{n}\,4} = 60$ kW
- Stima delle potenze richieste dai veicoli elettrici: $P_{\text{EV}\,1} = P_{\text{EV}\,2} = 100$ kW (dato irrilevante in questo calcolo: la stima serve solo all'ottimizzatore delle connessioni)
- Vere potenze richieste dai veicoli elettrici (massimo assorbimento): $P_{\text{REQ}\,1} = 130\,\text{kW}$, $P_{\text{REQ}\,2} = 100$ kW (uguale alla stima)

La richiesta totale di potenza "a valle" è di

$$P_{\text{REQ}\,1} + P_{\text{REQ}\,2} = 230 \text{ kW}$$

Ma le MPU connesse con interruttore chiuso (i=2,3,4) forniscono in uscita al massimo

$$P_{\mathrm{n}\,2} + P_{\mathrm{n}\,3} + P_{\mathrm{n}\,4} = 180 \text{ kW}$$

E la rete ha una disponibilità molto elevata (700 kW), quindi non crea ulteriori scompensi alla domanda degli utenti (gli EV connessi alle PU per la ricarica).

Calcoliamo quindi la potenza che giunge ad ogni singolo veicolo:

- EV_1: richiede 130 kW, e assorbe la somma delle potenze erogate da MPU_2 (che satura a 80 kW) e da MPU_3 (che satura a 40 kW). Quindi entrambe le MPU lavorano alla loro potenza nominale: la potenza assorbita (absorbed power) dall'EV sarà

  $$P_{\mathrm{ABS}\,1} = (80 + 40) \text{ kW} = 120 \text{ kW}$$

  Quindi il veicolo assorbe 120 kW (e non il massimo 130 kW) perchè la potenza è limitata dalle MPU connesse.

- EV_2: richiede 100 kW, e assorbe solo la potenza erogata da MPU_4, che satura a 60 kW. Quindi

  $$P_{\mathrm{ABS}\,2} = 60 \text{ kW}$$

  Anche in questo caso le batterie di EV_2 non assorbono il massimo di potenza richiesto a causa di una limitazione dovuta alla MPU che le alimentano.

1.3. Lato Fornitore e lato Utente

Consideriamo importante marcare un'ulteriore distinzione tra gli elementi dell'EVCI:

- Lato fornitore: MPU, PU, e la rete elettrica che le connette (cavi elettrici e switches) mantengono delle posizioni fisse nell'infrastruttura lungo il tempo. Il loro funzionamento e i loro parametri sono controllati dal fornitore del servizio EVCI. Quindi prelevano potenza dalla rete e la portano al consumatore finale, ossia l'eventuale veicolo connesso. Possono essere soggette a guasti (e quindi smettere di funzionare), ed eventualmente venire sostituite.

  Il fornitore conosce tutte le informazioni sulle componenti del lato fornitore, quindi possiede pieno controllo su di esse.

- Lato utente: al contrario, gli EV (veicoli elettrici) sono i consumatori del servizio, e quindi la loro presenza evolve nel tempo in modo non prevedibile. La User Experience può essere riassunta così:

- il conducente dell'EV giunge alla stazione di ricarica, parcheggia e spegne il veicolo, e si connette all'infrastruttura tramite la plug unit PU della postazione di ricarica scelta,
- l'EV ricarica completamente la sua batteria (nella maggior parte dei casi) oppure si disconnette dalla presa elettrica PU prima di aver raggiunto $SoC = 1$,
- infine si disconnette dalla PU e libera la stazione di ricarica per altri EV.

Come si può dedurre, i tempi di questo processo non sono prevedibili, perchè dipendono dalla frequenza con cui giungono i consumatori alle stazioni di ricarica, e dai parametri intrinsechi delle batterie di ogni singolo veicolo.

Questa distinzione mette in evidenza ancora una volta il problema della differenza di informazione. Come già detto, per esempio, il fornitore del servizio EVCI non conosce la potenza di ricarica delle batterie di ogni veicolo connesso. Questo lo vincola ad attuare degli stratagemmi per stimare questa potenza (vedi sezione 2.1.3).


ARIM VNIVERSITAS
PAX TIBI MARCE
EVA NGE LISTA MEUS

2 | Algoritmi per il funzionamento efficiente dell'infrastruttura

In questo capitolo spiegheremo quali sono gli algoritmi che permettono il funzionamento dell'infrastruttura di ricarica di veicoli elettrici nel suo insieme. Presenteremo una versione di tali algoritmi che potenzialmente potrebbe venire utilizzata in una vera EVCI fisica: l'intero senso di questo progetto, infatti, è quello di testare questi algoritmi in un ambiente di simulazione, per poi poterli impiegare anche nel mondo reale. Attraverso la costruzione di tale ambiente di simulazione è possibile testare diverse versioni di questi algoritmi, con diversi parametri o con variazioni logico-procedurali.

2.1. Iterazione Software

Spesso faremo riferimento a una singola esecuzione di tali algoritmi all'interno dell'infrastruttura simulata come **iterazione software**, per distinguerli dalla simulazione di una "iterazione hardware", che invece è la simulazione fisica degli elementi (rete, unità di potenza, switches, EV e il flusso di potenza tra di loro). Una simulazione hardware avviene all'interno della finestra temporale di un'iterazione software.

La parte di simulazione temporale è stata sviluppata ad hoc, ispirandosi ai principi della simulazione a eventi discreti (DES). Tuttavia, rispetto alla DES classica, in cui il tempo avanza secondo la coda di eventi asincroni (es. arrivi e partenze di veicoli), il nostro modello evolve secondo iterazioni temporali fisse, scandite da un **clock software**, ciascuna delle quali simula un possibile stato istantaneo dell'infrastruttura. Quindi la nostra simulazione è completamente **deterministica**.

Ad ogni intervallo scandito da un clock si ricalcola la configurazione delle connessioni tra MPU e PU. Nella simulazione, per implementare tale **ciclo software**, si è usato il costrutto `for` lungo gli istanti temporali.

Ogni "iterazione software" avviene su un time-step dt deterministico e regolare (simulazione sincrona). Al contrario della DES (eventi asincroni), nel nostro caso, si conoscono

istante per istante tutti i parametri principali dell'infrastruttura (una parte di quelli riportati nell'elenco in sezione 1.1, e si vuole riuscire a simulare il flusso di potenza a partire da tali informazioni.

Finestra temporale di Esecuzione di una Iterazione

Il fornitore del servizio deve scegliere quanto intervallo di tempo debba passare tra un'esecuzione di un'iterazione software e l'altra. Questa **finestra temporale** può essere fissa o variabile. La scelta corretta di questo parametro è importante e dipende dall'affluenza dei veicoli clienti al servizio. Un'idea è quella di diminuire l'intervallo di tempo nelle ore di una giornata di maggiore affluenza per fornire un'allocazione delle risorse più efficiente, e aumentarlo durante le ore di minore affluenza per risparmiare il consumo di potenza di calcolo associato all'esecuzione di tali algoritmi, che possono essere energeticamente dispendiosi, soprattutto se l'infrastruttura agisce su grande scala.

Nella versione attuale del progetto, per semplicità, la finestra temporale dt è fissa e abbastanza piccola da fare in modo che la disconnessione di un veicolo EV da una plug unit PU sia esplicita dal punto di vista del fornitore, cioè tale che l'interruttore switch tra i due elementi rimanga aperto almeno per un'iterazione, prima dell'arrivo di un altro veicolo.

2.1.1. Array delle Connessioni

Per formalizzare matematicamente gli algoritmi e, in particolare, il problema di ottimizzazione, traduciamo le connessioni elettriche della rete in matrici o vettori "booleani", cioè contenenti solo 0 (numero zero) o 1 (numero uno). Con zero (0), si intende che l'interruttore è aperto e non trasmette tensione (e quindi non trasmette potenza) in quel ramo di connessione, con uno (1) si intende che l'interruttore è chiuso in quel ramo di collegamento e quindi trasmette tensione.

Chiamiamo $\boldsymbol{x}$ la **Matrice delle Connessioni** (la matrice rossa in figura 2.1), l'elemento x_{ij} codifica la connessione assente (0) o presente (1) tra la MPU i e la PU j. Come già detto, grazie all'implementazione fisica modellata da interruttori di posizione, ogni MPU può essere totalmente scollegata, e quindi avere tutti 0 (zeri) lungo la riga i corrispondente della matrice $\boldsymbol{x}$, oppure essere collegata esclusivamente a una PU, e quindi avere un unico 1 sulla riga i, alla colonna j corrispondente alla PU collegata.

Per massimizzare l'efficienza della ricarica dei veicoli, bisogna risolvere un problema di ottimizzazione della matrice delle connessioni $\mathbf{x}$. Ottimizzare la matrice delle connessioni

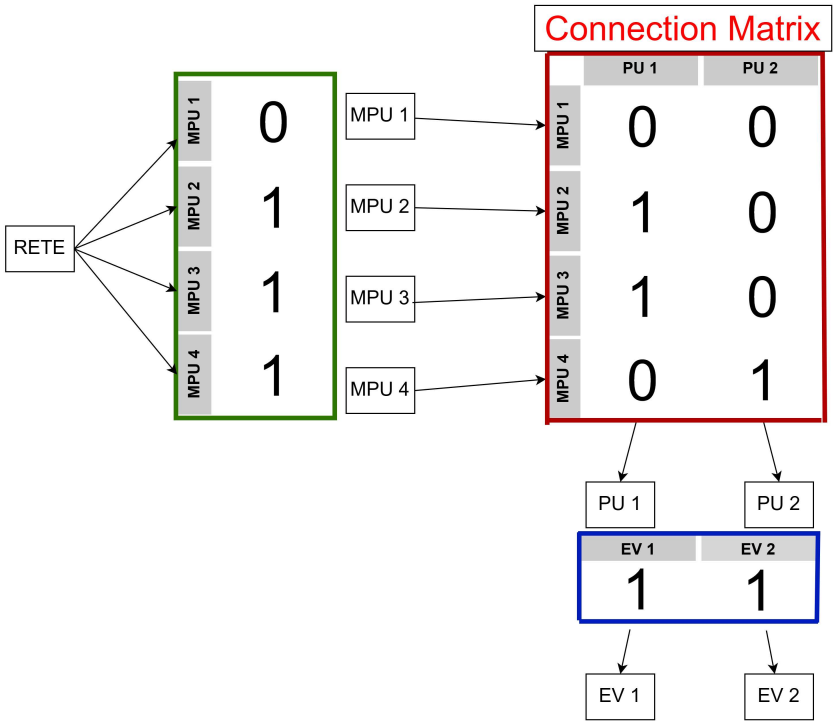


Figura 2.1: Arrays di connessione tra gli elementi nel contesto della rete elettrica della EVCI nella configurazione di figura 1.2b. Vettore verde: interruttori on-off tra rete e MPU; Matrice rossa (Connection Matrix): interruttori di posizione tra MPU e PU; vettore blu: interruttori on-off tra PU ed EV.

significa trovare le migliori connessioni in base agli elementi presenti nella EVCI in quel momento, e in particolare in base ai loro parametri (potenza di rete, potenze nominali, potenze richieste dai veicoli etc. etc.).

Inoltre, un altro vettore a cui prestare attenzione è $\boldsymbol{C_{ev}}$, ovvero il vettore delle connessioni tra le unità di spina elettrica (PU) e i veicoli elettrici (EV) (il vettore blu in figura 2.1). La componente C_{ev}^{j} vale 0 se non c'è alcun veicolo connesso alla PU j, 1 altrimenti. Se un veicolo completa la ricarica e si disconnette da tale PU, la componente corrispondente assume sempre valore 0 prima che un altro veicolo si colleghi, poiché abbiamo assunto che la finestra temporale di esecuzione sia abbastanza piccola da essere minore del tempo che due veicoli impiegano per scambiarsi nella posizione di sosta e collegare la spina elettrica per la ricarica.

2.1.2. Sequenza di Esecuzione degli Algoritmi

Gli algoritmi che vengono eseguiti ad ogni finestra temporale di esecuzione sono 3, e sfruttano i dati a disposizione del fornitore del servizio EVCI. I primi 2 algoritmi servono

solo a preparare i "dati giusti", cioè i vettori corretti, da dare in input all'ottimizzazione delle connessioni (l'ultimo algoritmo). Ad ogni iterazione software, gli algoritmi che vengono eseguiti sono, in ordine:

1) Disconnessione virtuale degli EV carichi completamente Sezione 2.1.2

2) Stima della potenza di ricarica richiesta da ciascun EV Sezione 2.1.3

3) Ottimizzazione della distribuzione della potenza Sezione 2.2.1

Ad ogni iterazione, gli algoritmi hanno in memoria tutti i dati dell'iterazione precedente e quelli dell'iterazione corrente.

Disconnessione Virtuale degli EV Carichi Completamente

Abbiamo assunto, per semplicità, che il fornitore del servizio conosca lo stato di carica SoC delle batterie di ogni EV connesso. Può capitare abbastanza frequentemente che un veicolo EV_j raggiunga un livello completo di ricarica $SoC = 1$, ma che rimanga connesso alla PU_j mantenendo $C_{ev}^j = 1$, poichè il conducente possessore dell'EV deve ancora tornare al parcheggio per liberare la postazione. In questo caso, tramite un algoritmo che itera lungo tutte le coppie PU-EV, si forza $C_{ev}^j = 0$, deallocando le risorse (MPU) connesse a tale EV. Vedi il codice 2.1 per l'implementazione in Python.

Listing 2.1: Disconnessione virtuale degli EV che hanno già raggiunto ricarica completa

```
def disconnect_full_evs(...):
    """
    Parameters
    cev_np : current PU-EV connections
    cev_prev_np : previous PU-EV connections
    ev_is_full : boolean array, True if EV is full

    Returns
    Updated connections array
    """
    cev_fake_np = np.copy(cev_np)
    for j in range(len(cev_np)):
        if cev_prev_np[j] == cev_np[j]:   # EV still connected
            if ev_is_full[j]:             # EV is full
                cev_fake_np[j] = 0        # --> disconnect
    return cev_fake_np
```

2.1.3. Stima della Potenza di Ricarica Richiesta dalle Batterie di ciascun EV

L'algoritmo di stima della potenza di ricarica richiesta dalle batterie di ciascun EV è basato sull'aumentare con un incremento fisso *dp*, iterazione dopo iterazione, la potenza disponibile su una plug unit PU, fino al raggiungimento della potenza richiesta dal veicolo o finché una componente dell'infrastruttura non raggiunga la massima potenza disponibile in erogazione (condizione di saturazione).

La scelta del valore dell'incremento *dp* è a discrezione del fornitore del servizio e deve essere testata. In questa versione del progetto abbiamo scelto di porre questo incremento uguale alla potenza nominale della MPU meno potente. In questo modo, in uno scenario in cui c'è abbondante disponibilità di potenza dalla rete e un grande numero di unità di potenza, ad ogni EV collegato per la ricarica verrà assegnata almeno una MPU.

Il codice Python 2.2 è abbastanza autoesplicativo, ma riassumiamo i passaggi principali:

- Se non c'è alcun veicolo connesso ($C_{ev}^{j} = 0$) alla PU_j, manteniamo la base di partenza per la crescita della stima della potenza richiesta dal veicolo (inesistente in questo caso) uguale all'incremento. Ci permettiamo di dare un valore non nullo alla potenza richiesta dalle postazioni di ricarica non occupate perché l'algoritmo di ottimizzazione (spiegato in sezione 2.2.1) è costruito in modo tale da non collegare alcuna MPU se non ci sono veicoli connessi alla PU_j.
- Se invece un veicolo EV_j è connesso per la ricarica ($C_{ev}^{j} = 1$), calcoliamo innanzitutto la massima potenza disponibile sul plug all'iterazione precedente P_{AVL}^{j} come

$$P_{\text{AVL}}^{j} = \sum_{i=1}^{M} x_{ij} \cdot P_{n}^{i}$$

  Sia P_{ABS}^{j} la misura della potenza assorbita dal veicolo all'iterazione precedente.

  Se all'iterazione precedente la potenza assorbita dalle batterie del veicolo era minore di quella massima disponibile all'iterazione corrente ($P_{\text{ABS}}^{j} < P_{\text{AVL}}^{j}$), allora significa che la stima coincide con il vero valore della potenza di richiesta, quindi siamo riusciti a svolgere a pieno il nostro intento ($P_{\text{EV}}^{j} = P_{\text{REQ}}^{j}$). Memorizziamo questa scoperta in una variabile booleana (chiamata `pev_found`).

  Se invece il veicolo è connesso e $P_{\text{ABS}}^{j} = P_{\text{AVL}}^{j}$, probabilmente vuol dire che qualche componente sta limitando la sua crescita oppure che abbiamo sottostimato la po-

tenza richiesta all'iterazione precedente. In questo caso, verifichiamo quale di queste condizioni è soddisfatta:

- Se la somma delle potenze assorbite all'iterazione precedente è maggiore o uguale alla potenza di rete all'iterazione corrente ($\sum_{i=1}^{N} P_{ABS}^{j} \geq P_{AC\ MAX}$), allora la ricarica verrà limitata da un'insufficienza di potenza di rete, e di conseguenza la potenza assorbita dai veicoli non potrà aumentare in nessun caso. Quindi mantengo il valore della stima ottenuto precedentemente, fino a quando non ci sarà una maggiore disponibilità di rete.
- Se invece tutte le MPU disponibili stanno erogando potenza, allora non ha senso aumentare la stima, visto che in ogni caso non potrei fornire maggiore potenza agli EV connessi. Quindi, mantengo il valore della stima ottenuto precedentemente.
- Se invece precedentemente ho stimato con accuratezza la vera potenza richiesta dal veicolo, e ho memorizzato questa scoperta nella apposita variabile booleana `pev_found`, allora mantengo fisso il valore della stima.
- Se nessuna delle 3 condizioni precedenti si avvera, allora incremento il valore della stima, perché vuol dire che sto sottostimando la richiesta di potenza di ricarica del veicolo.

Listing 2.2: Stima della potenza di ricarica richiesta dalle batterie di ciascun EV

```
def estimate_pev_np(...) -> np.array:
    """
    Parameters
    ----------
    pacmax : maximum power availability from the grid
    pabs_np : absorbed powers by the EVs at the previous iteration
    pev_np_prev : request estimations at the previous iteration
    pev_found : booleans telling if the request estimation matches
        the real request
    x_np : connection matrix at the previous iteration
    x_avl_np : connection availabilities at the current iteration
    cev_np : array of connections PU-EV at the current iteration
    pn_np : array of nominal powers at the current iteration

    Returns
    -------
    power request estimations for each EV
```

```
    """

    # Increment of power for the estimation algorithm
    dp = min(pn_np)
    pev_np = np.zeros(shape=len(pev_np_prev))

    for j in range(len(pev_np)):
        pabs = pabs_np[j]
        pev_prev = pev_np_prev[j]
        cev = cev_np[j]

        if cev == 0:
            pev = dp
            pev_found[j] = 0

        else:  # cev == 1
            pavl = sum(x_np[:, j]*pn_np)
            pabs = pabs_np[j]

            if pabs < pavl:
                pev = pabs
                pev_found[j] = 1  # True

            else:  # pabs == pavl
                if sum(pabs_np) >= pacmax:
                    pev = pev_prev
                elif not at_least_one_mpu_not_used(x_np, x_avl_np)
                    :
                    pev = pev_prev
                elif pev_found[j]:
                    pev = pev_prev
                else:
                    pev = pev_prev + dp

        pev_np[j] = pev

    return pev_np
```

2.2. Strategia di Allocazione delle Risorse

La strategia di allocazione ha lo scopo di distribuire efficientemente la potenza della rete $P_{AC\ MAX}$ ai veicoli connessi all'EVCI durante un'iterazione software. Le risorse da allocare, in questo caso, sono le potenze erogate dalle MPU dell'EVCI, in base alla domanda di potenza degli EV connessi.

Come messo in evidenza nel capitolo precedente, la domanda di potenza "a valle" (lato utente) è variabile nel tempo. Per avere una distribuzione efficiente della potenza in entrata dalla rete è necessario ottimizzare le connessioni mediate dagli interruttori della rete elettrica, ad ogni iterazione software.

2.2.1. Programmazione Lineare Applicata alla Strategia di Allocazione

Il problema di ottimizzazione è stato risolto con un algoritmo di Programmazione Lineare Intera Mista (MILP). Questa tecnica matematica si utilizza per risolvere problemi di ottimizzazione, in cui si desidera massimizzare o minimizzare una funzione obiettivo lineare, soggetta a un insieme di vincoli anch'essi lineari. Viene impiegata ogni volta che è necessario allocare risorse limitate in modo ottimale, come nel nostro caso. Un tipico problema di programmazione lineare prevede:

- una funzione obiettivo da minimizzare,
- un insieme di variabili decisionali,
- una serie di vincoli che rappresentano le limitazioni del sistema.

La soluzione ottimale si trova nell'intersezione dei vincoli (regione ammissibile), dove la funzione obiettivo assume il valore massimo o minimo.

La programmazione lineare intera mista (MILP) si differenzia dalla programmazione lineare pura perché combina variabili intere (che possono assumere solo valori interi) e variabili continue (che possono assumere qualsiasi valore reale) all'interno di un modello di programmazione lineare. In pratica, si ha un problema di programmazione lineare pura in cui alcune variabili devono essere intere, mentre altre possono essere continue.

2.2.2. Modellazione Matematica della Strategia di Allocazione

Presentiamo il modello matematico del problema di ottimizzazione. Manteniamo uno stile e una sintassi simile alla modellazione matematica che è stata implementata nel

framework Pyomo (spiegato in sezione 2.2.3), utilizzato in questo progetto per risolvere il problema di MILP in Python.

Insiemi Sono dei contenitori astratti che contengono le istanze del nostro problema.

- $\mathcal{I}$: insieme delle unità di potenza (MPU).
- $\mathcal{J}$: insieme delle unità di ricarica (Coppie PU / EV).

Parametri Alcuni di questi parametri sono già stati definiti nella sezione 1.1. In pratica, sono gli input dell'algoritmo di ottimizzazione.

- P_n^i: potenza nominale dell'unità di potenza (MPU) $i \in \mathcal{I}$.
- C_{ev}^j: presenza di veicolo elettrico sulla plug unit (PU) $j \in \mathcal{J}$ (1 se presente, 0 altrimenti), con modifica dovuta alla disconnessione virtuale degli EV collegati con batteria completamente carica (sezione 2.1.2).
- P_{EV}^j: stima della potenza richiesta dal veicolo elettrico sulla plug unit j, ottenuta grazie all'algoritmo spiegato nella sezione 2.1.3 sopra.
- $P_{\text{AC MAX}}$: potenza totale massima disponibile dalla rete.
- x_{ij}^{avl}: disponibilità della connessione tra unità di potenza i e plug unit j (1 se disponibile, 0 se rotta)
- x_{ij}^{prev}: connessione stabilita tra i e j nell'iterazione precedente.
- K_{EFF}: coefficiente di penalizzazione per potenza non utilizzata.
- K_{VAR}: coefficiente di penalizzazione per variazioni di connessione.

Variabili decisionali Sono le variabili che vengono ottimizzate per massimizzare la funzione obiettivo.

- $x_{ij} \in \{0, 1\}$: variabile binaria che vale 1 se l'unità i è connessa alla plug unit j. La matrice costituita dagli elementi x_{ij} è l'output utile dell'algoritmo.
- $p_j \in \mathbb{R}_+$: potenza "fittizia" erogata alla plug unit j. Vedi il paragrafo 2.2.2 per capire perchè è "fittizia".
- $\delta_{ij} \in 0, 1$: differenza assoluta tra lo stato attuale e quello precedente della connessione (i, j), ossia $\delta_{ij} = |x_{ij} - x_{ij}^{\text{prev}}|$.

  Sappiamo che il valore assoluto è un'operatore non lineare, quindi useremo uno stratagemma matematico nei vincoli per imporre tale definizione senza l'uso di tale

operatore, e mantenendo la linearità. In breve, trasformiamo la definizione canonica di valore assoluto in due disequazioni.

Variabili ausiliarie Sono i termini della somma che compone la funzione obiettivo, pesati in base ai coefficienti di penalizzazione.

- P_{TOT}: potenza totale erogata.
- $P_{\text{NOT USED}}$: potenza inutilizzata.
- n_{VAR}: numero di variazioni delle connessioni rispetto alla configurazione precedente.

Funzione obiettivo La funzione obiettivo da massimizzare è

$$\max \left(P_{\text{TOT}} - K_{\text{EFF}} \cdot P_{\text{NOT USED}} - K_{\text{VAR}} \cdot n_{\text{VAR}} \right)$$

dove

$$P_{\text{TOT}} = \sum_{j \in \mathcal{J}} p_j$$

$$P_{\text{NOT USED}} = \sum_{j \in \mathcal{J}} \left(\sum_{i \in \mathcal{I}} P_n^i x_{ij} - p_j \right)$$

$$n_{\text{VAR}} = \sum_{i \in \mathcal{I}} \sum_{j \in \mathcal{J}} \delta_{ij}$$

La funzione obiettivo è stata costruita in modo tale da:

- Massimizzare la potenza totale erogata P_{TOT}, ovvero la somma delle potenze assorbite (e nuovamente erogate) dalle plug unit PU. In questo modo si velocizza la ricarica degli EV connessi.
- Minimizzare la potenza inutilizzata $P_{\text{NOT USED}}$. Lo spreco di potenza su ciascun ramo PU_j è la differenza tra la somma delle potenze nominali delle MPU connesse a quel ramo e la potenza p_j effettivamente presente su quel ramo. La somma di questi sprechi lungo j è il totale della potenza inutilizzata.

  Questa aggiunta nella funzione obiettivo totale serve a fare in modo che le MPU lavorino ad alta efficienza, visto che dalla figura 1.1 si nota che per avere efficienze

elevate, le MPU devono erogare valori di potenza prossimi alla potenza nominale P_n.

Il fornitore del servizio può scegliere quanto peso attribuire a questo obiettivo tramite un coefficiente di penalizzazione per potenza non utilizzata K_{EFF}.

- Minimizzare il numero di variazioni delle connessioni rispetto alla configurazione precedente n_{VAR}. Questa aggiunta serve a mantenere una stabilità nella EVCI: evitando cambi troppo repentini delle connessioni è possibile abbassare la frequenza dei guasti agli switch di posizione.

  Il fornitore del servizio può scegliere quanto peso attribuire a questo obiettivo tramite un coefficiente di penalizzazione per variazioni di connessione K_{VAR}.

Nel paragrafo 2.2.2 verrà approfondita la questione della scelta del valore dei coefficienti di penalizzazione K_{EFF} e K_{VAR}.

Vincoli

$\sum_{j\in\mathcal{J}} x_{ij} \leq 1$	$\forall i \in \mathcal{I}$	(ogni MPU è connessa al massimo ad una PU)
$\sum_{i\in\mathcal{I}} x_{ij} \geq C_{\text{ev}}^{j}$	$\forall j \in \mathcal{J}$	(plug attiva solo se c'è un EV connesso ad essa)
$\sum_{i\in\mathcal{I}} P_n^i x_{ij} \geq p_j$	$\forall j \in \mathcal{J}$	(massima potenza disponibile sul plug)
$p_j \leq P_{\text{EV}}^{j} \cdot C_{\text{ev}}^{j}$	$\forall j \in \mathcal{J}$	(limite di potenza stimata richiesta dal veicolo)
$\sum_{j\in\mathcal{J}} p_j \leq P_{\text{AC MAX}}$		(vincolo sulla potenza massima totale limitata dalla rete)
$x_{ij} \leq x_{ij}^{\text{avl}}$	$\forall i \in \mathcal{I},\ j \in \mathcal{J}$	(connessioni disponibili)
$\delta_{ij} \geq x_{ij} - x_{ij}^{\text{prev}}$	$\forall i, j$	
$\delta_{ij} \geq x_{ij}^{\text{prev}} - x_{ij}$	$\forall i, j$	(trasformazione del valore assoluto in due disequazioni per mantenere linearità)

Utilità dell'Ottimizzazione

Dentro questo insieme di equazioni e disequazioni algebriche lineari è riassunto il problema che vogliamo risolvere. Per trovare i valori ottimali delle variabili decisionali si possono usare degli strumenti di calcolo automatici, come spiegheremo in sezione 2.2.3. Per questa classe di problemi, infatti, esistono algoritmi di risoluzione esatti, che per-

mettono di trovare la miglior soluzione ammissibile, qualora questa esista, che risolve il problema studiato. Un esempio celebre è l'algoritmo del simplesso. Ci limitiamo a usare gli strumenti di risoluzione, senza entrare in dettaglio sulla teoria che ne descrive il funzionamento.

L'unico output davvero importante nel flusso di esecuzione sono i valori x_{ij} che compongono la Connection Matrix $\mathbf{x}$. Verosimilmente, ad ogni iterazione temporale (clock software) il fornitore del servizio EVCI verifica se i parametri dell'infrastruttura sono cambiati rispetto all'iterazione precedente.

- Se i parametri non sono cambiati: l'EVCI mantiene le connessioni uguali a quelle dell'iterazione precedente.
- Se i parametri sono cambiati: viene risolto questo problema di ottimizzazione, per assicurare una distribuzione efficiente delle risorse ai consumatori del servizio (i veicoli EV connessi per la ricarica) istante per istante. In pratica:
  - Tramite l'esecuzione dell'ottimizzazione si ottiene la nuova matrice $\boldsymbol{x}$ delle connessioni.
  - L'EVCI riconfigura in modo automatico le connessioni degli switch tra le MPU e le PU in base ai valori x_{ij} di tale matrice.

Precisazioni sull'Ottimizzazione

Significato di p_j Ricordiamo ancora una volta che in input all'ottimizzatore delle connessioni è stata data P^j_{EV}, che è solo una stima della vera potenza richiesta P^j_{REQ} dall'EV, tale che

$$P^j_{\text{EV}} \leq P^j_{\text{REQ}} \quad \forall j \in \mathcal{J}$$

Abbiamo definito P^j_{DEL} (dall'inglese delivered power) come la vera potenza trasportata sulla plug unit j, e quindi assorbita dal veicolo connesso ad essa.

Il fatto di usare una potenza richiesta dagli EV "fittizia" nella modellizzazione matematica del problema rende anche la variabile p_j una potenza trasportata alla plug unit j "fittizia". Le variabili p_j e P^j_{DEL} coincidono solo nel caso in cui la stima è corretta, cioè quando $P^j_{\text{EV}} = P^j_{\text{REQ}}$.

Questo paragrafo è servito solo ad allarmare il lettore sul fatto che la variabile p_j è solo una variabile interna dell'ottimizzatore, senza alcun significato fisico.

Design dell'ottimizzazione e altre proposte La scelta della programmazione lineare intera mista come metodo di risoluzione del problema di ottimizzazione è dovuta principalmente alla sua semplicità, chiarezza matematica e soprattutto alla convergenza quasi sempre assicurata a una soluzione ottimale, secondo la definizione di "ottimo" data dalla funzione obiettivo. Inoltre, un altro vantaggio della MILP è che permette di sapere quando la regione ammissibile è vuota e quindi non esiste una soluzione al problema.

Come si nota dalla definizione della funzione obiettivo, in realtà stiamo riducendo un problema multi-obiettivo, con 1 funzione da massimizzare (P_{TOT}) e altre 2 funzioni da minimizzare ($P_{\text{NOT USED}}$ e n_{VAR}), in un problema con un'unica funzione obiettivo attraverso una combinazione lineare degli addendi.

Alcuni algoritmi alternativi per l'ottimizzazione multi-obiettivo possono essere ad esempio NSGA-II (basato su algoritmi genetici), MOPSO (versione multi-obiettivo di Particle Swarm Optimization), o più in generale l'analisi di Pareto, combinata con il metodo dei vincoli (ϵ-constraints). Il metodo utilizzato deve assicurare convergenza ad un'unica soluzione in tempi brevi, per poter aggiornare le connessioni in maniera automatica ad ogni clock software.

Coefficienti di penalizzazione K_{EFF} e K_{VAR} Abbiamo visto che lo scopo dell'ottimizzazione è trovare la configurazione che massimizza la funzione obiettivo

$$\max\left(P_{\text{TOT}} - K_{\text{EFF}} \cdot P_{\text{NOT USED}} - K_{\text{VAR}} \cdot n_{\text{VAR}}\right)$$

Nasce spontaneo domandarsi quale sia la migliore scelta per i coefficienti di penalizzazione K_{EFF} e K_{VAR}. Uno dei motivi di questo intero progetto è proprio quello di creare un ambiente di simulazione dove si possa testare l'algoritmo di ottimizzazione con diversi valori di questi parametri, per capire quale sia la scelta migliore.

Innanzitutto notiamo che n_{VAR} (il numero di variazioni delle connessioni rispetto all'iterazione precedente) è adimensionale, mentre P_{TOT} e $P_{\text{NOT USED}}$ sono potenze espresse in kiloWatt (kW). Una rapida analisi dimensionale ci fa capire che quindi i due coefficienti di penalizzazione devono avere unità di misura diverse.

Probabilmente non esiste una scelta migliore delle altre. Il fornitore del servizio EVCI, ad esempio, può dare a questi coefficienti dei valori basati su delle speculazioni economiche. Seguendo questa proposta, il parametro K_{EFF} può essere ottenuto a partire dalla potenza sprecata a causa dell'inefficienza delle MPU a regimi di potenza bassa, associando a questo spreco di corrente un costo in denaro (reperibile, per esempio, a partire dalla bolletta

dell'energia elettrica). D'altra parte, il parametro K_{VAR} può essere calcolato in base ai costi di riparazione o sostituzione degli switch nella rete elettrica EVCI, visto che ad alti valori di n_{VAR} aumenta anche la probabilità che gli switch vengano danneggiati.

2.2.3. Strumenti Computazionali per la Programmazione Lineare

Il Framework Pyomo

La modellazione del problema di ottimizzazione è stata realizzata mediante il framework **Pyomo** (Python Optimization Modeling Objects), selezionato in base ai seguenti requisiti del progetto:

- **Integrazione nativa con Python**: L'ambiente di sviluppo principale del progetto richiedeva compatibilità con l'ecosistema Python, in particolare con le librerie per l'analisi dati (NumPy, Pandas) utilizzate per la gestione dei parametri di simulazione.
- **Flessibilità di modellazione**: La struttura a oggetti di Pyomo permette di rappresentare fedelmente la formulazione matematica del problema (Sezione 2.2.1)
- **Portabilità**: La possibilità di esportare il modello in formati standard (LP) ha facilitato l'implementazione, simile al linguaggio umano nella sintassi.

Selezione del Solver GLPK

Per la risoluzione del problema di programmazione lineare intera mista (MILP) è stato adottato il solver *GLPK* (GNU Linear Programming Kit) in virtù della sua accessibilità: essendo software libero (licenza GPL), GLPK elimina barriere di costo per l'uso accademico, a differenza di alternative commerciali come CPLEX o Gurobi.

L'algoritmo del simplesso implementato in GLPK ha garantito convergenza in configurazioni limitate dell'EVCI, ma in scenari più critici o di grande scala non ha sempre raggiunto una soluzione ottimale in tempi brevi. Per questo motivo è stato necessario impostare un **tempo limite** per la ricerca di una soluzione ottimale. Se questo tempo viene superato, vengono mantenute le connessioni dell'infrastruttura all'iterazione precedente. In questa versione del progetto, abbiamo impostato come valore di tempo limite la durata di un'iterazione software, come sarebbe in un caso reale.

Per questo motivo, sebbene GLPK abbia soddisfatto i requisiti principali del progetto, per estensioni future si valuteranno anche CBC (COIN-OR): alternativa open-source con

migliore supporto per problemi MILP di larga scala, e IPOPT.

Una statistica utile si otterrebbe eseguendo l'ottimizzazione con diversi solver, e confrontando i tempi di esecuzione per ciascuno di essi (facendo la media dei tempi rispetto a diverse configurazioni di EVCI per ogni solver).


ARIM VNIVERSITAS
PAX TIBI MARCE
EVA NGE LISTA MEUS

3 | Simulazione Temporale

La simulazione temporale è una tecnica computazionale che permette di riprodurre il comportamento di un sistema dinamico nel tempo, discretizzando l'evoluzione del sistema in intervalli (time-step) e calcolando lo stato del sistema a ogni passo. Questo approccio è essenziale per analizzare sistemi complessi in cui le interazioni tra componenti e l'evoluzione temporale sono critiche per la valutazione delle prestazioni.

La simulazione temporale è strumentale per la valutazione delle prestazioni dell'EVCI per diversi motivi:

Modellazione di Comportamenti Non Lineari Molti sistemi reali (come un'EVCI) hanno dinamiche non lineari, dipendenti da eventi discreti (es. arrivo/partenza di veicoli) e variabili continue (es. potenza erogata). La simulazione temporale permette di integrare queste dinamiche in un unico framework.

Validazione degli algoritmi Consente di testare algoritmi in condizioni realistiche, verificandone l'efficacia al variare del carico o dei guasti. Nel nostro caso, ad esempio, si vuole verificare se l'algoritmo di stima di P_{EV} assieme alla MILP applicata all'allocazione delle MPU massimizza davvero l'efficienza nella distribuzione della potenza agli EV.

Analisi di Scenari Complessi Simulando un periodo esteso (es. 24 ore), si possono valutare:

- Affidabilità: come reagisce il sistema a picchi di domanda o guasti.
- Efficienza Energetica: se l'allocazione delle risorse minimizza gli sprechi (es. potenza non utilizzata).
- User Experience: tempi medi di ricarica e soddisfazione degli utenti.

Riduzione di Costi e Rischi Evita sperimentazioni fisiche costose o pericolose (es. sovraccarichi nella rete).

Flessibilità nella Progettazione Parametri come la durata del time-step o la finestra temporale tra iterazioni software possono essere modificati per studiare trade-off tra precisione e velocità di calcolo.

3.1. Simulazione Hardware

Nella sezione 2.1 abbiamo spiegato che durante la prima fase di un'iterazione software vengono eseguiti una serie di algoritmi che portano ad avere come output finale una nuova matrice delle connessioni x_{ij}. A quel punto, la configurazione della EVCI cambia automaticamente: gli switch presenti sui rami di connessione si aprono o chiudono per riprodurre quanto codificato nella matrice delle connessioni.

Durante la seconda fase dell'iterazione software la potenza viene trasportata dalla rete di distribuzione agli elementi dell'infrastruttura in base a tali connessioni. Nella realtà ciò avviene (quasi) istantaneamente. Lo scopo di questo capitolo è spiegare come è stato simulato tale comportamento istantaneo nel nostro progetto. Chiamiamo **simulazione hardware** tale simulazione dello scambio di potenza tra gli elementi fisici dell'EVCI.

Le esigenze specifiche del nostro progetto ci hanno portato a implementare un design della simulazione peculiare. Per spiegare didatticamente come abbiamo implementato la simulazione hardware, marchiamo la distinzione tra:

- **Flusso di Potenza**: è il flusso reale della potenza tra gli elementi. Ogni elemento riceve in entrata la potenza disponibile dall'elemento precedente e fornisce una potenza in uscita che entrerà nell'elemento successivo. Questo flusso di potenza parte dalla Rete (disponibilità totale iniziale) e arriva agli EV (richiesta "a valle").
- **Flusso di Informazioni**: avviene in ordine inverso rispetto al flusso di potenza. Questa è la maniera con cui abbiamo costruito la simulazione hardware.

  Ogni elemento riceve in input l'informazione sulla potenza disponibile dall'elemento precedente e l'informazione sulla potenza richiesta dall'elemento successivo, e il suo output è il minimo tra richiesta e disponibilità: la potenza effettiva in entrata dall'elemento precedente. Questo flusso parte dagli EV e arriva alla Rete.

3.2. Flusso di Potenza

Nella sezione 1.1 veniva mostrato il modello di EVCI, elencando i diversi componenti. Nelle immagini 1.2 abbiamo mostrato come viene implementata la rete elettrica a livello pratico. Nella realtà, corrente $i(t)$, tensione $V(t)$, e potenza $P(t) = i(t) \cdot V(t)$ trasportate

nei rami tra la rete e le MPU sono in regime AC, ma abbiamo detto che il comportamento AC è stato tradotto in comportamento DC anche in quei rami, tramite il concetto di RMS. D'altra parte, negli altri rami della rete, ossia nelle connessioni tra le MPU e gli EV (mediate dalle PU), il regime della potenza è DC anche in un caso reale, poiché le unità di potenza raddrizzano la corrente proveniente dalla rete. Quindi, nel nostro modello, la potenza è sempre in regime continuo DC.

La nostra simulazione si occupa di simulare la potenza che "fluisce" a partire dalla rete, attraverso le MPU collegate, fino ad arrivare agli EV connessi. Ovviamente, questo ordine di passaggio della potenza attraverso gli elementi dell'infrastruttura è solo una costruzione mentale: nella realtà, all'accensione dell'EVCI, la potenza viene assorbita o erogata istantaneamente da tutti gli elementi dell'infrastruttura. Tuttavia, per la simulazione di questo trasporto istantaneo di potenza su tutti gli elementi, risulta utile immaginare un flusso di potenza di questo tipo.

Riduciamoci allo studio del flusso di potenza che giunge a una singola coppia PU-EV j. Il lettore dovrebbe avere capito a questo punto che la potenza che giunge a tale coppia è la somma delle potenze erogate dalle MPU i collegate. Ripetiamo quali sono le potenze in entrata e in uscita per ogni elemento connesso su questa linea dell'infrastruttura. Per tutti gli elementi, tranne per le unità di potenza MPU, la potenza in entrata e in uscita è la stessa perchè assumiamo che non ci siano perdite interne. Al contrario, nella conversione da AC a DC che avviene nella MPU è associata una perdita descritta dall'efficienza η dell'MPU. Il flusso di potenza attraverso gli elementi è stato riportato nella figura 3.2a, ed è il seguente:

- **Rete**: in entrata e in uscita riceve ed eroga la potenza di rete che chiameremo P_{GRID} proveniente da un Sistema di Gestione dell'Energia (EMS = Energy Management System) esterno di cui l'EVCI fa parte. Questa potenza di rete è superiormente limitata dal parametro $P_{\text{AC MAX}}$.
- **MPU con connessione attiva**: La potenza di rete si distribuisce in maniera bilanciata tra tutte le MPU connesse. Nella sezione 3.4.4 spiegheremo in dettaglio come viene simulata la distribuzione bilanciata della potenza di rete P_{GRID} tra le MPU connesse. Per il momento, ci basti sapere che ogni MPU:
  - riceve in entrata una frazione di P_{GRID}
  - eroga in uscita la potenza entrante moltiplicata per l'efficienza η.

  La potenza in uscita è limitata superiormente dalla potenza nominale P_n dell'MPU.

- **PU**: riceve in entrata ed eroga in uscita la potenza P_{DEL} (delivered power), che è la somma delle potenze uscenti dalle MPU collegate.
- **EV**: assorbe in entrata una potenza $P_{\mathrm{ABS}} = P_{\mathrm{DEL}}$, uguale a quella trasportata dalla plug. Questa potenza viene misurata dal fornitore del servizio EVCI tramite uno strumento di misura. La potenza assorbita dall'EV è limitata superiormente dalla massima potenza richiesta P_{REQ}.

Condizioni di Saturazione

Nella descrizione del flusso di potenza, sono stati messi in evidenza i limiti superiori della potenza degli elementi Rete, MPU, EV. Quando uno di questi elementi lavora al suo massimo di potenza, diremo che l'elemento lavora in **condizioni di saturazione**. In questo regime, quindi, l'elemento ha raggiunto il massimo in modo continuativo.

Nel linguaggio tecnico, questo termine è molto usato per i raddrizzatori (MPU), ma ad esempio è meno usato per la massima potenza di ricarica degli EV. Ad ogni modo, noi useremo questo termine per tutti gli elementi che hanno un massimo di potenza da assorbire/erogare: la rete è in saturazione quando eroga all'EVCI una potenza equivalente al massimo $P_{\mathrm{AC\ MAX}}$; una MPU è in saturazione se eroga una potenza equivalente a quella nominale P_n; un EV è in saturazione se assorbe una potenza uguale alla massima potenza di ricarica P_{REQ}.

Notare che, a qualunque istante di tempo, almeno un intero insieme di elementi dello stesso tipo è tutto in saturazione. Ad esempio: in uno scenario in cui la massima potenza di rete e la potenza di ricarica di ogni veicolo sono molto alte, l'unico parametro che limita il flusso di potenza attraverso i rami della rete elettrica EVCI sono le potenze nominali delle MPU connesse. In questo scenario, ogni MPU lavora alla sua potenza nominale (in saturazione).

3.3. Simulazioni Ripetute con Potenza Crescente fino a Saturazione

Dati i parametri dell'infrastruttura, per simulare direttamente il flusso di potenza, bisognerebbe sapere a priori quali elementi della rete elettrica lavorano in condizioni di saturazione con tali parametri. Con tale informazione, sarebbe possibile calcolare "a cascata" (sia in avanti che all'indietro rispetto alla direzione del flusso di potenza) le potenze erogate e assorbite dagli altri elementi dell'EVCI. Sebbene nel nostro modello sa-

rebbe possibile svolgere un calcolo di questo genere, per trovare quali elementi si trovano in condizioni di saturazione, abbiamo preferito approcciare il problema in una maniera differente. Il metodo del calcolo diretto sarebbe troppo vincolato alla configurazione specifica che abbiamo scelto di implementare: se in futuro il progetto venisse ampliato e la rete elettrica si complicasse, quel calcolo diverrebbe errato.

La fase di simulazione hardware è realizzata tramite un ciclo iterativo che incrementa progressivamente la potenza richiesta dagli EV, fino al raggiungimento della saturazione di almeno un sottosistema di elementi (rete, MPU o EV). Questo meccanismo può essere interpretato come una forma di **convergenza incrementale**, che simula lo scambio di potenza in **condizioni quasi stazionarie**.

Quindi, è stato implementato un ciclo in cui si aumenta linearmente step-by-step di un incremento piccolo (in questa versione del progetto $dp = 0.1$ kW) la potenza disponibile in entrata P^j_{DEL} ai veicoli EV j che rispettano tutte queste condizioni:

- sono connessi per la ricarica ($C^j_{ev} = 1$)
- non hanno già raggiunto la condizione di saturazione ($P^j_{\text{DEL}} < P^j_{\text{REQ}}$)
- non hanno già raggiunto la carica completa delle batterie ($SoC_j < 1$)

Quindi, ad ogni incremento infinitesimo di potenza, aumenta la richiesta totale di potenza "a valle" del flusso di potenza, e si esegue il flusso di informazioni che viene spiegato nella prossima sezione (3.4). Queste "simulazioni a potenza minore" vengono eseguite fino a quando la potenza che fluisce nella rete elettrica è abbastanza elevata da portare un intero gruppo di elementi dello stesso tipo in **condizione di saturazione** oppure fino a quando tutti gli EV hanno completato la ricarica, cioè quando si verifica almeno una delle seguenti condizioni:

- tutti gli EV hanno raggiunto saturazione, cioè assorbono la potenza di ricarica massima P_{REQ}.
- tutti gli EV hanno raggiunto una carica completa ($SoC = 1$).
- tutte le PU hanno raggiunto saturazione: non abbiamo ancora mai parlato di saturazione per le plug unit, vedi sezione 3.4.
- tutte le MPU hanno raggiunto saturazione: lavorano tutte alla potenza nominale P_n.
- la rete ha raggiunto saturazione: eroga una potenza equivalente al suo massimo $P_{\text{AC MAX}}$.

In questo modo si ottiene lo **stato quasi stazionario** della rete elettrica al tempo t (tempo dato dal loop "software" esterno). Si parla di stato quasi stazionario quando il sistema cambia lentamente nel tempo, ma ogni stato istantaneo può essere approssimato come stazionario, cioè come se fosse in equilibrio fisico o elettrico istante per istante.

Nell'ambiente di programmazione Python, tale **ciclo hardware** di crescita della potenza è stato simulato con un ciclo `for` (annidato dentro l'altro ciclo `for` del software). La potenza fornita ad ogni EV parte da 0 e potenzialmente può aumentare fino a valori molto alti, ma il ciclo si interrompe tramite il costrutto `break` non appena un intero gruppo di elementi dello stesso tipo raggiunge la saturazione. In base alle nostre esigenze, si sarebbe potuto usare il costrutto while, ma è stato evitato per evitare di incorrere in loop infiniti (caso poco probabile ma non impossibile).

Usando questo metodo, vengono sfruttati al massimo i vantaggi dell'implementazione modulare che verrà spiegata nella prossima sezione 3.4. Infatti, se si vogliono aggiungere altri oggetti nella simulazione della rete elettrica, è sufficiente integrarli nel flusso di informazioni in base alla politica utilizzata dal simulatore.

3.4. Flusso di Informazioni: Confronto tra Richiesta e Disponibilità

Il flusso informativo usato nella simulazione è l'inverso del flusso di potenza, quindi va dalla domanda degli EV fino alla rete. Questa modellazione garantisce una stima coerente della potenza realmente erogabile, in assenza di un modello fisico continuo. Tale logica, sebbene non formalizzata nella letteratura standard DES, richiama i principi della **retroazione (feedback)** nei sistemi dinamici, e permette una stima realistica delle condizioni operative dell'infrastruttura.

3.4.1. Programmazione ad Oggetti

Per realizzare la simulazione del flusso di potenza nella rete EVCI, è stato adottato un approccio di **programmazione a oggetti** (OOP) in Python. Questo paradigma consente di strutturare un **codice modulare**, modellando ogni elemento dell'infrastruttura come un oggetto simulato con proprietà e comportamenti ben definiti.

In Python, un oggetto è un'istanza di una classe, che può essere vista come un "modello" generale di un certo tipo di elemento (ad esempio: rete, MPU, PU, EV). Una classe definisce:

- attributi: le informazioni che ogni oggetto contiene (es. potenza massima, efficienza, stato di saturazione, etc.)
- metodi: le funzioni che ogni oggetto è in grado di eseguire (es. aggiornare la potenza disponibile, rispondere a una richiesta di potenza, etc.)

Quando si crea un oggetto, si utilizza il costruttore della classe (una funzione chiamata `_init_`) per inizializzare i suoi attributi.

Per rendere il codice ancora più modulare ed estensibile, abbiamo fatto uso anche del concetto di **ereditarietà**. In Python, è possibile creare una nuova classe che eredita attributi e metodi da una classe già esistente, modificandone o estendendone il comportamento. Questo è particolarmente utile nel nostro progetto: ad esempio, tutte le classi che rappresentano elementi dell'infrastruttura (rete, MPU, PU, EV) derivano da una classe base comune, che definisce il comportamento standard di un oggetto simulato (gestione della potenza, verifica della saturazione, ecc.). In questo modo, si evitano duplicazioni di codice e si mantiene una struttura coerente e facilmente espandibile.

Questo approccio permette di gestire facilmente scenari complessi e ampliamenti futuri, poiché ogni nuovo elemento della rete può essere integrato semplicemente definendo una nuova classe (o sottoclasse), coerente con le altre già esistenti.

3.4.2. Oggetto Simulato

La classe Python **oggetto simulato** è la classe "madre" di tutti gli oggetti dell'infrastruttura che vengono simulati. Un oggetto simulato nella sua forma più semplice è stato riportato in figura 3.1, che esemplifica l' I/O (input/output) di questo. In breve, un oggetto simulato:

- riceve in **input**:
  - la richiesta di potenza in uscita (nel flusso di potenza) $P_{\text{REQ OUT}}$ dall'elemento successivo
  - la potenza disponibile in entrata (nel flusso di potenza) P_{AVL}. Questa tipicamente è la potenza di saturazione dell'oggetto, cioè un parametro interno.
- Mentre ha come **output** l'effettiva potenza in entrata (nel flusso di potenza) P_{IN}. In generale l'effettiva potenza che l'elemento può trasferire è il minimo tra i due dati in input, infatti:
  - se $P_{\text{AVL}} < P_{\text{REQ OUT}}$, allora sono limitato dalla disponibilità di potenza proveniente dall'elemento precedente, quindi $P_{\text{IN}} = P_{\text{AVL}}$.

– se $P_{\text{AVL}} \geq P_{\text{REQ OUT}}$, allora sono limitato dalla potenza di saturazione, cioè mi trovo in regime di saturazione ($P_{\text{IN}} = P_{\text{REQ OUT}}$). Se mi trovo in questa condizione, memorizzo tale stato in un attributo dell'oggetto. Per recuperare tale informazione, ogni oggetto possiede un metodo `reached_saturation` che alla chiamata restituisce `True` se l'oggetto è in condizione di saturazione, `False` altrimenti.

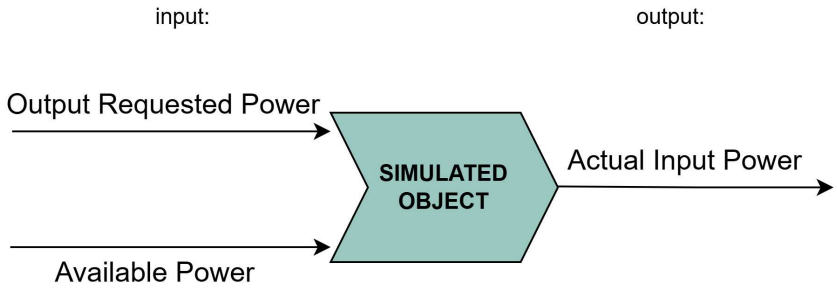


Figura 3.1: Schema Input/Output di un oggetto simulato nel flusso di informazioni

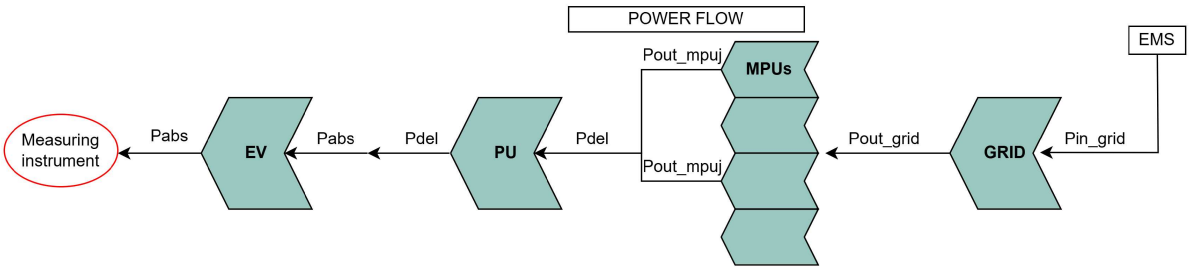
Per avere un design della simulazione ordinato nella sua esecuzione, facilmente leggibile e soprattutto scalabile, la classe "oggetto simulato" possiede 3 **metodi** che verranno ereditati da tutti gli oggetti dell'EVCI. Questi metodi vengono chiamati su un singolo oggetto in maniera sequenziale per aggiornare lo stato interno di questo. Nell'ordine di chiamata, i metodi sono:

- `setInput`: riceve nuovi parametri in input e modifica gli attributi dell'oggetto in base a questi.
- `step`: aggiorna gli attributi in base alle logiche interne dell'oggetto o altri calcoli.
- `getOutput`: recupera l'output utile per l'oggetto successivo nel flusso di informazioni.

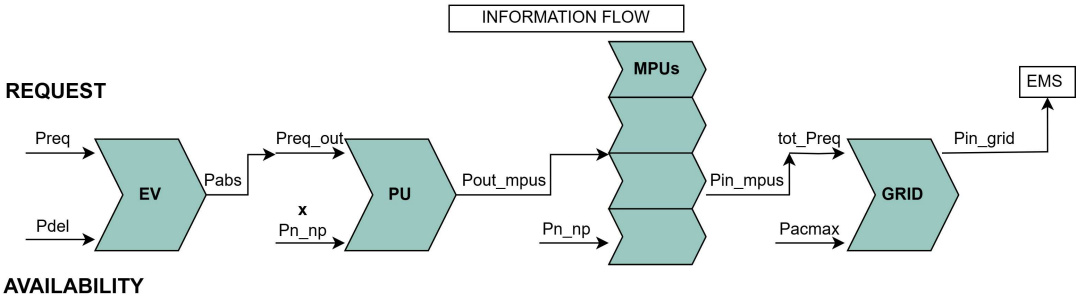
3.4.3. Input/Output degli Oggetti e Implementazione Python

Nella figura 3.2b è stato schematizzato il flusso di informazioni nell'infrastruttura simulata, confrontata simmetricamente con la figura 3.2a superiore, che mostra il flusso di potenza.

In questa sezione entriamo nel dettaglio dell'I/O di ogni singolo oggetto e di cosa avviene all'interno dei metodi `step`. Per alleggerire la trattazione sull'implementazione delle classi, nelle parti di codice mostriamo solo i metodi più interessanti e meno scontati, e nelle spiegazioni ci concentriamo sugli aspetti più rilevanti. In particolare, spesso ometteremo



(a) Schema del flusso di potenza, da destra verso sinistra come indicato dalle frecce.



(b) Schema del flusso di informazioni, da sinistra verso destra come indicato dalle frecce. Ogni oggetto simulato della catena, rispetta l' I/O della figura 3.1. La riga di frecce superiore degli input di ogni oggetto indica la richiesta di potenza in uscita; la riga di frecce inferiore degli input indica la disponibilità di potenza dell'oggetto, o i parametri utili per il suo calcolo. L'output di ogni oggetto è l'effettiva potenza in entrata.

Figura 3.2: Confronto simmetrico dei flussi di potenza e di informazioni.

i metodi costruttore `_init_` (inizializzazione dei parametri) e setter `setInput` (modifica dei parametri in base a nuovi input).

Per una comprensione maggiore, il lettore deve sapere che la **costruzione di un oggetto** è associata a una **posizione** nell'EVCI. Questo vuol dire che i costruttori (definiti col costrutto `_init_` nelle classi) vengono chiamati una sola volta all'esterno sia del ciclo delle iterazioni software (lungo il tempo), sia del ciclo delle iterazioni hardware (incrementi di potenza). Ad ogni nuova iterazione hardware, contenuta in una iterazione temporale (software), la simulazione integra i nuovi dati della simulazione e modifica gli attributi di ciascun oggetto già esistente tramite i metodi `setInput`.

Nel ciclo di esecuzione, si eseguono i metodi di tutti gli EV, poi di tutte le PU, di tutte le MPU, e infine della rete.

Oggetto Electric Vehicle

Descriviamo come i metodi ereditati dalla classe madre (oggetto simulato) vengono ridefiniti per l'oggetto EV e come è fatto il flusso di informazioni di questo oggetto, come esemplificato nella figura 3.2b.

Input Gli input dell'oggetto EV, trasformati in attributi, sono

- P_{REQ}: richiesta di potenza di ricarica massima.
- P_{DEL}: potenza disponibile in entrata (proveniente dalla PU collegata), ossia la variabile che sta aumentando nel ciclo di incrementi infinitesimi dp fino a saturazione, come spiegato nelle sezioni precedenti.

Output

- P_{ABS}: effettiva potenza assorbita dall'EV, il minimo tra richiesta e disponibilità:

$$P_{\text{ABS}} = \min(P_{\text{REQ}}, P_{\text{DEL}})$$

.

Saturazione Se $P_{\text{ABS}} = P_{\text{REQ}}$ allora l'EV è in condizione di saturazione: memorizzo tale stato nell'attributo `saturated`, che verrà recuperato nel momento del bisogno tramite il metodo getter `reached_saturation`. Questo attributo con apposito getter per il recupero è presente in tutti gli oggetti.

Metodi per la Simulazione della Ricarica delle Batterie Ora, invece, descriviamo i metodi posseduti esclusivamente dalla classe EV. Questi metodi aggiuntivi sono dedicati alla gestione dell'informazione sulla carica delle batterie del veicolo.

Siano: E_n l'energia nominale delle batterie dell'EV e SoC lo stato di carica delle batterie. I metodi esclusivi per l'oggetto EV sono:

`setSoC`: aumenta lo stato di carica SoC in base alla carica che il veicolo ha ottenuto durante un'iterazione software, cioè durante una finestra temporale dt. Si assume che il tempo di esecuzione degli algoritmi per il funzionamento efficiente dell'infrastruttura sia trascurabile rispetto alla durata della finestra temporale: di conseguenza il veicolo assorbe una potenza P_{ABS} (espresso in kiloWatt) per un tempo dt (espresso in secondi). Quindi la variazione di energia (espressa in kiloWattora, kWh) delle

batterie in questo intervallo di tempo vale

$$\Delta E = P_{\text{ABS}} \cdot \frac{dt}{(3600 \text{ s/h})}$$

E quindi lo stato di carica si aggiorna così:

$$SoC = \frac{E_i + \Delta E}{E_n}$$

dove E_i è l'energia iniziale delle batterie.

`getSoC`: recupera l'informazione sull'attributo SoC.

`battery_full`: ritorna un valore booleano: `True` se la batteria è piena, `False` altrimenti.

Listing 3.1: Metodi dell'oggetto EV

```
class ElectricVehicle(SimulatedObject):

    def __init__(self,
                 preq: float = 0,
                 en: float = 0,
                 isoc: float = 0): ...
    def setInput(...): ...
    def step(self):
        # New EV connected  cev = 0 --> 1 ... reset SoC
        if self.cev_prev == 0 and self.cev == 1:
            self.soc = self.isoc  # initial SoC

        pabs = min(self.pdel, self.preq)
        self.pabs = pabs
        self.saturated = self.preq <= self.pdel
        self.cev_prev = self.cev

    def reached_saturation(self): return self.saturated
    def getOutput(self): return self.pabs

    def setSoc(self,
               dt: float = 1) -> None:

        e_prev = self.soc*self.en
        e_now = e_prev + self.pabs*(dt/3600)  # time in hours
```

```
        self.soc = min(e_now/self.en, 1)

    def getSoc(self): return self.soc
    def battery_full(self): return self.soc >= 1
```

Oggetto Plug Unit

Input

- $P_{\text{REQ OUT}}$: effettiva potenza assorbita dall'EV a cui è connessa la PU.
- dati necessari per il calcolo della potenza disponibile P_{AVL}:
  - $\boldsymbol{x}$: matrice delle connessioni.
  - $\boldsymbol{P_n}$: vettore con le potenze nominali di tutte le MPU dell'infrastruttura.

  Il calcolo della massima potenza disponibile P_{AVL} si effettua sommando le potenze nominali delle MPU connesse:

$$P^j_{\text{AVL}} = \sum_{i=1}^{M} x_{ij} \cdot P^i_n$$

Output Come già detto, ogni oggetto simulato nel flusso di informazioni deve fornire in output qual è la potenza prelevata dall'elemento precedente nel flusso di potenza. In questo caso, una PU deve fornire in output quale potenza preleva da ciascuna MPU connessa ad essa. Chiamiamo $P_{\text{OUT MPU}}$ la potenza uscente da una MPU e diretta alla PU. Il calcolo di questa potenza è incapsulato in una funzione dall'implementazione complicata (`mpu_p_iter_calc`), spiegata in sezione 3.4.4.

Tramite tale funzione si ottiene l'output dell'oggetto PU, che sarà un vettore contenente la potenza inviata da ciascuna MPU i a tale PU. Ovviamente, se una MPU non è collegata alla PU in questione, allora la potenza inviata è nulla.

Svolgendo questo calcolo per ogni PU, e "incollando" in senso orizzontale i vettori si crea una matrice $\boldsymbol{P}_{\text{OUT MPUs}}$ contenente la potenza inviata dalla MPU i alla PU j. Ogni MPU può inviare potenza a una sola PU, quindi ogni riga della matrice ha un unico valore non nullo.

Saturazione Se $P_{\text{OUT MPU}} = P_{\text{AVL}}$ allora la PU è in condizione di saturazione: la potenza in entrata è limitata dalla massima potenza disponibile sulla spina elettrica. Questa nuova condizione di saturazione è fisicamente inesistente, ma serve, nell'ambito

del flusso di potenza, a simulare un "collo di bottiglia" che non è simulato dagli altri oggetti.

Con questa aggiunta, il ciclo di crescita della potenza (simulazione hardware) si blocca (tramite il comando `break`) anche quando tutte le PU stanno trasportando la massima potenza erogabile dalle MPU connesse. Questo significa che se le PU sono tutte in saturazione, anche le MPU sono tutte in saturazione: quindi un lettore attento nota che in questa configurazione questa condizione di saturazione è inutile, perché coincide con la condizione di saturazione di tutte le MPU. Si è scelto di aggiungerla comunque per possibili aggiunte future al progetto, nelle quali verrà integrata anche la potenza proveniente da altre fonti, ad esempio rinnovabili.

Listing 3.2: Metodi dell'oggetto PU

```
class PlugUnit(SimulatedObject):
    def __init__(...): ...
    def setInput(...): ...

    def step(self) -> None:
        j = self.j
        xj = self.x[:, j]  # desired column
        pn = self.pn
        self.pavl = sum(xj*pn)
        self.pout_mpu = mpu_p_iter_calc(pn, xj, self.preq_out)
        self.saturated = self.preq_out >= self.pavl

    def getOutput(self): return self.pout_mpu
    def reached_saturation(self): return self.saturated
```

Oggetto Modular Power Unit

Input

- $P_{\text{REQ OUT}}$: richiesta di potenza da parte della PU connessa (l'unico elemento non nullo lungo la riga i della matrice $\boldsymbol{P}_{\text{OUT MPUs}}$.
- P_n: disponibilità massima data dalla potenza nominale.

Output

- $P_{\text{IN MPU}}$: effettiva potenza in entrata, il minimo tra richiesta e disponibilità, diviso per l'efficienza η dell'MPU:

$$P_{\text{IN}} = \frac{min(P_{\text{REQ OUT}}, P_n)}{\eta}$$

Saturazione Se $P_{\text{REQ OUT}} = P_n$, allora l'MPU è in saturazione. Ricordiamo che è la potenza uscente da una MPU ad essere superiormente limitata dalla potenza nominale: quella entrante può essere maggiore della nominale, se dopo la riduzione dovuta all'efficienza si ottiene una potenza uscente minore o uguale alla nominale.

Listing 3.3: Metodi dell'oggetto MPU

```
class PowerUnit(SimulatedObject):
    def __init__(...): ...
    def setInput(...): ...
    def step(self):
        self.pin = min(self.preq_out, self.pn) / self.eff
        self.saturated = self.preq_out >= self.pn
    def getOutput(self): return self.pin
    def reached_saturation(self): return self.saturated
```

Oggetto Rete

Input

- $P_{\text{TOT REQ}}$: richiesta di potenza totale, calcolata come la somma delle potenze:

$$P_{\text{TOT REQ}} = \sum_{i=1}^{M} P^i_{\text{IN MPU}}$$

- $P_{\text{AC MAX}}$: disponibilità massima data dall'equivalente in DC della potenza di rete massima.

Output

- $P_{\text{IN GRID}} = min(P_{\text{TOT REQ}}, P_{\text{AC MAX}})$

  Questo output "finale" nel flusso di informazioni è inutile per questo specifico progetto, ma l'idea è che tale informazione sulla potenza effettivamente trasportata dalla Rete possa servire al Sistema di Gestione dell'Energia (EMS = Energy Management System) di cui l'EVCI fa parte.

Saturazione Se $P_{\text{IN GRID}} = P_{\text{AC MAX}}$ allora l'oggetto Rete è in saturazione.

Listing 3.4: Metodi dell'oggetto Rete

```
class Grid(SimulatedObject):
    def __init__(...): ...
    def setInput(...): ...
    def step(self):
        self.pin = min(self.tot_preq, self.pacmax)
        self.saturated = self.tot_preq >= self.pacmax
    def getOutput(self): return self.pin
    def reached_saturation(self): return self.saturated
```

3.4.4. Allocazione di Potenza erogata dalle MPU

L'algoritmo `mpu_p_iter_calc` calcola come una potenza totale richiesta in entrata P_{OUT} da una PU j, si distribuisce fisicamente in erogazione tra le MPU connesse a tale PU.

Vincoli principali:

- Ogni MPU può erogare al massimo la propria potenza nominale P_n^i.
- Solo le MPU attive (indicate da $x_{ij} = 1$) ricevono potenza.
- La distribuzione di potenza erogata cerca di essere equa tra tutte le MPU attive.

Supponiamo di avere n MPU collegate, ciascuna con una potenza massima P_n^i. Vogliamo distribuire una potenza totale P_{OUT} fra di esse, ma rispettando il vincolo che nessuna MPU eroghi più del proprio limite.

1. Inizializzazione Si contano le MPU attive (cioè con $x_{ij} = 1$) e si assegna inizialmente a ciascuna la stessa quota di potenza:

$$p_i = \frac{P_{\text{OUT}}}{n}$$

2. Redistribuzione iterativa Il problema si presenta quando alcune MPU non possono erogare tutta questa quota, perché il loro limite P_n^i è inferiore a p_i. L'algoritmo rileva questo caso e:

- assegna loro il massimo consentito P_n^i
- calcola quanta potenza è avanzata (`dp`) e la redistribuisce tra le MPU rimanenti

Il processo è iterativo: ogni volta che si "bloccano" una o più MPU, la potenza eccedente viene ricalcolata e si riparte da capo con le restanti.

3. Perché il `break`? Il ciclo interno ha un `break` per forzare l'algoritmo a rivedere la distribuzione ogni volta che cambia il numero di MPU rimaste. Così si garantisce che la nuova distribuzione tenga conto della potenza liberata.

Convergenza garantita Ogni iterazione riduce il numero di MPU attive e redistribuisce la potenza in modo controllato, quindi l'algoritmo converge sempre dopo un numero finito di passi.

Questo algoritmo è una forma di **allocazione iterativa con soglie**, usata spesso quando ci sono limiti locali (come vincoli fisici o tecnici) e una risorsa da distribuire equamente. L'approccio usato è semplice ma efficace, e mantiene la somma delle potenze assegnate uguale a quella richiesta.

Listing 3.5: Algoritmo usato per l'allocazione della potenza erogata dalle MPU

```
def mpu_p_iter_calc(pn: np.array,
                    xj: np.array,
                    preq_out: float = 0) -> np.array:
    """
    Calculates the power allocation for a given array
    of nominal power values and a total power request.

    Parameters
    ----------
    pn : array of nominal powers of each power unit
    xj : array of the active connections with pn array
    preq_out : requested output power

    Returns
    -------
    array of output powers for each power unit
    """

    n = sum(xj)  # number of connected mpus
    p = np.zeros(shape=len(pn))

    if n == 0:  # if no mpu connected to puj
        return p
```

```
pi = preq_out/n  # initial power guess, equally divided for
    each mpu
dp = 0

for k in range(len(pn)):

    for i in range(len(pn)):

        if xj[i] != 0:  # mpu is connected

            # power absorbed by mpu[i] is lower than its
                nominal power
            # ...continue the iteration
            if p[i] < pn[i]:

                # initial power is lower than the mpu[i]
                    nominal power
                # ...assign more power
                if pi < pn[i]:

                    p[i] = pi + dp/n

                    # check if mpu nominal power is reached
                    # ...if so, mpu[i] absorbed power is found
                        ,
                    # calculate exceeding power and restart
                        iteration
                    if p[i] > pn[i]:
                        dp += pi - pn[i]
                        p[i] = pn[i]
                        n -= 1
                        break

                # initial power is higher than the mpu[i]
                    nominal power
                # ...mpu[i] absorbed power is found,
                # calculate exceeding power and restart
                    iteration
```

```
                else:
                    dp += pi - pn[i]
                    p[i] = pn[i]
                    n -= 1
                    break
    return p
```

3.5. Flusso di Esecuzione della Simulazione

Il processo di simulazione completo combina **cicli annidati** nello schema concettuale mostrato nel flusso di esecuzione 3.6. Ogni passaggio della simulazione è stato spiegato nelle sezioni precedenti.

I due cicli principali annidati sono:

Ciclo `for` Software : opera su scala temporale uguale alla durata di ogni iterazione software, per tutti gli istanti di tempo ottenuti dalla tabella dell'input file.

Ciclo `for` Hardware : opera su una scala di potenza crescente, con incremento piccolo. Procede fino a quando un intero gruppo di oggetti della stessa classe giunge a saturazione.

Listing 3.6: Flusso di Esecuzione della Simulazione

1. Creazione degli oggetti e dell'ottimizzatore
2. Inizializzazione delle variabili
3. Per ogni istante temporale t (ciclo software):
3.1 Ottieni i dati dall'input file tabellare
3.2 Esegui gli algoritmi per il funzionamento efficiente:
– *Disconnetti virtualmente gli EV con batteria carica*
– *Stima la potenza di ricarica richiesta dai veicoli*
– *Se i parametri sono cambiati rispetto all'iterazione precedente:*
Esegui l'ottimizzatore e applica nuova configurazione
3.3 Per ogni incremento di potenza (ciclo hardware):
Per ogni coppia EV−PU:
Se (EV connesso) and (EV non saturo) and
(PU non satura) and (carica non completa):
Aumenta potenza disponibile all'EV
Per ogni EV:
– *Esegui simulazione dell'oggetto (setInput, step, getOutput)*
– *Compila vettore potenze assorbite (Input/Output)*
Per ogni PU:
– *Esegui simulazione dell'oggetto*
– *Compila matrice potenze uscenti dalle MPU*
Per ogni MPU:
– *Esegui simulazione dell'oggetto*
– *Compila vettore potenze entranti*
Per la Rete:
– *Esegui simulazione dell'oggetto*
– *Calcola potenza totale utilizzata*
Criteri di arresto:
Se (tutti EV saturati) or (tutte batterie piene) or
(tutte PU saturate) or (tutte MPU saturate) or
(Rete saturata):
Termina ciclo hardware
3.4 Aggiorna lo Stato di Carica (SoC)
3.5 Aggiorna le variabili di sistema
4. Output: Evoluzione temporale delle potenze

3.6. Integrazione dei Dati ed Esecuzione

Integrazione dei dati I dati per la simulazione sono ottenuti a partire da un foglio di calcolo (ad esempio un file csv o xlsx), convertibile in un dataframe bidimensionale dalla libreria Pandas. Ogni riga della tabella deve contenere i dati dell'infrastruttura per un'iterazione temporale software, ordinati secondo un ordine fissato per convenzione. Avendo posto un vincolo sull'ordine, non ci sarà alcun vincolo sui nomi dei parametri riportati nell'header.

L'ordine convenzionale dei parametri (con la nomenclatura usata nel progetto) è mostrato nell'header della tabella 3.1, che riporta un esempio di dati di un'EVCI 2x1 (2 MPU, 1 coppia PU-EV). La matrice $\boldsymbol{x_{avl}}$ è riportata in ogni riga in maniera appiattita.

Tabella 3.1: Parametri di simulazione di un'EVCI 2x1 per 3 iterazioni temporali

t	Pacmax	Pn1	Pn2	eff1	eff2	Preq1	En1	iSoC1	Cev1	x_avl11	x_avl21
0	700	40	80	0.95	0.93	100	60	0.3	1	1	1
1	650	40	80	0.95	0.93	90	60	0.4	1	1	1
2	720	40	80	0.95	0.93	110	60	0.5	0	1	1

La funzione `get_arrays_from_row` si occupa di trasformare i dati di ogni riga in parametri (in forma numerica, vettoriale o matriciale) gestibili nelle fasi successive della simulazione. La chiamata a questa funzione avviene al punto 3.1 del flusso di esecuzione della simulazione (procedimento simulativo 3.6).

- Input della funzione:
  - riga del dataframe corrispondente all'istante t.
  - numero di MPU nell'infrastruttura.
  - numero di PU nell'infrastruttura.
- Output della funzione: lista degli oggetti Python associati ai dati della tabella.

Esecuzione della Simulazione La funzione `simulate` esegue esattamente il procedimento 3.6. Vediamo input e output di questa funzione.

- Input della funzione:
  - input_file_path: percorso per l'ottenimento dell'input file.
  - numero di MPU nell'infrastruttura.

  - numero di PU nell'infrastruttura.
  - dt: durata dello step temporale di una iterazione in secondi.
  - K_{EFF}: coefficiente di penalizzazione per la potenza non utilizzata nella funzione obiettivo.
  - K_{VAR}: coefficiente di penalizzazione per il numero di variazioni delle connessioni nella funzione obiettivo.
  - show_p_matrix: impostare True per visualizzare il plot delle matrici delle connessioni (accensione/spegnimento degli switch) istante per istante.
  - opt_timeout: tempo limite per l'esecuzione di un'ottimizzazione, se non viene fornito verrà posto uguale a dt, come in un caso reale.

- Output della funzione: lista di diversi dati utili per tracciare l'evoluzione temporale degli elementi e delle potenze nell'infrastruttura.


ARIM VNIVERSITAS
PAX TIBI MARCE
EVA NGE LISTA MEUS

4 | Risultati delle Simulazioni e Data Visualization

Per studiare le performance dell'algoritmo di simulazione sono state sviluppate delle funzioni per la creazione di grafici. In questo capitolo spieghiamo brevemente i risultati ottenuti per una configurazione base di EVCI.

Configurazione usata per i Test

I test che verranno mostrati in questo lavoro fanno riferimento a una simulazione di un'EVCI con 4 MPU e 3 coppie PU-EV, quindi con una matrice delle connessioni di dimensioni 4x3.

Lo scenario che abbiamo scelto di simulare ha le seguenti caratteristiche parametriche:

- $dt = 10$ s: durata di una finestra di iterazione software.

  La simulazione esegue 400 iterazioni software, quindi simula 4000 secondi totali (circa 1 ora e 7 minuti).

- Rete: $P_{\text{AC MAX}} = 700$ kW costante lungo tutte le iterazioni.

- MPU: sono quattro unità di potenza modulari, tutte con efficienza $\eta_i = 1$. Le potenze nominali sono fisse nel tempo e valgono:

  - $P_{n1} = P_{n2} = 40$ kW.
  - $P_{n3} = P_{n4} = 80$ kW.

- EV: assumiamo che ogni EV che si connette per la ricarica è dello stesso tipo, con parametri ispirati dalle specifiche energetiche e comportamento di ricarica di modelli di veicoli elettrici diffusi. Per ogni EV $j \in \{1, 2, 3\}$:

  - $P^j_{\text{REQ}} = 150$ kW: potenza di ricarica massima.
  - $E^j_n = 60$ kWh: energia nominale delle batterie.

  - $iSoC^j = 0.3$: stato di carica iniziale, quando giunge alla stazione di ricarica.

- Disponibilità delle connessioni: $x_{\text{ij}}^{avl} = 1 \ \forall i \in \{1,2,3,4\} \ \forall j \in \{1,2,3\}$. Quindi le connessioni sono sempre tutte disponibili.

I parametri sono scelti in modo da riprodurre una potenziale situazione reale, in cui la potenza di rete è molto elevata rispetto a quella che possono erogare le MPU o assorbire gli EV. Quindi la rete non satura mai. D'altra parte, la potenza è sicuramente limitata dalle potenze nominali delle MPU: la potenza totale che giunge agli EV non può essere superiore alla loro somma (240 kW). Quindi, a prescindere dagli EV connessi, in questo caso la potenza utilizzata sarà al massimo 240 kW (condizione di saturazione delle MPU).

Attraverso la manipolazione della colonna della tabella in input corrispondente alla connessione attiva o disattiva C_{ev}^j dell'EV j lungo le 100 iterazioni, simuliamo una sequenza di eventi di questo tipo:

Iterazioni $[0, 40]$:	Un unico EV_1 connesso in posizione $j = 1$.
Iterazioni $[40, 100]$:	Rimane EV_1 e si aggiunge un nuovo veicolo EV_2.
Iterazione 101:	Si disconnettono tutti i veicoli.
Iterazioni $[102, 200]$:	Si connettono 2 nuovi veicoli: EV_2 ed EV_3.
Iterazione 201:	Si disconnettono tutti i veicoli.
Iterazioni $[202, 300]$:	Si connettono 3 nuovi veicoli: EV_1, EV_2, EV_3.
Iterazioni $[301, 400]$:	Si disconnette EV_1, rimangono EV_2 e EV_3.

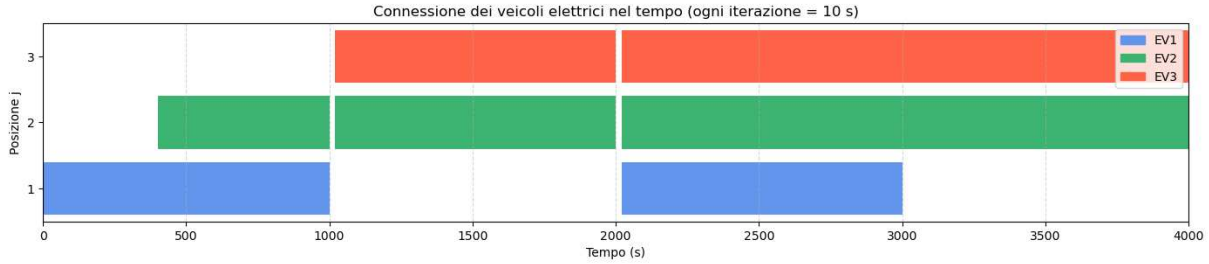


Figura 4.1: Evoluzione della presenza di EV nello scenario di test lungo il tempo.

Tramite questa sequenza di eventi, esemplificata in immagine 4.1, possiamo testare come l'ambiente simulativo gestisce l'allocazione delle risorse in situazioni di tipo diverso.

Scelta dei Coefficienti di Penalizzazione

Per questa fase di test abbiamo utilizzato come coefficienti di penalizzazione $K_{\text{EFF}} = K_{\text{VAR}} = 1$. Questo vuol dire che all'interno dell'ottimizzazione, la funzione obiettivo è

$$\max \left(P_{\text{TOT}} - P_{\text{NOT USED}} - n_{\text{VAR}} \right).$$

Quindi stiamo associando lo stesso peso ai 3 termini della somma. Con questa scelta, l'ottimizzazione darà molta importanza a fare in modo che le MPU lavorino vicino alla potenza nominale e che l'infrastruttura abbia una configurazione stabile (poche variazioni delle connessioni), anche a discapito dell'obiettivo principale, cioè la massimizzazione della potenza erogata.

Se il progetto dovesse diventare realtà e venisse implementato fisicamente, bisognerà scegliere accuratamente i coefficienti di penalizzazione, tramite uno studio accurato, per pesare correttamente gli obiettivi. Per adesso, ci basti questa scelta semplicistica.

4.1. Visualizzazione dei Dati

Nel progetto, i calcoli degli indici di prestazione e la creazione di grafici sono stati implementati tutti nella funzione `plot_sim_results`.

I grafici sono strumenti di visualizzazione per presentare i risultati in modo chiaro. In particolare, ci interessa studiare l'evoluzione dei dati salienti dell'infrastruttura all'avanzare del tempo, per valutare la validità degli algoritmi per il funzionamento efficiente dell'infrastruttura.

Allocazione della Potenza per ogni EV nel tempo I grafici 4.2, 4.3, 4.4 sono riferiti ciascuno ad un EV diverso. Per ogni EV, si vede la potenza assorbita da ciascuna MPU ad ogni istante di tempo, tramite delle barre sovrapposte. Ogni barra monocolore è la potenza erogata dalla MPU del colore corrispondente (legenda in alto a destra). La sovrapposizione delle barre permette di visualizzare la potenza totale assorbita dall'EV come somma delle potenze ricevute da ogni singola MPU connessa.

EV 1

Il grafico 4.2 è riferito all'EV_1 in posizione $j = 1$ nell'EVCI. Con l'aiuto dell'immagine 4.1, sappiamo che un primo EV si connette a tale postazione di ricarica per il primo slot di 1000 secondi.

Durante le prime iterazioni, si può vedere che l'allocazione delle MPU cambia poiché la stima della potenza di ricarica sta aumentando, fino a stabilizzarsi al valore corretto $P^1_{\mathrm{EV}} = P^1_{\mathrm{REQ}} = 150\mathrm{kW}$. Quindi l'algoritmo di stima è riuscito a svolgere il suo scopo con successo.

Da quel momento in poi, rimangono stabili le connessioni di 3 MPU (2 da 40 kW in saturazione, 1 da 80 kW che però eroga solo 70 kW) fino alla disconnessione di EV_1, anche se a metà slot giunge un altro veicolo. Sicuramente, se avessimo dato meno peso

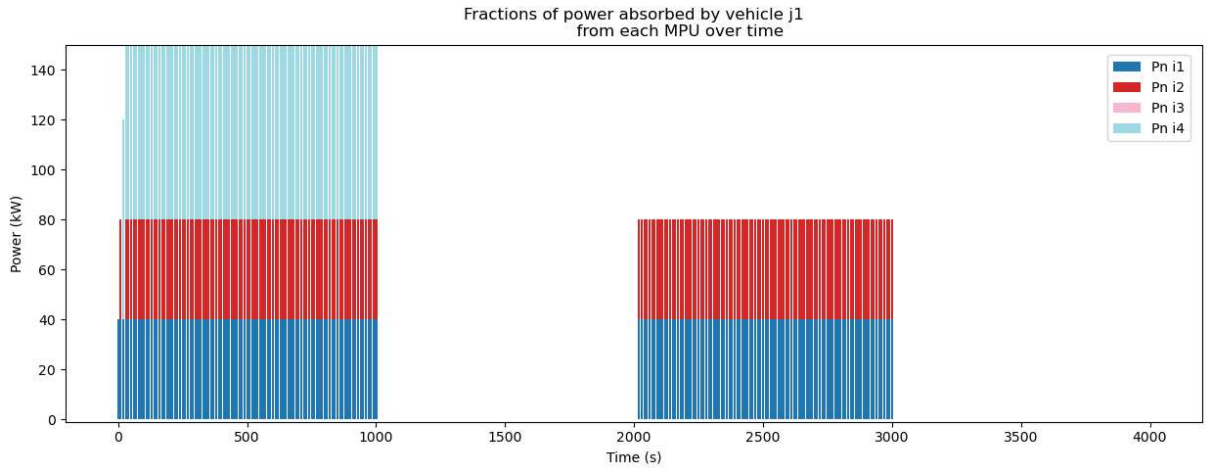


Figura 4.2: Grafico a barre sovrapposte per visualizzare l'allocazione della potenza assorbita dall'EV_1 lungo il tempo

alla stabilità delle connessioni (con un coefficiente K_{var} minore), l'allocazione delle MPU sarebbe cambiata all'arrivo dell'altro EV.

Nell'ultima fase (da circa 2000 a 3000 secondi), giungono 3 nuovi EV, quindi le MPU si distribuiscono in modo da fornire equamente la potenza disponibile. All'EV_1, come si vede dal grafico a barre, vengono assegnati i 2 moduli da 40 kW.

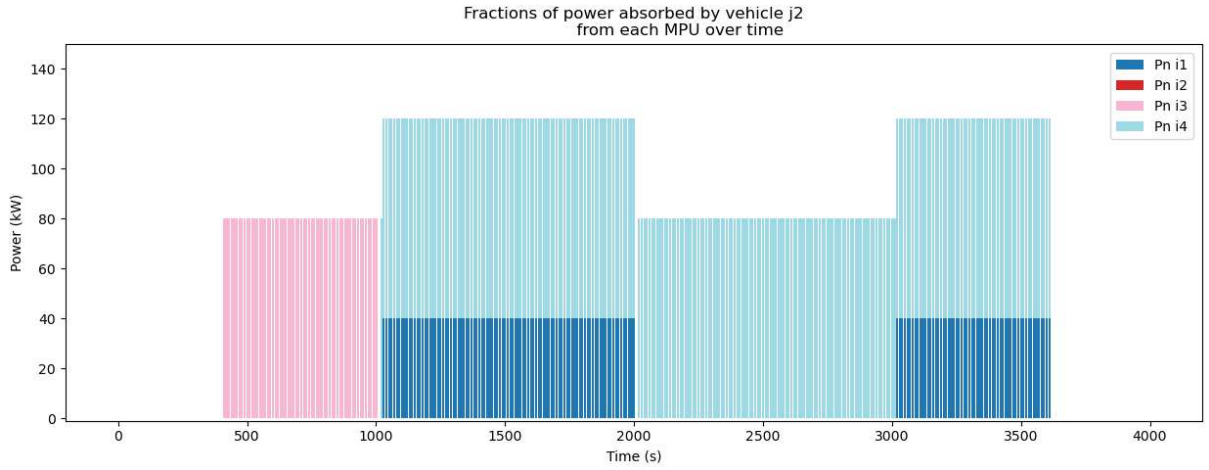


Figura 4.3: Grafico a barre sovrapposte per visualizzare l'allocazione della potenza assorbita dall'EV_2 lungo il tempo

EV 2

Il grafico 4.3 è riferito all'EV_2 in posizione $j = 2$ nell'EVCI.

In una prima fase si connette un EV da 400 a 1000 secondi. Per mantenere la stabilità, l'EVCI assegna un'unica MPU da 80 kW a tale EV (le altre 3 stanno caricando EV_1 da tempo).

Nella seconda fase, si connettono (e poi disconnettono) 2 EV contemporaneamente. Per distribuire equamente la potenza, gli algoritmi assegnano 1 MPU da 40 kW e 1 MPU da 80 kW a ciascun EV.

Nella terza fase, quando si connettono 3 EV contemporaneamente, EV_2 prima riceve 80 kW da MPU_4, mentre dopo la disconnessione di EV_3 riceve anche 40 kW da MPU_1.

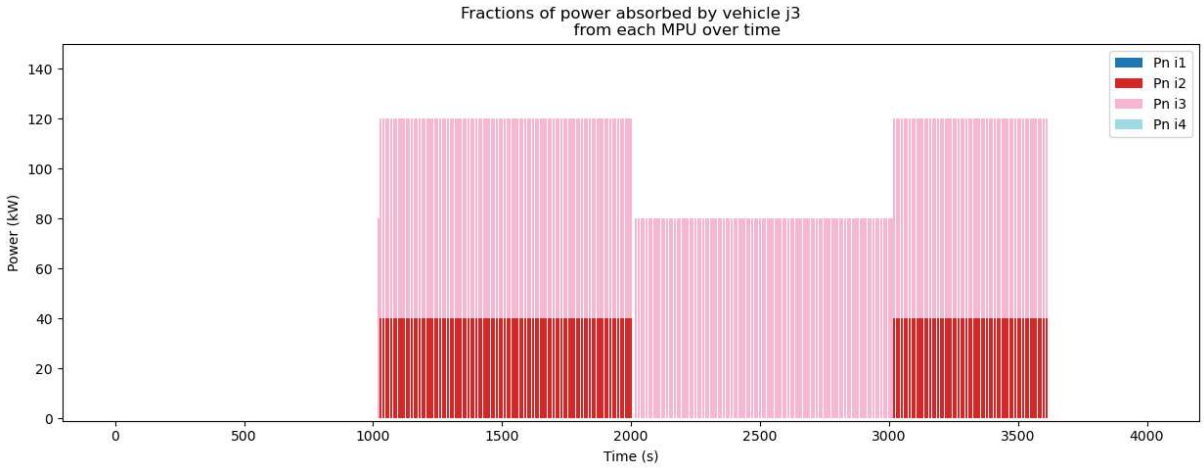


Figura 4.4: Grafico a barre sovrapposte per visualizzare l'allocazione della potenza assorbita dall'EV_3 lungo il tempo

EV 3

Il grafico 4.4 è riferito all'EV_3 in posizione $j = 3$ nell'EVCI. L'allocazione delle MPU è la seguente:

- $[1000, 2000]$ secondi: 2 EV connessi, EV_3 riceve 40 kW da MPU_2 e 80 kW da MPU_3.
- $[2000, 3000]$ secondi: 3 EV connessi, EV_3 riceve solo 80 kW da MPU_3.
- $[3000, 4000]$ secondi: 2 EV connessi, EV_3 torna a ricevere come nel primo slot in cui era presente.

Potenze assorbite dagli EV nel tempo Nel grafico 4.5 vengono riassunte informazioni già visibili nei grafici precedenti, concentrandosi sul confronto tra la potenza assorbita dai veicoli e la potenza richiesta (150 kW) da ciascuno di essi.

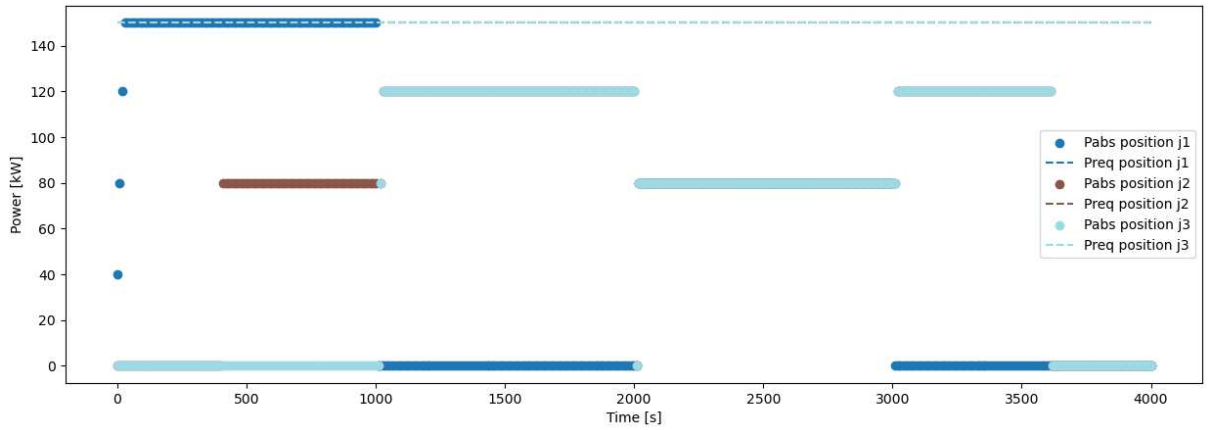


Figura 4.5: Potenza assorbite dagli EV in riferimento alle potenze richieste (tutte uguali in questo caso, rappresentate con una linea tratteggiata).

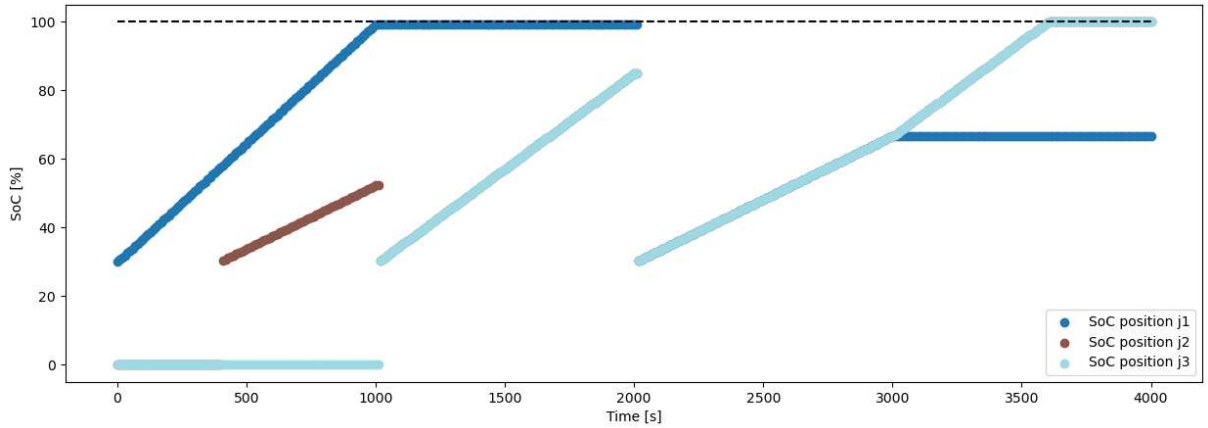


Figura 4.6: Stato di carica di ogni EV nel tempo.

Stato di Carica (SoC) degli EV nel tempo Concentriamoci sui primi 1000 secondi del grafico 4.6 che rappresenta lo stato di carica di ogni EV al variare del tempo. Come si vede, il veicolo EV_1, presente durante l'intero slot temporale iniziale, completa la sua carica prima di disconnettersi avendo 3 MPU connesse; invece a quello che si connette solo durante la seconda metà (per 500 secondi) viene assegnata una sola MPU, e quindi passa da SoC 30 % a circa 50 %, lasciando insoddisfatto il cliente.

Questo forse suggerisce che scegliere K_{VAR} così alto è sbagliato, perché talvolta l'EVCI mantiene stabili le connessioni a discapito dell'obiettivo principale: ricaricare velocemente gli EV dei consumatori.

Potenza Totale nel tempo Dalla figura 4.7 vediamo che la potenza totale trasportata dalla rete segue la domanda totale di potenza, fino al raggiungimento di un massimo,

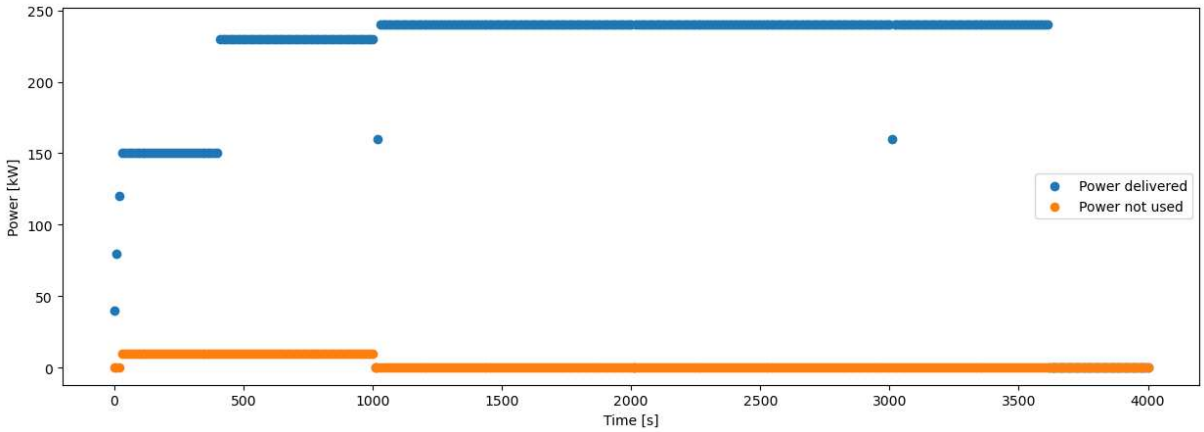


Figura 4.7: Potenza totale trasportata nella rete e potenza delle MPU non utilizzata al variare del tempo.

quando la domanda degli EV eguaglia o supera la massima potenza erogabile dalle MPU (240 kW).

D'altra parte, con potenza non utilizzata si intende la differenza tra la somma delle potenze nominali delle MPU (massima potenza erogabile) e la potenza totale effettivamente erogata. Questo parametro è importante perchè le MPU hanno alta efficienza solo quando lavorano vicino alla loro potenza nominale. Con un coefficiente K_{EFF} alto, come in questo caso, si spingono le MPU a lavorare vicino alla loro potenza nominale: infatti dal grafico si vede che la potenza non utilizzata totale è sempre molto bassa o nulla.

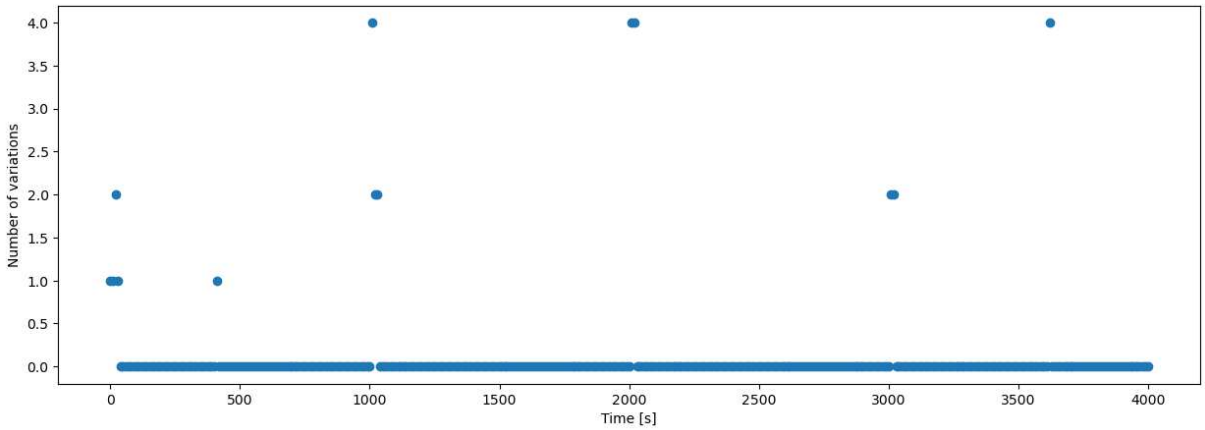


Figura 4.8: Numero di variazioni nelle connessioni della rete elettrica test al variare del tempo.

Numero di Variazioni nel tempo Ovviamente, come ci si può aspettare, avvengono variazioni nelle connessioni della rete (cambiamenti della matrice $\boldsymbol{x}$) solo quando cambia

la configurazione degli EV, e quindi cambia la richiesta di potenza, e anche durante tali cambi il numero di variazioni è limitato.

4.2. Key Performance Indicators (KPI)

Discutiamo i KPI per valutare l'efficacia delle soluzioni proposte.

Lo scopo principale di questo lavoro era quello di creare l'ambiente di simulazione dell'EVCI. Il prossimo passo sarebbe quello di eseguire uno studio accurato delle performance dell'EVCI, eseguendo dei test con diverse configurazioni e parametri dell'ambiente simulativo, per trovare il migliore **set-up parametrico**.

I KPI principali sono quelli riassunti nella funzione obiettivo dell'ottimizzazione: per valutare l'efficacia di una EVCI simulata, si studia quanto bene essa riesca a soddisfare questi indicatori di performance:

- Stimare correttamente la richiesta di potenza da parte degli EV.
- Massimizzare la potenza totale erogata per la ricarica degli EV.
- Spingere le MPU a lavorare alla loro massima efficienza, quindi a potenze alte, vicino alla potenza nominale.
- Mantenere una configurazione stabile delle connessioni tra MPU e PU.

Nell'unico test eseguito e mostrato in questo lavoro, tutti questi requisiti sono stati soddisfatti con discreto successo. Infatti, la visualizzazione dell'andamento di molteplici dati nel tempo ha dimostrato che i KPI sopra elencati sono stati raggiunti, anche con una scelta poco ragionata dei parametri.

Abbiamo constatato che probabilmente una migliore scelta dei coefficienti di penalizzazione, ad esempio, avrebbe un impatto positivo sulle performance, portando a risultati ancora più soddisfacenti. Lasciamo l'analisi parametrica come possibile estensione futura del lavoro.

Conclusioni

Sintesi del Lavoro

Nel corso di questa tesi è stato sviluppato un ambiente di simulazione finalizzato alla validazione di algoritmi di ottimizzazione per l'allocazione della potenza in infrastrutture di ricarica per veicoli elettrici (EVCI), con particolare attenzione alle stazioni DC Fast Charging e all'utilizzo di architetture modulari.

Il sistema, realizzato interamente in linguaggio Python, è composto da tre livelli logici principali: l'algoritmo di ottimizzazione basato su programmazione lineare intera mista (MILP), la struttura hardware simulata (MPU, PU, EV) e il motore di simulazione temporale. L'interazione tra questi elementi ha permesso di riprodurre il comportamento dinamico dell'infrastruttura in scenari con domanda variabile, vincoli di potenza e configurazioni flessibili.

Il contributo originale del lavoro risiede nell'integrazione tra simulazione temporale quasi stazionaria e ottimizzazione centralizzata, in un contesto software modulare ed estendibile. In particolare, sono stati definiti e implementati:

- un algoritmo iterativo per la stima della potenza massima erogabile in condizioni di saturazione progressiva;
- una logica di flusso informativo inverso;
- una formulazione MILP per l'allocazione ottimale delle risorse, parametrizzata e testabile in scenari differenti.

I risultati ottenuti dimostrano la flessibilità e la robustezza dell'ambiente simulativo, nonché la validità dell'approccio adottato per il testing algoritmico in contesti realistici. Il sistema si configura come uno strumento utile per analizzare criticamente il funzionamento di EVCI complesse e per supportare lo sviluppo di strategie di controllo intelligenti.

Possibili Estensioni del Lavoro

Esistono diverse possibilità per gli sviluppi futuri del lavoro. Alcune di queste sono legate al miglioramento del framework che è già stato implementato, altre sono proposte di aggiunta al lavoro già esistente.

Innanzitutto, come già spiegato, è necessaria un'analisi per trovare il **set-up parametrico** più adatto, con un'attenzione particolare ai coefficienti di penalizzazione degli addendi della funzione obiettivo del problema MILP.

Inoltre, sarebbe utile valutare le performance di altri solver nella ricerca di una soluzione del problema MILP. Utilizzare un solver più potente permetterebbe di minimizzare i tempi di calcolo e assicurare la convergenza anche con reti elettriche più complesse.

Una ulteriore direzione di estensione riguarda l'introduzione di **modelli stocastici**, per simulare la variabilità reale degli arrivi dei veicoli elettrici, dei tempi di connessione e delle richieste energetiche. Questo permetterebbe di valutare la robustezza degli algoritmi anche in presenza di incertezza e comportamenti non deterministici.

Dal punto di vista algoritmico, si potrebbe implementare un approccio di **ottimizzazione multi-obiettivo**, in cui la funzione obiettivo tenga conto simultaneamente di criteri come l'efficienza energetica, la rapidità di ricarica e la minimizzazione dell'usura degli switch, attraverso tecniche come il fronte di Pareto.

Un ulteriore miglioramento del modello consiste nell'estensione della **formulazione MILP**, integrando variabili discrete legate alla priorità degli utenti. Ciò consentirebbe di differenziare tra utenti in emergenza, utenti ordinari e profili premium, nonché di gestire vincoli temporali o di carico della rete a monte.

Dal punto di vista funzionale, l'aggiunta di una **interfaccia grafica interattiva** e di una **dashboard di monitoraggio** permetterebbe di facilitare l'uso del simulatore anche da parte di utenti non tecnici, visualizzando in tempo reale metriche chiave come potenza erogata, stato delle connessioni e performance energetiche.

Un altro sviluppo rilevante riguarda l'**integrazione di fonti rinnovabili e sistemi di accumulo (BESS)**, al fine di simulare scenari ibridi in cui parte dell'energia è fornita localmente e gestita in modo intelligente per ridurre il prelievo dalla rete.

Infine, il framework potrebbe essere adattato per gestire **reti topologicamente complesse non matriciali**, abbandonando l'assunzione di connessioni centralizzate o tabellari, e modellando configurazioni reali con ramificazioni, moduli in cascata o stazioni distribuite su più nodi.

Questi sviluppi aprono la strada a una maggiore fedeltà del modello rispetto alle infrastrutture reali e alla possibilità di impiegarlo come strumento di supporto decisionale in fase di progettazione o gestione operativa di sistemi di ricarica intelligenti.


ARIM VNIVERSITAS
PAX TIBI MARCE
EVA NGE LISTA MEUS

Bibliografia

[1] I. Safak Bayram, George Michailidis, Michael Devetsikiotis, and Fabrizio Granelli. Electric power allocation in a network of fast charging stations. *IEEE Journal on Selected Areas in Communications*, 31(7), 2013. doi: 10.1109/JSAC.2013.130707.


ARIM VNIVERSITAS
PAX TIBI MARCE
EVA NGE LISTA MEUS

Lista degli Acronomi

Acronimo	Significato
EVCI	Electric Vehicle Charging Infrastructure
DCFC	Direct Current Fast Charging
MPU	Modular Power Unit
PU	Plug Unit
EV	Electric Vehicle
MILP	Mixed Integer Linear Programming
BESS	Battery Energy Storage System
LP	Linear Programming
DES	Discrete Event Simulation
RMS	Root Mean Square
SoC	State of Charge


ARIM VNIVERSITAS
PAX TIBI MARCE
EVA NGE LISTA MEUS

Elenco delle figure


ARIM VNIVERSITAS
PAX TIBI MARCE
EVA NGE LISTA MEUS

Ringraziamenti

Questo progetto è frutto di un tirocinio di 200 ore effettuato nel reparto Ricerca e Sviluppo (team Energy Storage) dell'azienda Socomec (Sicon SRL) di Isola Vicentina. Ringrazio l'azienda per l'opportunità che mi è stata data.

Soprattutto, ringrazio vivamente Riccardo Caliaro ed Eliseo Tiso per il supporto datomi durante lo sviluppo di questo lavoro. Il percorso effettuato a fianco del team di Ricerca e Sviluppo mi ha dato moltissimi spunti di ragionamento e di crescita. Ho avuto la possibilità di imparare come trasformare un'idea in fase embrionale in realtà, creando costantemente nuove proposte, per poi smontarle e ricrearle con una nuova consapevolezza, ispirato dalla lucidità ed esperienza di Riccardo ed Eliseo.

Riccardo mi è stato molto vicino anche durante la stesura della tesi, sempre disponibile per una chiamata o un consiglio di ogni genere, con passione e curiosità per il progetto che assieme stavamo seguendo.

Infine, ringrazio il mio relatore Marco Salvatore Nobile per il supporto durante la stesura della tesi e per la disponibilità.


ARIM VNIVERSITAS
PAX TIBI MARCE
EVA NGE LISTA MEUS