Alma Mater Studiorum · Università di Bologna

SCUOLA DI SCIENZE
Corso di Laurea in Informatica

SVILUPPO DI UN PATH PLANNER BASATO SU STIME DI RUMORE AMBIENTALE

Relatore:
Dott.
FEDERICO MONTORI

Presentata da:
JACOPO RIMEDIOTTI

Sessione II
Anno Accademico 2022-2023

Abstract

Le città essendo molto importanti dal punto di vista sociale ed economico sono caratterizzate dalla notevole presenza di flussi di traffico e persone. Per chi non conosce bene una certa città molto spesso è utile l'utilizzo di un navigatore che consiglia un percorso ottimale per raggiungere una certa destinazione in breve tempo, ma può non tenere in considerazione particolari preferenze dell'utilizzatore. L'utente può essere una persona che, per motivi di salute o personali, preferirebbe svolgere una passeggiata tranquilla evitando, se possibile, aree della città caratterizzate dall'inquinamento acustico o particolarmente rumorose a causa di affollamenti quali ad esempio scuole, mercati, stazioni, cantieri, ... Certe aree però possono essere caratterizzate da rumori ambientali di intensità variabile che dipende molto da particolari fasce orarie e/o giorni della settimana o date specifiche; quali ad esempio mercati, traffico, scuole, ...

Per questo motivo, si propone in questa tesi un sistema in grado di consigliare il miglior percorso in funzione alla preferenza di quiete dell'utente. Il sistema ideato approccia soluzioni diverse di previsione del rumore ambientale in base alla disponibilità di dati acustici, monitorati in tempo reale o storici, ed in caso di assenza considera un'euristica approssimativa.

Introduzione

L'inquinamento acustico è un tema importante che affligge principalmente le città, ed è un tema sensibile per quanto riguarda la qualità della vita dei cittadini. Secondo un recente rapporto AEA (European Environment Agency)[1], si stima che almeno il 20% della popolazione europea vive in aree dove i livelli di rumore ambientale causato dal traffico sono dannosi per la salute. In particolare si stima che 113 milioni di persone siano esposte a livelli di rumore da traffico diurno / notturno di almeno 55 db(A) e ulteriori stime interessano alti livelli di rumore provenienti da aerei e fabbriche. Specialmente in contesto urbano il rumore ambientale ha importanti effetti sulla salute che vanno dal malessere fisico a effetti psicologici fino alla perdita di udito temporanea o permanente[2;3]. Per questi motivi gli stati dell'Unione Europea hanno cercato di applicare soluzioni per l'analisi e riduzione del rumore, e la mappatura della distribuzione del rumore nelle aree urbane rappresenta un passo verso la soluzione del problema.

Un progetto verso questa direzione è il progetto LIFE+ "DYNAMAP", sperimentato sulle città di Milano e Roma, che ha lo scopo di sviluppare un sistema dinamico di mappatura acustica per rilevare e rappresentare in tempo reale l'impatto acustico generato dalle infrastrutture stradali, tramite l'utilizzo di sensori a basso costo[4].

In letteratura esistono diversi studi che impiegano l'uso dei sensori negli smartphone in vari ambiti, tra cui la misurazione dei livelli di rumore ambientali in zone interne ed esterne delle città. I risultati di molti studi che rientrano nell'ambito del mobile crowdsensing e mobile crowdsourcing, mostrano che i microfoni degli smartphone sono sufficiente precisi per essere impiegati nel monitoraggio preliminare del rumore ambientale a patto di implementare negli algoritmi degli accorgimenti per non compromettere la qualità delle misurazioni a causa delle abitudini degli utenti (per esempio ignorare le misurazioni quando viene rilevato che lo smartphone è tenuto in tasca)[5].

Molti progetti sviluppati negli ultimi anni propongono piattaforme e applicazioni per smartphone capaci di monitorare il rumore ambientale in intervalli di tempo, ed etichettarlo con coordinate geografiche fornite dal GPS; tra le quali NoiseProbe[6], NoiseTube[7], Noise Planet[8], ... Alcune piattaforme come Noise Planet sono basate su contributi di monitoraggio anonimi collettivi di vari utenti.

I risultati degli studi di NoiseProbe, mostrano che i livelli di rumore ambientale in città variano in modo non lineare in funzione di orario e posizione e inoltre sono

influenzati principalmente da pattern del traffico e dal movimento delle persone, e feste locali[6].

Per quanto detto in questa introduzione, con il lavoro presentato in questa tesi si vuole mostrare alcuni approcci di previsione del rumore ambientale e la loro applicazione in una piattaforma che implementi un servizio di navigazione con il fine, di fornire delle previsioni sui livelli di rumore ambientale per ogni percorso stradale di una città e in funzione di queste consigliare un percorso ottimale in base alle preferenze dell'utente. Le previsioni del caso di studio sono stimate a partire da un dataset di 26016 misurazioni campionate su alcune strade della città di Bologna tra il 27 maggio 2023 e il 7 luglio 2023 tra le fasce orarie 14:00 - 17:00, non caratterizzati dalla presenza di precipitazioni. La raccolta dati è avvenuta con l'ausilio di un'applicazione sviluppata per smartphone orientata per contribuzioni collettive che sarà anche essa discussa in questa tesi.

Indice

Elenco delle figure

Lista dei codici

Elenco delle tabelle

Capitolo 1

Stato dell'arte

In questo capitolo saranno introdotte le tecnologie utilizzate e gli approcci seguiti. nell'implementazione del progetto

1.1 Introduzione al Mobile Crowdsensing

Le applicazioni per smartphone sono diventate componenti importanti durante attività di lavoro, sociali e di svago; capaci di soddisfare diversi tipi di bisogni in base al contesto. Queste applicazioni posso richiedere l'uso di sensori integrati nei dispositivi per soddisfare un determinato bisogno. I sensori come GPS e bussola, sono di solito impiegati in applicazioni che forniscono servizi di mappe geografiche, di navigazione, o previsioni meteo; i microfoni per mandare messaggi vocali ed effettuare chiamate; i sensori di luce per illuminare lo schermo in base all'illuminazione disponibile, ecc ...

I sensori però possono essere impiegati anche a fine di monitoraggio di un determinato fenomeno riguardo un certo ambiente o contesto. Questa tipologia di attività è definita con termine "mobile crowdsensing" (MCS) e si riferisce a un'ampia gamma di paradigmi di monitoraggio basati su contribuzioni collettive[11].

Il monitoraggio di un determinato fenomeno può avvenire in modalità diverse in base al contesto per cui necessita un diverso coinvolgimento da parte dell'utente e possono essere classificate in due categorie:[11]

1. Participatory crowdsensing
2. Opportunistic crowdsensing

La modalità "participatory crowdsensing" o "community sensing" consiste in un approccio fondato su contribuzioni collettive e volontarie da parte degli utenti, per collezionare informazioni per un determinato contesto o fenomeno monitorato[11].

La modalità "opportunistic crowdsensing" è basata invece su un monitoraggio e aggregazione di contribuzioni ottenute passivamente in modo automatizzato, dove l'intervento da parte dell'utente è minimale o totalmente rimosso[11].

Il lavoro di questa tesi è basato sullo sviluppo di un'applicazione che approccia una modalità volontaria di contribuzione, però tenendo in considerazione una visione futura per integrare le funzionalità di monitoraggio del rumore ambientale nella componente client di un servizio di navigazione, applicando eventualmente un approccio opportunistico; quindi ad esempio monitorando passivamente il rumore ambientale mentre l'utente usufruisce attivamente il servizio di navigazione.

1.2 Sistema di navigazione

L'utilizzo dei servizi di mappe geografiche e di navigazione si sono diffusi ormai da moltissimi anni e con la diffusione degli smartphone sono diventati uno strumento essenziale per viaggiare. Alcuni servizi di questo tipo ormai molto conosciuto sono TomTom[12], Google Maps[13] che già da alcuni anni effettuano la mappatura della rete stradale di moltissimi paesi sia su percorsi percorribili su veicoli che a piedi.

Un altro servizio di mappe geografiche similare è OpenStreetMap, che con una politica di distribuzione del servizio tramite Open Data, ha mappato la rete stradale di tutto il mondo per mezzo di contributi volontari della comunità popolati da conoscenze locali e mappature di tracciati con dispositivi GPS personali[14]. I contributi permettono a OpenStreetMap di fornire informazioni semantiche riguardo ogni tratto stradale, come ad esempio la categoria di strada, lunghezze del tracciati e limiti di velocità.

Per tutti questi tipi di servizi Il risultato della mappatura della rete stradale consiste in un multi grafo orientato, su cui è possibile applicare alcuni algoritmi su cui basano i servizi di navigazione che consistono principalmente nella risoluzione di problemi di cammini minimi su grafi.

In letteratura gli algoritmi di ricerca di cammini minimi sono stati abbondantemente studiati, alcuni di distinguono per l'efficienza computazionale, altri per l'applicabilità in contesti reali; sicuramente tra tutti il più conosciuto è l'algoritmo di Dijkstra.

L'algoritmo di Dijkstra risulta molto efficiente dal punto di vista computazionale e, anche se è applicabile solo su grafi con pesi non negativi, risulta ideale, rispetto ad altri algoritmi, se applicato su grafi i cui pesi associati agli archi possono variare frequentemente.

Normalmente il peso associato ad ogni arco dipende dal contesto in cui è utilizzato, ed in generale può essere associato a un indice di qualità, ma in contesti di rete della rete stradale il peso è frequentemente associato alla lunghezza di un arco o tratto stradale.

Google Maps è noto per utilizzare le funzionalità dei sensori GPS negli smartphone per migliorare i propri servizi[13]. Infatti da alcuni anni per alcuni paesi, è presente nella piattaforma la funzione di visualizzazione del traffico nei tratti stradali; e la piattaforma

sfrutta la collezione delle posizioni date dai sensori GPS degli stessi utenti per fornire delle previsioni su un indice di quantità traffico in base al numero di utenti che utilizzano il servizio di navigazione sullo stesso tratto stradale o su un arco del grafo. Considerate queste premesse, si intuisce che il servizio di Google Maps applica un approccio "opportunistic crowdsensing" per considerare l'indice di traffico come euristica per il peso degli archi oltre a tempi e distanze di percorrenza.

Una questione simile sarà affrontata in questa tesi, dove si vedrà che una certa previsione del rumore ambientale può essere utilizzata come indice di quiete o disturbo di un arco del grafo, applicabile in scenari con indici di quiete basate su dati acustici storici o aggiornati in tempo reale.

Uno strumento importante impiegato nel lavoro di questa tesi che ha permesso di recuperare dati geografici della rete stradale da OpenStreetMap, per modellarli in multi grafo orientato è il pacchetto Python OSMnx[15].

1.3 Cenni di acustica

I microfoni degli smartphone misurano il suono come variazione della pressione atmosferica, e le variazioni sono dette onde di pressione sonora o pressione acustica, e infatti i sensori campionano l'ampiezza dell'onda di pressione. In acustica si utilizza il livello di pressione sonora (SPL) su un intervallo di tempo T come misura logaritmica della pressione sonora efficace (RMS) di un'onda sonora rispetto a una sorgente sonora di riferimento, che in genere è espressa in decibel. Il livello di pressione sonora per un certo intervallo di tempo T è definito come[9;10]:

$$L_p(T) = 10 \log_{10}(\frac{p_{rms}}{p_{ref}})^2 \tag{1.1}$$

dove:

- $p(t)$ è la pressione istantanea misurata all'istante t
- $p_{rms} = \sqrt{\frac{1}{T}\int_0^T p(t)^2\,dt}$
- $p_{ref} = 20\mu Pa$ è la pressione sonora di riferimento in aria, considerata in letteratura come la soglia di udibilità per l'uomo.

Inoltre, per esprimere i livelli di rumore a livello internazionale, sono comunemente utilizzate le curve di ponderazione, cioè filtri per regolare i livelli sonori a diverse frequenze per tenere conto della diversa sensibilità dell'orecchio umano. Il livello sonoro in dB(A), che applica una curva di ponderazione "A", è di norma utilizzato come indice per valutare gli effetti del rumore sull'uomo[9].

Livello sonoro dB	Sorgente del rumore
0	Percezione minima del suono
20-25	Fruscio di foglie, bisbiglio
25-35	Ambiente silenzioso di notte, biblioteca
40-50	Strada tranquilla, conversazione tranquilla
50-60	Conversazione normale
60-70	Strada trafficata a 1m, ristorante, TV e radio ad alto volume
70-80	Sveglie, asciugacapelli
80-90	Camion vicino, macchinari industriali
100-110	Discoteca, martello pneumatico
110-120	Sirene, clacson a 1 metro
120	Decollo aereo

Tabella 1.1: Sorgente del rumore e livelli di pressione sonora[9;10]

Molti studi riguardo il rumore ambientale presenti in letteratura fanno uso di livelli di pressione acustica (SPL) o livelli equivalente del rumore L_{eq}. La seconda metrica rappresenta indicativamente una sorta di media del livello sonoro su un certo periodo di tempo ed è utilizzata perché si presta meglio a caratterizzare gli effetti sull'udito combinato con scale di ponderazione ed è richiesta in alcune norme internazionali[9]. Il livello di rumore equivalente non ponderato su un intervallo di tempo T è definito come[10] :

$$L_{eq}(T) = 10 \log_{10}(\frac{1}{T} \int_0^T 10^{L(t)/10}\, dt) \tag{1.2}$$

dove

- $L(t)$ rappresenta il livello di pressione sonora (SPL) al tempo t.

1.4 Motivazioni

Creare un modello di previsione del rumore ambientale è complicato perché come confermato da diversi studi, il rumore ambientale dipende da moltissimi fattori: fascia oraria, data, pattern del traffico, comportamenti delle persone, eventi, condizioni meteorologiche, e altri fattori. La componente principale sembra dipendere dal pattern del traffico[6;16;17;18]. Uno studio riguardante il progetto DYNAMAP suggerisce che un approccio basato sul raggruppare le strade per pattern del rumore e portata del flusso di traffico è molto promettente per prevedere il rumore ambientale del traffico, ma fa affidamento sulla disponibilità delle informazioni del segmento di stradale non legate all'acustica come il flusso di traffico giornaliero complessivo[16;17].

In mancanza di informazioni sul flusso del traffico, l'approccio di previsione del rumore ambientale seguito nel lavoro di questa tesi, si basa puramente su dati acustici "freschi" o di un dataset storico su un certo segmento stradale. Dunque sono stati individuati i seguenti casi:

- Caso `A`: Si ha a disposizione dati acustici molto recenti o ottenuti in tempo reale durante una certa fascia oraria.
- Caso `B`: Non si ha disposizione dati acustici recenti, ma è disponibile un certo dataset storico sufficientemente popolato che rispetta certe condizioni.
- Caso `C`: Non sono disponibili dati recenti o storici riferiti a una strada, ma sono disponibili dati storici sulle strade adiacenti.

Per ciascun caso è stata individuata e adottata una soluzione diversa.

Soluzione al caso A: Si vuole osservare che certe situazioni non sono molto prevedibili con un dataset storico. Infatti certi eventi dipendono troppo da altri fattori, ad esempio i cantieri potrebbero venir aperti o terminati in giorni imprevedibili; Dunque la presenza di dati storici ha meno rilevanza in questi scenari. Per questo motivo, nei casi di tipo `A`, è meglio che siano considerati in modo prioritario i dati acustici relativamente freschi riferiti a certa fascia oraria e rilevati su un segmento stradale, piuttosto che applicare un modello di previsione basato su dati storici o euristiche.

Soluzione al caso B: Alcuni risultati ottenuti in letteratura, indicano non solo che il rumore ambientale segue certi pattern che dipendono in modo non-lineare in funzione a tempo e posizione, ma dipende anche da fattori esterni quali eventi locali, comportamenti delle persone e cambiamenti sul flusso del traffico[6;16;17;18]. Infatti è intuitivo pensare che certe aree siano più caratterizzate dal rumore rispetto ad altre in base al loro uso, e a potenziali eventi ciclici o sporadici a cui sono interessati certi segmenti stradali. Dunque la considerazione di un modello sufficientemente affidabile che possa prevedere una tendenza del rumore ambientale per ogni fascia oraria, basandosi su uno storico associato a un segmento stradale, lo rende ideale per essere applicato in scenari dove si verifica il caso `B`.

Soluzione al caso C: Altri risultati indicano inoltre che il pattern del traffico dipende dallo scopo e tipologia di veicoli che transitano sulla strada[16;17;18]. Per esempio il pattern di traffico su una ciclabile o percorso pedonale potrebbe essere diverso rispetto a una strada principale, ma è anche vero che in città spesso le ciclabili sono parallele a segmenti stradali su cui transitano altri veicoli, e dunque probabilmente il pattern di molti tratti di ciclabili è similare a quelle del segmento stradale vicino. Considerate queste osservazioni, una soluzione che è stata adottata sui segmenti stradali dove si verifica

la situazione del caso `C`, consiste nel considerare una componente che consideri il trend del rumore ambientale previsto per un modello di previsione associato alla categoria del tratto stradale, e un'altra che consideri il trend relativo ai segmenti adiacenti che lo potrebbero influenzare. Dunque un euristica approssimativa che tiene conto di queste componenti, può essere applicabile su un arco ogniqualvolta esso non rientri nel caso `A` o `B`, ma si verifichi il caso `B` invece sugli archi adiacenti e su un dataset relativo alla categoria stradale affinché sia possibile crearne i modelli di previsione da cui dipendono.

Con queste considerazioni, i modelli di previsione del rumore ambientale basati su dataset storici, adottati per le soluzioni `B` e `C` su cui si basa questo lavoro, è quello fornito dal tool Prophet.

Prophet usa un modello di scomposizione di serie storiche basato su frequenze di cambiamento di tre componenti principali: tendenza, stagionalità e festività[19].

Tenute in considerazione le osservazioni precedentemente introdotte, Prophet è sembrato ideale per essere utilizzato come fondamenta per il modello di previsione adottato nel lavoro di questa tesi.

Capitolo 2

Architettura

In questa sezione si vuole mostrare una panoramica dell'architettura adottata nel sistema di questa tesi, descrivendo le componenti e le interazioni tra di esse. L'intera architettura è sintetizzata nello schema (2.1).

2.1 Flusso esecutivo

Il funzionamento della piattaforma è garantito da tre fasi: Raccolta dati, Mappatura e allenamento del modello di previsione, Calcolo del percorso.

La fase di raccolta dei dati consiste nel popolamento di un database collettivo su cui si baseranno i modelli di previsione per una certa città. Il popolamento avviene per mezzo di campionamenti geo-localizzati del rumore ambientale per le strade di una città in varie fasce orarie, tramite i contributi volontari anonimi collettivi ottenuti frequentemente per mezzo di un'applicazione "Tracker" (1) per smartphone che sarà discussa in seguito.

L'interfacciamento con il database remoto avviene tramite il servizio (2) Back4app [20] . Quando si ritiene che il database ha sufficienti dati, si può procedere con la fase di costruzione e allenamento del modello.

La fase di costruzione e allenamento del modello consiste inizialmente con la modellazione dei segmenti stradali di una città in un multi grafo non orientato, che in questo caso avviene tramite le API del pacchetto OSMnx che si interfacciano a sua volta con le API REST fornite dal servizio OpenStreetMap e Nominatim per ottenere i dati geografici [15;21]. Successivamente segue la fase di mappatura dei record nel dataset con i corrispettivi archi del grafo, in base alla loro vicinanza spaziale. In questo lavoro è stato considerato empiricamente un raggio di 15 metri da ciascuna posizione del record, entro cui associare gli archi vicini. L'allenamento del modello consiste nel creare e applicare i modelli Prophet associati ai casi di previsione `B` e `C`. Al termine di queste operazioni si

ottiene un multi grafo non orientato e mappato con un corrispettivo modello per ciascun arco e un modello modelli associato a ciascuna categoria stradale (3).

Prima di poter rendere disponibile il servizio di navigazione, è necessaria una fase intermedia di inizializzazione del servizio per ridurre i tempi di elaborazione. Questa operazione si può considerare come un' operazione di pre-caching e consiste in un'interrogazione preliminare su tutti i modelli per ottenerne delle previsioni su frazioni orarie prefissate distribuite su una certa finestra temporale giornaliera (4) (ad esempio previsioni su due giorni successivi ogni 15 minuti).

Questa operazione è necessaria per aggiornare periodicamente le previsioni senza impattare sui tempi di risposta delle singole richieste e possono essere svolte in parallelo all'esecuzione del servizio.

A questo punto il servizio di navigazione può essere reso disponibile per calcolare e fornire il percorso ottimale richiesto in certe fasce orarie. Quando l'utente utilizza l'app di navigazione (5) e fornisce le posizioni di partenza e destinazione con una preferenza di quiete, il server si occupa di eseguire l'algoritmo di Dijkstra applicato al grafo della città per trovare il percorso ottimale in funzione della preferenza tra distanza e quiete fornita dall'utente (6).

A questo punto vengono applicate le soluzioni di previsione considerate in base a in quale caso rientra ciascun arco con ordine di priorità `A`, altrimenti `B`, altrimenti `C`.

Infine i percorsi ottenuti vengono visualizzati sul client di navigazione su una mappa della rete stradale della città (7).

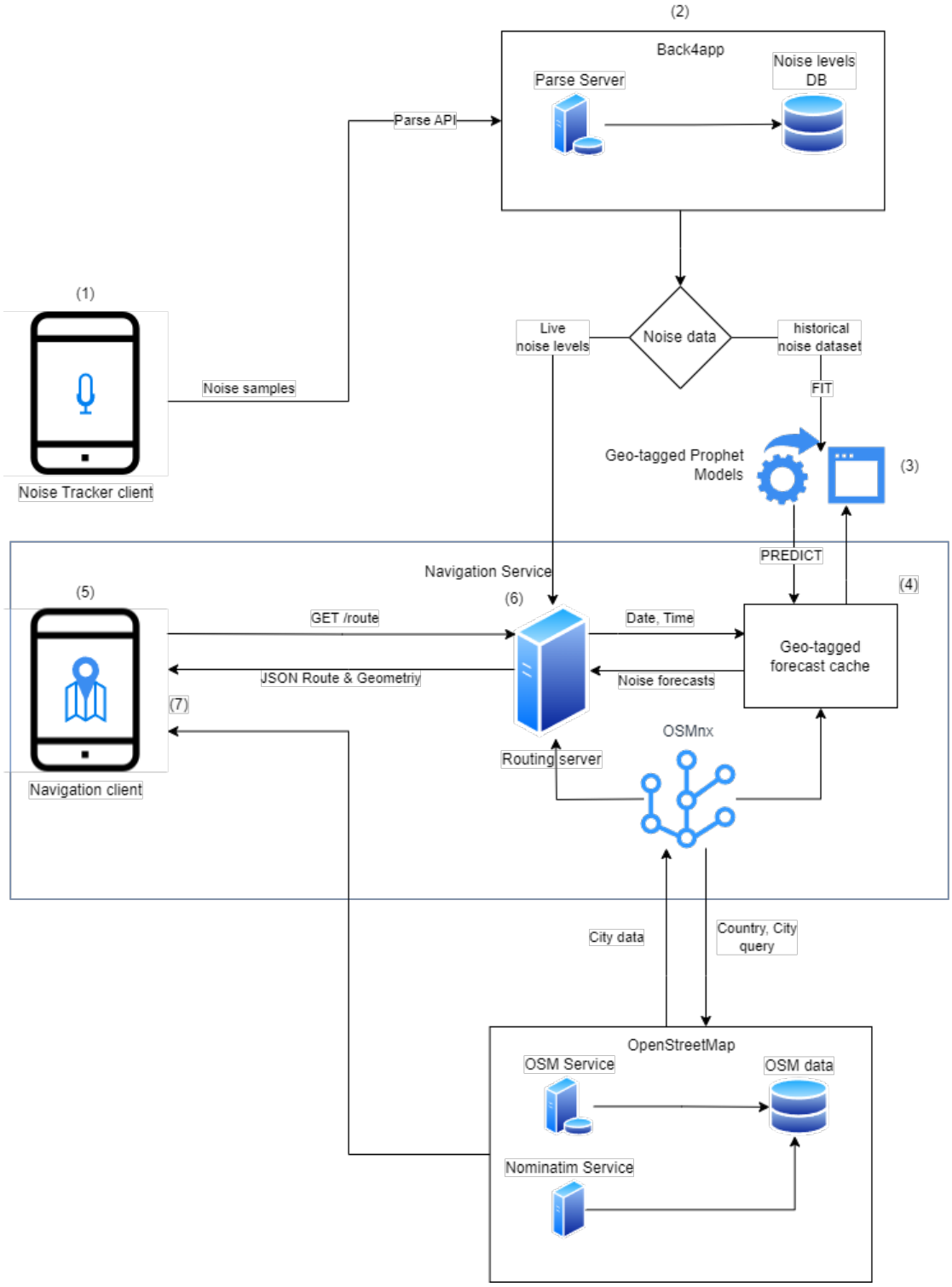


Figura 2.1: Diagramma architetturale

Capitolo 3

Dataset

La fase iniziale del lavoro è stata caratterizzata da raccolta e analisi dei dati su un database remoto.

3.1 Database

Il database remoto scelto per il lavoro è fornito dal servizio Back4app, che permette di accedere e manipolare un database NoSQL utilizzando API REST fornite dallo standard backend "Parse Server". Come già accennato il database può essere popolato tramite l'app "Tracker" ma il database potrebbe essere popolato anche da fonti esterne siccome Il servizio Back4app può rendere i database open source; il che lo renderebbe un candidato ideale per ottenere ulteriori contribuzioni collettive e accessibili dalla comunità.

Il database è costituito da una tabella "AmbientNoiseRecords" con il seguenti campi:

```
AmbientNoiseRecords(
  objectId,
  countryName, cityName, postalCode, location, radiusM
  time, SPL, minSP, maxSP, rmsSP
)
```

3.2 Raccolta dati

La fase di raccolta di dati acustici è avvenuta grazie all'implementazione e all'uso di un'applicazione che in questo lavoro è stato nominata "Tracker".

In un contesto di mobile crowdsensing, Le funzionalità di questa app sarebbero implementate come componente eseguita in background facente parte di un'applicazione client di un servizio di navigazione, con il fine di favorire le contribuzioni collettive mentre il servizio di navigazione viene utilizzato seguendo quindi un approccio di tipo "Opportunistic crowdsensing".

In questa sezione saranno discussi i dettagli implementativi e tecniche applicate per ottenere le misurazioni del rumore ambientale su cui è stato svolto il lavoro di questa tesi.

3.2.1 Interfacciamento con database

L'interfacciamento dell'applicazione "Tracker" con il database avviene tramite il Parse SDK che si interfaccia con le API compatibili con il framework "Parse Platform".

L'interfacciamento è inizializzato durante l'avvio dell'applicazione con apposite chiavi fornite dal servizio Back4app.

```
public class MyApplication extends Application {
  @Override
  public void onCreate() {
      super.onCreate();
      Parse.initialize(new Parse.Configuration.Builder(this)
              .applicationId(BuildConfig.PARSE_APPLICATION_ID)
              .clientKey(BuildConfig.PARSE_CLIENT_KEY)
              .server(BuildConfig.PARSE_SERVER_URL)
              .enableLocalDataStore()
              .maxRetries(5)
              .build()
      );
  }
}
```

3.2.2 Logica di campionamento

L'intera logica di campionamento è gestita con la seguente modalità: Quando viene rilevato che il dispositivo è in condizione di poter campionare il rumore, allora viene aggiornata una condizione di registrazione `shouldRecord`. Questa condizione dipende dai permessi concessi all'applicazione e da euristiche che contribuiscono al rilevamento di condizioni per abilitare o disabilitare il campionamento, che saranno trattate in seguito. Quando la condizione viene aggiornata, lo stato di registrazione viene orchestrato dalla seguente logica sintetizzata

```
if( shouldRecord ) {
  if( !isRecording ) {
    isRecording = true
    startRecording();
  }
}
else {
  if( isRecording ) {
    isRecording = false
    stopRecording();
```

```
  }
}
```

Quando il campionamento viene avviato tramite la funzione `startRecording()`, La registrazione del rumore viene avviata e gestita da un campionatore insieme a richieste periodiche ottenute ogni 5 secondi per accedere alla posizione fornita dal GPS del dispositivo. Una volta ottenute le posizioni all'inizio e al termine di ogni intervallo, le misurazioni elaborate vengono lette dal campionatore, ed etichettate con la posizione intermedia tra le posizioni ottenute e timestamp di fine campionatura per infine essere salvate. Per organizzare i record di diverse città, vengono inoltre raccolte informazioni come nome del paese. della città e codice postale di riferimento. Queste informazioni sono fornite dall' endpoint di reverse geoconding disponibili dal motore di ricerca `Nominatim` alimentato con i dati di OpenStreetMap[21]. Queste ultime operazioni sono riassunte dal seguente codice:

```
void gatherRecordedData(
  Location prevLocation, Location currentLocation) {

  SoundMetric soundMetric = sampler.read();

  LinkRecord record = new LinkRecord(
    prevLocation, currentLocation
  );
  record.setSoundMetric(soundMetric);
  record.setDistance(
    prevLocation.distanceTo(currentLocation)
  );

  mGeocoder.getAddress(
    record.getStartNode(),
    new GeoDecoderListener() {
      @Override
      public void onResult(AddressResult addressResult) {
          CoarseLocation coarseLocation = new CoarseLocation();
          coarseLocation.countryName
            = addressResult.address.country;
          // get first available on this priority order
          coarseLocation.cityName = Stream.of(
              addressResult.address.city,
              addressResult.address.town
          ).filter(Objects::nonNull).findFirst()
          .orElse(null);
          coarseLocation.postalCode
          = addressResult.address.postcode;

          record.setCoarseLocation(coarseLocation);
          repository.addRecord(record);
      }
```

```
  });
}
```

Il repository dei record agisce come Single Source Of Truth (SSOT) così da poter potenzialmente memorizzare in un certo numero di record in locale sul dispositivo e successivamente caricarli periodicamente sul database remoto su Back4App.

Nell'app Tracker, l'architettura SSOT e operazioni sul database sono gestiti in autonomia dal Parse SDK.

```
public void addRecord(LinkRecord record) {
  ParseObject obj = new ParseObject("AmbientNoiseRecords");
  obj.put("cityName", record.getCoarseLocation().cityName );
  obj.put("countryName", ... );
  obj.put("postalCode", ... );
  // centroid location
  obj.put("location", new ParseGeoPoint(
    getCentroidLocation(record.getStartNode(), record.getEndNode())
  ));
  obj.put("radiusM", record.getDistance() / 2.0d);
  obj.put("time", record.getEndNode().timestamp);
  obj.put("SPL", record.getSoundMetric().soundPressureLevel);
  obj.put("minSP", ... );
  obj.put("maxSP", ... );
  obj.put("rmsSP", ... );

  obj.pinInBackground();

  // ... save to remote logic
}
```

3.2.3 Misurazione del rumore

l'implementazione della fase di campionamento del rumore consiste nel calcolare il SPL su intervalli di tempo di 5 secondi. Dunque l'applicativo effettua continuamente le misurazioni della pressione istantanea per tutto intervallo e ne calcola il SPL.

Su Android è possibile campionare la pressione sonora istantanea per mezzo delle interfacce offerte dalla classe `AudioRecord`[22].

L'inizializzazione di un'istanza di `AudioRecord` avviene con il seguente frammento di codice, che considera un campionamento a 16000Hz con codifica PCM a 16 bit su canale MONO, rimuovendo o disabilitando elaborazioni del suono come sistemi automatici di riduzione del rumore che possono alterare la bontà dei dati.

```
int SAMPLE_RATE = 16000;
int CHANNEL_CONFIG = AudioFormat.CHANNEL_IN_MONO;
int ENCODING = AudioFormat.ENCODING_PCM_16BIT;
boolean isUnprocessedAudioSourceSupported = false
int bufferSize;
```

```
AudioRecord recorder;

private boolean initRecorder(Context context) {
  isUnprocessedAudioSourceSupported =
    mAudioManager.getProperty(
      AudioManager.PROPERTY_SUPPORT_AUDIO_SOURCE_UNPROCESSED
    ) != null;
  int audioSource;
  if (isUnprocessedAudioSourceSupported) {
    audioSource = MediaRecorder
      .AudioSource.UNPROCESSED;
  } else {
    // unprocessed alternative
    audioSource = MediaRecorder
      .AudioSource.VOICE_RECOGNITION;
  }

  bufferSize = AudioRecord.getMinBufferSize(
    SAMPLE_RATE, CHANNEL_CONFIG, ENCODING
  );

  recorder = new AudioRecord(
    audioSource, SAMPLE_RATE, CHANNEL_CONFIG,
    ENCODING, bufferSize
  );

  // Disable Automatic Gain Control (AGC)
  if (AutomaticGainControl.isAvailable()) {
    agc = AutomaticGainControl
      .create(recorder.getAudioSessionId());
    agc.setEnabled(false);
  }
  return recorder != null;
}
```

Siccome i microfono degli smartphone forniscono misurazioni di pressione istantanea, per calcolare il livello sonoro equivalente sull'intervallo ($L_{eq}(5s)$) è necessario concentrarsi sulla formula per semplificare il calcolo affinchè dipenda solamente dalle pressioni istantanee. Si procede a semplificare la formula (1.2) precedentemente introdotta grazie alle proprietà dei logaritmi:

Sia $L(t)$ il livello di pressione sonora al tempo t

Sia $p(t)$ la pressione istantanea al tempo t

Allora:

$$\begin{aligned}
\mathcal{L}_{eq}(T) &= 10\log_{10}(\frac{1}{T}\int_0^T 10^{L(t)/10}\,dt) \\
&= 10\log_{10}(\frac{1}{T}\int_0^T 10^{(10\log_{10}(\frac{p(t)}{p_{ref}})^2)/10}\,dt) \\
&= 10\log_{10}(\frac{1}{T}\int_0^T \frac{p(t)^2}{p_{ref}^2}\,dt) \\
&= 10\log_{10}(\frac{1}{T}\int_0^T p(t)^2\,dt) + 20\log_{10}(\frac{1}{p_{ref}}) \\
&= 10\log_{10}(\frac{1}{T}\int_0^T p(t)^2\,dt) + costante
\end{aligned} \tag{3.1}$$

In questo modo è possibile calcolare L_{eq} in funzione della media integrale delle pressioni quadratiche istantanee, considerando l'integrale come somma di Riemann.

Tornando all'implementazione, quando il campionamento è avviato si effettua la decodifica dei campioni misurati provenienti da un buffer, collezionando alcune metriche nell'intervallo di misurazione. Le misurazioni ottenute da un'istanza `AudioRecord` sono da intendere come i livelli di pressione sonora istantanea campionati alla frequenza di 16000Hz.

Questa operazione viene sintetizzata dai seguenti frammenti di codice:

```
public void startRecord() {
  if( !isRecording ) {
    if( initRecorder(mContext) ) {
      isRecording = true;
      final short[] buffer = new short[bufferSize];
      // Recording started
      recorder.startRecording();
      while (isRecording) {
        isMuted = isMuted();
        int readSize = recorder.read(
          buffer, 0, bufferSize
        );
        if( readSize > 0){
          double[] samples = decodeSamples(
            buffer, readSize
          );
          analyzeSamples(samples);
        }
      }
      // Recording finished
```

```
    }
  }
}
```

dove le funzioni `decodeSamples()` e `analyzeSamples()` sono ispirate dall'implementazione del progetto NoiseTube[7;23].

La funzione `decodeSamples()` effettua la decodifica del canale MONO dai campioni signed a 16bit in valori normalizzati tra 0 e 1.

```
private double[] decodeSamples(short[] buffer, int count) {
  final double[] samples = new double[count];
  for (int i = 0; i < count; i++) {
    double decodedSample =
    (double) buffer[i] /
    (buffer[i] < 0
      ? (-Short.MIN_VALUE * -1.0d)
      : (double) Short.MAX_VALUE
    );

    samples[i] = decodedSample;
  }
  return samples;
}
```

La funzione `analyzeSamples()` invece si occupa di tracciare le metriche durante la fase di registrazione:

```
private void analyzeSamples(double[] samples) {
  double min = 0.0d, max = 0.0d, sum = 0.0d;
  boolean isFirst = true;
  int totalCount = 0;

  for (double sample : samples) {

    if( (isMuted != null && isMuted ) || isMuted(sample) )
      continue;

    if( isFirst ) {
      isFirst = false;
      min = sample; max = sample;
    }
    else {
      min = Math.min(min, sample);
      max = Math.max(max, sample);
    }

    // used by RMS or mean squared
    sum += sample * sample;
    totalCount++;
  }
```

```
  if( totalCount > 0 ) {
    boolean shouldInit = metricsBuffer == null;
    if (shouldInit) metricsBuffer = new RawSoundMetrics();
    metricsBuffer.countOfSoundPressures += totalCount;
    metricsBuffer.sumSquaredSoundPressures += sum;
    metricsBuffer.maxSoundPressureValue = shouldInit
      ? max
      : Math.max(max, metricsBuffer.maxSoundPressureValue);
    metricsBuffer.minSoundPressureValue = shouldInit
      ? min
      : Math.min(min, metricsBuffer.minSoundPressureValue);
  }
}
```

Dunque periodicamente su intervalli di 5 secondi una funzione `read()` viene richiamata per ottenere le metriche raccolte durante le misurazioni quali pressioni massime, minime e valore di $L_{eq}(5s)$ ottenuta dalla formula (3.1).

Questo calcolo è implementato nella funzione `read()`:

```
static final double REFERENCE_SOUND_PRESSURE
  = 2.0e-5d;
static final double ADDITIVE_REFERENCE_SPL
  = 20.0d * Math.log10(1.0d/REFERENCE_SOUND_PRESSURE);

public SoundMetric read() {
  RawSoundMetrics lastM;

  // save and clear
  lastM = metricsBuffer; metricsBuffer = null;

  if( lastM != null ) {
    SoundMetric soundMetric = new SoundMetric();
    // max
    soundMetric.maxSoundPressureValue
      = lastM.maxSoundPressureValue;

    // min
    soundMetric.minSoundPressureValue
      = lastM.minSoundPressureValue;

    // mean squared
    double soundPressureMeanSquare =
      lastM.sumSquaredSoundPressures
      / ((double) lastM.countOfSoundPressures);

    // rms in pressure
    soundMetric.rmsSoundPressureValue =
```

```
        Math.sqrt(soundPressureMeanSquare);

      // Leq SPL in db
      soundMetric.soundPressureLevel =
        10.0d * Math.log10(soundPressureMeanSquare)
        + ADDITIVE_REFERENCE_SPL;
    }

    return soundMetric;
}
```

Si vuole ora analizzare la precisione di rilevamento che si può ottenere.

Considerata la precisione di misurazione di un campione di pressione istantanea con un signed short normalizzato, ciascun campione normalizzato ha un intervallo di valori compreso tra $\frac{1}{2^{15}} = \frac{1}{32768}$ e $\frac{2^{15}}{2^{15}} = 1$

Si ha che:

- $20\log_{10}(\frac{1}{p_{ref}}) = 93.979db$
- $10\log_{10}(\frac{1}{T}\int_0^T \frac{1}{32768}^2\,dt) = 10\log_{10}(\frac{1}{32768}^2) = -90.309$
- $10\log_{10}(\frac{1}{T}\int_0^T 1^2\,dt) = 10\log_{10}(1) = 0$

Dunque per la formula (3.1) il range di rilevamento risulta essere circa tra 3.67 db e 93.98 db. Questo intervallo di misurazione rientra infatti nelle scale dei livelli di pressione sonora su eventi riscontrabili in contesto urbano, che includono situazioni di quiete fino a situazioni in presenza di mezzi pesanti rumorosi[9;10]. L'adozione di funzionalità di guadagno fornito eventualmente dai microfoni potrebbero incrementare la sensibilità, ma questo caso non rientra negli obbiettivi di questa tesi e per questo non sarà approfondito.

3.2.4 Tecniche per migliorare l'affidabilità delle misurazioni

Come raccomandato da alcuni studi[6], è necessario che le piattaforme che si basano su contributi collettivi dalla comunità adottino tecniche algoritmiche per mitigare interferenze delle misurazioni dovute alle abitudini degli utenti. Per risolvere questi problemi, i sensori degli smartphone posso offrire un enorme aiuto su questo punto di vista. Infatti per quanto riguardo gli smartphone Android alcune tecniche possono essere facilmente impiegate tramite l'utilizzo di librerie e servizi integrati "Context-aware" che rilevano una tipologia di attività da parte dell'utente. Le API di Activity Recognition su Android possono essere impiegate per rilevare se l'utente è in movimento, e se si trova su un veicolo. In questo caso si possono già rilevare i casi in cui l'applicazione "Tracker" è utilizzata impropriamente in automobile o mentre l'utente sta correndo, così da ignorare il campionamento durante questi tipi di attività.

API native del sistema operativo e i sensori di prossimità possono essere utilizzati per rilevare se l'utente sta effettuando una chiamata o tenendo il dispositivo vicino a una superficie che può attenuare il rumore come in una tasca o in una borsa.

Una tecnica di rilevazione di questo caso può essere implementata ad esempio disattivando le misurazioni Quando il sensore di prossimità rileva che il dispositivo si trova vicino a qualcosa per alcuni secondi. Questa logica sull'applicazione "Tracker" è implementata con il seguente frammento di codice:

```
mProximitySensor
  = mSensorManager.getDefaultSensor(Sensor.TYPE_PROXIMITY);

mProximitySensorListener = new SensorEventListener() {

  final static int timeRequiredBeforeCallback = 3000;
  Runnable onNearRun = this::onNearCallback;
  Runnable onFarRun = this::onFarCallback;
  Handler handler = new Handler(Looper.getMainLooper());

  public void onNearCallback() {
    // Disabilita misurazioni
  }

  public void onFarCallback() {
    // riabilita o continua misurazioni
  }

  @Override
  public void onSensorChanged(SensorEvent event) {

    float distance = event.values[0];
    float maxRange = event.sensor.getMaximumRange();

    // If device is near then will trigger a callback
    // which will update the detection after some time
    // but if sensor change the state before
    // the callback ends, then it will eventually
    // be interrupted if device is at
    // a different distance type
    if (distance >= Math.min(maxRange, 10f)) {
      handler.removeCallbacks(onNearRun);
      handler.postDelayed(
        onFarRun, timeRequiredBeforeCallback
      );
    } else {
      handler.removeCallbacks(onFarRun);
      handler.postDelayed(
        onNearRun, timeRequiredBeforeCallback
      );
```

```
    }
  }
};
mSensorManager.registerListener(
  mProximitySensorListener, mProximitySensor,
  SensorManager.SENSOR_DELAY_NORMAL
);
```

Capitolo 4

Modelli e euristiche di previsione

Si vuole ora descrivere dettagliatamente come sono implementati i modelli e le euristiche di previsione adottati nel progetto per ciascun caso precedentemente introdotto nella sezione 1.4. Ogni soluzione viene applicata con priorità dipendentemente a in quale caso rientra un certo arco e durante una certa fascia oraria richiesta:

- Se (caso `A`) sono presenti rilevazioni in tempo reale durante una fascia oraria T su un certo arco e allora è applicata la soluzione `A` che applica una previsione basata sui dati recenti della fascia oraria T.
- Altrimenti (caso `B`) se è presente un dataset di rilevazioni storiche associate all'arco e, sufficientemente popolato affinché si possa costruire un modello di previsione di Prophet ritenuto affidabile su certe fasce orarie, allora è applicata la soluzione `B`, che considera le previsioni simulate dal modello basato sul dataset storico.
- Altrimenti (caso `C`), se un certo arco e non rientra nei casi `A` o `B` ma gli archi adiacenti rientrano nel caso `B`, allora è considerata la soluzione `C`, che applica un'euristica approssimativa che tiene conto di due componenti in funzione della fascia oraria T: Una componente dipende dalle previsioni degli archi adiacenti e una componente dipende dalle previsioni associate alla stessa categoria stradale di e.

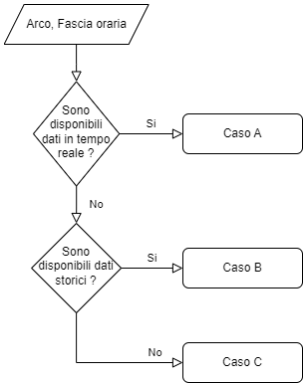


Figura 4.1: Diagramma di flusso che applica una soluzione per ogni caso in funzione di un arco e una fascia oraria di riferimento

Soluzione A

La soluzione per il caso `A` consiste nella previsione del livello di rumore equivalente su un arco di riferimento che ha ricevuto misurazioni recenti e in tempo reale durante una fascia oraria T.

La previsione consiste nel calcolo del livello medio di rumore equivalente $L_{eq}(T)$ sulla fascia oraria T. In particolare $L_{eq}(T)$ è calcolato applicando la formula (1.2), dove:

- t dipende dagli intervalli delle rilevazioni $L(t)$ dell'applicazione Tracker avvenuti a intervalli di tempo $t = 5$ *secondi*.
- T è considerata come la fascia oraria di riferimento, ad esempio considerando $T = 15$ si intende come fascia oraria $15:00 - 15:59$

Soluzione B

L'euristica di previsione adottata come soluzione al caso `B`, dipende da un modello di previsione del trend del rumore ambientale, basato su serie storiche fornito dal tool Prophet. Questo tipo di modello Prophet è istanziato per ciascun arco del grafo per cui è stato mappato un sottoinsieme del dataset storico di livelli di pressione sonora (SPL) rilevati entro 15 metri da ciascuno di essi.

La previsione consiste nel calcolo del livello medio di rumore equivalente $L_{eq}(T)$ su una fascia oraria T. In particolare $L_{eq}(T)$ è calcolato applicando la formula (1.2) considerando ciascun valore $L_{eq}(t)$ come i valori simulati dal modello Prophet associato all'arco durante la fascia oraria T distribuiti su intervalli di tempo t.

$$L_{eq,t}(T) = 10 \log_{10}(\frac{1}{N} \sum_{t}^{N=\frac{T}{t}} 10^{L_{eq}(t)/10}) \tag{4.1}$$

In questa soluzione è stato scelto empiricamente $t = 5$ *minuti* come intervallo di previsione. Dunque ciascun calcolo di $L_{eq,5min}(T)$ richiede $\frac{60}{5} = 12$ valutazioni del modello Prophet.

Soluzione C

La soluzione al caso `C`, come precedentemente introdotta, è adottata su un arco di riferimento dove le soluzioni `A` e `B` non sono applicabili su un arco di riferimento e, perché non rientra nei casi `A` e `B` rispettivamente per mancanza di dati recenti su una fascia oraria e mancanza di dati storici sufficienti. Però questa soluzione vuole considerare almeno i dati storici disponibili sugli archi adiacenti e riguardanti la categoria stradale. Dunque essa dipende da due tipologie di modelli di previsione: Un modello applicato agli archi ad esso adiacenti, identificato da quello adottato per la soluzione `B`, e un modello di previsione ma basato su serie storiche appartenenti alla stessa categoria stradale dell'arco di riferimento.

Il modello Prophet associato a una categoria stradale è basato su un sottoinsieme del dataset, i cui record sono mappati a un arco del grafo con la stessa categoria stradale di e.

L'euristica di questa soluzione si basa su una combinazione convessa che bilancia un valore ottenuto interpolando le previsioni $L_{eq,5min}(T)$ degli archi ad esso adiacenti dove risulta applicato la soluzione `B`, e il valore $L_{eq,5min}(T)$ di previsione calcolato in modo analogo applicando la formula (4.1) considerando le valutazioni $L_{eq}(t)$ del modello associato alla categoria stradale. L'euristica di previsione associata alla soluzione `C`, nominata anche `WINH` (Weighted Interpolation of Neighbours and Highway), è stata definita come:

$$WINH^e(T) = I(T)\alpha + C(T)(1 - \alpha) \tag{4.2}$$

dove:

- e è l'arco di riferimento
- $I(T) = L^{adj(e)}_{eq,5min}(T)$ è il valore interpolato come livello medio equivalente delle previsioni $L_{eq,5min}(T)$ su archi adiacenti ad e.
- $C(t) = L^{cat(e)}_{eq,5min}(T)$ è il valore del livello medio di rumore equivalente calcolato a partire dalle previsioni fornite dal modello associato alla categoria stradale dell'arco e, etichettata da OpenStreetMap (vedere [24]);
- α è il parametro regolatore ottenuto per ottimizzazione tramite Cross Validation, che sarà discusso in seguito nei risultati in questa tesi.

4.1 Funzionamento di Prophet

Prophet usa un modello di scomposizione di serie storiche basato su frequenze di cambiamento di tre componenti principali: tendenza, stagionalità e festività[19]. Queste componenti sono combinate e legate da Prophet con la seguente equazione[19]:

$$y(t) = g(t) + s(t) + h(t) + \epsilon_t \tag{4.3}$$

Dove:

- $g(t)$ rappresenta la funzione della tendenza che modella cambiamenti non periodici;
- $s(t)$ rappresenta la funzione che modella i cambiamenti periodici;
- $h(t)$ rappresenta gli effetti delle festività, o meglio le "irregolarità" su un certo periodo;
- ϵ_t rappresenta un termine di errore per cambiamenti non previsti dal modello.

Le componenti possono essere rilevate automaticamente o configurate per fornire previsioni più accurate o flessibili in base al caso d'uso. Queste configurazioni consistono in:

- Scegliere il comportamento della funzione g, scegliendo tra quello di una funzione lineare o logistica;
- Specificare manualmente o automaticamente riguardo la funzione s i periodi su cui modellare i cambiamenti periodici a livello annuale, mensile, settimanale o giornaliero agendo tramite regressori additvi o moltiplicativi;
- Aggiungere manualmente periodi di irregolarità su uno storico come ad esempio festività o eventi locali, per non influenzare le previsioni future.

Per il lavoro di questa tesi il modello Prophet sarebbe stato adottato su una funzione g logistica per meglio rappresentare il pattern del rumore ambientale, siccome alcuni risultati ottenuti in letteratura, indicano che il rumore ambientale segue certi pattern che dipendono in modo non-lineare in funzione al tempo e alla posizione[6;16;17;18].

Per le previsioni dipendenti dalle festività, Prophet può già inserire le festività nazionali per l'Italia o altre nazioni. Inoltre è anche possibile specificare manualmente periodi o date di festività in base alla città o la nazione di riferimento, configurando i modelli per adattare il trend durante tali periodi. Un caso per questo caso d'uso per la città di Bologna potrebbe essere il 4 Ottobre per la festa di San Petronio, giorno del Patrono di Bologna.

Alcune stagionalità possono essere catturate in base alla periodicità di alcuni eventi, come nel caso del mercato storico che si svolge ogni Venerdì e Sabato in Piazza VIII Agosto, oppure in base a periodi dell'anno come i periodi scolastici.

Il frammento di codice Python che segue ne mostra un'applicazione di quest'ultimo caso: Le stagionalità settimanali e giornaliere sono gestite in automatico, e inoltre altre due stagionalità dei periodi interni ed esteri al periodo scolastico compreso tra le date 2022-09-15 e 2023-06-07, vengono gestite su un modello Prophet allenato con un certo dataset.

```
school_days_range_per_city = {
  'Bologna': [(pd.to_datetime('2022-09-15', format='%Y-%m-%d'),
               pd.to_datetime('2023-06-07', format='%Y-%m-%d'))]
}

city_name = "Bologna"
df = pd.DataFrame({
    "ds": [record["time"] for record in dataset],
    "y": [record["SPL"] for record in dataset]
})

def is_on_school_days(ds):
  is_school_day = False
  for (start, end) in school_days_range_per_city[city_name]:
      is_school_day = date.weekday() != 6 and \
          (start.date() <= date.date() <= end.date())
      if is_school_day:
          break
  return is_school_day

prophet_parameters = {
  'changepoint_prior_scale': 0.1,
  'seasonality_prior_scale': 0.05,
  'growth': 'logistic',
  'weekly_seasonality': True,
  'daily_seasonality': True,
  'seasonality_mode': 'additive'
}
model = Prophet(**prophet_parameters)

# Aggiunge la stagionalita settimanale per riflettere
# il comportamento dei periodi scolastici
model.add_seasonality(
  "weekly_on_school_days",
  condition_name="on_school_days",
  period=7, fourier_order=3,
)
model.add_seasonality(
  "weekly_off_school_days",
```

```
  condition_name="off_school_days",
  period=7, fourier_order=3
)

# Esplicita le stagionalita sul dataset
df['on_school_days'] = df['ds'] \
  .apply(is_ds_on_school_days_func)
df['off_school_days'] = ~df['ds'] \
  .apply(is_ds_on_school_days_func)

# Esplicita il range di valori ammessi (0 - 100 db) sul dataset
df["floor"] = 0
df["cap"] = 100

model = model.fit(df)
```

Codice 4.1: Codice per adattare le stagionalità scolastiche sul modello Prophet

Le eventuali variazioni avvenute durante le stagionalità, saranno riflesse con un indice di variabilità del SPL rispetto al trend per previsioni richieste su un certo periodo di tempo.

Il seguente frammento di codice mostra come valutare un modello Prophet su un'insieme di timestamp esplicitando una stagionalità di interesse.

```
def make_future_dataframe(start_date, periods, freq):
    dates = pd.date_range(
        start=start_date,
        periods=periods + 1,
        freq=freq)
    return pd.DataFrame({'ds': dates})

# Scegliamo previsioni ogni 5 minuti su tutto
# il periodo del dataset aggiungendo + 1 settimana
minutes_freq = 5
start_datetime, end_datetime = <inizio e fine periodo di previsione>
hours = 24 * 7 + (start_datetime - endt_datetime).total_seconds() / 3600

# Periodi richiesti da prevedere
future = make_future_dataframe(
  start_datetime,
  periods=int((60 / minutes_freq) * hours),
  freq=f"{minutes_freq}min"
)

# Esplicitiamo le stagionalita sui periodi richiesti
future["floor"] = 0.0
future["cap"] = 100.0
future['on_school_days'] \
  = future['ds'].apply(is_ds_on_school_days_func)
future['off_school_days'] \
```

```
  = ~future['ds'].apply(is_ds_on_school_days_func)

# Valuta il modello e fornisce le previsioni richieste
forecast = model.predict(future)
```

Codice 4.2: Codice per ottenere le previsioni di un certo periodo dal modello Prophet

Le previsioni risultanti sono state valutate su un dataframe contenenti i timestamp del periodo richiesti a intervalli di 5 minuti sul modello allenato.

Questa tecnica si può applicare sui record di tutte le strade intorno a Piazza VIII Agosto a Bologna, sul tutto il periodo coperto dal dataset considerato in questa tesi, aggiungendo una settimana di previsioni.

Il grafico risultate mostra previsioni di 24 ore includendo sabati, domeniche e fasce orarie privi di dati storici, che potrebbero mostrare variazioni non basate sui dati storici.

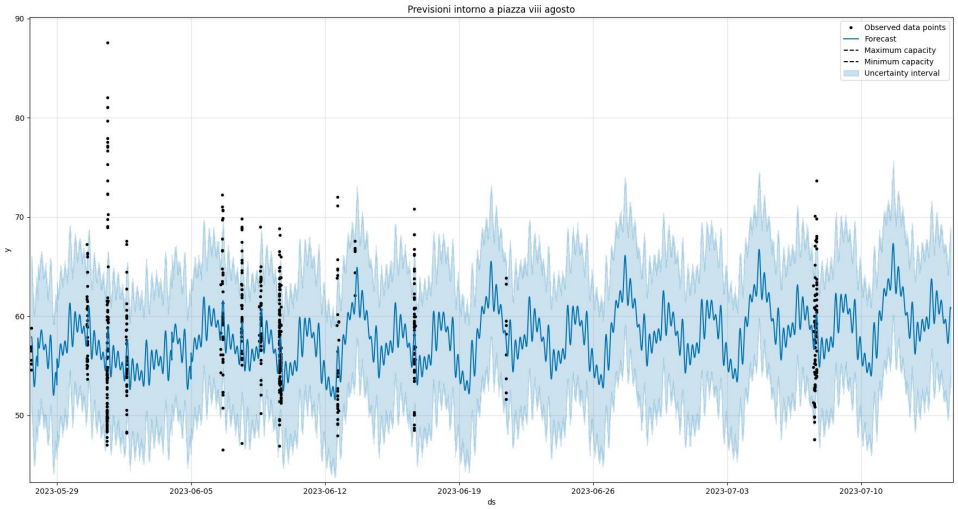


Figura 4.2: Previsione basata su strade aggregate intorno a Piazza VIII Agosto

Per avere una reale previsione basata sui dati, è necessario filtrare le previsioni richieste, e mantenere solo i giorni settimanali e fasce orarie su cui si cui il modello è stato allenato. Il grafico risultante seguente avente previsioni filtrate sulle fasce orarie 14, 15 e 16 mostra segmenti lineari interpolati tra gli estremi delle fasce orarie di giorni adiacenti che sono stati rimossi.

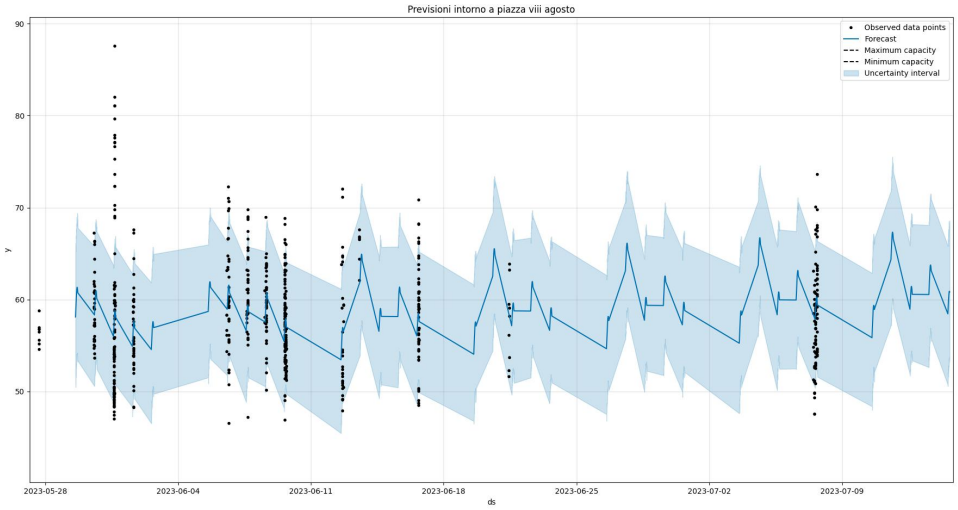


Figura 4.3: Previsione basata su strade aggregate intorno a Piazza VIII Agosto, filtrate da Lunedì a Venerdì dalle 14:00 alle 17:00

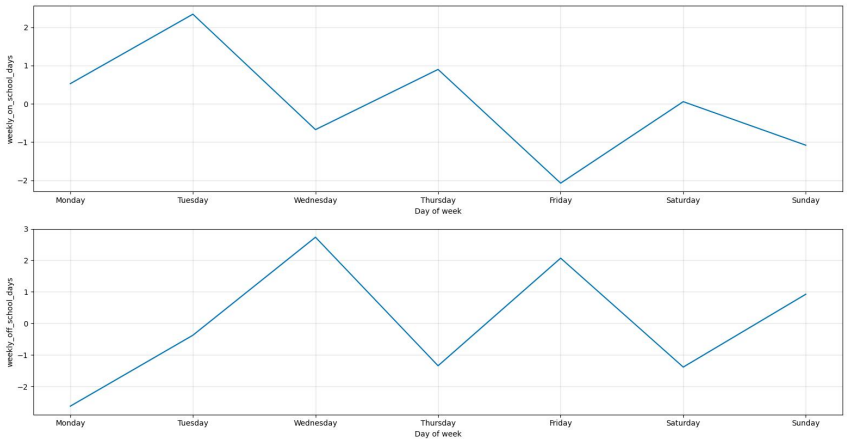


Figura 4.4: Grafico delle componenti settimanali sul periodo scolastico in Piazza VIII Agosto

Il grafico delle componenti evidenzia piccole variazioni del trend tra i periodi antecedenti e posteriori al termine delle attività didattiche. Infatti dal grafico delle previsioni, non si notano grandi variazioni nei due periodi.

Se invece si considera un'area diversa, ma nelle vicinanze di una scuola superiore, come nel caso di Via Giacomo Matteotti, si nota maggiormente una differenza sui picchi di livelli sonori in questi periodi.

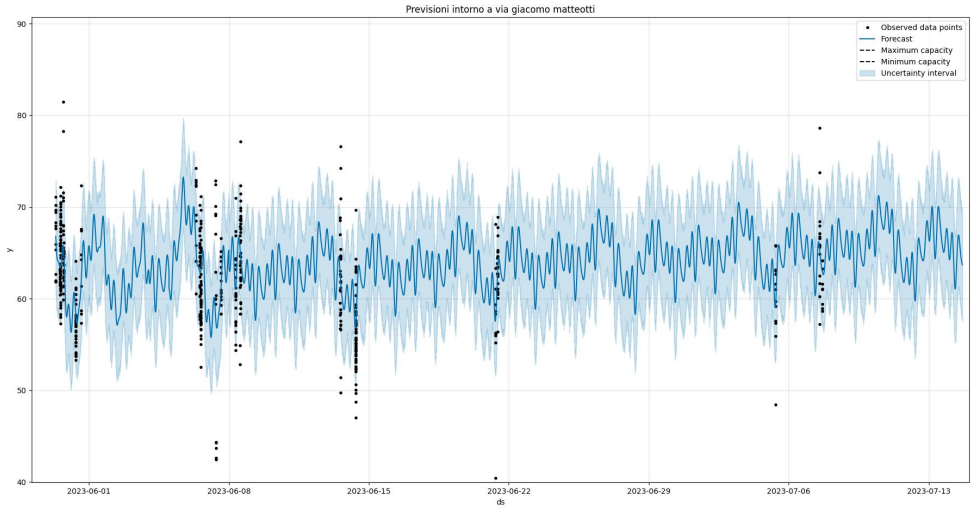


Figura 4.5: Grafico delle previsioni interne ed esterne al periodo scolastico in Via Giacomo Matteotti

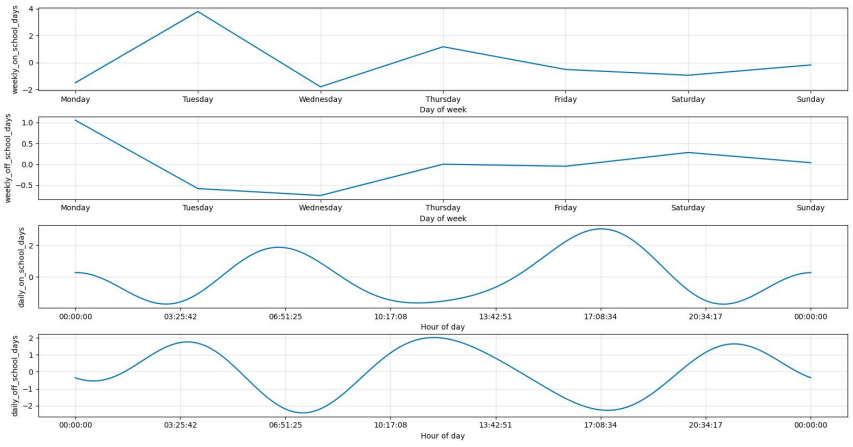


Figura 4.6: Grafico delle stagionalità scolastiche giornaliere e settimanali interne ed esterne al periodo scolastico in Via Giacomo Matteotti

4.2 Implementazione euristiche di previsione

Come già anticipato, il lavoro di questa tesi si basa sull'implementazione delle soluzioni di previsioni per i casi `A`, `B`, `C` applicati a un servizio di navigazione.

Per avere previsioni significative, sono stati regolati i parametri dei modelli Prophet per avere un pattern similare allo storico ed adattarsi in modo coerente ai cambiamenti riflessi dalle stagionalità settimanali e giornaliere riducendo l'errore di previsione. I seguenti parametri ottimali per il dataset sono stati ottenuti empiricamente tramite la tecnica di Cross Validation fornita dal tool Prophet.

```
prophet_parameters = {
  'changepoint_prior_scale': 0.01,
  'seasonality_prior_scale': 0.01,
  'growth': 'logistic',
  'weekly_seasonality': True,
  'daily_seasonality': True,
  'seasonality_mode': 'additive'
}
```

Codice 4.3: Parametri dei modelli Prophet

Le implementazioni delle soluzioni `B` e `C` richiedono una mappatura preliminare di ciascun record a un arco del grafo della città e successivamente affrontare i diversi approcci dei casi.

La mappatura record-arco consiste nel considerare la funzione di mappatura `group_records_by_around_edge` ispirata al comportamento della utility di OSMnx `osmnx.distance.nearest_edges` che si basa sull'utilizzo di un `R-tree` per indicizzare ed effettuare query su coordinate spaziali. Nello specifico `get_around_nearest_edges` associa tutti gli archi le cui geometrie intersecano o sono interne ad un cerchio con raggio di 15 metri dalla posizione di ciascun record.

```
from osmnx_utils import get_around_nearest_edges
def group_records_by_around_edge(self, graph, records):
  LATITUDE_INDEX = 0
  LONGITUDE_INDEX = 1

  # 15 meters ~ 0.00013 deg at Bologna
  meters_to_degree_distance = 0.00013

  records_points = [data["location"] for data in records]
  edges_per_record = get_around_nearest_edges(
      graph,
      [point[LONGITUDE_INDEX] for point in records_points],
      [point[LATITUDE_INDEX] for point in records_points],
      max_distance=meters_to_degree_distance
  )

  # Map records to edge keys
```

```
edge_records = dict()
for index, edge_list in enumerate(edges_per_record):
    for edge in edge_list:
        if edge not in edge_records:
            edge_records[edge] = []
        edge_records[edge].append(records[index])

return edges_per_record, edge_records
```

Codice 4.4: Codice di mappatura di ciascun record su un arco del grafo

I modelli Prophet, per poter creare previsioni affidabili necessitano di avere un certo ammontare di record storici in base al tipo di previsioni temporali richieste. Infatti per poter avere previsioni settimanali si dovrebbe avere dati che coprono almeno ogni giorno della settimana per almeno 1 settimana; mentre per aver previsioni giornaliere si dovrebbe avere dati che coprono ogni ora del giorno. Per questi motivi, i modelli associati alle soluzioni `B` e `C` è necessario che siano valutati su giorni settimanali e fasce orarie su cui sono stati allenati.

Considerate queste premesse, si è scelto di considerare una condizione di affidabilità per entrambe le soluzioni `B` e `C` che indica se un modello Prophet può essere considerato affidabile in funzione di un periodo temporale da prevedere:

Un modello allenato con un sottoinsieme H del dataset associato si ritene affidabile per una fascia oraria T e un giorno della settimana D richiesti se:

1. H copre almeno 2 settimane di record nella fascia oraria T dei giorni D
2. oppure se copre almeno 1 settimana di record nella fascia oraria T nel giorno D e vale una delle seguenti due condizioni:
    - H copre almeno 2 giorni della settimana diversi con almeno 2 settimane ciascuno coperte nella fascia oraria T
    - H copre almeno 3 giorni della settimana diversi con almeno 1 settimana ciascuna coperta nella fascia oraria T

Soluzione A

L'implementazione della soluzione `A` consiste nel recuperare dal database di riferimento le misurazioni campionate durante una fascia T contemporanea all'inoltro della richiesta, mapparle su i corrispettivi archi del grafo, e fornire la previsione calcolata con la formula 1.2.

Soluzione B

L'implementazione della soluzione `B` consiste nell'allenamento di un modello Prophet per ciascun arco del grafo con il relativo sottoinsieme di record del dataset precedentemente mappato.

L'istanziamento e allenamento del modello Prophet avviene in modo analogo al frammento di codice (4.1) con il relativo dataset storico e i parametri precedentemente specificati (4.3).

A questo punto l'ultima operazione consiste nel valutare il modello per richiedere le previsioni dei livelli di pressione sonora su un certo periodo a intervalli di tempo di tempo fissati. La valutazione avviene in modo analogo al codice (4.2) e se un modello fosse allenato principalmente su alcune fasce orarie, saranno valutate solo su orari e giorni delle settimana su cui il modello è stato allenato.

Soluzione C

Siccome l'euristica di questo caso dipende dagli archi adiacenti, la componente $I(T)$ considera solo gli archi per cui il modello è stato allenato sulla fascia T, ed in modo analogo deve valere anche per il modello della categoria stradale associato alla componente $C(T)$.

La costruzione di un modello Prophet basato su ogni categoria stradale avviene considerando la mappatura di ciascun record del dataset con la stessa categoria stradale etichettata della proprietà `highway` relativa a un segmento stradale su OpenStreetMap (vedere[24]).

L'allenamento e la valutazione di questo modello Prophet avviene in modo analogo ai frammenti di codice 4.3, 4.1 e 4.2.

Capitolo 5

Implementazione

5.1 Server - Servizio di navigazione

Il servizio di navigazione è stato implementato come applicazione server basata sul framework Flask in python.

Il server fornisce un API REST con un unico endpoint: `/route` con le seguenti specifiche:

Parametri di query:

- `src`: Le coordinate del punto di partenza, da fornire come array nel formato [latitudine, longitudine];
- `dest`: Le coordinate del punto di arrivo, analogo a `src`
- `nav_mode`: La modalità di percorrenza del percorso per poter considerare strade adatte, da fornire come uno tra i valori `walk`, `bike`, `drive`;
- `loudness`: Percentuale di preferenza della quiete espresso tra 0 e 1.

Output:

- codice 404: Indica che la città appartenente alle coordinate non è stata mappata
- codice 200: Restituisce in formato JSON un oggetto rappresentante il percorso consigliato:

```
{
  "nodes": Lista di coordinate in formato [latitudine, longitudine]
    che rappresentano ogni segmento stradale del percorso.
  "geometries": Lista di geometrie associate a ciascun segmento.
```

```
        Ogni geometria è rappresentata come lista di coordinate in sequenza.
    }
```

L'implementazione server dipende dalle funzionalità del pacchetto Python `OSMnx` per ottenere e modellare dati geografici di OpenStreetMap in grafi. `OSMnx` è basato sulle API del pacchetto Python `NetworkX` per creare e manipolare i grafi, mentre si affida a OpenStreetMap e le API `Nominatim` per ottenere informazioni e metadati riguardo la rete stradale.[15;21]

Durante la fase di inizializzazione per ogni città del dataset viene inizialmente istanziato un multi grafo non orientato, mappato con le previsioni preliminari ottenute dalle valutazioni dei modelli nelle modalità descritte nel capitolo 4.

Successivamente quando il servizio è stato inizializzato, esso rimane in ascolto sul endpoint `/route`. Quando questo endpoint riceve una richiesta, la città di riferimento è decodificata interrogando il servizio `Nominatim` con le coordinate associate ai parametri `src` e `dest`. Da queste viene quindi considerato il multi grafo non orientato della città mappato con le previsioni dei modelli e la sua versione orientata su cui saranno applicate le euristiche relative alle soluzioni `A`, `B` e `C` ed eseguito l'algoritmo di Dijkstra.

Durante questa fase si considerano le previsioni preliminarmente ottenute a intervalli periodici di 5 minuti, ma filtrate sul periodo della fascia oraria in cui viene effettuata la richiesta. Dunque su queste si applicano le euristiche di previsione 4.1 e 4.2.

L'implementazione di questa logica per quanto riguarda le soluzioni `B` e `C` è sintetizzata nei seguenti frammenti di codice.

```
def get_Leq(forecast, start_ds, end_ds):
  forecast = forecast[
    (forecast["ds"] >= start_ds) & (forecast["ds"] < end_ds)
  ]
  SPL = forecast["yhat"]
  return 10.0 * math.log10(np.mean(10.0 ** (SPL / 10.0)))

def apply_case_B(start_datetime, end_datetime):
  # Assegnati in fase di inizializzazione
  # Edge-forecast map
  edge_forecasts \
    = nx.get_edge_attributes(graph, "edge_forecast")

  def get_edge_forecast_value_func(edge, forecast):
    return get_Leq(forecast, start_datetime, end_datetime)

  case_b_predictions_per_edge = get_loudness_predictions_for_edge(
    edge_forecasts, get_edge_forecast_value_func
  )
```

```
  nx.set_edge_attributes(graph, \
    values=case_b_predictions_per_edge,
    name="edge_spl_prediction"
  )
```

Codice 5.1: Codice applicazione soluzione B

```
def apply_case_C(start_datetime, end_datetime):

  edge_highways = nx.get_edge_attributes(graph, "highway")
  # Highway-forecast map
  highway_forecasts = get_graph_highway_forecasts(graph)

  # Assegna Leq SPL categoria
  def get_highway_forecast_value_func(highway, forecast)
    return get_Leq(forecast, start_datetime, end_datetime)

  nx.set_edge_attributes(graph, \
    name="highway_spl_prediction",
    values=get_loudness_predictions_for_edge_highway(
        edge_highways, highway_forecasts,
        get_highway_forecast_value_func,
        ignored_edges = predictions_per_edge.keys()
    )
  )

  # Assegna Leq SPL interpolato
  def get_edge_prediction(edge, data):
    return data.get("edge_spl_prediction", None)

  case_c_edges = graph.edges() - case_b_predictions_per_edge.keys()

  nx.set_edge_attributes(graph,\
    name = "edge_spl_interpolated",
    values = aggregate_adjacent_edges(
      graph, case_c_edges,
      get_edge_value_func = get_edge_prediction
      aggregator_func = np.mean
    )
  )
```

Codice 5.2: Codice applicazione soluzione C

Infine le previsioni finali vengono normalizzate in un valore che bilancia quiete e distanza in funzione alla preferenza di quiete $0 \leq \beta \leq 1$ fornita, applicando la seguente combinazione convessa:

$$W(e,T) = L(e,T)\beta + D(e)(1-\beta), \tag{5.1}$$

dove, rispetto all'intervallo di tempo T e arco del grafo e, $D(e)$ = Lunghezza in metri del segmento e e $L(e,T)$ è definita con le modalità viste nel capitolo 4, seguendo l'ordine di priorità indicato in figura 4.1:

$$
\begin{aligned}
D(e) &= \text{Lunghezza in metri del segmento } e \\
L(e,T) &= L_{eq} \text{ previsto dalle euristiche su } e \\
&= \begin{cases}
L_{eq,e}(T) \text{ se } e \text{ è in caso A,} \\
L_{eq,e}(5min,T) \text{ se } e \text{ è in caso B,} \\
WINH_e(T) \text{ se } e \text{ è in caso C,} \\
I_{adj(e)}(T) \vee C_e(T), \text{ se disponibile} \\
50 \text{ (default), altrimenti}
\end{cases}
\end{aligned}
\tag{5.2}
$$

Quello che si è ottenuto è un peso $W(e,T)$ applicato a ciascun arco del grafo, che bilancia distanza e livello del rumore ambientale previsto dalle euristiche applicabili.

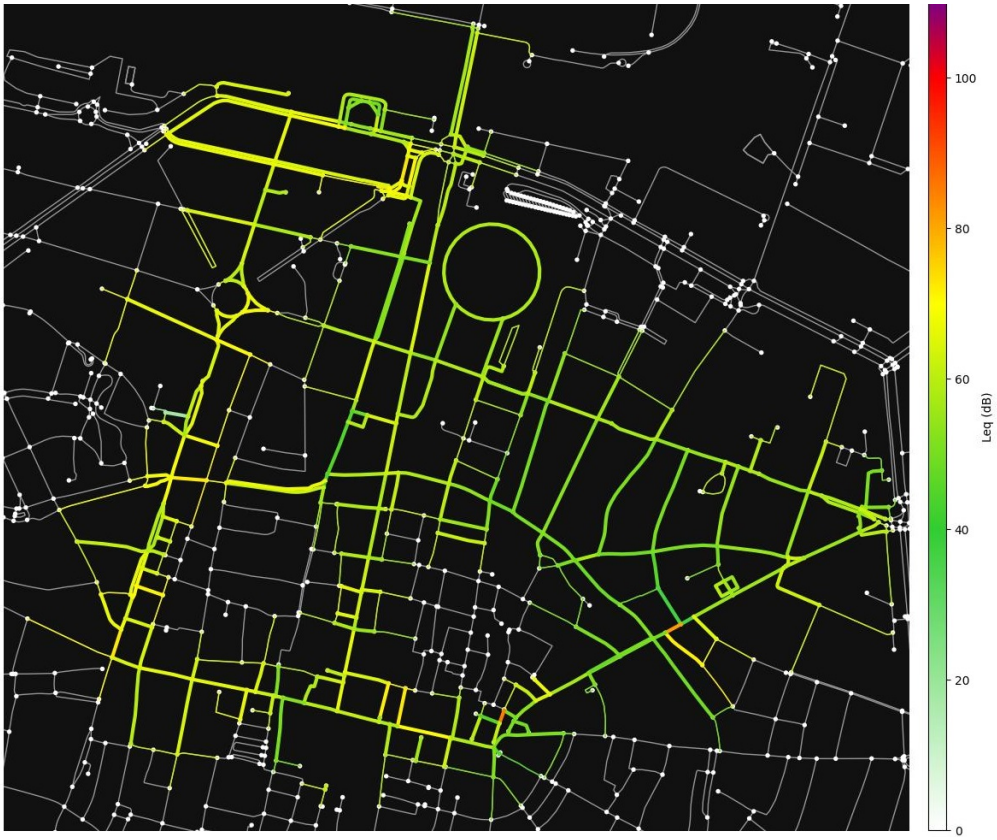


Figura 5.1: Heatmap su livello equivalente di rumore ambientale a Bologna previsto in data 21/07/2023 per la fascia oraria T=15:00

A questo punto basta eseguire l'algoritmo di Dijkstra sul multi grafo orientano avente i pesi $W(e,T) \geq 0$ ottenendo la sequenza di nodi che costituiscono il percorso di peso minimo da un nodo sorgente ad un nodo destinazione. L'algoritmo è già fornito da OSMnx come wrapper alla funzione `nx.shortest_path` del pacchetto Python `NetworkX`.

```
multi_di_graph = osmnx.graph_from_place(
  { "city": city_name, "country": country_name },
  network_type=navigation_type,
  simplify=True, retain_all=False
)
multi_graph = get_graph_with_forecasts(multi_di_graph)

apply_graph_weights_from_graph(
  src_graph = multi_graph, dest_graph = multi_di_graph,
  get_weight_func(user_preference)
)

src_node = osmnx.distance.nearest_nodes(multi_di_graph, src_point)
dest_node = osmnx.distance.nearest_nodes(multi_di_graph, dest_point)

route = osmnx.distance.shortest_path(multi_di_graph, \
  src_node, dest_node, weight="weight")
```

Codice 5.3: Codice rissuntivo con applicazione algoritmo di Dijkstra sul grafo

Infine tramite il percorso ottenuto con l'algoritmo di Dijkstra, vengono ricavate le coordinate e le geometrie relative ai segmenti dell'intero percorso, per fornirli entrambi come risposta alla richiesta del client.

5.2 Client

Il client del servizio di navigazione consiste in un'applicazione per smartphone capace di visualizzare una mappa geografica di una città e interagire su di essa per specificare meta e posizione di partenza di un percorso.

Il client è stato implementato come applicazione Android che sfrutta le funzionalità della libreria `osmdroid`. Questa libreria ha permesso infatti di integrare una mappa interagibile ed esplorabile con le mappe geografiche fornite sempre da OpenStreetMap [25].

Le interazioni sulla mappa aggiungono o rimuovono fino a due marcatori associabili alle posizioni di partenza e di arrivo.

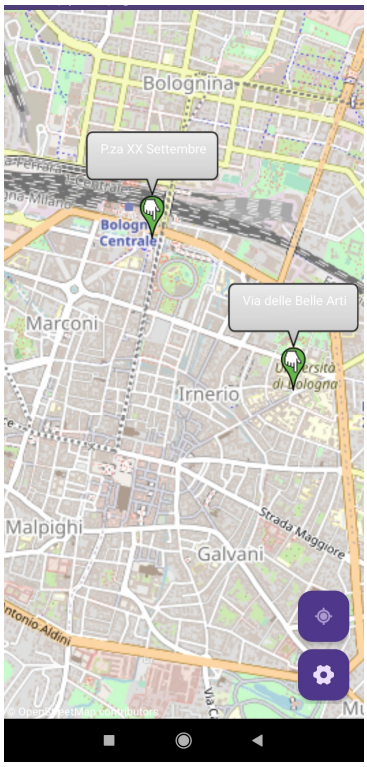


Figura 5.2: Anteprima della mappa con i marcatori

Una sezione dedicate alle impostazioni dell'applicazione permette di scegliere alcune preferenze riguardo la ricerca del percorso ottimale. Tra queste prefereze c'è il modo o il mezzo con cui l'utente vuole percorre il percorso tra cui guidare un veicolo, andare in bicicletta o camminare. Dal punto di vista del calcolo del percorso, questa preferenza è importante per considerare solo tragitti compatibili con il mezzo usato, questo è reso possibile grazie alle informazioni semantiche dei segmenti stradali indicati dalle contribuzioni su OpenStreetMap.

Inoltre è possibile scegliere un parametro del percorso come indice di preferenza tra quiete e lunghezza del percorso. Questa preferenza è esprimibile tramite un selettore di questo parametro in percentuale da 0 a 100. Un valore vicino a 100 indica una preferenza per il livello di quiete, che comporta una considerazione principalmente di percorsi con bassi livelli di rumore ambientale al costo di potenziali tragitti più lunghi; viceversa un valore vicino a 0 indica una preferenza per percorsi brevi, al costo di potenzialmente percorrere strade particolarmente rumorose.

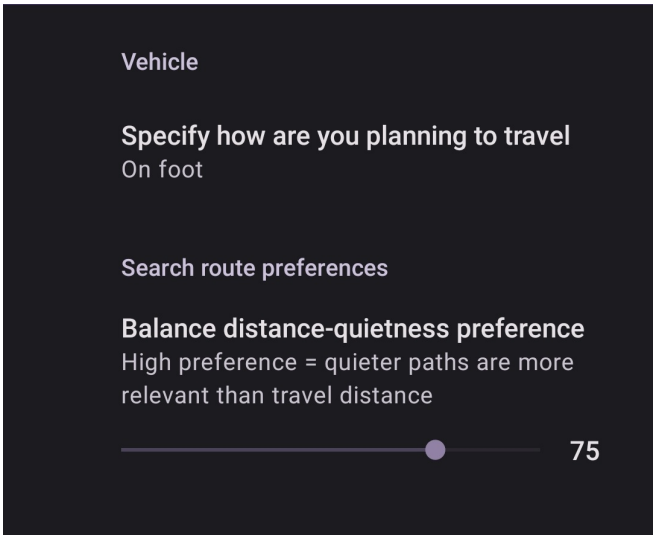
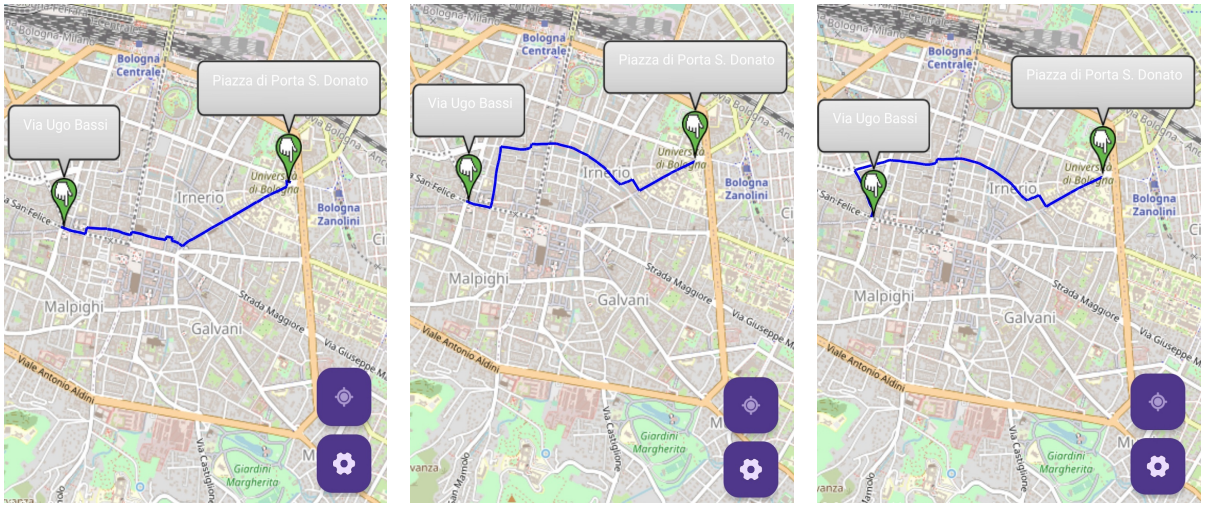


Figura 5.3: Sezione delle preferenze di ricerca del percorso

Quando l'utente ha espresso una preferenza dei parametri di ricerca e posizionato i marcatori sulla mappa, un client HTTP effettua la richiesta all endpoint discusso nella sezione 5.1 e procede a calcolarne il percorso ottimale con i parametri e le modalità specificate. In seguito alla ricezione della risposta, sulla mappa sarà visualizzato il tracciato del percorso consigliato seguendo le geometrie relative a ciascun arco del grafo coinvolto.

Una dimostrazione pratica è stata svolta, richiedendo un percorso ottimale percorribile in bici a Bologna, specificando un punto di partenza in Via Ugo Bassi e un punto di destinazione in Piazza Porta San Donato.



(a) Preferenza quiete 0% (b) Preferenza quiete 50% (c) Preferenza quiete 100%

Figura 5.4: Percorsi suggeriti da Via Ugo Bassi a Piazza Porta San Donato su diverse preferenze di quiete alle ore 15

I percorsi suggeriti in figura 5.4 su vari valori di preferenza, evidenziano chiaramente percorsi diversi: Una preferenza di quiete $\beta = 0.0$ (figura 5.4a), indica all'algoritmo di Dijkstra di minimizzare solo la distanza del percorso ignorando la quiete sul percorso, invece al contrario una preferenza $\beta = 1.0$ (figura 5.4c) indica di minimizzare solo il rumore totale previsto, prolungando eventualmente il tragitto totale. Una preferenza $\beta = 0.5$ (figura 5.4b) bilancia equamente distanza e livello di quiete su ciascun segmento stradale; infatti il percorso risultante evita di prolungare troppo il percorso, ma considerando percorsi non particolarmente rumorosi.

Via dell'Indipendenza, Via Ugo bassi e Via Rizzoli sono quotidianamente caratterizzate dalla presenza di flussi notevoli di persone e di mezzi pubblici nel traffico stradale, influenzando il rumore anche nei segmenti interconnessi. I livelli di rumore ambientale previsti in figura 6.3 confermano questa considerazione e la figura 5.4a indica infatti che esse fanno parte del percorso più diretto per raggiungere Piazza Porta San Donato. Al contrario Via Augusto Righi e Via delle Moline sono molto meno trafficate e perciò considerate maggiormente per le preferenze $\beta \geq 0.5$.

Si vuole osservare inoltre un fatto interessante per quanto riguarda la preferenza $\beta = 1.0$ nel caso in esame: Piuttosto che tagliare il percorso per Via dell'Indipendenza, come accade per $\beta = 0.5$, il percorso viene allungato su vie alternative per poi passare per Via Indipendenza, con lo scopo di percorrere Via Augusto Righi passando per pochissimi segmenti stradali in Via dell'Indipendenza, riducendo il rumore ambientale a cui si sarebbe esposti.

Capitolo 6

Risultati

In questo capitolo saranno discusse le tecniche di analisi e i risultati ottenuti riguardanti le euristiche di previsione adottate per le soluzioni ai casi `B` e `C`.

Le soluzioni `B`, `C` come già discusso nei precedenti capitoli di questa tesi, dipendono abbastanza dal modello di previsione adottato dallo strumento Prophet, e dunque la loro precisione dipende da una corretta parametrizzazione di questo affinché il modello riesca ad adattarsi quanto possibile al pattern del rumore ambientale del dataset storico. I parametri ottimali generalizzati che sono stati considerati per questo lavoro (4.3) dipendono da uno storico di dati limitato, e dunque in questo capitolo saranno affrontati analisi e discussioni più da un punto di vista generalizzato, sugli esperimenti condotti.

6.1 Analisi euristiche di previsione

Per quanto riguarda la soluzione `C` è possibile confrontare la precisione e performance dell'euristica `WINH` (4.2) al variare del parametro α.

Per fare questo è stata applicata una tecnica di Leave-One-Out Cross Validation (LOOCV) su un sottoinsieme di archi che rispettano la condizione di affidabilità precedentemente descritta per una fascia oraria T nel giorno della settimana D.

Questa tecnica è stata inoltre applicata anche a un'euristica "naive" per metterla a confronto con l'euristica `WINH` (4.2) sugli stessi archi e dataset coinvolti.

L'euristica "naive", nominata da qui in poi `Random(min, max)`, è definita come:

$$Random_{min,max}(T) := random(\min_{\forall e \in adj(e_{ref})} L^e(T, D), \max_{\forall e \in adj(e_{ref})} L^e(T, D)) \tag{6.1}$$

dove:

- e_{ref} è un arco di riferimento
- $adj(e_{ref})$ è l'insieme degli archi adiacenti a e_{ref}

- $L^e(T, D)$ è il sottoinsieme del dataset mappato sull'arco e costituito dai valori SPL storici. Se tale storico su T ha un sottoinsieme di dati per il giorno della settimana D, allora tale sottoinsieme viene considerato in modo prioritario, altrimenti sono considerati i dati storici nella fascia T per i giorni della settimana coperti.

La tecnica LOOCV è stata applicata con particolari modalità su test e training set con il fine di misurare l'errore medio dell'euristica su ciascun arco, sottoponendola a una condizione "di stress" senza il contributo di un arco adiacente.

Il dataset considerato è costituito dagli archi per cui la condizione di affidabilità è valida su una fascia oraria T e giorno D per il sottoinsieme del dataset associato; per cui saranno considerate le previsioni $L_{eq,10min}(T)$ fornite dall'euristica `WINH` (4.1) per ciascun arco considerato.

l'euristica `WINH` considera in una componente il contributo di tutti gli archi adiacenti; dunque si vuole misurare le prestazioni su un certo arco e, facendo finta che un arco $adj(e)_i$ adiacente non abbia record associati.

Dunque durante ogni iterazione di verifica dell'arco, ciascun arco del training set esclude i record che influenzano e (che si sta verificando) e quelli che influenzano $adj_i(e)$. Questa iterazione, in termini di archi esclusi dal training set, è equivalente alla iterazione "opposta" dove si vuole testare l'euristica sul'arco $adj_i(e)$, facendo finta che l'arco e adiacente non abbia record associati. Dunque in entrambi i casi il training set avrebbe dimensione N-2 dove N sono tutti gli archi del grafo considerati, e il test set dai 2 archi esclusi.

Considerata questa introduzione, si definisce la tecnica LOOCV con le seguenti specifiche semplificate:

Si consideri gli archi per cui la condizione di affidabilità è valida su una fascia oraria T e giorno D per il sottoinsieme del dataset associato; per cui saranno considerate le previsioni $L_{eq,10min}(T)$ fornite dall'euristica `WINH` (4.1) per ciascun arco considerato.

Il dataset considerato è costituito dall'insieme delle combinazioni semplici di coppie di archi tra loro adiacenti con le caratteristiche sopra descritte.

Per ogni iterazione l'elemento di test è identificato dalla la coppia di archi adiacenti di test facente parte del dataset. Il training set considerato invece è costituito dagli archi facenti parte del dataset escludendo quelli verificati durante l'iterazione, dove ciascun arco mantiene la mappatura del subdataset associato, escludendo i record che sono mappati su gli archi di test associati all'iterazione.

Durante ogni iterazione, è necessario aggiornare dunque il subdataset per alcuni archi del training set che hanno un sottoinsieme di record in comune con gli archi di test durante una certa iterazione. Si necessita di istanziare nuovi modelli con il sottoinsieme del dataset senza i record coinvolti su gli archi di test, per poi ripristinarli all'iterazione successiva.

Se i nuovi modelli non rispettano la condizione di affidabilità, allora vengono esclusi dalle valutazioni durante quella iterazione. Chiaramente queste modalità avvengono sia per i modelli degli archi che per i modelli delle categorie stradali; dunque ignorando eventualmente i relativi archi e categorie interessati.

Al termine di ogni iterazione vengono raccolte le metriche dei valori osservati e previsti per ciascun arco di test su entrambe le euristiche; dunque includendo i risultati al variare di 21 parametri α equidistanti nell'intervallo tra 0 e 1.

Questo esperimento è stato condotto su 3 tipologie di dataset: uno nel centro di Bologna rilevato con l'applicazione "Tracker", e due dataset pubblici del progetto Noise-Planet[26] associati ad aree delle città di Lyon e Parigi.

Il progetto Noise-Planet popola il suo database di livelli di rumore ambientale tramite l'applicazione "NoiseCapture" [8]. Quest'ultima misura il livello medio di pressione sonora equivalente ponderato con un filtro `B` ($L_{eq,A}(T)dB(A)$) a intervalli di $t = 1$ *secondi*, il che differisce da quello applicato in questo lavoro dall'applicazione "Tracker". Nonostante questo i concetti di applicazione delle euristiche rimangono valide e applicabili in modo analogo.

I modelli Prophet allenati su questi dataset, per semplicità non considerano le stagionalità dei periodi scolastici precedentemente applicati nella sezione 4.1, ma considerano le festività nazionali dei relativi paesi già configurate da Prophet.

Per le città di Bologna e Lyon sono stati considerati per semplicità i multi grafi non orientati su tracciati che OSMnx considera percorribili in bici, cioè una rete stradale di tipo `bike`, mentre per la città di Parigi è stata considerata una rete stradale percorribile a piedi, di tipo `walk`.

Inoltre sono stati considerati sottoinsiemi di dataset costituiti con circa lo stesso numero di settimane di campionature paragonabili al dataset di Bologna.

Esperimento su Bologna

Il primo esperimento è stato condotto in particolare nel dataset di Bologna effettuando previsioni per venerdì 21/07/2023 nelle fasce orarie $T = 14, 15, 16$ (14:00-17:00) considerando le valutazione dei modelli Prophet su intervali di $t = 10$ *minuti*. Il particolare è stato condotto su 315 archi che rispettano la condizione di affidabilità per le fasce orarie nel giorno specificato. Il seguente grafico mostra l'errore quadratico medio (RMSE) per ciascun arco applicando l'euristica `WINH` (4.2) al variare del parametro α.

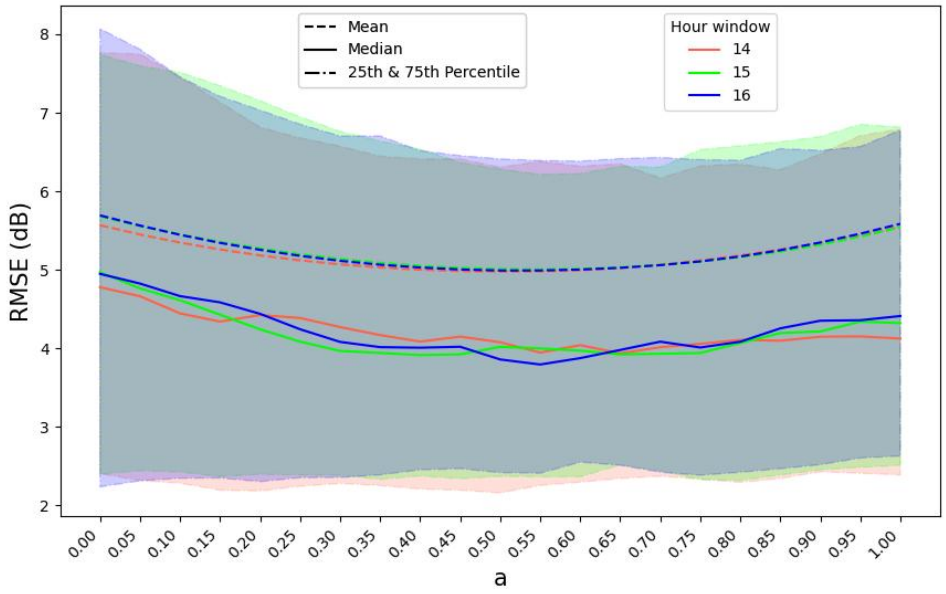


Figura 6.1: Lineplot RMSE sull'euristica WINH durante le fasce orarie 14, 15, 16 al variare del parametro α sperimentato su Bologna

T	α	**RMSE medio (dB(A)**
14	0.5	4,98
15	0.55	5.0
16	0.5	4.99

Tabella 6.1: α e fasce orarie per cui si ha un RMSE medio minimo sull'esperimento di Bologna

Categorie	Perc. degli archi
cycleway	10.2%
living_street	2.5%
pedestrian	15.6%
primary	5.7%
residential	28.7%
secondary	11.1%
service	7.6%
tertiary	22.6%

Tabella 6.2: Categorie di OpenStreetMap e relative percentuali degli archi considerati per l'esperimento su Bologna

I grafici dei risultati, mostrano che il parametro α è molto simile per tutte le fasce orarie con un RMSE medio intorno a 5 dB; dunque le componenti $I(T)$ e $C(T)$ dell'euristica `WINH` (4.2) sono quasi bilanciate. D'altro canto però si può notare che l'euristica produce una varianza notevole tra tutti gli archi coinvolti.

Considerati i valori RMSE per $\alpha = 0.5$ fissato, sono stati messi a confronto con i risultati ottenuti con l'euristica `Random(min, max)`, riassunti dai seguenti grafici.

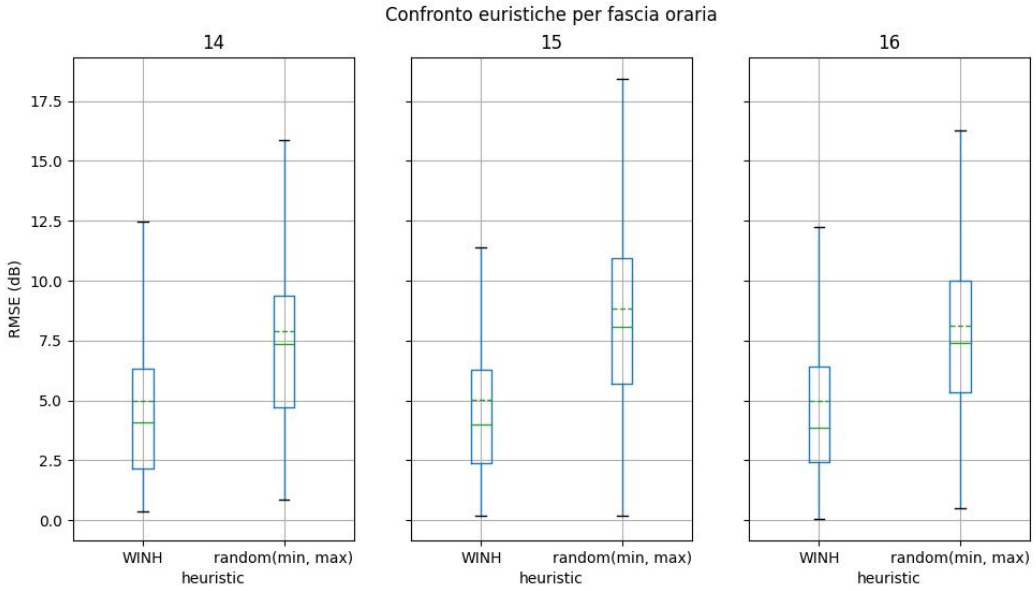


Figura 6.2: Boxplot RMSE delle euristiche WINH e random(min, max) messe a confronto e raggruppate per fascia oraria sperimentato su Bologna

Dai grafici si può notare che l'euristica `random(min, max)` ha mediamente un valore $7.9 \leq RMSE \leq 8.8$ maggiore rispetto all'euristica `WINH` (4.2); dunque quest'ultima rimane l'euristica più affidabile.

Per avere un'idea della distribuzione dell'errore a livello geografico, i seguenti grafici Heatmap confrantano i valori RMSE delle due euristiche per ciascun arco considerato e in ogni fascia oraria T.

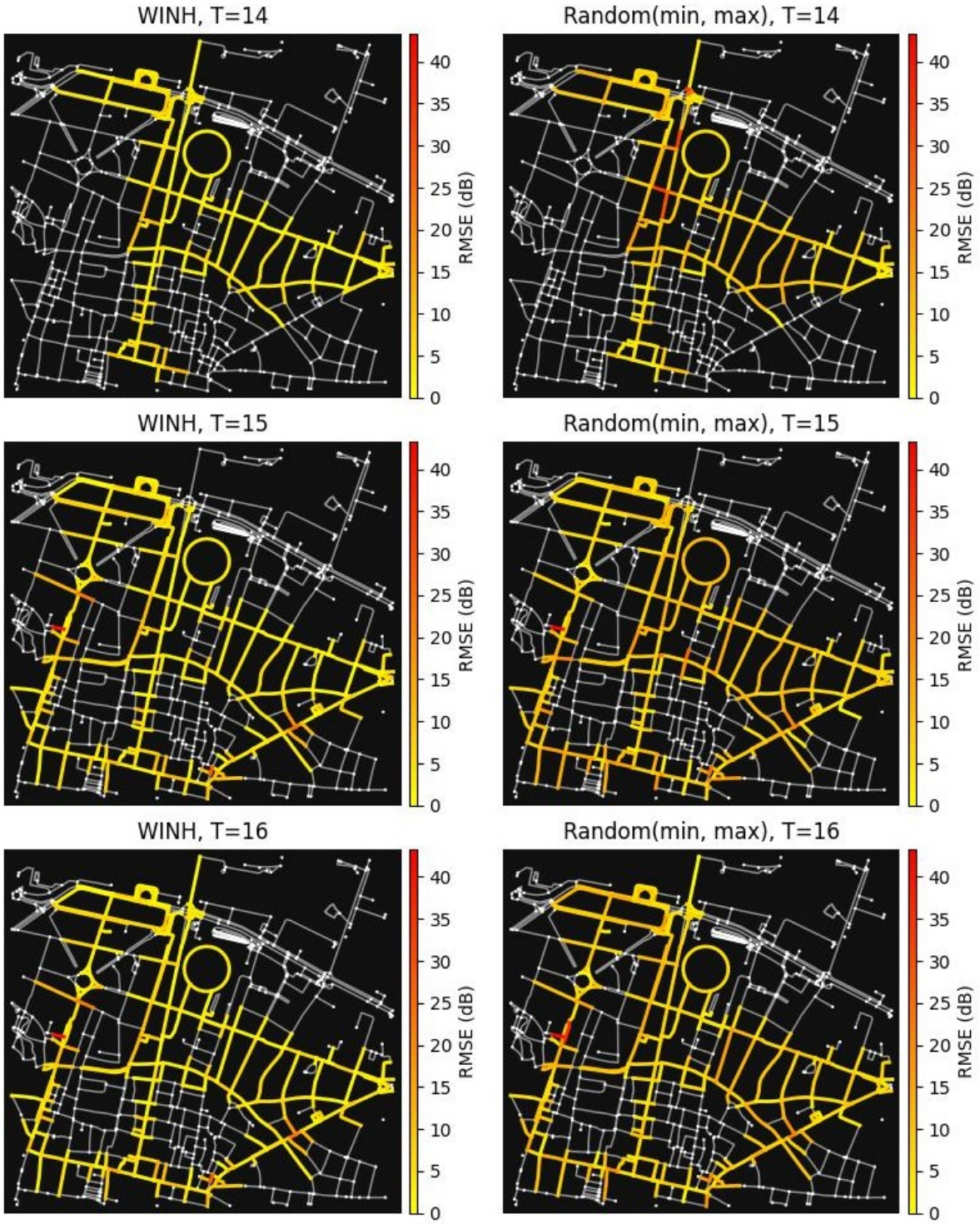


Figura 6.3: Grafici Heatmap del grafo di Bologna sui valori RMSE raggruppati per fascia oraria ed euristica ($\alpha = 0.5$)

Esperimento su Lyon

Un test analogo è stato condotto considerando un sottoinsieme di un dataset riguardante la città francese Lyon, ottenuto dalla piattaforma Noise-Planet.

In particolare è stata considerata una zona centrale della città dove il sottoinsieme del dataset associato comprende misurazioni tra il 5 marzo 2018 e il 15 maggio 2018 con circa 4 settimane di dati rilevanti per la zona individuata che coprono quasi tutti i giorni della settimana ma con maggior distribuzione su martedì. L'esperimento è stato svolto su previsioni per martedì 22 maggio 2018 nelle fasce orarie $T = 11, 12$ su 37 segmenti stradali considerati.

I risultati dell'esperimento sono riportati dai seguenti grafici:

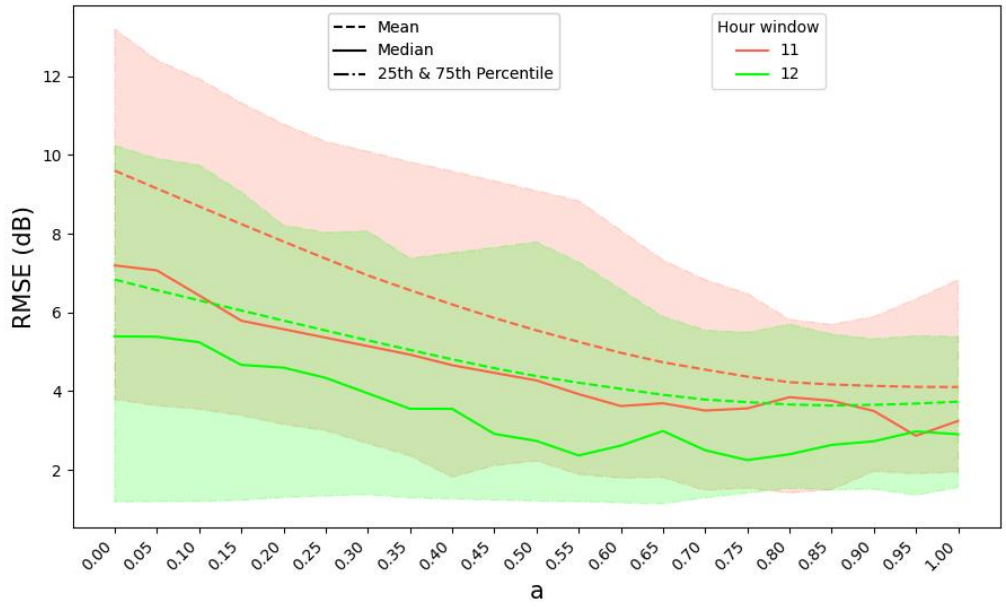


Figura 6.4: Lineplot RMSE sull'euristica WINH durante le fasce orarie 11, 12 al variare del parametro α sperimentato su Lyon

T	α	**RMSE medio (dB)**
11	1.0	4.11
12	0.85	3.68

Tabella 6.3: α e fasce orarie per cui si ha un RMSE medio minimo sull'esperimento di Lyon

Categorie	Perc. degli archi
cycleway	35.1%
path	29.7%
pedestrian	2.7%
primary	5.4%
residential	29.7%

Tabella 6.4: Categorie di OpenStreetMap e relative percentuali degli archi considerati per l'esperimento su Lyon

Questa volta Il parametro α ottimale in entrambe le fasce tende verso 1.0, indicando dunque che l'errore risulta minore se si considera prevalentemente il valore interpolato con gli archi adiacenti, piuttosto che la componente dovuta alla categoria stradale.

Questo risultato è probabilmente dovuto al fatto che gli archi considerati in questo esperimento erano pochi e con poca diversificazione delle categorie stradali associate. Infatti gli archi considerati sono principalmente associati a un'unica strada e strade ad esse parallele come si può notare dalla tabella 6.4 e dai grafici 6.6.

Fissando un valore intermedio $\alpha = 0.9$ su entrambe le fasce orarie, il seguente grafico evidenzia come l'euristica `WINH` (4.2) rimane più precisa rispetto l'euristica `random(min, max)`.

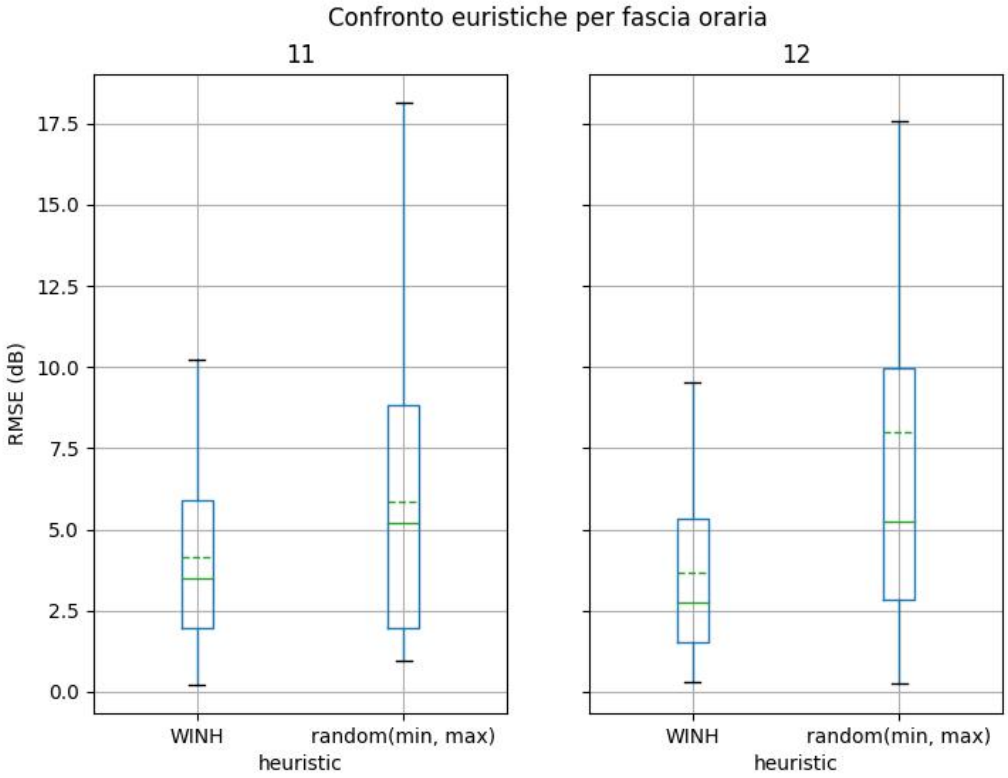


Figura 6.5: Boxplot RMSE delle euristiche WINH e random(min, max) messe a confronto e raggruppate per fascia oraria sperimentato su Lyon

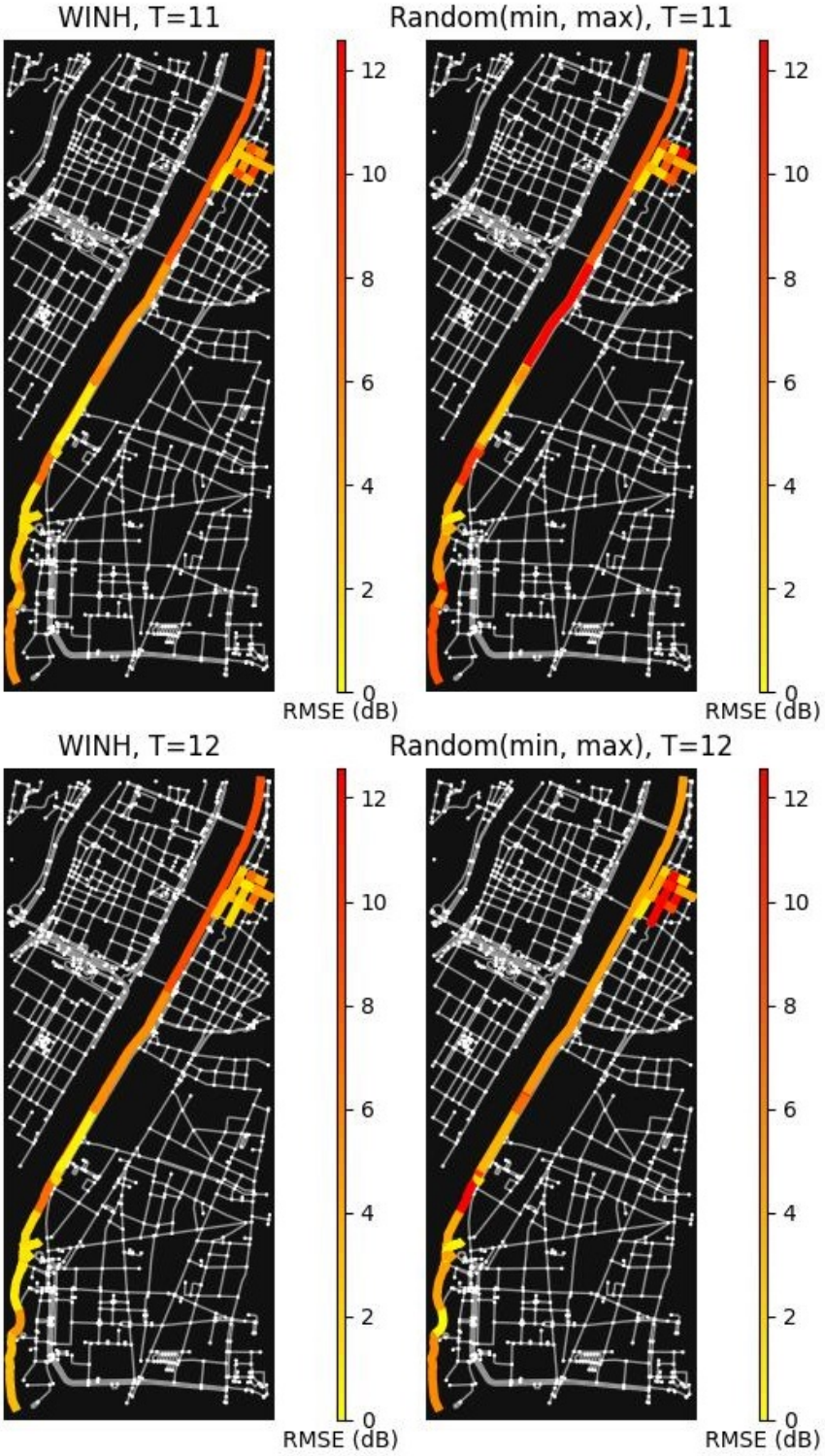


Figura 6.6: Grafici Heatmap del grafo di Lyon sui valori RMSE raggruppati per fascia oraria ed euristica ($\alpha = 0.9$)

Esperimento su Parigi

Un test analogo è stato condotto considerando un sottoinsieme del dataset mappato a Parigi, ottenuto da Noise-Planet in modo analogo all'esperimento su Lyon.

In particolare sono stata considerata l'area che comprende 6 Arrondissment o circondai municipali adiacenti: 5, 6, 7, 13, 14, 15. In particolare dove il sottoinsieme del dataset associato comprende misurazioni tra il 9 ottobre 2017 e il 12 novembre 2017 con circa 4 settimane di dati rilevanti per la zona individuata che coprono quasi tutti i giorni della settimana ma con maggior distribuzione su giovedì. L'esperimento è stato dunque svolto su previsioni per giovedì 16 novembre 2017 nella fascia oraria $T = 8$ su 411 segmenti stradali considerati di un grafo avente una rete stradale percorribile a piedi, associate alla rete `walk` fornita da OSMnx.

I risultati dell'esperimento sono riportati dai seguenti grafici:

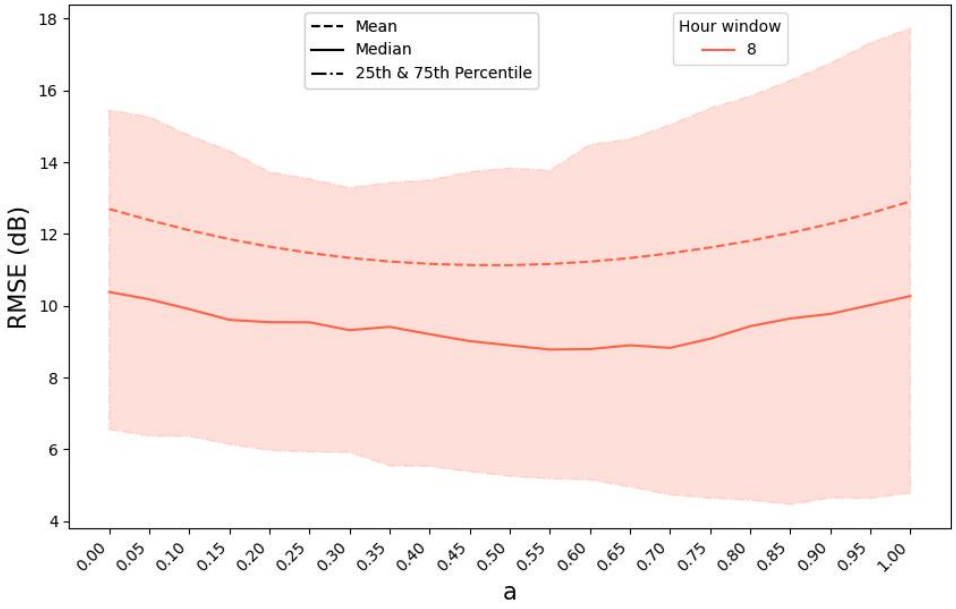


Figura 6.7: Lineplot RMSE sull'euristica WINH durante la fascia oraria 8 al variare del parametro α sperimentato su Parigi

T	α	**RMSE medio (dB)**
11	0.5	11.13

Tabella 6.5: α e fascia oraria per cui si ha un RMSE medio minimo sull'esperimento di Parigi

Categorie	Perc. degli archi
footway	56.5%
pedestrian	1%
primary	12.2%
residential	15.2%
secondary	3.7%
service	5.1%
steps	0.7%
tertiary	5.9%

Tabella 6.6: Categorie di OpenStreetMap e relative percentuali degli archi considerati per l'esperimento su parigi

I grafici dei risultati, mostrano che il parametro α mediamente ottimale bilancia equamente le componenti $I(T)$ e $C(T)$ dell'euristica `WINH` (4.2). D'altro canto però si può notare che l'euristica produce una varianza molto notevole tra tutti gli archi coinvolti, superiore all'esperimento su Bologna.

Considerati i valori RMSE per $\alpha = 0.5$ fissato, sono stati messi a confronto con i risultati ottenuti con l'euristica `Random(min, max)`, riassunti dai seguenti grafici.

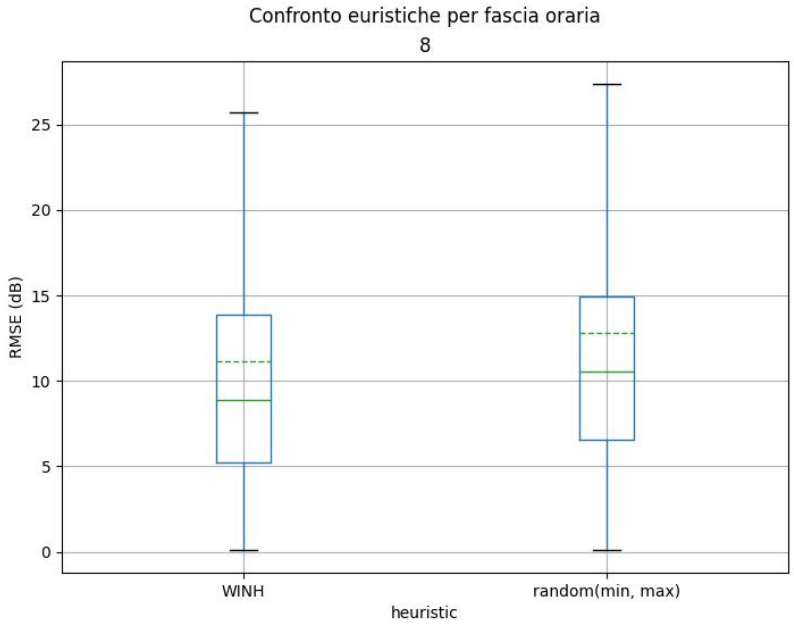


Figura 6.8: Boxplot RMSE delle euristiche WINH e random(min, max) messe a confronto e raggruppate per fascia oraria sperimentato su Parigi

Dai grafici si può notare che l'euristica `random(min, max)` ha mediamente un $RMSE = 12.8dB$, cioè poco più di 1 dB maggiore rispetto all'euristica `WINH` (4.2);

I risultati dell'esperimento sulla zona di Parigi, non sembrano buoni quanto negli altri esperimenti, dunque l'area è stata divisa rispettivamente in 2 parti, considerando solo 4 dei 6 Arrondissment più rilevanti raggruppati a coppie tra loro adiacenti, considerando dunque le aree (7, 15), e (5,13) con ciascuna il suo dataset associato differente.

Esperimento circoscrizione 7 e 15 Nell'esperimento sulle circoscrizioni 7 e 15 di Parigi sono stati coinvolti 171 archi.

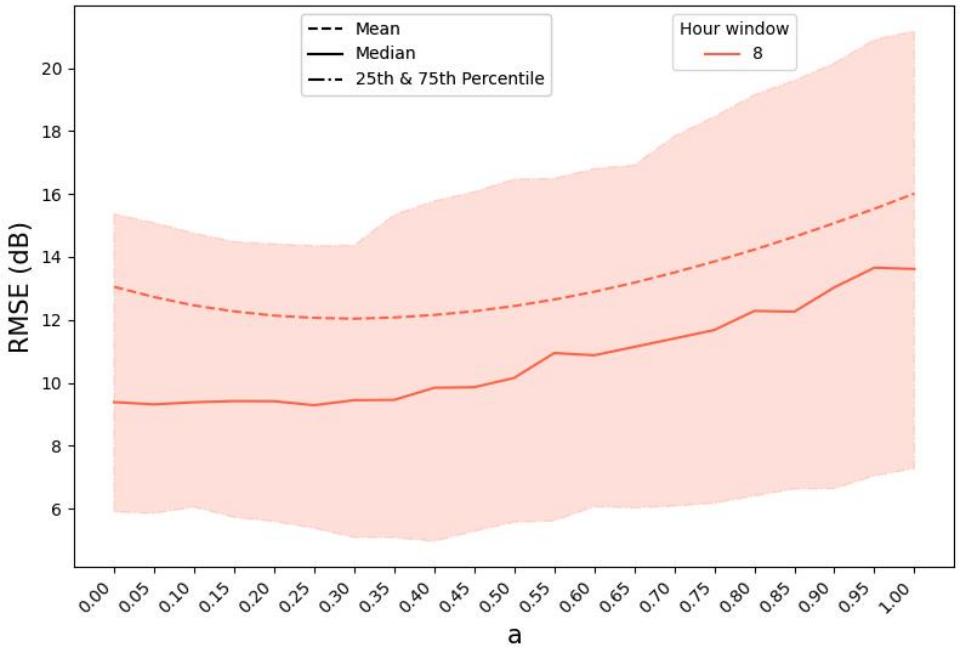


Figura 6.9: Lineplot RMSE sull'euristica WINH durante la fascia oraria 8 al variare del parametro α sperimentato su Parigi (circoscrizioni 7, 15)

T	α	**RMSE medio (dB)**
8	0.3	12.04

Tabella 6.7: α e fascia oraria per cui si ha un RMSE medio minimo sull'esperimento di Parigi (circoscrizioni 7, 15)

Categorie	Perc. degli archi
footway	49.1%
primary	12.9%
residential	18.7%
secondary	1.8%
service	5.8%
tertiary	11.7%

Tabella 6.8: Categorie di OpenStreetMap e relative percentuali degli archi considerati per l'esperimento su Parigi (circoscrizioni 7, 15)

Dal lineplot associato alle circoscrizioni 7 e 15 di Parigi, si deduce che ha più rilevanza il rumore fornito dalla categoria stradale, rispetto al contributo degli archi adiacenti. Infatti se α tende a 1 si ha che il valore mediano dell'errore e la varianza aumentano notevolmente.

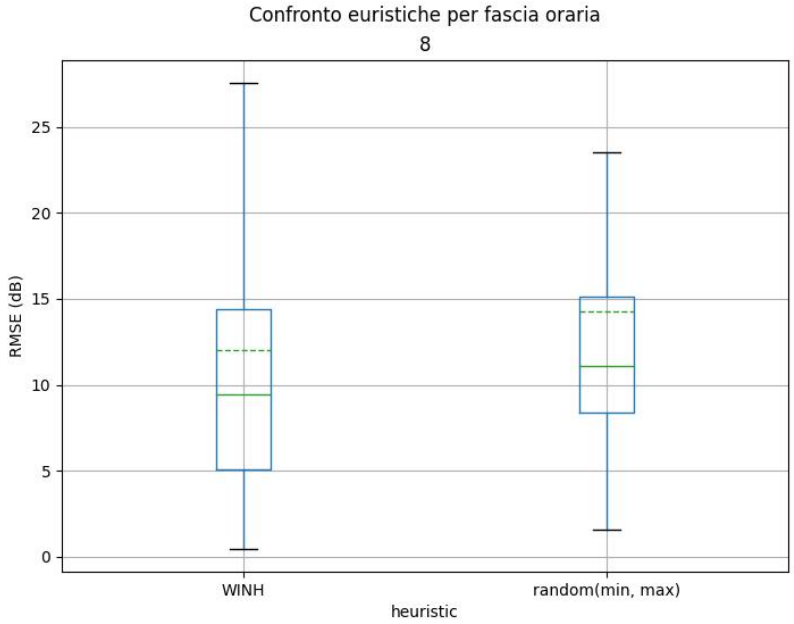


Figura 6.10: Boxplot RMSE delle euristiche WINH e random(min, max) messe a confronto e raggruppate per fascia oraria sperimentato su Parigi (circoscrizioni 7, 15)

Dai grafici si può notare che l'euristica `random(min, max)` ha mediamente un $RMSE = 14.3dB$, cioè circa di 2 dB maggiore rispetto all'euristica `WINH` (4.2). D'altro canto la varianza dell'errore risulta più limitata con l'euristica `random(min, max)`.

Analizzandone i grafici heatmap, si può notare infatti che le due euristiche hanno prodotto risultati simili e sono presenti situazioni anomale in entrambi sugli stessi segmenti dove si presenta un errore superiore ai 30 dB, ma in linea generale gli altri archi hanno un errore intorno all'errore medio.

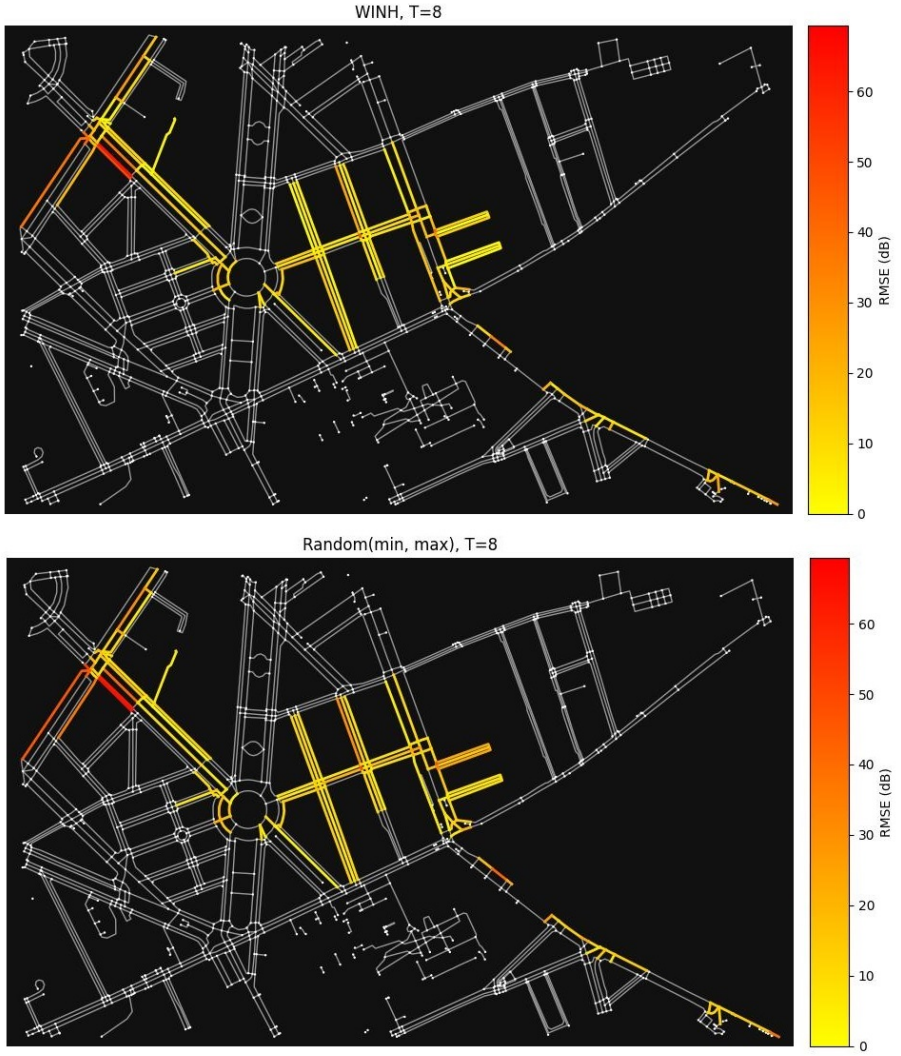


Figura 6.11: Grafici Heatmap del grafo di Parigi (circoscrizioni 7, 15) sui valori RMSE raggruppati per fascia oraria ed euristica ($\alpha = 0.3$)

Le categorie stradali sono abbastanza diversificate, ma per alcune di esse c'è una bassa percentuale di presenza e probabilmente la causa degli errori anomali, potrebbe essere dovuta proprio al fatto che certi segmenti sono poco rappresentati dallo storico della categoria associata, ma al contempo gli archi adiacenti potrebbero appartenere a

categorie molto diverse da quella di un arco di riferimento, come accade ad esempio nell'incrocio tra "Avenue de Saxe" e "Avenue de Ségur" (Nella parte superiore sinistra dei grafico).

Esperimento circoscrizione 5 e 13 Nell'esperimento sulle circoscrizioni 5 e 13 di Parigi sono stati coinvolti 139 archi.

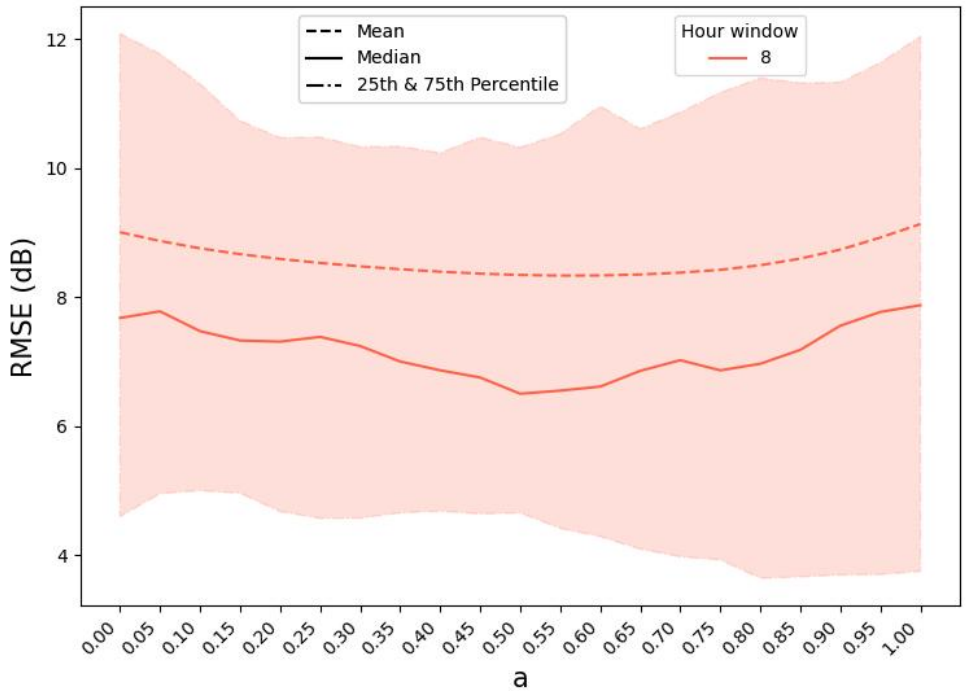


Figura 6.12: Lineplot RMSE sull'euristica WINH durante la fascia oraria 8 al variare del parametro α sperimentato su Parigi (circoscrizioni 5, 13)

T	α	**RMSE medio (dB)**
8	0.55	8.34

Tabella 6.9: α e fascia oraria per cui si ha un RMSE medio minimo sull'esperimento di Parigi (circoscrizioni 5, 13)

Categorie	Perc. degli archi
footway	66.2%
pedestrian	2.9%
primary	9.4%
residential	13.7%
secondary	5.8%
steps	0.7%
tertiary	2.2%

Tabella 6.10: Categorie di OpenStreetMap e relative percentuali degli archi considerati per l'esperimento su Parigi (circoscrizioni 5, 13)

Il lineplot associato alle circoscrizioni 5 e 13 di Parigi, evidenzia che l'errore RMSE medio ha lievissime variazioni con un valore minimo medio e mediano per $\alpha = 0.55$. Dunque le componenti dell'euristica `WINH` sono quasi bilanciate. La varianza rimane molto alta, ma l'errore medio risulta comunque inferiore rispetto alle precedenti osservazioni su Parigi.

Seguono i risultati ottenuti fissando $\alpha = 0.55$, e confrontando i risultati delle due euristiche, sintetizzate dai seguenti grafici:

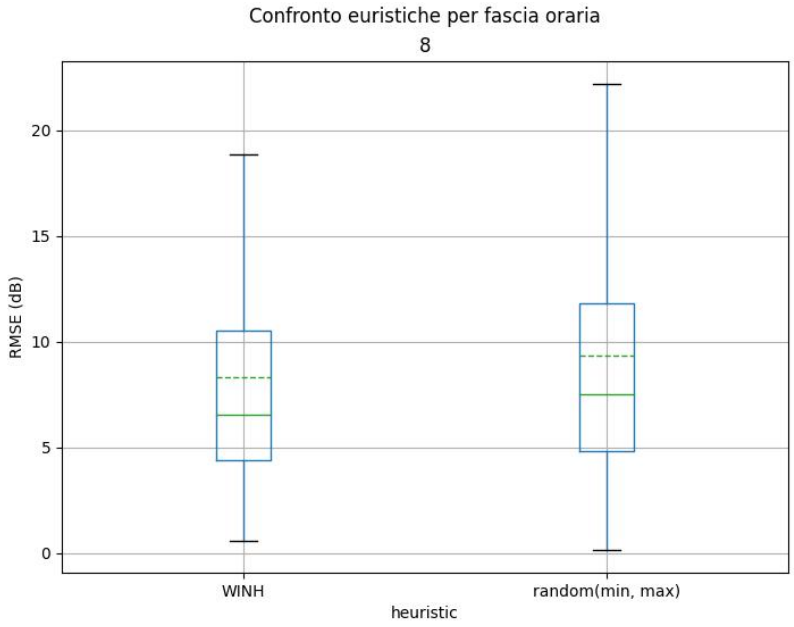


Figura 6.13: Boxplot RMSE delle euristiche WINH e random(min, max) messe a confronto e raggruppate per fascia oraria sperimentato su Parigi (circoscrizioni 5, 13)

I grafici evidenziano che l'euristica `random(min, max)` ha mediamente un $RMSE = 9.35dB$, cioè 1 dB maggiore rispetto all'euristica `WINH` (4.2);

Analizzandone i grafici heatmap, si può notare infatti che le due euristiche hanno prodotto risultati simili con un errore ≥ 15 dB presente sugli stessi segmenti, ma in linea generale gli altri archi hanno un errore intorno all'errore medio.

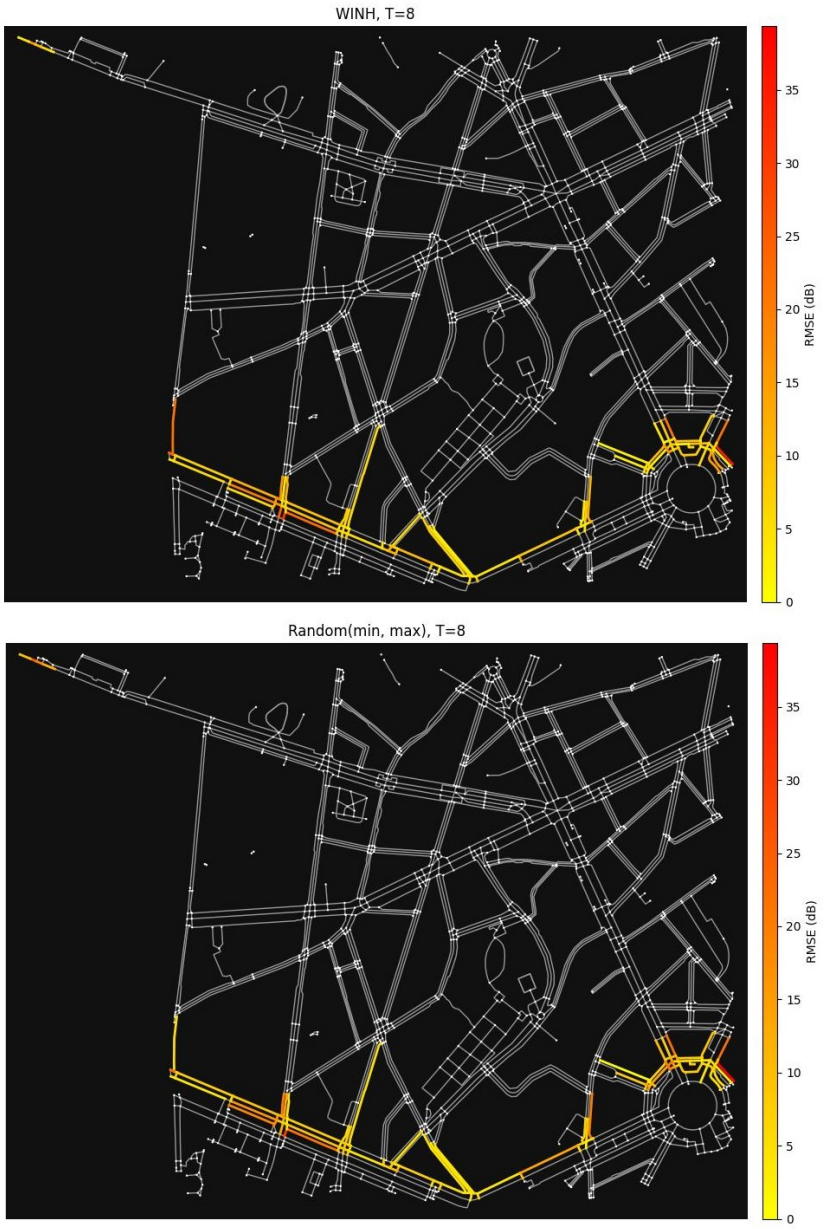


Figura 6.14: Grafici Heatmap del grafo di Parigi (circoscrizioni 5, 13) sui valori RMSE raggruppati per fascia oraria ed euristica ($\alpha = 0.55$)

Conclusioni e sviluppi futuri

Questo lavoro ha mostrato gli aspetti per implementare un sistema di previsione del rumore su un servizio di navigazione.

I risultati hanno evidenziano come effettivamente l'euristica `WINH` (4.2) sia in generale più efficace di un approccio naive che segue l'euristica `random(min, max)` (6.1) negli esperimenti affrontati. La varianza dell'errore spesso risulta minore con l'euristica `WINH` (4.2) ma può risultare comunque alta in determinate aree a causa del rumore che è intrinsecamente molto variabile. Le cause potrebbero essere diverse, probabilmente potrebbero essere svolti ulteriori approfondimenti considerando un dataset che copre un periodo storico maggiore.

Il parametro α dell'euristica `WINH` (4.2) è risultato variabile in città diverse. I risultati sull'esperimento di Lyon, potrebbero indicare che tale parametro dipende dalla varietà di categorie stradale vicine a ciascun arco; mentre i risultati di Parigi probabilmente indicano che può variare da una circoscrizione ad un'altra. Inoltre il valore ideale di questo parametro indicativamente sembra essere comunque compreso tra 0.45 e 0.55 su aree composte da diverse categorie stradali, mentre un valore che tende a $\alpha = 1.0$ su aree con poca varietà di categorie stradali coperte sul dataset di riferimento. Un'estensione dell'euristica `WINH` potrebbe riguardare dunque l'introduzione di una componente pesata sul contributo fornito dalle categorie stradali presenti intorno ad un arco di riferimento.

L'errore di previsione in generale è molto variabile, ma se si considera i risultati ottenuti su aree circoscritte, l'errore di previsione dell'euristica `WINH` (4.2) rispetto al valore previsto dal modello Prophet su ciascun arco risulta essere mediamente $5, 4$ e 11 dB rispettivamente nei casi osservati.

La precisione ottenuta, sopratutto considerando l'alta varianza dei risultati, è un po' lontana rispetto alla precisione ottenuta dai $3 - 5.9dB$ di errore rilevati con il metodo proposto dallo studio "Reliability of Dynamap traffic noise prediction" [16] nel conteso del progetto Dynamap[4]. D'altro canto però è difficile confrontare i due lavori siccome in questa tesi sono state considerate città con caratteristiche stradali diverse, su un dataset limitato, facendo esclusivamente uso di dati acustici e parametri dei modelli Prophet generalizzati; dunque applicando metodologie e contesti diversi dallo studio che ha considerato invece dati acustici campionati da stazioni di monitoraggio fisse e informazioni relative a caratteristiche del traffico stradale[16].

Dunque si può dire che il lavoro proposto in questa tesi, può essere considerato come un valido compromesso per un sistema di previsione del rumore in un contesto di mobile crowdsensing, soprattutto se si considerano i livelli di rumore previsti come livelli indicativi di quiete implementati su un servizio di navigazione.

In generale com'è stato osservato e già esplicitato da altri lavori in letteratura, è complesso modellare un sistema di previsione di rumore ambientale affidabile. La variabilità del rumore può essere notevole, anche senza considerare fattori esterni quali ad esempio il flusso del traffico e le condizioni meteo. Il sistema di previsione adottato può essere sicuramente migliorato in futuro ad esempio configurando i modelli Prophet con regressori che dipendono da condizioni meteorologiche. Infatti dati relativi a temperature, precipitazioni e venti ottenuti da servizi pubblici potrebbero essere considerati fattori importanti per migliorare le precisione dell'approccio affrontato, sia applicando previsioni meteorologiche che rilevazioni storiche per calibrare meglio i modelli Prophet.

Bibliografia

[1] "Environment noise in europe," European Environment Agency (EEA), Tech. Rep. 22/2019, 2020. [Online]. Available: https://doi.org/10.2800/686249

[2] W. Babisch, "The noise/stress concept, risk assessment and research needs," *Noise and Health*, vol. 4, no. 16, pp. 1–11, 2002. [Online]. Available: https://www.noiseandhealth.org/text.asp?2002/4/16/1/31833

[3] T. Hellmuth, T. Claßen, R. Kim, and S. Kephalopoulos, *Methodological guidance for estimating the burden of disease from environmental noise.* World Health Organization (WHO), 12 2012.

[4] "DYNAMAP Project," acceduto il 06-09-2023. [Online]. Available: https://life-dynamap.eu

[5] M. Zappatore, A. Longo, and M. A. Bochicchio, "Using mobile crowd sensing for noise monitoring in smart cities," *2016 International Multidisciplinary Conference on Computer and Energy Science (SpliTech)*, pp. 1–6, 2016.

[6] A. Ghosh, K. Kumari, S. Kumar, M. Saha, S. Nandi, and S. Saha, "Noiseprobe: Assessing the dynamics of urban noise pollution through participatory sensing," *2019 11th International Conference on Communication Systems & Networks (COMSNETS)*, pp. 451–453, 2019.

[7] N. Maisonneuve, M. Stevens, M. Niessen, and L. Steels, "Noisetube: Measuring and mapping noise pollution with mobile phones," *Information Technologies in Environmental Engineering*, pp. 215–228, 01 2009.

[8] J. Picaut, A. Boumchich, E. Bocher, N. Fortin, G. Petit, and P. Aumond, "A smartphone-based crowd-sourced database for environmental noise assessment," *International Journal of Environmental Research and Public Health*, vol. 18, no. 15, 2021. [Online]. Available: https://doi.org/10.3390/ijerph18157777

[9] Agenzia Regionale della Protezione dell'Ambiente (ARPA) Valle d'Aosta, "Il rumore, i decibel e il livello equivalente," 2013, acceduto il 08-07-2023.

[Online]. Available: https://www.arpa.vda.it/it/agenti-fisici/rumore-ambientale/nozioni-generali/decibel-e-leq

[10] D. Bies and C. Hansen, *Engineering Noise Control: Theory and Practice, Fourth Edition*. Taylor & Francis, 2009. [Online]. Available: https://doi.org/10.1201/9781315273464

[11] R. K. Ganti, F. Ye, and H. Lei, "Mobile crowdsensing: current state and future challenges," *IEEE Communications Magazine*, vol. 49, no. 11, pp. 32–39, 2011.

[12] "TomTom." [Online]. Available: https://www.tomtom.com/

[13] "Google Maps." [Online]. Available: https://maps.google.com

[14] "OpenStreetMap." [Online]. Available: https://www.openstreetmap.org/about

[15] G. Boeing, "Osmnx: New methods for acquiring, constructing, analyzing, and visualizing complex street networks," *Computers, Environment and Urban Systems*, vol. 65, pp. 126–139, 2017. [Online]. Available: https://doi.org/10.1016/j.compenvurbsys.2017.05.004

[16] R. Benocci, A. Molteni, M. Cambiaghi, F. Angelini, H. E. Roman, and G. Zambon, "Reliability of dynamap traffic noise prediction," *Applied Acoustics*, vol. 156, pp. 142–150, 2019. [Online]. Available: https://doi.org/10.1016/j.apacoust.2019.07.004

[17] M. Smiraglia, R. Benocci, G. Zambon, and H. Roman, "Predicting hourly traflc noise from traflc flow rate model: Underlying concepts for the dynamap project," *Noise Mapping*, vol. 3, no. 1, 2016. [Online]. Available: https://doi.org/10.1515/noise-2016-0010

[18] G. Cannelli, K. Gluck, and S. Santoboni, "A mathematical model for evaluation and prediction of the mean energy level of traffic noise in italian towns," *Acustica*, vol. 53, no. 1, p. 31 – 36, 1983, acceduto il 06-09-2023. [Online]. Available: https://www.scopus.com/inward/record.uri?eid=2-s2.0-0020749362&partnerID=40&md5=86f67e51342babf62146b66a33eec3ae

[19] S. J. Taylor and B. Letham, "Forecasting at scale," *PeerJ Preprints*, 2017, acceduto il 06-09-2023. [Online]. Available: https://doi.org/10.7287/peerj.preprints.3190v2

[20] "Back4app." [Online]. Available: https://www.back4app.com

[21] "Nominatim: Open-source geocoding with OpenStreetMap data," acceduto il 05-07-2023. [Online]. Available: https://nominatim.org/

[22] “Android API reference - AudioRecord,” acceduto il 20-06-2023. [Online]. Available: https://developer.android.com/reference/android/media/AudioRecord

[23] “NoiseTube project,” acceduto il 06-07-2023. [Online]. Available: https://gitlab.soft.vub.ac.be/NoiseTube/

[24] *Key:highway*, acceduto il 06-09-2023. [Online]. Available: https://wiki.openstreetmap.org/wiki/Key:highway

[25] “Osmdroid.” [Online]. Available: https://github.com/osmdroid/osmdroid

[26] “Noise-planet,” acceduto il 02-09-2023. [Online]. Available: https://noise-planet.org/

Ringraziamenti

Per prima cosa, vorrei ringraziare il mio relatore Federico Montori, per i suoi preziosi consigli e per la sua disponibilità. Grazie per avermi fornito spunti fondamentali e avermi indirizzato durante la stesura di questo lavoro.

Ringrazio i miei genitori, che mi hanno permesso di raggiungere questo traguardo senza farmi mancare mai niente e supportandomi ogni giorno.

Ringrazio Silvia per essermi stata vicina in ogni momento e per aver avermi strappato continuamente dei sorrisi. Grazie per avermi trasmesso le qualità di Bologna, fino a essere di ispirazione per la realizzazione di questo lavoro.

Ringrazio anche Alessio, Elisa, Erica, Davide, Manuel, Mattia, Samuel e Simone per avermi supportato e sopportato durante il periodo di studi.