Agenzia nazionale per le nuove tecnologie,
l'energia e lo sviluppo economico sostenibile

MINISTERO DELLO SVILUPPO ECONOMICO

Ricerca di Sistema elettrico

# Sviluppo di algoritmi per l'ottimizzazione del processo di elettrificazione di reti di trasporto pubblico urbano attraverso l'analisi delle possibili configurazioni di ricarica su sottoinsiemi della rete di trasporto

Giuseppe Francesco Italiano

Università di Roma Tor Vergata

Report RdS/PAR2017/235

# SVILUPPO DI ALGORITMI PER L'OTTIMIZZAZIONE DEL PROCESSO DI ELETTRIFICAZIONE DI RETI DI TRASPORTO PUBBLICO URBANO ATTRAVERSO L'ANALISI DELLE POSSIBILI CONFIGURAZIONI DI RICARICA SU SOTTOINSIEMI DELLA RETE DI TRASPORTO

Giuseppe Francesco Italiano (Università degli studi di Roma Tor Vergat)

Settembre 2018

Report Ricerca di Sistema Elettrico

Accordo di Programma Ministero dello Sviluppo Economico - ENEA
Piano Annuale di Realizzazione 2017
Area: "Efficienza energetica negli usi finali elettrici e risparmio energia negli usi finali elettrici ed interazione con altri vettori elettrici"
Progetto: "Mobilità elettrica sostenibile"
Obiettivo: "Scenari di mobilità elettrica", sub-obiettivo "Strumenti di supporto TPL".
Responsabile del Progetto: Maria Pia Valentini, ENEA

Il presente documento descrive le attività di ricerca svolte all'interno dell'Accordo di collaborazione "Sviluppo di algoritmi per l'ottimizzazione del processo di elettrificazione di reti di trasporto pubblico urbano attraverso l'analisi delle possibili configurazioni di ricarica su sottoinsiemi della rete di trasporto"
Responsabile scientifico ENEA: Massimo Celino
Responsabile scientifico Università degli studi di Roma Tor Vergata: Giuseppe Francesco Italiano

# Indice

# Sommario

Questo lavoro di ricerca ha come obiettivo quello dell'ottimizzazione delle soluzioni per l'elettrificazione delle reti di trasporto pubblico. In questo ambito, l'attività dell'Università degli Studi di Tor Vergata si è concentrata sulla progettazione, analisi e ingegnerizzazione di algoritmi che siano in grado di trovare la soluzione ottima, esplorando lo spazio delle possibili soluzioni in modo parallelo e sfruttando l'infrastruttura di supercalcolo CRESCO per il calcolo scientifico ad alte prestazioni disponibili all'interno dell'ENEA.

Le principali sfide affrontate sono state la realizzazione di un'architettura del programma diversa da quella adottata nel precedente anno di attività e l'analisi della rete di trasporto con l'architettura C. In particolare, la realizzazione di un'architettura del codice in grado di sfruttare la potenzialità dei job array e la progettazione di algoritmi in grado di facilitare l'analisi per il caso di autobus con ricarica anche alle fermate sono state due delle attività su cui si è posto maggiormente l'attenzione.

Infatti, grazie all'utilizzo dei job array si è riusciti ad ottenere un programma in grado di poter migliorare le prestazioni e la grandezza delle reti analizzate, nonché rendere più semplice la lettura del codice. Si sono implementati 3 programmi distinti, che rappresentano i diversi step che devono essere eseguiti per poter ottenere la soluzione ottima al problema. Nel dettaglio si è implementato un primo step in grado di generare molteplici file che rappresentano gli input analizzabili separatamente da ogni singolo processore. Nel secondo step si utilizza la tecnica dei job array, in cui ogni singolo job si occupa di analizzare l'input contenuto in un sottoinsieme di file di input generati nello step uno. Infine, il terzo step serve per poter comprendere il lavoro svolto durante la fase due e capire se si sono analizzati tutti i casi possibili, oppure se è necessario analizzarne altri prima di capire quale sia la soluzione migliore al problema di elettrificazione.

La differenza principale di questa architettura con la precedente è che si tratta di un'architettura statefull, che ci permette di poter salvare lo stato e poi effettuare un'altra analisi successiva. In tal modo, è possibile sfruttare tutto il tempo macchina per calcoli utili a trovare la soluzione ottima. Invece, nel programma precedente, molti test che sono falliti per mancanza di tempo, non hanno prodotto dei risultati utili.

La realizzazione di una nuova architettura è nata dalla volontà di provare ad analizzare la rete di Roma, che è molto più grande di quella di Firenze, analizzata nel precedente anno di attività.

Inoltre, si sono analizzate alcune caratteristiche della rete C, in particolar modo soffermandoci sul numero di ricariche massime installabili sui nodi fermata. Tale analisi ci ha permesso di scoprire quali sono i nodi fermata dove passano con frequenza alta diversi autobus di varie linee. Delle migliaia di nodi fermata presenti nella rete di Roma, solo 19 necessitano di più di 1 stazione di ricarica per soddisfare il traffico degli autobus che vi transitano.

Questo risultato insieme ad altre future analisi, saranno alla base di ciò che consentirà di analizzare la soluzione ottima anche in caso di architettura di elettrificazione di tipo C, ossia con linee di autobus che si ricaricano anche alle singole fermate.

# 1 Introduzione

L'accordo di collaborazione tra ENEA e il dipartimento di Ingegneria Civile ed Ingegneria Civile (DICII) dell'Università di Roma Tor Vergata ha come obiettivo l'ottimizzazione delle soluzioni per l'elettrificazione delle reti di trasporto pubblico. In particolare, l'Università contribuisce allo sviluppo di modelli di ottimizzazione attraverso l'analisi delle sottoreti di trasporto nelle differenti configurazioni di ricarica dei veicoli.

Con la realizzazione di questi modelli si vuole verificare la fattibilità tecnica ed economica dell'alimentazione elettrica autonoma, tramite opportune tecnologie di ricarica, per il trasporto pubblico su gomma di una città di dimensioni significative. Inoltre, tale modello consente di capire se la soluzione elettrica risulta economicamente vantaggiosa rispetto alle alternative convenzionali e di individuare le linee di trasporto maggiormente adatte alla trasformazione in alimentazione elettrica.

In particolare, l'Università degli Studi di Tor Vergata, in questo anno di lavoro, si è concentrata non solo sulla progettazione, analisi e ingegnerizzazione di algoritmi che siano in grado trovare la soluzione ottima, esplorando lo spazio delle possibili configurazioni di ricarica su sottoinsiemi opportuni delle linee della rete del trasporto pubblico considerata, ma anche sulla realizzazione di un'architettura del codice nuova e sull'analisi della modalità di ricarica di tipo C. Gli algoritmi e la nuova architettura del codice sono stati implementati sull'infrastruttura di supercalcolo CRESCO (Computational RESearch centre on Complex system) per il calcolo scientifico ad alte prestazioni HPC (High Performance Computing) disponibili all'interno dell'ENEA.

Risolvere un problema che va alla ricerca delle soluzioni ottime può non essere così semplice. Per trovare l'ottimo spesso si deve esplorare un insieme elevato di soluzioni, e ciò può richiedere un grande tempo di calcolo.

Per realizzare un software applicativo che calcoli la soluzione esatta, e che sia applicabile anche a istanze di dimensioni significative, si ha bisogno di poter realizzare e progettare applicazioni in grado di parallelizzare il calcolo e di bilanciare il carico di lavoro tra molteplici risorse di calcolo.

Ciò che si deve risolvere per tale lavoro è un problema comune, ossia si ha un grande numero di job da eseguire e tali job sono identici in termini di istruzioni da eseguire. L'idea è che si hanno molteplici insiemi di dati che si vogliono analizzare con il run di un singolo programma, usando il cluster di macchine a disposizione. La soluzione banale sarebbe quella di generare molteplici scripts di shell e sottometterli alla coda. Tale soluzione non è efficiente né per chi deve effettuare il run del codice, né per il sistema stesso.

Durante il periodo di progetto si è realizzata una nuova architettura del programma in grado di sfruttare la tecnica dei job array disponibile sull'infrastruttura di CRESCO. Tale tipo di architettura ci assicura un bilanciamento del carico efficiente. Ed ha i seguenti vantaggi:

- Permette di scrivere un solo script per la shell.
- Se si sottomette un array job e ci si accorge di aver commesso un errore, si può annullare tutto cancellando un unico id job dalla coda che identifica l'intero array.
- Il bilanciamento del carico non è più un elemento critico. Il job array utilizza le risorse che ha a disposizione, ed una volta terminato il lavoro assegnato ad un job rilascia le risorse utilizzate.

L'obiettivo di questa applicazione software è quello di riuscire a calcolare la soluzione esatta per questo problema di ottimo, in cui vengano prese in considerazione le 3 architetture di elettrificazione: arch. A (con ricarica delle batterie solo al deposito), arch. B (con ricarica delle batterie al deposito e ai nodi capolinea) e arch. C (con ricarica al deposito, ai capolinea e ai nodi fermata). Per ottenere la soluzione migliore, si deve usare un approccio di tipo ricerca esaustiva, in grado di considerare tutte le possibili soluzioni ammissibili, al

fine di trovare quella ottima. Tuttavia, lo spazio delle soluzioni del problema è tale che, volendo esaminare una rete con L linee elettrificabili con al più due tipi di architetture, si dovrebbero esaminare:

$$\sum_{k=1}^{L} \binom{L}{k} * 3^k \text{ soluzioni}$$

Questo numero ci fa comprendere che analizzare uno spazio di soluzioni così grande in tempi ragionevoli e con una semplice architettura di elaborazione (come ad esempio un PC) è impensabile.

Per tale motivo si è implementato un algoritmo in grado di:

- Esplorare lo spazio delle soluzioni in modo parallelo.
- Esplorare ogni possibile soluzione una e una sola volta, facendola esplorare ad uno e un solo processore.
- Sfruttare al meglio le risorse hardware di CRESCO con l'utilizzo dei job array.
- Effettuare tagli di porzioni di rete che non conterranno soluzioni ottime.
- Analizzare in maniera preventiva le caratteristiche delle linee elettrificabili con l'architettura di tipo C.

Quest'ultimo obiettivo è perseguibile analizzando le diverse caratteristiche dei nodi fermata. Ad esempio, si è data importanza nel comprendere quale fosse il numero massimo di postazioni di ricarica necessarie per ogni singolo nodo fermata. Tale analisi fatta preventivamente ci consente di evitare il calcolo del numero di postazione di ricarica in fase di esecuzione dell'algoritmo di rete.

Nei prossimi capitoli verranno descritti la formalizzazione del problema di ottimizzazione del trasporto pubblico, la progettazione dell'architettura del programma con i job array, il manuale utente e le analisi fatte a riguardo dell'architettura C.

# 2 Formalizzazione del problema di ottimizzazione del trasporto pubblico

L'obiettivo di questo lavoro è trovare le migliori sottoreti (insieme di linee di bus) da elettrificare, sapendo di avere un budget limitato per far ciò. Il programma esplorerà tutte le linee di autobus attraverso una ricerca esaustiva in grado di considerare tutte le possibili combinazioni di sottoreti e tutti i possibili modi di elettrificare tali sottoreti.

## *2.1 Funzione obiettivo e vincoli*

### 2.1.1 Premessa

Per poter comprendere la modellizzazione del problema è giusto fare una breve panoramica a riguardo dei trasporti pubblici su gomma. Una rete di TPL può essere vista come un insieme di linee, ed ognuna di esse avrà un determinato numero di autobus che è in grado di soddisfare l'intero servizio giornaliero.

Per il nostro obiettivo, una linea (*L*) è in realtà vista come un insieme di costi: i costi di investimento (*I*) e i costi operativi (*C*). In particolare:

- Nel caso del trasporto tradizionale (*T*) si hanno:
  - $IL_l^T$ Costo di investimento di linea per il caso di alimentazione tradizionale (diesel).
  - $CL_l^T$ Costo operativi di linea per il caso di alimentazione tradizionale (diesel).
- Nel caso del trasporto elettrico (*E*) si hanno:
  - $IL_{la}^E$ Costo di investimento di linea.
  - $CL_{la}^E$ Costo operativo di linea.
  - $IN_{la}^{DE}$ Costo di investimento di nodo deposito.
  - $CN_{la}^{DE}$ Costo operativo per il nodo deposito.
  - $IN_{la}^{CE}$ Costo di investimento per i nodi capolinea.
  - $CN_{la}^{CE}$ Costo operativo per i nodi capolinea.
  - $IN_{la}^{FE}$ Costo di investimento per i nodi fermata.
  - $CN_{la}^{FE}$ Costo operativo per i nodi fermata.

Inoltre, nel caso elettrico è da specificare che si può decidere di elettrificare una linea utilizzando tre tipi diversi di architetture (*a*) di autobus:

1. Tipologia *A*: autobus che necessitano di ricarica della batteria solo a deposito (*D*).
2. Tipologia *B*: autobus che necessitano di ricarica della batteria a deposito e capolinea (*C*).
3. Tipologia *C*: autobus che necessitano di ricarica della batteria a deposito, capolinea e alle fermate (*F*).

### 2.1.2 Modello matematico

Il programma che si è realizzato vuole risolvere un problema di ottimizzazione: dato un budget di finanziamento destinato agli investimenti iniziali (acquisto di veicoli e infrastrutture di ricarica), quale sottorete della rete complessiva di servizio conviene elettrificare e quale tipologia di ricarica scegliere per ogni linea?

La funzione obiettivo (si veda l'eq. 2.1) è quella di massimizzare la differenza (intesa come guadagno) tra i costi complessivi del trasporto tradizionale e i costi complessivi del trasporto elettrico.
L'unico vincolo a cui si è soggetti è il vincolo di budget disponibile per gli investimenti iniziali (si veda l'eq. 2.2). Infatti, si vuole che i costi di investimento per il trasporto elettrico siano compresi tra il budget massimo ($I^M$) e una percentuale di questo valore ($tol \cdot I^M$).

Ne segue che il modello matematico del problema può essere scritto nel seguente modo:

Funzione obiettivo (*eq. 2.1*):

$$\max \sum_{l \epsilon L} \sum_{a \epsilon \{A,B,C\}} \left( \left( IL_l^T + CL_l^T \right) - \left( IL_{la}^E + CL_{la}^E \right) - \left( IN_{la}^{DE} + CN_{la}^{DE} \right) \right) \cdot X_{la} - \sum_{l \epsilon L} \sum_{n \epsilon \{C,S\}} \sum_{a \epsilon \{B,C\}} K_a^n \cdot \left( IN_{la}^{nE} + CN_{la}^{nE} \right) \cdot X_{la}$$

Vincoli (*eq. 2.2*):

$$tol \cdot I^M \leq \sum_{l \epsilon L} \sum_{a \epsilon \{A,B,C\}} \left( IL_{la}^E + IN_{la}^{DE} \right) \cdot X_{al} + \sum_{l \epsilon L} \sum_{n \epsilon \{C,S\}} \sum_{a \epsilon \{B,C\}} K_a^n \cdot IN_{la}^{nE} \cdot X_{al} \leq I^M$$

In cui il significato dei parametri è:

- $K_a^n$ indica la riduzione di costo che possono subire le infrastrutture di ricarica per gli autobus di una architettura *a* ad un nodo *n*, dovuto al fatto che più linee possono condividere le infrastrutture di ricarica. In questo modo si possono abbassare i costi operativi e di investimento relativi ai nodi capolinea.
- $X_{al}$ indica se nella sottorete che si sta considerando è presente la linea *l*, e se si è scelto di elettrificarla con l'architettura *a*. Vale 1 se è presente tale configurazione, 0 altrimenti.

Nota bene: per quanto riguarda la riduzione di costo per le linee elettrificate con l'architettura B e C si cercherà di effettuare un'analisi dettagliata che non tenga conto di un fattore di riduzione di costo generico $K_a^n$, bensì si terrà conto del vero effetto di rete, andando ad analizzare le reali stazioni di ricarica condivise dalle diverse linee.

## 2.2 Lo spazio delle soluzioni

Ogni soluzione di questo problema di ottimizzazione è individuata da:

- Una sottorete *S*, ossia un sottoinsieme di linee di autobus contenuto nell'insieme di *L* linee della rete completa.
- Le architetture con cui si è deciso di elettrificare ogni linea della sottorete *S*.
- I nodi capolinea di tipo B che si è deciso di elettrificare ed il relativo numero di infrastrutture previste al nodo.
- I nodi capolinea di tipo C che si è deciso di elettrificare ed il relativo numero di infrastrutture previste al nodo.
- I nodi fermata di tipo C che si è deciso di elettrificare ed il relativo numero di infrastrutture previste al nodo.

Indicando con *L* il numero di linee della rete complessiva che si sta analizzando, e sapendo che per questo anno di lavoro si analizzeranno *tutte le 3 architetture possibili* con cui elettrificare ogni linea (l'architettura *A*, la *B e la C*), si ottiene che lo spazio possibile delle soluzioni è pari a:

$$\sum_{k=1}^{L} \binom{L}{k} \cdot 3^k \qquad (eq. 2.3)$$

L'equazione 2.3 mostra la grandezza dello spazio delle soluzioni. Infatti, le soluzioni possibili vanno cercate tra tutte le possibili combinazioni delle linee di autobus e per ognuna di queste combinazioni vanno analizzate tutte le possibili disposizioni con ripetizione delle tre architetture.

In questo modo, con le combinazioni delle linee andremo ad esplorare tutte le possibili sottoreti della rete complessiva. Invece, andando ad analizzare tutte le disposizioni con ripetizione delle tre architetture si analizzeranno tutte le scelte possibili di come elettrificare le diverse linee di una sottorete.

L'equazione 2.3 ci fa capire come il voler effettuare una ricerca esaustiva della soluzione esatta di questo problema, sia un problema np-hard, la cui complessità cresce esponenzialmente con il numero di linee della rete da analizzare.
In particolar modo l'analisi si complica ancora di più con l'introduzione dell'architettura C. In questo caso, ogni qual volta si considerano delle linee che possono essere elettrificate anche nelle fermate, si deve cercare di comprendere quale siano le fermate da elettrificare che garantiscano il servizio e che abbiano il costo minore. In tal modo si introduce un nuovo problema di ottimizzazione nel già citato problema. Infatti, bisogna trovare una delle possibili disposizioni ottime delle postazioni di ricarica ai nodi fermata per la sottorete contenente linee elettrificabili con l'architettura C.

## *2.3 Descrizione della rete di input usata come dataset di input*

Ogni caso di studio è composto da due file:

- *linee_output_best.csv*: In questo file sono presenti tutte le linee e per ogni linea tutte le voci di costo relative ad ogni possibile architettura. In particolare, si ha una struttura del tipo *(id linea, id architettura, voci di costo)*. In cui le voci di costo si differenziano in:
    - *invest_iniziale_bus*: Costo di investimento (di linea) iniziale per gli autobus.
    - *costo_energia*: Costo operativo di linea per l'energia.
    - *manut_bus*: Costo operativo di linea per la manutenzione degli autobus.
    - *invest_impianti_dep*: Costo d'investimento degli impianti ai depositi.
    - *invest_impianti_cap*: Costo d'investimento degli impianti ai capolinea.
    - *invest_impianti_ferm*: Costo d'investimento degli impianti alle fermate.
    - *manut_impianti_dep*: Costo operativo per la manutenzione degli impianti ai depositi.
    - *manut_impianti_cap*: Costo operativo per la manutenzione degli impianti ai capolinea.
    - *manut_impianti_ferm*: Costo operativo per la manutenzione degli impianti alle fermate.
    - *costo_allacci_dep*: Costo operativo per gli allacci ai depositi.
    - *costo_allacci_cap*: Costo operativo per gli allacci ai capolinea.
    - *costo_allacci_ferm*: Costo operativo per gli allacci alle fermate.
    - *costo_potenza_dep*: Costo operativo per la potenza erogata ai depositi.
    - *costo_potenza_cap*: Costo operativo per la potenza erogata ai capolinea.
    - *costo_potenza_ferm*: Costo operativo per la potenza erogata alle fermate.

- *percorsi.csv*: In questo file sono presenti tutti i percorsi effettuati dalla rete che si sta analizzando e in particolare viene specificato:
    - *percorso_id*: L'id del percorso considerato
    - *linea_id*: L'id della linea a cui si riferisce quel percorso.
    - *capolinea_i*: Il capolinea di partenza del percorso.
    - *capolinea_f*: Il capolinea di arrivo del percorso.
    - *ora_arrivo*: La fascia oraria che si sta considerando.
    - *frequenza*: Il numero di corse effettuate in quella fascia oraria.

Il file *linee_output_best.csv* contiene tutte le voci di costo con cui sono stati calcolati i valori presenti nella funzione obiettivo e nei vincoli, con l'eccezione del valore $K_a^n$ che viene calcolato in base ai percorsi che una linea percorre.

In particolare, si ha che:

- $IL_{la}^{E} = invest\_iniziale\_bus$
- $CL_{la}^{E} = costo\_energia + manut\_bus$

- $IN_{la}^{DE} = invest\_impianti\_dep$
- $CN_{la}^{DE} = manut\_impianti\_dep + costo\_allacci\_dep + costo\_potenza\_dep$
- $IN_{la}^{CE} = invest\_impianti\_cap$
- $CN_{la}^{CE} = manut\_impianti\_cap + costo\_allacci\_cap + costo\_potenza\_cap$
- $IN_{la}^{FE} = invest\_impianti\_ferm$
- $CN_{la}^{FE} = manut\_impianti\_ferm + costo\_allacci\_ferm + costo\_potenza\_ferm$

Di grande importanza per l'analisi delle linee elettrificabili con l'architettura C sono anche i file:

- *percorsi_fermate_C.csv*: In tale file per ogni nodo fermata sono riportati l'identificativo della linea che passa per quel nodo, e il numero di corse che tale linea effettua nelle diverse ore della giornata.
- *percorsi_fermate.csv*: In questo file per ogni percorso è riportato il consumo che c'è in ogni tratto tra due fermate consecutive.

### 2.3.1 Il caso di studio di Roma

Tra i casi di studio di maggiore interesse vi è quello di Roma. Si tratta di una rete molto più grande della precedente rete di Firenze che è stata analizzata nello scorso anno.

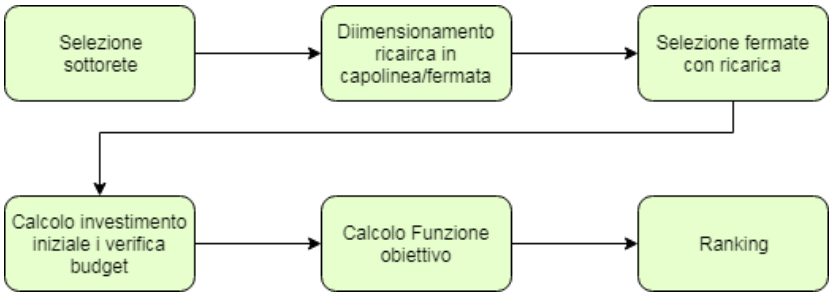
**Tabella 2.1: Confronto tra i numeri della rete di Firenze e la rete di Roma.**

| | Numero di linee | Numero di capolinea |
|---|---|---|
| Firenze | 85 | 101 |
| Roma | 292 | 238 |

## *2.4 Flow Chart della procedura generale*

Per comprendere il flusso delle operazioni che sono necessarie per la risoluzione del problema di ottimizzazione si riporta il seguente diagramma di flusso:

**Figura 2.1: Flow chart che illustra la procedura per ottenere il ranking delle migliori sottorete da elettrificare.**




Si può notare come le operazioni da fare sono molto simili a quelle effettuate nello scorso anno di lavoro, con l'aggiunta del dimensionamento ai nodi ricarica e fermata, e la conseguente scelta dei nodi fermata da elettrificare:

- Il dimensionamento della ricarica ai nodi capolinea e fermate ci permette di capire quante stazioni di ricarica si debbano mettere in ciascun nodo, in base alla sottorete considerata.
- Una volta effettuato il dimensionamento si può passare alla selezione delle fermate su cui installare realmente le stazioni di ricarica. Tale scelta deve essere fatta in modo tale da essere la scelta ottimale, rispetto a tutte le possibili disposizioni delle fermate con ricarica. Inoltre, deve essere rispettato il vincolo dei consumi tra i diversi archi fermata.

# 3 Progettazione della nuova architettura del programma con i job array

In questo capitolo si spiegherà la nuova tecnica di programmazione utilizzata per scrivere il codice ed effettuarne il run. Verrà descritto l'uso dei job array, e come sia cambiata la struttura del programma rispetto al precedente anno.

## *3.1 Precedente struttura del programma e metodologie usate*

In questa sezione si analizzerà la struttura del programma realizzato durante il precedente anno di lavoro. Tale paragrafo si rende necessario per poter comprendere le motivazioni e i benefici che la nuova architettura del programma ha portato al progetto.

### 3.1.1 Metodologie: MPI e C++

Per l'esecuzione dei test del programma realizzato per il lavoro di ricerca dell'anno 2017, si sono utilizzate le risorse di calcolo di Cresco 3 e Cresco 4. La scelta per la scrittura del programma è ricaduta sul linguaggio C++ e sul protocollo di comunicazione tra processi MPI [1].

Il motivo di usare il C++ piuttosto che altri linguaggi è legato alle prestazioni offerte dal linguaggio. Inoltre, il C++ offre un mezzo di astrazione che, al contrario di altri linguaggi, non produce overhead delle prestazioni a runtime. In questo modo si può scrivere un codice efficiente, che ha anche un alto livello di astrazione.

Il programma pensato per essere eseguito su molteplici nodi del cluster necessita di un protocollo di comunicazione tra i computer. La scelta in questo caso è ricaduta su MPI (Message Passing Interface). Infatti, MPI è lo standard per la comunicazione tra nodi appartenenti a un cluster di computer che eseguono un programma parallelo sviluppato per sistemi a memoria distribuita.

Message passing Interface è un protocollo il cui consenso è condiviso da più di 40 organizzazioni partecipanti, tra cui vendor, ricercatori, sviluppatori di librerie software e utenti. Tuttavia, MPI non è uno standard IEEE o ISO, ma è diventato di fatto uno standard delle industrie per poter scrivere programmi che hanno bisogno di scambiare messaggi su piattaforme di HPC.

L'obiettivo principale di MPI è quello di stabilire uno standard flessibile, efficiente e portabile. Infatti, MPI rispetto alle precedenti librerie utilizzate per il passaggio di parametri tra nodi, ha il vantaggio di essere molto portabile (è stata implementata per moltissime architetture parallele) e veloce (viene ottimizzata per ogni architettura).

È importante sottolineare il fatto che MPI non è una libreria, ma piuttosto la specifica di come una libreria di message passing dovrebbe essere realizzata. MPI si occupa principalmente del modello di programmazione parallela a passaggio di messaggi: i dati si muovono dallo spazio di indirizzamento di un processo a quello di un altro, attraverso operazioni combinate su ciascun processo. Tutto il parallelismo è esplicito: il programmatore è responsabile dell'identificazione corretta del parallelismo e dell'implementazione di algoritmi paralleli usando i costrutti MPI. Per la compilazione del programma si è usata l'implementazione di MPI che si chiama "Open MPI" (versione 1.4.3).

### 3.1.2 Struttura del programma

Il programma che si è realizzato per il caso applicativo è costituito da quattro fasi principali:
1. Inizializzazione dell'ambiente MPI, delle variabili e degli oggetti necessari per l'analisi della rete.
2. Il bilanciamento del carico, effettuato dal nodo master, per comunicare a tutti i processori il carico di lavoro da svolgere.
3. L'esplorazione della porzione di rete che è stato deciso dall'algoritmo di load balancing.
4. Il merge dei risultati che sono stati calcolati dai diversi processori.

L'analisi di tale programma è stata ampiamente discusso nel report del PAR consegnato a Settembre 2017. In tale contesto si è visto come tale architettura aveva diversi punti deboli:

- Mancanza di salvataggio di stato: Se i diversi processori non riescono a terminare l'analisi delle linee di autobus a loro assegnata prima del termine massimo assegnato loro dalla coda, il calcolo effettuato viene perso.
- Complessità nel bilanciamento del carico: Il bilanciamento del carico ha rappresentato uno dei problemi principali. Sono stati implementati ben quattro diverse versioni di bilanciamento che hanno richiesto sforzo mentale e impegno nella scrittura del codice. Nonostante ciò, l'utilizzo delle risorse non è risultato ottimale, seppur si sia raggiunto un bilanciamento del carico accettabile.
- Errori di comunicazione tra master e slaves: La comunicazione tra master e slaves gestita tramite MPI introduce latenze e alcune volte anche errori che bloccavano l'esecuzione del programma prematuramente.
- Scrittura del codice complessa: L'utilizzo di MPI per far comunicare master e slaves e la scrittura di algoritmi di bilanciamento del carico hanno implicato la scrittura di un codice complesso e difficile da leggere.

## *3.2 Nuova struttura del programma e metodologie usate*

Come analizzato nel precedente paragrafo, l'architettura e la struttura del programma realizzato per l'anno 2017 aveva diversi punti deboli. Per tali ragioni si è resa necessaria la progettazione di una nuova architettura, con nuove metodologie che verranno discusse in questa sezione.

### 3.2.1 Metodologie: Job array e C++

Per il nuovo anno di lavoro si è continuato a scrivere codice in C++, ed i motivi sono già stati spiegati nella precedente sezione. Tuttavia, si è eliminato completamente l'utilizzo di MPI per far fronte all'introduzione dei job array [2].

Come descritto nel capitolo 2, nel nostro caso si è dunque di fronte ad un problema in cui si ha un unico programma con diversi input. A tal proposito, una tecnologia in grado di sfruttare al meglio le risorse di CRESCO e che ci permetterebbe di poter risolvere istanze del problema maggiormente significative, è il job array.

Con LSF è possibile sottomettere jobs multipli in un unico run o step, usando un "Job array", ossia un insieme di oggetti dello stesso tipo, con lo stesso nome, identificati da un indice, ma che assumono valori diversi. In particolare, per i jobs di un array la differenza è nell'input al programma sottomesso. Questo approccio è particolarmente utile se i job dell'array condividono lo stesso eseguibile (ad es. uno script o un'applicazione) e tipo di risorsa, ma hanno differenti file di input oppure l'eseguibile può gestire una variabile associata ad un indice.

I job array sono gruppi di job con lo stesso eseguibile e gli stessi requisiti di risorse, ma differenti file di input. I job array possono essere sottomessi, controllati e monitorati come una singola unità o come job individuali o gruppi di job. Infatti, ogni job sottomesso da un job array condivide lo stesso job ID in quanto fa parte dello stesso job array, e al tempo stesso è unicamente referenziato grazie all'uso dell'indice dell'array. La dimensione e la struttura di un job array è definita quando il job array viene creato.

Ciò che migliorerebbe rispetto all'approccio che si è implementato in precedenza, è che LSF schedulerebbe e distribuirebbe i jobs dell'array indipendentemente l'uno dall'altro, in quanto pur condividendo l'ID ogni job è identificato univocamente da un indice. I principali vantaggi dell'utilizzo dei job array sono:

- Non si deve gestire la comunicazione tra i diversi nodi del cluster: Infatti, i nodi non hanno bisogno di comunicare tra di loro, ogni job è gestito da un nodo in maniera separata dal resto dei processori del cluster. Solo una volta terminata l'intera esecuzione dei job presenti nel job array si può

controllare lo stato dell'esecuzione. Ma anche in questo caso non vi è comunicazione tra diversi nodi del cluster.

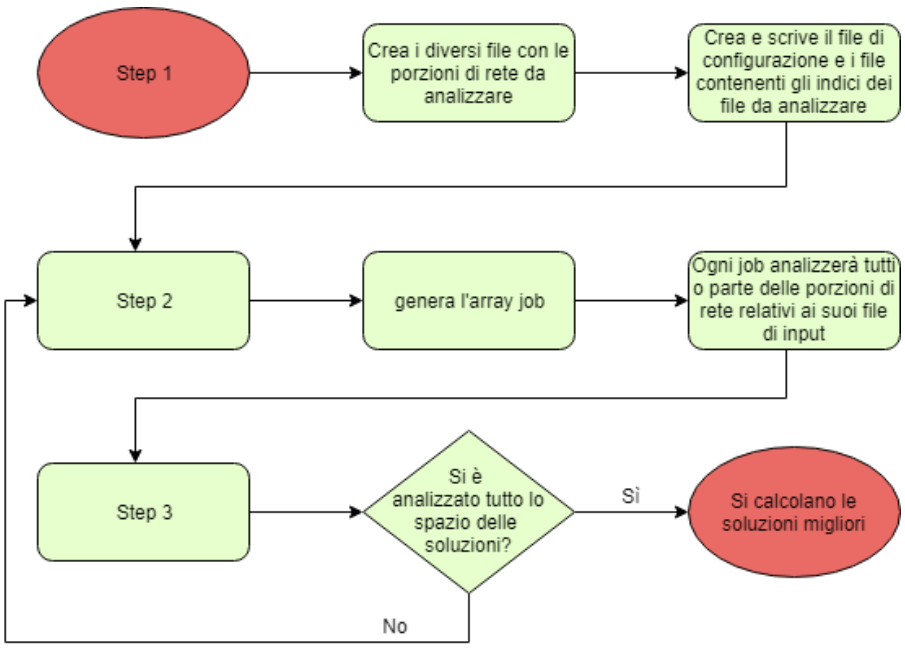
- Non si devono scrivere algoritmi per il bilanciamento del carico: Il bilanciamento del carico viene fatto sulla base dei file di input che diamo in input ad ogni singolo job. Ma la gestione dei nodi e la comunicazione viene affidata al sistema LSF.
- Scrittura del codice semplificata: Con l'utilizzo dei job array si devono scrivere dei semplici programmi che utilizzano C++, senza dover utilizzare MPI o scrivere algoritmi complessi per il bilanciamento del carico.
- Si rende necessario il salvataggio dello stato del lavoro: Tra uno step e l'altro del programma si ha la necessità di salvare il lavoro fatto nello step precedente per poterlo recuperare e analizzare nello step successivo.

Al fine di capire al meglio quest'ultimo punto si approfondirà la struttura del programma nella successiva sottosezione.

### 3.2.2 Struttura del programma

La struttura del nuovo programma è ripartita in 3 step. Per ognuno degli step viene riportata la sequenza dei principali passi che vengono effettuati, insieme ad una immagina che mostra la schermata di esecuzione da terminale del programma ed evidenzia i file e le cartella che va ad utilizzare o a modificare.

**Figura 3.1: Diagramma di flusso che mostra la sequenza delle operazioni che si devono fare per ottenere il risultato finale.**

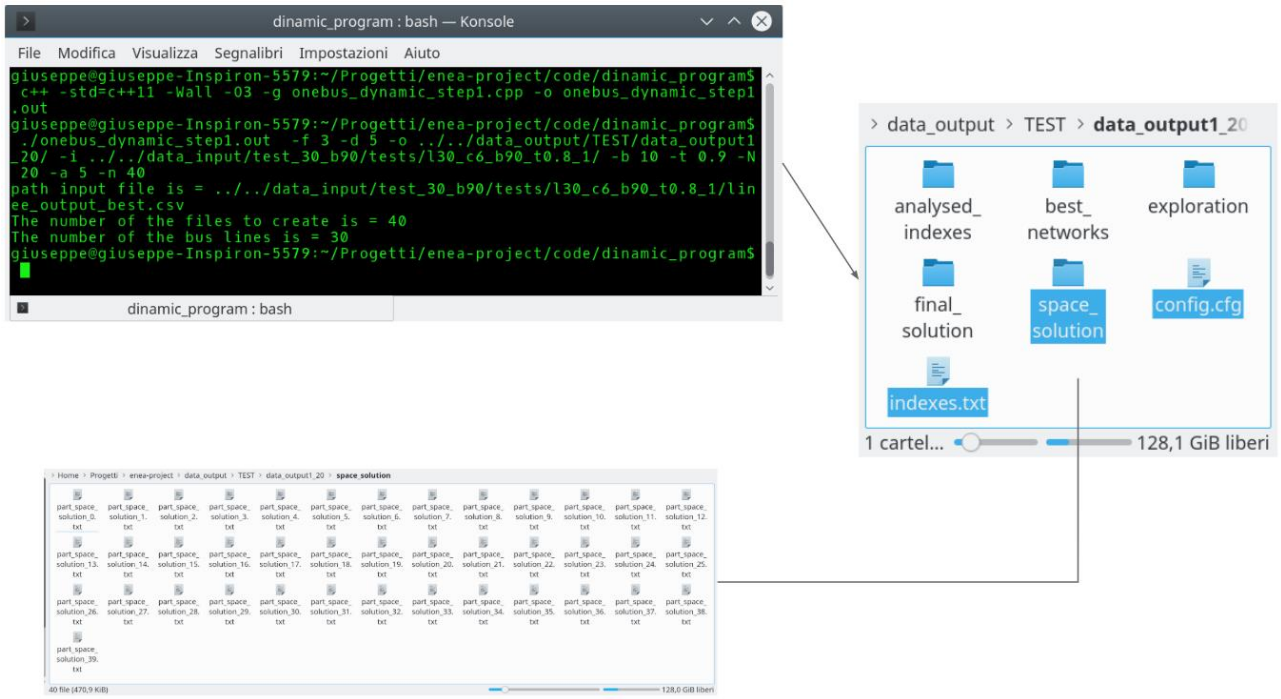


### Primo step

Nel primo step contenuto nel file *onebus_dynamic_step1.cpp* vengono eseguiti i seguenti passi:

a. Controllo dei parametri di input scelti dall'utente: si analizza se tali parametri sono corretti e consistenti, anche per gli step futuri.
b. Creazione delle cartelle necessarie per l'esecuzione di tutti gli step successivi.
c. Creazione di molteplici file in cui vengono definiti porzioni dello spazio delle soluzioni da esplorare. In ogni file ci sarà un insieme di combinazioni di linee che devono essere analizzate. È questo il principale compito dello step 1, ossia generare file diversi che contengono porzioni di rete differenti da analizzare.
d. Creazione di un file di configurazione (`config.cfg`) che conterrà le informazioni base, che riguardano l'esecuzione corrente.
e. Creazione del file `indexes.txt` che contiene la lista degli indici dei file di input che si devono ancora esplorare. Alla prima iterazione tale file conterrà tutti gli indici dei file che sono stati creati al punto c.

**Figura 3.2: Esecuzione da terminale dello step 1. La figura evidenzia inoltre come tale primo step si occupi di creare i file contenenti le porzioni dello spazio delle soluzioni, di scrivere il file di configurazione e il file degli indici dei file da analizzare.**

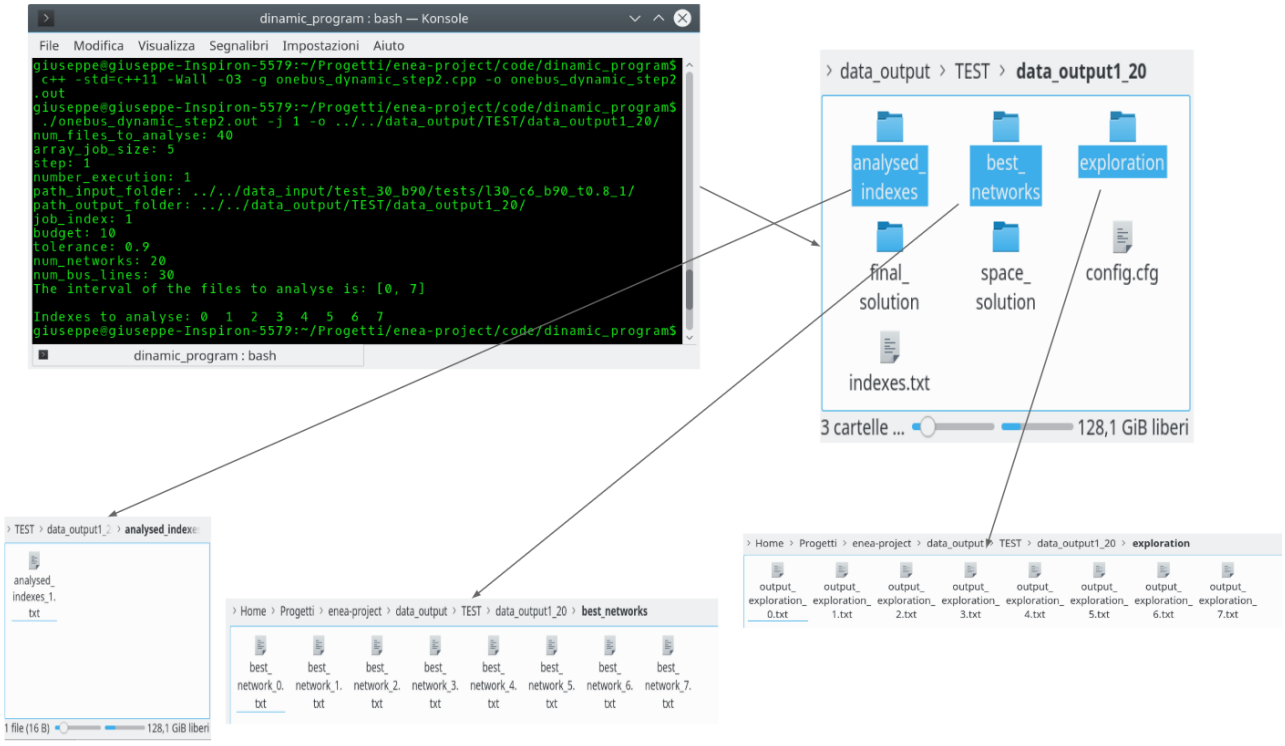


## Secondo step

Nel secondo step contenuto nel file `onebus_dynamic_step2.cpp` vengono eseguiti i seguenti passi:

a. Parser dei parametri passati tramite la riga di comando ed il file di configurazione.
b. Controllo se l'indice del job analizzato è all'interno dell'intervallo della grandezza del job array.
c. Lettura del file `indexes.txt`, dal quale si possono ricavare gli indici dei file che contengono le porzioni di rete ancora da esplorare. Questo file viene preparato dallo step 1, per la prima esecuzione dello step 2. Invece, questo file è preparato dallo step 3, se ci saranno altre esecuzioni dello step 2.
d. Si calcola l'intervallo dei file che deve analizzare il job index. Per fare questo si ha bisogno di conoscere il numero dei file ancora da analizzare e la grandezza del job array.
e. Per ogni file con la porzione dello spazio delle soluzioni, si esplorano le soluzioni, e poi si crea il file con le migliori sottoreti.
f. Salvare in un file (uno per ogni job nell'array) l'indice dei file che sono stati processati interamente con successo

**Figura 3.3: Esecuzione da terminale dello step 2. La figura evidenzia inoltre come il secondo step si occupi di generare i file di output in cui sono memorizzate le migliori sottoreti analizzate da ciascun job e i file dove sono memorizzati gli indici dei file analizzati con successo da ogni singolo job.**




## Terzo step

Nel terzo step contenuto nel file `onebus_dynamic_step3.cpp` vengono eseguiti i seguenti passi:

a. Parser dei parametri passati tramite la riga di comando ed il file di configurazione.
b. Controllo se ci sono alcuni file dello spazio delle soluzioni che ancora non sono stati analizzati.
c. Mantengo un valore intero per il numero di file ancora da analizzare
    a. Se tale numero è 0 significa che tutto lo spazio delle soluzioni della rete è stato analizzato
        i. Dunque, si procede a calcolare la migliore soluzione tra quelle prodotte dai diversi job dell'array.
    b. Altrimenti, se il numero è maggiore di 0, significa che ci sono ancora dei file da analizzare.
        i. Si sceglie la grandezza del job array, in base a quanti file sono rimasti da analizzare.

ii. Si aggiorna il file `indexes.txt` con gli indici dei file da analizzare durante la successiva esecuzione dello step 2.

**Figura 3.4: Esecuzione da terminale dello step 3. La figura evidenzia inoltre come il terzo step si occupi di generare il file di output in cui sono memorizzate le migliori sottoreti di tutta la rete. Inoltre, si occupa di modificare il file di configurazione e quello contenente gli indici.**

```
\---progetto_onebus
    +---code
    |   +---dinamic_program
    |   |   |   onebus_dynamic_step1.cpp
    |   |   |   onebus_dynamic_step2.cpp
    |   |   |   onebus_dynamic_step3.cpp
    |   |   |
    |   |   \---parse_arguments
    |   |           parse_arguments_step1.cpp
    |   |           parse_arguments_step1.hpp
    |   |           parse_arguments_step2.cpp
    |   |           parse_arguments_step2.hpp
    |   |           parse_arguments_step3.cpp
    |   |           parse_arguments_step3.hpp
    |   |
    |   +---libraries
    |   |   +---constants
    |   |   |       constants.cpp
    |   |   |
    |   |   +---exploration
    |   |   |       exploration.cpp
    |   |   |       exploration.hpp
    |   |   |       objective_function.cpp
    |   |   |       objective_function.hpp
    |   |   |
    |   |   +---objects
    |   |   |       arch_b_info.cpp
    |   |   |       arch_b_info.hpp
    |   |   |       arch_c_info.cpp
    |   |   |       arch_c_info.hpp
    |   |   |       electrical_arch.cpp
    |   |   |       electrical_arch.hpp
    |   |   |       line.cpp
    |   |   |       line.hpp
    |   |   |       network.cpp
    |   |   |       network.hpp
    |   |   |       node_b.cpp
    |   |   |       node_b.hpp
    |   |   |       node_c.cpp
    |   |   |       node_c.hpp
    |   |   |       traditional_arch.cpp
    |   |   |       traditional_arch.hpp
    |   |   |
    |   |   +---print
    |   |   |       print_functions.cpp
    |   |   |       print_functions.hpp
    |   |   |       write_functions.cpp
    |   |   |       write_functions.hpp
    |   |   |
    |   |   \---util
    |   |           balancing_help_functions.cpp
    |   |           balancing_help_functions.hpp
    |   |           computation_size.cpp
    |   |           computation_size.hpp
    |   |           file_management.cpp
    |   |           file_management.hpp
    |   |           initialization.cpp
    |   |           initialization.hpp
    |   |           list_network.cpp
    |   |           list_network.hpp
    |   |           split.cpp
    |   |
    |   +---preliminay_analysis
    |   |         recharge_eol_B.cpp
    |   |         recharge_eol_C.cpp
```

```
|   |         recharge_Stops.cpp
|   |
|   \---recharge_lines
|           recharge_lines.cpp
|
+---data_input
\---data_output
```

La nuova struttura del programma ha subito diversi cambiamenti rispetto alla precedente versione. Di seguito si riportano le descrizioni dei nuovi file o delle nuove funzionalità implementate all'interno dei diversi file.

Il progetto è costituito da 3 cartelle principali:

1. `code`: contiene il codice eseguibile con le diverse librerie implementate in C++. In particolare, di seguito verranno descritti i nuovi file che si sono implementati rispetto a quelli già presenti nello scorso anno di attività:
    a. `onebus_dynamic_step1.cpp` `onebus_dynamic_step2.cpp` e `onebus_dynamic_step3.cpp`: Sono i file che contengono l'implementazione dei tre step del programma.
    b. `parse_arguments`: All'interno di questa cartella sono stati aggiunti i file in grado di effettuare il parser degli argomenti dei diversi step.
    c. `objects`: All'interno della cartella objects sono stati aggiunti diversi file che contengono l'implementazione di oggetti volti a memorizzare le informazioni della nuova architettura C che si è considerata e del dimensionamento delle infrastrutture di ricarica dei vari nodi della rete.
    d. `recharge_eol_B.cpp` `recharge_eol_C.cpp` e `recharge_Stops.cpp`: Si tratta dei file che contengono i programmi in grado di analizzare la rete preliminarmente per capire quali siano i nodi che potrebbero necessitare di più di un'infrastruttura di ricarica.
    e. `recharge_lines.cpp`: Si tratta del file che analizza ogni linea singolarmente e che ci comunica quante sono il numero di infrastrutture di ricarica per elettrificare una singola linea.
2. `data_input`: contiene i dati di input del programma. Questa cartella contiene altre cartelle in cui in ognuna vengono riportati i diversi file di input necessari per il corretto svolgimento del programma. Il contenuto all'interno di ogni cartella di input è il seguente:
    a. `corse_ferm_C.csv`: Il numero di corse effettuata da una linea in ogni fermata e in ogni singola fascia oraria.
    b. `linee_output_best.csv`: Le linee contenute dalla rete che si sta analizzando. Ci sono informazioni relative alle diverse architetture di implementazione della linea, le diverse voci di costo e informazioni riguardo l'analisi preliminare effettuato dal programma Best.
    c. `percorsi_fermate.csv`: I percorsi delle linee con i diversi archi fermata attraversati e i rispettivi consumi.
    d. `utiliz_cap_B.csv`: Le utilizzazioni medie per ogni fascia oraria in un nodo capolinea a seconda della linea considerata, quando la linea viene elettrificata con architettura B.
    e. `utiliz_cap_C.csv`: Le utilizzazioni medie per ogni fascia oraria in un nodo capolinea a seconda della linea considerata, quando la linea viene elettrificata con architettura C.
    f. `utiliz_ferm_C.csv`: Le utilizzazioni medie per ogni fascia oraria in un nodo fermata a seconda della linea considerata, quando la linea viene elettrificata con architettura C.
    g. `consumptions.csv`: Si tratta di un file che si è generato per poter rendere la lettura dei consumi tra due fermate per qualsiasi linea. Infatti, tale file ha memorizzato al proprio interno:
        i. identificativo della linea, identificativo del percorso, identificativo della fermata ed il consumo del tratto tra la fermata considerata e quella precedente.

h. `eol_B_more_one.csv`: Si tratta di un file che si è generato per poter memorizzare i nodi capolinea delle linee elettrificabili con l'architettura di ricarica B, che potrebbero necessitare di più di una sola infrastruttura di ricarica.
i. `eol_C_more_one.csv`: Si tratta di un file che si è generato per poter memorizzare i nodi capolinea delle linee elettrificabili con l'architettura di ricarica C, che potrebbero necessitare di più di una sola infrastruttura di ricarica.
j. `stops_more_one.csv`: Si tratta di un file che si è generato per poter memorizzare i nodi fermata delle linee elettrificabili con l'architettura di ricarica C, che potrebbero necessitare di più di una sola infrastruttura di ricarica.
k. `file_recharge_stops_for_line.csv`: Si tratta del file che dice quante infrastrutture di ricarica alle fermate sono necessarie per la singola linea.

3. `data_output`: contiene i dati di output prodotti dal programma. Questa cartella contiene altre cartelle in cui in ognuna vengono riportati i diversi file che servono per la generazione dei file che salvano lo stato di avanzamento del programma, ma anche il file contenente la soluzione finale. Il contenuto all'interno di ogni folder di output è:
   a. `config.cfg`: File di configurazione usato per il passaggio dei diversi parametri tra i vari step del programma.
   b. `Indexes.txt`: File con gli indici dei file contenenti porzioni di rete ancora da analizzare.
   c. `Analysed_indexes`: Cartella contenente m file, con m uguale al numero di array job dell'esecuzione corrente. In questi file ogni job scrive gli indici dei file contenenti le porzioni di rete che ha esaminato con successo.
      i.
      ```
      +---analysed_indexes
      |     analysed_indexes_1.txt
      |     ...
      |     analysed_indexes_m.txt
      ```
   d. `Best_networks`: Cartella contenente i file con le migliori reti analizzate per ogni singola porzione di rete di input.
      i.
      ```
      +---best_networks
      |     best_network_0.txt
      |     ...
      |     best_network_n.txt
      ```
   e. `exploration`: Cartella contenente i file che descrivono il numero di reti assegnate e quelle realmente analizzate.
      i.
      ```
      +---exploration
      |     output_exploration_0.txt
      |     ...
      |     output_exploration_n.txt
      ```
   f. `final_solution`: Cartella contenente i file di output finale. In tale file c'è l'elenco delle migliori reti da elettrificare. Nel nome del file sono riportati anche i dati relativi al budget (`b`) e alla tolleranza (`t`).
      i.
      ```
      +---final_solution
      |    1283_c224_b1000.000000_t0.900000.csv
      ```
   g. `space_solution`: Cartella contenente i file di input generati dal primo step. In ogni file sono riportati una parte dell'intero spazio delle soluzioni.
      i.
      ```
      \---space_solution
            part_space_solution_0.txt
            ...
            part_space_solution_n.txt
      ```

### 3.2.3 Manuale utente

In questa sezione sono riportate le informazioni necessarie per poter sfruttare al meglio il programma realizzato.

Il programma può essere eseguito in locale o direttamente sull'infrastruttura Cresco. Tuttavia, per poter effettuare il run dell'array job si ha bisogno dell'infrastruttura di Cresco.

Una volta che si hanno i file contenuti nella cartella `Onebus`, per effettuare la compilazione del programma si deve accedere alla cartella `dynamic_program` contenuta nella cartella `code` in cui sono presenti i main dei diversi step del progetto. Da questa posizione si può procedere alla *compilazione del programma*, ad esempio nel seguente modo:

```
c++ -std=c++11 -Wall -O3 -g onebus_dynamic_step1.cpp -o onebus_dynamic_step1.out
```

In questo modo avremo ottenuto il file eseguibile `onebus_dynamic_step1.out`.
Allo stesso modo si può trattare lo step 3. Mentre fa eccezione lo step 2 che deve essere eseguito su Cresco e sfruttando la tecnica del job array. A tal proposito si riportano le istruzioni necessarie per poter effettuare l'esecuzione di tale step 2:

- Si compila il file onebus_dynamic_step2.cpp nel seguente modo:
  - `c++ -std=c++11 -Wall -O3 -g onebus_dynamic_step2.cpp -o onebus_dynamic_step2.out`
- Si scrive un file che possiamo denominare `script.sh` il cui contenuto è:
  - `#!/bin/sh`
    `./onebus_dynamic_step2.out -j "$LSB_JOBINDEX" -o ./data_output/`
- Ed infine si può sottomettere un array job con il seguente comando:
  - `bsub -J "array_dynamic[1-5]" -q cresco4_h6 -o "output.%J.%I" -e "error.%J.%I" ./script.sh`

In tal modo si è riusciti ad eseguire un array job di grandezza 5, quindi ci sono 5 job che eseguono lo stesso codice e che si differenziano per l'indice del job. Come si può notare il programma in esecuzione si riesce a rendere conto dell'indice del job grazie alla variabile d'ambiente `"$LSB_JOBINDEX"` che viene passata come parametro al programma all'interno del file `script.sh`.

La variabile d'ambiente deve essere richiamata dentro il `script.sh`, altrimenti non riesce ad essere passata al programma correttamente.

Ogni step può avere i propri parametri con il quale essere eseguito, ed è bene familiarizzare con tali parametri per poter sfruttare al meglio il programma e poter eseguire diverse tipologie di test.

Lo step 1 può avere come parametri:

- **-a** permette di indicare la grandezza del job array.
- **-b** è l'opzione che permette di specificare il *budget* dedicato agli investimenti.
- **-c** permette di indicare il numero di cores in cui dividere calcolo.
- **-d** è l'opzione che permette di specificare il grado di divisione del calcolo. Serve per poter suddividere lo spazio delle soluzioni in diversi file.
- **-f** indica la funzione di bilanciamento scelta per il run:
  - 0 indica il bilanciamento equo di linee.
  - 1 indica il bilanciamento firsLessLastMore.
  - 2 indica il bilanciamento fisso in profondità.
  - 3 indica il bilanciamento variabile in profondità.
- **-h** per avere in stampa tutte le indicazioni sull'uso dei parametri.
- **-i** per indicare il file di input principale. Si tratta del file contenente le informazioni delle linee che sono state estrapolate tramite il programma BEST (linee_output_best.csv).
- **-n** per specificare il numero di file in cui si deve suddividere le diverse porzioni dello spazio delle soluzioni.
- **-N** per specificare il numero di reti migliori che si devono restituire in output.
- **-o** per specificare la cartella in cui si andranno a memorizzare tutti i file di output generati dal programma.
- **-p** permette di abilitare le stampe che riguardano l'esplorazione della rete.

- **-r** per specificare il file dei percorsi che si deve considerare (percorsi.csv).
- **-t** per indicare la tolleranza usata per specificare il vincolo di budget.

Lo step 2 può avere come parametri solamente i seguenti:

- **-j** che permette di specificare l'indice del job array che si sta considerando.
- **-h** per avere in stampa tutte le indicazioni sull'uso dei parametri.
- **-o** per specificare la cartella in cui si andranno a memorizzare tutti i file di output generati dal programma.

Lo step 3 può avere come parametri solamente i seguenti:

- **-h** per avere in stampa tutte le indicazioni sull'uso dei parametri.
- **-o** per specificare la cartella in cui si andranno a memorizzare tutti i file di output generati dal programma.

Il motivo per cui gli step 2 e 3 non hanno bisogno di altri parametri è perché la maggior parte dei parametri li leggono nel file di configurazione che viene scritto dopo l'esecuzione del primo step.

Se l'esecuzione del programma avviene con successo, nella cartella di output `final_solution` ci si ritroverà con un file .csv che contiene il risultato con l'elenco delle migliori sottoreti da elettrificare.

Esempio: *l30_c10_b90.000000_t0.800000.csv* è il file dove viene memorizzato l'output del test fatto su di una rete con 30 linee, 10 capolinea, budget 90 e tolleranza pari a 0.8. Quando si apre il file si ha la seguente schermata:

**Figura 3.5: In questa figura viene riportato l'esempio di un file di output in cui vengono specificate le migliori sottoreti che si possono elettrificare.**

| obj_function_value | number_of_lines | line_id | arch_id | … | CapB | … | CapC | … | Stops | … | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 802969.06600000 | 2 | 80 | A | 196 | A | CapB | CapC | Stops | | | | | | | | | | | | | | |
| 777426.81400000 | 1 | 196 | A | CapB | CapC | Stops | | | | | | | | | | | | | | | | |
| 684920.12900000 | 2 | 80 | A | 196 | B | CapB | 8172 | 1 | 8190 | 2 | CapC | Stops | | | | | | | | | | |
| 659377.87700000 | 1 | 196 | B | CapB | 8172 | 1 | 8190 | 2 | CapC | Stops | | | | | | | | | | | | |
| 260178.93810000 | 2 | 80 | B | 196 | A | CapB | 8114 | 1 | 8261 | 1 | CapC | Stops | | | | | | | | | | |
| 142130.00110000 | 2 | 80 | B | 196 | B | CapB | 8114 | 1 | 8172 | 1 | 8261 | 1 | 8190 | 2 | CapC | Stops | | | | | | |
| -2586891.5500666 | 2 | 80 | C | 196 | A | CapB | CapC | 8114 | 1 | 8261 | 1 | Stops | 89 | 1 | 401 | 1 | 421 | 1 | 745 | 1 | | |
| -2704940.4870666 | 2 | 80 | C | 196 | B | CapB | 8172 | 1 | 8190 | 2 | CapC | 8114 | 1 | 8261 | 1 | Stops | 89 | 1 | 401 | 1 | | |
| -3072644.5720000 | 2 | 80 | A | 196 | C | CapB | CapC | 8172 | 1 | 8190 | 1 | Stops | 162 | 1 | 186 | 1 | 215 | 1 | 220 | 1 | | |
| -3098186.8240000 | 1 | 196 | C | CapB | CapC | 8172 | 1 | 8190 | 1 | Stops | 162 | 1 | 186 | 1 | 215 | 1 | 220 | 1 | 222 | 1 | | |
| -3364318.3640666 | 1 | 80 | C | CapB | CapC | 8114 | 1 | 8261 | 1 | Stops | 89 | 1 | 401 | 1 | 421 | 1 | 745 | 1 | 982 | 1 | | |
| -3615434.6999000 | 2 | 80 | B | 196 | C | CapB | 8114 | 1 | 8261 | 1 | CapC | 8172 | 1 | 8190 | 1 | Stops | 162 | 1 | 186 | 1 | | |

In tale file vengono riportate le 15 migliori soluzioni possibili e sono specificati:

- Valore della funzione obiettivo.
- Grandezza della sottorete da elettrificare.
- Elenco di id linea e architettura con cui specificare quella linea.
- Elenco di id capolinea e relativo numero di infrastrutture per quanto riguarda i nodi in cui si ricaricano gli autobus elettrificati con l'architettura B.
- Elenco di id capolinea e relativo numero di infrastrutture per quanto riguarda i nodi in cui si ricaricano gli autobus elettrificati con l'architettura C.
- Elenco di id fermata e relativo numero di infrastrutture per quanto riguarda i nodi in cui si ricaricano gli autobus elettrificati con l'architettura C.

# 4 Analisi architettura C e dimensionamento della ricarica ai nodi

L'analisi delle linee elettrificabili con la tipologia di ricarica C, ossia con ricarica sia al deposito, al capolinea che alle fermate, ha richiesto un'analisi approfondita. Calcolare la soluzione esatta al nostro problema di ottimizzazione per quanto riguarda le linee che possono ricaricarsi alle fermate è un problema molto complicato.

Per poter calcolare tale soluzione ottima per ogni sottorete analizzata si dovrebbe:

- Effettuare il dimensionamento della ricarica per ogni nodo capolinea e fermata contenuti nella sottorete analizzata. Questo tipo di calcolo ci permette di capire quante stazioni di ricarica si devono installare in un nodo, nel caso in cui si sceglierà di elettrificare tale nodo.
- Scegliere quali nodi fermata elettrificare, facendo in modo che tali costi spesi per questi nodi rappresenti la spesa minima attuabile per garantire il continuo servizio agli autobus della sottorete considerata. Per ottenere la soluzione ottima, si potrebbe procedere attraverso l'enumerazione delle fermate. Con tale metodo, nell'analisi di una rete con N nodi fermata, si andrebbe ad effettuare un numero di prove massimo pari a:

$$\sum_{k=1}^{N} \binom{N}{k} \qquad \text{(eq. 4.1)}$$

Pensare di dover effettuare un calcolo di questa portata per ogni sottorete analizzata sarebbe davvero molto oneroso. È per tale motivo che si è pensato a diverse tecniche di pre-processing.

## *4.1 Numero massimo di postazioni di ricarica alla stessa fermata*

La tecnica che verrà esposta di seguito riguarda l'analisi del numero di postazioni di ricarica alla stessa fermata.
Se infatti, non dovessero esistere nodi per cui sono necessarie più di una postazione di ricarica, o sono relativamente pochi, si procederebbe a fare il dimensionamento dei nodi solo per i pochi nodi per cui è necessario, e non per tutti, evitando una grande quantità di calcolo al nostro programma.

L'analisi di una singola fermata *x* della rete prevede i seguenti passi:

- Si calcola il numero di linee *num_lines* che passano per la fermata *x.*
- Si calcola il numero di corse complessivo nell'ora di punta *peak_bus_rides* assoluta fatte dalle linee.
- In base a *num_lines* e a *peak_bus_rides* riusciamo a capire il numero di postazioni di ricarica che sono necessarie per quel nodo *x* grazie alla seguente matrice:

**Tabella 4.1: La tabella mostra quante postazioni di ricarica sono necessarie sulla base del numero di linee e del numero di corse che passano per quel nodo nell'ora di punta.**

| | **2 linee** | **3 linee** | **4 linee** | **5 linee** | **> 5 linee** |
|---|---|---|---|---|---|
| **N. di postazioni di ricarica** | **Numero di corse nell'ora di punta assoluta** | | | | |
| **1** | (0;46] | (0;41] | (0;37] | (0;34] | (0;31] |
| **2** | [47;81] | [42;72] | [38;65] | [35;59] | [32;54] |
| **3** | [82;114] | [73;101] | [66;91] | [60;83] | [55;76] |
| **4** | [115;123] | [102;110] | [92;99] | [84;90] | [77;83] |
| **5** | [124;128] | [111;114] | [100;103] | [91;94] | [84;86] |

Tale algoritmo è stato implementato tramite un programma scritto in c++ (`recharge_stops.cpp`) e contenuto nella cartella `preliminary_analysis`. Utilizzando i dati di input contenuti nel file `percorsi_fermate_C.csv`, è emerso che nella rete completa di Roma:

- Il numero massimo di stazioni di ricarica da installare in una singola fermata è 3.
- Ci sono 3 nodi che al massimo necessiterebbero di 3 stazioni di ricarica.
- Ci sono 16 nodi che al massimo necessiterebbero di 2 stazioni di ricarica.
- Tutti gli altri nodi fermata, necessitano al più di 1 stazione di ricarica.

Tali risultati ci consentono di capire che per la maggior parte dei nodi non è necessario effettuare il dimensionamento del nodo, poiché sappiamo già che qualunque sia la rete sicuramente in quel nodo sarà necessario installare solo un unico impianto di ricarica.

Diversamente nel caso dei 19 nodi che possono richiedere più di una stazione di ricarica, si deve procedere al dimensionamento dei nodi, per capire quante stazioni di ricariche installare sulla base della sottorete analizzata.

## *4.2 Numero massimo di postazioni di ricarica ai capolinea*

La tecnica che verrà esposta di seguito riguarda l'analisi del numero di postazioni di ricarica allo stesso capolinea.
Se infatti, non dovessero esistere nodi per cui sono necessarie più di una postazione di ricarica, o sono relativamente pochi, si procederebbe a fare il dimensionamento solo per questi nodi, e non per tutti, evitando una grande quantità di calcolo al nostro programma.

L'analisi di un singolo capolinea *x* della rete prevede i seguenti passi:

- Si calcola l'utilizzazione massima relativa ad un nodo capolinea nell'ora di punta.
- Si divide tale utilizzazione massima per il valore del massimo consumo consentito per singola infrastruttura di ricarica.
- Si prende la parte intera superiore del numero ottenuto,

In tale analisi sopradescritta, per ogni capolinea viene considerato il caso in cui si decida di elettrificare tutte le linee che transitano nel nodo x. Inoltre, ogni linea effettua un numero di corse variabile durante l'arco della giornata e per tale motivo si deve considerare il più grande valore ottenuto dalla somma delle utilizzazioni prodotte nelle diverse fasce orarie considerate.

Per comprendere tale algoritmo viene riportata la struttura dei dati di input:

**Tabella 4.2: La tabella mostra le informazioni riguardo l'utilizzazione nelle 24 fasce orarie della linea "Id Linea" che si ricarica al capolinea "Id nodo".**

| Id nodo | Id linea | 0 | 1 | 2 | 3 | 4 | ..... | 19 | 20 | 21 | 22 | 23 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 1 | 46 | 0.3 | 0.3 | 0.2 | 0.4 | 0.1 | ..... | 0.6 | 0.9 | 0.9 | 0.6 | 0.3 |
| 1 | 49 | 0.5 | 0.6 | 0.5 | 0.8 | 0.7 | ..... | 0.2 | 0.3 | 0.7 | 0.5 | 0.4 |
| … | … | … | … | … | … | … | ..... | … | … | … | … | … |

Come si può evincere da tale esempio la fascia oraria con utilizzazione maggiore è la numero ventuno, in cui la somma delle utilizzazioni del nodo "1" raggiunge il valore di "1,6" (ottenuto da 0,9+0,7).
Tale algoritmo è stato implementato tramite due programmi scritti in c++ (`recharge_eol_B.cpp` e `recharge_eol_C.cpp`) e contenuti nella cartella `preliminary_analysis`.
Questo tipo di analisi è stato effettuato sia per i capolinea delle linee elettrificabili con architettura B, e sia per i capolinea elettrificabili con architettura C. Nel caso di Roma i risultati che si sono ottenuti sono i seguenti:

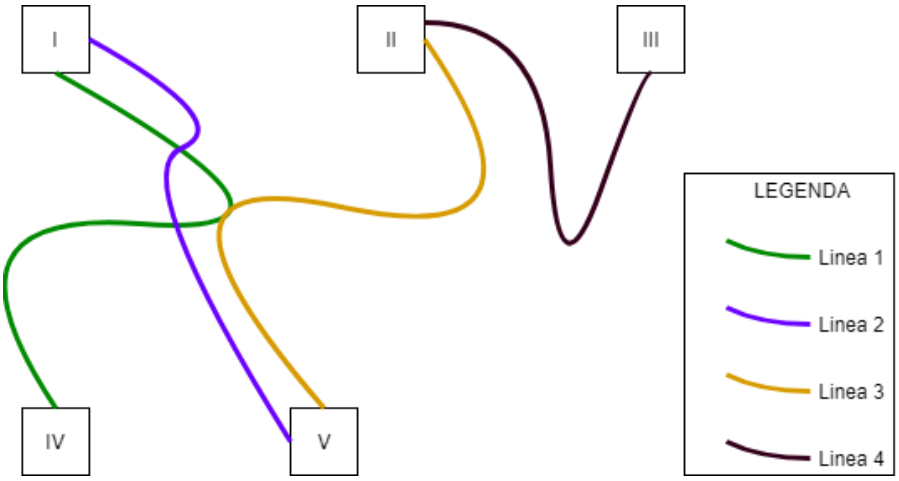
- 74 capolinea elettrificati con architettura B necessitano di più di una stazione di ricarica. Fino a 11 stazioni di ricarica ad un capolinea.
- 29 capolinea elettrificati con architettura C necessitano di più di una stazione di ricarica. Fino a 6 stazioni di ricarica ad un capolinea.

## *4.3 Numero minimo di postazioni da ricaricare per ogni singola linea*

La seconda tecnica che verrà esposta riguarda il calcolo del numero minimo di postazioni di ricarica in ogni singola linea. Questo tipo di calcolo fatto a priori ci consente di risparmiare tempo macchina prezioso.

Infatti, grazie a tale calcolo è possibile sapere a priori quali e quante fermate elettrificare in una linea, nel momento in cui tale linea non condivide fermate con altre linee della sottorete considerata. Immaginiamo il caso in cui stiamo considerando di elettrificare con architettura C la sottorete composta dalle linee 1, 2, 3 e 4 (cfr. Figura 4.1). E che in tale sottorete, la linea 4 sia completamente separata dalle altre tre. In tal caso, si ha che la linea 4 non condivide fermate con nessuna delle altre linee presenti nella sottorete considerata. Per tale motivo, per la linea 4 è possibile sfruttare l'analisi a priori, che ci dice quante e quali fermate elettrificare.

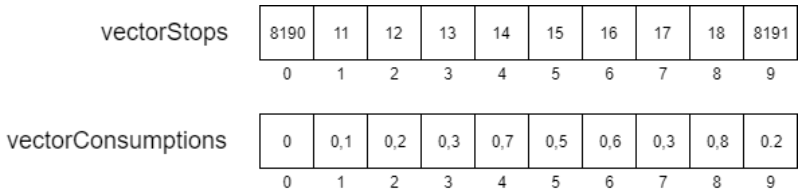
**Figura 4.1: Sottorete con quattro linee, in cui tre linee condividono almeno una fermata, mentre la linea 4 non condivide alcuna fermata con le altre. Per tale linea si possono utilizzare i risultati ottenuti per singola linea.**




L'analisi di una singola linea *x* della rete prevede i seguenti passi:

- Verifica della fattibilità della linea per l'elettrificazione con l'architettura C. Se tale linea non è elettrificabile, non si effettua l'analisi della stessa.
- Costruzione dei due vettori contenenti i dati:
  - vectorStops: Il vettore contenente tutti i nodi (capolinea e fermata) attraversati dalla linea considerata.
  - vectorConsumptions: Il vettore contenente tutti i consumi tra due nodi consecutivi della linea.
- Costruzione dell'insieme di fermate che si ripetono all'interno della stessa linea.
  - setRepeatedStops: Si tratta delle fermate che sono presenti più volte all'interno del vettore vectorStops.
- Ricerca della soluzione ottima per la linea considerata.

**Figura 4.2: In figura sono rappresentati i due vettori utilizzati nel calcolo delle fermate da elettrificare. vectorStops contiene la sequenza dei nodi attraversati da un autobus mentre percorre il percorso della linea. vectorConsumptions contiene l'insieme dei consumi.**




Dalla figura 4.2 si può vedere come siano riportati i dati all'interno dei due vettori. In vectorStops c'è il capolinea 8190 seguito dalle fermate {11, 12, 13, 14, 15, 16, 17, 18} e infine c'è il capolinea 8191. In vectorConsumptions, il valore contenuto in posizione i-esima ci fornisce il valore (espresso in kwh) del consumo medio tra il nodo i e il nodo i-1. Ad esempio:

- il consumo tra il capolinea 8190 e la fermata11 è 0,1.
- Il consumo tra la fermata 12 e la fermata 11 è 0,2.

La ricerca della soluzione ottima per la singola linea ha richiesto l'implementazione di un algoritmo esatto in grado di restituire il numero minimo di postazioni di ricarica da installare.

Per poter comprendere appieno l'algoritmo si può cominciare con il descrivere la procedura utilizzata per le linee che non hanno fermate che si ripetono. Tale algoritmo è greedy, in quanto analizza in ordine i consumi degli archi fermata, e non installa architetture di ricarica fino a che non si sia violato il vincolo di consumo. Infatti, un autobus può erogare una quantità di energia pari al massimo ad 1,73Kwh, senza effettuare una ricarica della batteria.

Dunque, l'algoritmo greedy analizza iterativamente i vettori delle fermate e quello dei consumi:

- Per prima cosa si analizza il consumo dell'arco fermata che si sta considerando, così da calcolare l'energia ancora spendibile dal bus, dopo aver attraversato tale tratto.
- Se l'energia con cui arriva il bus alla fermata i-esima è inferiore a 0, significa che nella fermata i-1 si deve installare una stazione di ricarica.
- Inoltre, si controlla il vettore delle fermate (vectorStops):
    - Se la fermata analizzata corrisponde al nodo di un capolinea, presupponiamo che l'autobus si sia ricaricato e che l'energia spendibile dall'autobus sia pari al massimo previsto.

Per comprendere proviamo a descrivere l'algoritmo sulla base della figura 4.2. La soluzione trovata per questo caso di esempio è quella che va ad elettrificare i nodi 14 e 17. Infatti, l'autobus che inizia il percorso nel nodo capolinea 8190 inizierà il percorso completamente carico (perché i capolinea si immaginano tutti elettrificati). Arrivati alla fermata 14 l'autobus in media avrà speso una quantità di energia pari a:

$$(0{,}1 + 0{,}2 + 0{,}3 + 0{,}7)\,\text{kwh} = 1{,}3\text{kwh}$$

La ricarica al nodo 14 si rende necessaria, perché per fare il tratto dalla fermata 14 alla fermata 15 si dovrebbero spendere altri 0,5kwh andando a superare il limite massimo di consumo medio consentito pari a 1,73kwh.

Per un ragionamento del tutto analogo si rende necessaria la ricarica al nodo 17.

Tale algoritmo è alla base di quello utilizzato per il caso in cui in una linea si hanno fermate ripetute. Nel momento in cui in una linea ci sono delle fermate ripetute, il semplice algoritmo greedy non è più sufficiente a garantirci l'esattezza della soluzione. Infatti, potrebbe capitare che la soluzione greedy ci dia una soluzione che sembra essere ottima, ma in realtà essendoci fermate che si ripetono lungo la linea potrebbe non esserlo.

Per tale motivo una considerazione iniziale era stata quella di provare ad analizzare tutte le combinazioni delle fermate di una linea, così da ottenere un risultato esatto. Tuttavia, ci si può rendere facilmente conto che analizzare tutte le combinazioni di tutte le fermate di una singola linea impiegherebbe un tempo macchina davvero molto grande. Infatti, il numero di combinazioni da testare in una linea con F fermate è:

$$\sum_{k=1}^{F} \binom{F}{k} \qquad (eq.\,4.2)$$

Pensare di poter eseguire una tale quantità di calcolo per ogni singola linea, e poi per ogni sottorete contenente linee che possono essere elettrificate con architettura C è impensabile, nonostante la grande infrastruttura di calcolo usata.

È per tale motivo che per ottenere una soluzione esatta ed effettuare una mole di calcolo considerevolmente minore si è pensato di analizzare nel dettaglio i nodi fermata ripetuti. A tal proposito l'algoritmo greedy viene ripetuto per diverse configurazioni della linea. Immaginiamo di avere R fermate ripetute all'interno della linea considerata:

- Per prima cosa si manda in esecuzione l'algoritmo greedy e si calcola il numero di fermate che verrebbero elettrificate.
- Successivamente si analizzano le configurazioni in cui a priori si prova ad elettrificare alcune fermate tra le R ripetute, fino a che non si sono prese in considerazione tutte le possibili combinazioni di fermate ripetute elettrificate a priori.

In questo modo il numero di casi che si devono analizzare è sicuramente minore od uguale al numero di casi descritto dall'equazione 4.2. Infatti, se R è il numero di fermate ripetute all'interno di una linea (o di una sottorete) e F è il numero di fermate complessive della linea (o della sottorete) si ha che:

$$\sum_{k=1}^{F} \binom{F}{k} \geq \sum_{k=1}^{R} \binom{R}{k} \qquad (eq.\,4.3)$$

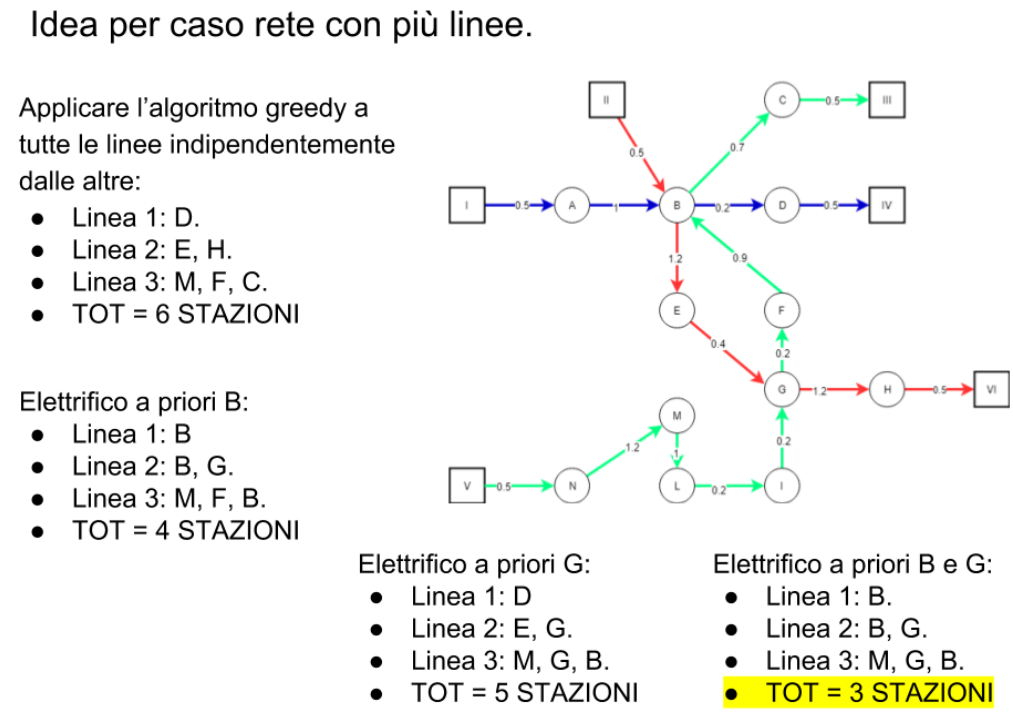
Inoltre, ci si aspetta che il numero di fermate che si ripetono rispetto al numero complessivo di fermate di una sottorete sia notevolmente più basso del numero di complessivo, ossia che R << F. Questa ipotesi è stata confermata dall'analisi di singola linea che ha rilevato come all'interno di una linea ci sono al massimo 12 fermate ripetute in confronto alle 71 complessive e come la maggiorparte delle linee elettrificabili con l'architettura C non abbiano percorsi in cui ci siano delle fermate ripetute.

Diverso è il caso in cui si vanno ad analizzare delle sottoreti, in cui il numero di fermate ripetute e quindi condivise tra diverse linee aumenterà considerevolmente.

## *4.4 Numero minimo di postazioni da ricaricare per una sottorete*

L'approccio usato per analizzare una singola linea è alla base del caso con N linee, in cui si andranno a testare le diverse configurazioni con le fermate ripetute in comune elettrificate a priori. Per comprendere meglio l'analisi che viene effettuata alla sottorete contenente le linee da elettrificare con l'architettura C viene riportato il seguente esempio (cfr. figura 4.3):

**Figura 4.3: Immagine che mostra l'applicazione dell'algoritmo di scelta delle fermate in una rete di 3 linee (linea1 = blu, linea2 = rossa, linea3 = verde) in cui ci sono complessivamente 2 fermate ripetute (la fermata B e G).**




Come si può notare in figura si sono analizzati i seguenti casi:

1) Caso in cui non si elettrificano a priori fermate ripetute.
   - L'algoritmo ci restituisce che debbono essere installate 6 stazioni di ricarica.
2) Caso in cui si elettrifica a priori la fermata ripetuta B.
   - L'algoritmo restituisce un totale di 4 stazioni di ricarica complessive. Il numero di infrastrutture è diminuito rispetto al caso precedente. Tale miglioramento è legato al fatto che tutte le tre linee condividono questo nodo fermata.
3) Caso in cui si elettrifica a priori la fermata ripetuta G.
   - L'algoritmo restituisce un totale di 5 stazioni di ricarica complessive. Il numero di infrastrutture è aumentato rispetto al caso precedente, ma migliorato rispetto al caso in cui non si elettrificano a priori fermate ripetute.
4) Caso in cui si elettrificano a priori le fermate B e G.
   - L'algoritmo restituisce un totale di 3 stazioni di ricarica complessive. Come si può notare il numero di infrastrutture è diminuito rispetto a tutti i casi precedenti.

Analizzando questi quattro casi si nota come la soluzione migliore sia quella che prevede l'elettrificazione di 3 stazioni, e corrisponde al caso 4).

Con questo algoritmo si è potuto ottenere un risultato ancora esatto, ma in grado di non far aumentare eccessivamente i tempi di analisi anche per sottoreti che non hanno molte fermate ripetute.

## *4.5 Nuove strutture dati*

Per poter effettuare l'analisi dell'architettura C e il dimensionamento della ricarica ai nodi capolinea e fermata si è resa necessaria l'introduzione di nuove strutture dati in grado di memorizzare i nuovi dati.

In particolar modo si è modificato l'oggetto linea a cui sono stati aggiunti due oggetti:

- `ArchInfoB`: oggetto contenente le informazioni relative al dimensionamento dei nodi capolinea delle linee elettrificate con l'architettura B.
- `ArchInfoC`: oggetto contenente le informazioni relative al dimensionamento dei nodi capolinea e dei nodi fermata delle linee elettrificate con l'architettura C.

Nella struttura dati `ArchInfoB` vengono memorizzati i nodi capolinea associati alla linea considerata, ed inoltre vengono memorizzati le utilizzazioni al nodo della linea per le diverse fasce orarie. Grazie a queste informazioni è poi possibile effettuare il dimensionamento dei nodi capolinea. Da notare è che si memorizzano solo le utilizzazioni dei capolinea che necessitano realmente del dimensionamento al nodo. Mentre non si memorizzano le utilizzazioni di quei capolinea che a priori si sa non avranno bisogno di più di una infrastruttura di ricarica.

Nella struttura dati `ArchCInfo` vengono memorizzati le informazioni dei capolinea e delle fermate. Per i capolinea la struttura dati è molto simile a quella descritta per `ArchBInfo`, mentre per i nodi fermata si sono aggiunte altre informazioni. In particolare, per le fermate che potrebbero necessitare di più di una singola infrastruttura di ricarica al nodo, vengono memorizzate il numero di corse per le diverse fasce orarie che una linea compie per ogni fermata. Inoltre, si memorizzano i due vettori descritti nel paragrafo 4.3, con la sequenza delle fermate e la sequenza dei consumi (vectorStops e vectorConsumptions).

## *4.6 Calcolo del fattore di riduzione*

Nel calcolo del vincolo riguardante il budget destinato all'investimento iniziale e nel calcolo della funzione obiettivo entra in gioco il fattore di riduzione. Quest'ultimo va a calcolare quanto si devono scontare i costi che si hanno in input, che sono costi riguardanti la linea e che non tengono conto di un possibile effetto rete.

Infatti, quando si considerano più linee insieme vi è la possibilità che tali linee condividano capolinea o fermate, e di conseguenza possono utilizzare la stessa infrastruttura di ricarica al nodo in comune. Ciò implica che i costi calcolati in precedenza da Best debbano essere scontati.

Per poter calcolare il fattore di sconto viene calcolato:

- *nodi_best*: Il numero di nodi che Best aveva previsto di elettrificare.
- *nodi_reali*: Il numero di nodi che realmente si ha bisogno nella sottorete analizzata.

Ne segue che il fattore di riduzione per una tipologia di nodo è data da:

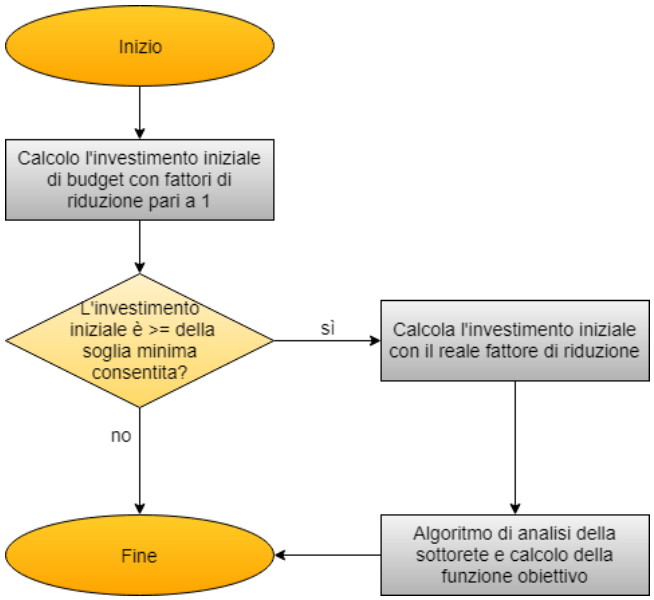
$$fattRiduz = \frac{nodiBest}{nodiReali} \qquad \text{(eq. 4.4)}$$

Per l'analisi del nostro problema si sono calcolati tre diversi tipi di fattore di riduzione:

1. Fattore di riduzione ai nodi capolinea elettrificati con architettura B.
2. Fattore di riduzione ai nodi capolinea elettrificati con architettura C.
3. Fattore di riduzione ai nodi fermata elettrificati con architettura C.

Il calcolo di questo fattore di riduzione per ogni sottorete introduce un aumento del tempo di calcolo considerevole. É per tale motivo che si è cercato di evitare tale calcolo se non nei casi dove era strettamente necessario con l'algoritmo descritto in figura 4.4.

**Figura 4.4: Immagine che mostra il modo in cui si è riusciti ad evitare il calcolo del fattore di riduzione nei casi in cui non era necessario.**




Al fine di evitare tale calcolo si è scelto di implementare un calcolo del vincolo di budget preliminare in cui tutti i fattori di riduzione sono pari a 1. In tal caso si va ad analizzare il caso in cui i costi di linea non subiscano riduzioni, ossia si effettua l'analisi in cui non venga considerato un eventuale effetto rete.

Dunque, con tutti i fattori di riduzione posti al valore massimo (ossia pari a 1), si calcola preliminarmente il costo di investimento iniziale per la sottorete considerata. Se con tale costo il vincolo di budget viene soddisfatto oppure si è nella situazione che i costi di investimento iniziale superano il budget previsti, allora si procederà al calcolo dei veri fattori di riduzione. Altrimenti si è nel caso in cui il costo di investimento iniziale è già al di sotto della spesa minima da effettuare, e dunque andare a calcolare il reale fattore di riduzione che al più introdurrebbe uno sconto ulteriore, non avrebbe senso.

Infatti, con il calcolo del reale fattore di riduzione si andrebbe ad ottenere un valore minore o uguale al precedente, che starebbe sempre sotto il minimo consentito. Per cui si può evitare di calcolare tale fattore di riduzione. Questo introduce un notevole miglioramento, specialmente per i casi in cui il budget è alto e lo è anche il valore della tolleranza.

Per comprendere come il valore di tolleranza possa incidere nel taglio del calcolo del fattore di riduzione si riporta la formula del calcolo del vincolo di budget:

$$tol \cdot Budget \leq costoInvestimentoIniziale \leq Budget$$

La tolleranza può essere compresa nell'intervallo [0, 1]. Se la tolleranza è vicina allo 0, il taglio del calcolo del fattore di riduzione viene effettuato per poche sottoreti. Al contrario se la tolleranza è vicina ad 1 e il budget è abbastanza grande, l'introduzione di questo calcolo preliminare sul calcolo del costo di investimento iniziale della rete porta notevoli benefici, come confermeranno i test riportati nel paragrafo successivo.

# 5 Test per il confronto con l'euristica

Dalle trattazioni effettuate nei precedenti capitoli, si può comprendere come in fase di test degli algoritmi entrino in gioco diversi fattori e molteplici variabili che possono far allungare o diminuire i tempi di esecuzione dell'algoritmo.
Infatti, il tempo di un run dipende da molteplici fattori:

- L'istanza del problema, ossia il numero di linee degli autobus della rete considerata, il budget che si ha a disposizione, la tolleranza che si è disposti ad accettare sul costo degli investimenti.
- La distribuzione dei valori dei costi di investimento, in quanto incidono sul vincolo di budget.
- La particolarità della rete considerata che può avere più o meno fermate che si ripetono.
- Il valore della tolleranza nel vincolo del budget.
- La granularità dei file di input che contengono le diverse sottoreti da analizzare.

Nelle successive sezioni verranno mostrati test che permettono di capire come tali parametri incidano sull'esecuzione del programma. Per ogni test fatto sono riportati i seguenti elementi:

- Una breve descrizione del test.
- i dati in formato tabulare.
- Commenti al risultato ottenuto.

## *5.1 Descrizione del dataset analizzato*

Per questo anno di ricerca ci si è voluti concentrare ad analizzare la rete di Roma e per tale motivo i test effettuati saranno tutti su sottoreti della rete di Roma.

Per quanto riguarda l'analisi di sottoreti dell'istanza di Roma si è scelto di analizzare due zone di notevole interesse:

- `Termini`, ossia le linee elettrificabili con l'architettura C che transitano per il nodo principale di Termini. Si tratta di una rete con 6 linee
- `Corso`, ossia le linee elettrificabili con l'architettura C che transitano per il nodo principale di via Del Corso. Si tratta di una rete con 13 linee.

Della rete di via Del Corso sono stati estratti i seguenti casi:

- `Corso_ABC`, che identifica le linee di via del Corso che risultano elettrificabili con tutte le 3 architetture possibili. Si tratta di una rete con 2 linee.
- `Corso_C`, che identifica le linee di via del Corso che risultano elettrificabili con l'architettura C. Si tratta di una rete con 7 linee.

L'istanza finale di Roma che si è cercato di analizzare ha 283 linee. Si sono voluti considerare anche i seguenti sotto casi:

- `Roma_ABC`, che identifica tutte le linee di Roma che sono elettrificabili con tutte le 3 architetture. Si tratta di una rete con 34 linee.
- `Roma_AB-C_grande`, che identifica tutte le linee di Roma che sono elettrificabili con l'architettura A e B, ma non con quella C. Si tratta di una rete con 99 linee.
- `Roma_AB-C_piccola`, si tratta di una sottorete della rete precedente che ha 34 linee. Si è considerata tale istanza per poterla confrontare con quella di `Roma_ABC`.

## *5.2 Risultati del caso di studio di Roma-Termini*

Il primo test significativo che si intende analizzare è quello che si è eseguito sulla rete di Roma-Termini. Ovvero si è analizzata la rete di Termini al variare del valore del budget per gli investimenti iniziali.

L'obiettivo di questo test è stato quello di poter analizzare il modo in cui variano i tempi di esecuzione, il numero di linee elettrificate nel caso della sottorete migliore, il valore della migliore funzione obiettivo e la tipologia delle architetture di ricarica scelte per la sottorete migliore.

**Tabella 5.1: Test sulla rete di Roma-Termini. Analisi della migliore soluzione con tolleranza pari a 0.9.**

| Analisi della migliore soluzione con Tol. 0.9 | | | | | |
|---|---|---|---|---|---|
| Budget (€) | Numero linee | linee A | linee B | linee C | F.O. (€) |
| 3000000 | 1 | 1 | 0 | 0 | 37843 |
| 5000000 | 2 | 1 | 1 | 0 | 170519 |
| 10000000 | 4 | 3 | 1 | 0 | 323266 |
| 20000000 | 6 | 5 | 1 | 0 | 435625 |
| 30000000 | 6 | 1 | 1 | 4 | -9729261 |

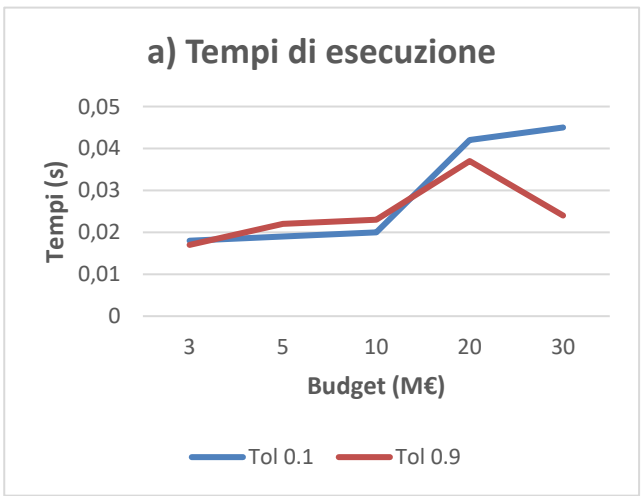
**Tabella 5.2: Test sulla rete di Roma-Termini. Analisi della migliore soluzione con tolleranza pari a 0.1.**

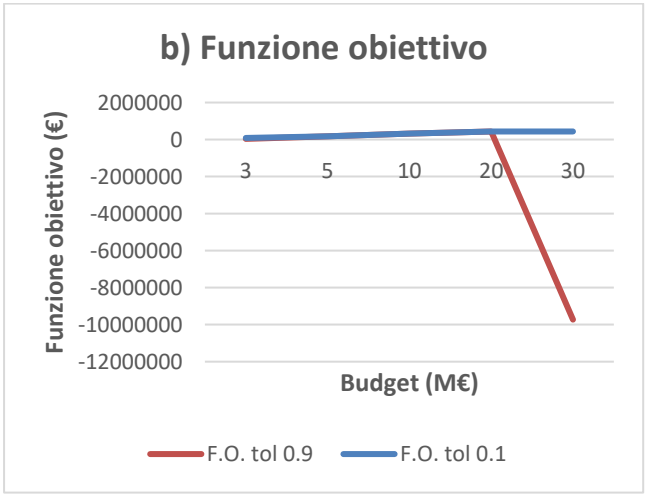
| Analisi della migliore soluzione con Tol. 0.1 | | | | | |
|---|---|---|---|---|---|
| Budget (€) | Numero linee | linee A | linee B | linee C | F.O. (€) |
| 3000000 | 1 | 0 | 1 | 0 | 86644 |
| 5000000 | 2 | 1 | 1 | 0 | 170519 |
| 10000000 | 4 | 3 | 1 | 0 | 323266 |
| 20000000 | 6 | 5 | 1 | 0 | 435625 |
| 30000000 | 6 | 5 | 1 | 0 | 435625 |

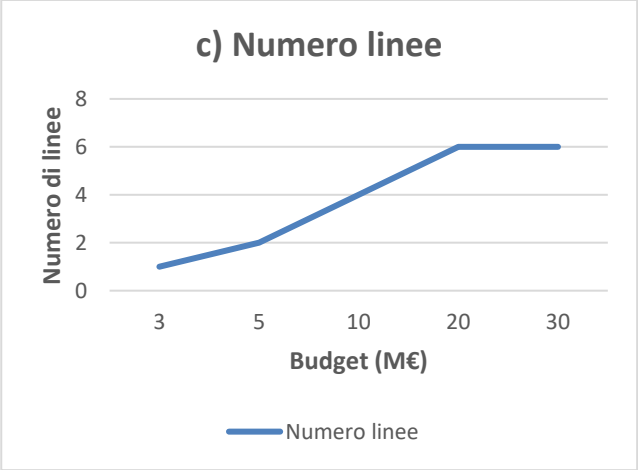
**Tabella 5.3: Test sulla rete di Roma-Termini. Confronto del tempo di esecuzione in base alla tolleranza.**

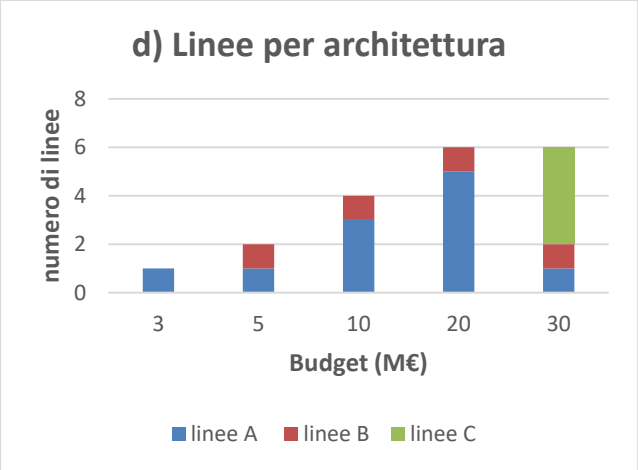
| Budget (€) | Tempo esecuzione Tol. 0.1 (s) | Tempo esecuzione Tol. 0.9 (s) |
|---|---|---|
| 3000000 | 0,018 | 0,017 |
| 5000000 | 0,019 | 0,022 |
| 10000000 | 0,02 | 0,023 |
| 20000000 | 0,042 | 0,037 |
| 30000000 | 0,045 | 0,024 |

**Figura 5.1: Tale figura mostra 4 grafici che riportano i seguenti risultati riguardo la rete di Termini: a) Tempi di esecuzione del programma, (b) Valori della funzione obiettivo, (c) Numero di linee elettrificate nel caso della migliore soluzione e (d) Numero di linee per architettura.**






Analizzando il tempo di esecuzione (cfr. Figura 5.1a) si evince che con budget molto piccoli i tempi sono confrontabili e seguono un andamento crescente sia per tolleranza 0.1 che 0.9. Si nota però che aumentando il budget fino a 30M€ il tempo di esecuzione dei due casi analizzati ha un margine di differenza notevole. La motivazione di questa differenza risiede proprio nell'implementazione del taglio del calcolo del fattore di riduzione. Con una tolleranza pari a 0.9 il taglio del calcolo del fattore di riduzione incide maggiormente rispetto al caso con una tolleranza pari a 0.1. Inoltre, in questo test il tempo di esecuzione si riduce di circa la metà, sintomo di come il calcolo del fattore di riduzione sia diventato un fattore che fa spendere notevole tempo di calcolo.

Osservando la figura 5.1b si può osservare come la funzione obiettivo nel caso di tolleranza 0.1 sia sempre maggiore o uguale al caso con tolleranza 0.9. Tale risultato è dovuto al fatto che a parità di budget, il caso con tolleranza pari a 0.1 analizza tutte le sottoreti considerate anche dal caso con tolleranza uguale a 0.9, ma ne considera anche altre, in quanto la soglia minima per soddisfare il vincolo di budget è diventata ancora più piccola. Inoltre, si nota come nel caso di tolleranza 0.9 ci siano valori della funzione obiettivo negativa. Ciò è dovuto al fatto che aumentando il budget ed essendo la tolleranza pari a 0.9, tra le linee ammissibili rimangono solo le più costose (quelle con architettura C). Tuttavia, le linee con architettura C non riescono a fare economia di scala, essendo poche le linee considerate. Ciò condurrebbe alla possibile conclusione per cui un budget basso escluderebbe l'architettura C, a favore delle altre due tipologie di elettrificazione.

Osservando gli ultimi due grafici della figura 5.1 si evince che aumentando il budget a disposizione cresce anche il numero di linee che nel caso migliore vengono elettrificate. Il risultato più interessante è quello mostrato nella figura 5.1d in cui viene mostrata la tipologia con cui si intende elettrificare le linee nel caso di tolleranza 0.9. Quello che si nota è che a valori alti della funzione obiettivo corrispondono linee elettrificate maggiormente con l'architettura A.

## *5.3 Risultati del caso di studio di Via del Corso*

Il secondo test significativo che si intende analizzare è quello che si è eseguito sulla rete di via Del Corso. Ovvero si è analizzata la rete di via Del Corso in modo simile a quella di Roma Termini per poter comprendere le differenze dei risultati ottenuti, essendo le due reti differenti tra loro.

**Tabella 5.4: Test sulla rete di Via Del Corso. Analisi della migliore soluzione con tolleranza pari a 0.9.**

| Analisi della migliore soluzione con Tol 0.9 | | | | | |
|---|---|---|---|---|---|
| budget (€) | Numero linee | linee A | linee B | linee C | F.O. (€) |
| 10000000 | 3 | 0 | 3 | 0 | 1499108 |
| 20000000 | 4 | 0 | 4 | 0 | 2894050 |
| 30000000 | 7 | 0 | 7 | 0 | 3731795 |
| 40000000 | 9 | 0 | 9 | 0 | 4136561 |
| 50000000 | 11 | 3 | 8 | 0 | 3955103 |
| 60000000 | 11 | 0 | 10 | 1 | -1330665 |
| 70000000 | 13 | 0 | 10 | 3 | -7339004 |

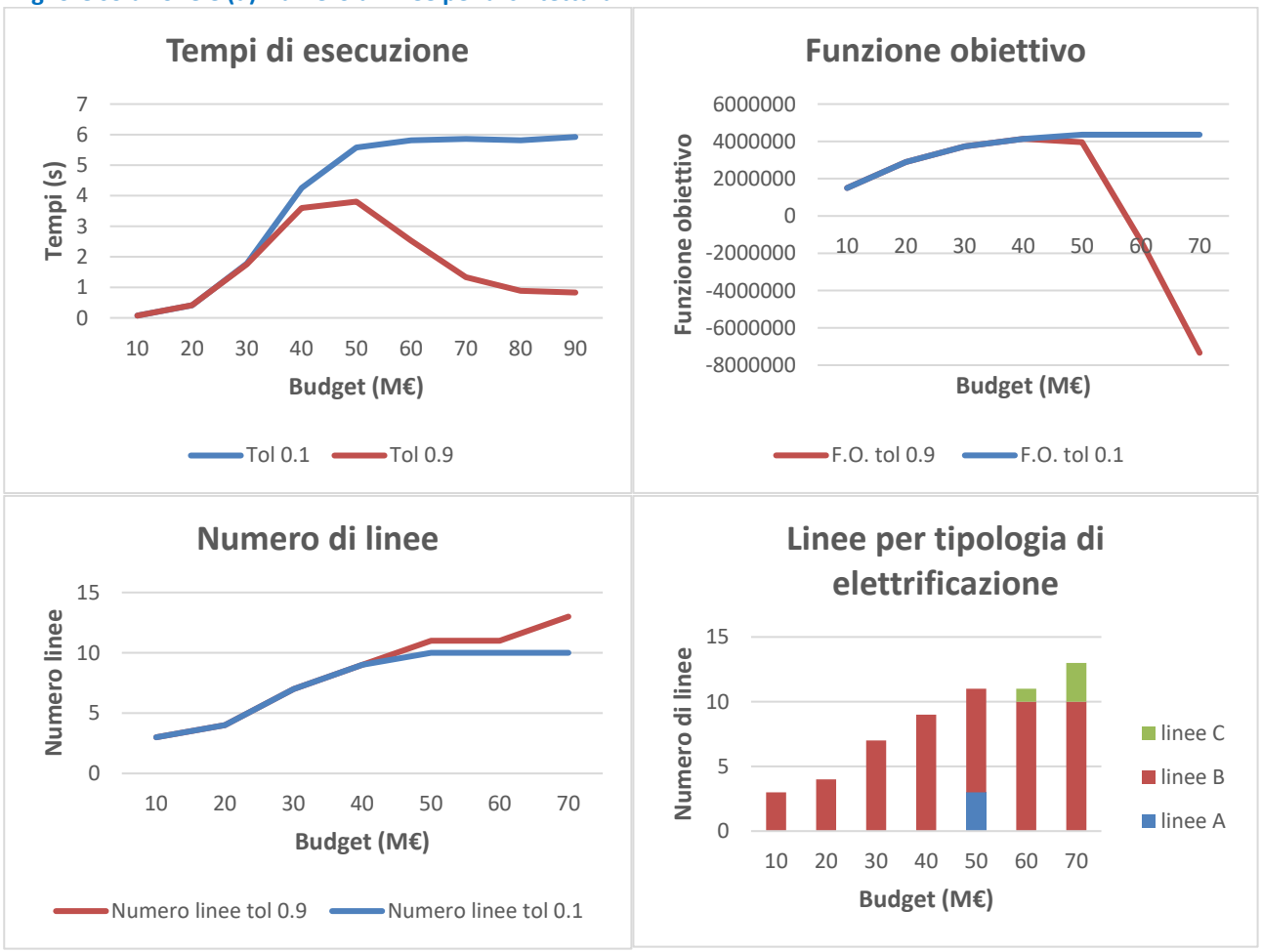
**Tabella 5.5: Test sulla rete di Via Del Corso. Analisi della migliore soluzione con tolleranza pari a 0.1.**

| Analisi della migliore soluzione con Tol 0.1 | | | | | |
|---|---|---|---|---|---|
| budget (€) | Numero linee | linee A | linee B | linee C | F.O. (€) |
| 10000000 | 3 | 0 | 3 | 0 | 1499108 |
| 20000000 | 4 | 0 | 4 | 0 | 2894050 |
| 30000000 | 7 | 0 | 7 | 0 | 3731795 |
| 40000000 | 9 | 0 | 9 | 0 | 4136561 |
| 50000000 | 10 | 0 | 10 | 0 | 4359131 |
| 60000000 | 10 | 0 | 10 | 0 | 4359131 |
| 70000000 | 10 | 0 | 10 | 0 | 4359131 |

**Tabella 5.6: Test sulla rete di Via Del Corso. Confronto del tempo di esecuzione in base alla tolleranza.**

| | Tempi (s) | |
|---|---|---|
| Budget (€) | Tempo esecuzione Tol 0.1 | Tempo esecuzione Tol 0.9 |
| 10000000 | 0,076 | 0,074 |
| 20000000 | 0,408 | 0,415 |
| 30000000 | 1,78 | 1,754 |
| 40000000 | 4,255 | 3,6 |
| 50000000 | 5,576 | 3,806 |
| 60000000 | 5,812 | 2,527 |
| 70000000 | 5,86 | 1,331 |
| 80000000 | 5,813 | 0,885 |
| 90000000 | 5,923 | 0,832 |

**Figura 5.2: Tale figura mostra 4 grafici che riportano i seguenti risultati riguardo la rete di via Del Corso: a) Tempi di esecuzione del programma, (b) Valori della funzione obiettivo, (c) Numero di linee elettrificate nel caso della migliore soluzione e (d) Numero di linee per architettura.**




La rete di via Del Corso mostra risultati simili a quelli di Roma-Termini, ma essendo una rete più grande e differente nella topologia ci consente di notare alcune sfumature interessanti.

Come nel caso precedente i tempi di esecuzione nel caso di tolleranza 0.9 risultano essere comparabili per budget piccoli. Tuttavia, in questo test è ancora maggiormente evidente l'andamento dei tempi di esecuzione con il crescere del budget. L'andamento per quanto riguarda la tolleranza 0.1 è di tipo monotono crescente, nel secondo caso si ha un andamento a campana che ci mostra ancora maggiormente quanto sia stato importante introdurre il taglio del calcolo del fattore di riduzione.

Il grafico relativo alla funzione obiettivo conferma come non sempre aumentare il budget dedicato all'investimento iniziale possa dare dei benefici. L'analisi della funzione obiettivo e dei tempi di esecuzione mostra come sia importante trovare il giusto budget da analizzare nel caso in cui si voglia utilizzare un fattore di tolleranza alto (0.9) che riduce notevolmente i tempi di esecuzione del programma.

Di notevole interesse è anche il risultato ottenuto dall'analisi delle linee. In primo luogo, si vede come nel caso di tolleranza 0.1 si ha che pur crescendo il budget il numero di linee associato alla miglior soluzione ad un certo punto rimane costante a 10, nonostante ci sarebbe la possibilità di considerare anche le altre tre linee con tale budget. Questo accade perché le tre linee che vengono escluse, se aggiunte andrebbero a far diminuire il valore della funzione obiettivo. Infatti, si tratta di linee che sono elettrificabili solo con architettura C o anche B, ma che non riescono ad ottenere vantaggio dall'effetto rete.

Infine, si può notare come ci sia una predominanza dell'architettura B che risulta essere la più vantaggiosa nel caso di studio di via Del Corso, al contrario di quanto visto con il caso di termini in cui si erano ottenuti risultati migliori attraverso l'elettrificazione con l'architettura A.

## 5.4 Confronto nella tipologia di architetture scelte nelle reti migliori di Termini e Corso

In questa sezione si riportano i dati relativi alla numerosità con cui una tipologia di architettura compare nel caso delle reti di Termini, Corso e Corso_ABC, nel caso in cui si utilizzi come parametri:

- Un budget illimitato e si voglia elettrificare le reti che massimizzano la funzione obiettivo.
- Un valore della tolleranza pari a 0, ossia che il vincolo di budget non esclude una rete se per elettrificarla si spende molto meno di quanto sia il budget a disposizione.

**Tabella 5.7: Analisi delle migliori sottoreti con funzione obiettivo positiva.**

| Analisi delle migliori sottorete con F.O. positiva | | | | |
|---|---|---|---|---|
| Rete | Numero di sottoreti con F.O. positiva | Linee elettrificate con Arch. A | Linee elettrificate con Arch. B | Linee elettrificate con Arch. C |
| Termini | 30 | 59% | 41% | 0% |
| Corso | 30 | 12% | 88% | 0% |
| Corso_ABC | 7 | 55% | 45% | 0% |

**Tabella 5.8: Analisi delle 3 migliori sottorete con funzione obiettivo positiva.**

| Analisi delle 3 migliori sottorete con F.O. positiva | | | |
|---|---|---|---|
| Rete | Linee elettrificate con Arch. A | Linee elettrificate con Arch. B | Linee elettrificate con Arch. C |
| Termini | 83% | 17% | 0% |
| Corso | 23% | 77% | 0% |
| Corso_ABC | 80% | 20% | 0% |

Analizzando le tabelle 5.7 e 5.8 è evidente che tra le 30 migliori sottoreti non ci sono sottoreti che abbiano funzione obiettivo positiva e al contempo una linea elettrificata con l'architettura C. Infatti, da tale analisi si evince che le migliori sottoreti di reti come Termini e Corso tendono ad escludere completamente le linee con architettura C.

Inoltre, è interessante vedere come in base alla rete considerata ci sia una tra le due architetture A e B che prevale sull'altra. Infatti, via Del Corso sembra protendere per un'elettrificazione di tipo B, e questo è evidente sia dall'analisi generale delle 30 migliori sottoreti (88%), che da quella più specifica fatta sulle 3 migliori sottoreti (77%).

Al contrario, Termini sembra essere meglio elettrificabile con l'architettura A, ma in tal caso questa conclusione è maggiormente evidente nel caso dell'analisi specifica fatta sulle 3 sottoreti migliori (in cui si ha l'83% delle linee elettrificate con l'architettura A).

## 5.5 Risultati del caso di studio della rete di Roma

Per ottenere risultati sulla rete di Roma si è cercato di sfruttare pienamente l'architettura di Cresco. I test sono stati eseguiti su Cresco4 e si sono eseguiti sulla coda cresco4_h6 che dà la possibilità di effettuare un run con un tempo limite di 6 ore.

Come analizzato in precedenza, la nuova struttura del programma ha permesso il salvataggio dello stato temporaneo, e dunque ci sono stati alcuni test che hanno necessitato di più run del codice.

Le tre reti analizzate in questo paragrafo sono Roma_ABC, Roma_AB-C_piccola e Roma_AB-C_grande.

A parità di numero di linee si è analizzato il numero di run necessari per poter trovare la soluzione nei due casi Roma_ABC e Roma_AB-C_piccola. Infatti, queste due reti hanno un numero uguale di linee ma si differenziano per il fatto che la prima contiene tutte e 3 le tipologie di elettrificazione, mentre la seconda contiene solamente la tipologia A e B. Tale confronto lo si è voluto analizzare per poter capire se effettivamente il tempo di analisi dell'architettura C influisse, e di quanto, sul tempo di esecuzione.

**Tabella 5.9: Test su due sottoreti di Roma: una elettrificabile con tutte le 3 architetture e l'altra con solo le architetture A e B. Il test è stato effettuato con un budget di 10milioni di euro, un numero di file di input da analizzare pari a 500 e un numero di job array pari a 16.**

| Test con budget=10M€, fileInput=500, jobArray=16, tol=0.8 | |
|---|---|
| Rete analizzata | Numero di run |
| Roma_ABC | 5 |
| Roma_AB-C_piccola | 1 |

**Tabella 5.10: Qui viene riportato il numero di file di input che si è riusciti ad analizzare ad ogni singolo run nel caso della rete di Roma_ABC.**

| Analisi dei diversi run effettuati sulla rete Roma_ABC | |
|---|---|
| Numero del run del programma | Numero di file di input analizzati con successo |
| 1 | 295 |
| 2 | 129 |
| 3 | 48 |
| 4 | 14 |
| 5 | 1 |

Come è evidente dalla tabella 5.9, si può evincere che il calcolo relativo all'architettura C ha introdotto un notevole rallentamento nell'esecuzione, rendendo necessari un numero di run pari a 5, rispetto all'unico run che è stato sufficiente per analizzare tutta la rete Roma_AB-C_piccola.

Dalla tabella 5.10 si può osservare come il numero di file che si riescono ad analizzare ad ogni run tende a diminuire. La spiegazione di tale diminuzione nel numero di file analizzati con successo, è dovuto al fatto che ci sono dei file di input che prevedono maggiore tempo di calcolo rispetto ad altri.

Inoltre, si evince che ci sono 13 dei 500 file che non si è riusciti ad analizzare. La motivazione è legata alla granularità troppo grossa dei file di input, per cui essendoci in un file di input troppe sottoreti da analizzare, non si riesce a terminare con successo l'analisi del singolo file.

Il modo per poter aggirare tale ostacolo è di generare i file di input con una granularità ancora più fine. A tal proposito si è voluto effettuare un test riguardante il numero di file analizzati durante il primo run della rete Roma_ABC sulla coda cresco4_h6.

**Tabella 5.11: Test sulla rete Roma_ABC che analizza il numero di file che il programma riesce ad analizzare durante il primo run del programma**

| Test riguardo il primo run sulla rete Roma_ABC con budget=20M€ e tol = 0.8 | |
|---|---|
| Numero di file di input | Numero di file analizzati |
| 500 | 5 |
| 1000 | 36 |
| 1500 | 51 |
| 2000 | 119 |

Come si evince dalla tabella 5.11 si può notare come il numero di file che il programma riesce ad analizzare con l'aumentare della granularità dei file di input aumenta sempre. Tale analisi ci fa comprendere che maggiore è la granularità dei file di input, e maggiore sarà il numero di sottoreti che si riuscirà ad analizzare.

### 5.5.1 Risultati delle migliori sottoreti trovate

In questo paragrafo vengono riportati i diversi risultati ottenuti da alcuni test sulle sottoreti più importanti di Roma denominate Roma_AB-C_piccola e Roma_AB-C_grande.

I risultati sono stati ottenuti tramite dei test effettuati con un valore di budget pari a 10 milioni di euro e una tolleranza del valore di 0.8.

**Tabella 5.12: Qui viene riportato il risultato del test effettuato su Roma_AB-C_piccola. Si tratta di una rete con 34 linee, non aventi linee elettrificabili con l'architettura C.**

| obj_function_value | number_of_lines | line_id | arch_id | line_id | arch_id | line_id | arch_id | line_id | arch_id | line_id | arch_id |
|---|---|---|---|---|---|---|---|---|---|---|---|
| 6281588.98700000 | 4 | 20 | B | 33 | B | 47 | B | 86 | B | | |
| 6243741.97260000 | 4 | 20 | B | 33 | B | 50 | B | 86 | B | | |
| 6183562.97360000 | 4 | 8 | B | 33 | B | 47 | B | 50 | B | | |
| 6123299.06600000 | 4 | 33 | B | 38 | B | 47 | B | 83 | B | | |
| 6113180.56232222 | 4 | 16 | B | 33 | B | 83 | B | 86 | B | | |
| 6112677.36600000 | 4 | 33 | B | 38 | B | 47 | B | 97 | B | | |
| 6102536.68232222 | 4 | 18 | B | 33 | B | 83 | B | 86 | B | | |
| 6085452.05160000 | 4 | 33 | B | 38 | B | 50 | B | 83 | B | | |
| 6074830.35160000 | 4 | 33 | B | 38 | B | 50 | B | 97 | B | | |
| 6048873.09170000 | 5 | 16 | B | 20 | B | 33 | B | 47 | B | 50 | B |
| 6048715.05200000 | 4 | 20 | B | 33 | B | 47 | B | 83 | B | | |
| 6038229.21170000 | 5 | 18 | B | 20 | B | 33 | B | 47 | B | 50 | B |
| 6038093.35200000 | 4 | 20 | B | 33 | B | 47 | B | 97 | B | | |
| 6010868.03760000 | 4 | 20 | B | 33 | B | 50 | B | 83 | B | | |
| 6005897.88085714 | 4 | 8 | B | 33 | B | 38 | B | 86 | B | | |

**Tabella 5.13: Qui viene riportato il test effettuato su Roma_AB-C_grande. Si tratta di una rete con 99 linee, non aventi linee elettrificabili con l'architettura C.**

| obj_function_value | number_of_lines | line_id | arch_id | line_id | arch_id | line_id | arch_id | line_id | arch_id | line_id | arch_id |
|---|---|---|---|---|---|---|---|---|---|---|---|
| 7228821.82600000 | 4 | 33 | B | 167 | B | 257 | B | 296 | B | | |
| 6959461.66410000 | 4 | 33 | B | 236 | B | 257 | B | 296 | B | | |
| 6956764.99267143 | 3 | 236 | B | 283 | B | 296 | B | | | | |
| 6949221.57800000 | 4 | 33 | B | 47 | B | 254 | B | 296 | B | | |
| 6941796.52000000 | 2 | 257 | B | 283 | B | | | | | | |
| 6937298.51800000 | 4 | 33 | B | 164 | B | 254 | B | 296 | B | | |
| 6911374.56360000 | 4 | 33 | B | 50 | B | 254 | B | 296 | B | | |
| 6886442.79200000 | 4 | 33 | B | 187 | B | 290 | B | 296 | B | | |
| 6885681.86010000 | 5 | 33 | B | 187 | B | 236 | B | 254 | B | 296 | B |
| 6884905.97200000 | 5 | 20 | B | 33 | B | 167 | B | 254 | B | 296 | B |
| 6858253.53200000 | 4 | 33 | B | 237 | B | 290 | B | 296 | B | | |
| 6857492.60010000 | 5 | 33 | B | 236 | B | 237 | B | 254 | B | 296 | B |
| 6816314.49200000 | 4 | 33 | B | 83 | B | 187 | B | 296 | B | | |
| 6810800.72600000 | 4 | 167 | B | 257 | B | 290 | B | 296 | B | | |
| 6808172.43010000 | 4 | 16 | B | 33 | B | 257 | B | 296 | B | | |

Dai risultati riportati nelle precedenti tabelle si nota come nel caso di Roma, l'architettura B risulterebbe essere la più vantaggiosa. In effetti, si tratta di una rete più grande rispetto a quelle di Termini e Corso e in tal caso l'effetto rete fa abbassare i costi dell'architettura B.

Inoltre, si può notare che analizzando una rete più grande come quella con 99 linee si ottiene un valore della funzione obiettivo maggiore. Questo accade perché la rete Roma_AB-C_piccola è proprio un sottoinsieme di quella di Roma_AB-C_grande.

# 6 Conclusioni

I primi esperimenti fatti e lo studio teorico riportato in questo report hanno messo in evidenza come non sia semplice riuscire a risolvere un problema così vasto e di natura np-hard.

L'attività svolta dall'Università di Tor Vergata ha portato alla creazione di una nuova architettura del codice in grado di far eseguire dei test più a lungo e con salvataggio dello stato. Infatti, l'utilizzo dei job array ha portato a notevoli migliorie, come una gestione del workload fatta in maniera automatica da LFS, una scrittura del codice molto più semplificata in quanto non si è dovuto gestire la comunicazione tra i diversi nodi.

Si è inoltre visto come un'analisi preliminare delle linee elettrificate con l'architettura C, può portare grandi benefici durante l'analisi di rete.

L'introduzione dell'analisi della linea C ha comportato un notevole aumento del tempo di calcolo, che si è riuscito ad arginare limitatamente grazie al taglio del calcolo del fattore di riduzione e grazie ad una granularità più fine dei file di input, contenenti le sottoreti da analizzare.

Tuttavia, molteplici sono gli sviluppi futuri che si intendono perseguire:

- Analisi di reti ancora più grandi per poter restituire risultati di notevole interesse. Tali test su Cresco potrebbero richiedere un tempo macchina davvero elevato.
- Effettuare ulteriori attività di pre-processamento in grado di far diminuire il tempo complessivo di analisi degli algoritmi.

# 7 Riferimenti bibliografici

1. *"MPI Documents"*, http://mpi-forum.org/docs/
2. "Job Array", https://www.eneagrid.enea.it/Resources/Comandi-LSF/Stuff/LSF_multicase.htm

# 8 Abbreviazioni ed acronimi

CRESCO: Computational RESearch centre on Complex system.
DICII: Dipartimento di Ingegneria Civile ed Ingegneria Civile.
ENEA: Agenzia nazionale per le nuove tecnologie, l'energia e lo sviluppo economico sostenibile.
HPC: High performance computing.
TPL: Trasporto pubblico locale.
Lines: Numero di linee degli autobus di una rete.
Tol: Tolleranza accettata per gli investimenti iniziali.

# 9 Curricula

**Prof. Giuseppe Italiano**

Giuseppe F. Italiano si è laureato in Ingegneria Elettronica presso l'Università di Roma "La Sapienza" nel 1986 con 110 e lode. Dopo aver conseguito un Ph.D. in Computer Science presso la Columbia University nel 1991, ha lavorato come Research Staff presso il T. J. Watson Research Center dell'IBM, a Yorktown Heights, NY, USA, dal 1991 al 1996. Nel 1994, a 33 anni, è risultato vincitore del concorso nazionale a professore ordinario, ed è rientrato in Italia prendendo servizio all'Università di Salerno. E' stato chiamato dall'Università "Ca' Foscari" di Venezia nel 1995 e dall'Università di Roma "Tor Vergata" nel 1998. Attualmente è professore ordinario presso la LUISS. Negli anni, ha trascorso frequenti periodi di ricerca presso università e istituzioni straniere: è stato Visiting Professor presso Columbia University (NY, USA), Université Paris-Sud, Max–Planck–Institut fur Informatik, Saarbrücken (Germania), Hong Kong University of Science & Technology, e Visiting Scientist presso AT&T Research Labs (NJ, USA) e Microsoft Research (CA, USA). La sua attività di ricerca si è concentrata principalmente sul progetto di algorimi per grandi quantità di dati (Big Data), con applicazioni in diverse aree, come reti e grafi, analisi di reti sociali, cybersecurity e biologia computazionale. Nel 2016 è stato nominato Fellow dell'European Association for Theoretical Computer Science *per "fundamental contributions to the design and analysis of algorithms for solving theoretical and applied problems in graphs and massive data sets, and for his role in establishing the field of algorithm engineering".*